KEHOP-4.3.0-VEKOP-15-2021-00003

A HAZAI BIOLÓGIAI SOKFÉLESÉG ÁLLAPOTÁNAK KORSZERŰ MÓDSZEREKKEL TÖRTÉNŐ NYOMON KÖVETÉSÉHEZ SZÜKSÉGES ADATBÁZISOK, MÓDSZERTANOK ÉS INFORMÁCIÓS RENDSZEREK INTEGRÁLT FEJLESZTÉSE (PROJEKT ELŐKÉSZÍTÉS)

Magyarország Kormánya
Európai Unió
Európai Regionális Fejlesztési Alap
BEFEKTETÉS A JÖVŐBE
SZÉCHENYI 2020

A HAZAI BIOLÓGIAI SOKFÉLESÉG ÁLLAPOTÁNAK KORSZERŰ MÓDSZEREKKEL TÖRTÉNŐ NYOMON KÖVETÉSÉHEZ SZÜKSÉGES ADATBÁZISOK, ADATKEZELÉSI MÓDSZERTANOK ÉS INFORMÁCIÓS RENDSZEREK FEJLESZTÉSI LEHETŐSÉGEINEK AZONOSÍTÁSA, FEJLESZTÉSI ALTERNATÍVÁK ÖSSZEHASONLÍTÓ ELEMZÉSE

KEDVEZMÉNYEZETT: AGRÁRMINISZTÉRIUM (AM)

Debrecen,

2022. március 16.

Dr. Bán Miklós, Gáspár Ákos

1. Bevezetés

A tanulmány célja egy olyan műszaki dokumentum elkészítése, amely elősegíti Magyarország természeti állapotának átfogó értékelése – tényleges, terepen gyűjtött, minősített adatokon alapuló informatikai rendszer fejlesztési irányának és módjának meghatározását. A megvalósítandó feladat leírása szerint az elkészített rendszernek alkalmasnak kell lennie komplex elemzések végzésének támogatására, különféle adatforrások problémamentes importálására, változatos adatkapcsolati felületek kiszolgálására saját és integrált felhasználói felületeken keresztül elsősorban biológiai háttér ismeretekkel rendelkező szakemberek számára. Továbbá a rendszernek technikai szempontból követnie kell a recens informatikai trendeket (a jelen és jövőbeli kompatibilitási igények érdekében) figyelembe véve fenntarthatóságot is.

A megvalósítandó feladat leírása szerint a rendszer elsődleges célja egy biodiverzitási adatközpont létrehozása, magas konnektivitással, beépített elemző eszközökkel és elemzés támogatással. Beépített térképi megjelenítő eszköz készlettel és alacsony szintű *GIS* támogatással. A rendszernek több száz felhasználó kiszolgálására kell felkészülnie és ehhez mérten a rendszer használatának és a kezelt adatok hozzáférésének megfelelő szabályozására.

A megfogalmazott célok alapján számos kérdés merül fel, amelyek megválaszolásához a dokumentumban alternatív javaslatokat fogalmazunk meg.

Véleményünk szerint az egyik legalapvetőbb kérdés a fejlesztési irány megválasztása kapcsán a fejlesztési és üzemeltetési ráfordítás tervezett aránya. Ugyanis minél rugalmasabb a rendszer, annál magasabb a fejlesztési és fenntartási költsége is. Például egy komplex és rugalmas rendszer használata egészen biztosan több tanulást igényel és gyakoribb támogatást mint egy meghatározott cél feladatokat kiszolgáló rendszeré. A tanulmány összefoglalásában ezeket a szempontokat is figyelembe vevő javaslatokat fogalmazunk meg.

A rendszer összetevő egyes komponensek tervezése során is különféle prioritásai szempontokat lehet figyelembe venni. Ezek a prioritások lehetnek a rendelkezésre állás, emberi erőforrások, fejlesztési költség, megbízhatóság, fenntartási költség. Ezeket a szempontokat minden egyes a dokumentumban részletezett komponens kialakítás javaslatának megfogalmazásakor figyelembe vettük.

A biodiverzitási adatok kezelése az utóbbi évek során nemzetközi és világszerte nemzeti szinten is sok figyelmet kapott tekintettel arra, hogy természetvédelem érdekeinek érvényesítése a növekvő ipari és mezőgazdasági nyomás hatására egyre nehezebb feladat mindenhol. Valós adatokon és érthető értelmezéseken alapuló érveléssel képes csak a természetvédelem az általa képviselt hosszú távú érdekeket érvényesíteni a rövid távú érdekekkel szemben. A döntéshozási folyamatok is felgyorsultak amelynek következménye, hogy az adatoknak gyorsan és egyszerűen elérhetőnek kell lenniük, amelyet csak jól átgondolt adat életút folyamatok mentén lehet megvalósítani az adatok gyűjtésétől kezdve az adatok rendszerezéséig és értelmezéséig.

A nyílt tudomány és a F.A.I.R. alapelvek (Findable, Accessible, Interoperable, Reusable) terjedése kifejezetten kedvező a biodiverzitási adatok kezelésének fejlődése szempontjából. Nemzetközi szinten alapvető elvárásként létezik, hogy a biodiverzitási adatok legyenek nyílt hozzáférésűek és sztenderd eszközökkel kezelhetőek. Ennek az irányvonalnak a legjelentősebb nemzetközi képviselője és megvalósítója a *GBIF* amely 2001 óta létezik és az online publikus adatbázisa jelenleg több mint kétmilliárd biológiai megfigyelési adatot tartalmaz amely közel 70.000 adatforrásból származik. Ezen óriási adattömeg kezelése kapcsán számos nyílt forráskódú szoftvert fejlesztettek és szabadon használható szolgáltatást tartanak fenn, amellyel folyamatosan hozzájárulnak a biodiverzitási adatok legmagasabb szintű hasznosításához.

Az EU is egyre jelentősebb ráfordításokkal támogatja ezt a területet, közvetve (pl. *EOSC*, INSPIRE), vagy közvetlenül (pl.: EU Biodiversity Strategy for 2030, EU BON, Natura 2000), amely fejlesztési irányokkal jelentős hatást gyakorol a nemzeti szintű biodiverzitási adatkezelési megoldások fejlődésére.

Az eddig felsorakoztatott okok miatt egy mai követelményeknek megfelelő biodiverzitási adatkezelési rendszer alapvető jellemzői a magas fokú konnektivitás és az adatokat mint a komplex adat életút részét képező entitást kezelő rendszer, amely alacsony fejlesztési költség mellett (közösségi fejlesztési eszközökre támaszkodva) a legmodernebb technikai eszközökkel biztosítja a változatos igények reaktív és rugalmas kiszolgálását. A tanulmány elkészítése során ezeket a szempontokat figyelembe véve, eddigi biológiai adatbázis tervezési és üzemeltetési tapasztalatainkat is figyelembe véve fogalmaztuk meg a javaslatainkat.

2. Az Alapelvek megvalósíthatóságának vizsgálata

2.1. Meglévő adatok beemelése és tömeges import

A kitűzött céloknak megfelelő rugalmas adatkezelő és szolgáltató rendszer alapvető képességei közé kell tartozzon, hogy tetszőleges adatállományokat lehet vele egyszerűen importálni, akár az importált adatok struktúráit megőrizve, vagy azt újra strukturálva.

Az importálási feladatokat a következő szinten kell támogatnia az elkészült rendszernek:

- **Alkalmi importálások**: Ez valójában a ma már rendelkezésünkre álló digitális eszközöknek köszönhetően a legtöbb esetben nem bonyolult feladat. Fontos, hogy rendelkezésre álljon egy "erős" CSV import-export eszköz, amelyre bármilyen további importálással kapcsolatos folyamat könnyen ráépíthető, mivel a táblázatos szerkezetű adat viszonylag könnyen átalakítható CSV állománnyá, mert a legtöbb adatbázis kezelőbe egyszerűen lehet CSV állományokat importálni. A struktúrák átalakítását és kiegészítését pedig már egyszerűen el lehet végezni SQL parancsokkal az importált állományon.
  Például PostgresSQL-ben a beépített COPY paranccsal több millió soros csv fájlokat lehet táblázatba beolvasni pár perc alatt. A tömeges importálások esetén egy fontos szempont az importálandó adatok minősége és ennek kapcsán pedig a minőség ellenőrzés időzítése. A különféle rendszerek ebben alapvetően eltérnek. Az egyik véglet, amikor a rendszer teljesen előkészített adatokat vár és nem biztosít adatjavítási eszközöket, csak a hibákat jelzi. A másik véglet, amikor a rendszer a bejövő adatfolyamot strukturálatlan adatként beolvassa és a későbbi feldolgozási folyamat során biztosít eszközöket az adatok ellenőrzésére és a strukturált formára alakításra. Köztes megoldások is létezhetnek, amikor rendszer az importálási folyamatba épít be interaktív ellenőrzési eszközöket és ilyen módon opcionálissá teszi az adat előkészítés fő munkaterét. Ez utóbbi megoldás igen népszerű mivel lényegesen több lehetőséget biztosít a felhasználó számára, hogy kiválassza a számára legalkalmasabb adatkezelő platformot. Ezen túl viszont egy fontos szempont az is, hogy az adatellenőrzés fáradságos és aprólékos munkáját kire terheljük? Amennyiben a rendszer csak tökéletesen rendezett adatokat fogad be, úgy az adatfeltöltőre hárítjuk az adatok ellenőrzésének és javításának összes munkáját, ami jelentősen hátráltathatja az adatbevitelt, amennyiben nem központosított adatbeviteli úton történik az adatok importálása. Az adatok rendezése és

teljes validálása utólag is megvalósítható, ilyen esetben az adatfelvivő munkája minimalizálható ami olyan esetben praktikus, ha önkéntes adatfelvivőkkel dolgozunk. Ilyen esetben viszont központi adatrendező (adatkurátori) munkára van szükség az adatok rendszerezéséhez és validálásához.

- **Rendszeres importálások**: Azonos vagy hasonló struktúrájú állományok rendszeres importjának is hatékony köztes állománya lehet a CSV, mivel ezek alacsony szintű tartalmi és strukturális ellenőrzése kiválóan megvalósítható például *DBT* segítségével, de a *DBT* jól használható komplex struktúrák alkalmi importjának a felépítésére is. Az *OpenBioMaps*-ból jó példa számos táblázatos fájl típus rendszeres importálásának támogatására a tetszőleges mező összerendelés eszköze, ami automatikus mező felismerést és elmenthető felhasználói beállításokat is kezel.

  A rendszeres importálások során is a legnagyobb emberi ráfordítást igénylő feladat az adatok ellenőrzése és javítása. Az ellenőrzés nagyobb részben automatizálható, míg a javítás csak kisebb részben.

Tömeges adatok importálása esetén nem érdemes a felhasználói felületbe integrált ellenőrzéseket alkalmazni, mivel úgy sem lehet jól áttekinteni az importálandó nagy adattömeget, hanem az importálást ideiglenes struktúrába érdemes betölteni és onnan a validációt is elvégző adatbázis műveletekkel átemelni a végleges struktúrába. A validációt végző parancsok eltárolását, megosztását és ismételt végrehajtását támogató felület kialakítása sokat segíthet a rendszeres tömeges importálási folyamatokban. Az ideiglenes struktúrákban tárolt adatok áttekintését specifikus elemző alkalmazások készítésével lehet segíteni (például *R Shiny* környezetben fejlesztett dinamikus adatábrázolások), vagy általános adatelemző és rendszerező eszközöket lehet alkalmazni mint például az *OpenMetadata* (*lásd még 3. fejezet*) és *MetaBase*. Továbbá adattárház eszközök alkalmazása is megfontolandó különösen komplex adatfolyamok feldolgozása esetén. Az ilyen elemző eszközök használatával könnyebben át lehet tekinteni a nagy adattömegeket, amelynek köszönhetően jelentősen lehet csökkentheti a hibás adatok importálását és az emberi ráfordítást is.

Az importált adatok struktúrája indokolt esetben megőrizhető, amennyiben az adatbázisunk struktúrákra épülő struktúrákra (nézetek és materializált nézetek) támaszkodva tud dolgozni, vagy nagy fokú struktúra függetlenség jellemző rá. Mindkét megoldás támogatása jellemző az *OpenBioMaps*-ra. Adat repozitóriumok esetén alkalmazott megoldás, hogy a behozott adatok struktúráján nem változtatnak, hanem csak a meta adatokon keresztül teszik láthatóvá, elérhetővé az adatokat. Azaz, ilyen esetben az adatokkal nem lehet további műveleteket végezni az adatbázison belül, csak a metaadatokkal.

Adatok metaadatokkal együtt különböző komplex struktúrájának importálása is megoldható egy hatékony CSV felületen keresztül ilyenkor a metaadatok importálását külön kell megoldani és felépíteni. Érdemes a legismertebb metaadat sztenderdek importálására felkészülni, mint a *Frictionless (lásd még 3. fejezet)* és a *DarwinCore (lásd még 3. fejezet)*.

Érdemes a metaadatok önálló importálásának lehetőségét is támogatni azért, hogy a meglévő adatstruktúrák metaadatolását importált metaadatokkal is el lehessen végezni. A metaadatok importálását is lehet a CSV a köztes állomány alkalmazásával végezni, amennyiben a metaadatokat egyszerű táblázatos formában vagy az adattáblákhoz kapcsolódó adatbázis szintű metaadat táblákban szeretnénk tárolni.

2.2. Konnektivitás, API

Egy mai követelményeknek megfelelő adatkezelő rendszer egyik legalapvetőbb képessége a külső eszközökkel és szolgáltatásokkal való együttműködés képessége. Továbbá a rendszer belső felépítésének is jellemzően progresszív megoldása, ha önálló és egymáshoz kapcsolódó, egymással kommunikáló részegységekből (mikroszerviz architektúra) épül fel, amely megoldás lehetővé teszi az egyes rendszerösszetevők igény szerinti lecserélését az egész rendszer megváltoztatása nélkül. Ilyen esetben a belső komponensek egy belső *API*-val kommunikálnak egymással, ami egyúttal lehetővé teszi új, előre nem definiált szolgáltatások kialakíthatóságát is a belső struktúra módosítása nélkül. Ezzel szemben egy monolitikus rendszerben a rendszerkomponensek kapcsolatának merevsége miatt a rendszerbe tervezett kapcsolódási típusoktól csak jelentős fejlesztési ráfordítás mellett lehet eltérni. Bár rendszerüzemeltetés szempontjából az utóbbi egyszerűbb, az elosztott rendszerek üzemeltetésére próbál megoldást nyújtani például a Terraform (https://www.terraform.io/) ami egy IAsC (infrastructure as code) vagy például Helm (https://helm.sh/), ami a *Kubernetes* alkalmazás csomag kezelője. Ezek az eszközök az infrastruktúra más-más szintjein képesek deklaratívan leírni az üzemeltetés különböző lépéseihez tartozó folyamatok - telepítés, konfigurálás, integrálás, frissítés, leállítás stb. - részleteit.

