

# ETDK-dolgozat

Kovács Kinga

Visky Áron

**XXIX. Reál- és humántudományi Erdélyi Tudományos Diákköri Konferencia (ETDK)**

**Informatika II: Innovatív számítástechnikai termékek, alkalmazások szekció**

**Kolozsvár, 2026. május 14–17.**

**KMO**

**Holisztikus eseménykezelő rendszer**

***A Kocka Másik Oldala* diákszervezet számára**



**Szerzők:**

**Kovács Kinga**

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

**Visky Áron**

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

**Témavezetők:**

**dr. Sulyok Csaba**, egyetemi adjunktus,

Babeş–Bolyai Tudományegyetem, Matematika és Informatika Kar

**Bándi Nándor**, doktorandusz,

Babeş–Bolyai Tudományegyetem, Matematika és Informatika Kar

**Nagy Kristóf-Bálint**, szoftverfejlesztő,

SOFTECH SRL



## Kivonat

A *Kocka Másik Oldala* a kolozsvári *Babeş–Bolyai Tudományegyetem* Matematika és Informatika Karának diákszervezete, amely heti rendszerességgel szervez közösségépítő programokat a hallgatók számára. Az események előkészületeinek során a csapat több különböző online platform szolgáltatásait veszi igénybe: a rendezvények meghirdetése közösségi médián keresztül valósul meg, a jelentkezések digitális kérdőívek segítségével történnek, a résztvevők nyilvántartására pedig táblázatkezelő eszközök vannak alkalmazva. Az interaktív események külső szolgáltatók alkalmazásaira támaszkodnak, amelyeknek ingyenes változatai nem testre szabhatóak a diákszervezet igényeihez mérten. Továbbá a rendezvényeken készült médiatartalmak nehezen hozzáférhető módon kerülnek tárolásra és megosztásra.

A jelen projekt keretében megvalósult webalapú platform az említett nehézségeket küszöböli ki azáltal, hogy centralizálja a rendezvényszervezés főbb aspektusait, lehetőséget nyújtva az események egységes kezelésére és közzétételére, a jelentkezések feldolgozására, valamint a médiatartalmak tárolására és megosztására. A platform összetett jogosultsági mechanizmust biztosít a szervezési feladatok szétosztására, valamint bővítményrendszert kínál külső webes alkalmazások integrálására, a különböző eseményeknél használatos interaktív játékok megvalósításához.

A dolgozat célja a weboldal fejlesztési folyamatának és működésének ismertetése. Bemutatásra kerül a rendszer architektúrája, valamint a projekt során felhasznált technológiák. Szó lesz a biztonsági és infrastrukturális megoldásokról, illetve az elemzés tárgyát képezik a rendszer bővíthetőségének lehetőségei és a jövőbeli fejlesztési irányok is.

# Tartalomjegyzék

|                                                                    |           |
|--------------------------------------------------------------------|-----------|
| <b>1. Bevezető</b>                                                 | <b>1</b>  |
| <b>2. A projekt leírása</b>                                        | <b>3</b>  |
| 2.1. Szerepkörök és jogosultságok . . . . .                        | 3         |
| 2.2. Funkcionalitások . . . . .                                    | 4         |
| 2.2.1. Főoldal . . . . .                                           | 4         |
| 2.2.2. Eseménykezelés . . . . .                                    | 6         |
| 2.2.3. Galéria . . . . .                                           | 6         |
| 2.2.4. Javaslatok . . . . .                                        | 7         |
| 2.2.5. Felhasználók menedzselése . . . . .                         | 8         |
| 2.2.6. Bővítménykezelés . . . . .                                  | 9         |
| <b>3. Architektúra</b>                                             | <b>10</b> |
| 3.1. Szerveralkalmazás . . . . .                                   | 10        |
| 3.1.1. Többrétegű architektúra . . . . .                           | 11        |
| 3.1.2. Adatkezelés és perzisztencia . . . . .                      | 12        |
| 3.1.3. Adatmodellek . . . . .                                      | 13        |
| 3.1.4. Biztonság és autentikáció . . . . .                         | 13        |
| 3.1.5. Hibakezelés . . . . .                                       | 14        |
| 3.2. Kliensalkalmazás . . . . .                                    | 15        |
| 3.2.1. Könyvtárstruktúra és a felelőségek szétválasztása . . . . . | 15        |
| 3.2.2. Navigáció és útvonalkezelés . . . . .                       | 16        |
| 3.2.3. Adatlekérés és gyorsítótárazás . . . . .                    | 16        |
| 3.2.4. Teljesítményoptimalizálás . . . . .                         | 16        |
| 3.3. Bővítményrendszer . . . . .                                   | 17        |
| 3.3.1. Szoftverfejlesztői csomag . . . . .                         | 17        |
| 3.3.2. Adatmodell és entitáskezelés . . . . .                      | 18        |
| <b>4. Felhasznált technológiák és eszközök</b>                     | <b>20</b> |
| 4.1. Szerveroldali technológiák . . . . .                          | 20        |
| 4.2. Kliensoldali technológiák . . . . .                           | 21        |
| 4.3. Adattárolási megoldások . . . . .                             | 23        |

|                                                            |           |
|------------------------------------------------------------|-----------|
| 4.4. Infrastrukturális eszközök . . . . .                  | 25        |
| 4.5. Fejlesztői eszközök és módszerek . . . . .            | 25        |
| <b>5. Biztonság</b>                                        | <b>27</b> |
| 5.1. Rendszerszintű biztonság . . . . .                    | 27        |
| 5.2. Bővítményszintű biztonság . . . . .                   | 29        |
| <b>6. Az alkalmazás infrastruktúrája</b>                   | <b>31</b> |
| 6.1. Build folyamat . . . . .                              | 31        |
| 6.2. Konténerizáció . . . . .                              | 31        |
| 6.3. Kitelepítés . . . . .                                 | 32        |
| 6.3.1. A fejlesztői környezet orkesztrációja . . . . .     | 32        |
| 6.3.2. Klaszter alapú üzemeltetés . . . . .                | 33        |
| <b>7. Következtetések és továbbfejlesztési lehetőségek</b> | <b>34</b> |

# 1. Bevezető

*A Kocka Másik Oldala* egy 2014-ben alakult diákszervezet, amelynek fő célja közösségépítő programok szervezése a kolozsvári magyar matematika és informatika szakos hallgatók számára.

**Motiváció** Az eseményszervezés során a szervezet olyan logisztikai és adminisztratív nehézségekkel szembesült, amelyek szükségessé tették egy átfogó, digitális környezet létrejöttét. A hatékonyságot hátráltató egyik tényező a decentralizált adatkezelés volt. A rendezvényekre való jelentkezés *Google Forms* űrlapok segítségével történt, a tagnyilvántartás pedig *Microsoft Excel* táblázatok segítségével zajlott, ami az adatok redundanciájához és nehézkes kereshetőségéhez vezetett. Kihívást jelentett az is, hogy az eseményeken használt interaktív eszközök külső szolgáltatóktól függtek. Ezen platformok ingyenes változatai korlátozzák a résztvevők számát, nem testre szabhatóak, valamint nem nyújtanak lehetőséget az adatok permanens tárolására. Továbbá gondot okozott a szervezetnek a médiatartalmak kezelése is, mivel a rendezvényeken készült fényképek megosztása közösségi média platformokon vagy felhő alapú tárhelyeken (*Google Drive*) történt, ahol a képek minősége romlott, valamint a hozzáférési jogosultságok kezelése nehézkessé vált.

**Célkitűzések** A projekt célja egy olyan weboldal fejlesztése, amely egységes környezetként szolgál *A Kocka Másik Oldala* eseményszervezési folyamatainak kezelésére, kiküszöbölve a fent említett problémákat. A dolgozatban bemutatott alkalmazás 2025 nyarán jött létre, a *Softech* szoftverfejlesztő cég által szervezett szakmai gyakorlat keretein belül.

A tervezési folyamat során több irányelv lett meghatározva: elsődleges célkitűzésként a központosított eseménykezelés került megfogalmazásra. A diákszervezet számára fontos egy olyan holisztikus eseménykezelő platform létrehozása, amelynek segítségével egyetlen felületen képesek véghez vinni a rendezényszervezés minden szakaszát. Ennek szerves részét képezi egy online jelentkezési rendszer megvalósítása, amely lehetővé teszi a résztvevők számára, hogy közvetlenül a weboldalon keresztül jelezzék részvételi szándékukat, felváltva ezzel a korábban alkalmazott, fragmentált külső megoldásokat.

Fontos szempont a médiatartalmak strukturált kezelése is, ami biztosítja az eseményekhez kapcsolódó fényképek minőségromlás nélküli, átlátható tárolását és könnyű böngészhetőségét. Egy másik alapvető irányelv az adminisztratív folyamatok egyszerűsítése. Olyan automatizált eszközök biztosítása a cél, amelyek csökkentik a szervezők ismétlődő, manuális feladatait –

legyen szó véleménynyilvánító kérdőívek vagy javaslatgyűjtő űrlapok generálásáról.

Ezt a technológiai ökoszisztémát egészíti ki az alkalmazásintegráció támogatása, amelynek célja egy bővítményalapú rendszer létrehozása, lehetővé téve különböző webes applikációk beillesztését és futtatását. Mindemellett a fejlesztés elengedhetetlen pillére a biztonság garantálása: olyan mechanizmusok alkalmazása a cél, amelyek nem csupán a felhasználói adatok védelmét biztosítják, hanem megfelelőképpen szabályozzák a rendszer erőforrásaihoz való hozzáférést is.

**Piaci alternatívák** A szervezet eseményszervezéshez kapcsolódó folyamatainak egyszerűsítésére számos digitális eszköz áll rendelkezésre, azonban ezek nem felelnek meg teljes mértékben a csapat igényeinek.

Az olyan általános adatgyűjtő platformok, mint a *Google Forms*, bár kiválóak a gyors jelentkezési űrlapok létrehozására, megmaradnak az egyszerű adatfelvétel szintjén. A kifejezetten rendezvények menedzselésére kiélezett szolgáltatások – mint az *Eventbrite* vagy a *Meetup* – kereskedelmi célokra készültek, és korlátozott lehetőséget biztosítanak kisebb közösségek számára. Médiatartalmak tárolására és megosztására is számos online szolgáltatás létezik, mint például a *Google Photos* vagy a *Flickr*, de nem kínálnak megoldást az események tematikus, több évre visszamenőleg is könnyen kereshető és rendszerezett megőrzésére.

A rendezvények interaktivitását támogató eszközök, mint például a *Kahoot* vagy a *Mentimeter*, díjmentes csomagjai alacsony résztvevői limittel rendelkeznek. Ugyanakkor fellelhetőek olyan keretrendszerek és *widget-platformok* – mint amilyen az *Y8* játékporthál is –, amelyek külső webes alkalmazások integrációját teszik lehetővé. Ezek azonban alapvetően általános tartalommegosztásra lettek kiélezve, és egyszerűbb beágyazást kínálnak, mintsem valódi integrációt egyedi alkalmazásokkal.

Fontos kihangsúlyozni, hogy a felsorolt megoldások mindegyike csak egy-egy részét fedi le a szervezet feladatainak. Ez a jelenség azonban nem egyedi: egy hasonló profilú nonprofit szervezet digitális helyzetét vizsgáló tanulmányban [12] olvasható, hogy az olyan egymástól független eszközök párhuzamos alkalmazása, mint a *WhatsApp* vagy a *Google Drive*, operatív hatékonyságsökkenéshez és információvesztéshez vezet. A cikk szerzői arra a következtetésre jutottak, hogy a szervezeteknek nem elszigetelt digitális eszközökre, hanem egységes, integrált digitális stratégiára van szükségük. Mindezek alapján *A Kocka Másik Oldala* esetében is célszerű egy olyan webalkalmazás fejlesztése, amely az események szervezését, a résztvevők jelentkezését, valamint a médiatartalmak megjelenítését ugyanazon a helyen valósítja meg.

## 2. A projekt leírása

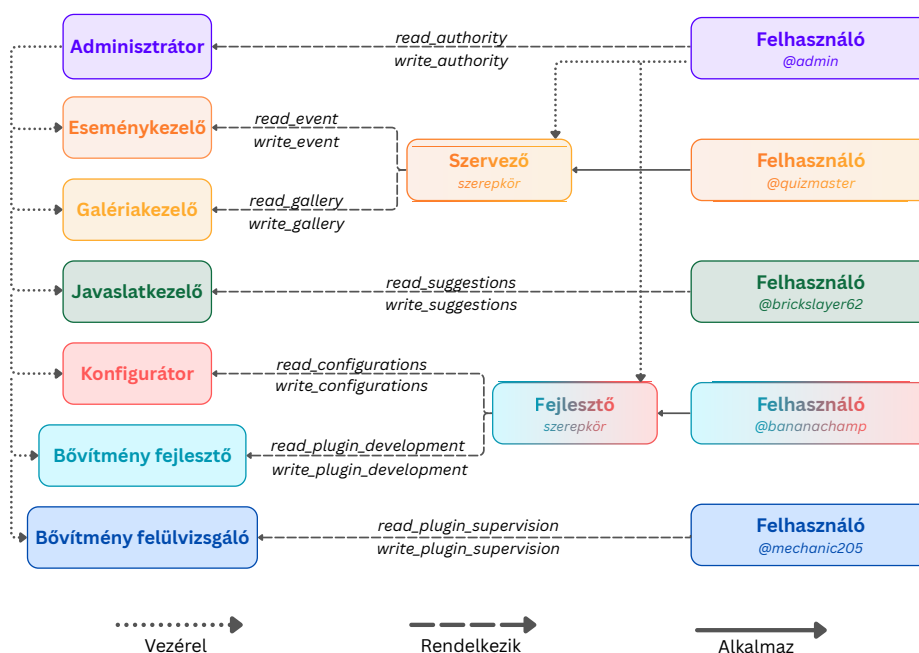
Ahogy sok más diákszervezet, úgy *A Kocka Másik Oldala* is lelkes önkéntesekből és szervezőkből áll össze, akiken rengeteg felelősség oszlik el, hiszen egy több mint 200 főből álló diákközösség összefogása komoly feladat. Törekvéseik a heti rendszerességgel megszervezett eseményekben és a szakmai együttműködésekben nyilvánulnak meg, amit a közösségi felületeken zajló digitális tartalomgyártás is kiegészít. Emellett a fejlődés igénye miatt, a résztvevők visszajelzései alapján a rendezvények folyamatosan továbbfejlesztéseken esnek át, biztosítva a programok relevanciáját. Érezhető tehát, hogy mennyi szerepkör és felelősségi terület alakul ki a diákkör életében, amelyek sokszor át is fedik egymást.

