ETDK-dolgozat

Daczó Dávid

Szász Lilla-Emese

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

Kolozsvár, 2022. május 19–22.

Madarak megfigyelése kutatás és szórakozás céljából

Szerzők:

Daczó Dávid

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

Szász Lilla-Emese

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

Témavezetők:

dr. Sulyok Csaba, egyetemi adjunktus

Babeş–Bolyai Tudományegyetem, Matematika és Informatika Kar

Bartha Vivien, szoftverfejlesztő,

Codespring

Fosztó Mátyás, szoftverfejlesztő,

Codespring

Tamás Andrea, szoftverfejlesztő,

Codespring

Kivonat

A Madármegfigyelő alkalmazás célközönsége magába foglalja nem csak a madarakat tanulmányozó zoológusokat, hanem az amatőr madárlesőket is. A projekt célja, hogy egy alkalmazásrendszert biztosítson tudományos célú adatgyűjtésre, statisztikai kimutatások generálására, és hobbi szintű madárkövetésre is. Erre van kifejlesztve egy mobil- és webalkalmazás is, melyek egy központi szerverhez csatlakoznak.

A webalkalmazással minden felhasználó meg tud jeleníteni korábban megfigyelt egyedeket a térképen, valamint ezeket tudja több szempont szerint szűrni. Tudósok képesek adatok listázására és felvitelére.

A mobilalkalmazás az egyszerű felhasználó számára van elképzelve, aki, mint a webes felületen, képes adatmegjelenítésre és -szűrésre, és ezek mellett lehetősége van értesítések beállítására, amelyek a jelenlegi helyzethez közeli megfigyelésekről tartalmaznak információkat.

Tartalomjegyzék

Bevezető

A Madármegfigyelő alkalmazás célja az, hogy lehetőséget nyújtson kutatás, valamint szórakozás céljából megfigyelt madarak archívumának a vizualizálására. Az alkalmazáson belül egy felhasználó meg tudja tekinteni, hogy egy madárfaj populációja hogyan helyezkedik el a térképen, valamint ez hogyan változik az idők során. Egy másik célkitűzése a Madármegfigyelő alkalmazásnak, hogy élvezetesebbé tegye a túrázást, felfokozza a kíváncsiságát a természetkedvelő embereknek új információk ismertetésével.

A rendszer magába foglal két kliens alkalmazást, valamint egy ezeket kiszolgáló központi szervert. A webes felület célközönsége a természetkutatók, biológusok csoportja, akik számára lehetőséget biztosít madármegfigyelések felvezetésére és kilistázására. A mobilkliens egy bővebb csoportot céloz meg: a természet kedvelőit, túrázókat vagy bárkit, aki a környezetében élő madarakról új ismereteket szeretne gyűjteni.

A webalkalmazást főként a térképnézet és az idetartozó műveletek teszik ki. A bonyolult, többrétegű szűrési lehetőségek jelentősen felgyorsíthatják a kutatók által óhajtott megfigyelések elérését. A lekért adatokból különböző következtetéseket tudnak levonni, bizonyos állatfaj vagy állatcsoport tulajdonságaira vonatkozóan.

A mobilalkalmazás a hétköznapi természetkedvelőknek biztosít felületet a közelben lévő madarak megismeréséhez. Egy átlagos felhasználó új információkat kaphat a környezetéről, számára eddig ismeretlen fajokat fedezhet fel és megújult kíváncsisággal, "nyitottabb" szemmel barangolhat az eddig megszokott hétköznapi élővilágban.

A dolgozat bemutatja először a projekt funkcionalitásait, majd a 2. és 3. fejezetben leírja a kliens- és szerver oldali technológiákat. Ezek után, a következő két fejezet ismerteti a projekt alatt használt eszközöket és módszereket. Végezetül, a dolgozatot a projekt működésének rövid leírásával, és a továbbfejlesztési lehetőségek felsorolásával zárul le.

A projekt ötletét a *Milvus*[1] természetvédelmi csoport kezdeményezte a Codespring mentorprogram[2] keretein belül, azzal a motivációval, hogy továbbfejlessze a már létező *OpenBirdMaps* [20] alkalmazás kinézetét, viszont a későbbiekben független projektté vált.

A projekten, a dolgozat szerzőin kívűl dolgoztak: Péter Kristóf a nyári gyakorlat végéig volt a csapat tagja, valamint az egyetemi csoportos projekt tantárgy alatt, egy fél évre csatlakozotak Lukács Panna, Kelemen Réka, Péter Anna Fanni és Szász Csongor egyetemi kollégák is.

[1] https://milvus.ro/en/
[2] https://edu.codespring.ro

1. A Madármegfigyelő projekt

A Madármegfigyelő alkalmazás elsősorban azzal a céllal készült, hogy egy platformot biztosítson a madarak életét tanulmányozó tudósok, kutatók és hétköznapi túrázok környezetében észlelhető madarak megfigyelésére, vizsgálatára. Emellett a szakemberek által összegyűjtött adatok feldolgozásában is hasznos szerepet tölt be. A szűrési feltételek alkalmazásával kinyerhetők a különböző madárfajok előhelyei és ezek változásai bizonyos időintervallumok között, ezáltal következtetéseket tudunk levonni a madárpopulációk előhelyeinek tulajdonságairól.

1.1. Funkcionalitások

A két kliens-, illetve a közös szerveralkalmazásban három szerepkört lehet megkülönböztetni: adminisztrátor, tudós illetve vendégfelhasználó. A szerepkörökhöz tartozó funkcionalitások a következő alfejezetben kerülnek bemutatásra.

1.1.1. A webes felület funkcionalitásai

A webes felület mindhárom szerepkör típusú felhasználó által használható, ámbár az elérhető funkcionalítások a jogosultságok alapján változnak. A felhasználó bejelentkezése után az alábbi funkcionalitások lesznek elérhetőek.

Bárki lekérheti az adatbázisban található összes madármegfigyelést, szűrést tud elvégezni rajtuk, illetve kilistázhatja egy megfigyeléshez tartozó részletesebb információkat. A felület többnyelvűséget is biztosít a jobb felhasználói élményért, valamint a térkép kinézete személyre szabható, például színes, egyszínű, topografikus.

A tudósok, az előbb leírt funkcionalitások mellett rendelkeznek adatmodósító, valamint adatfelvivő lehetőségekkel. Egy tudós szerepkörrel rendelkező felhasználó megfigyelést tud hozzáadni az adatbázishoz, ami ezután rögtön láthatóvá válik minden felhasználó számára. Ezesetben neki kell megadnia a megfigyeléshez tartozó információkat, későbbiekben módosíthatja ezeket, akár törölheti is. Saját vezérlőpulttal rendelkezik, amin elvégezhetők az előbb említett műveletek, valamint saját profiloldalát is megtekintheti, amin a személyes információit tudja szerkezteni.

Az adminisztrátor, azaz rendszergazda szintén tud megfigyelést hozzáadni. Az ő vezérlőpultja az adatok tisztán tartásáért, helyességéért felel. Külön táblázatokban elérheti

a felhasználókhoz tartozó adatokat, tudóssá tud statuálni egy átlag felhasználót. Emellett az országokhoz tartozó információkat ellenőrizheti, illetve a megfigyelések adatait tudja módosítani, helyesbíteni.

1.1.2. Az Android alkalmazás funkcionalitásai

A mobilos felület a standard felhasználók számára készült, az érdeklődésük növelése, valamint a túra élményének fokozása céljából. Ezt az applikáció új információk megjelenítésével kivitelezi. A felhasználó az alkalmazás megnyitása után gombnyomásra megkaphatja a szervertől a megfigyeléseket, amelyeket egy térképen megjelenítve tekinthet meg.