A várható kapcsolódási területek a következők lesznek:

- Adat állományok (fájlok, stream tartalmak) behozatala és kivitele külső szolgáltatók vagy alkalmazások felé. Például Google Drive, Dropbox, Nextcloud, *QGIS*, *R* támogatás.
- Külső adatforrások integrálásának lehetősége. Például *GBIF*, vagy iNaturalist adatok, vagy szolgáltatások integrálása saját szolgáltatásokba. *OpenBioMaps* adatbázisokból adatok, vagy adatok nézetének integrálása, vagy *OpenBioMaps* adatbázisokkal adatok szinkronizálása. Metaadat szolgáltatások integrálása (metaadat keresés és publikálás).

Figyelembe vehető sztenderdek, szolgáltatások és technológiák:

- *OpenApi* használata a szabványos felületű kliens oldali *library* generálás miatt jelentősen csökkentheti új kliens alkalmazások fejlesztési költségeit, vagy meglévő alkalmazásokhoz fejlesztendő csatolófelületek fejlesztési költségeit.
- *OpenMetadata* használata jelentősen csökkentheti az adatok rendszerezésének költségét és így jelentősen segítheti az adattartalmi hibák feltárását.
- *DarwinCore* használatával az adat megosztásokat, exportálások és archiválásokat lehet sztenderd módon megvalósítani.
- *FrictionLess* egy sztenderd eszközkészlet biztosít az adatok és metaadatok tárolására és cseréjére (import-export) amelynek alkalmazásával nagyban növelhető a tervezett

rendszer interoperabilitása.

- *FDW (Foreign Data Wrapper)*: Adattáblák lekérdezése távoli adatbázisokból olyan módon, hogy a távoli adat nem tárolódik a helyi szerveren. Az FDW elérhető technológia PostgreSQL szerverekhez kapcsolódásra. Sokféle adatbázis szerver felől lehetőséget nyújt egyéni adatkapcsolati eszközök (egyéni FDW) fejlesztésére is. Az Oracle DBLink kiterjesztéssel is nyújt hasonló szolgáltatásokat és a MySQL Federated Storage Engine lényegesen korlátozottabban nyújt ilyen jellegű lehetőségeket. A PostgreSQL dblink modullal a távoli adatbázisokból lekérdezéssel a helyi adatbázis struktúrában lehet eltárolandó adatokat lekérdezni. Az *OpenBioMaps* alkalmaz FDW kapcsolódásokat külső adatbázisokhoz és FDW alkalmazással hozzá is lehet kapcsolódni, habár ennek jelenleg még nincs adminisztratív kezelő felülete.

2.3. Elemző eszközök

Az elemzési funkciók alapvetően kétféleképpen megvalósíthatóak. Vagy előre meghatározott elemző funkciók érhetőek el a rendszerben, vagy pedig tetszőleges elemzési eszközök kapcsolódására biztosít a rendszer felületeket. Ez a két megoldás irány természetesen kombinálható egymással. Az előbbi előnye, hogy azonnal használható a felhasználóknak, míg a hátránya magasabb fejlesztési költség, ami jelentősen növekszik, ha ezek az előre lefejlesztett elemző megoldások rugalmasan konfigurálhatóak a felhasználók által.

A tetszőleges elemző eszközök kapcsolódási felületének biztosítása lényegesen egyszerűbb (és így kevésbé költséges) és bármilyen potenciális elemzéshez használható, viszont nagyobb szakértelmet igényel a használata. A külső elemző eszközök kapcsolódásához alapvető és sztenderd adatinterfészekhez való hozzáférést szükséges biztosítani, mint például *PostgreSQL*, vagy WMS/WFS kapcsolat.

A külső elemző eszközök rugalmas kapcsolódásának lehetőségét úgy is lehet biztosítani, hogy az alapvető szolgáltatások elé specifikus proxy alkalmazásokat helyezünk (SQL vagy *Mapserver* proxy), amelyek a külső kliens számára úgy viselkednek mintha azok az alapvető szolgáltatásokhoz kapcsolódnának, de valójában csak előre definiált funkciókat vagy az alap szolgáltatás behatárolt lehetőségeit biztosítják a kliens számára. Az utóbbi esetben transzparens módon, az előbbi esetben az alkalmazás belső *API* felületén keresztül.

Az alacsony szintű adatkapcsolatok bármilyen megoldású biztosítása lehetővé teszi külső alkalmazások alacsony szintű integrációját, ami nagyban megkönnyíti a komplex elemzési munkákat.

A *QGIS* asztali térinformatikai rendszer például számos adatforrás között támogatja távoli *PostGIS* réteg kezelést is, amely *PostgreSQL* szerverek *PostGIS* kiterjesztéssel rendelkező adatbázisaiban elérhető adattábláinak közvetlen kezelését teszi lehetővé asztali grafikus környezetben.

Az *R* az egyik legnépszerűbb statisztikai programozási környezet, amely szintén rendelkezik *PostgreSQL* kapcsolattal és képes kezelni a *PostGIS* kiterjesztés által nyújtott geometriai eszközöket is és kimagasló eszköze egyéni analízisek készítésének.

A Jupyter egy web-alapú interaktív platform amely alapvetően támogatja az R környezetet és *PostgreSQL* adatkapcsolat támogatással is rendelkezik. Alapvető eszköze lehet interaktív és kollaboratív elemzések készítésének.

Az *OpenMetadata* szintén támogatja a *PostgreSQL* intergrációt és további elemző és adatkezelő eszközök kapcsolódását is mint például a *Metabase*, amely kiváló eszköz egyszerű elemzések készítéséhez és megosztásához, vagy külső *Adattárház* szolgáltatókat is.

A itt felsorolt egyéni és komplexebb alkalmazásokon túl léteznek olyan adatkezelő platformok is amelyek elsődlegesen az elemzésre fókuszálnak, ezek a *Data Warehouse* (*Adattárház*) platformok teljes körű megoldást nyújtanak az adatok tárolására, kezelésére és elemzésére, de egyes eszközökön keresztül integrálhatóak is.

2.4. Térkép alapú adatleválogatás

Térképi alapú adatmegjelenítés minden biodiverzitási adatokat szolgáltató (webes alapú) rendszernek alapvető tulajdonságai közé tartozik, míg a térképi alapú leválogatás leginkább csak a megjelenített rétegek ki-be kapcsolására korlátozódik. Számos példát lehet találni a nemzetközi szinten országos és globális biodiverzitási adatportálokon a térképi adatkezelő funkciók megvalósítására, amelyek többsége nyílt forráskódú, úgyhogy az alkalmazott megoldások részletesen tanulmányozhatók és felhasználhatók. Például:

Horvátország nemzeti biodiverzitási adat portálján:
https://www.bioportal.hr/gis/?lang=en

A *GBIF* faj előfordulási adatlekérdező felületén:
https://www.gbif.org/occurrence/map?occurrence_status=present

Forráskódok: https://github.com/gbif/maps, https://github.com/gbif/gbif-basemaps, https://github.com/gbif/tile-server, …

India nemzeti biodiverzitási adat portálján:
https://indiabiodiversity.org/map
Forráskódok: https://github.com/strandls?q=biodiv

Írország nemzeti biodiverzitási adat portálján:
https://maps.biodiversityireland.ie/Map

Az angol nemzeti biodiverzitás hálózat atlasz oldalán:
https://records.nbnatlas.org/search#tab_spatialSearch

Svédországi Biodiverzitási adat infrastruktúra oldalán:
https://collections.biodiversitydata.se/?lang=enq
Forráskódok: https://github.com/bioatlas

Európai Biodiverzitás portálján:
http://biodiversity.eubon.eu/web/guest/dataset-search

Az ICES (tengeri adatok) adat portálján:
https://data.ices.dk/

Az OBIS (Óceán Biodiverzitási Információs Rendszer) weboldalán:
https://mapper.obis.org/

Kenya Biodiverzitás portáljának atlasz oldalán:
http://biodiversityatlaskenya.org/data/kenya-biodiversity/#kenya

Ausztrália Biodiverzitás Adatportálja
https://www.ala.org.au/
Forráskódok: https://github.com/AtlasOfLivingAustralia

Antartic Biodiversity Portal:
https://www.biodiversity.aq/
Forráskódok: https://github.com/biodiversity-aq

OpenBioMaps biodiverzitási adatkezelő platform:
https://openbiomaps.org
Forráskódok: https://gitlab.com/openbiomaps

Német biodiverzitási adatkezelő platform:
https://www.gfbio.org/
Forráskód: https://github.com/orgs/gfbio/repositories?type=all

Az összes fenti példában a térképi adatleválogatás habár elérhető funkció, de nem határozza meg a felület megjelenését és a rendszer logikai működését. Egyes biodiverzitási adatportálok esetében önálló, független webes szolgáltatásként jelenik meg. Több biodiverzitási adatportálon a térképi megjelenés órási adattömegek kezelését oldja meg meglehetősen változatos adatforrásokra támaszkodva, míg más esetben a térképi adatmegjelenítés elsősorban a metaadatok kezelésén keresztül történik csak.

A legtöbb esetben a térképi leválogatás csak a megjelenítendő rétegek kiválasztására korlátozódik, míg az olyan esetekben ahol önálló *GIS* alkalmazás felelős a térkép kezelésért, ott lehetőség van a térképi rétegek megjelenítésének finomabb szabályozására is. Az olyan esetekben ahol a térkép kezelés sztenderd webes térkép szolgáltatásra épülő egyedi kliens alkalmazáson alapul, ott a térképi megjelenítés felhasználó szintű finom szabályozása nem jellemző. Gyakran van lehetőség szöveges alapú leválogatásra és ritkábban egyéni felrajzolt vagy importált poligonokkal is lehet adatokat leválogatni.

A fenti webes példákon túl azonban fontos figyelembe venni, hogy a térkép alapú adatleválogatás nemcsak webes felületen történhet, hanem különféle asztali alkalmazásokból is. Egy kellően rugalmas háttérszolgáltatás lehetővé teszi a felhasználóknak, hogy ugyanazokat a térképi funkciókat elérjék weben és asztali alkalmazásból is, továbbá a szolgáltatás beépíthető legyen más alkalmazásokba is. Ez a rugalmasság sztenderd térképkezelő eszközök alkalmazásával valósítható meg. Ezek a követendő sztenderdek az Open Geospatial Consortium (*OGC*) által rögzített nyílt sztenderdek térbeli adatok kezelésére és megosztására. A térbeli adatok web alapú megjelenítésére és kezelésére vonatkozó sztenderdek alkalmazása lehetővé teszi az elvárt rugalmasság megvalósítását, habár vannak olyan fizetős (főleg régebbi) térinformatikai alkalmazások, amelyek használata nem javasolt (például ArcSDE 9.3-tól támogatja csak a *PostgreSQL* kapcsolatot, míg az ArcGIS 10.3-es

verziótól kezdve a *PostGIS* kapcsolatot ld.: *Esri Support for Open Geospatial Standards*[1], *Connect to PostgreSQL from ArcGIS*[2]).

Összességében elmondható, hogy a webes térképi tartalmakat támogató sztenderdek (*OGC* Sztenderd Térkép Szolgáltatások) minden jelentős nyílt forráskódú (*GeoServer*, *Mapserver*, *QGIS Server*) és a legtöbb fizetős térinformatikai szerver eszköz által támogatottak (ArcGIS Szerver, GeoMedia Szerver) és nincs alternatívája rugalmasság és támogatottság terén. Az *OGC* Sztenderd Térkép Szolgáltatások használata jelentősen csökkenti a fejlesztési költségeket, növeli a szoftver fenntarthatóságát, a komponenseinek konnektivitását és a fejlesztetett kódok és eszközök hordozhatóságát.

A Web alapú térképi adatlekérdezés kliens oldali megvalósításához egy *OGC* kompatibilis webes térképi (*JavaScript*) *library* vagy egy komplett *OGC* kompatibilis keretrendszer használata szükséges. Mivel a megvalósítandó rendszernek nem a térinformatika műveletek végzése az elsődleges célja, hanem ez csak egy eszköz lesz a biológiai adatok kezelése során így várhatóan webes környezetben csak egyszerűbb térinformatikai adatleválogatásra és térképi adat megjelenítésre kell felkészíteni. Ezért egy komplex webes térinformatikai platform alkalmazása helyett csak téradatok kezelésére és megjelenítésére tervezett (*JavaScript*) *library* használatára épülő egyedi alkalmazás fejlesztés lehetőségeit tekintjük csak át. Ezt indokolja az is, hogy a rendelkezésre álló szabadon felhasználható webes térinformatikai technológiák száma nagy és közösségi támogatottságuk kimagasló, ezért az ezekre épülő egyedi fejlesztés költsége várhatóan messze alulmarad egy komplett webes térképi megjelenítő platform megvásárlásának költségeitől (pl. ARCGis for Server, Oracle Spatial and Graph), különösen, ha figyelembe vesszük, hogy a komplex platformokhoz is szükseges egyedi fejlesztéseket végezni, de ezeknek a fejlesztése jóval költségesebb mint a nyílt forráskódú rendkívül elterjedt alternatívákra épülő fejlesztési költségek.

A két legjelentősebb nyílt forráskódú webes térképi adatkezelő *JavaScript library* a *Leaflet* és az *OpenLayers*. Mindkettő kiválóan dokumentált és óriási felhasználó közösség használja és nagyon jól működik. A *Leaflet* használata egyszerűbb és megbízhatóságát jelzi, hogy pl. a FaceBook, OpenStreetMap, vagy a Flickr is használja ezt a *library*-t a térképei megjelenítéséhez.

A webes térképi megjelenítést és térképi lekérdezési felületek fejlesztésének kiszolgálását végző kliens *library*-k a térképi adatokat különféle adatforrások dinamikus lekérdezésével tudják megvalósítani. Az *OGC* Sztenderd Térkép Szolgáltató Szerver (pl. Web Map Service) különféle kliens programokat képes kiszolgálni, webes és asztali alkalmazásokat is. A web protokoll a webes térképi szerverek esetében nem jelenti azt, hogy csak web alapú klienseket képes kiszolgálni. A két legalapvetőbb (*OGC*) Webes Térképi Szerver Szolgáltatás a WMS (Web Map Service), amely a web-api kérésekre térképi kép állományokat küld válaszképpen a kliensnek és a WFS (Web Feature Service), amely a web-api kérésekre térbeli megjelenítést lehetővé tévő szöveges állományokat küld válaszképpen (geographical features). A térkép alapú adatleválogatás kiszolgálása ezek egyikével, vagy másikával, vagy ezeknek a kombinációjával

[1]. https://www.esri.com/content/dam/esrisites/en-us/media/technical-papers/esri-support-for-open-geospatial-standards.pdf

[2]. https://desktop.arcgis.com/en/arcmap/10.3/manage-data/gdbs-in-postgresql/connect-postgresql.htm

is megvalósítható. Ezeken túl további Sztenderd térképi megjelenítéssel kapcsolatos szolgáltatások is léteznek, amelyek közül leginkább a WCS (például raszter megjelenítés tér-időbeli lefedettség lehatárolással) és a WMTS (nagy hatékonyságú - előre definiált térkép csempék szolgáltatása, ritkán változó adatforrásokhoz) sztenderdeket érdemes megemlíteni.

