ETDK-dolgozat

Zoltáni Szabolcs

XXVIII. reál- és humántudományi Erdélyi Tudományos Diákköri Konferencia (ETDK)

Informatika II.: innovatív számítástechnikai termékek, alkalmazások szekció

Kolozsvár, 2025. május 15–18.

Graveyard Navigator

Temetőkertek bejárását és síremlékek felkeresését segítő alkalmazás

Szerzők:

Zoltáni Szabolcs

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

Témavezetők:

dr. Simon Károly, egyetemi adjunktus,

Babeş–Bolyai Tudományegyetem, Matematika és Informatika Kar

CODESPRING 25 YEARS

Graveyard Navigator: Temetőkertek bejárását és síremlékek felkeresését segítő alkalmazás

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

dr. Simon Károly, egyetemi adjunktus,
Babeş–Bolyai Tudományegyetem, Matematika és Informatika Kar

A Graveyard Navigator projekt célja egy olyan alkalmazás létrehozása, amely segít megismerni a temetőkben nyugvó jeles személyeket és a művészeti szempontból kiemelkedő síremlékeket. A projekt kiindulási pontját egy a székelyudvarhelyi református temetőkerttel kapcsolatos kutatás szolgáltatta, de a rendszer általános, más temetőkertek esetében is alkalmazható. Az alkalmazást információval a gondnokok vagy más kijelölt személyek tölthetik fel, egy erre a célra kifejlesztett weboldalon. A felhasználók egy mobil alkalmazás vagy egy weboldal használatával tudhatnak meg információkat a temetőről, síremlékekről, elhunyt személyekről. Az alkalmazás navigációs szolgáltatással is segíti a sírhelyek felkeresését, útvonalak megtervezésére ad lehetőséget, és kiterjesztett valóságon, illetve 3D technológiákon alapuló funkcionalitásokat is biztosít.

Tartalomjegyzék

Bevezető

A Graveyard Navigator projekt célja egy olyan weboldal és mobilalkalmazás létrehozása, amely segítséget nyújt navigálni a temetőkertekben a virtuális tér használatával. Az alkalmazás lehetőséget ad a neves személyek sírhelyeinek felkutatására javasolt útvonal megjelenítésére, illetve további információk böngészésére, fényképek és 3D modellek megtekintésére.

A weboldal a keresésen túl lehetőséget ad a megfelelő jogosultsággal ellátott felhasználók számára az adminisztrációs tevékenységek elvégzésére. A hasonló Find A Grave [1] temetők felfedezésére használatos weboldallal ellentétben a Graveyard Navigator csak erre a célra megbízott személyek által lenne karbantartva információ ellátás szempontjából. Ilyen személyek lehetnek például a temetők gondnokai, egyházi személyek, történészek.

A megjelenítésért felelős statikus elemek támogatják a többnyelvűséget: angol, magyar és román nyelven lehet megtekinteni a weboldalt és a mobil alkalmazást.

A dolgozat célja leírni ennek a projektnek a különböző részeit, több szempontból is, mint például architektúra, technológiai infrastruktúra, alkalmazás üzemeltetése. Az 1. fejezet a funkcionalitásokat, illetve a rendszer architektúráját írja le. A 2. fejezet a szerver funkcionalitásait, felépítését, kódminőséget és konténerizációját taglalja. A 3. fejezetben a weboldal bemutatása kerül sorra. A 4. fejezetben a mobilalkalmazás funkcionalitásai szerepelnek. Az 5. fejezetben a rendszer kitelepítéséhez kapcsolódó információk szerepelnek. A 6. fejezetben a rendszer működése van részletezve. A dolgozat a következtetésekkel és továbbfejlesztési lehetőségekkel zárul.

A projekt ötletét Fecső Zoltán vetette fel a református temetőkerttel foglalkozó kutatására alapozva. A BBTE posztgraduális képzésén résztvevő, a Codespring székelyudvarhelyi kirendeltségénél szakmai gyakorlatozó hallgatók fektették le az alapokat 2024 tavaszán. A backendet[2] Józsa Lóránt Csaba fejlesztette, részt vett Nagy Mónika és Lukáts István is a fejlesztésben, Szász Erika, Hegedüs Hunor és Szakács Tas mentorálása mellett. A 2024-es nyári szakmai gyakorlat során további funkciókkal bővítettük a rendszert Marosi Alpár-Gellért társammal, Szász Erika és Szakács Tas mentorálásával. Az egyetemi év kezdetével továbbfejlesztődött a projekt az egyetemi csoportos projekt tantárgy során további hat évfolyamtársam közreműködésével: Kolozsi Norbert, Nagy Béla, Nagy Gyopár, Pongrácz Kristóf, Portik Márk-Krisztián és Török Zoltán vettek részt ebben a munkában. A Digitális

[1] https://www.findagrave.com
[2] https://jit.jtools.ro/server-side-services-gravewalk-application

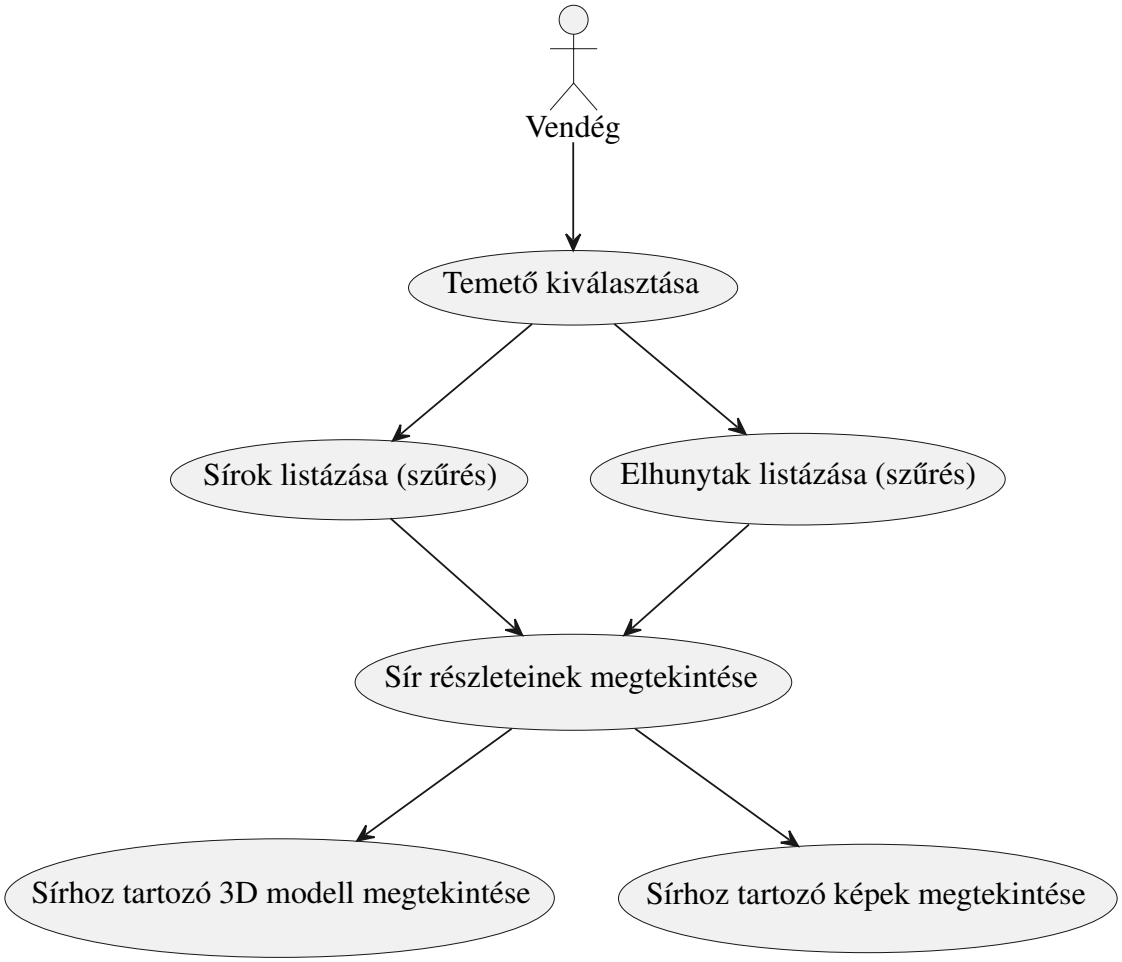
Kaláka negyvennyolc órás székelyudvarhelyi hackhatonon is indultunk a projekttel, Oláh Mátyás és Moroșanu Norbert társaságában, 2025 tavaszán, ahol 3D modellek támogatásával bővült az alkalmazás. A teljes időszakban Szász Erika, Szakács Tas mentorálása és dr. Simon Károly szakmai koordinálása segített a fejlesztésben. Ezúton szeretnék köszönetet mondani nekik, hálás vagyok a rengeteg segítségért és a bíztató szavakért.

1. A Graveyard Navigator projekt

1.1. Funkcionalitások

Az szoftver két szerepkörre osztja a felhasználóit: admin és vendég felhasználók. A vendég csak olvasói jogosultságokat kap, ellentétben az adminnal, aki a temetők adatainak menedzseléséért felelős.

Ahogy a 1. ábrán is látható a vendég információkat tudhat meg az általa kiválasztott síremlékekről, illetve jeles személyekről. A sírhelyekről és elhunytakról fényképeket és kapcsolódó 3D modelleket tekinthet meg.



1. ábra. A vendég felhasználó által elérhető funkciók

Az adminisztrációs folyamatokat végrehajtó admin felhasználó ezeken a funkcionalitásokon túl még szerkeszteni tudja az adatokat és új adatokat tud hozzáadni az adatbázishoz.

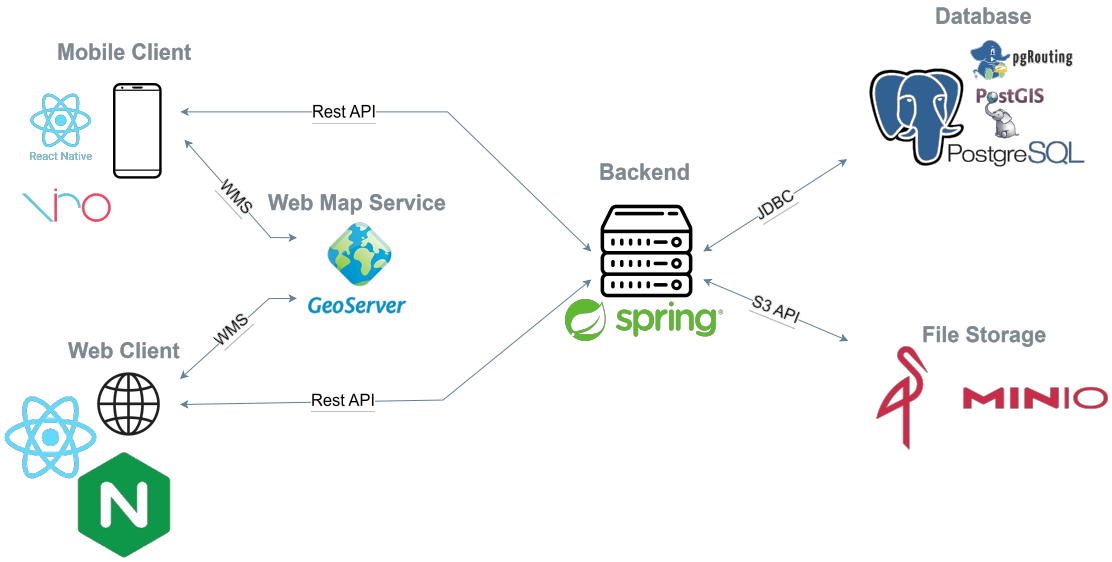
1.2. Architektúra

A rendszert a 2. ábra alapján lehet elemeire bontani, az ezekhez tartozó technológiai infrastruktúrával.

A model réteg az adatokkal és a logikai reprezentációval foglalkozik. A Java Spring Backend[38] több funkciót is ellát, az üzleti logikának megfelelően feldolgozza az adatbázis-kezelő rétegből származó entitások adatait. A PostgreSQL[15] adatbázis segít perzisztálni az adatokat. A PostGIS[31] bővítménnyel földrajzi objektumokat lehet tárolni, amit a pgRouting[28] bővítménnyel további feldolgozásnak vethetünk alá. A MinIO[17] médiafájlok tárolására alkalmas (képek, 3D modellek). A GeoServer[9] WMS (Web Map Service)[24] funkciót tölt be a rendszerben, amely a feltöltött ortofotó csempéit szolgáltatja.

A kontroller réteg összekapcsolja a nézet és a model réteget. Esetünkben a Java Spring Rest Controllerek fogadják a kéréseket a web vagy mobil kliens felől, és szolgáltatják az adatokat a RESTful API[13] konvencióknak megfelelően.

A nézet réteghez sorolható a web kliens és a mobil kliens. A web kliens React[21] JavaScript keretrendszert, a mobil kliens React Native[22] keretrendszert használ.



2. ábra. Az alkalmazás részei a hozzá tartozó technológiai infrastruktúrával

2. Szerver

A szerver RESTful API[13] konvenciók szerint valósítja meg az adatáramlást a megjelenítési és az adattárolási réteg között, a megfelelő autentikációs és autorizációs mechanizmus alapján.

2.1. Funkcionalitások

A következőkben a szerver megvalósítása lesz részletezve, koncepció, működés és használat szempontjából.

2.1.1. Rest API

A REST egy architekturális konvenció halmaz. A legelterjedtebb megvalósítás a **HTTP** protokollon és **JSON** (Javasript Object Notation) formátumon alapuló megközelítés. A **DTO** (Data Transfer Object)[14] mintának megfelelően, DTO objektumok lesznek közvetítve a kommunikáló részek között.