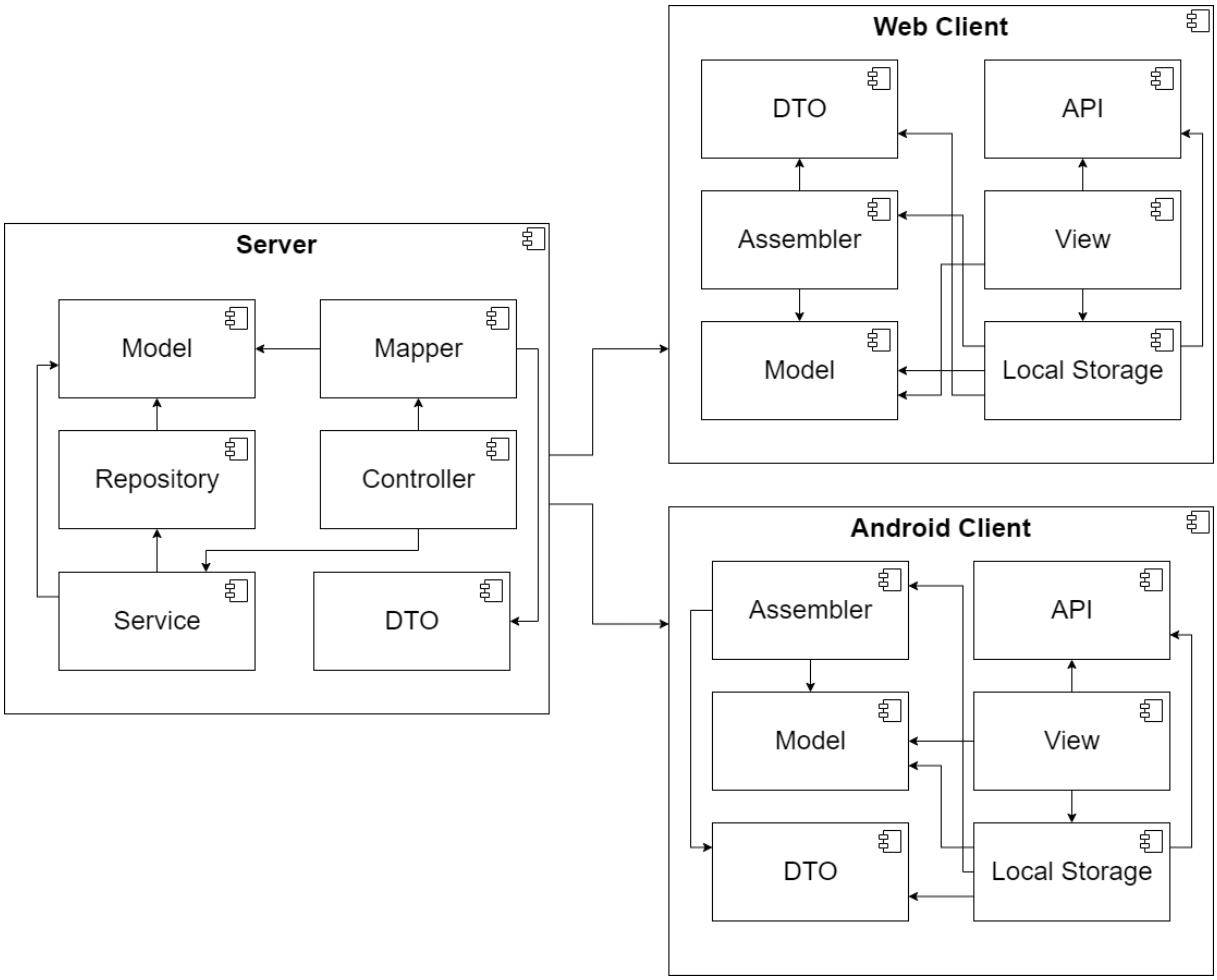
Ezen kívül a felhasználóknak lehetősége van egy adott sugáron belüli madármegfigyelések lekérésére. Ehhez meg kell határozni a koordinátáit, majd az érdekelt terület hozzávetőleges sugarát, ezen információkkal egy körterületet határozhat meg.

1.2. Architektúra

Az rendszer egy szerverből és két kliens alkalmazásból áll, lásd 1. ábra. Az ezek közötti kommunikáció HTTP kéréseken keresztül történik, DTO-k (Data Transfer Objects) [6] segítségével. A DTO-k szerepe, hogy a modellek megfelelő adattagjait fogják össze és így a teljes modell helyett csak a szükséges adatok kerüljenek bele a HTTP kérésekbe.

A szerver középpontjában egy adatbázis áll, amely tartalmazza a madarakhoz tartozó adatokat, valamint a felhasználok profiljait. Másik fontos feladata a kliensalkalmazások kéréseinek kiszolgálása.

Rétegei a Repository, Service, Model és Controller komponensek. A Model komponensekben találhatóak az adatbázisra leképezett adatmodellek. Ezek a szerver bármely rétegén elérhetőek és használhatóak. A Repository réteg felelős az adatbázissal való kommunikációért, ezt az alkalmazás a Hibernate ORM [14] rendszer segítségével végzi el. A Service az alkalmazás üzleti logika rétege, bizonyos műveletek elvégzése után előállítja az adatokat olyan formában, amilyenben az alkalmazásnak később szüksége lesz rájuk. A szerver a kliensektől érkező REST [29] típusú kéréseket a Controller rétegben fogadja, és a Service megfelelő részeit felhasználva állítja elő a szükséges választ rájuk. Amennyiben a kérés nem végrehajtható, könnyen értelmezhető hibaüzenetet, megfelelő hibakóddal küld visszajelzésként.



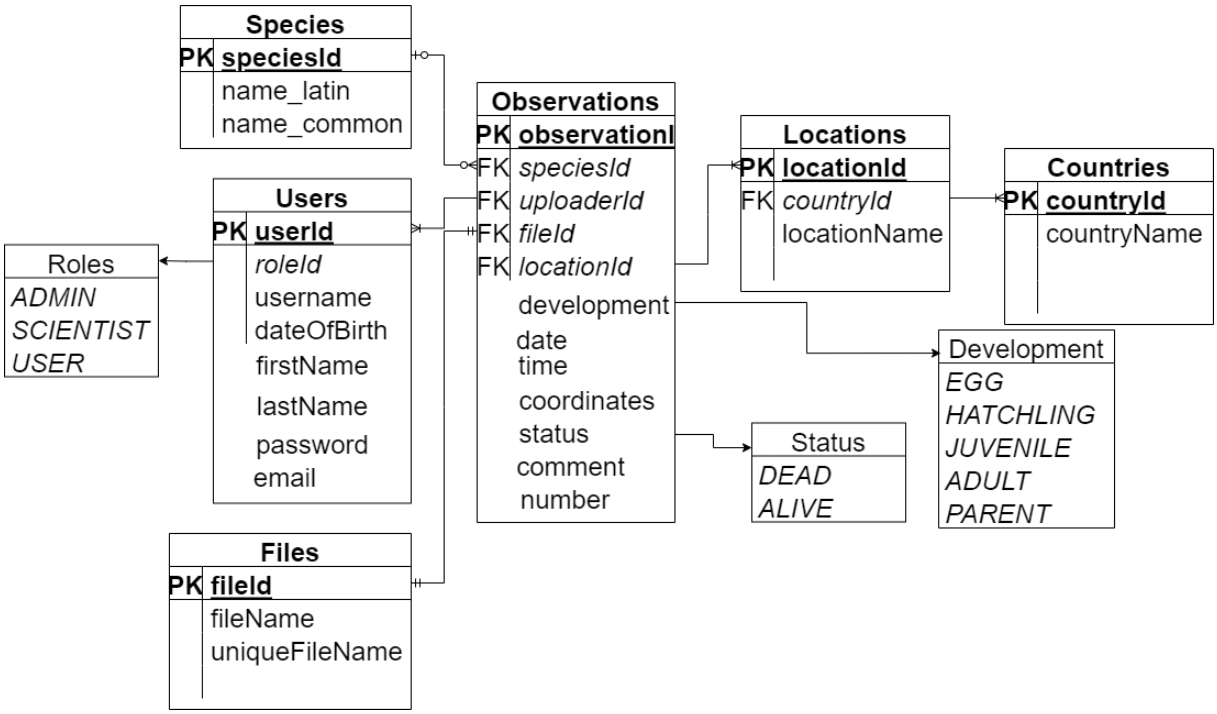
1. ábra. Architekturális diagram

A két kliens komponens hasonló felépítésű és hasonlóan is működik. Mindkettőben létezik egy API modul, amelyben a különböző kérést küldő függvények válaszát asszambler-ek `DTO`-ról egyszerű modellre alakítják. A kapott adatokat a `Local Storage`-ban tárolódnak el.

Mindkét kliens alkalmazás az MVC tervezési mintát implementálja, ahol a Model a DTO-ból átalakított modell, a View modul felelős az alkalmazás megjelenítéséért, valamint az irányításáért is.

1.3. Adatmodell

A Madármegfigyelő projektben a modell osztályok és mezőik (lásd 2. ábra) a Java Persistence API különböző annotációival vannak ellátva. A következő jelölések vannak a használva az alkalmazás helyességének megőrzése érdekében: `@Entity`, amely megjelöli az osztályról, hogy entitás, az `@Id` és `@GeneratedValue`, amelyek egy mezőről jelzik, hogy egyedi azonosító és generált értéket kap, és a `@NotNull`, amely értesíti a rendszert, hogy a



2. ábra. Adatmodell diagram

megjelölt adattag nem kaphat `null` értéket.

A projekt központi entitása az `Observation` osztály, amely egy fajnak egy megfigyelését tükrözi. A legfontosabb mezői az entitásnak a *geometry*, amelyben a megfigyelés koordinátái vannak eltárolva, valamint a *date*, *time*. Ezeken kívül még vannak olyan adattagok is, amelyek más entitásokra hivatkoznak. Az *uploader*, a megfigyelést feltöltő `User` (felhasználó), a *species* és *location* a megfigyelt fajt valamint a helységet tárolják.