A WMS és WFS protokollokat minden említésre érdemes térkép szolgáltató szerver alkalmazás támogatja, míg további szolgáltatások támogatásában az egyes szerver megvalósítások eltérnek.

A két legelterjedtebb *OGC* kompatibilis nyílt forráskódú térkép szerver a *Mapserver* és a *GeoServer*. Ezeknek az összehasonlítása megtalálható a 2.6. fejezetben. A térképi adatleválogatás szempontjából többnyire mindegy, hogy milyen térkép szolgáltató szerver szolgálja ki a kéréseket és, hogy milyen protokollt alkalmazva milyen információkat alapján fogja a kliens a térképeket megrajzolni. Ami miatt itt mégis érdemes foglalkozni vele, az éppen az egyes szerverek közötti tudás különbség az extra szolgáltatások terén, mivel a térképi leválogatások különleges, elsősorban nem képi adatokat tartalmazó forrásállományokat vagy éppen nagy méretű képi állományok kezelését, vagy külső forrásállományok tovább szolgáltatását is jelenthetik. Mindennek ellenére nem javasoljuk egy univerzális térinformatikai kiszolgáló szerver alkalmazását a feladatra, mert a mindennapi használatban a leggyakrabban csak olyan egyszerű térinformatikai feladatokat kell kiszolgálnia ezeknek a szervereknek, ami miatt nem érdemes egy nagyon magas fenntartási költségű komplex kiszolgáló szervert üzemeltetni. Továbbá a komplex rendszerek használatát nehezíti a túl sok választási opció, amely a felhasználók és adminisztrátorok számára is nehezíti a használatot. A különleges térinformatikai igényű adatmegjelenítések és adatleválogatások egymástól szétválasztott specifikus szerver szolgáltatásokkal is megvalósíthatóak, figyelembe véve, hogy a térképszerver mögötti adatforrás közvetlenül is kezelhető a fejlettebb asztali *GIS* alkalmazásokban, amelyek minden további extra adatforrás kezelését is meg tudják oldani és így egy összetettebb szerver szolgáltatás folyamatos fenntartása helyett az alkalmi szolgáltatás használókra hárítjuk szolgáltatás felépítését (Pl. *QGIS* projekt összerakása különböző forrás rétegekből).

2.5. Jogosultságkezelés

A jogosultság kezelés kapcsán a következő feladatokat kell megoldani:

- Hozzáférési szintek meghatározása a következő funkciókkal kapcsolatban
  - nyers adat (raw data) hozzáférés,
  - adminisztratív funkciók,
  - felhasználói funkciók
- Adatmegosztási lehetőségek (mint például google share, vagy nextcloud share)
  - Belső megosztás - regisztrált felhasználók közötti tartalom megosztás
  - Külső (visszavonható) megosztás webes linkkel
  - Tartós (publikált) megosztás külső hivatkozással (pl. DOI)
  - Metaadatok megosztása
- Jogosultságok rendszerezése: csoportok kezelése, jogosultságok öröklése (mint például a *PostgreSQL* jogosultság kezelése)
- Felhasználók közötti kommunikáció jogosultság támogatása
- Rendszerek közötti kommunikáció jogosultság támogatása

A jogosultság kezelés a fejlesztendő rendszer várható nagy komplexitása miatt hiába egy alapvető funkció, mégis valószínűleg csak több egymáshoz kapcsolódó komponens együtteseként lehet megvalósítani. Nem tudunk általános kész eszközt javasolni, hanem csak gyakorlati példákat az egyes komponensek jogosultság kezelésére. A jogosultságkezelés megbízhatósága miatt viszont fontos normalizálni a funkciókat és lehetőség szerint egyetlen meta-funkció használatával egy jogosultságkezelő *API*-n keresztül kialakítani egyetlen jogosultságkezelő réteget.

A jogosultságkezelés megvalósítása az egész rendszer működésére hatással van, mind a felhasználói használati élmény (mint például az alkalmazás sebessége, vagy egyes funkciók használatának intuitivitása), mind pedig a fejlesztési ráfordítás szempontjából.

2.6. Változáskövetés

A változáskövetés kialakítása és alkalmazása hatással lehet az adatok tárolási struktúrájára. Amennyiben a változáskövetés célja a minőségbiztosítás és hitelesség bizonyítása, úgy más technikák alkalmazhatók, mintha az elsődleges cél az adatok valamilyen korábbi állapotának megtekintése és/vagy visszaállíthatósága. Amennyiben csak a változás tényét kell tárolni, úgy elegendő a sorokból egy hash azonosítót (egyedi karakterlenyomat) generálni és azt eltárolni valahol - más független adatbázisban vagy repozitóriumban. Amennyiben, viszont a változás tartalma is érdekes, úgy a változott tartalom valamilyen strukturált tárolásában kell gondolkodni és a változás állapotokat a legalacsonyabb szinten követni (pl. trigger függvények). Amennyiben nincs rá igény, hogy az egyes sorok változásai külön megtekinthetőek legyenek, úgy lehet a változás követést az adattáblák napi mentésével, esetleg inkrementális (változás függő) mentésével végezni. Ilyen esetben el lehet tárolni a változás tényét és időpontját adatbázisból elérhető módon, ami megkönnyíti megfelelő archív állomány visszakeresését szükség esetén.

Amennyiben igény van az egyes sorok változásainak folyamatos követésére, úgy valamilyen adatsor alapú struktúrába érdemes menteni a változásokat. Az *OpenBioMaps* például egy cél adattáblát alkalmaz erre, amelybe minden módosításnál és adattörlésnél kerül át az érintett adatsorok tartalma tömörítve, amelyet egy SQL trigger függvény hajt végre. A módszer előnye, hogy alacsony szinten (kliens alkalmazástól függetlenül) minden módosítás eltárolódik és a változások könnyen megtekinthetőek. A hátránya, hogy a rendszeres és tömeges (automatikus) változásokat is követi a rendszer, amely változásokat egyébként nem biztos, hogy szeretnénk eltárolni. Ez utóbbi hátrány bármilyen más soronkénti alacsony szintű archiválási megoldást is érinthet. A soronkénti változáskövetés egy komplexebb megoldása lehet a változás történeti állományok kiemelése az SQL környezetből és ezek szöveges fájlként kezelése valamilyen verzió követővel (pl. *GIT*). Ilyen esetben a változás történeti állomány mérete nem növekszik folyamatosan, csak a különbségek tárolódnak el, és azok is tömörítve. A módszer hátránya, hogy saját alacsony szintű SQL szerver függő fájlkezelő megoldást kell hozzá fejleszteni.

2.7. AI támogatás

Az AI (mesterséges intelligencia) alkalmazás támogatás megvalósítása jelentősen eltérhet az alkalmazás céljától függően. A tervezett alkalmazás megvalósítása szempontjából szöveges vagy képi információk tanításon alapuló osztályozása lehet leginkább érdekes, továbbá szöveges információk automatikus csoportosítása is az adat elemzések során.

Szöveges és képi tanuláson alapuló osztályozás lehetséges felhasználási területe lehet az adatok validálása és a hibák keresése.

Habár vannak kiváló, kész keret rendszerek különféle gépi tanuláson alapuló fejlesztések megvalósításához, de az egyes feladatok megoldásához mindig egyedi modelleket kell fejleszteni, ami miatt nem reális egy általános AI kezelő felület fejlesztése és beépítése a rendszerbe. Az adatok elemzéshez való előkészítésének támogatása egyes elemzési típusok igényei szerint viszont megvalósítható feladatnak tűnik, habár érdemes minél pontosabban meghatározni az igényeket, mert az általános megoldások nagyságrenddel nagyobb fejlesztési igényűek, ami AI fejlesztés esetén különösen jelentős lehet.

Ilyen adat előkészítés támogatás lehet például a faj elterjedési modellek készítéséhez az adatok automatikus ellenőrzése és rendszerezése (például Random Forest, vagy XGBoost alapú elemzésekhez), továbbá képi tartalmak címkézése (például *CVAT* alkalmazással és így a *CVAT* beépítése a munkafolyamatba).

A mesterséges intelligencia (AI) mint döntéstámogatási eszköz támogatása az adatbázis szempontjából egy adat hozzáférést igénylő funkció. Így, ez magában foglalja a megfelelő hozzáférési jogosultságok és a rendszerhez való megfelelő kapcsolódási lehetőségek kezelését is. Ennek megalapozása lehet az olyan adatbázis kezelő választása, amely egyrészt széleskörűen támogatja a szabványos adat hozzáférési felületeket (kliens kapcsolódás, export funkciók) másrészt jól cizellálható jogosultság kezelő rendszert tartalmaz. Ezen túl pedig az adat elérést a jogosultság kezeléssel összekapcsolva leginkább egy jól definiált *API*-n keresztül (pl. *OpenApi*) lehet a leghatékonyabban megvalósítani. Ezen funkciók meglétével, bármely iparági szoftverrel (TensorFlow, PyTorch, Keras, Theano, Singa, RetinaNet) vagy egyedi fejlesztett alkalmazással az AI használata támogatható.

A RetinaNet az egyik legjobb egylépcsős objektum felismerő modell, amely bizonyítottan jól működik sűrű és kis méretű objektumok esetén. Emiatt vált népszerű objektumfelismerő modellé, amelyet például légi és műholdas képeken használnak.

A Keras (https://keras.io) egy hatékony, Python nyelven írt, magas szintű neurális hálózati alkalmazásprogramozási felület (*API*). Ezt a nyílt forráskódú neurális hálózati könyvtárat úgy tervezték, hogy gyors kísérletezést biztosítson mély neurális hálózatokkal, és a *CNTK-ra*, a TensorFlow-ra, vagy a Theano-ra épülve futhat. *ONNX* kompatibilis.

A PyTorch (https://pytorch.org) egy viszonylag új, Torch-on alapuló mélytanulási keretrendszer. A Facebook AI-kutatócsoportja fejlesztette ki, és 2017-ben nyílt forráskódúvá tette a *GitHub*on, és természetes nyelvfeldolgozó alkalmazásokhoz használják. *ONNX* kompatibilis.

A TensorFlow (https://tensorflow.org) a Google által fejlesztett nyílt forráskódú mélytanulási keretrendszer. A TensorFlow a közösségi erőforrások, könyvtárak és eszközök rugalmas, átfogó ökoszisztémáját kínálja, amely megkönnyíti a gépi tanulási alkalmazások építését és telepítését. *ONNX* kompatibilis.

A Theano korábban az egyik legnépszerűbb mélytanulási könyvtár volt, egy nyílt forráskódú projekt, amely lehetővé teszi a programozók számára, hogy matematikai kifejezéseket definiáljanak, értékeljenek és optimalizáljanak, beleértve a többdimenziós tömböket és mátrix értékű kifejezéseket. A Theano-t a Montreali Egyetem fejlesztette ki 2007-ben, és a Pythonban a mélytanuláshoz használt egyik legfontosabb alapkönyvtár, de ma már kevésbé használt.

Nemzetközi AI alapú, projektrendszerű, automatikus faj és egyed felismerő rendszerek, amelyek működési példája, vagy az elérhető felületei hasznosíthatók lehetnek a tervezett rendszerben:

- Wildlife Insights (fajfelismerés kameracsapdás képekről)
- Wildlabs (egyed felismerés)

Képekről, vagy hangról fajfelismerést AI alapon is végző közösségi rendszerek, amelyek működési példája, vagy az elérhető felületei hasznosíthatók lehetnek a tervezett rendszerben:

- iNaturalist
- Merlin Bird ID
- Flora Incognita
- PlantNet (Automated plant species identification—Trends and future directions)

2.8. Testre szabható felület

A tervezett alkalmazást a korábban (2.2 részben) tárgyalt okoknál fogva mikro-komponens architechtúrával érdemes elkészíteni, ami a felhasználói felület rugalmasságát már alacsony szinten biztosítja. A felhasználói felület ilyen módon az alkalmazás többi komponensétől független módon fejleszthető és konfigurálható, vagy igény szerint lecserélhető.

Általában jellemző, hogy a felhasználói felületek a megvalósítható feladatok számának növekedésével egyre bonyolultabbá válnak, ami miatt az egyénileg konfigurálható és moduláris struktúrájú felületek a felhasználók számára egy jó alapkonfiguráció esetén kényelmesebben használhatóak lehetnek, mint egy sok funkciót alapértelmezetten tartalmazó felület. Habár a moduláris felhasználói felület elkészítése a funkciók nagyobb fokú általánosítása miatt és a komplexebb kapcsolódási lehetőségek miatt több energiát igényel, a felhasználói támogatás viszont várhatóan kevesebb ráfordítással lesz megoldható.

A rugalmasság tovább növelhető, hogy ha a már egyébként moduláris felület beépülő alkalmazások (plugin) használatával is bővíthető. Ilyen módon lehetőséget tud a rendszer biztosítani akár külső, független fejlesztők által készített mikroalkalmazások integrálására is, illetve új, előre nem tervezett igényeket kiszolgáló funkciók egyszerű megvalósítására.

2.9. Riport funkciók

Riport funkciók megvalósítására különféle sablon-motor (template engine) megoldások alkalmazásával lehet felkészülni. A sablon-motor lehetővé teszi, hogy statikus sablon fájlokat használjunk az alkalmazásban. Futáskor a sablon-motor a sablon fájlban lévő változókat tényleges értékekkel helyettesíti, és a sablont a kliensnek küldött cél-fájllá alakítja át.

Az OpenTBS a TinyButStrong PHP Template Engine egy plugin-je, amely excel (ODS, XLSX, …) és word (ODT, DOCX, …) dokumentumok kitöltésére szolgál. Ezen kívül számos sablon-motor program és könyvtár létezik, amelyek felhasználhatók egyedi vagy rugalmas riport funkciók készítésére.

Az *OpenBioMaps*-ban elérhető egy sablon készítő alkalmazás, melynek használatával a felhasználók tetszőleges sablonokat tudnak készíteni, azaz maguk tudják meghatározni a dokumentum sablon elemeit. Ez a funkció meglehetősen nagy rugalmasságot biztosít egyéni riport funkciók elkészítéséhez.

Ezen túl szüksége lehet automatizált riportok készítésre is, amelynek fejlesztése elsősorban a hibakezelések és bejövő tartalmi ellenőrzések miatt bonyolult feladat. Tulajdonképpen unit-teszteknek megfeleltethető tartalmi egység teszteléseket kell elvégeztetni, amelyek kimenete, hogy a dokumentum előállítható, vagy sem. Egy ilyen eszköz alkalmazásával a riport funkciók alkalmazása előtt már fel lehet tárni olyan adat hiányosságokat vagy pontatlanságokat, amelyek megakadályozhatják egy riport összeállítását. Habár ezeknek a tartalmi-unit-teszteknek a megírása feladatonként egyéni fejlesztést igényel, a fejlesztést előre elkészített példa tesztek elkészítésével lehet támogatni.

2.10. Nyílt forráskód és vagy fizetős alkalmazások

A fejlesztendő rendszer esetében minden fő komponens amelyet nem kell újonnan fejleszteni (adatbázis szerver, térkép szerver, térkép *UI*, alkalmazás szerver, *API*) több alternatív nyílt forráskódú szabadon felhasználható és piaci alkalmazás felhasználásával is megvalósítható. Nincsen egyetlen olyan komponense sem, amelyre a megvalósítandó célok tekintetében a piaci megoldások magasabb szintű szolgáltatást nyújtanának.