A kliens-szerver architektúrában állapot nélküli **HTTP** kéréseken alapuló kommunikáció által közvetítődnek az erőforrások. Minden HTTP kérés különálló és független, ezért meg kell határozza, hogy milyen erőforrást kér (GET), illetve milyen erőforrással szeretne műveletet végezni (POST, PUT, DELETE). A HTTP állapotnélkülisége azt is jelenti, hogy a kliens adatait nem tároljuk el a kérések között, hanem **JWT** (JSON Web Token)[19] segítségével igazoljuk személyazonosságát.

Robusztus szervernél elengedhetetlen egy konzisztens **REST** erőforrás elnevezési eljárásmód [25]. Minden erőforráshoz tartozik egy **URI** (Uniform Resource Identifier). A rendszer esetében a legfőbb endpointok a */api/graves/* és a */api/departed*, amelyek a sírhelyek, illetve az elhunytak adatait engedik megtekinteni, létrehozni, módosítani, törölni. A megfelelő entitást az azonosítója megadásával lehet meghatározni */api/graves/*{*uuid*}. A további entitásoknak is REST endpointjai vannak: *artifacts*, *gravetypes*, *users*, *centuries*, *images*, *gravemakers*, *places*, *cemeteries*, *occupations*.

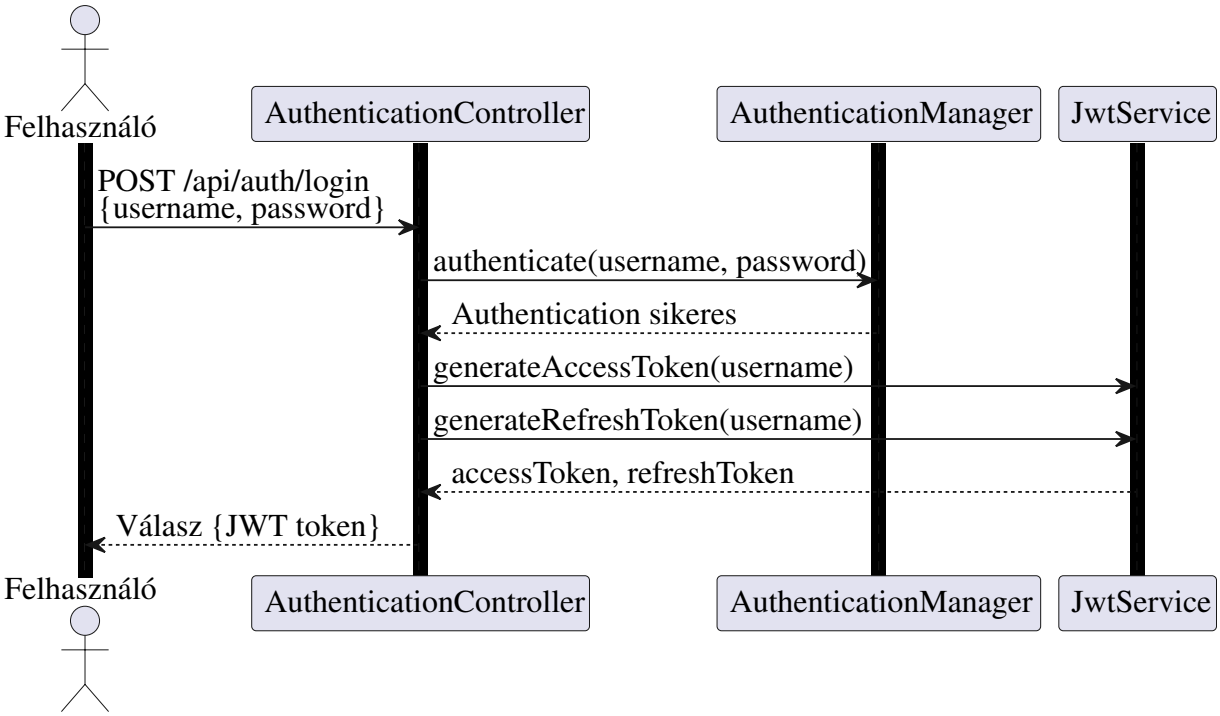
A szerver több temető adatainak kezelését teszi lehetővé a **Spring Data JPA Multitenancy** [37] segítségével. Több tenant esetünkben azt jelenti, hogy a szerver egyetlen példánya több temetőt tud kezelni, és a hozzá tartozó összes erőforrást a *tenant-id* egyedi header beállításával tudja csak elérni az adott temető adminisztrátor felhasználója. Minden kérés során

egy *OncePerRequestFilter* [6] filterrel lesz ellenőrizve a megfelelő header és be lesz állítva a *CurrentTenantIdentifierResolver* szervíznek a temető azonosítója.

2.1.2. Biztonság

A szerver csak megfelelő jogosultságokkal rendelkező felhasználóknak ad engedélyt az erőforrások létrehozására, módosítására és törlésére. Ezt a **Spring Security**[34]-vel valósítja meg, ami egy erős és nagyon testre szabható hitelesítési és hozzáférés-szabályozó keretrendszer. A rendszer kitelepítésekor létrejön az adatbázisban egy szuperadmin[7], aki minden jogosultsággal rendelkezik. Ő tud további adminokat létrehozni.

A 3. ábrán látható a bejelentkezés folyamata. Egy **POST** metódussal ellátott **HTTP** kéréssel, aminek a body részében szerepel a felhasználó neve és jelszava tud egy **JWT** (JSON Web Token)[12] kérni a szervertől. Ezzel a tokennel tudja hitelesíteni az elkövetkezendő kéréseiben a tulajdonost. A **JWT**[18] egy token, ami base64 kódolt JSON-t tartalmaz, *header* része tartalmazza az algoritmus nevét, amivel enkriptálva lett, a *Payload* része pedig a felhasználó nevét, *iat* (Issued At) kibocsátási és az *exp* (Expiration Time) lejárati időt.



3. ábra. JWT alapú bejelentkezés és hitelesítés folyamata

Sikeres bejelentkezés esetében a válasz tartalmaz egy *accesToken*t és egy *refreshToken*t. A két token csak élettartamban különbözik, mivel a *refreshToken* több ideig marad érvényes. Az *accesToken* lejárta után még van lehetőségünk új token párost kérni a

`/api/auth/refreshToken` végponton, ha a *refreshToken* nincs még lejárva. Ez az eljárás **Refresh Token Rotation**[2] néven ismert.

Az 1. táblázatban szerepelnek a szerver végpontjai, a kérés metódusa és annak megfelelően az autentikáció szükségessége.

Végpont	**HTTP metódus**	**Autentikáció szükséges?**
`/api/auth/register`	POST	Igen
`/api/auth/login`, `/api/auth/refreshToken`	POST	Nem
`/api/users/**`	GET, POST, PUT, DELETE	Igen
`/api/departed/**`, `/api/graves/**`, `/api/places/**`, `/api/occupations/**`, `/api/centuries/**`, `/api/gravemakers/**`, `/api/gravetypes/**`, `/api/images/**`, `/api/artifacts/**`, `/api/cemeteries/**`	GET	Nem
	POST, PUT, DELETE	Igen
`/api/cemetery/coordinates/**`, `/api/cemetery/edges/**`	POST	Igen
`/api/cemetery/coordinates/**`, `/api/cemetery/routing/**`	GET	Nem
`/swagger-ui/**`, `/v3/api-docs/**`	GET	Nem

1. táblázat. A rendszer REST API végpontjainak összefoglalása és autentikációs követelményei

2.1.3. Adatelérés

A szerver **Spring Data JPA** adatelérési keretrendszer **JpaRepository** interfészével oldja meg az adatbázis lekéréseket. Rengeteg ismétlődő kódot ki lehet szűrni ezáltal az adatelérési rétegben. Másik előnye a lapozás, amivel az adatokat oldalanként lehet lekérni az adatbázisból.

```
public interface CemeteryRepository extends JpaRepository<Cemetery, String> {
}
```

1. kódrészlet. Temető típusú, String azonosítóval rendelkező **JpaRepository** interfész

Az 1. kódrészletben a **JpaRepository** interfész generikus paraméterei közül az első az entitás típusa (**Cemetery**), a második az entitás azonosítójának típusa (**String**).

Ez az interfész a **JpaRepository** kiterjesztésével automatikusan biztosít számos gyakran használt **CRUD** (Create, Read, Update, Delete) műveletet. A Spring Data JPA generálja az alapértelmezett implementációt, így például az alábbi metódusok már használhatók:

- `findAll(Pageable page)` – összes entitás lekérdezése lapozással
- `findById(String id)` – entitás lekérdezése azonosító alapján
- `save(Cemetery entity)` – új entitás mentése vagy meglévő frissítése
- `deleteById(String id)` – entitás törlése azonosító alapján

A `JpaRepository` nyújtotta metódusok mellett, a *Criteria API* segítségével van megvalósítva egyedi keresés a **GraveSearchDao** és a **DepartedSearchDao** DAO (Data Access Object) osztályokban az adatelérési rétegen.

A keresés dinamikusan épül fel: a `findAllBySearch` metódus a sírhelyhez tartozó személyek születési, család vagy keresztneve alapján végez részegyezés keresést, míg a `findAllBySearchAndFilter` metódus egy összetettebb keresést valósít meg a 2. kódrészletben részletezett osztály alapján, ahol a keresési kifejezéseken kívül opcionális szűrők és rendezés is megadható.

A lekérdezések végrehajtása az **EntityManager** segítségével történik. A visszaadott találatok lapozva érkeznek vissza a kliens felé a **Page** generikus osztály segítségével, amit a **Pageable** interfész állít be.

```
public class GravePagedSearchParams {
    private Optional<Integer> page;
    private Optional<String> sortBy;
    private Optional<String> direction;
    private Optional<Integer> pageSize;
    private Optional<String> searchIdentifier;
    private Optional<String> searchDescription;
    private Optional<Long> searchGraveType;
    private Optional<Long> searchCentury;
    private Optional<Long> searchGraveMaker;
}
```

2. kódrészlet. Oldalszámozott keresési paraméterek a sírhelyeknek

2.1.4. Útkeresés

A terméktulajdonos igényt tartott egy olyan funkció implementálására, amivel javasolni lehet egy legrövidebb utat a sírhelyek megközelítéséhez. A temetőkben a műholdfelvételről nehezen lehet kikövetkeztetni, hogy hol található járható ösvény. Példának felhozható a Házsongárdi temető, ahol kerítéssel van elválasztva egy bizonyos rész. Ha meg szeretnénk látogatni ott egy olyan sírhelyet, ami a kerítésen belül helyezkedik el, akkor vissza kell sétálni a temető főbejáratához és ott lehet bejutni az említett síremlékhez. Rengeteg időt lehet megspórolni, ha az ilyen helyzetekről előre tájékoztat az applikáció.

A javasolt útvonalak kizárhatnák azokat a részeket a temetőben, ahol nem ajánlott sétálni, mert például az lezárt terület, veszélyes az átjárás, nehéz a terep.

A szerver képes menedzselni egy olyan koordinátákból álló gráfot, amivel legrövidebb utat tud ajánlani két pont között *A* algoritmussal*[26]. Ezt a gráfot két entitással tárolja el: *routing_coordinate* és *edge*. A *routing_coordinate* csak egy *org.locationtech.jts.geom.Point* pont adatmezővel rendelkezik a *org.locationtech.jts:jts-core:1.19.0* csomagból, az *edge* pedig két *routing_coordinate* azonosítóját tárolja el, ami a gráfban egy élet alkot.

```
@Query(value = """
        SELECT id
                FROM cemetery_coordinate
                ORDER BY ST_Distance(point, ST_MakePoint(:x, :y), false)
                LIMIT 1
    """, nativeQuery = true)
Long getClosestCoordinateId(@Param("x") double x, @Param("y") double y);
```

3. kódrészlet. Legközelebbi koordináta azonosítójának lekérése

A legrövidebb utat úgy találja meg, hogy megkeresi a kezdeti és a végső koordinátához legközelebbi pontot a gráfban szereplő lekéréssel és a kettő között *A* algoritmussal*[26] egy metaheurisztika alapján Euklideszi távolságot számol.

```
@Query(value = """
        WITH     astar_result AS (
                SELECT *
                FROM pgr_aStar(
                        'SELECT e.id, e.source, e.target, e.cost,
                                st_x(c.point) as x1, st_y(c.point) as y1,
                                st_x(c2.point) as x2, st_y(c2.point) as y2
             FROM edge as e
             JOIN cemetery_coordinate as c on e.source = c.id
             JOIN cemetery_coordinate as c2 on e.target = c2.id',
             :fromId, :toId, directed => false, heuristic => 4) AS a
            )
        SELECT r.seq, r.node, r.edge, r.cost, r.agg_cost,
                        e.source, e.target,
            st_x(c1.point) AS x1, st_y(c1.point) AS y1,
            st_x(c2.point) AS x2, st_y(c2.point) AS y2
        FROM astar_result r
        JOIN edge e ON r.edge = e.id
        JOIN cemetery_coordinate c1 ON e.source = c1.id
        JOIN cemetery_coordinate c2 ON e.target = c2.id
        """, nativeQuery = true)
List<EdgeCoordinatesProjection> getRoute(
                        @Param("fromId") Long fromId,
                        @Param("toId") Long toId);
```

4. kódrészlet. Útkeresés

A 3. kódrészletben szereplő lekérdezés meghatározza azt a koordinátát, amely a legközelebb esik a megadott *(x, y)* ponthoz. Az **SQL** lekérdezés az `ST_Distance`[29] függvényt használja a térbeli távolság kiszámolására, majd a találatokat növekvő sorrendbe rendezi és az első legközelebbi koordináta azonosítóját adja vissza. Ez a lekérdezés natív SQL-ként kerül végrehajtásra a Spring Data JPA `@Query`[5] annotációja segítségével.