A projekt tartalmaz három felsorolás típust is, a *role*, amelynek értéke egy felhasználó szerepkörét jelöli, és a *development* és *status* típusok, az előbbi a madár életkorárát jelzi, az utóbbi pedig azt, hogy életben van vagy már elhunyt.

1.4. Kommunikáció

A Madármegfigyelő alkalmazásban a kliens és szerver közötti kommunikáció a REST szoftverarchitektúrára épül. A kliens alkamazások HTTP kéréseken keresztül tudnak információt lekérni vagy felküldeni. Minden kéréshez tartozik egy művelet típus és egy elérési útvonal, ami 1. kódrészlet példában "GET" és "/api/users/username".

```
@PreAuthorize("hasAuthority('SCIENTIST') or hasAuthority('USER')
or hasAuthority('ADMIN')")
@GetMapping("/username")
public UserDto findByUsername(@RequestParam("username") String username)
throws NotFoundException {
    try {
        return userMapper.modelToDto(userService.findByUsername(username));
    } catch (NotFoundException e) {
        throw e;
    }
}
```

1. kódrészlet. A User lekérése felhasználónév szerint

1.4.1. Komponensek közötti kommunikáció

A kliensoldalon a komponensek közötti kommunikáció Mobx segítségével történt. Különböző adathalmazok tároló osztályokban vannak elkülönítve. Egy adott osztály példánya állapotmegfigyeltté válik (lásd 2. kódrészlet), ha ennek referenciája (`this` által) paraméterként átadódik a Mobx `makeObservable` vagy `makeAutoObservable` függvénynek. Az `makeAutoObservable` használata lényegesen egyszerűbb az előzőtől, hiszen nem szükséges átadni közvetlenül az osztály összes megfigyelt property-jét paraméterként. Ezt felváltja a `@observable` dekoráció. A felannotált property-k automatikusan megfigyelté válnak. A felépített objektum Proxy-ként működik, ha az adott megfigyelt property értéket kap vagy vált, akkor az hozzáadódik a proxy objektumhoz. A megváltozott objektum pedig a következő ciklusban elérhető is lesz, a eseményre feliratkozottak számára.

1.5. Biztonság

Ahogy a 3. fejezetben is említve van, az alkalmazás biztonságáért a Spring Security keretrendszer felelős. Ez a felhasználók hitelesítésére, és a jogosultságok ellenőrzésére van alkalmazva a projektben.

A hitelesítés egy JWT token generálás által történik meg, amely belekerül egy "jwt" nevű sütibe. A JWT token használata segít kiküszöbölni azt, hogy egy kívülálló személy valós felhasználó nevében küldjön HTTP kéréseket a szervernek. A token tartalmaz egy digitális aláírást, amely bizonyítja, hogy érvényes felhasználótól érkezik a kérés.

A jogosultság ellenőrzése egy szűrő segítségével valósul meg. A sütiből kinyert JWT token validításának tesztelése ezen keresztül történik. A különböző jogkörök elkülönítése

```
class LoginData {
    @observable username = "";
    @observable password = "";
    @observable cookie = false;
    @observable usernameCookie = "";
    @observable role = "";



    constructor() {
        makeAutoObservable(this);
        ....
    }

    ...
}
```

2. kódrészlet. A bejelentkezéshez tartozó adatokat tároló megfigyelő osztály

a `@PreAuthorize` annotáció által működik, amelyet a különböző végpontokat lekezelő függvényekre lehet elhelyezni a Controllerben.

1.6. Adatelérési réteg

Az adatelérési réteg feladata az adatbázissal való kommunikáció, az alkalmazás szervere és az adatbázis közötti adatátvitel megvalósítása mindkét irányban. A Spring Data JPA (Java Persistence API) keretrendszer van alkalmazva a projekten belül, segítségével egy absztrakciós rész képződik a Hibernate ORM (Objective Relational Mapping) keretrendszer fölé, így könnyedén végezhetőek az adatbázissal kapcsolatos CRUD (Create, Read, Update, Delete) alapműveletek. Emellett a JPA annotációkkal a modellekben lévő adattagokra is megszorításokat tudunk helyezni így biztosítva a szükséges információk helyességét. Az alábbi 3. kódrészlet, egy JPQL (Java Persistence Query Language) nyelvbeli példát mutat be, szemléltetve az összetettebb lekérdezések egyszerű implementációját.

1.7. Nemzetköziesítés

A webes alkalmazás felhasználói kiválaszthatják, hogy milyen nyelven szeretnék a felületet használni. Ezt a funkcionalitást az i18-next könyvtár segítségével valósítja meg az alkalmazás. A program legfelső részén található a `i18n.use(initReactI18next)`, amely

```
public interface SpeciesRepository extends JpaRepository<Species, Integer> {

    @Query(value = "SELECT s.nameCommon from Species s where lower
    (s.nameCommon) like lower(concat(:search, '%') )")
    public List<String> findSpeciesByNameCommonStartingWith(
        @Param("search") String search
        );
}
```

3. kódrészlet. A Species (faj) entitáshoz tartozó adatelérési interfész egy metódusa

egy i18n példányt húz be a React alkalmazásba. A React ezután a context funkciójának segítségével elérhetővé teszi ennek használatát az összes alkomponensben. Maga a fordítás a `useTranslation()` hookon keresztül történik a komponenseken belül. A hook visszatéríti a `t()`[3] függvényt, amelynek paramétere egy JSON kulcsnév lesz. Ezeket a kulcsneveket külön angol és magyar nyelvnek létrehozott fájlokban, a hozzájuk tartozó értékek követik. A `t()` függvény meghívásakor, az i18-next, a kiválasztott nyelvhez tartozó fájlból, a kulcsnevek beilleszti a megfelelő értéket. Abban az esetben, ha a kiválasztott nyelven hiányzik a fordítás, akkor az alapbeállított nyelven, jelen esetben angolul, jeleníti meg a hiányzó szövegeket.

[3] t a translate rövidítése

2. Kliens oldali technológiák

A következő alfejezetben a két kliensalkalmazás által használt közös technológiák kerülnek bemutatásra.

2.1. Közös technológiák

2.1.1. TypeScript

A TypeScript [3] a Microsoft tulajdonába tartozó, nyílt forráskódú programozási nyelv. A jól ismert JavaScript nyelv szuperszetje. Az alap funkciókat egy úgynevezett típusrendszerrel fűszerezi meg a TypeScript. Nagyméretű alkalmazások fejlesztésekor javasolt a használhata, mivel a kód írása több időt vehet igénybe, de komplexebb projektek esetén hosszútávon számos fejfájást tud megelőzni. Multi-paradigmális nyelv, a funkcionális, generikus, objektum-orientált illetve imperatív programozás paradigmái alkalmazhatóak.

A TypeScript fordító a natív kódot átalakítja, transzpilálja minden böngésző számára értelmezhető JavaScript kóddá.[36]

Mivel rendszerre jellemző a komplex objektumok kezelése, ezért a nyelv által biztosított statikus típusellenőrzés is feltárhat elrejtett hibaforrásokat kód szintjén. Kevesebb tesztelést, gyorsabb hibajavítást és jobb átláthatóságot biztosít a fejlesztőknek a TypeScript.

2.1.2. MobX

A MobX [19] egy nyílt forráskódú állapotkezelő könyvtár. Önálló rendszer, használható számos kliensoldali technológiával, mint például a React, Vue vagy Angular. Segítségével futási időben figyelhetők az adatok és azok változásai. Architektúra szempontból is egy jó választásnak minősül, mivel az állapotkezelő függvények és hozzá tartozó változók elkülöníthetőek a többi komponenstől.

A megfigyelt értékeket a konstruktor által lehet inicializálni, ellenben az alkalmazás során, mivel állapotkezelésről van szó szükség van ezeknek a változtatására is. Minden egyes metódus amely action dekorációval van felruházva, módosíthat egy vagy akár több olyan property-t is amelyek állapotai figyelve vannak. Ha a változtatás megtörtént, akkor ez a proxy megváltozását vonja magaután. Alapértelmezetten nem változtathatunk állapotot action annotált metódusokon kívül, mivel az adatok természtükből kifolyólag nem módosíthatók közvetlenül, ellenben, ha mégis megtörténik, akkor ez a változás nem fog tükröződni a komponenseken.

Azokat a komponenseket amelyek felhasználják valamelyik megfigyelt állapotot vagy tőle függően renderelnek szükséges `observer()` (pl. observer(mapDataPoints)) deklarációval ellátni. Végeredményként a komponens automatikusan újrarenderelődik amikor a megfigyelt adaton változás jön létre, de csak azoktól az adatoktól függ amelyeket ténylegesen kiolvas a megfigyelt objektum példányából.