Viszont számos olyan potenciálisan használható internetes szolgáltatás van, amelynek van ingyenesen használható és fizetős változata is. A fizetős szolgáltatás változatokat többnyire azok intenzív használata esetén kell választanunk és egyes esetekben további kényelmi szolgáltatásokat is nyújtanak az ingyenes változataikhoz képest.

Ilyen potenciálisan igénybe vehető ingyenes-fizetős hibrid, vagy csak fizetős online szolgáltatások lehetnek például:

- *Metabase* (ún. *Business Intelligence* eszköz)
- *GitLab DevOps* Platform
- *AWS* (Cloud Computing, Machine Learning, RedShift, …)
- *Google Cloud* (Vertex AI, Google Maps, Cloud Computing, BigData ….)
- Microsoft Azure (AI, Cloud Computing)

- Xplenty, Teradata, Domo (*Adattárház* szolgáltatók)

2.11. Jó gyakorlatok áttekintése, alkalmazása

Fejlesztési jó gyakorlatok

- A nagy méretű és komplex rendszerek fejlesztése folyamatos, amelynek hosszú távú fenntarthatósága miatt legjobb gyakorlatnak a nyílt forráskódú és közösségi fejlesztésre is támaszkodó fejlesztési modell alkalmazása tűnik. Erre azért van szükség, hogy a szoftver fejlesztése és fenntartása ne függjön egyetlen cégtől és hosszú távon a lehető legkedvezőbb árú legyen. A szoftver fejlesztés költsége leghatékonyabban nyílt forráskódú modell követésével valósítható meg (*CNECT OpenSourceStudy 2021*[3]). A nyílt forráskód magában még nem garantálja, hogy további fejlesztők is csatlakoznak a fejlesztéshez. Ehhez a szoftver minőségének és dokumentáltságának és felhasználó licencének is olyannak kell lennie, hogy ezek motiválják további fejlesztések kialakulását és további fejlesztők csatlakozását a projekthez.
- A fejlesztői környezet nagyon sokat számít a fejlesztés sikerességében és különösen meghatározó jelentőségű olyan esetben, amikor a fejlesztők nincsenek közvetlen kapcsolatban egymással, ami jellemző a nagyobb nyílt forráskódú fejlesztésekre. A legelterjedtebb fejlesztői környezetek manapság *GIT*-re épülő megoldások, mint például a *Github* és *Gitlab.*
- Unit tesztek fejlesztése a kód fejlesztéssel együtt.
- *CI/CD* alkalmazása, például *Gitlab* CI
- Meglévő nyílt forráskódú megoldások felhasználása a komponensek kialakítása során, de nem minden áron, csak olyan esetekben, amikor a beépítendő kész eszköz valóban pont megfelelőnek tűnik a feladatra.
- Óriás, mindenre megoldást nyújtó rendszerek integrálását kerülni kell (pl. Bootstrap)
- Ökológiai lábnyom figyelembe vétele (*Microsoft Docs 2022*[4]) a fejlesztés és az üzemben tartás tervezése során.

Kód fenntarthatósága

Az elkészített alkalmazásnak fejlesztői szempontból is fenntarthatónak kell lennie, ami azt jelenti, hogy a lehető legkisebb energiabefektetéssel tudnia kell követni a fejlesztésnek a változó igényeket és technikai környezetet. Ezt úgy lehet elérni, hogy alapvető szoftver tervezési alapelveket betartanak a fejlesztők. Ezek legalább a következőknek kell lenniük:

Clean Coding (tiszta kódolás):

- A fejlesztő adjon értelmes neveket a változóknak, függvényeknek, osztályoknak és a kódban található egyéb entitásoknak;
- Hozzon létre olyan függvényeket, amelyek kicsik és egyetlen dolgot végeznek;
- Helyezze a kapcsolódó adatokat és függvényeket kis, független osztályokba;
- Strukturálja a kódot a jobb olvashatóság érdekében. Az összefüggő kódot tartsa együtt, és a sorokat tartsa rövidnek;
- Fokozza az olvashatóságot megfelelő megjegyzésekkel. Minden függvény legyen dokumentálva a kódban.

---

[3]. https://digital-strategy.ec.europa.eu/en/library/study-about-impact-open-source-software-and-hardware-technological-independence-competitiveness-and

[4]. https://docs.microsoft.com/hu-hu/learn/modules/sustainable-software-engineering-overview/

(Unit) tesztek írása: automatikusan futtatható, gyors és az alkalmazástól független tesztek készüljenek az alkalmazás funkciók tesztelésére.

DRY (Don't Repeat Yourself): Ne legyenek logikai ismétlődések a kódban.

SOLID (könnyen érthető, karbantartható és rugalmas szoftver tervezés):

- Egyetlen felelősség elve: "Soha nem lehet egynél több oka annak, hogy egy osztály változzon". Más szóval, minden osztálynak csak egy felelőssége legyen;
- Nyitott-zárt elv: "A szoftver entitásoknak nyitottnak kell lenniük a bővítésre, de zártnak a módosításra";
- Liskov helyettesítési elv: "Azoknak a függvényeknek, amelyek mutatókat vagy hivatkozásokat használnak az alaposztályokra, képesnek kell lenniük arra, hogy a származtatott osztályok objektumait használhassák anélkül, hogy ezt tudnák";
- Interfészek elkülönítésének elve: "Sok ügyfélspecifikus interfész jobb, mint egy általános célú interfész";
- Függőségi inverzió elve: "Absztrakcióktól függjünk, [ne] konkréciókról".

Üzemeltetési jó gyakorlatok

- Kevés központi adminisztrátor, a rendszer felhasználói többnyire maguk meg tudják oldani a problémáikat;
- Közösségi kommunikációs fórum fenntartása (email lista, fórum);
- Közösségi dokumentáció készítés (pl. *Mapserver* dokumentáció);
- További rendszerek létrejöttének támogatása a nagyobb fejlesztői és felhasználói bázis kialakításának érdekében;
- Közös szolgáltatások fenntartása a többi hasonló rendszer fenntartójával a hatékonyabb eszköz kapacitás és nagyobb üzemelési stabilitás kialakítása érdekében.

Más rendszerekben kifejlesztett átvehető megoldások, szabadon használható szolgáltatások és követendő szabványok

- *GBIF* nyílt forráskódú alkalmazásai
  - Species matching (fajnév listák automatikus ellenőrzése. *API*-n keresztül is használható)
  - Name parser (Különböző írásmódú fajnevek automatikus darabolása. *API*-n keresztül is használható tömeges név darabolásra)
  - Relative observation trends (fajonként időszakok szerinti dinamikus trend állapot változás ábrázolása)
- *Frictionless* szoftverek
  - *Frictionless* Components (*UI* eszközök a metadata adatok dinamikus megjelenítésére)
  - *Frictionless* Framework (adatok leírása, kivonatolása, ellenőrzése és átalakítása)
- *EOSC* keresztül (is) igénybe vehető szolgáltatások,
  - GEOSS, Geo DAB *API* (http://api.eurogeoss-broker.eu/)
  - Remote Monitoring and Smart Sensing (https://marketplace.eosc-portal.eu/services/remote-monitoring-and-smart-sensing)
  - RPrototypingLab Virtual Research Environment
- INSPIRE
- F.A.I.R. alapelvek (https://ec.europa.eu/info/sites/default/files/turning_fair_into_reality_0.pdf)
- *OGC API*

- OpenStreetMap

2.12. Sok felhasználó támogatása

A sok felhasználó támogatását két szempontból lehet vizsgálni. Egyrészt az erőforrás optimalizáció szempontjából (hardver és szoftver), másrészt pedig a felhasználó hozzáférési szintek kezelésének szempontjából.

Erőforrás optimalizáció szempontból sok felhasználó kiszolgálására úgy lehet a legjobban felkészülni, ha a tervezett rendszer nem egyetlen szoftver, hanem különálló szolgáltatók (szerverek) összekapcsolásából áll. Ezek a különálló szerverek lehetnek az

- alkalmazás szerver
- térkép szerver
- adatbázis szerver

és ezek mindegyikét tovább lehet még skálázni. Például az alkalmazás szerver szétválasztható egyes alkalmazás részek szerinti önálló szerverekre, de ebben az esetben egy erősebb belső alkalmazás API szükséges, amely képes a rész alkalmazások egymáshoz kapcsolására.

A térképszerver a kiválasztott megoldástól függetlenül is eleve skálázható, mivel a klienseket lehet úgy konfigurálni, hogy egyidejűleg több térkép szerverről kérje le a térképi állományokat és kapcsolódó mapcache szerverek alkalmazásával lehet csökkenteni a térkép szerverek terheltségét. Az elosztott térképszerverek alkalmazhatósága függ a forrás állományoktól is. Állandó forrásállományok esetén (például raszteres alaptérképek) nagyon hatékony lehet, míg dinamikus alapadatok esetén (SQL kiszolgáló) az adatbázis szervert is együtt kell a térkép szerverrel skálázni a hatékonyság tényleges növeléséhez. Itt figyelembe kell még venni azt is, hogy amennyiben az alkalmazás szerver állítja össze a lekérdezést, akkor annak tudnia kell kommunikálni az elosztott háttér erőforrásokkal, amire már nincsen triviális megoldás, jó eséllyel az alkalmazás szintjén kell felkészülni az elosztott szerverekkel való együttműködésre.

Egy másik megoldás a teljes rendszer egyben “fürtözése” és egy terhelés elosztó alkalmazása. Ez a megoldás lényegesen egyszerűbb, de egyértelműen költségesebb a létrehozása és a fenntartása is.

Amennyiben a szerver szolgáltatás egyes komponensei önálló szerverek és belső API felületeken kapcsolódnak egymáshoz úgy az egyes részek kiszolgálója szükség esetén a többitől függetlenül bővíthető ami jelentősen csökkenti a rendszer fenntartási és beszerzési költségét is. Ez a megoldás továbbá jelentősen növeli a rendszer rendelkezésre állási idejét is, mivel az egyes részek karbantartása nem feltétlenül vonják magukkal a rendszer teljes leállítását. Egy ilyen megoldás alkalmazása akkor javasolt, ha nagyjából prediktálható a felhasználó terhelés és nem kell felkészülni gyors és nagy mértékű kiszolgáló szint növekedésre.

Az *OpenBioMaps* a belső *API*-val kapcsolatot tartó önálló szerverek megoldásra épül, de egyes komponensei tetszőlegesen skálázhatóak, úgyhogy egy hibrid megoldást valósít meg.

A felhasználói hozzáférési szintek kezelése az “authentikáció” és az “authorizáció” megfelelő szabályozásán kell alapuljon, amelyet csoportok kialakításával és csoportszintű authorizáció

kezeléssel, továbbá jogosultság öröklés kezeléssel lehet támogatni. Jó példa lehet a PostgresSQL felhasználó kezelése.

3. Az interfész-kapcsolatok (*API*) szintjének lehetséges megoldásai

A 2. fejezetben tárgyalt okoknál fogva olyan API funkciók és API kezelő eszközök támogatását javasoljuk, amelyek sztenderdnek tekinthetők, vagy nagyon széles körben támogatottak. A javaslataink felhasználási területenként a következők:

OpenAPI az API felületek kialakítására

Az *OpenAPI* egy MIT licencű nyílt forráskódú termék és a legszélesebb körben elfogadott ipari szabvány új *API*-k leírására. Az *OpenAPI* használatának előnye (https://oai.github.io/Documentation/start-here.html), hogy az *API* szabatos, gépileg olvasható formátumban történő leírása lehetővé teszi, hogy automatizált eszközökkel feldolgozzák azt, ami azonnal megnyitja az utat a következő lehetőségek előtt:

- **Leírás érvényesítés és futtatás**: a leírási fájl szintaktikai ellenőrzése - megfelel-e a specifikáció egy adott verziójának és a projekt többi formázási irányelvének.
- **Adat ellenőrzés**: az *API*-n keresztül (mindkét irányba) áramló adatok helyességének ellenőrzése..
- **API dokumentáció automatikus létrehozása**: Hagyományos, ember által olvasható dokumentáció létrehozása a gépileg olvasható leírás alapján, amely mindig naprakész marad.
- **Kódgenerálás**: szerver- és kliens kód készítése bármilyen programozási nyelven, így a fejlesztőknek nem kell például adatérvényesítést végezniük vagy SDK “ragasztókódot” írniuk.
- **Grafikus szerkesztők**: Lehetővé teszi a leíró fájlok egyszerű létrehozását grafikus felhasználói felület segítségével a kézzel történő gépelés helyett.
- **Mock szerverek**: “Fake” szerverek létrehozása, amelyek példa válaszokat adnak, amelyekkel a fejlesztők és kliensek elkezdhetik a tesztelést, mielőtt egyetlen sor kódot írnának.
- **Biztonsági elemzés**: a lehetséges sebezhetőségeket feltárása az *API* tervezési szakaszában.

Ezen felül az *OpenAPI* specifikáció a következőket is biztosítja:

- **Egy nem védett formátumot**: A felhasználóknak lehetősége van beleszólni a specifikáció fejlődési irányába.
- **Fejlett eszközrendszer**: A közösségi fejlesztés következményeként az *OpenAPI* rengeteg eszközt kínál a vele való munkavégzéshez: https://openapi.tools/.
- **Gépek és emberek által egyaránt olvasható formátum**: Bár az *OpenAPI*-dokumentumok kézzel történő megírása nem a legkényelmesebb módszer, ezek egyszerű szöveges fájlok, amelyeket könnyen át lehet böngészni, ha valamit hibakeresésre van szükség.

RClone a külső fájl kapcsolatok kezelésére

Felhő alapú tárolókkal való adatkapcsolat támogatása jelentősen megkönnyíti a különböző rendszerek közötti átjárhatóságot és hatékony munkafolyamatok felépítését. A támogatott fájl kezelési kapcsolatoknak két szintje lehetséges:

1. adatok behozatala és kivitele távoli fájlrendszerekre
2. távoli fájlrendszer transzparens kezelése: adat elérések integrálása háttérfolyamatokba

Az **Rclone** egy parancssoros program felhő alapú tárhelyen található fájlok kezelésére. Funkciógazdag alternatívája a felhőszolgáltatók webes tároló felületeinek. Több mint 40 felhőalapú tárolási termék támogatja az rclone-t, beleértve az S3 objektum tárolókat, az üzleti és fogyasztói fájltárolási szolgáltatásokat, valamint a szabványos átviteli protokollokat. MIT licencű nyílt forráskódú szoftver. Többféle programozási nyelvhez is létezik hozzá "wrapper" amin keresztül programozott módon is használható. Azaz a fájlkezelés második szintjét a transzparens integrálását is támogatja.