### 2.1. Szerepkörök és jogosultságok

A webalkalmazás fejlesztése során kiemelt szempont volt a szervezet működési struktúrájának támogatása: az összetett jogosultsági rendszere különböző hozzáférési szinteket különít el, amelyeket az adminisztrátor rendelhet hozzá bármely, az oldalon regisztrált, felhasználóhoz – vagy szükség esetén vonhat vissza tőlük –, leegyszerűsítve a szerepkörök és jogosultságok szabályozását. Ide tartoznak az esemény-, galéria- és javaslatkezelési, valamint a konfigurációs, bővítménykezelési és hozzáférési jogosultságok.

Az esetek többségében egy tag nem csak egy feladattal van megbízva, hanem a szervezés több szegmensében is számítanak a segítségére: előfordulhat, hogy egy szervező az események lebonyolítása mellett a javaslatok feldolgozásáért is felel, vagy hogy a médiatartalmakért felelős egyénre hárul a támogatókkal való kapcsolattartás és a közösségi oldalak kezelése is. A weboldal jogosultsági rendszere ehhez igazodva lehetővé teszi, hogy egy felhasználónak több jogosultsága is legyen, így az oldal vezérlésének több funkcionális területéhez is van hozzáférése.

Egy másik fontos funkciója az oldalnak, hogy a jogosultságok csoportokba rendezhetők. Ezeket a csoportokat szerepkörökként lehet definiálni, amelyek lényegében sablonként szolgálnak. Az adminisztrátor a 3b. képen látható oldalon létrehozhat egy szerepkört, megadva a hozzá tartozó nevet, színkódot és jogosultságokat, melyeket a későbbiek során még szerkeszthet. Egy felhasználó jogosultságainak megadásakor kiválasztható a létrehozott szerepkör, amelynek következtében a hozzá tartozó jogosultságok automatikusan hozzárendelésre kerülnek. Ez megkönnyítheti az adminisztrátor feladatát, ha több embernek is ugyanazokat a jogokat szeretné megadni, hiszen egy szerepkör elkészítésével egyszerűen és hatékonyan tudja alkalmazni



1. ábra. Példa jogosultságok és szerepkörök eloszlására

azt minden szükséges felhasználónál. Az 1. ábra konkrét példák segítségével szemlélteti az adminisztrátor központi szerepét és a szerepkörök hozzárendelésének mechanizmusát.

## 2.2. Funkcionalitások

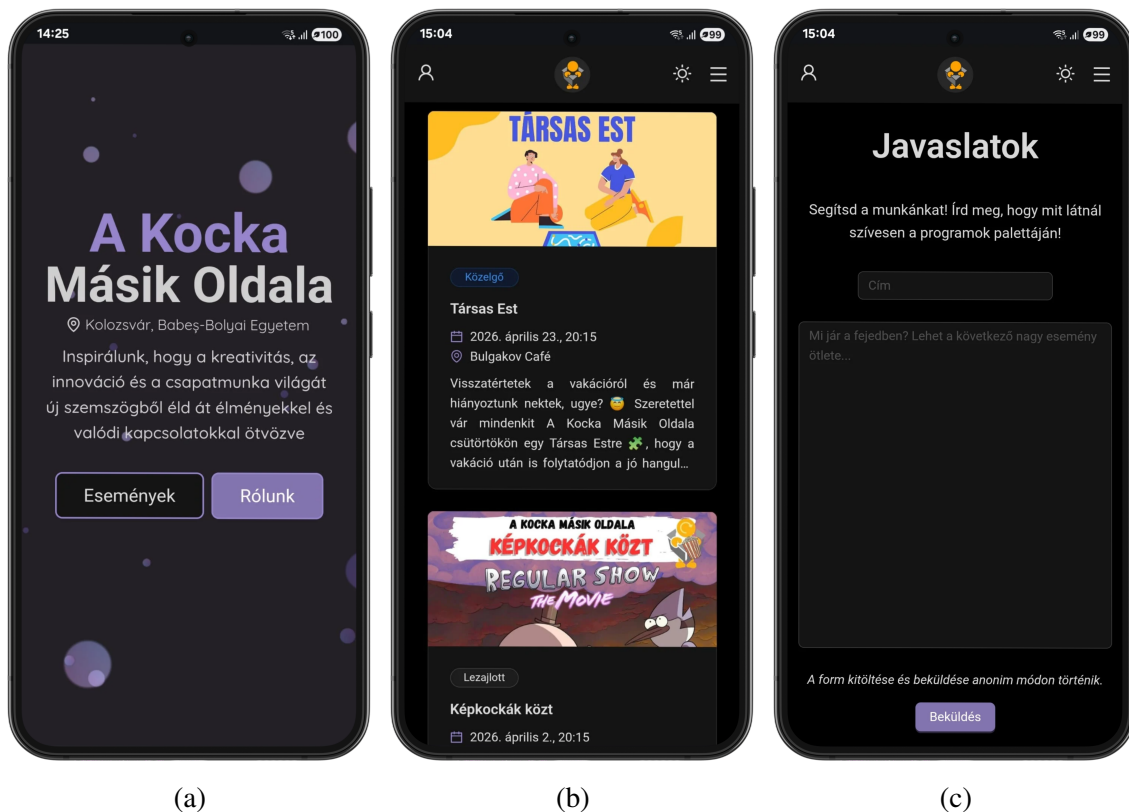
A webalkalmazás funkcionálisai a diákszervezet munkamenetét követik: az események szervezésétől, a közösségi visszajelzéseken át, a médiatartalmak kezeléséig, illetve az interaktív bővítmények támogatásáig.

### 2.2.1. Főoldal

A felhasználót a weboldalra való navigáláskor annak főoldala fogadja, amely célja, hogy *A Kocka Másik Oldala* arculatát közvetítse és információt nyújtson az aktuális eseményeiről.

A főoldal a 2a. ábrán látható üdvözlő felülettel kezdődik, amely egy figyelemfelkeltő, bemutatkozó szöveget és *call-to-action* elemeket tartalmaz az aktuális eseményekhez vagy a csapat egy részletesebb leírásához.

A navigációs menü az alkalmazás egészén átívelve biztosítja az átjárást a rendszer fő komponensei között. Dinamikusan alkalmazkodik a felhasználó állapotához: bejelentkezés előtt a menüpontok között regisztrációs és belépési lehetőségek szerepelnek, utána pedig profilkezelési és kijelentkezési opciók. Tartalmaz témaváltó ikonokat is, amelyek a világos



2. ábra. A Főoldal (a), Eseménylista (b) és Javaslatok (c) képernyők

és sötét színsémák közötti választást teszik lehetővé.

A jogosultságokkal rendelkező felhasználókat belépéskor azonban nem az imént bemutatott főoldal köszönti, hanem a 3a. képen látható vezérlőpult, amely a látogatóknak szánt nézet konfigurációjához szükséges eszközöket biztosítja. Ekkor a navigációs sávon megjelenő *Nézetváltás* opció segítségével megtekinthetik a látogatóknak szánt felületet a saját fiókjukból bejelentkezve, így szemügyre vehetik a változtatásaikat még mielőtt a felhasználó azokból bármit is érzékelne.

A főoldal további egységei közé tartoznak *A Kocka Másik Oldala* eredményeit bemutató statisztikák, a kiemelt helyen megjelenő legközelebbi esemény, valamint a diákszervezet együttműködéseit és támogatóit bemutató szekciók. Ezeket az információkat csak az ezekre vonatkozó, konfigurációs jogosultsággal rendelkező felhasználók szerkeszthetik a 3c. oldalon. A láblécben pedig a szervezet elérhetőségei, közösségi média hivatkozásai és jogi tájékoztatói találhatóak.

### 2.2.2. Eseménykezelés

A rendszer alappillére az eseménykezelő modul, amely a rendezvényszervezés teljes életciklusát lefedi, a közzétételtől egészen a regisztrációk feldolgozásáig.

Az események listáját megjelenítő oldalon a közelgő és a múltbéli programok kártyaalapú elrendezésben vannak megjelenítve, amint az a 2b. ábrán megfigyelhető. Minden eseménykártya a legfontosabb metaadatokat tartalmazza: megjelenít egy reprezentatív borítóképet, amelynek szerepe van a rendezvény hangulatának közvetítésében, valamint az esemény nevét, dátumát és helyszínét. A kártyákra kattintva a felhasználó a részletes nézetre navigál, ahol a bővebb leírás mellett megjelennek az eseményhez tartozó specifikusabb információk is a dinamikus jelentkezési gombbal együtt.

A rendezvényekre való jelentkezés a hozzájuk csatolt űrlapon keresztül történik, amely a korábbi jelentkezési kérdőívek mintáját követve, alapértelmezetten két mezőt tartalmaz: a jelentkező teljes nevét, valamint egy hozzájáruló nyilatkozatot, amelyben az illető beleegyezik, hogy az eseményen fényképek készüljenek róla. Amennyiben az egyén rendelkezik fiókkal, ezek a mezők automatikusan kitöltésre kerülnek a regisztráció során megadott neve, illetve a *GDPR*-preferenciái alapján, így a visszatérő látogatóknak nem szükséges újra begépelniük az adataikat.

Tekintve, hogy az egyes rendezvények jellege eltérő, a szervezőknek lehetőségük nyílik az űrlap bővítésére és személyre szabására. Egy társasest esetén például hozzáadható egy olyan mező, amelyben a jelentkező feltüntetheti, milyen társasjátékot tud magával hozni, míg egy kvízestnél megadható a csapat, valamint az összes további csapattag neve.

Az esemény kezelése, az űrlap szerkesztése és a beérkezett jelentkezések megtekintése kizárólag azok számára érhető el, akik rendelkeznek a szükséges eseménykezelési jogosultságokkal, ezáltal biztosítva, hogy az érzékeny adatok – mint a jelentkezők személyes információi – csak az arra feljogosított szervezők számára legyenek elérhetők.

### 2.2.3. Galéria

A weboldal egyik célja, hogy a diákszervezet rendezvényeiről készült fényképeket áttekinthető és esztétikus formában tegye elérhetővé a látogatók számára. A médiatartalmak két különböző módon böngészhetők: egyrészt az eseménynek szánt oldalon, ahol kizárólag az adott rendezvényhez tartozó képek jelennek meg. A második mód pedig a galériának dedikált oldal, amely az összes közzétett fényképet egyetlen, egységes felületen gyűjti össze.

A galéria esetén az *infinite scroll* technika került alkalmazásra a hagyományos, lapozáson alapuló megjelenítéssel szemben. Ez a döntés a galéria természetes használati mintájából következik: a látogató jellemzően nem egy konkrét képet keres, hanem böngészik – ilyen esetben a lapozásos navigáció felesleges interakciót követel meg a felhasználtól, és megbontja a böngészés folyamatát. Az *infinite scroll* ezzel szemben folyamatos, megszakítás nélküli élményt biztosít, miközben a lapozásos lekérdezés révén a képek nem egyszerre töltődnek be, hanem fokozatosan, a görgetés hatására jelennek meg újabb tartalmak. Ez egyrészt csökkenti a kezdeti betöltési időt, másrészt gördülékeny böngészési élményt biztosít nagy mennyiségű tartalom esetén is.