A 4. kódrészlet a `pgRouting` bővítmény `pgr_aStar` eljárást használja a két pont közötti legrövidebb útvonal meghatározására az A* algoritmus[26] segítségével. Az algoritmus az `edge` táblából összeállított irányítatlan gráfon keres, ahol az élekhez tartozó kezdőpontok és végpontok földrajzi koordinátáiból számolja a heurisztikus távolságot az Euklideszi metrika (`heuristic = 4`)[26] alapján. Az útvonal eredményeit tartalmazó ideiglenesen tárolt (`WITH`) lekérdezést (`astar_result`) tárolja, majd a lekérdezés eredményét csatlakoztatja az eredeti

koordinátákkal, hogy az útvonalat könnyebben lehessen megjeleníteni.

Ezzel a két lekérdezéssel már egy kezdetleges útkereső készen is van, viszont akadnak gondok, amelyeket még meg kell oldani. Le kell kezelni azt az esetet, amikor nincs egyetlen koordináta sem az adatbázisban. Ebben az esetben a kezdeti és a végső földrajzi koordinátákat adja vissza válaszként a kontroller réteg. Az adatbázisban a gráf egy temető esetében ajánlott, hogy egy komponensből álljon, viszont nem tesz kötelező megszorítást erre vonatkozóan a szerver.

Az 5. kódrészlet az adatbázisban tárolt gráf összefüggő komponenseit határozza meg a `pgRouting` kiterjesztés *pgr_connectedComponents*[27] függvényének segítségével.

Átmenetileg tárolt (WITH) *components* köztes lekérdezés lefuttatja a *pgr_connectedComponents*[27] függvényt. Ez a függvény meghatározza, hogy a gráfban melyik csúcs melyik összefüggő komponenshez tartozik. Ehhez az edge tábla gráf éleit leíró mezőit használja fel.

Az így létrejött *components* ideiglenes táblából a főlekérdezés kiválasztja az összes *component* értéket és csoportosítja ezeket a `GROUP BY` segítségével, így a visszatérési érték csak a különböző komponensek azonosítóit tartalmazza.

```
@Query(value = """
        WITH components AS (
                SELECT component
                FROM pgr_connectedComponents(
                        'SELECT id, source, target, cost FROM edge'
                )
            )
                SELECT component as component
        FROM components
        GROUP BY component
        """,
        nativeQuery = true)
List<ConnectedComponentsProjection> getConnectedComponents();
```

5. kódrészlet. Útkeresés

Az 5. kódrészlet segíthet például annak ellenőrzésében, hogy a gráf valóban egy komponensből áll, vagy több szigetszerű részre esik szét, amelyek között nem létezik út. Ez a 4. kódrészletben látott útvonaltervezés esetén elengedhetetlen, hogy biztosítsuk azt, hogy a kiindulópont és a cél ugyanahhoz a komponenshez tartozik.

2.1.5. Képek

A szerver médiafájlok eltárolására is alkalmas, egy **MinIO**[17] magas teljesítményű objektumtároló rendszerrel oldja meg ezt a feladatot.

A **MinIO**[17] osztott objektum tároló szerver **Golang** programozási nyelvben íródott és az objektumokat bucket-ekbe rendezi. A szerver az adatelérési rétegen levő komponensével a csatlakozáskor ellenőrzi a konfigurációs fájlból származó bucket létezését. Ha nincs ilyen nevezetű bucket, akkor a szerver létrehoz egyet és azt használja. Egy *MinioAsyncClient* kliens használatával implementálva van a fénykép fájl feltöltés, törlés és lekérés az adatelérési rétegen. A kontroller rétegen a REST konvenciónak megfelelően lehet kép fájlokat feltölteni, törölni és lekérni. Ennek a kommunikációnak az implementációját a *io.minio:minio:8.5.13* csomag biztosította.

Fénykép fájl feltöltésekor a fájl nevének használata helyett az aktuális idő függvényében lesz elnevezve a fájl. A fénykép elérési útvonala az adatbázisban van eltárolva egy *Image* táblában, amelyben van egy név, egy elérési út és egy külső kulcs a hozzá kapcsolódó sírhelyhez. Egy fénykép csak egy sírhelyhez tartozik, de egy sírnak több fényképe is lehet.

```
{
    "id": 3,
    "name": "Mohás sírkő",
    "path": "1745592584401.jpeg",
    "graveID": "02906362-6147-4c9e-998c-fe57dfd40341"
}
```

6. kódrészlet. JSON reprezentációja egy ImageDTO-nak

A 6. kódrészlet bemutatja, hogy milyen adatokat tart számon az adatbázisban és példát ad az adatok kinézetére. A *path* mezőben a fájl elnevezése látható, ami eredetileg *received_613096890755897.jpeg* volt. Ezután a fénykép fájl lekérése a `/api/images/{id}` vagy a `/api/images/path/{path}` végpontokon lehetséges.

```
minio.bucket=storage
minio.access.url=http://udv-minio:9000
minio.access.key=XUBkSi9VZqQVJq4TByNTxYb6D
minio.access.secret=Xz9Vne5kX39nePAVR7kV27FQwT921t0HnMH5w70r
```

7. kódrészlet. Példa konfiguráció az *application.properties*-ben

A 7. kódrészletben egy példával van szemléltetve a kapcsolat konfigurálása. A *minio.bucket* a használt bucket nevét, a *minio.access.url* a kapcsolat címét, a *minio.access.key* a felhasználó nevét, illetve a *minio.access.secret* a felhasználó jelszavát határozza meg. A Spring keretrendszer automatikusan létrehozza a megfelelő komponenseket, ezekkel az adatokkal konfigurálva.

2.1.6. 3D modellek

A 3D modellek megjelenítésének előkészítését a szerver végzi, lehetőséget ad ezeknek a médiafájloknak az elmentésére, lekérésére és törlésére. A 3D modellhez tartozó médiafájlok tárolásáért szintén a MinIO objektum tároló szerver felel. Egy sírhelyhez tartozhat több ilyen modell is, ezek lehetnek például kapcsolódó műtárgyak modelljei.

A fejlesztés során olyan modellezési megoldást igyekeztünk keresni, amely egy átlag felhasználó (pl. gondnok, egyházi személy) számára megoldható feladat. Az **Epic Games** fejlesztő közösség által karbantartott **RealityScan**[11] nevű applikációval bárki képes egy műtárgy minőségi lemodellezésére. Az alkalmazás a fotogrammetria, a távérzékelés mechanizmusát használja fel a modell létrehozására. A megjelenítésre szánt műtárgyról fényképek készítésével, és az azokon végzett számítások, mérések alapján, képes meghatározni azt a ponthalmazt (vertex) és a pontokhoz kapcsolható színeket (texture), ami elegendő egy objektum megjelenítésére. A csúcsok között élek szerepelnek. Három élből egy háromszög alakul ki, ami meghatároz egy síkot. Az ilyen csúcsok, élek és háromszögek halmazából poligon háló (polygon mesh) áll össze.

Egy objektumot három fájl használatával határozzuk meg. A *GLB*[20] (GL Transmission Format Binary file) egy 2015-ben bemutatott standardizált fájl formátum, amely 3D adatok leírására szolgál. Információt tartalmaz a 3D modellekről, elhelyezkedésükről, fényekről, anyagokról, csomópontok rangsorolásáról és animációkról. További két fájl írja le a 3D modelt. Egy diffúz térkép[4] fájl tartalmazza a színeket, azaz a felületről a fény diffúz reflexiójának

információit. Egy normál térkép [1] fájl pedig további részleteket ad hozzá az objektumhoz, amelyek módosítják a csomópontok normálisait.

Egy 3D modell elmentése során a kontroller réteg biztosítja, hogy a beérkező **HTTP** multimédiás fájlok esetében a név és a sírhely entitás azonosítója nem üres. Az üzleti logika rétegen további validálásokon mennek keresztül a megadott adatok. Az első fájl mindig egy GLB fájl kell legyen, a második a diffúz, a harmadik pedig a normál térkép információit tartalmazza. A rendszer csak akkor menti el az adatelérési rétegen levő *MinioArtifactDao* DAO segítségével a médiafájlokat, illetve az adatbázisba az elérést leíró adatokat, ha ezek megfelelnek a formai követelményeknek.

Egy 3D modell azonosítójának felhasználásával a szervertől le lehet kérni a modellt leíró fájlokat, amit Base64 kódolással küld HTTP válaszban. Ez a kódolás a bináris és speciális karakterek sorozatát ASCII táblázatban szereplő karakterek sorozatára alakítja. A válasz továbbá tartalmazza a műtárgy nevét és a multimédiás fájloknak a nevét. Lehetőség van egy lapozott lista lekérésére is a */api/artifacts* végponton, akár sírhoz kapcsolva is a */api/graves/{graveId}/artifacts* végponton, amelyben szerepel a műtárgyak azonosítója, neve és a kapcsolódó sírhely azonosítója. A műtárgy azonosítójának biztosításával törölhető a műtárgy,

```
@Transactional
    public ArtifactOutDTO getArtifactWithFiles(String uuid) {
        final Artifact artifact = artifactRepository.findById(uuid)
                        .orElseThrow(ArtifactNotFoundException::new);
        final String[] fileNames = new String[3];
        fileNames[0] = artifact.getModelPath();
        fileNames[1] = artifact.getDiffuseTexturePath();
        fileNames[2] = artifact.getNormalTexturePath();
        final String[] base64EncodedFiles = new String[3];
        base64EncodedFiles[0] = minioArtifactDao.getBase64EncodedFile(
                        artifact.getModelPath());
        base64EncodedFiles[1] = minioArtifactDao.getBase64EncodedFile(
                        artifact.getDiffuseTexturePath());
        base64EncodedFiles[2] = minioArtifactDao.getBase64EncodedFile(
                        artifact.getNormalTexturePath());
        return new ArtifactOutDTO(
                        artifact.getName(),
                        fileNames,
                        base64EncodedFiles);
    }
```

8. kódrészlet. A válasz objektum összeállításának folyamata az üzleti logika rétegen

amely során eltávolításra kerül az adatbázisból és a MinIO objektumtároló rendszerből is törölve lesznek a leírására szolgáló fájlok.

A 8. kódrészletben látható, hogy hogyan épül fel a válaszként adott DTO (Data Transfer Object). Az adatbázisból lekéri a műtárgyat leíró adatokat, majd sorra feltölti a DTO objektumot a műtárgy nevével, a fájlok neveivel és a Base64 kódolt médiafájlokkal. Ha nem találja a műtárgyat azonosító szerint, akkor kivételt dob.

2.2. Architektúra

A szerver részei Java csomagokba vannak csoportosítva, a 4. ábrán a csomagstruktúra látható. Az egymással funkcionalitás szempontjából összefüggő osztályok vannak egy-egy csomagba csoportosítva. Az *assemblers* csomag osztályai az adatokat állítják össze és alakítják át DTO-ba a modellekből. A *config* csomag osztályai konfigurációs szerepet töltenek be. A *controller* csomag osztályai a kliensek kéréseinek kezéléséért felelősek a REST API végpontokon. A *dto.incoming* a bejövő, a *dto.outgoing* pedig a kimenő adatok reprezentációjáért felelős osztályokat tartalmazzák. Az *exception.exceptions* az egyedi kivételek osztályait tartalmazzák. A *filter* csomag osztályai a kérések előfeldolgozását végzik, mint például az autentikáció és a tenant azonosító elmentése a TenantIdentifierService üzleti logika rétegen levő osztályba. A *mapper* csomag osztályai az adatbázis entitások (JPA entity)

```
java
├── assemblers
├── config
├── controller
├── dto
│   ├── incoming
│   └── outgoing
├── exception
│   └── exceptions
├── filter
├── mapper
├── model
├── projection
├── repository
│   └── implementation
├── service
└── util
```

4. ábra. Java csomagstruktúra

leírását tartalmazzák. A *projection* csomagban az adatbázisban szereplő adatok lekérése során használt adatok részleges leképezésére szolgáló osztályok találhatóak. A *repository* csomag osztályai leírják az adatbázis lekérdezésekkel foglalkozó osztályok interfészeit, míg a *repository.implementation* az implementációt biztosító osztályokat tartalmazza. A *service* csomagban találhatóak az üzleti logika (business logic) interfészei, a *service.implementation* csomagban ezek megvalósításai. A *util* csomagban található osztályok tartalmazzák a segédmetódusokat, mint például a geometriai pont reprezentációjának átalakítása stb.

Package	**Szerepkör**
`assemblers`	DTO összeállítás
`config`	Konfigurációk
`controller`	REST végpontok kezelése
`dto.incoming`	Bejövő DTO-k
`dto.outgoing`	Kimenő DTO-k
`exception.exceptions`	Egyedi kivételek
`filter`	Kérések szűrése
`mapper`	Entitásleképezés
`model`	Adatbázis entitások
`projection`	Részleges adatleképezés
`repository`	Adatelérés (interfészek)
`repository.implementation`	Adatelérés (implementációk)
`service`	Üzleti logika (interfészek)
`service.implementation`	Üzleti logika (implementációk)
`util`	Segédfüggvények

2. táblázat. A szerver oldali csomagstruktúra áttekintése

2.3. Swagger

A projekt fejlesztése során a REST API végpontok dokumentálására és tesztelésére a **Swagger** technológiát használjuk. A Swagger egy széles körben elterjedt, nyílt forráskódú eszközkészlet, amely lehetővé teszi az API-k automatikus dokumentálását, tesztelését és vizualizációját.

A dokumentáció automatikusan generálódik a kódban elhelyezett annotációk alapján,

így mindig naprakész képet ad az elérhető végpontokról, azok bemeneti paramétereiről, válaszaikról és státuszkódjaikról. A Swagger felületén keresztül lehetőség nyílik az API végpontok közvetlen kipróbálására is.

A projektben a *org.springdoc:springdoc-openapi-starter-webmvc-ui:2.5.0* könyvtár van felhasználva a Swagger integráció megvalósítására. Ez a könyvtár automatikusan feltérképezi a `kontroller` rétegben található végpontokat, és generálja belőlük az OpenAPI 3 szabvány szerinti leírást.