Repozitórium kapcsolatok (adat és metaadat kapcsolatok) kezelése

A megvalósítandó projekt szempontjából fontos a kutatói adatinterfészek biztosítása, amelynek egyik legalapvetőbb eszköze a rendszerben kezelt adatok metaadatainak karbantartása és ezen metaadatok publikus szolgáltatása. Az úgynevezett diszciplináris repozitóriumok elsősorban kutatók számára kínálnak publikációk és metaadatok számára tárhelyet és kézi, vagy automatikus keresésekhez felületet. Számos ilyen szolgáltatás létezik amelyet ingyenesen lehet igénybe venni, mint például a nemzetközi Dryad (https://datadryad.org/), DataONE (https://www.dataone.org/) vagy az ELKH Kutatóhálózat által létrehozott (Dataverse alapú) Concorda (https://science-data.hu/). Ezeknek a repozitórium rendszereknek a szolgáltatásait azok *API* felületén keresztül integrálni lehet a saját alkalmazásunkba, amellyel jelentősen meg lehet könnyíteni adatok és metaadatok publikálását.

A metaadatok kezelése mind a bejövő adatok mind pedig a rendszerben rögzített adatok esetén jelentős. A metaadatoknak legalább ezekre a kérdésekre kell tudni válaszolnia:

- Mennyire ismerjük ezeket az adatokat?
- Mennyire tiszták az adatok?
- Mikor frissítették utoljára?
- Ki tartja karban az adatokat?
- Hol tárolják az adatokat?
- Mit jelentenek az egyes oszlopok?

Ezeken a kérdéseken túl további tetszőleges kérdésekre lehet a metaadatokkal válaszolni, amelyekhez szükséges valamilyen metaadat kezelő eszköz, vagy metaadat kezelő eszközök támogatási felülete. A fenti metaadat kezelő szolgáltatások integrálásának támogatásán túl a következő metaadat platformok és szabványok integrálását és használatát javasoljuk:

- Az **OpenMetadata** (https://www.open-metadata.org/) egy nyílt szabvány, amely egy központosított metaadat-tároló és -beviteli keretrendszerrel rendelkezik, amely támogatja a szolgáltatások széles köréhez tartozó csatlakozó felületeket (például Postgres, Oracle, MySQL (https://docs.open-metadata.org/connectors), *DBT* (https://docs.open-metadata.org/features#dbt-integration)). A metaadat beviteli keretrendszer lehetővé teszi, hogy bármilyen szolgáltatást testre szabjunk, vagy hozzáadjunk egyedi támogatást. A *REST API*-k lehetővé teszik az OpenMetadata integrálását a meglévő rendszerekbe. Az OpenMetadata felhasználói felület (UI) segítségével az adat használók

megtalálhatják a megfelelő adatokat a munkájukhoz, az adat előállítók pedig értékelhetik a felhasználást és a használati tapasztalatokkal finomíthatják az adatok struktúráit.

- A **Darwin Core** (https://dwc.tdwg.org/) Standard (DwC) stabil, egyszerű és rugalmas keretet kínál a biológiai sokféleségre vonatkozó, változatos és változó forrásokból származó adatok összeállításához. A Darwin Core eredetileg a Biodiverzitási Információs Szabványok (TDWG) közösség által kifejlesztett, "fejlődő, a közösség által kifejlesztett biodiverzitási adatszabvány". Alapvető szerepet játszik a nyílt hozzáférésű biodiverzitási adatok megosztásában, felhasználásában és újra felhasználásában. A *GBIF*-en keresztül elérhető több százmillió faj előfordulási adat túlnyomó többségét teszi ki.

  A gyakorlatban a Darwin Core használata egy szabványos fájlformátum, a Darwin Core Archive (DwC-A) körül forog. Ez a kompakt csomag (ZIP-fájl) összekapcsolt szöveges fájlokat tartalmaz, és lehetővé teszi az adatok közzétevőinek, hogy közös terminológiával osszák meg adataikat. Ez a szabványosítás nemcsak a biológiai sokféleségre vonatkozó adatkészletek közzétételét egyszerűsíti, hanem a felhasználók számára is megkönnyíti az adatkészletek felfedezését, keresését, értékelését és összehasonlítását, amikor a mai adat-intenzív kutatási és szakpolitikai kérdésekre keresik a választ.

- A ***Frictionless*** **Data** (https://frictionlessdata.io/) egy progresszív, nyílt forráskódú keretrendszer az adat infrastruktúra - adatkezelés, adatintegráció, adatáramlás stb. - kiépítéséhez. Az adatok közzétételére, továbbítására és felhasználására vonatkozó specifikációkat és szoftvereket kínál. A *Darwin Core*-hoz hasonlóan az adatforrások CSV fájlokban kerülnek megjelenítésre, míg az adatmodell *JSON* struktúrában kerül leírásra. A *Frictionless* Data az Open Knowledge Foundation (https://okfn.org/) egy projektje. A Frioctionless Adat Csomag a *GBIF* által is használt eszköz az adatok metaadatokkal együtt történő exportálására. A *Frictionless* Adat Csomag könnyebben kezelhető mint az XML alapú *DarwinCore*, de egymásba alakíthatóak.

- A **BiCIKL** az Európai Unió Horizont 2020 projektje (https://cordis.europa.eu/project/id/101007492), amely a kulcsfontosságú kutatási infrastruktúrák új európai induló közösségét kezdeményezi és építi ki, amely a biológiai sokféleség területén nyílt tudományos gyakorlatokat hoz létre az adatokhoz, a kapcsolódó eszközökhöz és szolgáltatásokhoz való hozzáférés biztosításával a kutatási ciklus minden egyes szakaszában és a teljes kutatási ciklus mentén. A BiCIKL új módszereket és munkafolyamatokat fog biztosítani a szakirodalomból kinyert, alacsony szintű adatok (példányok, forrás idézetek, minták, szekvenciák, taxonómiai nevek, ábrák, táblázatok, …) összegyűjtéséhez, közzétételéhez, összekapcsolásához, eléréséhez és újra felhasználásához való integrált hozzáféréshez. A BiCIKL első alkalommal biztosít hozzáférést és eszközöket az adatok zökkenőmentes összekapcsolásához és használatuk nyomon követéséhez a következő vonalon: példányok > szekvenciák > fajok > elemzések > publikációk > biológiai sokféleség tudás grafikon > újrafelhasználás. Habár még nincs kész a BiCIKL platform, de a fejlődését nyomon kell követni és amint ez lehetséges a megfelelő intefészeket biztosítani kell a projektben a BiCIKL szolgáltatások használatához, mivel várhatóan a legjobb kapcsolati lehetőséget foga biztosítani természettudományos múzeumok tudásbázisához.

Adatbázis kapcsolatok

A következő adatbázis platformok, adatbázisok és szoftverekkel való kompatibilitás megvalósítását (*API* kliens fejlesztés, szolgáltatás integrálás) javasoljuk:

Az ***OpenBioMaps*** (*OBM*) (https://openbiomaps.org) egy hazai, közösségi fejlesztésű biológiai adatbázis kezelő platform. Nyílt forráskódú, GPL licencű. Az *OBM* PDS *API*-ja lehetővé teszi autentikáció után adatok lekérdezését, adatok feltöltését és *OBM* funkciók elérését jogosultságtól függően. Egyes OBM funkciók integrálhatóak más alkalmazásokba. Ilyen a WMS/WFS alapú

térkép, a Postgres/*PostGIS* alapú adatbázis. Az *OBM* egyedi, különlegesnek számító megoldása, hogy az adatbázis struktúrái emberek számára is érthetőek és így külső adatbázis kezelő rendszereken keresztül (mint például *QGIS*) könnyen használható, miközben olyan alkalmazás és adatbázis szintű adatkezelési optimalizációkkal rendelkezik, amelyek lehetővé teszik nagy adatmennyiségek kezelését is.

Az *OBM* számos eszközt nyújt az adatgyűjtéshez és az adat kezeléséhez, mint például az OBM mobil alkalmazás vagy az *OBM* R csomag, vagy az *OBM* háttérfolyamat kezelője, amelyen keresztül bármilyen rendszeres automatizálható feladat elvégezhető az adatbázissal kapcsolatban tetszőleges programozási környezetben megvalósított eszközökkel.

Az *OpenBioMaps docker*izált rendszerben telepíthető és igény szerint tetszőlegesen skálázható, az egyes elemei önálló szerverekre telepíthetőek. Az *OBM* lehetővé teszi tetszőleges struktúrájú adatbázisok létrehozását és üzemeltetését, amelynek következménye, hogy már létező más rendszerben létrehozott adatbázisok is viszonylag könnyen átemelhetőek *OBM* szerver környezetbe.

Az *OpenBioMaps* mobil alkalmazás tetszőleges, egyéni adatgyűjtő űrlapok kezelését nyújtja akár offline állapotban is felhasználóknak és a felülete egyéni alkalmazás változatok kialakítására is alkalmas.

Az *OpenBioMaps* felhasználói nemzeti parkok, kutatóintézetek, egyetemek, természetvédelmi civil szervezetek. A hazai 10 nemzeti parkból 9-nél használják az *OBM*-et a terepi adatgyűjtések kezelésére és a biotikai adatok karbantartására és több nagy nemzetközi kutatási projektben is használják.

Az *OBM API*-n keresztül lekérdezhetők és letölthetők az adatok és metaadatok. A közvetlen integráció egy eszköze lehet a Postgres FDW, aminek a használatát támogatja az *OpenBioMaps*.

A ***GBIF*** - Global Biodiversity Information Facility - (https://gbif.org) a világ kormányai által finanszírozott nemzetközi hálózat és adat infrastruktúra, amelynek célja, hogy bárkinek, bárhol, nyílt hozzáférést biztosítson a földi élettel kapcsolatos mindenféle adatokhoz. Az adatokat a világ számos intézménye szolgáltatja; a *GBIF* információs architektúrája ezeket az adatokat egyetlen portálon keresztül teszi hozzáférhetővé és kereshetővé. A *GBIF* portálon keresztül elérhető adatok elsősorban a növények, állatok, gombák és mikrobák világszintű elterjedési adatai, valamint a tudományos nevek adatai. A prioritások közé tartozik - a részvétel előmozdítására és a partnereken keresztül történő együttműködésre helyezve a hangsúlyt - a biológiai sokféleséggel kapcsolatos adatok mobilizálása, a tudományos integritást és átjárhatóságot biztosító protokollok és szabványok kidolgozása; a különböző forrásokból származó különböző adattípusok összekapcsolását lehetővé tevő informatikai architektúra létrehozása, a kapacitásépítés előmozdítása és a jobb döntéshozatalt szolgáló elemzési eszközök fejlesztésének katalizálása. A *GBIF* arra törekszik, hogy informatikai kapcsolatokat alakítson ki a biológiai szerveződések teljes spektrumából származó digitális adatforrások között, a génektől az ökoszisztémákig, és ezeket a tudomány, a társadalom és a fenntarthatóság szempontjából fontos kérdésekkel kapcsolja össze georeferencia és *GIS*-eszközök felhasználásával. A szervezet más nemzetközi szervezetekkel, például az Catalogue of Life partnerséggel, a Biodiversity Information Standards, a Consortium for the Barcode of Life (CBOL), az Encyclopedia of Life (EOL) és a GEOSS-szal együttműködve működik. A *GBIF*-en keresztül elérhető biodiverzitási adatok száma közel kétmilliárd (2022 február 17). A *GBIF* a *GBIF API*-kon (https://www.gbif.org/developer/summary) keresztül a következő területeken nyújt szolgáltatásokat:

Nyilvántartás *API*: Az adatkészletekre, szervezetekre (pl. adatkiadók), hálózatokra és az azokhoz való hozzáférés eszközeire (technikai végpontok) vonatkozó információk létrehozására, szerkesztésére, frissítésére és keresésére szolgáló eszközöket biztosít. A regisztrált tartalom szabályozza, hogy mi kerül feltérképezésre és indexelésre a *GBIF* adatportálon, de megosztott *API*-ként más kezdeményezésekhez is használható.

Fajok *API*: Szolgáltatásokat nyújt a fajokról és magasabb rendű taxonokról szóló információk felfedezéséhez és eléréséhez, valamint a nevek értelmezéséhez és a *GBIF* portálon a fajokhoz használt azonosítók és teljes tudományos nevek megkereséséhez szükséges segéd szolgáltatásokat.

Előfordulás *API*: Hozzáférést biztosít a *GBIF* által feltérképezett és indexelt előfordulási információkhoz, valamint keresőszolgáltatásokat valós idejű kereséshez és aszinkron letöltési szolgáltatásokat nagy kötegelt letöltésekhez.

Térképek *API*: Egyszerű szolgáltatásokat nyújt a *GBIF* mobilizált tartalmának más oldalakon való megjelenítéséhez.

Hírek *API*: Szolgáltatásokat nyújt hasznos információk, például a *GBIF* mobilizált tartalmainak felhasználásával közzétett, különböző témájú tanulmányok közvetítésére.

Ezen *API*-k közül legalább a Fajok és Előfordulás *API* használatának támogatását javasoljuk.

Az **iNaturalist** (https://www.inaturalist.org/) a Kaliforniai Tudományos Akadémia és a National Geographic Society közös kezdeményezése. Egy közösségi platform természetben tett megfigyelések rögzítésére és közzétételére, amelyet amatőr megfigyelők és szakemberek egyaránt használnak. Használható kutatási minőségű adatgyűjtésre és polgári tudományos projektek létrehozására is. Az iNaturalist faj azonosítási modellje gépi tanuláson alapuló osztályozással kezdeti javaslatokat nyújt a feltöltött képek azonosítására, de az iNaturalist közösség más tagjai segítenek azonosítani és megerősíteni megfigyeléseket.

Számos iNaturalist eszköz elérhető az iNaturalist *API*-n keresztül (https://api.inaturalist.org/v1/docs/) más alkalmazásokba és folyamatokba való integráláshoz.

Az iNaturalist adatok *API*-n keresztüli lekérdezésének támogatását javasoljuk, mivel ez egy jelentős Citizen Science adatforráshoz való hozzáférést biztosít.

A **GEO** (The Biodiversity Observation Network) (http://www.earthobservations.org/) koordinálja a Global Earth Observation System of Systems (Földmegfigyelési Rendszerek Globális Rendszerének) (GEOSS) kiépítésére irányuló nemzetközi erőfeszítéseket. Összekapcsolja a meglévő és tervezett Föld-megfigyelési rendszereket, és támogatja az újak fejlesztését a környezettel kapcsolatos információk ellátásában észlelt hiányosságok esetén. Célja a Föld-megfigyelések globális nyilvános infrastruktúrájának kiépítése, amely a rendszerek és tartalomszolgáltatók rugalmas és elosztott hálózatából áll.

A **GEOSS** (https://www.earthobservations.org/geoss.php) célja, hogy összekapcsolja a környezeti adatok és a döntéstámogató eszközök előállítóit e termékek végfelhasználóival, azzal a céllal, hogy növelje a Föld-megfigyelések relevanciáját a globális kérdések szempontjából. A GEOSS célja egy olyan globális nyilvános infrastruktúra létrehozása, amely átfogó, közel valós idejű környezeti adatokat, információkat és elemzéseket hoz létre a felhasználók széles köre számára. A

Föld-megfigyelő rendszerek olyan műszerekből és modellekből állnak, amelyek célja a Föld rendszer fizikai, kémiai és biológiai aspektusainak mérése, nyomon követése és előrejelzése.