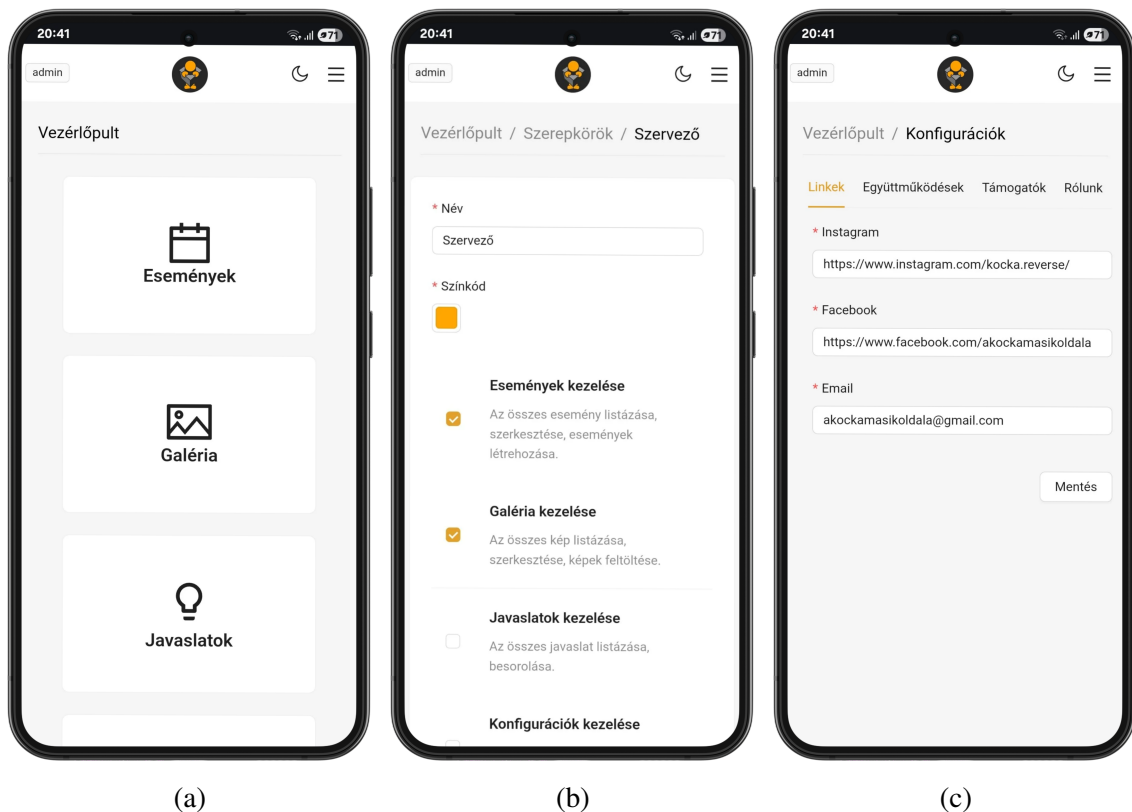
A teljesítmény további optimalizálása érdekében a rendszer csökkentett felbontású előnézeteket – úgynevezett *thumbnail*-eket – jelenít meg, amelyek jóval kisebb méretükből adódóan gyorsan betölthetők. A teljes felbontású változatok csak a felhasználó kattintására válnak elérhetővé egy *lightbox* nézetben.

A képek kezelése – feltöltése, közzététele és törlése – kizárólag azok számára érhető el, akik rendelkeznek a galériakezelési jogosultságokkal, így garantálva, hogy a vizuális tartalomért csak a kijelölt szervezők felelhessenek.

#### **2.2.4. Javaslatok**

Lehetőségük van mind a látogatóknak, mind a bejelentkezett felhasználóknak visszajelzést és új ötleteket adni a szervezők számára teljesen névtelenül, a javaslatoknak szánt oldalon elérhető űrlapon keresztül, amely a 2c. képen látható. A beküldőnek mindössze egy címet és egy rövid leírást kell megadnia, amely tartalmazhatja egy új rendezvény ötletét, egy korábbi eseménnyel kapcsolatos észrevételt vagy akár vonatkozhat egy filmestre szánt ajánlásra is.

A beérkezett visszajelzések kezelése csak a javaslatkezelési jogosultságokkal rendelkező szervezők számára érhető el, akik egy táblázatban tekinthetik meg az összes beküldött bejegyzést. A frissen létrehozott javaslatok automatikusan az *Új* címkével lesznek ellátva, amit a következő kategóriákkal lehet felváltani: *Jó ötlet*, *Fontos*, *Alacsony priorítás* vagy *Kísérleti*. Ez a kategorizálás segít az ötletek fontossági sorrendbe állításában és feldolgozásában. Amennyiben egy javaslat nem igényel további figyelmet, törölhető a rendszerből. A táblázat szűrési lehetőséggel is rendelkezik, így a szervezők cím vagy kategória alapján könnyen megtalálhatják a számukra releváns bejegyzéseket.



3. ábra. Adminisztrációs képernyők: *Főmenü* (a), *Szerepkörkezelés* (b) és *Konfiguráció* (c)

### 2.2.5. Felhasználók menedzselése

A rendszer felhasználói oldalának alapját a regisztráció és a bejelentkezés alkotja. A látogatók saját fiókot hozhatnak létre egy egyedi felhasználónév, a teljes nevük és egy jelszó megadásával, amelynek segítségével további funkciókhoz férhetnek hozzá.

A jogosultságalapú hozzáférés-kezelésnek köszönhetően az egyszerű látogatók és bejelentkezett felhasználók csak a nyilvánosan elérhető tartalmakat böngészhetik, míg a szervezők különböző adminisztratív műveleteket is elvégezhetnek, a hozzájuk rendelt jogosultságok függvényében.

A jogosultságok kiosztása kizárólag az adminisztrátor számára lehetséges, aki rendelkezik a hozzáférés-kezelési jogosultságokkal. Egy táblázatban tekintheti meg a regisztrált felhasználókat, ahol szükség esetén módosíthatja a hozzájuk rendelt jogköröket. Ez a megközelítés biztosítja, hogy a különös figyelmet igénylő funkciók – mint a képek kezelése vagy az események szerkesztése – csak az arra kijelölt és megbízható személyek számára legyenek elérhetők, ezáltal garantálva a rendszer biztonságos és szabályos működését.

### 2.2.6. Bővítménykezelés

A diákszervezet interaktív rendezvényeinek szervezése és lebonyolítása technikai szempontból eltérő platformok használatát követeli meg, jelentős terhet róva a szervezőkre. A weboldal egyik kiemelt funkciója éppen erre nyújt megoldást: a *bővítménykezelő rendszer* lehetővé teszi olyan webalapú alkalmazások fejlesztését és üzemeltetését, amelyek képesek kommunikálni a központi rendszerrel. Ez nem csupán nagyfokú rugalmasságot biztosít a változatos interaktív tartalmak létrehozásához, hanem a hallgatók számára is lehetőséget nyújt a fejlesztési folyamatba való bekapcsolódásra.

A bővítmények életciklusa jól definiált szakaszokra osztható: a fejlesztési folyamat a szervezeti szükségletekből adódó specifikációból indul ki, amelyet a követelmények meghatározása követ. A bővítmények fejlesztésére vonatkozóan két alapvető követelmény érvényes: egyrészt a bővítménynek webalapú alkalmazásként kell működnie – technológiai megkötés azonban nincs, minden olyan megoldás integrálható, amely böngészőben futtatható. Másrészt a fejlesztés során kulcsfontosságú a *kmo-plugin-sdk* nevű, saját fejlesztésű szoftverfejlesztői készlet használata, amely az *npm* csomagregiszterből érhető el. Ez a könyvtár biztosítja a központi szerverrel való kommunikációt és az adatmentéshez szükséges funkciókat.

A bővítményfejlesztői jogosultsággal rendelkező felhasználók feltölthetik és tesztelhetik a saját bővítményüket a weboldalon. Ha a fejlesztő befejezettnek ítéli a munkáját, a bővítményt ellenőrzési státusszal láthatja el. Ettől kezdve az elérhetővé válik az ellenőrzési jogosultsággal rendelkező felhasználók számára, akik manuálisan átvizsgálják a kódot a rosszindulatú tartalmak kiszűrése érdekében. Hibák esetén a bővítmény módosítási státuszba kerül, sikeres ellenőrzés után pedig közzétett státuszt kap, amely lehetővé teszi, hogy eseményekhez kapcsolják.

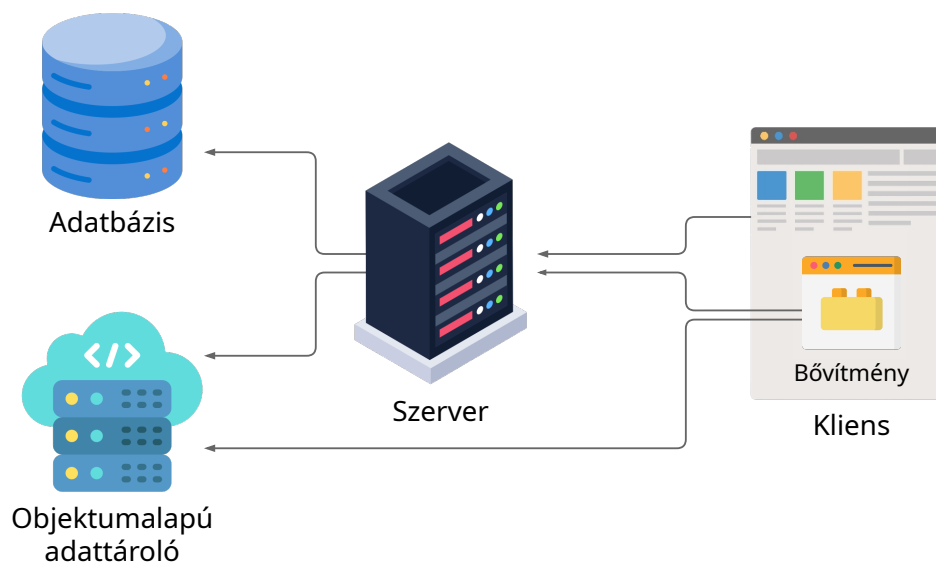
Egy rendezvényhez az eseménykezelésre jogosult tag egyetlen közzétett bővítményt rendelhet hozzá, viszont egy bővítmény több eseményhez kapcsolható, minden esetben önálló állapotkezeléssel. Az esemény publikálásával a hozzárendelt bővítmény is aktiválódik. Mivel a bővítmény felületét az összes látogató elérheti, a fejlesztő feladata a szerepkör-alapú hozzáférés-kezelés megvalósítása, azonban ez független a weboldal jogosultsági rendszerétől. Tipikusan két szerepkör létezik: a bővítmény működését irányító adminisztrátor, valamint a résztvevő. A rendezvény lezárulta után a bővítmény nem törlődik automatikusan, hogy lehetőség legyen például az egyéni pontszámok közzétételére. Egy rendezvény törlése csak akkor lehetséges, ha a hozzárendelt bővítménypéldány már nem aktív. A törlés során a bővítménypéldány is törlődik, az eredeti bővítmény azonban megmarad és újrafelhasználható.

### 3. Architektúra

A projekt architektúráis felépítése, a többretegű architektúra elvét követve, két elsődleges rétegre tagolódik: az adatok perzisztenciájáért és kiszolgálásáért a szerveralkalmazás felel, a felhasználói felület megjelenítésének vezérlése pedig a kliensalkalmazás hatáskörébe tartozik. A bővítményrendszer – bár implementációs szempontból horizontálisan átfedi mindkét réteget – architektúráis komplexitása, valamint a determinisztikus integrációt biztosító mechanizmusainak sajátossága miatt érdemesnek mutatkozik önálló tárgyalásra. A 4. ábra ezen három komponens egymáshoz való viszonyát szemlélteti.

#### 3.1. Szerveralkalmazás

A projekt rendeltetéséből adódóan szükséges egy olyan központi rendszer, amely megfelelően kezeli és tárolja a beérkező adatokat, valamint kiszolgálja a bejövő kéréseket. A szerveralkalmazás ezt a feladatot három egység összehangolt működésével valósítja meg: egy központi szerver, egy relációs adatbázis, valamint egy objektumalapú adattároló együttműködésével. A szerver mediátorként funkcionál a kliensréteg és az adatelérési réteg között, koordinálva az adatok áramlását és érvényesítve az üzleti szabályokat.



4. ábra. Az alkalmazás architektúrája

### 3.1.1. Többrétegű architektúra

A rendszer tervezése során alkalmazott többrétegű – más néven *n-tier* [5] – architektúra lehetővé teszi az egyes rétegek független fejlesztését, tesztelését és szükség esetén cseréjét, mivel a rétegek kizárólag a szomszédos réteg által definiált interfészen keresztül kommunikálnak egymással. A függőségi irány egyirányú: a magasabb szintű rétegek függenek az alacsonyabb szintű rétegektől, viszont ez fordítva nem igaz. A strukturális elkülönítés megfelel a *Separation of Concerns* [22] elvnek, amely az összetett rendszerek kezelhetőségének és karbantarthatóságának egyik alappillére.

A rendszer a következő főbb rétegekre tagolódik:

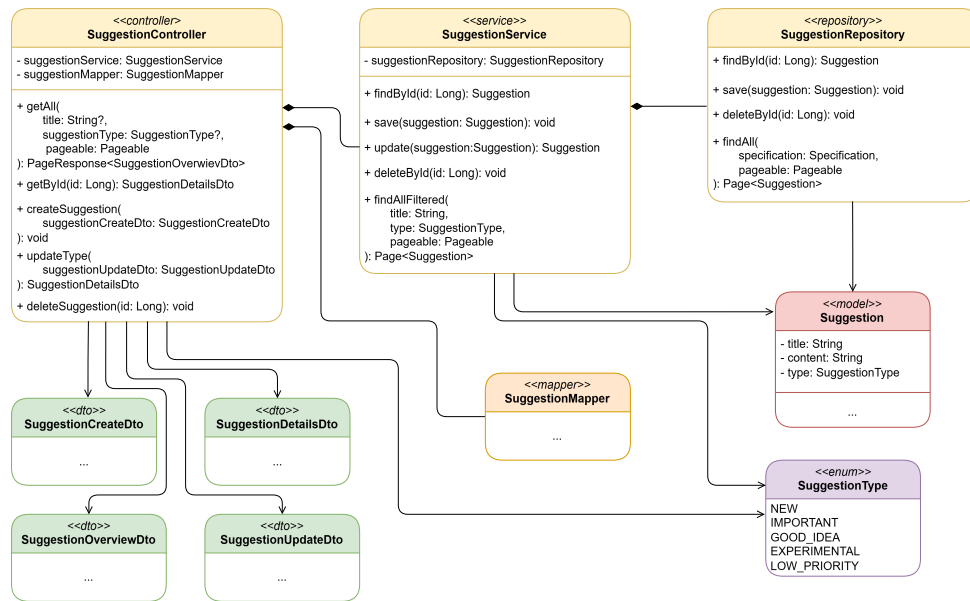
- **Kontroller réteg** – E réteg feladata a bejövő *HTTP*-kérések fogadása, a *DTO*<sup>1</sup> formátumban érkező adatok deszerializálása és az ezekre vonatkozó előzetes megszorítások alkalmazása. A réteg nem tartalmaz üzleti logikát; kizárólagos felelőssége a hálózati interfész és az alkalmazáslogika közötti leképezés.
- **Szerviz réteg** – Ez a réteg implementálja az üzleti logikát, amelyet az adatokon alkalmazni szükséges. Itt zajlanak a komplex validációk, a különböző entitások közötti összefüggések kezelése, valamint az adatok továbbítása az adatelérési réteg felé. A szerviz réteg függetlenül van mind a webes, mind az adatbázis-specifikus implementációs részletektől.
- **Adatelérési réteg** – A réteg feladata az adatok tartós tárolása és strukturált visszakeresése. Ide tartozik a relációs adatbázis, amellyel a rendszer *repository* mintán keresztül kommunikál, valamint az objektumalapú adattároló, amelyet a bináris állományok perzisztálására alkalmaz.