A generált dokumentáció böngészőn keresztül érhető el a `/swagger-ui/index.html` útvonalon keresztül. A Swagger UI segítségével a fejlesztők, tesztelők és külső integrátorok könnyen megismerhetik és kipróbálhatják a szerver által nyújtott funkcionalitásokat.

A Swagger használata a fejlesztési folyamat során nemcsak a dokumentáció készítését könnyíti meg, hanem segíti a rendszer megértését, az API-k helyes használatának ellenőrzését és a hibakeresést is.

2.4. Graves API végpontok

A következőkben a `GraveController` osztályban található végpontok Swagger alapú leírásának kifejtésére kerül sor. Az 5. ábrán látható végpontok segítségével megvalósíthatóak a sírhelyekkel kapcsolatos műveletek, mint például a sír adatainak lekérése, új sír létrehozása, és az információk frissítése vagy törlése. Nagyon hasonló végpontokkal lehet kezelni a többi modellt is: *műtárgy*, *sírtípus*, *él*, *koordináta*, *felhasználó*, *elhunyt évszázad*, *fénykép*, *sírkészítő*, *település*, *temető*, *foglalkozás*.

5. ábra. Swagger UI Graves API végpontok leírása

2.4.1. A sírok listájának lekérése

- **Végpont:** `GET /api/graves`
- **Leírás:** Ez a végpont visszaadja oldalazva az összes sír adatát egy listában. Az eredmény tartalmazza a sírok UUID-ját, azonosítóját, koordinátáját, típusát, évszázadát, sír készítőt, képeket, leírást és az elhunytak listáját.

Státuszkódok:

- `200 OK`: A sírok adatainak sikeres visszaadása.
- `404 Not Found`: Ha nem található sír.
- `400 Bad Request`: Ha valamilyen hiba történik a lekérés során.

2.4.2. Sírok lekérése UUID lista alapján

- **Végpont:** `GET /api/graves/byids`

- **Leírás:** Ez a végpont lehetővé teszi, hogy egy UUID lista alapján lekérdezzük az adott sírokat. A válasz kevesebb attribútumot tartalmaz, mint a teljes sír adatlapja.

Státuszkódok:

- `200 OK`: A sírok adatainak sikeres visszaadása.
- `404 Not Found`: Ha nem található sír a megadott UUID-k alapján.
- `400 Bad Request`: Ha a lekérés nem megfelelő.

2.4.3. A sírok listájának lekérése térképen történő megjelenítéshez

- **Végpont:** `GET /api/graves/maplist`
- **Leírás:** A végpont visszaadja az összes sírt egy minimalizált adatmodellben, amely alkalmas térképen vagy listában történő megjelenítésre.

Státuszkódok:

- `200 OK`: A sírok adatainak sikeres visszaadása.
- `404 Not Found`: Ha nem található sír.
- `400 Bad Request`: Ha valamilyen hiba történik a lekérés során.

2.4.4. Egy sír lekérése UUID alapján

- **Végpont:** `GET /api/graves/uuid`
- **Leírás:** Ez a végpont visszaadja egy adott sír adatait, beleértve az összes attribútumot, így az elhunytak listáját is.

Státuszkódok:

- `200 OK`: A sír adatainak sikeres visszaadása.
- `404 Not Found`: Ha a megadott UUID alapján nem található sír.
- `400 Bad Request`: Ha valamilyen hiba történik a lekérés során.

2.4.5. A sírhoz tartozó 3D műalkotások lekérése

- **Végpont:** `GET /api/graves/uuid/artifacts`
- **Leírás:** Ez a végpont visszaadja az adott sírhoz tartozó 3D műalkotások azonosítóit.

Státuszkódok:

- `200 OK`: A műalkotások sikeres visszaadása.
- `400 Bad Request`: Ha valamilyen hiba történik a lekérés során.

2.4.6. A sírok attribútumainak lekérése dropdown listákhoz

- **Végpont:** `GET /api/graves/attributeLists`
- **Leírás:** A végpont visszaadja a sírok attribútumainak listáját, amelyet fel lehet használni például legördülő listákban történő kiválasztáshoz.

Státuszkódok:

- `200 OK`: A sírok attribútumainak sikeres visszaadása.
- `400 Bad Request`: Ha valamilyen hiba történik a lekérés során.

2.4.7. Új sír létrehozása

- **Végpont:** `POST /api/graves`
- **Leírás:** Ez a végpont új sírt hoz létre az adatbázisban.

Státuszkódok:

- `200 OK`: A sír sikeres létrehozása.
- `422 Unprocessable Entity`: Ha a sír nem hozható létre (pl. érvénytelen adatok).
- `400 Bad Request`: Ha valamilyen hiba történik a létrehozás során.

2.4.8. A sír frissítése

- **Végpont:** `PUT /api/graves`
- **Leírás:** A végpont egy meglévő sír frissítésére szolgál.

Státuszkódok:

- `200 OK`: A sír sikeres frissítése.
- `422 Unprocessable Entity`: Ha a sír nem frissíthető (pl. érvénytelen adatok).
- `400 Bad Request`: Ha valamilyen hiba történik a frissítés során.

2.4.9. A sír törlése

- **Végpont:** `DELETE /api/graves/uuid`
- **Leírás:** A végpont egy meglévő sírt töröl az adatbázisból, feltéve, hogy a sírhoz tartozó elhunytak listája üres.

Státuszkódok:

- `204 No Content`: A sír sikeres törlése.
- `404 Not Found`: Ha a megadott UUID alapján nem található sír.
- `422 Unprocessable Entity`: Ha a sír nem törölhető, mert az elhunytak listája nem üres.
- `400 Bad Request`: Ha valamilyen hiba történik a törlés során.

2.5. Kódminőség

A megfelelő kódminőség fenntartása érdekében a projekt fejlesztése során számos eszköz volt alkalmazva a forráskód statikus elemzésére és az esetleges problémák, hibák időben történő felismerésére. A projekt a következő Gradle plugineket alkalmazza: `checkstyle`, `pmd`, `spotbugs`, és a `violations-gradle-plugin`. Ezek az eszközök segítenek a kód tisztaságának, követhetőségének és megbízhatóságának fenntartásában.

2.5.1. Linterek

A következőkben részletezve lesznek azok az eszközök, amelyek segítenek biztosítani a magas szintű kódminőséget.

A **Checkstyle**[8] egy statikus kódelemző eszköz, amely a kód formázására és stílusára vonatkozó szabályokat alkalmaz. Néhány dolgot szükséges konfigurálni használat előtt. A `checkstyle.xml` konfigurációs fájl leírja a betartandó szabályokat, szempontokat ad a kódellenőrzéshez. A `ignoreFailures` beállítás segítségével a build nem áll le, ha hibák vannak, de a hibák jelentésre kerülnek.

A **SpotBugs**[33] egy statikus kódelemző eszköz, amely a potenciális hibákat és problémákat, például null-pointer dereferenciákat vagy nem használt változókat észleli. Az `excludeFilter` fájlban definiált szűrők alapján végzi a hibák kiszűrését

A **PMD**[30] egy másik statikus kódelemző eszköz, amely a kód hibáit és potenciálisan problémás részeit, például a nem használt változókat, vagy a túl bonyolult kódrészleteket azonosítja.

A `lint.gradle` fájl tartalmazza a fenti eszközök konfigurációját. Ez a konfiguráció biztosítja, hogy minden egyes kódrészlet átvizsgálásra kerüljön a fent említett eszközök által, és bármilyen problémát időben észlelhessünk.

2.5.2. SonarQube

A SonarQube egy fejlett statikus kódminőség-ellenőrző eszköz, amely további elemzésekre ad lehetőséget. A SonarQube nemcsak a kód hibáit és potenciális problémáit találja meg, hanem segítséget nyújt a refaktorálásban és a kód karbantartásában is. Részletesen indokolja a hiba okát és példát is mutat annak javítására.

A SonarQube projekt alapú elemzéseket végez, és integrálható különböző build eszközökkel, mint például esetünkben a Gradle. A következő példában a konfigurálása látható a Gradle build szkriptben.

```
sonarqube {
    properties {
        property "sonar.projectKey", "mentor-graveyard-2024_server_5d0b8ec0"
        property "sonar.host.url", "http://localhost:9000"
        property "sonar.login", "sqp_c6320d32a38a50c12c593982a618b9e704fdc190"
    }
}
```

A fenti konfigurációban három kulcsfontosságú paramétert állítunk be. A **sonar.projectKey** az egyedi azonosító, amely a projektet jelöli a SonarQube rendszerében. Ez alapján történik a projekt azonosítása az analízis során. A **sonar.host.url** az az URL, amely a SonarQube szerverhez vezet. Az itt megadott helyi URL általában az, ahol a SonarQube szolgáltatás fut. A **sonar.login** a SonarQube szolgáltatás eléréséhez szükséges autentikációs token vagy jelszó. Ezt az információt a SonarQube felhasználói felületén lehet generálni.

6. ábra. A SonarQube a szerveren végzett első elemzés eredményei

Az elemzés eredményei a SonarQube felületén jelennek meg, ahol részletes információkat kapunk a kódminőségről, hibákról, kódduplikációkról és más problémákról. Az analízis során a következő főbb információk kerülnek megjelenítésre.

Security (Biztonság): A SonarQube a kód biztonságát is elemzi, és figyelmeztetéseket ad a potenciális biztonsági problémákra, mint például az SQL injekciók, keresztoldali scriptelés (XSS), sérülékeny titkosítási eljárások és más biztonsági résekkel kapcsolatos hibák.

Reliability (Megbízhatóság): A SonarQube figyeli a kód stabilitását és megbízhatóságát is. Ez magában foglalja a kód hatékonyságát és a váratlan hibák, kivételek kezelését. Az eszköz olyan hibákat azonosít, amelyek negatívan befolyásolhatják az alkalmazás működését, és javaslatokat ad a kód megbízhatóságának javítására.

Maintainability (Karbantarthatóság): A karbantarthatóság a kód olvashatóságát, módosíthatóságát és hosszú távú fenntarthatóságát jelenti. A SonarQube segít az olyan problémák felismerésében, amelyek megnehezíthetik a kód későbbi frissítését vagy bővítését. A kód bonyolultsága, a nem megfelelő kódszervezés és a túlzott kódduplikáció mind idesorolhatóak és javaslatokat ad a refaktorálásra.

Hotspots reviewed (Forró pontok): A SonarQube a kódot egy "forró pont" szemlélettel is elemzi. A forró pontok olyan részek a kódban, amelyek különösen hajlamosak a hibákra, vagy amelyek túl gyakran változnak. Az eszköz kiemeli ezeket a részeket, lehetővé téve a fejlesztők számára, hogy nagyobb figyelmet fordítsanak ezekre, és időben észrevegyék az

esetleges problémákat.

Coverage (Lefedettség): A SonarQube az egységtesztelés (unit testing) lefedettségét is mérheti, vagyis azt, hogy a kód hány százaléka van tesztelve. A tesztelési lefedettség megjelenítése segít kiemelni mely részeket kell még tesztelni, és biztosítani, hogy a legkritikusabb funkciók megfelelően tesztelve legyenek.

Duplications (Kódduplikációk): A kód duplikációk azonosítása az egyik legfontosabb feladat a karbantarthatóság javításában. A SonarQube képes észlelni a hasonló vagy teljesen azonos kódrészleteket a projektben. A kódduplikációk csökkentése nemcsak a kód tisztaságát növeli, hanem segít a későbbi refaktorálás során is, mivel kevesebb ismétlődő kódrészletet kell módosítani.

2.6. Konténerizáció

A konténerizáció egy fontos lépés a modern alkalmazások telepítésében és futtatásában. Ebben a részben bemutatásra kerül, hogyan lehet a projektet Docker konténerbe csomagolni, hogy könnyebben telepíthető és skálázható legyen. A következő szkript és Dockerfile bemutatják a konténerizálás folyamatát, amelyet a Gradle build és Docker segítségével lehet elvégezni.

A 9. kódrészletben szereplő `DockerImage.sh` szkript először a Gradle `clean` feladatát futtatja, hogy eltávolítsa a régi build fájlokat. Ezt követően a `build` feladatot futtatja a projektben, tesztek futtatása nélkül (`-x test` kapcsolóval), majd elkészíti a végső build fájlt. Ha bármelyik lépés során hiba történik, a szkript leállítja a folyamatot, és megfelelő hibaüzenetet ad.

A következő lépésben a Docker image építése történik. A Dockerfile-ban meg lehet határozni, hogy az alapimage az `eclipse-temurin` Java futtatókörnyezet lesz, amely a szükséges Java verzióval rendelkezik. A `build/libs/udv-.jar` fájlt bemásoljuk a konténerbe, majd jelzés lesz elhelyezve a használatra javasolt portot (`8080`) és a végrehajtandó parancsot (`java -jar udv-.jar`).

Ez a Docker alapú telepítési módszer lehetővé teszi, hogy az alkalmazást bármilyen környezetben, amely támogatja a Docker-t, könnyen futtathassuk, miközben biztosítjuk a szükséges függőségek és konfigurációk megfelelő kezelését.

A `DockerImage.sh` és a Dockerfile integrálása lehetővé teszi a gyors, konzisztens buildet és deploy-t, minimalizálva a telepítési hibák esélyét, valamint megkönnyítve a fejlesztői és

produkciós környezetek közötti átjárhatóságot. A konténerizálás egy olyan megoldás, amely segít a skálázhatóságban és a rugalmasságban, hiszen a Docker lehetővé teszi az alkalmazás

```
#!/bin/bash

# Store the current directory
SCRIPT_DIR="$(dirname "$0")"

# Run Gradle clean
echo "Running Gradle clean..."
cd "$SCRIPT_DIR" || exit
./gradlew clean
if [ $? -ne 0 ]; then
    echo "Gradle clean failed."
    exit 1
fi
echo "Gradle clean succeeded."

# Run Gradle build
echo "Running Gradle build..."
cd "$SCRIPT_DIR" || exit
./gradlew build -x test
if [ $? -ne 0 ]; then
    echo "Gradle build failed."
    exit 1
fi
echo "Gradle build succeeded."

# Build Docker image
echo "Building Docker image..."
cd "$SCRIPT_DIR" || exit
docker build -t udv-server-image -f Dockerfile .
if [ $? -ne 0 ]; then
    echo "Docker build failed."
    exit 1
fi
echo "Docker build succeeded."
```

9. kódrészlet. Image létrehozásáért felelős "DockerImage.sh" bash script Gradle build és Docker segítségével

```
FROM eclipse-temurin:21.0.3_9-jre-jammy
COPY build/libs/udv-*.jar ./
EXPOSE 8080
CMD java -jar udv-*.jar
```

10. kódrészlet. Image leírása Dockerfile-ban

könnyű másolását és futtatását különböző rendszereken.