A GEOSS a téradat-infrastruktúra létrehozásához való hozzájárulásként jellemezhető. Ez egyike annak a három kapcsolódó kezdeményezésnek, amelyek az EU 7. keretprogramjának égisze alatt zajló GIGAS (GEOSS, INSPIRE és GMES an Action in Support) harmonizációs projekt tárgyát képezik. A GEOSS a GEOSS portálból, GEONETCastból és a GEO DAB szolgáltatásokból áll

A "*GEOSS portál*" (http://www.geoportal.org/) egyetlen internetes hozzáférési pontot kínál a felhasználók számára, akik a világ minden részén releváns adatokat, képeket és elemző szoftvercsomagokat keresnek. Összekapcsolja a felhasználókat a meglévő adatbázisokkal és portálokkal, és megbízható, naprakész és felhasználóbarát információkat nyújt, amelyek létfontosságúak a döntéshozók, a tervezők és a vészhelyzet-kezelők munkájához.

A GEO Discovery and Access Broker (*GEO DAB*) (https://www.geodab.net/) az elsődleges mechanizmus, amely az összes adat és információ felkutatását és elérését biztosítja. A GEO DAB a szükséges közvetítő és harmonizációs szolgáltatásokat alkalmazásprogram-interfészeken (*API*-kon) keresztül valósítja meg (http://api.eurogeoss-broker.eu/). Ezek az *API*-k lehetővé teszik az adatszolgáltatók számára az erőforrások megosztását anélkül, hogy technológiájukon vagy szabványaikon jelentős változtatásokat kellene végrehajtaniuk.

Jelenleg a GEOSS Platform több mint 150 autonóm adatkatalógust és információs rendszert közvetít, amelyek hasznosak a GEO különböző társadalmi hasznot hozó területei számára, beleértve a CAFF, Data.gov, Data.uk, EEA, *GBIF*, Iris, JRC Open Data catalog, NASA, NCAR, NOAA, OCHA HDX, RCMRD, UNEP, UNOSAT, USGS, Web Energy Services, WMO WIS és sok más adatot.

Az adatszolgáltatókat folyamatosan bővítik és közvetítik, a felhasználói igények, az adatok tematikus és földrajzi egyensúlya és a megosztott források relevanciája alapján.

A *GEONETCast* a fenntartható és költséghatékony műholdas terjesztési rendszerek globális hálózata. A rendszer rutinszerűen szállít Föld-megfigyelési (EO) adatokat és termékeket a GEO-közösség tevékenységei, kezdeményezései és zászlóshajói számára, illetve azokból. A GEONETCast Kína (CMA), az EUMETSAT és az USA (NOAA) együttműködésén alapul. Architektúrája nyitott, és más partnerek számára is befogadható.

A **GEO BON** - The Biodiversity Observation Network - (https://portal.geobon.org/home) a GEO, a The Group on Earth Observations része. A GEO családon belül a GEO BON képviseli a biológiai sokféleséget, a GEO kilenc társadalmi hasznot hozó területének egyikét. A GEO BON, a GEO Biodiverzitás-megfigyelési Hálózata a biodiverzitással kapcsolatos adatok és metaadatok GEOSS-hoz, a “Global Earth Observation System” rendszeréhez való kapcsolódásának útját építi ki. A GEO BON egy Rest *API* felületet biztosít a kezelt adatok eléréséhez: https://portal.geobon.org/api-docs.

A **BON in a Box** (https://boninabox.geobon.org/) a GEO BON része, egy testre szabható és folyamatosan frissített eszköztár. Hozzáférést biztosít a legújabb biodiverzitás-megfigyelési tervekhez, adatgyűjtési protokollokhoz, valamint adatkezelési, elemzési és jelentéstételi eszközökhöz. Technológiaátadási és kapacitásépítési mechanizmusként szolgál, amely biztosítja, a hozzáférést biodiverzitás-megfigyelési rendszerek kiépítéséhez szükséges legjobb és legmodernebb eszközökhöz és technológiákhoz. *API* felületen keresztül is használható és integrálható: https://boninabox.geobon.org/frontend/api-docs.

A GEO eszközök olvasásának és a GEO BON - pl. a Bon in Box eszközén keresztüli kapcsolódásának támogatását javasoljuk különösen tekintettel arra, hogy ezzel részben az INSPIRE kompatibilitás megvalósítását is teljesíti a rendszer.

Az **Operatív Vízhiány Értékelő és Előrejelző Rendszer** (https://aszalymonitoring.vizugy.hu/) vízhiány/aszály jellemzésére alkalmas, az ország 112 mérőállomásán mért adatokat (levegő hőmérséklet, talajhőmérséklet, talajnedvesség, relatív páratartalom, csapadék, aszály index, vízhiány) szolgáltat amely *API* felületen keresztül is elérhető (http://aszalymonitoring.vizugy.hu/api.php). Az adatok szabadon felhasználhatóak.

Az aszály monitoring adatok integrált elérését, mint elemzés támogató adatforrást javasoljuk beépíteni a rendszerbe.

A **Chirovox** (https://chirovox.org) egy hazai kezdeményezésű nemzetközi nyílt denevér hang gyűjtemény és megosztó adatbázis (Görföl és mtsi 2022[5]), amely *OpenBioMaps* alapokon üzemel és így rendelkezésre állnak benne az *OpenBioMaps* API által nyújtott szokásos funkciók. Az Ökológiai Kutatóközpont üzemelteti.

A Chirovoxban elérhető adatok a hazai denevérfaunára vonatkozó fontos Citizen Science adatokat biztosíthatnak és az *OBM API* támogatása miatt nem jelentenek extra terhet a fejlesztés során így a Chrivox adatbázis publikus adatainak integrálását javasoljuk beépíteni a rendszerbe.

Magyarország Edényes Növény Adatbázisa (http://floraatlasz.uni-sopron.hu) a legnagyobb hazai nyílt növényes adatbázisa amely *OpenBioMaps* alapokon működik, így rendelkezésre állnak benne az *OpenBioMaps API* által nyújtott szokásos funkciók. A Nyugat-Magyarországi egyetem üzemelteti.

A Flóraatlasz *OBM API*-n keresztüli integrálását javasoljuk beépíteni a rendszerbe.

Az **Eurázsiai Kurgán Adatbázis** (https://openbiomaps.org/projects/kurgan) az Ökológiai Kutatóközpont munkatársai által üzemeltetett hazai kezdeményezésű, de nemzetközi részlegesen nyitott kunhalom adatbázis (Deák 2019[6], Deák és mtsi 2019[7]) amely *OpenBioMaps* alapokon működik, így rendelkezésre állnak benne az *OpenBioMaps API* által nyújtott szokásos funkciók az adatok automatizált olvasására és szerkesztésére.

---

[5]. Görföl T, Huang JC, Csorba G, Győrössy D, Estók P, Kingston T, Szabadi KL, McArthur E, Senawi J, Furey NM, Tu VT, Thong VD, Khan FAA, Jinggong ER, Donnelly M, Kumaran JV, Liu JN, Chen SF, Tuanmu MN, Ho YY, Chang HC, Elias NA, Abdullah NI, Lim LS, Squire CD, Zsebők S. ChiroVox: a public library of bat calls. PeerJ. 2022 Jan 13;10:e12445. doi: 10.7717/peerj.12445. PMID: 35070499; PMCID: PMC8761365.

[6]. Deák B. (2019): Eurasian Kurgan Database. doi: 10.18426/10.18426/obm.3mbbectm2bmg

[7]. Deák, Balázs, Tóth, Csaba Albert, Bede, Ádám, Apostolova, Iva, Bragina, Tatyana M., Báthori, Ferenc and Bán, Miklós. "Eurasian Kurgan Database – a citizen science tool for conserving grasslands on historical sites" Hacquetia, vol.18, no.2, 2019, pp.179-187. doi: doi.org/10.2478/hacq-2019-0007

A Kurgán adatbázis a hazai kunhalmok recens állapotára vonatkozó legnagyobb adatbázis, amelynek OBM API-n keresztüli integrálását javasoljuk beépíteni a rendszerbe.

Az **izeltlabuak.hu** (izeltlabuak.hu) Magyarország legnagyobb nyílt hozzáférésű közösségi ízeltlábú adatbázisa. Károlyi Balázs üzemelteti.

Habár nincsen publikus *API* felülete, de megfontolásra javasoljuk, hogy a fejlesztés során próbáljanak a fejlesztők egy közösen kialakított adatkapcsolati réteget létrehozni és állandó adatkapcsolaton keresztül integrálni a rendszerbe.

4. A felhasználói felület (*GUI*) kialakításának alternatívái

A felhasználó felület ma már nem kell az alkalmazás környezet szerves részét képezze, sőt kifejezetten javasolt az alkalmazás réteg által nyújtott API felületekre épülő felhasználói felület tervezésében gondolkodni, mivel ez a megoldás egyúttal megkönnyíti további, akár asztali *GUI* felületek kialakítását is. Az alkalmazás réteggel való kapcsolat egyik legkézenfekvőbb módja az *OpenAPI* használata.

A front-end webfejlesztés az adatok grafikus felületté alakításának folyamata CSS, HTML és *JavaScript* segítségével, azért, hogy a felhasználók láthassák az adatokat és kapcsolatokat alakíthassanak ki velük.

A front-end keretrendszerek segítik a fejlesztők munkáját, meggyorsíthatják a fejlesztést, rugalmas kész eszközöket biztosítanak, de egyúttal korlátokat is támasztanak amelyek meghatározzák majd a rendszerünk működési módját, megjelenését és teljesítményét.

A front-end keretrendszerek jellemzően a következő megjelenítéssel kapcsolatos komponenseket tartalmazzák:

- *Rács, amely* megkönnyíti a webhely tervezési jellemzőinek kialakítását.
- Jól definiált *betűtípusok*, amelyek a használati cél alapján különböznek (eltérő tipográfia a fejlécekhez, szemben a bekezdésekkel).
- Előre elkészített *komponensek*, például gombok, navigációs sávok és oldalsó panelek.

A projekt web ui fejlesztés szempontból potenciálisan érdekes JS keretrendszerek

A **React** a Facebook által kifejlesztett és létrehozott nyílt forráskódú keretrendszer. A React legfontosabb jellemzője egy virtuális dokumentum objektum-modell (DOM) egyirányú adatkötéssel. Leginkább az úgynevezett single page alkalmazások (SPA) számára jó választás. A virtuális DOM funkcionalitásnak köszönhetően kiváló teljesítményű és az egyik legkönnyebben tanulható keretrendszer, ami miatt viszonylag könnyű fejlesztőt találni hozzá. Mivel a React egy "*library*", más keretrendszerekkel ellentétben nem tartalmaz néhány fontos funkciót (állapotkezelés, útválasztás, *API* kommunikáció), viszont éppen ezért más könyvtárakkal való együttműködésben jó. A React összetevői újra felhasználhatóak.

Az **Angular** egy TypeScriptre épülő nyílt forráskódú keretrendszer, amelyet a Google hozott létre a technológia növekvő igényei és és a hagyományos elképzelések közötti távolság csökkentésének érdekében. Az Angular abban különbözik legjobban a React-tól, hogy kétirányú adatkötési funkcióval rendelkezik. Ez azt jelenti, hogy biztosítja a modell és a nézet valós idejű szinkronizálását, azaz, a modellben bekövetkező bármilyen változás azonnal megjelenik a nézetben, és fordítva. Komponens alapú architektúrája van. Ezt a keretrendszert progresszív webes alkalmazások, valamint többoldalas alkalmazások fejlesztésére is lehet használni. Nagymértékben tesztelhető / újrafelhasználható / kezelhető alkalmazások készíthetők vele.

Manapság az egyik legegyszerűbb keretrendszer a **Vue.js** amely nyílt forráskódú, közösségi fejlesztésű és MIT licencű keretrendszer. Az Angularhoz képest sokkal egyszerűbb, de több tulajdonsága is van, amelyek miatt annak jó alternatívája lehet. A Vue virtuális DOM, komponensalapú architektúrájú és kétirányú kötés jellemzi, ami a nagy sebességű teljesítmény alapja: mindez megkönnyíti a kapcsolódó komponensek frissítését és az adatváltozások követését, ami minden olyan alkalmazás esetében kívánatos, ahol a valós idejű frissítésekre szükség van. Elsősorban single page alkalmazások fejlesztéséhez javasolt.

Az **Ember.js** egy 2011-óta fejlesztett komponens alapú nyílt forráskódú keretrendszer. Kétirányú adatkötést is használ, ami az Angularhoz hasonló. Kifogástalanul kezeli a kortárs technológiák iránti növekvő igényt. Az Ember.js segítségével hatékony kialakítású és sokrétű webes és mobil alkalmazásokat lehet építeni. Mindazonáltal ez a keretrendszer az egyik legnehezebben megtanulható webes felhasználói felület keretrendszer a hagyományos és merev felépítése miatt, azaz kevesebb fejlesztő található aki tud ilyen keretrendszerben dolgozni. Egyszerű Ajax-funkcionalitások szkripteléséhez és felhasználói felületek építéséhez a keretrendszer nem biztos, hogy alkalmas, de jól alkalmazható reszponzív felhasználói felülettel rendelkező korszerű alkalmazások kialakításához.

A **jQuery** az egyik első front-end keretrendszer, amelyet 2006-ban vezettek be. Nyílt forráskódú. A jQuery nemcsak egyszerű használatot kínál, hanem csökkenti a széles körű *JavaScript*-kódok szkriptelésének igényét is.

Alapvetően a jQuery-t a DOM és a CSS manipuláció működtetésére, valamint a webhely interaktivitásának és funkcionalitásának növelésére használják.

Bár a jQuery a jQuery Mobile lehetővé teszi a fejlesztők számára, hogy a HTML5-alapú *UI*-sémával natív mobilalkalmazásokat hozzanak létre. A kiválóan kezeli böngészők közötti átjárhatóságot. Sok modern keretrendszerrel ellentétben a jQuery nem rendelkezik adatréteggel, így mindig közvetlenül a DOM-hoz kell hozzáférni és manipulálni, ami bonyolultabbá teszi a folyamatokat.

Az **alpine.js** a jquery alternatívája lehet. Támogatja a kétirányú kötést, VUE szerű szintaxisa van, de a DOM elemekkel dolgozik és nem igényel template kialakítást, hanem direktben behelyezhető a html kódba. Rendkívül rugalmas, minden megkötés nélküli, de mégis nagyon egyszerű.

A **Semantic UI** CSS és *JS* kombináló keretrendszer amely a LESS és a jQuery támogatására épül. Természetes nyelvet használ, amely az egész kódot önmagyarázóvá teszi. A keretrendszer habár viszonylag újszerű az ökoszférában, de feltűnő felhasználói felületével, egyszerű funkcionalitásaival és funkcióival az egyik legnépszerűbb front-end keretrendszerré vált a piacon elérhető összes többi között.

A webes és mobil alkalmazások építésétől kezdve a progresszív webes alkalmazásokig (PWA) egyszerűséggel képes kezelni a dinamikus és az egyszerű fejlesztéseket is.