Ahogy ezt a 5. ábra is szemlélteti, egy beérkező kérés életciklusa a következőképpen alakul: a *HTTP*-kérés először a kontroller rétegbe érkezik, ahol a *DTO*-vá deszerializált adatok előzetes ellenőrzésen esnek át. A kontroller ezt követően a szerviz rétegnek adja át a kérést, amely érvényesíti az üzleti szabályokat, majd az adatelérési réteget hívja meg a perzisztencia elvégzéséhez. A válasz ezzel ellentétes irányban halad vissza a klienshez.

A rétegek összekapcsolása a *dependency injection* tervezési mintán alapul, amelynek keretein belül az egyes komponensek függőségeiket nem maguk hozzák létre, hanem azokat a keretrendszer injektálja futási időben. Ez tovább csökkenti a komponensek közötti csatolást, és megkönnyíti az egységtesztelést.

---

<sup>1</sup>**DTO** – Data Transfer Object



5. ábra. A szerveralkalmazás rétegei közötti kapcsolatok a javaslat doménre vetítve

### 3.1.2. Adatkezelés és perzisztencia

A beérkező adatok két csoportba sorolhatók: tulajdonságokat hordozó entitások, illetve bináris állományok. Az entitások jól meghatározott modellek szerint épülnek fel, amelyek segítik azok kezelését, az üzleti szabályok alkalmazását, valamint a relációs adatmodellre épülő adatbázisban való tárolást. A relációs modell alkalmazása lehetővé teszi az entitások közötti összetett kapcsolatok hatékony reprezentációját és lekérdezését. A rendszer az entitásokon a szokásos *CRUD* műveleteket – létrehozás, olvasás, módosítás és törlés – támogatja, amelyeket a *repository* réteg egységes interfészen keresztül tesz elérhetővé a szervíz réteg számára.

A bináris állományok – amelyek a rendszerben képeket vagy forráskódokat képviselnek – kezelése eltér az entitásokétól: mivel nem rendelkeznek üzleti szempontból releváns tulajdonságokkal, validációra és üzleti logikára sincs szükség esetükben. Ezeket a binárisokat a rendszer egy *S3*<sup>2</sup> objektumalapú adattárolóban őrzi. A bináris állományok relációs adatbázisban való tárolása – *BLOB* formátumban – nem hatékony megoldás, mivel jelentősen növeli az adatbázis méretét és rontja a lekérdezési teljesítményt. Ezzel szemben az objektumalapú tárolás skálázható, az objektumok *URL*-alapon közvetlenül kiszolgálhatók, ami csökkenti a központi szerver terhelését (*offloading*). Az adattárolóban minden adat *objektumként* van kezelve, és ezek úgynevezett *bucket*-ekben kerülnek tárolásra, ami lehetővé teszi egy belső logikai struktúra kialakítását is.

A központi szerver *API*-szintű összeköttetéssel rendelkezik az objektumalapú tárolóval,

<sup>2</sup>S3 – Simple Storage Service

amelyen keresztül képes manipulálni a tárolt állományokat, a kiszolgálásuk felelősségét azonban az *S3*-egység veszi át. Ez a munkamegosztás összhangban van a felelősségek szétválasztásának elvével, és hozzájárul a rendszer horizontális skálázhatóságához.

### 3.1.3. Adatmodellek

A diákszervezet sokrétű tevékenységéből adódóan a rendszer több, egymástól részben független adatmodellt foglal magába, amelyek az egyes tematikus doméneket elkülönítetten reprezentálják és kezelik, ahogyan azt a 6. ábra is mutatja.

A modellek közös őse a *BaseEntity*, amely az entitások általánosan jellemző tulajdonságait összesíti. Ebből származik az *AuditEntity* is, amely az entitások létrehozásának és módosításának naplózását szolgálja.

A rendszerbe feltölthető képeket az *Image* modell reprezentálja, az eseményeket pedig az *Event* modell írja le. A kettő között egy-a-többhöz kapcsolat áll fenn: egyetlen eseményhez egy vagy több kép is tartozhat. Az eseményekhez kérdőívek is rendelhetők: az *EventForm* modell az egyes eseményekhez tartozó űrlapokat írja le, a *FormResponse* modell pedig a kérdőívekhez beküldött válaszokat tárolja. A felhasználók adatait a *User* modell határozza meg, a szerepköröket a *RoleTemplate* modell rögzíti, a beérkező javaslatokat pedig a *Suggestion* modell reprezentálja.

A bővítményrendszer több összetartozó modellre épül. A *Plugin* modell a bővítmények definícióját írja le, a *PluginInstance* modell az egyes bővítménypéldányokat határozza meg, a *PluginInstanceData* modell pedig a példányokhoz tartozó egyedi adatokat tárolja. A köztük fennálló kapcsolatok egy-a-többhöz jellegűek: egyetlen *Plugin*-hoz több *PluginInstance* tartozhat, és egyetlen *PluginInstance*-hez több *PluginInstanceData* rendelhető.

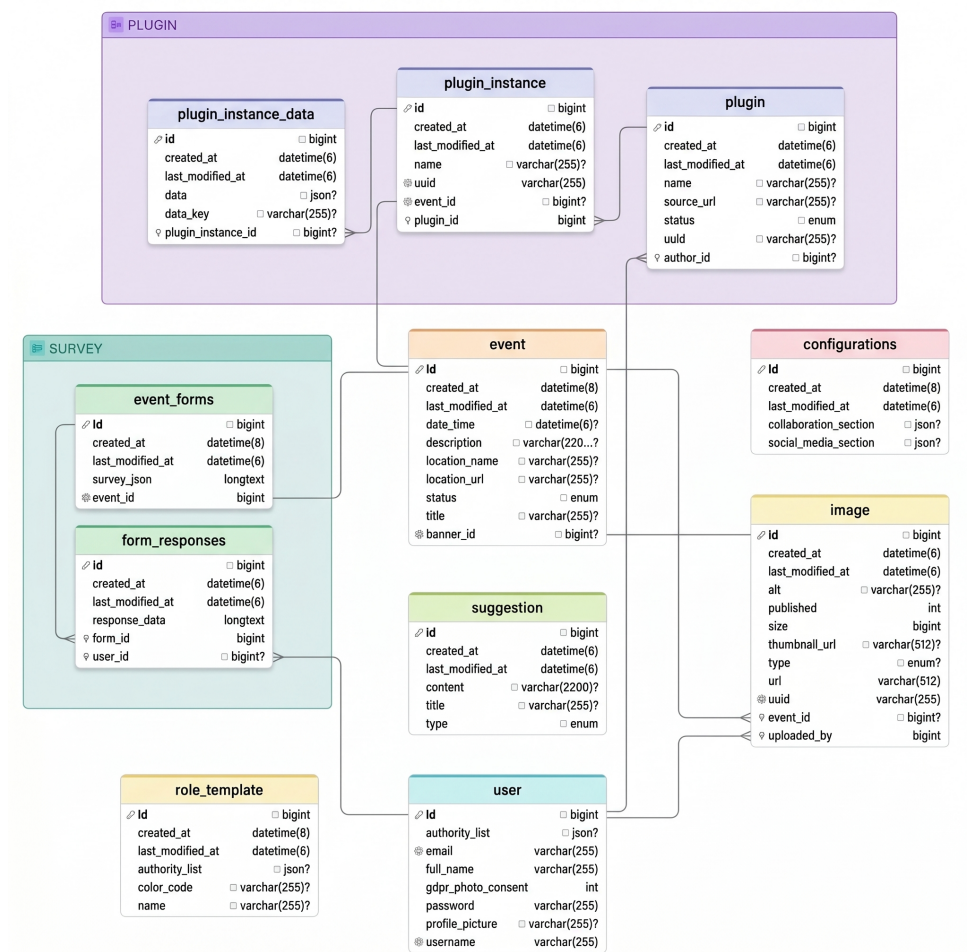
Végül a *Configuration* modell a rendszer konfigurálható adatait – köztük a közösségimédia-hivatkozásokat, a támogatók és együttműködések adatait, valamint a szervezet leírását – foglalja egységes struktúrába.

### 3.1.4. Biztonság és autentikáció

A rendszer biztonságát egy dedikált biztonsági réteg szavatolja, amelynek középpontjában a *JWT*<sup>3</sup> alapú tokenkezelés áll. Minden védett végpontra érkező kérés esetén a szerver validálja a kéréshez csatolt tokent: ellenőrzi annak integritását, lejáratát és az abban foglalt jogosultsági

---

<sup>3</sup>**JWT** – JSON Web Token



6. ábra. Az adatbázis architektúrája

információkat. A tokenvalidálás egy szűrőláncon (*filter chain*) keresztül valósul meg, amely az alkalmazás logikájától függetlenül, transzparensen működik. Nyilvános és hitelesítést igénylő végpontok konfigurálhatóan különíthetők el egymástól.

### 3.1.5. Hibakezelés

A rendszer kétszintű hibakezelési stratégiát alkalmaz. Az első szinten egy *WebExceptionHandler* komponens gondoskodik a rendszer által definiált specifikus kivételek kezeléséről, amelyek jól meghatározott hibaállapotokat – például validációs hibát, hiányzó erőforrást vagy jogosultsági problémát – reprezentálnak. Amennyiben egy kivétel egyik ismert típusba sem sorolható, azt egy *GlobalExceptionHandler* komponens fogja el, amely általános hibaválaszt állít elő. Mindkét kezelő egységes, strukturált *HTTP*-válasszá alakítja a kivételeket, ezáltal az egyes rétegeknek nem kell tudniuk a hálózati megjelenítés részleteiről, ami tovább erősíti a rétegek közötti elszakítottaságot.

## 3.2. Kliensalkalmazás

A kliensalkalmazás a *Single Page Application (SPA)* [21] tervezési mintára épül, amelynek lényege, hogy az alkalmazás egyetlen *HTML*-dokumentumként töltődik be, az ezt követő navigáció során pedig kizárólag a szükséges adatok cserélődnek, teljes lapújrátöltés nélkül. Ezt a megközelítést az alkalmazás jellege indokolja: egy szervezői irányítópult esetében a gyors, megszakítás nélküli munkafolyamat kiemelten fontos, és az állapot elvesztése nélküli navigáció ezt érdemben támogatja.

### 3.2.1. Könyvtárstruktúra és a felelősségek szétválasztása

A könyvtárstruktúra a típus alapú szervezési elvet követi (*type-based structure*), amelyben az egyes könyvtárak nem egy-egy funkcionális domént, hanem az alkotóelemek technikai szerepét tükrözik – például *components*, *hooks*, *models*. Ez az elrendezés kisebb és közepes méretű alkalmazásoknál átlátható marad, és megkönnyíti az újrafelhasználható egységek azonosítását.

Az alkalmazás forráskódja az *src* könyvtáron belül jól elkülönített egységekbe szerveződik, amelyek mindegyike egyértelműen körülhatárolt felelősséggel rendelkezik. A *components* könyvtár a felhasználói felület újrafelhasználható vizuális alapköveit tartalmazza, míg az alkalmazás oldalszintű komponensei a *pages* könyvtárban kapnak helyet.

Az állapotkezelés és a szerverrel való kommunikáció a *hooks* könyvtárban található egyedi *hook*-okon keresztül történik. Ezek olyan újrafelhasználható függvények, amelyek a *React* keretrendszerben az összetartozó állapotkezelési és adatlekérési logikát egyetlen, jól elhatárolt egységbe szervezik – lényegében elvonatkoztatják a komponensektől a szerverrel való kommunikáció részleteit, így azok kizárólag a megjelenítési feladataikra koncentrálhatnak. Minden egyes erőforrástípushoz – legyen szó eseményekről, képekről, bővítményekről vagy felhasználókról – egy dedikált *hook* tartozik, amely egységes és újrafelhasználható interfészt biztosít az adatlekérdezéshez és -manipulációhoz. A *models* könyvtár a *TypeScript* típusdefiníciókat foglalja magába, amelyek tükrözik a szerveralkalmazás által kezelt entitásmodelleket, így garantálva a típusbiztonságot a kliens és a szerver közti adatszere során.

A *store* könyvtárban található a globális állapottárolók, amelyek a bejelentkezett felhasználók adatait és az aktív témát tartják nyilván. A szerverrel való *HTTP*-kommunikáció a *config* könyvtárban található, központi *Axios* példányon keresztül valósul meg, amely gondoskodik a kérések előfeldolgozásáról, az autentikációs fejléc csatolásáról és az egységes

hibakezelésről.