A Docker alapú konténerizáció alapot ad a Kubernetes vagy más konténer alapú orkesztrációs szoftver használatához. Ez a folyamat biztosítja, hogy az alkalmazás minden környezetben ugyanúgy működjön, függetlenül attól, hogy a fejlesztéshez használt gépen, teszt környezetben vagy produkciós környezetben fut.

3. Web

A weboldal célja, hogy a felhasználók könnyen hozzáférjenek a temetőkben található sírhelyekkel és neves elhunyt személyekkel kapcsolatos információkhoz.

A weboldalon keresztül a felhasználók megtekinthetik a temető különböző sírhelyeit, ahol részletes információk találhatók, például az elhunytak nevei, életük fontosabb eseményei, valamint róluk készült fényképek. A sírhelyekhez kapcsolódó műtárgyak, például szobrok, emléktáblák és egyéb alkotások is megjelennek a felhasználók számára.

A weboldal navigációs funkciói között kiemelt szerepet kap a tájékozódás segítése. A rendszer képes meghatározni a felhasználó aktuális helyzetét, és megmutatni a legrövidebb utat a keresett sírhelyhez. Ezzel a funkcióval a látogatók könnyebben elérhetik a megtekinteni kívánt sírokat.

A weboldal emellett lehetőséget biztosít adminisztrációs feladatok elvégzésére is. Például a temető gondnokai, tulajdonosai, gondozói a rendszerben létrehozhatják, frissíthetik és törölhetik a sírhelyek adatait, menedzselhetik a temető információit, a hozzájuk tartozó elhunytak listáját, valamint a temetőben található műtárgyak adatainak kezelését is elvégezhetik.

A weboldal célja tehát, hogy egy átfogó és könnyen használható platformot biztosítson a felhasználóknak és adminisztrátoroknak egyaránt. Az összes funkcionalitás úgy lett kialakítva, hogy intuitív módon segítse a felhasználókat a kívánt információk gyors elérésében, miközben biztosítja a temető kezelésének hatékony adminisztrációját.

3.1. Funkcionalitások

Az alábbiakban a weboldal főbb funkcionalitásai lesznek bemutatva.

3.1.1. Bejelentkezés

A felhasználók vendég vagy admin felhasználóval léphetnek be az alkalmazásba. Az admin felhasználó számára megjelennek azok az ikonok, amelyek segítségével képes CRUD (Create, Read, Update, Delete) műveleteket elvégezni az entitásokkal. A vendég felhasználó számára csak megtekintési lehetőségek állnak rendelkezésre, nem módosíthatja az adatokat.

A bejelentkezési adatok és jogosultságok kezelésére a `useAuth` nevű egyedi hook került implementálásra. Ez a hook egy `AuthContext`-ből olvassa ki az aktuális hitelesítési állapotot és

az admin jogosultságokat. Az `AuthContext` egy szabványos React kontextus (`React.Context`), amely globális állapotként elérhetővé teszi a hitelesítési adatokat az alkalmazás komponensei számára.

A `useAuth` hook a következőket biztosítja az aktuális felhasználó bejelentkezési állapotát (admin vagy vendég), hozzáférési *accessTokent*, *refreshTokent*, illetve a CRUD műveletekre jogosultságát.

Az `AuthContext` provider komponens magasabb szintű komponensek köré van helyezve, így biztosítva, hogy az egész alkalmazásban bárhonnan könnyen lehessen hozzáférni a bejelentkezési adatokhoz és a hitelesítési műveletekhez.

3.1.2. Temetőválasztás

A bejelentkezés után a felhasználók számára elérhető egy temetőválasztó képernyő, amelyen a különböző temetők listája jelenik meg. Lehetőség van a temetők közötti választásra akár listás, akár térképes nézetben. A listás megjelenítés egyszerű szöveges nézetet ad, míg a térképes megjelenítés lehetővé teszi a temetők helyének vizuális megtekintését. A térképes nézet a *react-leaflet*[36] könyvtár használatával készült, amely egy népszerű, React-hoz készült wrapper a Leaflet.js[35] könyvtárhoz. Ez a könyvtár lehetővé teszi térképek interaktív megjelenítését és kezelését a React alapú alkalmazásokban.

A térképen a temetők helye **Marker**-ekkel van jelölve. Ha túl sok marker helyezkedne el egymás mellett, akkor a *react-leaflet-cluster*[32] csomag segítségével csoportosíthatóak a jelölők. Ennek az a lényege, hogy ha a felhasználó nagyít a térképen, akkor a marker csoportok felbomlanak és egyesével láthatóvá válnak, így a felhasználó könnyen megtalálhatja a kívánt temetőt. A markerre való kattintáskor egy **Popup** jelenik meg, amely a temető nevét és alapvető információkat tartalmaz. Ez a funkció segít abban, hogy a felhasználók könnyen tájékozódjanak a temetők elhelyezkedéséről és választhassanak a különböző helyszínek közül.

3.1.3. Végtelen görgetés és lapozás

Az alkalmazásban az adatok betöltése végtelen görgetéshez hasonlóan valósul meg. Ez azt jelenti, hogy az entitások listázásakor nem az összes adatot kérjük le egyszerre, hanem csak egy adott oldalt, amely a felhasználó görgetési műveletei alapján dinamikusan bővül. Ez egyrészt csökkenti a hálózati forgalmat, másrészt javítja az oldal teljesítményét és a felhasználói élményt.

Minden entitás lekérdezése lapozással történik: minden API-hívásnál meg kell határozni,

hogy melyik oldalt és oldalméretet szeretnénk lekérni. A lapozás paraméterei a `page` és `pageSize` paramétereken keresztül kerülnek átadásra a backend felé.

A lekérdezések kezelésére egyedi React hookok kerültek kialakításra. Minden fontosabb entitástípushoz tartozik egy dedikált hook, amely a megfelelő API végpontra hív, és kezeli a lapozási állapotot. A következő hookok felelnek az adatok lapozott lekérdezéséért: *useArtifactModelSearch*, *useArtifactSearch*, *useCemeteryMapSearch*, *useCemeterySearch*, *useCenturySearch*, *useDepartedSearch*, *useGraveMakerSearch*, *useGraveSearch*, *useGraveTypeSearch*, *useOccupationSearc*, *usePlaceSearch*.

Ezek a hookok általában több funkcionalitással rendelkeznek: az oldalszámot és adatmennyiséget kezelik, új adatokat töltenek be a görgetési eseményekre, követik a betöltési állapotot (`loading`), hibaállapotot (`error`) és az utolsó oldal ellenőrzése is az ő feladatuk.

A megközelítés lehetővé teszi, hogy a felhasználók zökkenőmentesen böngésszenek nagy adatmennyiségek között anélkül, hogy az alkalmazás teljesítménye romlana vagy a memóriahasználat jelentősen megnőne.

3.1.4. Szűrés

A végtelen görgetéssel kombinált adathozzáférés mellett a rendszer lehetőséget biztosít különböző szempontok szerinti szűrésre is. A szűrési funkciók a megfelelő hook-okban valósulnak meg, így a felhasználó célzottabban tud keresni az entitások között anélkül, hogy minden egyes találatot manuálisan kellene átnéznie.

A szűrés paraméterei a hook-ok bemeneti értékein keresztül kerülnek átadásra, amelyeket a backend a lekérdezések során figyelembe vesz. Ennek megfelelően csak azok az adatok kerülnek betöltésre, amelyek megfelelnek a megadott kritériumoknak.

Például a `useGraveSearch` hook esetében az alábbi szűrőfeltételek állnak rendelkezésre:

- `identifierFilter` – sírhely azonosító szerinti szűrés,
- `graveTypeFilter` – sírtípus szerinti szűrés,
- `centuryFilter` – század alapján történő szűrés,
- `graveMakerFilter` – sírkészítő szerinti szűrés,
- `descriptionFilter` – leírásban való keresés.

Hasonló szűrési lehetőségek elérhetők a többi hook esetében is, az adott entitás típusának megfelelően. A szűrők kombinálhatóak, így a felhasználók komplex keresési feltételeket is megadhatnak.

A szűrés működése során a hook minden szűrési paraméter változását figyeli, és ennek hatására új adatlekérdezést indít az API felé. Így biztosított, hogy a megjelenített lista mindig a legfrissebb és legrelevánsabb találatokat tartalmazza.

3.1.5. Útmutatás a sírhelyhez

A sírhely részleteit bemutató oldalon egy útjelző ikon segítségével a felhasználó könnyen átnavigálhat arra az oldalra, amely a jelenlegi tartózkodási helye és a kiválasztott sírhely közötti legrövidebb utat jeleníti meg.

Az útvonaltervezéshez a rendszer először engedélyt kér a geolokációs adatok elérésére. Amennyiben a felhasználó jóváhagyja a hozzáférést, a kliens egy kérést küld a */api/cemetery/routing* végpontra. A kérésben szereplő adatok:

- `fromLatitude` és `fromLongitude`: a felhasználó aktuális földrajzi koordinátái,
- `toLatitude` és `toLongitude`: a célként megadott sírhely koordinátái.

A szerver a megadott koordináták alapján kiszámolja a javasolt útvonalat, majd a koordináták sorozatát visszaküldi a kliensnek. Az útvonal megjelenítése a térképen kék szaggatott vonallal történik, amely csak javaslatként szolgál a felhasználó számára, kifejezve azt, hogy a tényleges bejárhatóság helyenként eltérhet a kijelölt útvonaltól.

A megjelenítés során a térképes komponens automatikusan fókuszba helyezi a temetőt, megkönnyítve ezzel a tájékozódást a temető területén belül.

3.1.6. Képek

A sírhelyekhez tartozó fényképek kezelését a szerver oldalon tárolt képfájlok és azok elérhetőségei biztosítják. A sírhelyek adatai között szerepelnek a képek URL-jei, amelyeket a weboldal kliensoldalon dinamikusan tölt be.

A képek előnézeti változatban, egy rugalmas elrendezésű *Flexbox* konténerben jelennek meg, amely biztosítja az egységes és esztétikus megjelenítést, különböző képernyőméreteken.

Amennyiben a felhasználó egy képre kattint, a kép teljes képernyős módban nyílik meg a *yet-another-react-lightbox* nevű npm csomag segítségével. A lightbox nézet több kényelmi

funkciót is kínál, mint például nagyítás, letöltés lokális eszközre, több kép közötti egyszerű váltás, reszponzív megjelenés biztosítása mobileszközökön is.

3.1.7. 3D modellek

A sírhelyekhez és műtárgyakhoz kapcsolódó 3D modellek elérhetőségeit a szerver szolgáltatja, hasonlóan a képfájlokhoz. Az adatok tárolásáért és kezeléséért a *useArtifactSearch* hook felel, amely tartalmazza a modellek elérési útvonalait is.

A modellek bináris fájljai Base64 kódolással vannak tárolva. A *useArtifactModelSearch* hook felel ezek visszaalakításáért a megjelenítéshez használható formátumba.

A 3D modellek megjelenítését a *Babylon.js* grafikus motor segítségével valósítjuk meg. A folyamat során a Base64 formátumból visszaállított fájl betöltődik a jelenetbe, a modell megjelenítődik egy `canvas` elemben, megfelelő világítással (fényforrások beállítása).

A betöltés során a felhasználói élmény javítása érdekében egy töltőkör jelenik meg, amely vizuális visszajelzést ad a folyamat előrehaladásáról. Ezzel a megoldással biztosítható a zökkenőmentes és felhasználóbarát 3D megjelenítés.

3.1.8. Adminisztráció

A weboldal nemcsak látogatói funkciókat lát el, hanem adminisztrációs felületként is szolgál. Az adminisztrátorok számára ikonok jelzik a szerkesztési lehetőségeket, amelyek csak akkor jelennek meg, ha a bejelentkezett felhasználó megfelelő jogosultságokkal rendelkezik. Ezzel biztosítva van, hogy a szerkesztési, létrehozási vagy törlési műveletekhez kizárólag illetékes személyek férjenek hozzá.

A CRUD műveletek kialakítása során kiemelt figyelem volt fordítva a felhasználóbarát megjelenésre. Az adatok frissítésekor a meglévő információk automatikusan betöltődnek a szerkesztőfelületre, megkönnyítve ezzel a módosítások végrehajtását. Az egyszerű, intuitív kezelőfelület célja, hogy az adminisztrátorok gyorsan és hibamentesen tudják karbantartani a temető adatait.

3.1.9. Ortofotó

A temetőn belüli tájékozódást jelentősen megkönnyíti egy ortofotó réteg használata. Az ortofotó egy olyan légi felvétel, amely torzításmentes, térképészeti pontosságú képet ad a

felszínről, így részletesebb és valósághűbb megjelenítést nyújt, mint a hagyományos térképi elemek.

A weboldalon jelenleg egy manuálisan konfigurált saját Geoserver[9] által szolgáltatott WMS (Web Map Service) réteg biztosítja az ortofotó megjelenítését. Ez a réteg a *react-leaflet* könyvtáron keresztül integrálódik a térképre, és lehetőséget ad a felhasználónak a pontosabb eligazodásra a temető területén.

Jelenleg egyetlen temetőről készült ortofotó, azonban a jövőben célkitűzés ennek a folyamatnak az automatizálása, hogy minél több helyszínen elérhető legyen ez a hasznos navigációs segédeszköz.