2.2. Webes technológiák

A webalkalmazás megvalósítása Reactben történt valamint a térkép megjelenítése és a rajta történő események kezelését Leaflet segítségével oldja meg a rendszer.

2.2.1. React

A React [28] egy mostanra már 8 éves múlttal rendelkező ingyenes, nyílt-forráskódú JavaScript könyvtár. A Meta (régebb Facebook) fejleszti, rich-kliensalkalmazásokat hozhatunk létre vele, azaz a logika és a kliensoldali renderelés mind a böngészőben fut le. A JavaScript egy kiterjesztett verzióját használja, azaz a JSX-et [4], amely transzpilálás által a böngészők által értelmezhetővé válik. Deklaratív nyelv, amely a komponensek újrahasznosításán és azok állapotkarbantartásán alapszik.

Állapotkezelés: A React 16.8 [23] óta minden komponens saját állapottal, adatokkal rendelkezik. Az állapotkezelés magába foglalja a kommunikációt és adatcserét az alkalmazás komponensei között. A kiváltott események a komponensek állapotának változását eredményezhetik, abban az esetben, ha az adott komponens figyel ezekre. Egy egyszerűbb alkalmazás esetén ezen folyamatok megvalósítása nem jelent nagy feladatot, ellenben a skálázódó komponensrendszer bonyolult állapotkezelést eredményez.

Egy beépített funkciója a Reactnek a React Hookok [24]. Általuk valósítható meg az állapotok kezelése komponenseken belül. Ilyenek például a useState, useEffect, useCallback, useReducer, useContext stb.[27]

A `useState` egy olyan függvény, amely bemeneti értékként egy kezdeti állapotértéket vár el és visszatérít egy állapot és függvény párost. Az állapot, mivel immutable tulajdonságú, csak az adott függvény által módosítható.

[4] JavaScript + XML

A `useEffect` hook által megszabható, hogy bizonyos műveletek mikor hajthatók végre. Első paramétere egy függvény, amely tartalmazza a szükséges műveleteket. Második paramétere egy függőségtömb, amely tartalmazza az összes figyelt értéket. Amennyiben valamelyik állapot értéke változik, akkor a függvény (újra)végrehajtódik. Üres tömb esetén a hatásfüggvény (első paraméter), csak egyszer hívódik meg, a render ciklus után.

2.2.2. Leaflet

A Leaflet [18] egy JavaScript alapú, nyílt forráskódú könyvtár. Pehelysúlyú könyvtár, ami egy megfelelő választás lehet szinte minden fejlesztőnek, aki a mobil- vagy webalkalmazásába egy interaktív és ugyanakkor szép térképet szeretne megjeleníteni.

A React-Leaflet [25] egy absztrakciós réteg, amely a Leaflet felé épül, biztosítva ennek alap funkcionalitásait, illetve ezek kibővítését állapot folyamatok által, melyek a React paradigmáknak felelnek meg.

A React a DOM-ba egyetlen `<div />` elem beszúrásával oldja meg a térkép komponens hozzáadást. A megjelenítési logikát, karbantartást és állapotok frissítését a `MapContainer` végzi el. Létrehoz egy React Contextet a térképnek. A contex létrehozása után ha egy újabb gyerekkomponenst akarunk megjeleníteni a térképen, akkor annak nem kell külön argumentunként átadni a térkép specifikáció értékeit, hanem a komponensek tudnak majd közvetlenül is adatokat cserélni egymással.

A térkép kinézete kiválasztható a TileLayer osztály konfigurálásával. Példányosításkor ennek az osztálynak megadható egy szolgáltatás URL-t, amiről lekérheti a "csemperéteget"[5].

Az alkalmazás a megfigyeléseket Point, Polyline illetve Polygon komponensek segítségével helyezi a térképre.

2.3. Mobilos technológák

A mobilapplikáció megírása React Native-ben történt, mivel sajátosságai hasonlóak a Reactéhez így nem igényelt új nyelv és technológia ismeretet.

2.3.1. React Native

A React Native [26] egy JavaScript keretrendszer, amely alkalmas platformfüggetlen (cross-platform) mobilapplikációk fejlesztésére. Meta tulajdonába tartozó nyílt-forráskódú

[5] csempe - a kezdetkor betöltött, kis négyzetalakú térképrészlet

könyvtár, amely burkoló (wrapper) könyvtár a React fölé. A fejlesztők számára lehetőséget ad, natív illetve platform specifikus komponensek deklarálására is. Működése hasonló a Reactéhez, kivéve, hogy DOM manipulálása helyett, a React Native egy háttér folyamatként fut az eszközünkön.

2.3.2. Expo

Az Expo [9] egyben keretrendszer és platform, rengeteg funkcionalitást biztosít, amellyel felgyorsul a fejlesztés.

React Native-al használva az esetek többségében a natív fejlesztés nehézségeitől kíméli meg a fejlesztőket az Expo. Használata abban az esetben ajánlott, ha a mobilos app nem igényel különösebb alacsony szintű, OS specifikus implementációt, amit csak natívan lehet megoldani. Ezek olyan különleges esetek (pl. hardware eszközök használata, kommunikáció más alkamlazással stb.), amelyek nem jellemzőek a jelen projekt esetén.

Megoldást nyújt a eszközhiány vagy nem megfelelő OS korlátozásokra, lehetővé téve a független fejlesztés lehetőségét. Szükség szerint iOS és/vagy Android simulátor is indítható bármilyen környezetben.

Az Expo világa annyira kiterjedt, hogy bármilyen React Native külső könyvtárnak megvan az Expo kompatibilis megfeleltetése.

3. Szerver oldali technológiák

A Madármegfigyelő alkalmazás egy Spring [31] keretrendszerre épül, amely egy átfogó infrastruktúrát nyújt a Java alkalmazások fejlesztésére. A Spring alapelve az Inversion of Control [16] és a Dependency Injection [7].

3.1. Spring Boot

A Spring Boot [32] [4] lehetővé teszi, hogy könnyen és egyszerűen lehessen egy már bekonfigurált Spring alapú alkalmazást létrehozni, majd futtatni. Automatikusan behúzza a szükséges függőségeket, és alapértelmezett értékeket ad a konfigurációnak.

3.2. Spring Data JPA

A Spring Data JPA [33] abban segít, hogy egy Spring alapú alkalmazás adatelérési rétege egyszerű és könnyen implementálható legyen. A saját DAO implementációk megírását ki lehet küszöbölni, előredefiniált interfész kiterjesztésével, amely a CRUD műveleteket implementálja a modell és ennek egyedi azonosítójának típusa alapján. Bonyolultabb adatlekérések esetén több lehetőség van: meg lehet adni egy függvénydeklarációt, amelynek neve, paraméterei és visszatérített típusa, ha megfelel a JPA konvencióinak, akkor annak implementációja automatikusan legenerálódik. Egy komplexebb művelet függvénye `@Query` annotációval felruházva testreszabhatóvá tehető. Az annotáció paramétere ezesetben egy JPQL-ben megírt lekérés, amely végrehajtódik a háttérben a függvény meghívásakor.

```
Optional<User> findUserByUsername(String username)
Boolean existsByUsername(String username)
```

3.3. Spring Web MVC

A Spring Web MVC[35] feladata, hogy a kliensektől beérkező HTTP kéréseket feldolgozza, majd a választ megfelelően felépítse annotációk segítségével.

Minden REST API végponthoz tartozik egy neki megfelelő HTTP metódus és útvonal, valamint egy függvény, amely az adott műveletet lekezeli. A Spring Web MVC különböző annotációkat szolgáltat, amelyekkel meg lehet határozni az előbbieket. A controller osztályra rakott `@RequestMapping` annotáció az adott osztályban felsorolt összes

végpontra meghatározza a HTTP kérés útvonalának előtagját, prefixumát. A `@GetMapping`, `@PostMapping`, `@PatchMapping` és @DeleteMapping annotációkkal lehet paraméterként a felépítendő URI további részét definiálni.

Ezen kívül, a osztálymetódus fejlécben lévő paraméterek szintén felannotálhatóak. Az annotáció típusától függően meghatározódik, hogy az adott paraméter értéke honnan lesz kinyerve. Az érték része lehet a kérés törzsének, lehet egyszerű paraméterérték vagy egy útvonalba ágyazott paraméter.

3.4. Spring Security

Hitelesítésre és jogosultság-ellenőrzésre a Spring Security [34] keretrendszer biztosít megoldást. A hitelesítés folyamata, a felhasználó bejelentkezési adatairól dönti el, hogy azok helyesen vannak-e megadva. Az autorizáció során egy, már valid felhasználóról kell eldönteni, hogy milyen jogosultsága van, vagyis milyen adatokhoz, funkcionalitásokhoz férhet hozzá. Abban az esetben, ha a felhasználó helyes adatokkal jelentkezik be, a Spring Security generál egy JWT-t, amely tartalmazza a felhasználónevet és a hozzá tartozó jogosultságot, amely a későbbiekben az autorizációra lesz használva. Ez a token, a kliensoldalon, egy generált, "jwt" elnevezésű sütiben kerül megjegyzésre. A JWT működése a 1.5. fejezetben kerül részletes magyarázatra.