A **Backbone.JS** egy ingyenes, nyílt forráskódú *JavaScript* könyvtár, MIT szoftverlicenc alatt használható. A Backbone.js az MVC/MVP fejlesztési koncepciót követi. Azaz az adatokat olyan modelleként reprezentálja, amelyek létrehozhatók, érvényesíthetők, megszüntethetők és elmenthetők a szerverre. Ezek a modellek lehetővé teszik a kulcs-érték kötést és az egyéni eseményeket: minden alkalommal, amikor egy *UI*-akció bizonyos változásokat hoz egy modell valamelyik attribútumában, a modell egy változás eseményt hoz létre. A változás átvihető a modell állapotát tükröző összes Nézetre, így azok ennek megfelelően reagálhatnak, és az új adatokkal újra megjeleníthetik magukat. A Backbone.js függvényekből álló gazdag *API*-t biztosít a kliensoldali webalkalmazások összeállításához, deklaratív eseménykezelést a nézetekhez, és lehetővé teszi, hogy mindezt egy *REST*ful *JSON* interfészen keresztül összekapcsoljuk a meglévő *API*-val. Leginkább single page alkalmazások létrehozásához alkalmas.

A **Tailwind CSS** alapvetően egy segédprogram-alapú CSS keretrendszer egyéni felhasználói felületek gyors létrehozásához. Igény szerint testre szabható, alacsony szintű CSS-keretrendszer, amely olyan építőelemeket biztosít, amelyek elsősorban a megjelenésre fókuszálnak a stilizálandó elem funkcionalitásával szemben (pont az ellentettje a Bootstrap-nak).

A **Pure CSS** inkább könyvtár mint keretrendszer egyedi dizájn kialakításához. Egy minimalista eszközkészlet, ami lehetővé teszi a tetszőleges egyéni bővítést.

Habár a teljes fejlesztői eszköztárak és környezetek használata egyszerűnek tűnik, mindegyiknek vannak fókusz területei és így egyúttal felhasználási határai is. A kódot rendszerspecifikussá teszik ami miatt a keretrendszerek későbbi lecserélése óriási feladat. Általában stabil és gyors fejlesztést hoznak a projekt kezdetén, de későbbi, bonyolultabb a rendszer használatának határán lévő funkciók esetén viszont számos nehezen megoldható hibát is eredményezhetnek. A fejlesztési költségük nagy rendszerek esetén magasabb lehet mint a “vanilla” (keretrendszerek nélküli) fejlesztésnek. Emiatt kifejezetten nem javasoljuk egyik nagy *JS* és *JS* CSS kombinált keretrendszer használatát sem, hanem, csak elemi eszközöket kínáló megoldások használatát, mint az Alpine JS, JQuery, Pure CSS, mivel ezekkel a fejlesztési eszközökkel alacsonyabb szintű (kevésbé rétegzett) kódokat lehet írni, ami a hosszú távú fejlesztés illetve a közösségi fejlesztés támogatása miatt is fontos. Ennek ellenére egyes egyéni komponensként létrehozható *UI* funkció kialakítása esetén - kifejezetten single page alkalmazásként megjelenő alkalmazásra gondolva előnyös lehet valamelyik *JS* keretrendszer használata (például a Vue.js Tailwind CSS-el).

Progressive Web Alkalmazások (PWA)

A Progressive Web alkalmazások alapvetően webes alkalmazások, amelyek a progresszív bővítéssel új képességeket jelenítenek meg a modern böngészőkben. A *service workers* és a *web app manifest* segítségével a webalkalmazás megbízhatóvá és telepíthetővé válik bármilyen eszközön egyetlen kódbázis segítségével, de szükség esetén hagyományos web alkalmazásként is képesek működni. A *WebAssembly* bevezetésével a fejlesztők más ökoszisztémákat, például a C, a C++ és a Rust fejlesztéseket is kihasználhatják, és így korábbi fejlesztések munkáját és képességeit a webre is átültethetik.

A telepített progresszív webes alkalmazások a böngésző lapja helyett egy önálló ablakban futnak. A felhasználó kezdőképernyőjéről, dokkolójából, feladatsorából vagy polcáról indíthatók. Lehetőség van az eszközön való keresésükre és az alkalmazásváltóval való ugrásra közöttük, így olyan érzést keltenek, mintha annak az eszköznek a részei lennének, amelyre telepítették őket.

Egy webes alkalmazás telepítése után új képességek nyílnak meg a felhasználó számára.

Elérhetővé válnak az általában a böngészőben történő futtatáskor fenntartott billentyűparancsok, továbbá a progresszív webalkalmazások regisztrálhatnak, hogy más alkalmazásoktól fogadjanak el tartalmat, vagy hogy alapértelmezett alkalmazásként kezeljenek különböző típusú fájlokat.

A *service worker*ek lehetővé teszik akár az offline munkát is az ilyen alkalmazásokkal.

Amennyiben felmerül igény a fejlesztett rendszerhez kapcsolódó mobil alkalmazás fejlesztésére úgy azt PWA alkalmazásként javasoljuk elkészíteni, amennyiben ennek nincs technikai kizáró oka (PWA-ban nem elérhető eszközök kezelése), mivel a PWA alapú mobil alkalmazás fejlesztési ideje és költsége jelentősen kedvezőbb mint a mobil specifikus platformok használata, ráadásul az alkalmazás egyúttal böngészőben is elérhető lesz! PWA alkalmazás fejlesztésnél is használhatók *JS* keretrendszerek amelyek használatára nem teszünk javaslatot, mert egyrészt nagyon dinamikus fejlődés jellemzi a területet, másrészt a megvalósítandó alkalmazás függvényében teljesen más eszközöket érdemes preferálni a vanilla *JS*- től kezdve a *Node.JS*-en át a React-ig és az Angular-ig.

5. Alkalmazás réteg megvalósítása

Javaslatunk szerint a felhasználói felületet kiszolgáló alkalmazás a megvalósítandó feladatokat figyelembe véve ne egyetlen egységes alkalmazás legyen, hanem egymással együttműködni képes komponensekből felépülő alkalmazás réteg. Az alkalmazás réteg *API* felületeken keresztül tudjon kommunikálni az *UI* réteggel és a saját komponenseivel. A következőkben ezt a szempontot figyelembe véve teszünk javaslatot az alkalmazás réteg fejlesztésére.

Programozási környezetek és keretrendszerek

PHP

A PHP egy szerver oldali nyelv. A legtöbb adatbázis kapcsolatokra épülő adatmegjelenítő rendszert PHP-ban fejlesztik. Előnye gyors és olcsó fejlesztés, nagy sebesség, skálázhatóság, és stabilitás. Számos keretrendszer és fejlesztői eszközkészlet létezik amelyek a PHP nyelvre épülnek és megkönnyíthetik a fejlesztést.

A **Laravel** egy ingyenes nyílt forráskódú PHP keretrendszer amely a modell-nézet-vezérlő (MVC) architektúrális mintát követő, *Symfony*-alapú webes alkalmazások fejlesztésére készült. A Laravel néhány jellemzője a moduláris csomagolási rendszer egy dedikált függőség kezelővel, a relációs adatbázisok elérésének különböző módjai, az alkalmazások telepítését és karbantartását segítő segédprogramok. Kifejezetten nagy és komplex rendszerek fejlesztésére ajánlott használni. A Laravel forráskódja a GitHubon található, és a MIT licensz feltételei szerint használható. A Laravel kiválóan együttműködik a VUE.js-el.

A ***Symfony*** egy népszerű PHP webes alkalmazás keretrendszer és egy újrafelhasználható PHP komponensekből/könyvtárakból álló fejlesztői készlet, stabil fejlesztői környezet. Innovatív és könnyen használható munkakörnyezet, köszönhetően a más környezetekben létrehozott megoldások integrálásának, mint például a függőségi injektálás (a Java-ból átvett) és az olyan speciálisan kifejlesztett megoldásoknak, mint a Web Debug Toolbar vagy a Web Profiler. Végezetül, a *de facto* szabványok elfogadásával a *Symfony* nem korlátozza a fejlesztőt a

környezetére, hanem lehetővé teszi, hogy kiválassza a használni kívánt szoftver komponenseket.

A **CakePHP** egy nyílt forráskódú webes keretrendszer. Hierarchikus modell-nézet-vezérlő (HMVC) megközelítést követ, PHP nyelven íródott és a Ruby on Rails koncepcióit követi. Kiemelendő a biztonsággal kapcsolatos tulajdonságai miatt. MIT licensz alatt terjesztik, a Cake Software Foundation, Inc. fejleszti. Kisebb és nagyobb projektek fejlesztésére is alkalmas.

További alkalmas PHP web fejlesztési keretrendszerek lehetnek a Zend, Yii, vagy a CodeIgniter. Amennyiben a fejlesztők szeretnének PHP keretrendszert használni, úgy elsősorban a *Symfony* használatát javasoljuk a népszerűsége és magas fokú interoperabilitás támogatása és jobb skálázhatósága miatt, de mindenképpen érdemes a fejlesztői preferenciákat figyelembe venni.

Python

A **Django** egy Python-alapú, ingyenes és nyílt forráskódú webes keretrendszer, amely a model-template-views (MTV) architektúrális mintát követi. Fenntartója a Django Software Foundation (DSF). A Django elsődleges célja, hogy megkönnyítse az összetett, adatbázis-alapú weboldalak létrehozását. A keretrendszer hangsúlyt fektet az újrafelhasználhatóságra és a komponensek "csatlakoztathatóságára", a kevesebb kódra és a gyors fejlesztésre. A Django egy opcionális adminisztrációs - létrehoz, olvas, frissít és töröl felületet is biztosít, amelyet dinamikusan generál és az admin modelleken keresztül konfigurálható.

További, a rendszer fejlesztése szempontjából figyelembe vehető Python alapú web fejlesztésre alkalmas keretrendszerek többek között a Flask, CherryPy, Pyramid, Grok, TurboGears.

Amennyiben webes back-end fejlesztés Pythonban történik úgy javasoljuk valamelyik Python alapú web fejlesztő keretrendszer használatát. Népszerűsége és ismertsége miatt leginkább a django-t.

JavaScript

A **Node.js** egy nyílt forráskódú *JavaScript* keretrendszer, vagy még inkább *JavaScript* futtatási környezet, amely a web böngészőn kívül működik és alkalmas webes fejlesztési környezetek futtatására (például Express.js). Elsősorban valós idejű esemény alapú alkalmazások fejlesztéséhez javasolt a használata. Nem javasoljuk elsődleges rendszernek az alkalmazás kialakításához, legfeljebb specifikus különálló feladatok (önálló komponensek) megvalósításához.

Ruby

A **Ruby on Rails**, vagy Rails egy szerveroldali webes alkalmazás keretrendszer, amely Ruby nyelven íródott a MIT licenc alatt. A Rails egy modell-nézet-vezérlő alapú keretrendszer, amely alapértelmezett struktúrákat biztosít egy adatbázis, egy webszolgáltatás és weboldalak számára. Az egyik legjobb teljesítményű webes keretrendszer. További Ruby alapú webes fejlesztési

keretrendszerek még például a Sinatra és a Grape, de a népszerűsége és robusztussága miatt a Rails használatát javasoljuk elsődleges fejlesztő környezetként és nagyobb önálló komponensek elkészítéséhez is javasoljuk.

A fenti keretrendszer használati javaslatok már eleve programozási nyelvek szerint vannak csoportosítva, habár a programozási nyelvnek nem kell meghatároznia az alkalmazás működését. A fentieken túl figyelembe lehet venni a Java és Rust nyelveket is mint webes fejlesztésre kiválóan alkalmas programozási nyelveket. Fejlesztési költséghatékonyság és közösségi támogatás és így hosszú távú fejlesztés támogatás szempontjából elsősorban mégis PHP nyelv használatát javasoljuk elsődleges programozási nyelvnek igény szerint kiegészítve Ruby, Python, *JS*, Java, Rust, vagy egyéb nyelven írt komponensekkel (például meglévő programkód felhasználása esetén).

Javaslatok az alkalmazás komponensekre

Adminisztratív komponensek

- Jogosultság kezelő felület
    - Belső jogosultság kezelés
    - Külső alkalmazások jogosultság kezelő rétege
- Adatfeltöltés konfiguráló felület
- Térképszerver konfiguráció
- Fájl kezelő
- Dokumentum szerkesztő / riport kezelő
- Háttérfolyamatok kezelője
- Pluginek, modulok kezelője
- Tömeges adat importáló felület
- Frissítés kezelő
- Fajnév karbantartó (lásd még értékkészlet lista karbantartás)
- Adattábla kezelő karbantartó
- SQL függvények, triggerek kezelője
- SQL konzol
- Érték készlet listák karbantartása (egyes adatmezők választható elemeinek készlete)
- Dinamikus riport definiálási felület
- Auditációs felület (felhasználói tevékenység követés)

Felhasználói komponensek

- Adat leválogatás szövegesen vagy térképpel
- Adat lap
- Adatmódosító felületek
- Adatváltozások megtekintése
- Tömeges adatátnézés (táblázatok, összefoglalók)
- Térképi réteg kiválogató
- Dokumentum/riport kereső
- Adatfeltöltő felületek (webes űrlap, fájl feltöltés, külső alkalmazások)
- Adatmegosztó felületek
- Adatelemzés (R felület)
- Adatelemzés, közös online munkatérben (Jupyter felület)
- Riportok készítése

6. Térinformatikai környezet (*GIS*) megoldási lehetőségeinek ismertetése

Jelentős a webes térképszerver kliens könyvtárak

- *OpenLayers*
- Leaflet
- ESRI ArcGIS *API* for JS (ArcGIS Online)
- Cesium
- NASA Web World Wind (WorldWind)
- MapBox
- Google Maps JS *API*

A Nasa WorldWind és a Cesium elsősorban alapvetően 3D megjelenítésre fókuszálnak. A Mapbox GL JS kifejezett erőssége a 3D térképi megjelenítés a kiváló minőségű 2D térképek mellett. Az ArcGIS leginkább csak az ArcGIS ökoszisztémában használható, de nem tartalmaz ebben az esetben semmi olyan plusz tudást amit a rendszerfüggetlen vetélytársai ne nyújtanának. A Google Maps JS *API* használatát szintén nem indokolja semmilyen projekt igény ami miatt megérné ezt a (bizonyos használati mennyiség felett már fizetős) kereskedelmi szolgáltatást igénybe venni. További ellenérv bármelyik további kereskedelmi *API* használatával szemben is (pl.: Bing, Yahoo), a szolgáltatótól való nagy fokú függőség.

A magas fokú *GIS* képességei, rugalmassága és a nyílt forráskódú közösségi fejlesztése miatt az *OpenLayers* használatát javasoljuk elsődleges webes *GIS* kliensként. Továbbá javasoljuk a Leaflet használatát is olyan önálló komponensek esetén (például publikus térképi felületek) amelyek csak adatmegjelenítést végeznek WMS vagy GeoJSON forrásra támaszkodva, mivel a Leaflet lényegesen kisebb és egyszerűbb mint az *OpenLayers*, de nagyon gyors és szép térképek készíthetők vele.