A *constants* könyvtár a navigációs menüelemek definícióit tartalmazza, amelyek a különböző felhasználói szerepkörökhöz igazodva határozzák meg az elérhető menüpontokat. A *utils* könyvtár általános segédfüggvényeket és típusdefiníciókat gyűjt egybe, amelyek az alkalmazás több pontján is újrafelhasználhatók – ide tartoznak például a dátumformázó, a fájlméret-konvertáló és a tömbkezelő segédeszközök.

### 3.2.2. Navigáció és útvonalkezelés

Az alkalmazás útvonalai a *navigation/routes* könyvtárban kerülnek deklaratív módon definiálásra, elkülönítve az elrendezést vezérlő komponenstől. Ez az elkülönítés összhangban áll a *Separation of Concerns* elvével: az útvonalak konfigurációja és az elrendezés megjelenítési logikája egymástól függetlenül módosítható. A navigációs komponens két eltérő nézetet kezel – a látogatói felületet és a szervezői irányítópultot –, amelyek között a váltás a felhasználó jogosultságai és a globális állapot alapján történik.

### 3.2.3. Adatlekérés és gyorsítótárazás

A szerverrel való kommunikáció az *SWR* könyvtáron keresztül valósul meg. Az *SWR* névadója a *stale-while-revalidate* gyorsítótár-stratégia, amelyet az *RFC 5861*-es szabvány [20] formalizál: a kliens elsőként a gyorsítótárban tárolt – esetleg elavult – adatot szolgálja ki azonnali megjelenítés céljából, miközben a háttérben elvégzi az újravalidálást, majd szükség esetén frissíti a megjelenített tartalmat. Ez a kétlépéses megközelítés csökkenti az észlelt töltési időt (*perceived latency*), és javítja az alkalmazás reszponzivitását anélkül, hogy az adatok konzisztenciáját feláldozná. Az egyedi *hook*-ok mindegyike az *SWR*-re épül, így az adatlekérési logika egységes és újrafelhasználható marad az egész kliensalkalmazásban. A galéria oldalon az *useSWRInfinite hook* kerül alkalmazásra, amely a lapozósos lekérdezés és az *infinite scroll* együttes megvalósítását teszi lehetővé: az egyes oldalak eredményei fokozatosan kerülnek összefűzésre, miközben az előzőleg betöltött lapok gyorsítótárban maradnak.

### 3.2.4. Teljesítményoptimalizálás

A kliensalkalmazás implementálása során kiemelt figyelmet kapott a teljesítmény tudatos optimalizálása. Az adatok lapozósos lekérdezéssel (*pagination*) kerülnek betöltésre, ami csökkenti mind a kezdeti válaszidőt, mind a szerver terhelését – fennáll a kompromisszum, hogy

a teljes adatkészlet egyszerre nem érhető el, ez azonban a legtöbb felhasználói interakciónál nem jelent hátrányt.

A képmegjelenítés kétlépéses stratégiát követ: a rendszer elsőként miniatűr előnézeti képeket tölt be, és a teljes felbontású változat csak a felhasználó explicit kérésére – egy *lightbox* megnyitásakor – kerül letöltésre. Ez a megközelítés a *lazy loading* elvét alkalmazza [11], és különösen nagy képszám esetén jelentős sávszélesség-megtakarítást eredményez, mivel a soha meg nem nyitott képek teljes felbontású változatai nem kerülnek átvitelre.

A *JWT*-token kliensoldali dekódolása szintén tudatos tervezési döntés: a jogosultság ellenőrzése helyben, plusz hálózati kérés nélkül elvégezhető. Ez azonban nem helyettesíti a szerveroldali validációt – minden végpont kötelezően validálja a token, az integritás, a lejárat és a jogosultságok ellenőrzésével. Ezáltal a kliensoldali dekódolás kiegészítő szerepet tölt be: gyorsabbá válik a kliensoldali logika, a biztonság pedig nem sérül.

### 3.3. Bővítményrendszer

Az alkalmazás legnagyobb architektúráis kihívása a központi rendszertől különálló, független webalapú programok – úgynevezett bővítmények – integrálása. Szükséges tehát egy olyan komponens, amely lehetőséget nyújt adatok feltöltésére, tárolására és kiszolgálására futási időben. E komponensnek megfelelően általános felületet kell biztosítani az integrálandó alkalmazások számára, mivel azokra *black-box*<sup>4</sup> rendszerekként kell tekinteni. Ezen a felületen keresztül valósul meg a bővítmények és a szerver közötti kommunikáció, amelynek révén az alkalmazások adatokat tudnak kezelni, illetve feliratkozás alapú üzenetküldést tudnak megvalósítani. A projektben a *bővítményrendszer* tölti be ezt a szerepet.

#### 3.3.1. Szoftverfejlesztői csomag

Mivel a bővítmények bárki által elkészíthetők, elengedhetetlen egy egyszerű és egységes segédeszköz, amelynek segítségével a fejlesztők felkészíthetik alkalmazásukat a zavartalan, integrált működésre. Erre a célra készült a *kmo-plugin-sdk* szoftverfejlesztői készlet (*SDK*)<sup>5</sup>, amely az *npm* csomagkezelő és -nyilvántartó rendszer online tárolóján keresztül bárki számára szabadon elérhető<sup>6</sup>. A csomag névterek és függvények gyűjteményét tartalmazza, amelyek *HTTP*, illetve *WebSocket* kommunikáció útján kapcsolatot létesítenek a központi

---

<sup>4</sup>**black-box** – olyan rendszer vagy folyamat, amelynek belső működése ismeretlen vagy nem látható

<sup>5</sup>**SDK** – Software Development Kit

<sup>6</sup><https://www.npmjs.com/package/kmo-plugin-sdk>

szerverrel. Bár az egyes kérések eltérő célokat szolgálnak, valamennyi művelet két alapvető kategóriába sorolható: adatkezelés és feliratkozás alapú kommunikáció. Ezeket a funkciókat a *KMOPluginClient* névtér fogja össze. A névtéren belül további alnévterek találhatók, emellett olyan metódusok is rendelkezésre állnak, amelyekkel inicializálható a bővítmény környezete a szerverrel való kommunikációhoz, valamint meghatározhatók a bővítményen belüli szerepkörök. A *KMOPluginClient.data* alnévtér lehetőséget biztosít adatok mentésére, lekérdezésére és törlésére. A széles körben elterjedt *JSON* formátumot alkalmazva minden tárolt adathoz egy fejlesztő által meghatározott kulcs tartozik, amelynek segítségével az adatok a szerveren perzisztálhatók. A feliratkozás alapú kommunikáció *WebSocket* protokollon keresztül valósul meg, amelyhez a csomag könnyen használható megoldásokat nyújt. A *KMOPluginClient.message* alnévtérben implementált metódusokkal tesztelhető a hálózati kapcsolat, valamint elvégezhető a fel- és leiratkozás. A */topic/event-listen* címre feliratkozott kliensekhez eljut minden olyan üzenet, amelyet a */event-trigger* végpontra küldenek.

### 3.3.2. Adatmodell és entitáskezelés

A háttérrendszer felépítése meghatározó szerepet játszik az összetett architektúrában, mivel a bővítmények entitásként való kezelése sajátos tervezési kérdéseket vet fel. Egyrészt szükséges minden bővítményt önálló, egyedi entitásként nyilvántartani, másrészt egy bővítmény több eseményhez is hozzárendelhető, és ilyenkor minden egyes példányának különálló életciklussal és adatkészlettel kell rendelkeznie. A megoldás e két fogalom pontos elkülönítésén alapul. A rendszer minden feltöltött bővítményt egyedi entitásként tart számon – rögzítve annak nevét, státuszát és forráskódját; ezt az entitást *Plugin*-nak nevezzük. A bővítmény forráskódja egy *S3* objektumtárolóban kerül tárolásra, amely lehetővé teszi a kód megfelelő karbantartását és a központi szervertől függetlenül történő kiszolgálását. Amikor egy bővítményt egy eseményhez rendelnek, valójában annak egy példánya jön létre, amely az esemény törléséig önálló entitásként él a rendszerben. Ezt az entitást *PluginInstance*-nek nevezzük.. Minden példány egyedi azonosítóval rendelkezik. Ahhoz, hogy a bővítmény megfelelően tudjon kommunikálni és adatokat menteni, szükséges ismerje az azonosítóját, ezért ez a beágyazás folyamata által jut el hozzá. Ez teszi lehetővé a példányokhoz kötött adatperzisztenciát, valamint a bővítményeken belüli elkülönített kommunikációt. További entitás a *PluginInstanceData*, amelyből egy vagy több is hozzárendelhető egy-egy példányhoz; ezek valósítják meg a kulcs-érték alapú adattárolást.

A fentiek alapján a bővítményrendszer legfőbb architekturális egységei a szoftverfejlesztői csomag, az integrálandó alkalmazás kódja, valamint a *Plugin*, *PluginInstance* és *PluginInstanceData* entitások. A bővítmény az SDK segítségével, a megfelelő végpontokat elérve képes manipulálni az említett entitásokat, ezáltal befolyásolva saját reprezentációját és adatait a központi szerveren – mindezt úgy, hogy teljes mértékben független marad a rendszer belső működésétől.

## 4. Felhasznált technológiák és eszközök

A fejlesztés során alkalmazott technológiai döntések minden esetben az adott feladat természetéhez és a rendszer hosszú távú karbantarthatóságához igazodtak. Az alábbiakban bemutatásra kerülnek a szerveroldali és kliensoldali keretrendszerek, az adattárolási megoldások, az infrastrukturális eszközök, valamint a fejlesztési folyamatot támogató módszerek.

### 4.1. Szerveroldali technológiák

A projekt célja egy megbízható és könnyen karbantartható háttérrendszer kialakítása. Ennek érdekében a szerveralkalmazás tervezése során alkalmazott eszközök egy ipari standardnak tekinthető, széles körben elterjedt technológiai készletből származnak, amelyek együttműködése hosszú évek alatt érett és jól dokumentált megoldásnak bizonyult.

**Spring Boot** A szerver működtetését a *Spring Boot* [25] keretrendszer biztosítja. A *Spring Framework*-re épülő, véleményalapú (*opinionated*) keretrendszer jelentősen leegyszerűsíti a modern Java nyelvben írt alkalmazások fejlesztését. Főbb jellemzői az *IoC* (*Inversion of Control*), a függőséginjektálás, az automatikus konfiguráció, valamint a beágyazott szerver támogatása. A keretrendszer moduláris felépítésű, a modulok kezelését az ún. *Spring Boot Starter* csomagok teszik egyszerűvé. A projektben az alábbi modulok kerültek felhasználásra:

- *spring-boot-starter-data-jpa* – A *JPA* és *Hibernate ORM* integrációját biztosítja, amelyek által a projektben az entitások perzisztenciájában és a *CRUD* műveletek elvégzésében van szerepe.
- *spring-boot-starter-validation* – Annotáció alapú adatvalidációt tesz lehetővé, a projektben a *REST API* végpontokra érkező kérések ellenőrzésére szolgál.
- *spring-boot-starter-web* – A *Spring MVC* keretrendszert és a beágyazott *Tomcat* webkonténer biztosítja, valamint alapértelmezetten tartalmazza a *Jackson JSON* szerializációs könyvtárat. A projektben az *API* végpontok kiszolgálásánál került alkalmazásra.
- *spring-boot-starter-security* – Hitelesítési és jogosultságkezelési mechanizmusokat biztosít. A projektben a hitelesítés és jogosultságkezelés megvalósítását biztosítja.

- *spring-boot-starter-websocket* – Kétirányú, valós idejű kommunikációt tesz lehetővé a szerver és a kliensek között, így a projektben a bővítményeken belüli vezérlés megvalósítására szolgál.

**JWT** A *JSON Web Token (JWT)* [15] egy nyílt szabványon (*RFC 7519*) alapuló, kompakt token formátum, amely három részből áll: a fejlécből (*header*), a törzsből (*payload*) és az aláírásból (*signature*). Állapotmentes (*stateless*) hitelesítést tesz lehetővé, mivel a hitelesítéshez szükséges adatok magában a tokenben találhatók, munkamenet tárolása nélkül. A projektben a *Spring Security*-vel együttműködve a végpontok védelmére és a jogosultságok ismertetésére van alkalmazva – sikeres bejelentkezés után a szerver egy aláírt tokent állít ki, amelyet a kliens minden kérés *Authorization* fejlécében *Bearer* séma szerint továbbít:

**Authorization:** Bearer <token>

## 4.2. Kliensoldali technológiák

A kliensalkalmazás tervezése során alkalmazott eszközök egy modern, széles körben elterjedt technológiai készletből származnak, amelyek együttesen biztosítják a felhasználói felület hatékony fejlesztését, a típusbiztonságot és a megfelelő teljesítményt.

**TypeScript** A kliensalkalmazás *TypeScript* [19] nyelven íródott, amely tekinthető különálló programozási nyelvnek, de akár úgy is, mint a *JavaScript* típusinformációval bővített kiterjesztése. A fordítási idejű típusellenőrzés lehetővé teszi a hibák korai felismerését, és jelentősen javítja a kód karbantarthatóságát. A projektben különösen fontos szerepet tölt be a *models* könyvtárban definiált típusok révén, amelyek pontosan tükrözik a szerver által kezelt entitásokat, így a kliens és a szerver közötti adatcsere típusbiztos módon valósul meg.

**React** A felhasználói felület megjelenítéséért a *React* [18] könyvtár 19-es verziója felel. A komponensalapú szemlélet lehetővé teszi az újrafelhasználható felületelemek kialakítását, amelyek jól illeszkednek az alkalmazás moduláris könyvtárstruktúrájához. A *React hook*-mechanizmusa az egyedi *hook*-ok alapját képezi, amelyek egységes interfészt biztosítanak az adatlekérdezéshez és az állapotkezeléshez.