Minden kérés előtt, egy saját szűrő hívódik meg, amely megpróbálja kinyerni a JWT-t, majd, ha sikerül, ennek használatával történik meg a validálás. A jogosultság ellenőrzése minden végpontnál a `@PreAuthorize` annotáció használatával történik. Paraméterként egy szerepkör nevét kapja meg, amely jelzi hogy mely szerepkörrel rendelkező felhasználók férhetnek hozzá az adott végponthoz vagy Controller osztályhoz.

Pl. `@PreAuthorize("hasAuthority('ADMIN') or hasAuthority('SCIENTIST')")`

3.5. Spring Validation

A 'spring-boot-starter-validation' [37] függőség feladata az API végpontokba beérkező adatok ellenőrzése. A DTO osztályokban a kívánt mezőkre ráhelyezetők bizonyos annotációk, amelyek az adattagok értékének helyességét ellenőrzik. A `@NotNull` és `@NotEmpty` annotációk figyelmeztet, hogy mely mezők nem lehetnek üresek, a `@Unique` jelzi, hogy az adott adattag értékei egyediek kell legyen az adatbázisban, a `@DateTimeFormat()` meg azt jelöli, hogy az elvárt érték egy dátum vagy időtípus kell legyen. A Controller rétegben fog végbemenni

a validálás a `@Valid` annotáció segítségével. Az annotáció jelenléte miatt a DTO-ban meghatározott validálási szabályok automatikusan kiértékelődnek, manuális ellenőrzés nélkül.

3.6. Hibernate Spatial

A projekt egyik legfontosabb komponense a térkép, és a rajta megjelenített geometriai elemek. Minden megfigyelés lementésekor van egy mező, amely a hely koordinátáit jegyzi meg. Ezen földrajzi pontok feldolgozása a Hibernate egy kiterjesztése, a Hibernate Spatial [15] segítségével történik meg. A Hibernate Spatial megengedi ponttal, vonallal és sokszöggel való műveletek elvégzését. Erre két könyvtárat is biztosít: a Geolatte-geom-t [10], és a JTS Topology Suite-t [17]. A projekten belül az utóbbi könyvtár került használatra, főként az általa biztosított Geometry típus miatt. A `Geometry` három geometriai elemet támogat: Pontokat (Point), több pont által meghatározott szakaszokat (LineString), és Sokszögeket (Polygon). A `geometry` mező értéke a jts `WKTReader` metódusának meghívásával jön létre. A függvény paraméterként kap egy WKT formájú karakterláncot, ahol a WKT [38] [6] egy geometriai adatok ábrázolására használt jelölőnyelv.

3.7. PostgreSQL

A projekt PostgreSQL [22] relációs adatbázissal müködik. Ennek létezik egy népszerű kiterjesztése, a postgis [21], ami geográfiai objektumok tárolását, és helyhez kapcsolódó beépített függvénykönyvtárat biztosít. Az alkalmazásban két ilyen függvény szerepel: a `ST_DWithin` [2], amely ellenőrzi, hogy egy megadott távolságon belül vannak-e a geometriai objektumok, és a `ST_Contains` [1], amely ellenőrzi, hogy az első objektum által leírt geometriai elem teljesen a második objektum által leírt sokszög határain belül van-e.

[6] Well Known Text

4. Eszközök és módszerek

Az alkalmazás fejlesztése során a csapat a Scrum módszereit használta, amelynek köszönhetően gördülékeny inkrementális és iteratív fejlesztés valósult meg. Emellett számos elterjedt eszköz van felhasználva a hatékonyság, pontosság megőrzésére, ilyen például a Git a verziókövetésre, Gradle a szerveroldalhoz illetve Npm a kliensoldalhoz tartozó függőségek karbantartásához vagy a statikus kódelemzők kliens- illetve szerveroldalon. A folytonos integráció (Continuous Integration - CI) elvének követése is jelen van az alkalmazás fejlesztése során. A következő fejezetek részletesebben ismertetik ezeket a módszereket és eszközöket.

4.1. Projektmenedzsment

A Scrum [39] egy inkrementális és iteratív Agile szoftverfejlesztési stratégia, amely nagyban elősegítette a csapat tagjai közötti kommunikációt és a munka naprakészen tartását. A Scrum alapjaiban sprinteket használ, azaz a több hónapos, akár éves fejlesztési folyamatokat felosztja kisebb, de emiatt jobban behatárolható intervallumokra. Minden ilyen sprintet a csapat egy tervezéssel kezdte és demózással, valamint egy kiértékelővel zárta be. A szakgyakorlat idején kéthetes iterációk voltak, mindennapos megbeszélésekkel a tagok között. Ezeket az iterációkat, sprinteket egy backlog elemzés előzte meg. Ekkor a csapat eldöntötte hogy a termék backlogjából az előre meghatározott, user story listából, melyek lesznek megvalósítva a következő sprintben. Kéthetente, majd a csoportos projekt tantárgy keretein belül háromhetente sor került a sprint lezárására, ekkor kiértékelés, retrospektíva és demozás történt. A sikeresen megoldott user storyk bemutatásra kerültek, a csapattagok kielemezték a sprint sikeres/sikertelen voltát, a sikertelenség okait megvitatták és ezek függvényében tervezték meg a következő sprint backlogot.

4.2. Verziókövetés

Az egyik legismertebb osztott verziókövető rendszer a Git [11], amely biztosította az egyidejű fejlesztést és az alkalmazás verzióinak nyomonkövetését. A Madármegfigyelő alkalmazás forráskód menedzsmentje is ennek segítségével valósult meg. Saját git tárolója van a szerver, web és mobil kódbázisainak, így bármely régebbi verzió elérhető és szükség esetén visszaállítható. A GitFlow [12] módszert alkalmazva minden user-storynak egy külön fejlesztői ág van létrehozva, ezáltal is csökkentve a konfilktusok számát, különböző részek

összeillesztésénél. Egy funkcionalitás-fejlesztés befejezése után sor került, a mentorok által az átvizsgálásra. Ezután hibajavítás, valamint optimalizálás elvégeztével a funkcionalitás bekerült a főfejlesztői ágra. Ez útobbi feladatokhoz szükséges egy projektmenedzsment kezelő rendszer amelyen keresztül végrehajtató a kommunikáció, és feladatok nyilvántartása, hogy ki mivel foglalkozik, egyidőben és mindenki számára elérhetően. A GitLab [13] sok más előnye mellett, alkalmas felületet biztosított a projekt fejlesztése során a csapatnak, hogy átlátható és folyamatosan integrált, naprakész kódbázis legyen. Ezt segíti a Gitlab által biztosított pipeline rendszer, amely minden kódfeltöltésnél lefuttatja a `.gitlab-ci.yml`-ba előre bekonfigurált programok sorozatát. A pipeline több fázisból(stage) áll, ahol minden stage egy vagy több job-t tartalmaz. A projektben a pipeline főképp kódellenőrzésre van használva, erre két stage meghatározva: a checkstyle, amely a statikus kódelemzést, és a build, amely a kód lefordítását végzi el.

4.3. Függőségkezelés

A függőségek karbantartása illetve a rendszer automatikus build-elése kliens- és szerveroldalon különböző segédeszközök által van végrehajtva. A szerveroldali függőségek menedzsmentjét a Gradle végzi. A kód írása IntelliJ Ideaban történik, amely támogatja a Gradle scriptek használatát. A webes illetve mobil modulok függőségeit az Npm (Node package manager) kezeli.

4.4. Statikus kódelemzés

A kódbázist naprakészen tartani hibák nélkül elengedhetetlen egy komplexebb és több személyből álló fejlesztői csapatnál. A statikus kódelemzők remek eszközök univerzális kódírási konvenciók betartására. A kliensoldalon ESLint [8] van használva. Az ESLint egy gyors statikus JavaScript kódelemző, nagyban testreszabható. Számos olyan hibát képes észrevenni a kód írása közben, amelyek az emberi figyelmetlenség, türelmetlenség miatt kerülnek bele a kódba. Ilyenek például a kódírási stílustól való eltérések, változónevek, amelyek nem felelnek meg a konvecióknak, illetve lehetséges jövőbeli hibák megelőzése érdekében is jelez az ESLint, az adott kódrészletnél. Ezekre a hibákra az ESLint általában egy automatikus javítást is el tud végezni.

A szerveroldali két statikus kódelemző, a Checkstyle [5] biztosítja a kódrészek egységes stílusát, a FindBugs [30] pedig megpróbálja előrevetíteni a futás közbeni hibákat.