3.1.10. Úthálózat

A temető úthálózatának pontos leképezése és fenntartása érdekében az adminisztrátorok számára lehetőség van utak rajzolására és szerkesztésére. Ezt a funkciót a *terra-draw* csomag biztosítja, amely lehetővé teszi Polyline-ok létrehozását, módosítását és törlését a térképen. Az adminok így tudnak felrajzolni egy olyan összefüggő gráfot, amely a későbbi útvonaltervezés alapját képezi.

A cél egy egyetlen összefüggő komponensből álló úthálózat kialakítása, amely minden fontos pontot összeköt. A rendszer biztosít egy intuitív ellenőrzési lehetőséget is: az *ellenőrzés* gomb megnyomására az adatbázisban tárolt gráf elemzésre kerül, és visszajelzést ad arról, hogy az úthálózat összefüggő-e.

Amennyiben a gráf több komponensre szakad, a rendszer automatikusan megkeresi a szétesett részek legközelebbi csomópontjait, majd a közöttük elméletileg hiányzó útvonal középpontjában egy kört jelenít meg a térképen, amely bekarikázza a problémás részt. Ez vizuálisan is kiemeli a potenciális hibát, és megkönnyíti a hiányzó él manuális pótlását, biztosítva ezzel az úthálózat teljességét és megbízhatóságát.

3.2. Nemzetköziesítés

A weboldal nemzetköziesítésének célja, hogy a különböző nyelvű felhasználók számára is könnyen használható és érthető felületet biztosítson. A statikus szöveges elemek, mint például címkék, gombok feliratai és üzenetek, a *react-i18next* csomag segítségével kerültek lefordításra.

A fordításokat nyelvenként külön-külön JSON fájlokban tároljuk. Jelenleg három nyelvet támogat az alkalmazás: magyar, angol és román. A megfelelő nyelvi fájl automatikusan

betöltésre kerül a felhasználó által választott nyelv alapján, vagy az alapértelmezett beállítás szerint. A kiválasztott nyelvi opció elmentésre kerül. A következő látogatás alkalmával már az azelőtt kiválasztott nyelv lesz betöltve.

A *react-i18next* rugalmasságának köszönhetően a nyelvváltás az oldalon dinamikusan, újratöltés nélkül történik. Ez különösen fontos a felhasználói élmény szempontjából, hiszen lehetővé teszi, hogy a látogatók gördülékenyen váltsanak a támogatott nyelvek között anélkül, hogy elveszítenék az aktuális munkamenetüket vagy navigációs helyzetüket.

4. Mobil

A mobilalkalmazás lehetőséget biztosít a sírhelyek, valamint a jeles elhunyt személyek adatainak megtekintésére. A felhasználók szűrési feltételek megadásával könnyen lehet keresni az adatbázisban szereplő bejegyzések között, például név, évszázad vagy foglalkozás szerint. A szűrési funkciók célja, hogy gyorsan és hatékonyan megtalálhatóak legyenek a sírhelyek vagy személyek.

Az alkalmazás felhasználóbarát felülettel rendelkezik, amely megkönnyíti az adatok böngészését és a részletes információk megtekintését. A sírhelyekhez kapcsolódó képek és egyéb műtárgyak is megjeleníthetőek, így vizuális élményt is nyújt a felhasználóknak. A rendszer támogatja a geolokáció alapú keresést is, amely segítségével a felhasználók aktuális helyzetük alapján találhatják meg a legközelebbi sírhelyeket.

Összességében a mobilalkalmazás célja az volt, hogy gyors hozzáférést biztosítson a temetők adatbázisához, megkönnyítve ezzel a helyszíni tájékozódást és az ismeretátadást. A következőkben a főbb funkcionalitásai lesznek részletezve.

4.1. Térkép

Hasonlóan a webes felülethez, a mobilalkalmazásban is lehetőség van a sírhelyek lokációinak megjelenítésére a temető térképén. A térképen megjelenített sírhelyek interaktív jelölőkkel vannak ellátva, amelyekre kattintva részletes információkat olvashatunk az adott sírról, például az elhunytak nevéről, életrajzi adatairól.

A térkép megjelenítése natív elemeket használva a `react-native-maps`[10] csomag segítségével történik. Ez biztosítja a térkép gördülékeny, gyors és interaktív kezelését, beleértve a helymeghatározást.

4.2. Ajánlott útvonal

Az alkalmazás támogatja az útvonaltervezést is: a felhasználó aktuális pozíciója alapján kiszámítja és kirajzolja a javasolt útvonalat a kiválasztott sírhelyhez. Ehhez a rendszer engedélyt kér a helyadatok elérésére, majd a szerver felé kérést küld a szükséges koordinátákkal. A megjelenített útvonal vizuálisan is jól követhető, szürke folytonos vonallal van feltüntetve, jelezve, hogy a felhasználó számára ajánlott ösvényt mutatja.

4.3. Sírfelismerés

A mobilalkalmazás egyik kiemelkedő funkciója a kiterjesztett valóság (AR), amely a `ViroReact` könyvtár segítségével valósult meg. Ennek a funkciónak a célja, hogy az alkalmazás a temetőben képes legyen felismerni a sírhelyeket a már meglévő fényképek alapján.

A rendszer úgy működik, hogy a felhasználó mobilkészüléke folyamatosan letölti a sírhelyekről készült képeket, miközben a készülék kameráján keresztül a környezetet figyeli. Ahogy a felhasználó egy sírhelyhez ér, a rendszer automatikusan váltogatja a képeket, amíg egy képet nem talál, amely megegyezik a jelenleg látható sírhely kinézetével. Ekkor a kiterjesztett valóság segítségével egy sötét háttérrel ellátott felirat jelenik meg, amely az adott sírhelyhez tartozó elhunytak nevét és foglalkozását tartalmazza.

A képernyő alján egy „További részletek” gomb is megjelenik, amely a felhasználót egy részletes információkat tartalmazó oldalra navigálja. Ezen az oldalon további információk érhetők el az elhunytak életéről, valamint egyéb, a sírhelyhez kapcsolódó adatok is megjelennek.

5. DevOps

A terméktulajdonos fontosnak tartotta, hogy az alkalmazás kipróbálásra kerüljön a fejlesztés korai szakaszában, ezt a csapat konténerizációval és orchesztrációval valósította meg, biztosítva ezzel a rendszer könnyű skálázhatóságát és telepíthetőségét. A Kubernetes Clusterre[3] történő telepítés lehetővé tette, hogy az alkalmazás valós környezetben is kipróbálható legyen, így gyorsabb visszajelzéseket kaphattunk, és ezek fényében folytathattuk a további fejlesztéseket.

5.1. Konténerizáció

Az alkalmazás Docker[16] konténerekben fut, amelyeket egy Kubernetes Cluster kezel és futtat. Ez a konténerizációs megoldás biztosítja a rugalmas és könnyű kitelepítést különböző környezetekben, csökkentve ezzel a telepítési hibák kockázatát. A konténerizált alkalmazás lehetővé teszi a környezetek közötti zökkenőmentes átviteleket, és az erőforrások optimális kihasználását. Emellett az automatikus skálázás, terheléselosztás és erőforrás-optimalizálás biztosítja, hogy az alkalmazás stabilan és hatékonyan működjön különböző terhelések mellett is.

A weboldalt, a React által generált statikus fájlokat egy nginx[23] szerver biztosítja, amely megfelelően kezeli az statikus tartalmak gyors betöltését és kiszolgálását.

A szerver alapjául az `eclipse-temurin:21.0.3_9-jre-jammy` Docker image szolgál, amely biztosítja a megfelelő Java futtatási környezetet az alkalmazás számára

5.2. Kubernetes Cluster

A Kubernetes lehetőséget ad a szolgáltatások automatikus skálázására, így az alkalmazás képes alkalmazkodni a forgalom dinamikus változásaihoz, ezáltal biztosítva a felhasználói élményt, függetlenül a terheléstől. Az orchesztrációs rendszer emellett támogatja a naplózást, hibakezelést és a rendszer figyelését, így minden komponens folyamatosan monitorozva van, és megfelelő módon reagál a problémákra.

Ez a megoldás lehetővé teszi a skálázhatóságot, a megbízhatóságot és a gyors hibakezelést, biztosítva ezzel az alkalmazás magas rendelkezésre állását és a folyamatos szolgáltatást.

A 11. kódrészlet részletezi az `udv-geoserver` Geoserver szolgáltatás Kubernetes klaszterben történő kitelepítését. Az alkalmazás a `docker.osgeo.org/geoserver:2.26.1` Docker image-t használja, amely a Geoserver egy stabil verzióját tartalmazza, és 8080-as porton

elérhető a konténeren belül.

A deployment konfigurációja tartalmazza az alapvető beállításokat, mint például a **Replicas**, amely jelzi, hogy a szolgáltatás egy példányban fut. Skálázhatóság érdekében szükség