**Vite** A fejlesztői környezet kiszolgálását és a statikus erőforrások összeállítását a *Vite* [14] eszközlánc végzi. Natív *ES*-modulokra épülő architektúrájának köszönhetően a fejlesztői szerver indítási ideje töredéke a hagyományos megoldásokénak. A *build* folyamat során a *TypeScript* fordító elvégzi a szigorú típusellenőrzést, majd a *Vite* végrehajtja a *bundling* folyamatot, amelynek eredményeként a *React* komponensek és a *CSS* modulok böngésző-kompatibilis statikus állományokká alakulnak át.

**React Router** Az alkalmazás útvonalkezelését a *React Router* [4] biztosítja, amely lehetővé teszi az egyoldalas webalkalmazáson belüli deklaratív navigációt. Az útvonalak logikailag elkülönítve vannak definiálva, és a védett útvonalak esetén az útvonal-szintű jogosultság-ellenőrzés is ezen a rétegen keresztül érvényesül.

**SWR** A szerverrel való adatszinkronizációt az *SWR* [24] könyvtár koordinálja, amelynek neve a *stale-while-revalidate* gyorsítótárazási stratégiára utal. A könyvtár a tárolt, esetleg elavult adatot azonnal visszaadja a felhasználói felületnek, miközben a háttérben elvégzi a frissítést – ezáltal a felhasználó számára az alkalmazás mindig reagál, és az adatok mégis naprakészek maradnak. A projektben minden erőforrástípushoz egy dedikált *hook* tartozik, amely az *SWR*-t alkalmazza az adatlekérések kezelésére.

**Axios** A *HTTP*-kérések küldéséért az *Axios* [8] könyvtár felel, amelyet a projektben egy központi példány formájában kerül alkalmazásra. Ez a példány köztes feldolgozó mechanizmusok (*interceptor*-ok) segítségével gondoskodik a *JWT* autentikációs fejléc automatikus csatolásáról minden kimenő kéréshez, valamint az egységes hibakezelésről, így az egyes *hook*-oknak nem kell tudniuk a hálózati kommunikáció részleteiről.

**unstated-next** A globális állapot kezelésére az *unstated-next* [16] könyvtár van alkalmazva, amely a *React Context API*-ra épülő, minimális többletterheléssel járó állapotkezelési megoldást kínál. A projektben három globális tároló található: a *UserStore* a bejelentkezett felhasználó adatait és jogosultságait tárolja, a *ThemeStore* az aktív színsémát tartja nyilván, a *PluginStore* pedig a bővítmények aktuális állapotát kezeli.

**Ant Design** A felhasználói felület komponenskészletét az *Ant Design* [1] könyvtár biztosítja, amely egy átfogó, vállalati szintű *React* komponensgyűjtemény. A projektben táblázatok, űrlapok

és más összetett vizuális elemek megvalósítására szolgál, lehetővé téve, hogy a fejlesztés az üzleti logikára összpontosíthasson az alapkomponeensek újraírása helyett.

**SurveyJS** Az eseményekhez csatolható, testre szabható jelentkezési űrlapokat a *SurveyJS* [9] könyvtár valósítja meg. A *SurveyJS* egy nyílt forráskódú, *JSON*-alapú űrlapmotor, amely lehetővé teszi összetett, dinamikusan konfigurálható kérdőívek és űrlapok megjelenítését és kitöltését. A projektben két fő modulja került felhasználásra: a *survey-core* a *JSON*-alapú sémából értelmezi és kezeli az űrlap struktúráját és állapotát, míg a *survey-react-ui* a *React*-kompatibilis kirajzolási réteget biztosítja. Ez a megközelítés lehetővé teszi, hogy a szervezők rugalmasan bővíthessék az egyes eseményekhez tartozó jelentkezési űrlapokat – például további mezők hozzáadásával –, anélkül hogy ehhez bármilyen forráskód-módosítás szükséges volna, mivel az űrlap teljes struktúrája *JSON* formátumban kerül tárolásra és kiszolgálásra.

**jwt-decode** A szerver által kiállított *JWT token* kliensoldali feldolgozásáért a *jwt-decode* [3] könyvtár felel: lehetővé teszi a *token* törzsének dekódolását, amelyből kinyerhetők a bejelentkezett felhasználóra vonatkozó adatok – köztük a hozzárendelt jogosultságok listája. Ezeket az adatokat a *UserStore* tárolja, és az alkalmazás különböző pontjain felhasználja a megfelelő felületelemek megjelenítéséhez vagy elrejtéséhez.

### 4.3. Adattárolási megoldások

A projekt adatainak tárolása két elkülönített megoldásra támaszkodik: a strukturált relációs adatok – mint az entitások és kapcsolataik – egy relációs adatbázisban, míg a bináris fájlok, vagyis a képek és a forráskódok, egy objektumtároló rendszerben kerülnek elhelyezésre.

**Hibernate és ORM** Az objektum-relációs leképezés (*Object-Relational Mapping, ORM*) egy programozási technika, amely lehetővé teszi, hogy a relációs adatbázis tábláit és azokat sorait közvetlenül *Java* objektumokként kezeljük, elkerülve a manuális *SQL* lekérdezések írását. A *Hibernate* [6] a legelterjedtebb *ORM* keretrendszer a *Java* alkalmazásokhoz. A fentebb említett *Spring Boot Starter Data JPA* modul alapértelmezett implementációjaként a *Hibernate* végzi az entitásobjektumok és az adatbázistáblák közötti leképezést, kezeli a tranzakciókat, valamint az adatbázis-sémát az entitásosztályok alapján automatikusan karbantartja.

**MySQL** A projekt relációs adatainak tárolására a széles körben elterjedt, nyílt forráskódú *MySQL* [10] relációs adatbáziskezelő rendszer (*RDBMS*) van alkalmazva, amely strukturált adatok táblákban való tárolására és hatékony lekérdezésére alkalmas. A projektben az összes entitás – mint például a események, felhasználók és egyéb doménobjektumok – adatai *MySQL* adatbázistáblákban kerülnek perzisztálásra. Az alkalmazás és az adatbázis közötti kommunikációt a *Hibernate ORM* réteg közvetíti, így az adatbázis szerkezete közvetlenül az entitásosztályok definícióiból származik.

**MinIO** A *MinIO* [2] egy nagy teljesítményű, nyílt forráskódú objektumtároló rendszer (*object storage*), amely az *Amazon S3 API*-val kompatibilis interfészt biztosít. Míg a relációs adatbázisok strukturált adatok perzisztenciájára alkalmasak, a bináris vagy nagyméretű fájlok – mint képek és forráskódfájlok – hatékonyabb tárolására objektumtároló megoldás szükséges. A projektben a felhasználók által feltöltött képek és forráskódok megőrzésére a *MinIO* szolgál, amely lokálisan futtatott szerverként üzemel. Az alkalmazás a *MinIO Java* kliens könyvtárán keresztül kommunikál a szerverrel, a fájlok feltöltését, letöltését és kezelését pedig *S3*-kompatibilis *API* hívások útján végzi.

**imgproxy** Az *imgproxy* [13] egy nagy teljesítményű, nyílt forráskódú képfeldolgozó *proxy* szerver, amelynek elsődleges feladata képek valós idejű transzformációja és optimalizált kiszolgálása. A projektben az *imgproxy* az objektumtároló elé van helyezve, és *S3*-kompatibilis interfészen keresztül közvetlenül onnan olvassa be a feltöltött képeket. Egy kép feltöltésekor a rendszer az *imgproxy* segítségével legenerálja a csökkentett felbontású előnézeti változatot, amelyet ezt követően a teljes felbontású képpel együtt az objektumtárolóba ment el. Ez az iparban széles körben alkalmazott, ún. *eager generation* stratégia: a transzformáció költsége a feltöltés pillanatára van tolvá, cserébe a kiszolgálás során a rendszer már csupán a megfelelő előre elkészített fájlt adja vissza, mindenféle képfeldolgozás nélkül. A biztonságos működést az *HMAC-SHA256* algoritmussal előállított *URL*-aláírás szavatolja: a szerver a megadott kulcs és só segítségével aláírja a transzformációs paramétereket tartalmazó útvonalat, az *imgproxy* pedig minden kérés esetén ellenőrzi ezt az aláírást, megakadályozva ezzel, hogy illetéktelenek tetszőleges transzformációs kéréseket intézzenek a szerver felé.

## 4.4. Infrastrukturális eszközök

Az alkalmazás megbízható és reprodukálható üzemeltetéséhez elengedhetetlen egy jól definiált infrastruktúra. Az alábbiakban bemutatásra kerülnek azok az eszközök, amelyek a lokális fejlesztési környezet kialakítását és az éles rendszer üzemeltetését egyaránt biztosítják.

**Docker** A projekt hordozhatóságának és környezeti konzisztenciájának biztosítása érdekében az alkalmazás konténerizálása *Docker* [17] segítségével valósul meg. A *Docker* lehetővé teszi, hogy az alkalmazás és annak összes függősége – a futtatókörnyezettől a konfigurációs beállításokig – egyetlen, önálló egységbe legyen csomagolva, amely minden környezetben azonos módon viselkedik. Ez különösen értékes egy több komponensből álló rendszer esetén, mint amilyen a jelen projekt is: a serveroldali alkalmazás, az adatbázis, az objektumtároló, valamint az *imgproxy* képkiszolgáló mind elkülönített konténerekben futnak, amelyek szabványosított interfészeken keresztül kommunikálnak egymással. A fejlesztői környezetben a *Docker Compose* egyetlen paranccsal elindítja a teljes rendszert, lerövidítve az új fejlesztők beilleszkedési idejét.

**Kubernetes** A produkciós környezetben történő üzemeltetés *Kubernetes* [7] klaszteren keresztül valósul meg, amely az iparágban széles körben elterjedt, nyílt forráskódú konténer-orkesztrációs rendszer. A *Kubernetes* olyan kritikus üzemeltetési igényeket kezel automatizáltan, amelyek manuálisan nehezen lennének fenntarthatók: gondoskodik a konténerek életciklusáról, automatikusan újraindítja a leállt példányokat, és lehetővé teszi a horizontális skálázást a terhelés függvényében. A platform szempontjából különösen fontos, hogy a *Kubernetes Secrets* mechanizmusa révén az érzékeny konfigurációs adatok – mint az adatbázis jelszava vagy az objektumtároló hozzáférési kulcsai – elkülönítve, titkosítva tárolhatók, és csak futási időben kerülnek a konténerek környezetébe. Ez a megközelítés nemcsak a biztonságot erősíti, hanem a karbantarthatóságot is: az infrastruktúra konfigurációja verziókezel, reprodukálható formában leírható, így a rendszer állapota bármikor visszaállítható vagy egy új klaszterre mozgatható.

## 4.5. Fejlesztői eszközök és módszerek

A technológiai döntések mellett a fejlesztési folyamat szervezése és az alkalmazott munkamódszerek is meghatározó szerepet játszottak a projekt sikerességében.

**Agilis fejlesztési módszertan** A projekt fejlesztése a *Scrum* [23] keretrendszer alapelveit követve, agilis módszertannal valósult meg. A *Scrum* egy iteratív és inkrementális fejlesztési keretrendszer, amelynek középpontjában a rövid, jól körülhatárolt ciklusok – ún. sprintek – állnak. Minden sprint egy kéthetes időszakot ölelt fel, amelynek végére egy működő és demonstrálható szoftvertermék készült. Ez a megközelítés lehetővé tette, hogy a fejlesztési prioritások rugalmasan igazodjanak a diákszervezet visszajelzéseihez.

A sprint mindig a tervezéssel indult, amelynek során a csapat kiválasztotta a termék teendői közül az adott iterációban megvalósítandó feladatokat. A fejlesztési ciklus végén pedig a csapat bemutatta a kész funkionalitásokat a témavezetőknek és a diákszervezet képviselőinek, amit egy retrospektív megbeszélés követett, amelyen a csapat értékelte az elvégzett munkát és azonosította a fejlesztendő területeket.

**Verziókövetés és tesztelés** A verziókövetés és a csapatmunka koordinálása a *GitLab*-on keresztül valósult meg. A forráskód a *Git* verziókövető rendszer segítségével került tárolásra, amely lehetővé tette a munkafolyamatok párhuzamosítását: a fejlesztés egymástól izolált ágakon történt, majd a változtatások egy *merge request* formájában integrálódtak a főágba. Ez a munkamenet biztosítja, hogy minden kódbeli változtatás egy felülvizsgálaton essen át, mielőtt bekerülne a közös kódbázisba, ezáltal csökkentve a hibák bekerülésének kockázatát.

A feladatok nyilvántartása a *GitLab* beépített *issue*-kezelő rendszerén keresztül történt. Minden tervezési egység – legyen szó új funkionalitásról vagy hibajavításról – egy-egy *issue* formájában lett rögzítve, amelyhez cím, leírás, prioritás és felelős személy rendelhető. Az *issue*-k mérföldkövekhez rendelve képezték le az egyes sprinteket, így a fejlesztési folyamat átlátható és nyomon követhető maradt.

A *GitLab* folyamatos integrációs és kitelepítési (*CI/CD*) rendszere is igénybe lett véve a fejlesztés során, amelynek konfigurációja a *.gitlab-ci.yml* állományban kerül meghatározásra. A *CI/CD pipeline* automatikusan lefut minden egyes kódváltoztatás hatására: elvégzi a *build* folyamatot, lefuttatja a teszteket, majd sikeres ellenőrzés esetén előkészíti a kitelepítésre szánt *artifact*-eket. Ez a megoldás biztosítja, hogy a kódbázis minden pillanatban kitelepíthető állapotban legyen, és a hibák minél korábban, automatizált módon kerüljenek a felszínre.