5. A projekt működése

A alábbi fejezet a projekt müködését írja le, valamint szemlélteti néhány képernyőfotó által. A többnyelvűség szemléltese miatt az ábrák különböző nyelven vannak beszúrva.

A webes felület megnyitásakor egy teljes képernyőn lévő térkép jelenik meg (2. ábra), fölötte egy navigáló gombokat tartalmazó sávval, amelyen a vendég felhasználó funkcionalitásai érhetőek el.

Az alkalmazás alapértelmezetten angol nyelven indul el, de ez könnyen megváltoztatható a jobb felső sarokban elhelyezett kis földgömb ikonra kattintva. Ez a művelet az alkalmazáson belül bármikor elérhető. Angol nyelv mellett, jelenleg a magyar nyelvet támogatja a weboldal.

A képernyő jobb felső sarkában található Login (Bejelentkezés) gombra kattintva előugrik egy ablak, amelyben a felhasználó e-mail cím és helyes jelszó megadásával bejelentkezhet a rendszerbe. A térkép jobb felső sarkában lévő ikonnal kiválasztható a térkép kinézete, színes illetve fekete-fehér opciók közül.

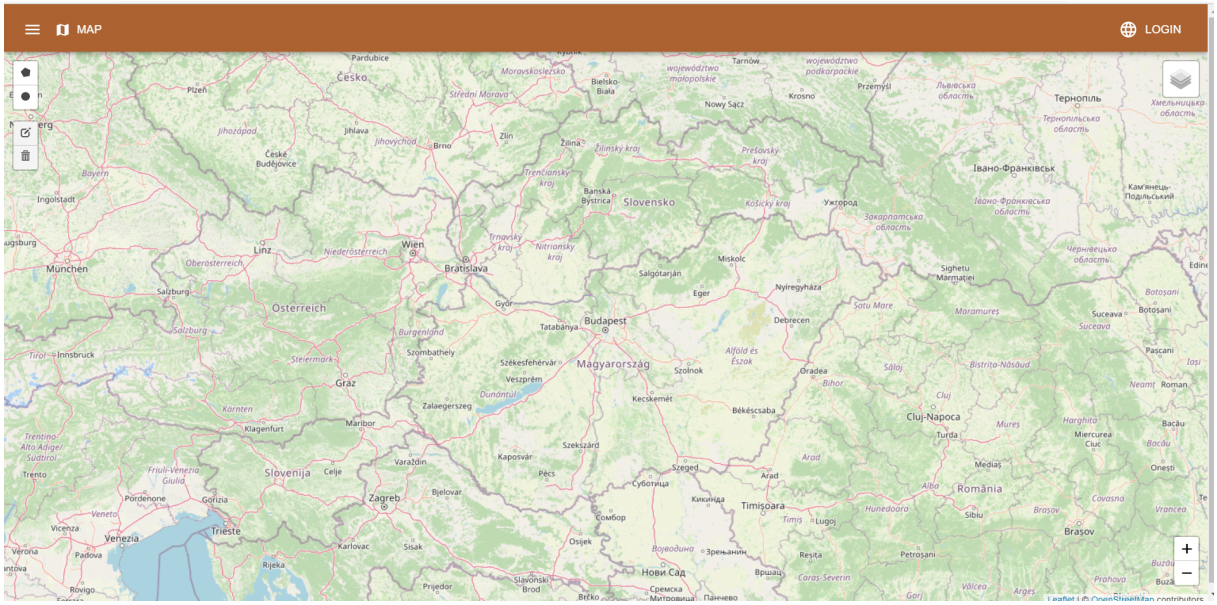
Bejelentkezés után, a gomb helyett egy avatár ikon jelenik meg, amely a jelenlegi alkalmazásban a felhasználó nevének kezdőbetűjét ábrázolja. Erre kattintva a felhasználó ki tud jelentkezni a fiókjából. Sikeres bejelentkezéskor ezenkívül még megváltozik a fejléc tartalma a felhasználó szerepkörétől függően.

A fejléc bal oldalán a hamburger ikonra kattintva balról előhozható egy "fiók"[7] ablak, amely az alkalmazás főfunkcionalitását tartalmazza. Itt találhatóak a szűrési lehetőségek, illetve a megfigyeléseket lekérő gomb.

A fiók bezárt állapotakor, egy újabb gombsor tárul fel a felhasználó számára, amely a térkép bal felső sarkában található. Ezek által tud terület szerinti szűrést végrehajtani. A poligon ikonra kattintva, aktiválódik a rajzoló felület. A térképre kattintva a felhasználó behatárolja a kívánt felületet, ahol minden egyes kattintás a poligon következő csúcsát jelöli. A rajzolás befejeződik, amikor a legelső csúcspont lesz megkattintva. Ez után a meghatározott poligon átlátszó rétegként befedi a térkép adott részét. (Lásd 4. ábra) Abban az esetben, ha a kirajzolt terület nem megfelelő, a felhasználó újrakezdheti az ábra törlésével. Ezután a fehasználó a hamburger ikonnal előhozza a szűrő panelt, amely tartalmazza a "Terület szűrés" címkéjű gombot. Az adatok töltődése közben egy betöltést jelző elem jelenik meg a képernyőn, amely biztosítja a felhasználót az alkalmazás helyes működéséről.

A terület szerinti szűrés mellett más lehetőségek is fel vannak sorakoztatva a szűrő panelen.

[7] Navigációs drawer



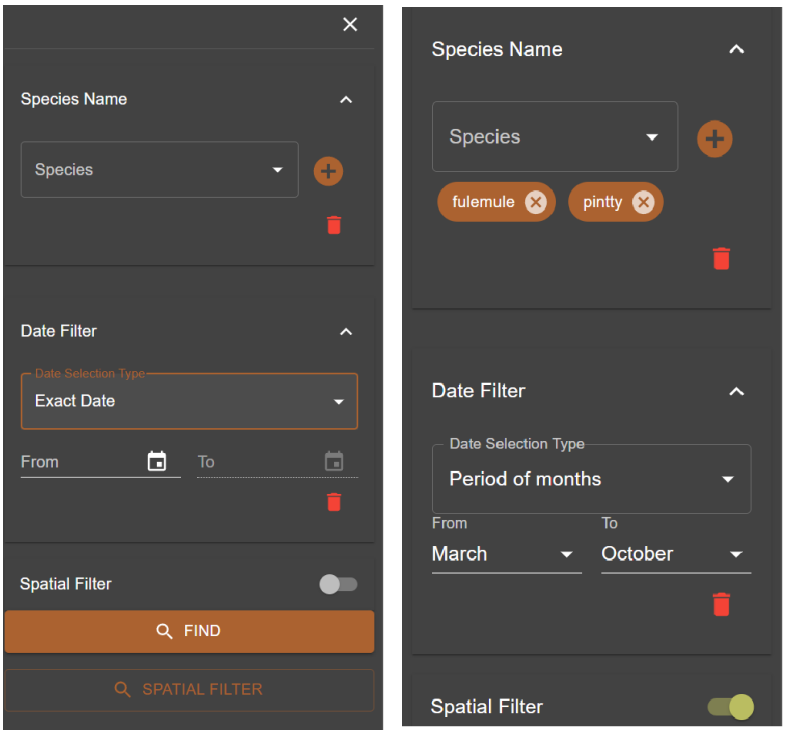
3. ábra. Főoldal, vendég felhasználóként

A panel tetején található az első szűrési opció, amely fajnevek szerint szűr. A felhasználó az adott szövegdobozba írhatja a fajneveket, majd a mellette található plusz gombbal adhatja hozzá a szűréshez. Ha a szövegdobozban három vagy több betű van begépelve, akkor az automatikus kiegészítés funkció egy legördülő listában felsorolja a lehetséges madárfajokat, amelyek nevei találnak az addig megadott szöveggel. A szűrőhöz hozzáadott fajok kicsi ovális csipeken jelennek meg a szövegdoboz alatt, amelyre kattintva ki is lehet törölni őket.

A következő szűrési elem a dátum szerinti szűrés. Az alkalmazás három különböző lehetőséget biztosít a dátum szerinti keresésre, amelyek közül egy legördülő listában válogathatunk. Az első dátum szerinti szűrés opció megengedi a felhasználónak, hogy pontos dátum szerint keresse a megfigyeléseket. Emellett lekérhetőek egy adott időintervallumban történt megfigyelések is, ekkor két dátumot kell kiválaszunk az előugró naptárból. A harmadik dátum szerinti szűrés opcióhoz két hónapot kell megadnia a felhasználónak és ebben az esetben az összes olyan megfigyelés válik láthatová, amelyeket azon hónapok között vittek fel bármelyik évben.