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: udv-geoserver
  namespace: graveyard-explorer-ns
spec:
  replicas: 1
  selector:
    matchLabels:
      app: udv-geoserver
  template:
    metadata:
      labels:
        app: udv-geoserver
    spec:
      containers:
        - name: udv-geoserver
          image: docker.osgeo.org/geoserver:2.26.1
          ports:
            - containerPort: 8080
          env:
            - name: GEOSERVER_ADMIN_USER
              valueFrom:
                secretKeyRef:
                  name: udv-geoserver-secret
                  key: GEOSERVER_USER
            - name: GEOSERVER_ADMIN_PASSWORD
              valueFrom:
                secretKeyRef:
                  name: udv-geoserver-secret
                  key: GEOSERVER_PASSW
            - name: SKIP_DEMO_DATA
              value: "true"
          volumeMounts:
            - mountPath: /opt/geoserver_data/
              name: geoserver-data
      volumes:
        - name: geoserver-data
          persistentVolumeClaim:
            claimName: udv-geoserver-pvc
```

11. kódrészlet. Geoserver Kubernetes kitelepítését leíró állomány

esetén egyszerűen növelhető a replikák száma. **Enviroment Variables** az adminisztrátor felhasználó és jelszó a Kubernetes titkos (secret) menedzsment segítségével kerül beállításra. Ez biztosítja, hogy az érzékeny adatokat ne tároljuk közvetlenül a konfigurációs fájlokban. A `GEOSERVER_ADMIN_USER` és `GEOSERVER_ADMIN_PASSWORD` változók értékei a `udv-geoserver-secret` nevű titokból származnak. **Persistent Volume** (PV) és **Persistent Volume Claim** (PVC) a Geoserver adatokat egy állandó tároló helyre, a (`geoserver-data`) menti. A PVC `udv-geoserver-pvc` nevű tároló lehetővé teszi, hogy az adatok a konténer újraindítása után is meg legyenek őrizve. **Volume Mounts** a `/opt/geoserver_data/` könyvtárat a tárolóhoz csatolják, így a Geoserver képes lesz elmenteni a konfigurációkat és egyéb adatokat.

6. Működés

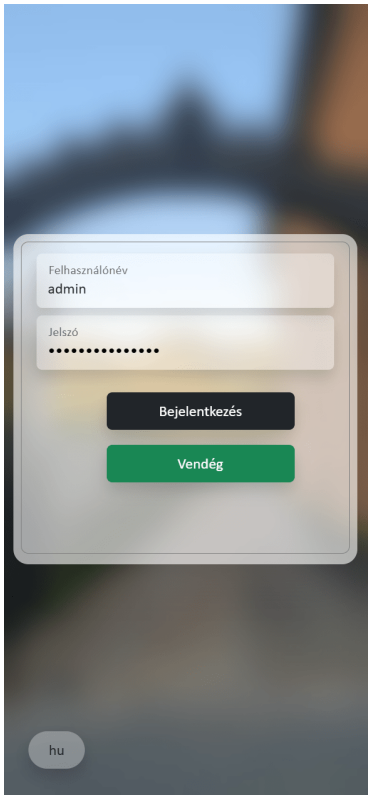
A következőkben a weboldal, illetve a mobilalkalmazás működése lesz részletezve.

6.1. Web

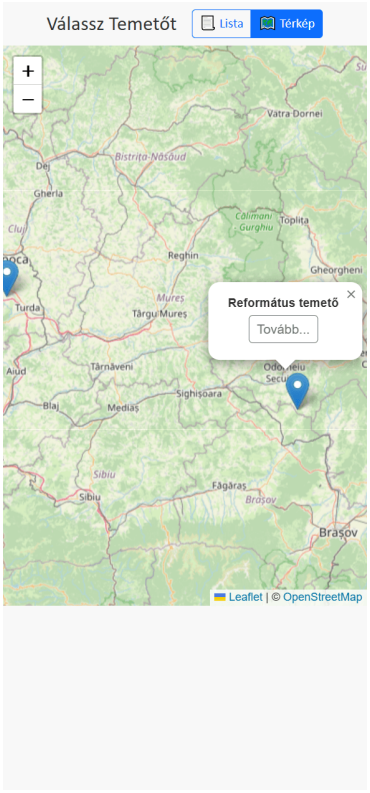
A weboldal kialakításánál fontos szempont, hogy a mobilos nézet is megfelelően működjön. Ennek érdekében a megjelenés reszponzív módon lett kialakítva, hogy a felhasználók különböző eszközökön is kényelmesen navigálhassanak.

A 7. ábrán látható bejelentkezési oldalon a felhasználók lehetőséget kapnak a bejelentkezésre vagy a vendég státuszú böngészésre. Ha nem történik bejelentkezés, a felhasználók vendégként tudják böngészni a temető adatait, azonban a CRUD műveletekhez szükséges ikonok nem állnak rendelkezésre.

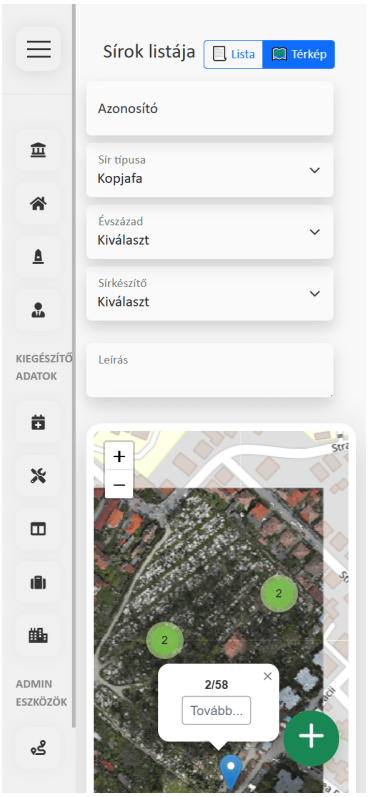
A 8. ábrán a temető kiválasztása látható. A temetőválasztó oldal lehetővé teszi a temetők listázását, és ezen kívül egy térképes megjelenítést is biztosít, ahol a temetők földrajzi elhelyezkedése szerint lehet keresni. A temetők jelölőire kattintva megjelenik a temető neve



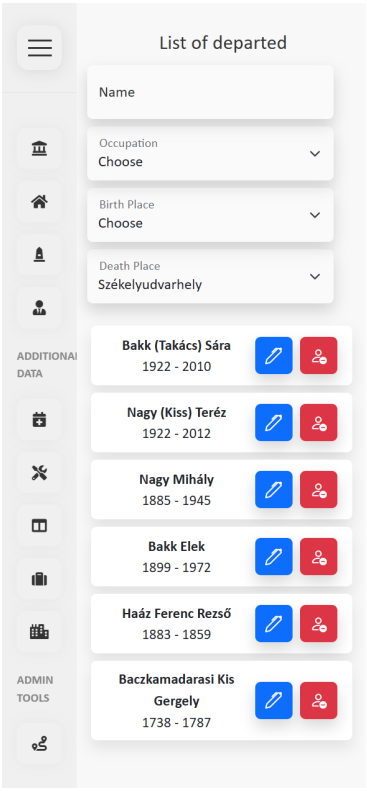
7. ábra. Bejelentkezési oldal mobil nézetben



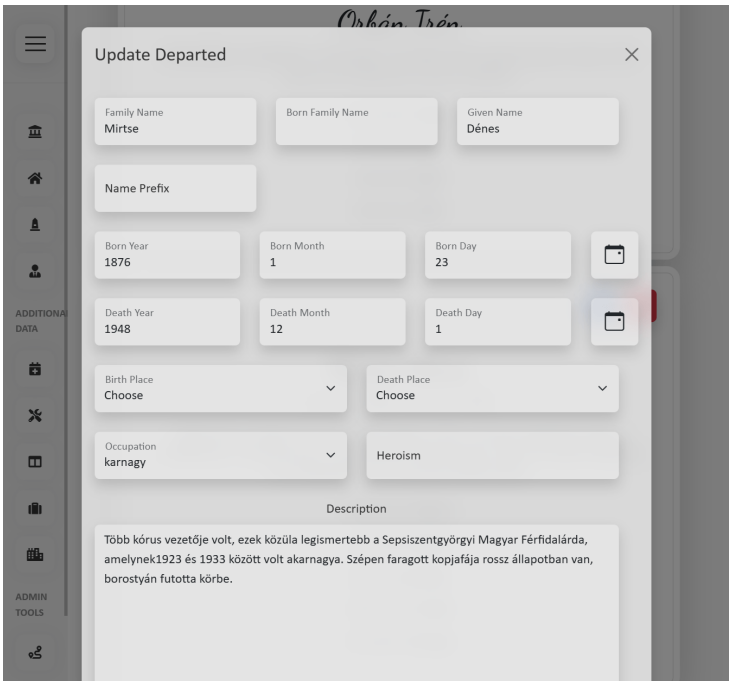
8. ábra. Temető kiválasztása térképen mobil nézetben



9. ábra. A szűrt sírhelyek helyének jelzése térképen mobil nézetben



10. ábra. Jeles személyek listázása adminisztrációs ikonokkal mobil nézetben



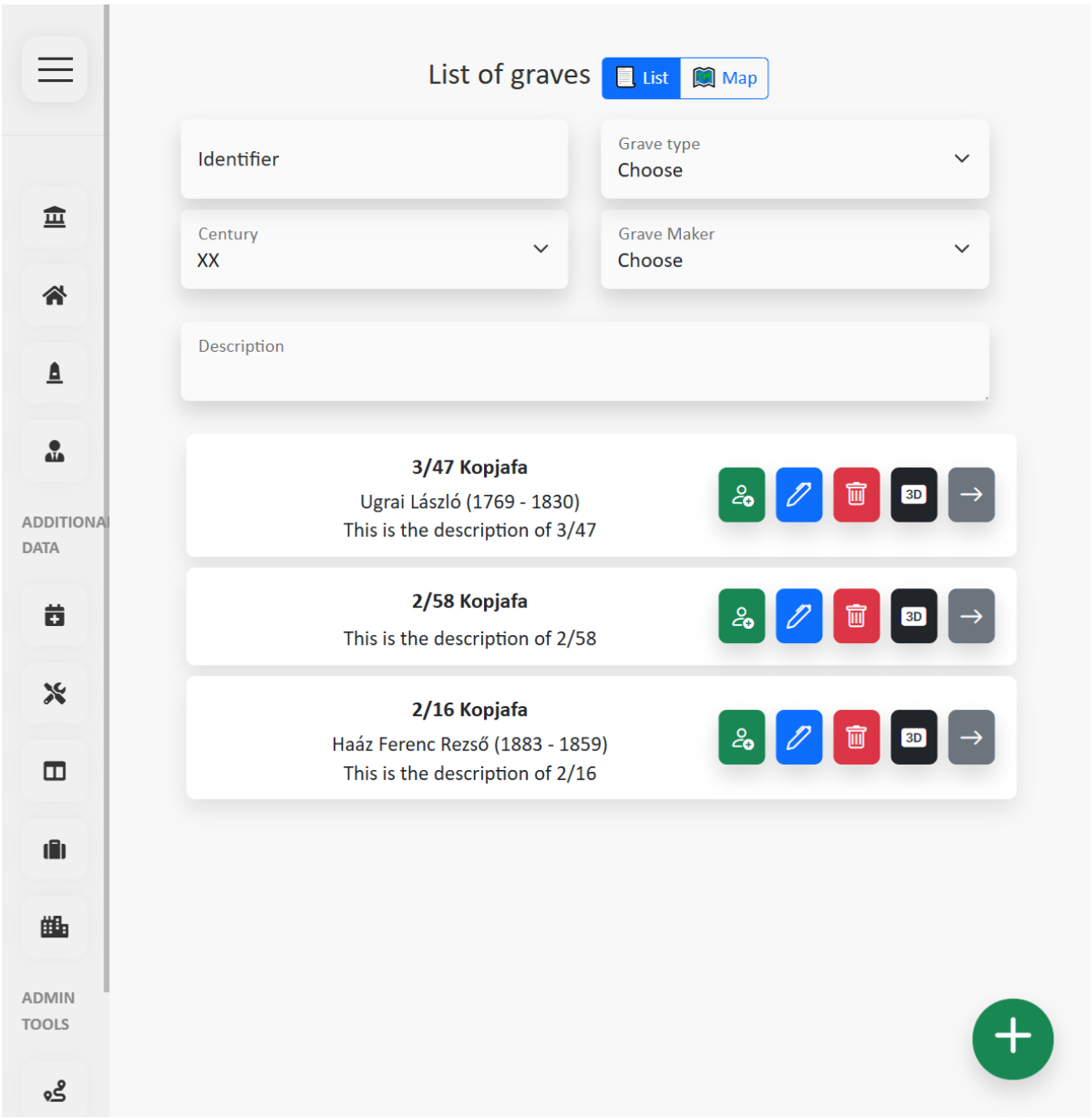
11. ábra. Jeles személy adatainak frissítésére alkalmas felugró ablak

és egy gomb, amely segítségével tovább lehet navigálni a temető oldalára.

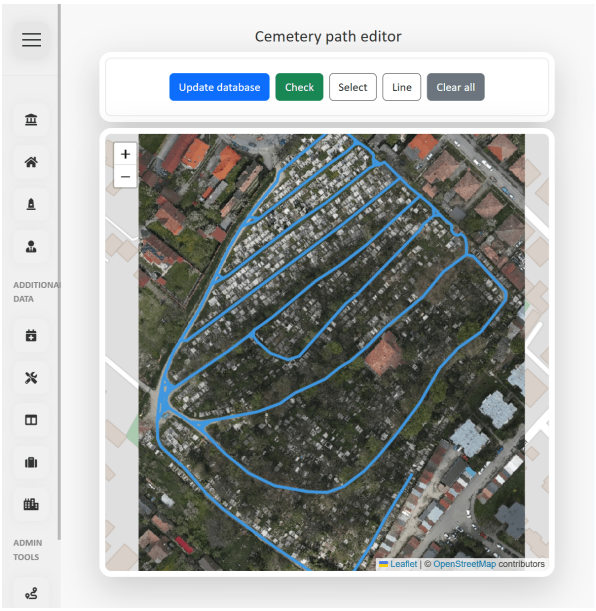
A temető kiválasztása után a rendszer lehetőséget biztosít a sírhelyek (9. ábra) vagy elhunytak listázására. A felhasználók szűrhetik a listát különböző paraméterek, mint például az elhunytak neve, születési és halálozási dátuma, települése (10. ábra), vagy a sírhely típusa alapján. Mivel a sírhelyek koordinátái rendelkezésre állnak, a térképes megjelenítés segítségével ezek is megjeleníthetők, így a felhasználók könnyen navigálhatnak a temetőben a keresett sírhelyekhez.

A rendszer lehetőséget ad a kiegészítő adatok, mint például az évszázadok, sírkészítők, sír típusok, foglalkozások és települések szűrésére is. Ezekre az adatokra kattintva az alkalmazás átirányítja a felhasználót a kapcsolódó információkhoz és a megfelelő szűrési feltételt automatikusan alkalmazza, ahogy a 12. ábrán látható. Így például ha valaki a sírkészítők listájára kattint, csak azok a sírhelyek fognak megjelenni, amelyek az adott sírkészítőhöz kapcsolódnak.

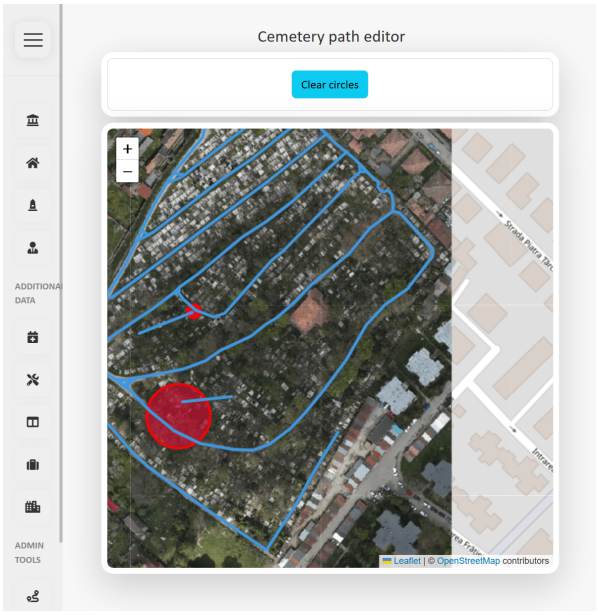
Az adatok létrehozása és szerkesztése felugró ablakokban történik, ahogyan az a 11. ábrán is látható.



12. ábra. XX. századból származó sírhelyek listázása



13. ábra. A temető úthálózatának a módosítási felülete a szerkesztő lehetőségekkel



14. ábra. Úthálózat gráfjának nem összefüggő komponenseinek kiemelése piros körrel

15. ábra. 3D modell megjelenítése

A 13. ábrán látható, hogy a weboldal lehetőséget ad útvonalak felrajzolására, amelyek alapján a későbbi útkeresés valósul meg. Az útkeresés hatékony megvalósítása érdekében ajánlott az útvonalakat egy komponensből álló gráfként elmenteni, így biztosítva az összefüggőséget és az optimális navigációt. Amennyiben az útvonalhálózat nem egyetlen összefüggő komponensből áll, a rendszer a 14. ábrán látható módon piros körrel jelöli a problémás, nem megfelelően kapcsolódó pontokat, ezzel segítve a hibák gyors azonosítását és javítását.

A 3D modellek megtekintésére is lehetőség van, amelyet a 15. ábra szemléltet. Itt

a felhasználók térbeli nézetben böngészhetik az objektumokat, forgathatják, nagyíthatják, részletesebben is megvizsgálhatják.

A kiválasztott sírhely információs oldalán egy útjelző tábla ikon jelenik meg, amely egy interaktív funkciót biztosít a felhasználók számára. A felhasználók rákattintva át lesznek irányítva egy egész képernyős térképre, amely az adott sírhelyhez vezető útvonalat mutatja.

Miután a felhasználó engedélyezi a helymeghatározási jogosultságokat, a térkép az aktuális pozícióját figyelembe véve megjeleníti a legoptimálisabb útvonalat a sírhelyhez. A térképen a felhasználó nyomon követheti az útvonalat, és az alkalmazás egyértelmű navigációs jelekkel segíti őt a célhoz való eljutásban. Ez a funkció különösen hasznos azok számára, akik először járnak a temetőben, mivel lehetővé teszi számukra, hogy gyorsan és kényelmesen elérjék a keresett sírhelyet.

A navigáció és a felhasználói élmény minimalista, intuitív módon van kialakítva, így a felhasználók könnyen át tudják megtekinteni a temető összes releváns adatát.

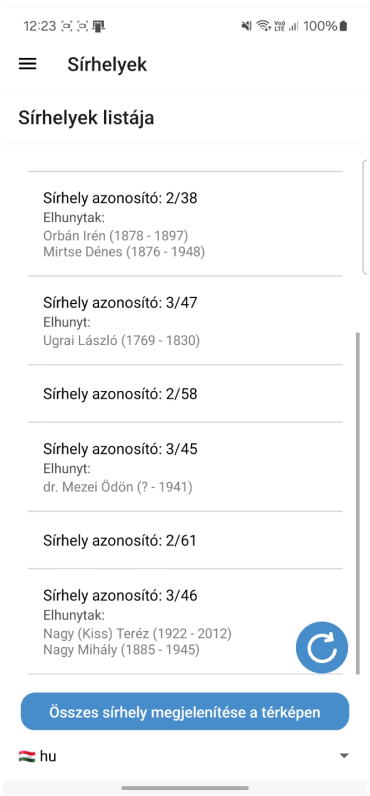
6.2. Mobil

A 16. ábrán látható mobilalkalmazás kezdőképernyőjén a képernyő bal felső sarkában elérhető egy ikon, amely megjeleníti a navigációs sávot, amelyen könnyen hozzá lehet férni a különböző funkciókhoz. A sávon keresztül lehetőség van a sírhelyek és elhunytak listázására (17. ábra), majd ezek közül egy kiválasztása után az alkalmazás automatikusan az odavezető utat mutatja a térképen, ahogy a 18. ábrán is látható, amennyiben a felhasználó által engedélyezve lett a helymeghatározás.

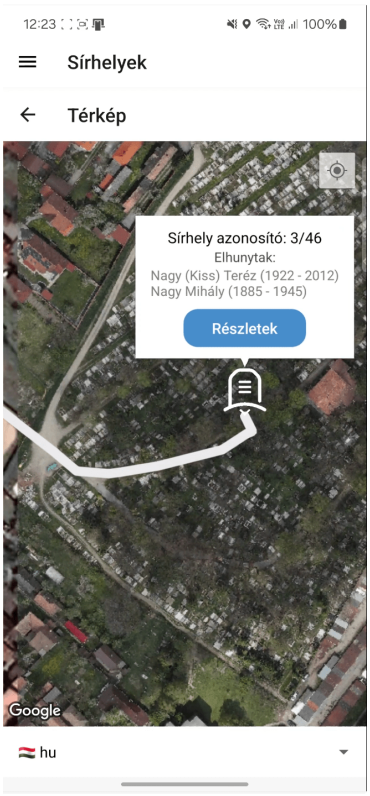
A térképen a kiválasztott sírhelyhez egy jelölő kerül elhelyezésre, amelyre kattintva egy felugró panel jelenik meg. Ezzel a panel segítségével a felhasználó könnyen átirányítható a sírhelyek részletes adatainak megtekintésére (19. ábra), ahol a sírhelyhez kapcsolódó információk, fényképek és az elhunytak adatai is megjelennek (20. ábra). Továbbá lehetőség van az összes sírhely megjelenítésére a térképen, hogy a felhasználók könnyebben navigálhassanak az adott temetőn belül.