**Fejlesztői környezet** A fejlesztés során a csapat a *JetBrains* fejlesztőkörnyezetét, az *IntelliJ IDEA*-t használta, mivel kiterjedt refaktorálási és hibakeresési funkciókat kínál, amelyek hozzájárulnak a munkafolyamat hatékonyságához.

## 5. Biztonság

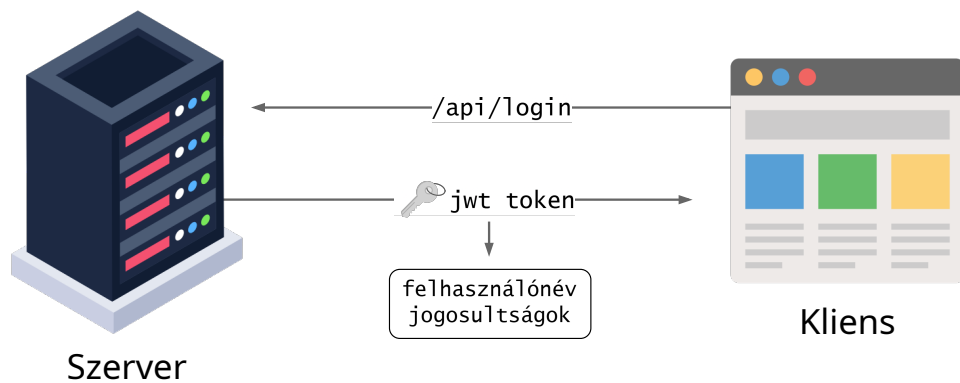
A projekt összetett jogosultságkezelési rendszeréhez igazodva elengedhetetlen egy ezzel egyenértékű biztonsági architektúra kialakítása, amely megfelelően valósítja meg a hitelesítést és a hozzáférés-vezérlést. Ez a biztonsági rendszer az érintett entitások szempontjából két jól elkülöníthető részre tagolható: rendszerszintű és bővítményszintű komponensre. A rendszerszintű biztonság a felhasználók hitelesítéséért és a bejövő kérések validálásáért felelős, míg a bővítményszintű biztonság a bővítmények által kezdeményezett kérések megfelelő érvényesítését biztosítja.

### 5.1. Rendszerszintű biztonság

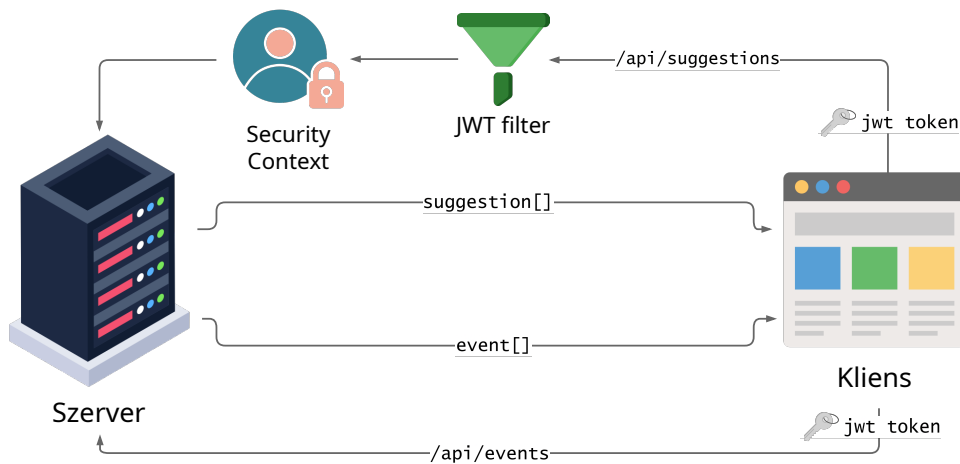
A rendszerszintű biztonság kialakításakor elsőként szükséges különbséget tenni a nyilvánosan elérhető és a hitelesítést igénylő végpontok között. Az előbbi kategóriába tartoznak például a publikus események lekérdezéséhez, képek megtekintéséhez vagy javaslatok beküldéséhez szükséges végpontok, amelyekhez a felhasználónak nem szükséges azonosítania magát. Az utóbbi kategóriába azon végpontok sorolhatók, amelyek érzékenyebb műveleteket végeznek, és amelyeknél a jogosultságok ellenőrzése elengedhetetlen.

A hozzáférés-vezérlés megvalósításához a rendszer a *Spring Security* keretrendszer szűrőlánc-mechanizmusát veszi alapul, amelyet egy egyedi szűrővel egészít ki. Ez a szűrő felépíti a kérés életciklusa alatt érvényes belső biztonsági modellt, az úgynevezett *Security Context*-et (lásd 8. ábra). A *Security Context* egy szálsként izolált tárolóként funkcionál, amely az aktuális felhasználó hitelesítési adatait és jogosultságainak listáját tartalmazza. A keretrendszer ezt a kontextust a kérés teljes feldolgozási folyamata során elérhetővé teszi az alkalmazás valamennyi rétege számára, így a jogosultság-ellenőrzés bármely ponton következetesen elvégezhető. A kérés lezárultával a *Security Context* törlődik, ezáltal megakadályozva az adatok véletlen kiszivárgását egymást követő kérések között.

A jogosultság-ellenőrzés metódusszinten, a szerviz rétegben elhelyezett `@PreAuthorize(hasAuthority(...))` annotációk segítségével valósul meg. Ezek az annotációk deklaratív módon határozzák meg, hogy az adott művelet elvégzéséhez milyen jogosultság szükséges, és a metódus végrehajtása előtt kiértékelésre kerülnek. Amennyiben a *Security Context*-ben tárolt jogosultságok nem felelnek meg az annotációban meghatározott feltételeknek, a keretrendszer a hívást megszakítja és kivételt dob, így a jogosulatlan hozzáférés



7. ábra. Bejelentkezés



8. ábra. Hitelesítés az /api/suggestions és /api/events végpontokon

már a szerviz réteg szintjén megakadályozható. Ez a megközelítés egyrészt átlátható és karbantartható jogosultság-kezelést tesz lehetővé, másrészt biztosítja, hogy az üzleti logika ne legyen közvetlenül felelős a hozzáférés-vezérlési döntésekért.

A felhasználók hitelesítése *JSON Web Token (JWT)* alapú mechanizmuson keresztül történik. A *JWT* egy nyílt szabványon (*RFC 7519*) alapuló, kompakt és önleíró token formátum, amely digitálisan aláírt formában képes hordozni a felhasználóra vonatkozó állításokat (*claim*-eket). A rendszer a sikeres bejelentkezést követően állítja elő a token, amelybe beépíti a felhasználó bejelentkezés pillanatában érvényes jogosultságait (lásd 7. ábra). Ez azt jelenti, hogy a token a kiállítás időpontjában fennálló állapotot rögzíti, és az abban szereplő jogosultságok a token érvényességi idején belül változatlanok maradnak, függetlenül az esetleges jogosultság-módosításoktól.

A rendszer a *HMAC-SHA256* aláírással ellátott token a *localStorage*-ben őrzi. Habár ez *JavaScript* kódból közvetlenül hozzáférhető, a szerver oldali aláírás-validáció biztosítja, hogy módosított token nem lehet felhasználni. A felhasználó az aláírt token minden egyes

*HTTP* kérés fejlécének *Authorization* mezőjében, *Bearer* tokenként küldi el a szervernek, amely azt minden kérésnél ellenőrzi és feldolgozza: mivel a *JWT* token előállításához és aláírásához szükséges kulcs kizárólag a szerver számára ismert, harmadik fél nem képes érvényes token létrehozni vagy meglévőt módosítani anélkül, hogy az aláírás érvénytelenné ne válna. Érvénytelen, lejárt vagy manipulált token esetén a szűrőlánc a kérést azonnal elveti, még mielőtt az az alkalmazás belső rétegeibe jutna, így a jogosulatlan hozzáférési kísérletek hatékonyan és korán kiszűrhetők.

## 5.2. Bővítményszintű biztonság

A bővítményszintű biztonság a rendszerszintű megoldáshoz hasonló elveken nyugszik, azonban az érintett entitás természetéből adódóan eltérő megközelítést igényel. A legfontosabb biztonsági követelmény, hogy minden bővítmény kizárólag a saját adatait legyen képes elérni és módosítani, a többi bővítmény adataihoz ne férhessen hozzá.

A bővítmények egy további lényeges jellemzője a biztonság szempontjából, hogy megjelenítésük *i.frame*-ben történik. Az *iframe* (*inline frame*) egy olyan *HTML* elem, amely lehetővé teszi, hogy egy weboldalon belül egy másik, önálló *HTML* dokumentumot ágyazzunk be. A beágyazott tartalom a böngésző által elkülönített környezetben fut, saját *DOM*-fával és *JavaScript* futtatási kontextussal rendelkezik, így természetes izolációs határt képez a befogadó oldal és a beágyazott tartalom között.

Az *iframe* elszigeteltége azonban számos kockázat által fenyegetett. A legjelentősebb fenyegetések közé tartozik a *Cross-Site Scripting* (*XSS*) támadás, amelynek során a bővítményben futó kompromittált *JavaScript* kód, a böngésző által naplózott felhasználó hozzáférési jogosultságait kihasználva, jogosulatlan műveletek elvégzésére vagy érzékeny információk ellopására használható. A támadó számára különösen vonzó célpont a hitelesítési token, amely egy sikeres *XSS* támadás esetén ellopható lenne.

Ezen kockázatok enyhítésére a rendszer több lépcsős védekezési stratégiát alkalmaz. Elsőként, az *iframe sandbox* attribútumát a lehető legkorlátozottabb engedélyekkel konfigurálja: az *allow-scripts* attribútum a *JavaScript* kód futtatásának engedélyezésére szolgál, amely a bővítmény működéséhez szükséges. Az *allow-forms* a *HTML* űrlapok beküldésének engedélyezését biztosítja a felhasználói interakciókhoz, az *allow-modals* pedig a párbeszédablakok megnyitásának engedélyezéséért felel. Az előugró ablakok megnyitásának engedélyezése pedig az *allow-popups* attribútumnak köszönhető.

Kritikus biztonsági intézkedésként az *allow-same-origin* attribútum szándékosan kihagyásra kerül, mivel ez lehetővé tenné az *iframe*-ben futó kód számára, hogy ugyanolyan forrásszabályzat alá essen, mint a befogadó dokumentum, és ezáltal hozzáférhessen a befogadó oldal *DOM*-jához, adattárolóihoz és más érzékeny erőforrásaihoz. Az attribútum kihagyásával az *iframe* elszigetelt marad, és nem férhet hozzá a befogadó alkalmazáshoz még *XSS* támadás esetén sem.

A bővítmény inicializálásakor a rendszer egy *JWT* token állít elő, amelybe beépíti az adott bővítmény egyedi azonosítóját. Ezt a token-t a bővítmény minden, a rendszer felé intézett kérésében *Bearer* tokenként küldi el az *Authorization* fejlécmezőben. A szerver az azonosítót a tokenből olvassa ki, és ennek alapján korlátozza az adathozzáférést, biztosítva, hogy a bővítmény kizárólag a saját erőforrásait érje el. Mivel a *JWT* token aláírásához szükséges kulcs kizárólag a szerver számára ismert, a bővítmény az azonosítóját nem képes megváltoztatni, így más bővítmény adataihoz nem férhet hozzá.

A bővítmények izolációjának további szintjét a kimenő hálózati kérések céljának korlátozása jelenti, amelyet a rendszer *Content Security Policy (CSP)* mechanizmus segítségével valósít meg. Mivel a bővítmények forráskódját egy *MinIO* objektumtároló szolgálja ki, a *CSP* fejléc érvényesítése egy, a *MinIO* elé helyezett *Nginx reverse proxy* segítségével történik. Az *Nginx* az objektumtárolótól érkező válaszokat továbbítja a böngészőnek, és minden kiszolgált fájl *HTTP* válaszához hozzáadja a *CSP* fejlécet. Ez a fejléc a *connect-src* direktíva segítségével meghatározza, hogy az *iframe*-ben futó bővítmény kizárólag a központi szerver felé intézhet hálózati kéréseket és építhet fel *WebSocket* kapcsolatokat. A *default-src*, *script-src*, *style-src* és egyéb direktívák pedig biztosítják, hogy a bővítmény további erőforrásait – *JavaScript* és *CSS* fájlokat, képeket, betűtípusokat – szintén csak az *Nginx*-en keresztül tölthesse le. Amennyiben egy bővítmény ettől eltérő célra próbálna kérést küldeni, a böngésző azt automatikusan blokkolja, így biztosítva, hogy a bővítmények ne kommunikálhassanak jogosulatlan külső szerverekkel.

## 6. Az alkalmazás infrastruktúrája

### 6.1. Build folyamat

Az alkalmazás életciklusának kritikus szakasza a forráskód futtatható állományokká történő alakítása, amelyeket automatizált *build*-eszközök végeznek. Ez a folyamat végzi a kód statikus ellenőrzését, illetve garantálja, hogy az általa előállított végtermék tartalmazza az összes szükséges függőséget.