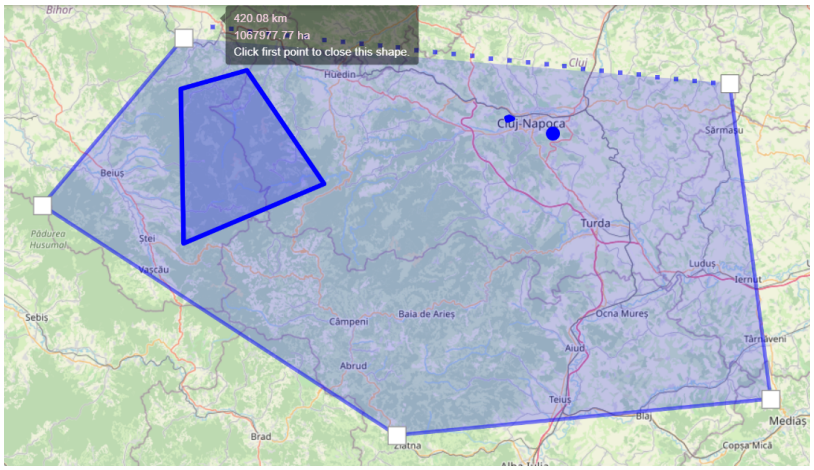
Mindkét szűrésopciónak a jobb alsó sarkában található egy vörös szemeteskuka ikon, rákattintásával az adott szűrőhöz tartozó beállítások véglegesen törlődnek.

Azok után, hogy a felhasználó beállított minden általa kívánt szűrőt, és lekérte a megfigyeléseket, a térképen megjelennek az adatok. Egy megfigyelés lehet pontszerű, ha egy adott koordinátához köthető egy egyed megfigyelése. Emelett lehet vonalszerű vagy sokszögű koordináta adatcsoport, ha a megfigyelés egy adott útvonalon/területen történik. Ez utóbbi típus



(a) Alapértelmezetten a szűrési opciók értékei üresek

(b) Példa egy komplexebb szűrés beállításra



(c) Poligon rajzolás

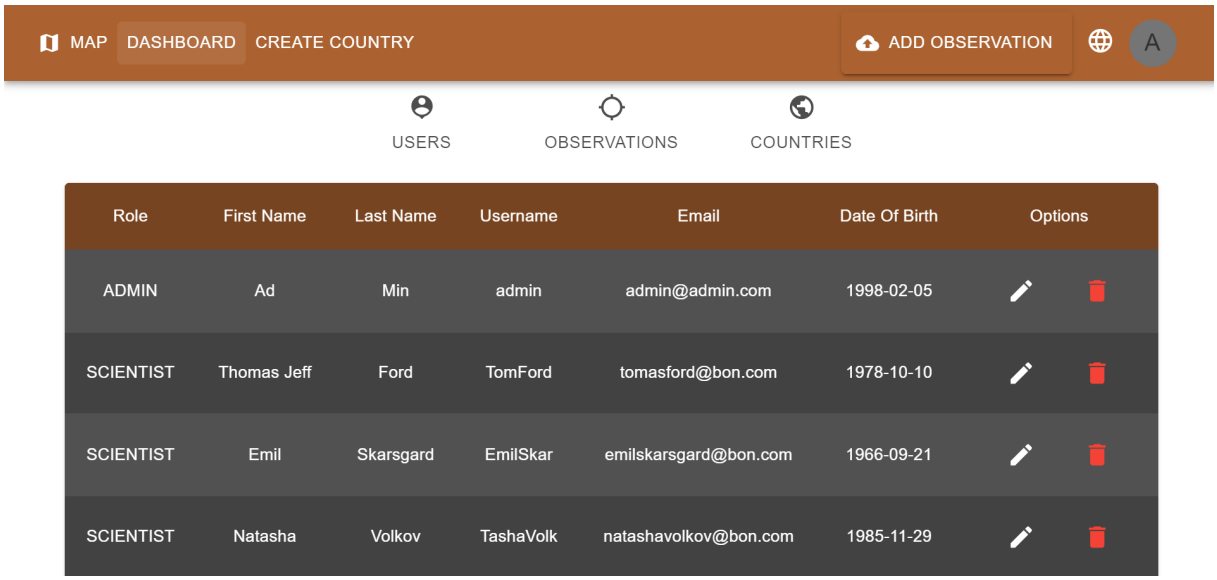
4. ábra. Szűrési opciók beállításának szemléltetése

a megfigyelt egyed követésével, vagy többszöri észlelésének következtében jön létre.

A 4. ábra tartalma egy szűrés lépéseit szemlélteti.

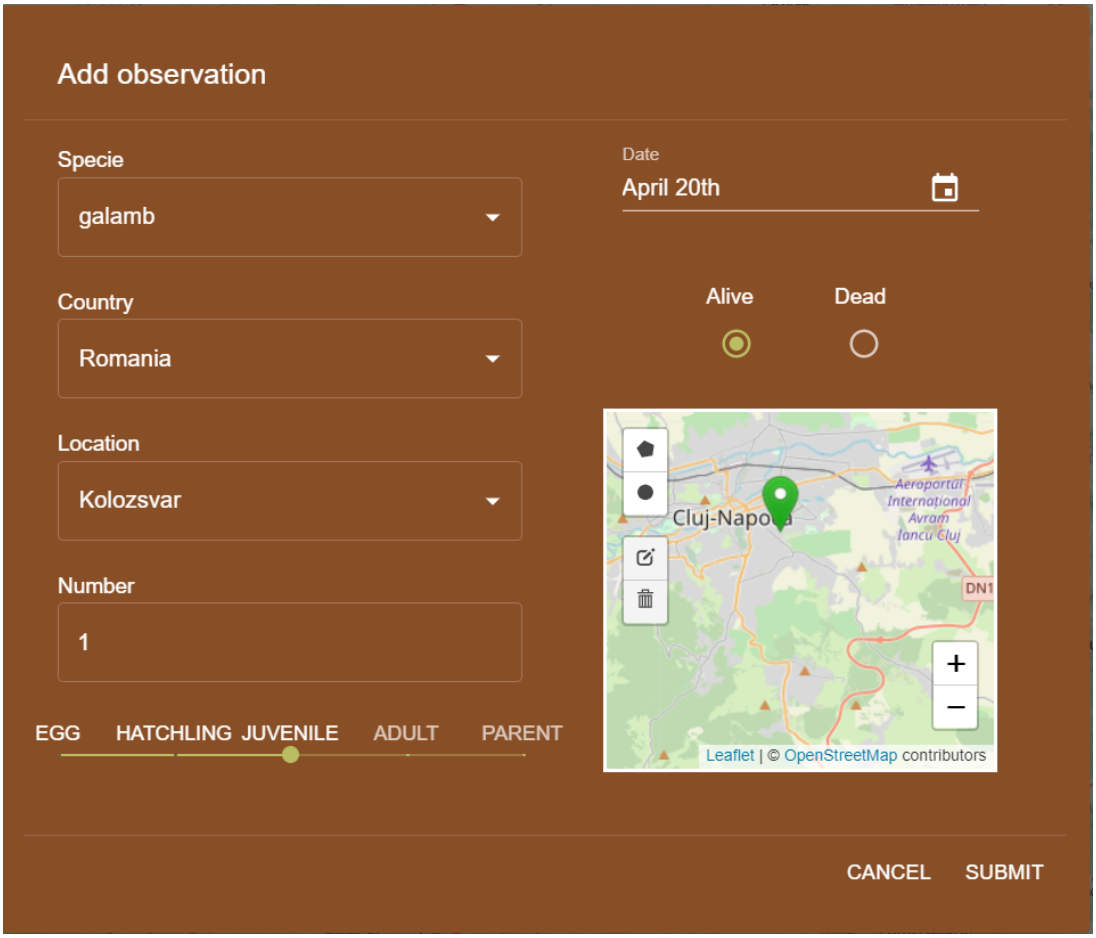
Az eddig említett funkcionalitások elvégzésére mindhárom szerepkör képes, viszont az adminisztrátor és tudós szerepkörök ezen kívűl más funkcionalitásokat is elérnek.

Az adminisztrátor végzi az alkalmazás adatainak a karbantartását, menedzselését. Saját vezérlőpulttal rendelkezik, amin belül külön ablakokból választhat, a felhasználók, megfigyelések és országok adatai közül. Bármelyik adathalmazt megtekintheti egy táblázat



5. ábra. Az admin vezérlőpultja

formájaban. A táblázat utolsó oszlopában a ceruza és szemetes ikonra kattintva lehetősége van a felhasználónak módosítani, illetve törölni őket (lásd 5. ábra).



6. ábra. Új megfigyelést felvivő dialógus

Egy tudós szerepkörrel rendelkező felhasználó szintén rendelkezik hasonló vezérlőpulttal, csak ő a saját adatait listázhatja ki, valamint csak a saját megfigyeléseit tudja modósítani, törölni.