A mobilalkalmazás lehetőséget biztosít a sírhelyek felismerésére, ahogy a 21. ábrán is látható. Ez a funkcionalitás a kameraképet elemezve megpróbálja azonosítani a sírhelyet. Sikeres felismerés esetén a rendszer megjeleníti a sírban nyugvó jeles személy vagy személyek neveit, valamint foglalkozásukat. A felhasználó további részleteket is megtekinthet a sírhelyről a *Tudj meg többet* kiválasztása után.

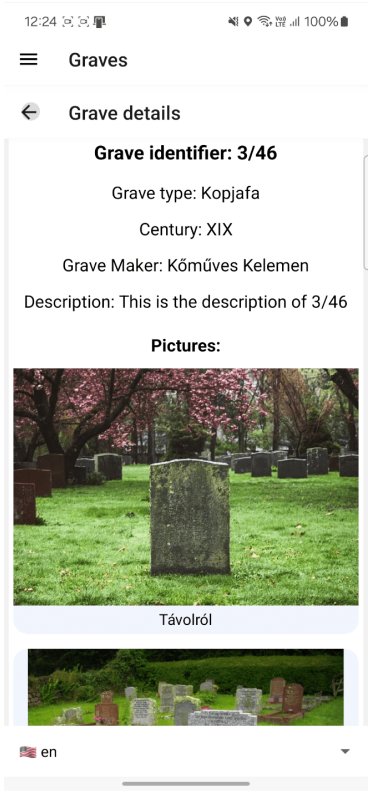
16. ábra. Kezdőképernyő magyar nyelven



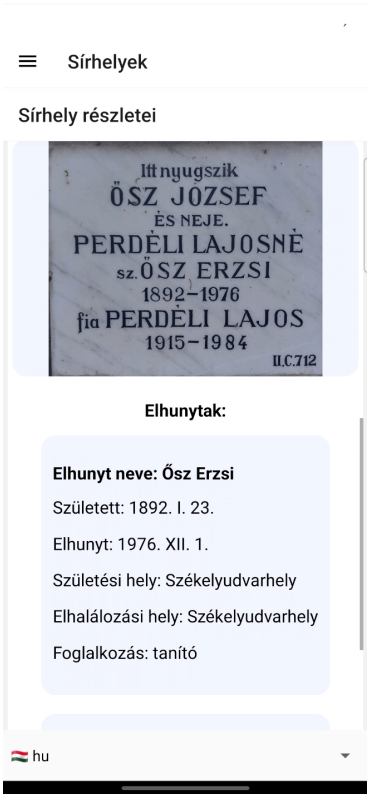
17. ábra. A sírhelyek listázása



18. ábra. Javasolt útvonal a kiválasztott sírhelyig



19. ábra. Sírhely adatainak részletezése



20. ábra. Az elhunyt adatainak bővebb változata

21. ábra. Kiterjesztett valóság használata a sírhely felismerésére

Következtetések és továbbfejlesztési lehetőségek

A Graveyard Navigator projekt eredményeként egy olyan rendszer jött létre, amely már használható állapotban van, és alapvető funkcióival képes támogatni a temetőkben való tájékozódást és információkeresést. Ugyanakkor számos továbbfejlesztési lehetőség is felmerült, amelyek még tovább bővítenék és színesítenék az eddigi funkciókészletet.

Ilyen bővítési irány lehet például ortofotók feltöltésének és használatának automatizált lehetősége a temetők esetében. Szintén fontos lenne az útkeresési funkció továbbfejlesztése olyan módon, hogy a navigáció kizárólag a temető területén belül történjen, a temető bejáratainak figyelembevételével és jelzésével.

Hiányosságként említhető még az impresszum, illetve a temető gondozóinak elérhetőségeit tartalmazó információs oldal létrehozása, amely jogi és tájékoztatási szempontból is alapvető fontosságú lenne.

További jelentős fejlesztési lehetőség egy átfogó felhasználókezelési rendszer kialakítása, amely magában foglalná a felhasználók létrehozását, szerepkörök kiosztását, jogosultságkezelést, valamint a felhasználói profilok szerkesztését és menedzselését. Ez lehetővé tenné például adminisztrátorok, gondozók eltérő jogosultsági szinteken történő beléptetését és kezelését, hozzájárulva a rendszer biztonságosabb és testreszabottabb működéséhez.

Hivatkozások

[1] Adobe. *What is Normal Mapping?* Hozzáférés dátuma: 2025-04-26. 2025. URL: https://www.adobe.com/products/substance3d/discover/normal-mapping.html.

[2] Auth0. *What Are Refresh Tokens and How to Use Them Securely*. Hozzáférés dátuma: 2025-04-20. 2021. URL: https://auth0.com/blog/refresh-tokens-what-are-they-and-when-to-use-them/.

[3] The Kubernetes Authors. *Kubernetes Documentation*. https://kubernetes.io/docs/. Hozzáférés dátuma: 2025-04-27. 2015.

[4] Autodesk. *Diffuse Mapping - Autodesk Flame Family 2024*. Hozzáférés dátuma: 2025-04-26. 2024. URL: https://help.autodesk.com/view/FLAME/2024/ENU/?guid=GUID-624238CC-71B2-4990-9EEA-0B237AD2A6C1.

[5] Baeldung. *Spring Data JPA @Query*. Hozzáférés dátuma: 2025-04-20. 2024. URL: https://www.baeldung.com/spring-data-jpa-query.

[6] Baeldung. *What Is OncePerRequestFilter?* Hozzáférés dátuma: 2025-04-19. 2025. URL: https://www.baeldung.com/spring-onceperrequestfilter.

[7] Teric Cabrel. *Role-based access control in Spring Boot*. Hozzáférés dátuma: 2025-04-20. 2024. URL: https://blog.tericcabrel.com/role-base-access-control-spring-boot.

[8] Checkstyle. *Checkstyle*. Hozzáférés dátuma: 2025-04-27. 2023. URL: https://checkstyle.sourceforge.io.

[9] GeoServer Community. *GeoServer User Manual*. 2024. URL: https://docs.geoserver.org/.

[10] React Native Maps contributors. *React Native Maps*. https://github.com/react-native-maps/react-native-maps. Hozzáférés dátuma: 2025-04-27. 2015.

[11] Epic Games. *RealityScan: 3D Scanning App*. Hozzáférés dátuma: 2025-04-26. 2023. URL: https://www.unrealengine.com/en-US/realityscan.

[12] Farhan. *Spring Security JWT Authentication & Authorization*. Hozzáférés dátuma: 2025-04-20. 2023. URL: https://medium.com/code-with-farhan/spring-security-jwt-authentication-authorization-a2c6860be3cf.

[13] Roy T. Fielding. „Architectural Styles and the Design of Network-based Software Architectures”. Disszertáció. University of California, Irvine, 2000. URL: https://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm.

[14] Martin Fowler és mások. *Patterns of Enterprise Application Architecture*. Addison-Wesley Professional, 2002.

[15] PostgreSQL Global Development Group. *PostgreSQL Documentation*. 2024. URL: https://www.postgresql.org/docs/.

[16] Docker Inc. *Docker Documentation*. https://docs.docker.com/. Hozzáférés dátuma: 2025-04-27. 2013.

[17] MinIO Inc. *MinIO Object Storage Documentation*. 2024. URL: https://min.io/docs.

[18] *Introduction to JSON Web Tokens*. Hozzáférés dátuma: 2025-04-20. URL: https://jwt.io/introduction.

[19] M. Jones, J. Bradley és N. Sakimura. *JSON Web Token (JWT)*. https://www.rfc-editor.org/rfc/rfc7519. RFC 7519. 2015. URL: https://www.rfc-editor.org/rfc/rfc7519.

[20] Khronos Group. *glTF 2.0 Specification*. Hozzáférés dátuma: 2025-04-26. 2021. URL: https://registry.khronos.org/glTF/specs/2.0/glTF-2.0.html.

[21] Inc. Meta Platforms. *React – A JavaScript library for building user interfaces*. Hozzáférés dátuma: 2025-04-19. 2025. URL: https://react.dev/.

[22] Inc. Meta Platforms. *React Native Documentation*. Hozzáférés dátuma: 2025-04-19. 2025. URL: https://reactnative.dev/docs/getting-started.

[23] F5 Networks. *NGINX Documentation*. https://docs.nginx.com/. Hozzáférés dátuma: 2025-04-27. 2004.

[24] OGC. *Web Map Service (WMS) Implementation Specification*. 2024. URL: https://www.ogc.org/standards/wms.

[25] Nadin Pethiyagoda. *REST API Naming Conventions and Best Practices*. Hozzáférés dátuma: 2025-04-19. 2022. URL: https://medium.com/@nadinCodeHat/rest-api-naming-conventions-and-best-practices-1c4e781eb6a5.

[26] pgRouting contributors. *pgr_aStar - A* algorithm for the shortest path*. Hozzáférés dátuma: 2025-04-20. 2025. URL: https://docs.pgrouting.org/3.6/en/pgr_aStar.html.

[27] pgRouting contributors. *pgr_connectedComponents — Connected components of an undirected graph*. Hozzáférés dátuma: 2025-04-20. 2025. URL: https://docs.pgrouting.org/3.6/en/pgr_connectedComponents.html.

[28] pgRouting contributors. *pgRouting: Routing on PostgreSQL*. 2025. DOI: 10.5281/zenodo.15004469. URL: https://pgrouting.org/.

[29] pgRouting contributors. *ST_Distance - PostGIS*. Hozzáférés dátuma: 2025-04-20. 2024. URL: https://postgis.net/docs/ST_Distance.html.

[30] PMD. *PMD*. Hozzáférés dátuma: 2025-04-27. 2023. URL: https://pmd.github.io.

[31] PostGIS Development Group. *PostGIS Documentation*. Hozzáférés dátuma: 2025-04-19. 2025. URL: https://postgis.net/documentation/.

[32] Akurat Software. *React-Leaflet-Cluster Documentation*. Hozzáférés dátuma: 2025-04-27. 2023. URL: https://akursat.gitbook.io/marker-cluster.

[33] SpotBugs. *SpotBugs*. Hozzáférés dátuma: 2025-04-27. 2023. URL: https://spotbugs.github.io.

[34] Spring Team. *Spring Security Reference*. Hozzáférés dátuma: 2025-04-20. 2024. URL: https://docs.spring.io/spring-security/reference/.

[35] Leaflet Team. *Leaflet Documentation*. Hozzáférés dátuma: 2025-04-27. 2023. URL: https://leafletjs.com/reference.html.

[36] React-Leaflet Team. *React-Leaflet Documentation*. Hozzáférés dátuma: 2025-04-27. 2023. URL: https://react-leaflet.js.org/docs/start-introduction.

[37] Spring Team. *How to integrate Hibernate's Multitenant feature with Spring Data JPA in a Spring Boot application*. Hozzáférés dátuma: 2025-04-19. 2022. URL: https://spring.io/blog/2022/07/31/how-to-integrate-hibernates-multitenant-feature-with-spring-data-jpa-in-a-spring-boot-application.

[38] Spring Team. *Spring Framework Documentation*. https://spring.io/projects/spring-framework. 2024.