A *backend* és a *frontend* *build*-folyamatai technológiailag elkülönülve, viszont egymást kiegészítve történnek: a szerveroldali modul *build* folyamatát a *Gradle* automatizált *build*, függőség- és projektmenedzsment eszköz végzi, a *build.gradle* konfigurációs állományban definiált szabályrendszer alapján. A folyamat során a forráskódot a rendszer bájt kódra fordítja, majd lefuttatja a teszteket. A *bootJar* task segítségével egyetlen, önállóan futtatható (úgynevezett *fat JAR*) állomány jön létre, amely tartalmazza a beágyazott webszervert és a projekt összes függőségét.

A *frontend* oldalon a *package.json* fájlban definiált *build* script fut le, amely egy többlépcsős folyamat: először a *TypeScript* fordító elvégzi a szigorú típusellenőrzést, majd a *Vite* eszközlánc végrehajtja a *bundling* folyamatot. Ennek eredményeként a *React* komponensek és a *CSS* modulok böngésző-kompatibilis statikus állományokká alakulnak, javítva az alkalmazás betöltési sebességét.

### 6.2. Konténerizáció

A hordozhatóság és a környezeti függetlenség biztosítása érdekében az alkalmazás a *Docker* technológiát használja, amely garantálja, hogy a szoftver minden környezetben – fejlesztői, tesztelési és éles környezet – azonos módon viselkedik. A *Docker* ezt a problémát azzal oldja meg, hogy az alkalmazás futtatásához szükséges teljes környezetet – a függőségektől a konfigurációig – egyetlen, önálló és reprodukálható egységbe zárja.

A szerveroldali *Dockerfile* felépítése több elkülönített szakaszra tagolódik. Ennek lényege, hogy a fordítási és a futtatási környezet elkülönül egymástól: az első, előkészítő szakaszban a teljes *Gradle* környezet rendelkezésre áll a fordításhoz, azonban a végleges lemezképbe csak a második szakaszban definiált kisméretű *Eclipse Temurin Java* futtatókörnyezet és az elkészült *JAR* fájl kerül bele. Ez a döntés két szempontból is indokolt: egyrészt a fordítóeszközök – fordítók, csomagkezelők, fejlesztői segédprogramok – a futtatáshoz nem szükségesek, jelenlétük az éles

lemezképben felesleges támadási felületet jelent biztonsági szempontból. Másrészt a kisebb lemezképméret gyorsabb kitelepítést eredményez, mivel a lemezképeket minden kitelepítéskor le kell tölteni a célkörnyezetbe.

A folyamat eredményeként létrejövő, izolált egységek szabványosított interfészeken keresztül valósítják meg a rendszerelemek közötti kommunikációt, és hordozhatóságuk révén biztos alapot képeznek a flexibilis kitelepítéshez.

## 6.3. Kitelepítés

Az infrastruktúra kialakítása során a technológiai választásokat két fő szempont vezérelte: a fejlesztési ciklusok lerövidítése automatizált lokális környezetek segítségével, valamint egy biztonságos, skálázható és reprodukálható üzemeltetési modell kialakítása.

### 6.3.1. A fejlesztői környezet orkesztrációja

A lokális futtatásért a *docker-compose.yml* állomány felel, amely egyetlen parancs segítségével elindítja a szerveralkalmazás, a *MySQL* adatbázis, a *MinIO* tároló és az *imgproxy* képkiszolgáló konténereit. A *Docker Compose* melletti döntést az indokolja, hogy a lokális fejlesztési környezetben a *Kubernetes* járulékos komplexitása – a klaszterkezelés, a hálózati konfiguráció és a verzióváltási folyamatok bonyolultsága – indokolatlanul lassítaná a fejlesztési ciklust, ahol a gyors iteráció és az azonnali visszajelzés a legfontosabb. A *Docker Compose* ezzel szemben egyszerű, könnyen módosítható konfigurációt kínál, amellyel a teljes rendszer percek alatt elindítható.

A konfiguráció lehetővé teszi a szolgáltatások közötti indítási függőségek definiálását, ezáltal biztosítva, hogy a szerver csak az adatbázis és az objektumtároló sikeres elindulása után kezdjen el kéréseket fogadni – megelőzve ezzel az inicializálási sorrendből adódó hibákat. Az adatok perzisztenciájának érdekében a rendszer *Docker volumes* technológiát alkalmaz. Ennek hiányában a konténerek újraindításakor az adatbázis tartalma és a feltöltött médiatartalmak elvesznének, mivel a konténer fájlrendszere alapértelmezetten nem perzisztens. A *volumes* mechanizmus az adatok életciklusát leválasztja a konténerek futási állapotától, így az adatok a konténerek újraindítása esetén is megmaradnak. A szolgáltatások közötti kommunikáció izolált, virtuális hídhálózaton (*bridge network*) keresztül történik, amely biztosítja, hogy a konténerek egymást szolgáltatásnéven érik el, miközben a hálózati forgalom el van szigetelve a gazdagép többi folyamatától.

### 6.3.2. Klaszter alapú üzemeltetés

A produkciós környezetben történő futtatás során az alkalmazás egy *Kubernetes* klaszter erőforrásait veszi igénybe. Míg a *Docker Compose* egyetlen gépen futtatja a konténereket, a *Kubernetes* több csomópont között osztja el a terhelést, és automatikusan kezeli a meghibásodásokat – a leállt konténereket újraindítja, egészségtelen példányokat kicserél, és szükség esetén újabb replikákat hoz létre. Ez a szintű megbízhatóság és skálázhatóság kézzel, manuális beavatkozással nem tartható fenn.

Ebben a környezetben az üzemeltetési konfiguráció szigorúan elválik az alkalmazás kódjától. A szoftverkomponensek *Deployment* objektumokként futnak, amelyek meghatározzák a szükséges replikaszámot és kezelik a konténerek életciklusát. A hálózati elérhetőséget és a belső terheléelosztást a szerviz réteg biztosítja, amely stabil végpontokat kínál a kommunikációhoz a klaszteren belül.

A bizalmas adatok – például az adatbázis jelszava vagy az objektumtároló hozzáférési kulcsai – kezelésére a *Kubernetes Secrets* mechanizmusa szolgál. Ennek indoka biztonsági természetű: ha ezek az értékek közvetlenül a konfigurációs állományokban szerepelnének, könnyen a verziókövetőbe kerülhetnének, ahol még privát tároló esetén is jogosulatlan személyek számára válhatnának hozzáférhetővé. A *Secrets* ezzel szemben az érzékeny adatokat a konfigurációtól elkülönítve, titkosítva tárolja, és azok csak a konténerek indításakor, környezeti változókként injektálódnak a futó környezetbe – minimalizálva ezzel az adatszivárgás kockázatát.

## 7. Következtetések és továbbfejlesztési lehetőségek

Mint minden diákszervezet, a *Kocka Másik Oldala* közössége is a folyamatos megújulás és fejlődés útját járja. Évről évre elengedhetetlen lépést tartani a modern technológiai követelményekkel és a hallgatói igények változásával. Éppen ezért a *KMO* projekt egyik alapvető célkitűzése a szervezet digitális infrastruktúrájának hosszú távú fenntarthatóságát és bővíthetőségét biztosító fejlesztési lépések megtervezése és előkészítése.

A projekt eredményeit összegezve számos fontos megállapítás tehető. A diákszervezet számára egy egységes, jól strukturált webes felület került kialakításra, amely centralizáltan kezeli az eseményeket, a médiatartalmakat és a szervezeti információkat. A képek és egyéb erőforrások optimalizált formában váltak elérhetővé, a felelősségek és jogosultságok pedig teljes mértékben átültetésre kerültek a rendszerbe, átlátható és karbantartható struktúrát teremtve. Sikeresen megvalósítható továbbá a külső bővítmények integrációja a már működő rendszerbe, ami a diákközösség életét tekintve létfontosságú. A projekt vezérlő célkitűzése mindazonáltal az, hogy az alkalmazás a diákszervezet mindennapjainak szerves részévé váljon, és folyamatos, aktív használata biztosított legyen – bevonva a hallgatókat szervezőként, fejlesztőként vagy akár csak közösségi tagként.

A továbbfejlesztési lehetőségek tekintetében számos irány körvonalazódik, amelyek a rendszer modernizálását és funkcionalitásának bővítését célozzák. Tervben van a bővítményplatform kiterjesztése, különös tekintettel a *Unity* és más korszerű szoftverfejlesztési környezetek irányába, ezzel szélesítve az integrálható eszközök körét. Szintén értékes fejlesztési irányt jelent az események automatizált közzététele a diákszervezet közösségi média felületein, amely jelentősen megkönnyítené a kommunikációs és hirdetési feladatokat. Végül, hosszabb távon mérlegelhető egy blogszerű tartalomkezelő modul bevezetése is, amely lehetőséget biztosítana cikkek, beszámolók és egyéb szöveges tartalmak rendszeres publikálására.

Összességében elmondható, hogy a *KMO* projekt nemcsak egy funkcionális webalkalmazás létrehozásáról szól, hanem egy olyan digitális ökoszisztéma alapjainak lerakásáról, amely képes hosszú távon kiszolgálni egy aktív és fejlődő hallgatói közösség igényeit. A munka eredményei szilárd alapot nyújtanak a folytatáshoz, és reményt adnak arra, hogy a *Kocka Másik Oldala* diákszervezet a jövőben is megújult eszközökkel, de változatlan lelkesedéssel folytathassa tevékenységét.

## Hivatkozások

- [1] *Ant Design - The world's second most popular React UI framework*. 2015. URL: <https://ant.design/> (elérés dátuma: 2026. ápr. 16.)
- [2] Hrachya Astsatryan és mások. *Performance-Driven and Cost-Efficient Convergence of Cloud and HPC: Evaluating MinIO and LustreFS*. 2025. URL: <https://doi.org/10.22364/bjmc.2025.13.1.10> (elérés dátuma: 2026. ápr. 16.)
- [3] Auth0. *jwt-decode: Decode JWT tokens for browser applications*. 2014. URL: <https://github.com/auth0/jwt-decode> (elérés dátuma: 2026. ápr. 16.)
- [4] Alex Banks és Eve Porcello. *Learning React: Modern Patterns for Developing React Apps*. 2nd. O'Reilly Media, 2020.
- [5] Len Bass, Paul Clements és Rick Kazman. *Software Architecture in Practice*. 4th. Addison-Wesley Professional, 2021.
- [6] Christian Bauer, Gavin King és Gary Gregory. *Java Persistence with Hibernate*. 2nd. Manning Publications, 2015.
- [7] Brendan Burns és mások. „Borg, Omega, and Kubernetes”. *Communications of the ACM* 59.5 (2016), 50–57. oldal.
- [8] Axios Contributors. *Axios: Promise based HTTP client for the browser and node.js*. 2014. URL: <https://axios-http.com/> (elérés dátuma: 2026. ápr. 16.)
- [9] SurveyJS Contributors. *SurveyJS: Open-source JavaScript form builder library*. 2015. URL: <https://surveyjs.io/> (elérés dátuma: 2026. ápr. 16.)
- [10] Paul DuBois. *MySQL Cookbook: Solutions for Database Developers and Administrators*. 3rd. O'Reilly Media, 2014.
- [11] Ilya Grigorik. *High Performance Browser Networking*. O'Reilly Media, 2013.
- [12] Rahmat Hidayat és Dina Dellyana. „Proposing Digital Transformation As Business Strategy for Non-Profit Organisations: Case Study of FGMI (Forum Geosaintis Muda Indonesia)”. *International Research Journal of Economics and Management Studies (IRJEMS)* 4.7 (2025), 107–117. oldal.
- [13] imgproxy team. *imgproxy documentation*. 2024. URL: <https://docs.imgproxy.net/> (elérés dátuma: 2026. ápr. 16.)
- [14] VoidZero Inc. és Vite Contributors. *Vite - Next Generation Frontend Tooling*. 2019. URL: <https://vitejs.dev/> (elérés dátuma: 2026. ápr. 15.)

- [15] Michael B. Jones, John Bradley és Nat Sakimura. *JSON Web Token (JWT)*. 2015. URL: <https://datatracker.ietf.org/doc/html/rfc7519> (elérés dátuma: 2026. ápr. 15.)
- [16] Jamie Kyle. *unstyled-next: 200 bytes to never think about React state management libraries ever again*. 2019. URL: <https://github.com/jamiebuilds/unstyled-next> (elérés dátuma: 2026. ápr. 16.)
- [17] Dirk Merkel. „Docker: lightweight linux containers for consistent development and deployment”. *Linux Journal* 2014.239 (2014), 2. oldal.
- [18] Meta Platforms, Inc. *React Reference Documentation*. 2024. URL: <https://react.dev/> (elérés dátuma: 2026. ápr. 15.)
- [19] Microsoft. *TypeScript Documentation*. 2024. URL: <https://www.typescriptlang.org/docs/> (elérés dátuma: 2026. ápr. 15.)
- [20] M. Nottingham. *HTTP Cache-Control Extensions for Stale Content*. RFC 5861. 2010. URL: <https://www.rfc-editor.org/rfc/rfc5861>.
- [21] Khoi Phan. *A Comprehensive Study on Single-Page Applications: Pros, Cons, and Practical Guidelines*. 2024. URL: <https://www.theseus.fi/handle/10024/851025> (elérés dátuma: 2026. ápr. 15.)
- [22] Mark Richards és Neal Ford. *Fundamentals of Software Architecture: An Engineering Approach*. O'Reilly Media, 2020.
- [23] Ken Schwaber és Jeff Sutherland. *The Scrum Guide*. 2020. URL: <https://scrumguides.org/scrum-guide.html> (elérés dátuma: 2026. ápr. 15.)
- [24] Vercel. *SWR: Modern data fetching, built for React*. 2019. URL: <https://swr.vercel.app/> (elérés dátuma: 2026. ápr. 16.)
- [25] Craig Walls. *Spring in Action*. Manning Publications, 2022.