Adminisztrátor és tudós szerepkörön belül elérhető a megfigyelés hozzáadása funkció. A gombra kattintás után megjelenik egy dialógus a szükséges információkat tartalmazó űrlappal. A kis térkép a megfigyelés helyének meghatározására alkalamas. A marker elhelyezése egy pontos koordináta kijelölését jelenti. Ez abban az esetben alkalmazandó amennyiben egy madárt egy pontos helyszínen figyeltünk meg. Ha egy madárcsoportot vagy egy madarat több helyszínen figyelt meg a tudós/tudóscsoport, alkalmazható a polygon rajzolás ennek pontos koordinátáinak kijelöléséhez. A többi kitöltendő mezőt automatikus kiegészítés segíti elő, valamint azonnali hibajelzés válik láthatóvá, amennyiben a felhasználó nem megfelelő adattal próbálja kitölteni azokat. Egy új megfigyelés felvitele után az egyből látható lesz a térképen a többi felhasználó számára is (lásd 6. ábra).

Következtetések és továbbfejlesztési lehetőségek

A Madármegfigyelő alkalmazás fejlesztése során, egy olyan alkalmazás jött létre, amely lehetővé teszi a madarak iránt érdeklődő személyek számára, hogy egy egységes felületen kövessék ezek megfigyeléseit.

A webes felületen a felhasználó lekérheti az eddig archivált megfigyeléseket akár szűrők használatával is. Ezen kívűl, még lehetőséget ad a tudósoknak arra, hogy új adatokat vigyenek fel és az általuk korábban felvitt megfigyeléseiket karbantartsák.

Az mobilos felületen az mindennapi felhasználó értesítéseken keresztül kapja meg a körülötte lévő korábbi megfigyelésekkel kapcsolatos információkat.

A projekt bemutatása után még előnyös lehet újabb funkcionalitásokat megvalósítani, hogy további lehetőségeket adjanak az aplikáció használatára.

A mobilos felhasználóknak nagy segítség lenne, ha minden megfigyeléshez tartozna egy kép, vagy egy rövid leírás, azért, hogy megkönnyítse a madárlest. Ezen kívűl a tudósoknak előnyös lenne, ha lenne egy nekik szabott mobilos felület is, ahol adatlekérésen kívűl képesek megfigyeléseket felvinni az archívumba, amint a megfigyelés megtörtént.

Egy funkcionalitás, amely a tudósok számára nagyon hasznos lehet, az a madár populáció vándorlásának megfigyelése, és egy faj népességének változásának a felügyelése. Ennek egy kivitelezési lehetősége egy grafikus reprezentáció, amely adott időintervallumonként csoportosított adatokat térképen jelenítene meg. Az intervallumok közti animált váltás kimutatná a megfigyelések közti különbségeket. Ennek egy kivitelezési lehetősége egy grafikus reprezentáció, amely adott időintervallumonként csoportosított adatokat térképen jelenítene meg. Az intervallumok közti animált váltás kimutatná a megfigyelések közti különbségeket.

Hivatkozások

[1] $ST_Contains$. URL: $https://postgis.net/docs/ST_Contains.html$ (elérés dátuma: 2022. ápr. 8.)

[2] $ST_DWithin$. URL: $https://postgis.net/docs/ST_DWithin.html$ (elérés dátuma: 2022. ápr. 8.)

[3] $WhatisTypeScript?$. URL: $https://www.typescriptlang.org/why-create-typescript$ (elérés dátuma: 2022. ápr. 21.)

[4] *A Comparison Between Spring and Spring Boot*. URL: https://www.baeldung.com/spring-vs-spring-boot#spring-boot (elérés dátuma: 2022. ápr. 5.)

[5] *CheckStyle*. URL: https://checkstyle.sourceforge.io (elérés dátuma: 2022. ápr. 17.)

[6] *Data Transfer Object*. URL: https://www.baeldung.com/java-dto-pattern (elérés dátuma: 2022. ápr. 12.)

[7] *Dependency Injection*. URL: $https://en.wikipedia.org/wiki/Dependency_injection$ (elérés dátuma: 2022. ápr. 24.)

[8] *Eslint*. URL: https://eslint.org (elérés dátuma: 2022. ápr. 17.)

[9] *Expo For React Native*. URL: https://upstack.co/knowledge/should-i-use-expo-for-react-native (elérés dátuma: 2022. ápr. 24.)

[10] *Geolatte-Geom*. URL: https://javadoc.io/doc/org.geolatte/geolatte-geom/latest/index.html (elérés dátuma: 2022. ápr. 24.)

[11] *Git*. URL: https://git-scm.com (elérés dátuma: 2022. ápr. 17.)

[12] *GitFlow*. URL: https://www.atlassian.com/git/tutorials/comparing-workflows/gitflow-workflow#:~:text=Gitflow%20is%20a%20legacy%20Git,software%20development%20and%20DevOps%20practices (elérés dátuma: 2022. ápr. 24.)

[13] *GitLab*. URL: https://gitlab.com (elérés dátuma: 2022. ápr. 17.)

[14] *Hibernate ORM*. URL: https://hibernate.org/orm/ (elérés dátuma: 2022. ápr. 22.)

[15] *Hibernate Spatial*. URL: https://www.baeldung.com/hibernate-spatial (elérés dátuma: 2022. ápr. 5.)

[16] *Inversion of Control*. URL: $https://en.wikipedia.org/wiki/Inversion_of_control$ (elérés dátuma: 2022. ápr. 24.)

[17] *JTS Topology Suite*. URL: https://locationtech.github.io/jts/javadoc/ (elérés dátuma: 2022. ápr. 24.)

[18] *Leaflet*. URL: https://leafletjs.com (elérés dátuma: 2022. ápr. 22.)

[19] *Mobx*. URL: https://mobx.js.org/README.html (elérés dátuma: 2022. ápr. 20.)

[20] *OpenBirdMaps Documentation*. URL: http://openbiomaps.org/documents/ (elérés dátuma: 2022. ápr. 22.)

[21] *Postgis*. URL: https://postgis.net/ (elérés dátuma: 2022. ápr. 8.)

[22] *PostgreSQL*. URL: https://www.postgresql.org/about/ (elérés dátuma: 2022. ápr. 5.)

[23] *React*. URL: https://reactjs.org (elérés dátuma: 2022. ápr. 21.)

[24] *React Hooks*. URL: https://reactjs.org/docs/hooks-reference.html (elérés dátuma: 2022. ápr. 24.)

[25] *React Leaflet*. URL: https://react-leaflet.js.org/docs/start-introduction/ (elérés dátuma: 2022. ápr. 22.)

[26] *React Native*. URL: https://reactnative.dev (elérés dátuma: 2022. ápr. 21.)

[27] *React State Management*. URL: https://dev.to/workshub/state-management-battle-in-react-2021-hooks-redux-and-recoil-2am0 (elérés dátuma: 2022. ápr. 23.)

[28] *React Wikipedia*. URL: https://en.wikipedia.org/wiki/React_(JavaScript_library) (elérés dátuma: 2022. ápr. 21.)

[29] *Representational state transfer*. URL: $https://en.wikipedia.org/wiki/Representational_state_transfer$ (elérés dátuma: 2022. ápr. 24.)

[30] *SpotBugs*. URL: https://spotbugs.github.io (elérés dátuma: 2022. ápr. 17.)

[31] *Spring*. URL: $https://docs.spring.io/spring-framework/docs/3.2.x/spring-framework-reference/html/overview.html$ (elérés dátuma: 2022. ápr. 24.)

[32] *Spring Boot*. URL: https://spring.io/projects/spring-boot (elérés dátuma: 2022. ápr. 5.)

[33] *Spring Data JPA*. URL: https://spring.io/projects/spring-data-jpa (elérés dátuma: 2022. ápr. 5.)

[34] *Spring Security*. URL: https://spring.io/projects/spring-security (elérés dátuma: 2022. ápr. 5.)

[35] *Spring Web MVC*. URL: $https://www.tutorialspoint.com/spring/spring_web_mvc_framework.htm$ (elérés dátuma: 2022. ápr. 11.)

[36] *Transpiler vs Compiler*. URL: $https://howtodoinjava.com/typescript/transpiler-vs-compiler/$ (elérés dátuma: 2022. ápr. 21.)

[37] Alejandro Ugarte. *Spring Boot Bean Validation*. URL: https://www.baeldung.com/spring-boot-bean-validation (elérés dátuma: 2022. ápr. 5.)

[38] *Well Known Text*. URL: $https://en.wikipedia.org/wiki/Well-known_text_representation_of_geometry$ (elérés dátuma: 2022. ápr. 4.)

[39] *What is scrum*. URL: https://www.scrum.org/resources/what-is-scrum (elérés dátuma: 2022. ápr. 17.)