Népszerű Térkép szerviz szerver megoldások

A ***Mapserver*** (https://www.mapserver.org) vagy az eredetére (University of Minnesota) is utaló hosszabb nevén *UMN MapServer* egy nyílt forrású (MIT stílusú licencelésű) C/C++ nyelven írt FastCGI/CGI (Common Gateway Interface) térinformatikai fejlesztő környezet, amely képes internetes térinformatikai alkalmazások felépítésére Windows, Linux és MacOS X környezetben is. A *MapServer* számos bejövő adatformátumot képes kezelni, néhány fontosabb ezek közül:

- MapInfo Files
- KML
- Oracle Spatial
- Geography Markup Language (GML)
- ESRI Binary Coverages (ADF)
- ESRI ArcSDE (SDE)
- ESRI Personal Geodatabase (MDB)
- ESRI Shapefiles (SHP)
- MySQL MYGIS Format
- *PostGis*/*PostgreSQL*
- GeoJson
- GeoPackage

Az úgynevezett Mapfile definiálja a kapcsolatot az objektumok között, megmutatja a *MapServerver*nek, honnan olvashatja ki az adatokat, és megmondja, hogyan kell azokat megjeleníteni. A Mapfile egy szöveges fájl, amelynek elkészítése nem egyszerű feladat, különösebb komplexebb layer struktúra és kartográfia kialakítása mellett. A *MapServer* telepítésével együtt általában egy Apache webszerver is telepítésre kerül. A *MapServer* projektet CGI változók és template fájlok segítségével lehet vezérelni.

A MapScript a *MapServer* szkript interfésze, amely webes vagy önálló alkalmazások fejlesztésére szolgál. A MapScript lehetővé teszi, hogy közvetlenül a *MapServer* objektumaival programozzunk ahelyett, hogy a *MapServer*rel a CGI és a Mapfile segítségével lépnek kapcsolatba.

A **Mapbender** PHP/*JavaScript* alapú *OGC* kompatibilis nyílt forráskódú webes térinformatikai keretrendszer intuitív és nagy teljesítményű WebGIS alkalmazások létrehozására. A *CMS*-hez hasonlóan a dizájn, a funkciók köre vagy a jogok kezelése is kényelmesen, a böngészőn keresztül módosítható. Az alkalmazási terület az önkormányzati várostérkép-szolgáltatásoktól a rendkívül összetett, integrált, speciális alkalmazásokig terjed. A MapBender elosztott WMS (webes térkép szolgáltatás) megjelenítésére, szerkesztésére és kezelésére szolgál. Magukat a térképeket a szerver szoftver generálja. Ebből a szempontból a Mapbender egy kliensszoftver. A kliens felületeket a Mapbender kiszolgálón lévő PHP szkriptek generálják dinamikusan.

Az **Oracle Mapviewer** (MapViewer) egy programozható eszköz az Oracle Spatial vagy az Oracle Locator (más néven Locator) által kezelt térbeli adatok felhasználásával készült térképek megjelenítésére. A MapViewer olyan eszközöket biztosít, amelyek elrejtik a térbeli adatok lekérdezésének és a térképi megjelenítésnek a bonyolultságát, miközben a haladóbb felhasználók számára testre szabható lehetőségeket biztosít. Ezek az eszközök platformfüggetlenül telepíthetők, és úgy tervezték őket, hogy integrálhatók legyenek a térkép-rendező alkalmazásokkal. Főbb részei a következők:

- Térkép-renderelő „engine”: Feladata az adatbázisban tárolt információkból képernyőn megjeleníthető (raszteres) térkép-részletek generálása. Többféle formátumban tud gyártani: GIF, JPEG, SVG, PNG.
- Térkép definíciók: Adatbázisban tárolt (XML) adatok. Ezek az adatok írják le, hogy mi legyen a térképen, és az hogyan nézzen ki (vonalak vastagsága, színe, stb.).
- Programozói interfészek: Segítségükkel elérhetőek a MapViewer szolgáltatásai. A támogatott interfészek a következők: Java, PL/SQL, *JavaScript*, és XML.
- Oracle MapBuilder: Grafikus segédeszköz, mely a térkép definíciók létrehozását, és kezelését könnyíti meg. Ezen kívül számos hasznos funkciója van, például shapefile-ok importálása, metaadatok importálása és exportálása, stb.
- Az Oracle MapViewer (Oracle Fusion Middleware MapViewer) letölthető az Oracle honlapjáról, és használható az OTN Developer Licence keretén belül.

Az **Azure Maps** olyan térinformatikai szolgáltatások és SDK-k gyűjteménye, amelyek “élő” térképi adatok felhasználásával földrajzi kontextust biztosítanak a webes és mobil alkalmazások számára. Az Azure Maps a következő szolgáltatásokat biztosítja:

- *REST API* vektoros és raszteres térképek többféle stílusban és műholdképek megjelenítéséhez.
- Létrehozó szolgáltatásokat a saját térképi adatokon alapuló térképek létrehozásához és megjelenítéséhez.

- Keresési szolgáltatások címek, helyek és érdekes pontok kereséséhez világszerte.
- Különböző útvonalválasztási lehetőségek; például pont-pont, többpontos, többpontos optimalizálás, izokron, elektromos jármű, haszongépjármű, haszongépjármű, forgalom által befolyásolt és mátrix útvonalválasztás.
- Forgalomáramlás nézet és incidens nézet, a valós idejű forgalmi információkat igénylő alkalmazásokhoz.
- Időzóna- és geolokációs szolgáltatások.
- Magassági szolgáltatások digitális magassági modellel.
- Geofencing szolgáltatás és térképi adattárolás, Azure-ban tárolt helymeghatározási információkkal.
- Helymeghatározási intelligencia a térinformatikai elemzések révén.
- Emellett az Azure Maps szolgáltatásai a webes SDK-n és az Android SDK-n keresztül is elérhetők. Ezek az eszközök segítenek a fejlesztőknek olyan megoldások gyors kifejlesztésében és skálázásában, amelyek helyinformációkat integrálnak az Azure megoldásokba.

Ingyenes Azure Maps-fiók regisztrációval el lehet kezdeni a fejlesztést, de éles környezetben már fizetni kell az igénybe vett szolgáltatásokért.

A ***GeoServer*** egy közösségi fejlesztésű, nyílt forráskódú, Java nyelven írt térinformatikai szerver alkalmazás, amely lehetővé teszi a felhasználók számára a térbeli adatok megosztását, feldolgozását és szerkesztését. Az interoperabilitásra tervezték, és nyílt szabványok segítségével bármely nagyobb térbeli adatforrásból származó adatok közzétételére alkalmas. A *GeoServer* nagyon egyszerűen biztosítja meglévő információk összekapcsolását olyan virtuális földgömbökkel, mint a például a Google Earth és a NASA World Wind, valamint az olyan webalapú térképekkel, mint az *OpenLayers*, a Leaflet, a Google Maps és a Bing Maps. A *GeoServer* az Open Geospatial Consortium (*OGC*) Web Feature Service (WFS) szabványának referencia implementációjaként működik, és a Web Map Service, Web Coverage Service és Web Processing Service specifikációkat is megvalósítja.

A *GeoServer* számos adatformátumot olvas, többek között:

- *PostGIS*
- Oracle Spatial
- ArcSDE
- MySQL
- MongoDB
- Apache Solr
- Shapefiles
- GeoTIFF
- GTOPO30
- ECW, MrSID
- JPEG2000

A *GeoServer* tartalmaz egy integrált *OpenLayers* klienst az adatrétegek előnézetéhez.

A *GeoServer* továbbá támogatja a térbeli adatok hatékony közzétételét a Google Earth-be hálózati hivatkozások segítségével, KML használatával. A Google Earth kimenethez a fejlett funkciók közé tartoznak az egyéni felugró ablakok, az idő- és magassági megjelenítések, valamint a "szuper-overlayek" sablonjai.

A ***QGIS Server*** egy nyílt forráskódú WMS 1.3, WFS 1.0. 0 és WCS 1 1.1. 1 implementáció, amely emellett fejlett kartográfiai funkciókat is tartalmaz a tematikus térképezéshez. A *QGIS Server* egy C++ nyelven írt FastCGI/CGI (Common Gateway Interface) alkalmazás, amely egy webszerverrel (pl. Apache, Lighttpd) működik együtt.

A *QGIS Server* webes térkép-szolgáltatást (WMS) biztosít, amely ugyanazokat a könyvtárakat használja, mint a *QGIS* asztali alkalmazás. A *QGIS* Desktopban létrehozott térképek és nyomtatási sablonok webes térképként is közzétehetők, egyszerűen a *QGIS* projekt fájlnak a kiszolgáló könyvtárba történő másolásával. Az így kapott webes térképek pontosan ugyanúgy néznek ki, mint az asztali alkalmazásban.

Az **ArcGIS** Server az Esri által *GIS* rendszer központi szerver szoftvere. Az ArcGIS Server a *GIS* webes szolgáltatások, alkalmazások és adatok létrehozására és kezelésére szolgál. Az ArcGIS Server-t jellemzően az Esri szolgáltatásorientált architektúráján (SOA) belül, vagy, felhő alapú számítástechnikai környezetben telepítik. Az ArcGIS Server szolgáltatásai térképezési és *GIS* funkciókat biztosítanak az ArcGIS Online for Esri Web és az olyan kliens alkalmazásokon keresztül, mint az ArcGIS Desktop, ArcLogistics, az ArcGIS.com Viewer, ArcGIS Explorer, ArcGIS Explorer Online, ArcGIS Viewer for Flex, ArcGIS Mapping for SharePoint, Esri Business Analyst Online (BAO), valamint az ArcGIS for iOS vagy BAO for iOS segítségével készült alkalmazások. Számos harmadik féltől származó alkalmazás is rendelkezik engedéllyel az ArcGIS Server szolgáltatásainak használatára.

Az ArcGIS Server elérhető a Microsoft Windows .NET Framework és a Java Platform számára.

Az ArcGIS Server-t a szoftverfejlesztők és a webfejlesztők használják webes, asztali és mobil alkalmazások létrehozására. Az Esri a fejlesztőknek alkalmazásfejlesztési keretrendszert (ADF) és alkalmazásprogramozási interfészt (*API*) biztosít, többek között ArcGIS *API* for *JavaScript*, ArcGIS *API* for Flex, ArcGIS *API* for Microsoft Silverlight/WPF, ArcGIS *API* for iOS, BAO *API*, BAO for iOS, valamint az ArcGIS Mobile szoftverfejlesztési készletet (SDK), és az ArcGIS Server *REST* és SOAP *API*-kat.

Az ArcGIS Server három funkcionális kiadásban érhető el: Basic, Standard és Advanced, amelyek közül a Advanced kiadás nyújtja a legtöbb funkciót. Az ArcGIS Server Basic kiadás elsősorban többfelhasználós geoadatbázisok és geoadat-szolgáltatások kezelésére szolgál. Mind az ArcGIS Server Standard, mind az Advanced kiadás a következő típusú webszolgáltatásokat támogatja: Feature (webes szerkesztéshez), Geodata (geoadatbázisok replikációjához), Geocode (címek/települések kereséséhez és térképen való megjelenítéséhez), Geometry (geometriai számításokhoz, például terület- és hossz számításhoz), Geoprocessing (tudományos modellezéshez és térbeli adatelemzéshez), Globe (3D és földgömb megjelenítéséhez), Image (a raszteres adatok kiszolgálása és a képek, például műholdképek vagy ortofotók szállítása feletti ellenőrzés biztosítása érdekében), Keyhole Markup Language (KML), Map (gyorsítótárazott és optimalizált térképszolgáltatás), Mobile (szolgáltatások futtatása terepi eszközökön), Network Analyst (útvonaltervezéshez, a legközelebbi létesítmény helyének meghatározásához vagy szolgáltatási terület elemzéséhez), Search (a *GIS*-eszközök vállalati kereséséhez), Web Coverage Service (WCS), Web Feature Service (WFS) és Transactional Web Feature Service (WFS-T), valamint Web Map Service (WMS).

A GeoServer és a Mapserver összehasonlítása a különbségekre koncentrálva

	GeoServer	**MapServer**
WFS támogatás	Kiváló, Natívan támogatja a WFS-T protokollt	Kiváló, Csak a TinyOWS használattal együtt támogatja a WFS-T protokollt
Technológia	J2EE	CGI
Projekt kezdete	2003	1996
Adminisztráció	Webes	Mapfájlok használatával
Kiterjeszthetőség	Java fejlesztői támogatás	Natív PHP, Python, Perl, Ruby, Tcl, Java és .NET támogatás a Mapscript-en keresztül.
Kartográfia	sztenderd SLD-t használ	SLD támogatás, vagy a mapfájlban beépített stílus támogatás
Szolgáltatások	egy WMS/WFS/WCS szolgáltatás minden felhasználónak	Egy mapfájl jelent egy szolgáltatást
Lekérdezés	CQL és *OGC* szűrők	Beágyazott SQL
Támogatott protokollok és szolgáltatások	Web Map Service (WMS) Web Feature Service (WFS) Web Coverage Service (WCS) Web Processing Service (WPS) Catalog Services for the Web (CSW) Web Map Tile Service (WMTS) SLD (Styled Layer Descriptor) támogatás	Web Map Service (WMS) Web Feature Service (WFS) Web Coverage Service (WCS) WFS-T (transactional Web Feature Service) a TinyOWS együtt megvalósítható WMC (https://www.ogc.org/standards/wmc) WMS alapú INSPIRE Service (https://inspire.ec.europa.eu/documents/Network_Services/TechnicalGuidance_ViewServices_v3.0.pdf) SLD (Styled Layer Descriptor) támogatás

Összességében áttekintve a jelenleg ismert igényeket:

- térképi adatok megjelenítése adatbázisból,
- raszteres állományokból (pl.: légifotók, műholdképek),
- térképi adat állományokból (pl.: KML, GeoPackage, GeoJSON)
- távoli térkép szolgáltatók állományaiból (WMS, WFS)

és további változatos távlati igényekre felkészülve mindenképpen olyan *GIS* rendszer integrálását javasoljuk, amelyik az

- eddigi igényeket kifogástalanul kielégíti és megfelelően rugalmas további bővítésre is.
- Minimális költségek mellett egyszerűen skálázható amennyiben a használati igénye váratlanul és jelentősen megnövekedne.
- A rendszer fejlesztéséhez nem szükséges speciális platform tudású fejlesztőket alkalmazni.
- A rendszer futtatása nem igényel újabb platformok és futtatási környezetek használatát.

Ezen feltételek teljesülése szerint a Mapserver használatát javasoljuk *OpenLayers* és LeafLet WebGIS kliens könyvtárak használatával együtt.

7. Adatbázis-kezelő rendszerek vizsgálata

Jelentős Adatbázis kezelő megoldások

Relációs adatbázis kezelők

- MySQL/MariaDB
- Oracle
- *PostgreSQL*
- MS SQL Server

NoSQL adatbázis kezelők

- MongoDB
- Cassandra
- Redis

A projekt szempontjából figyelemre méltó *Data Warehouse* platformok

- Snowflake,
- BigQuery,
- Redshift
- Cloudera
- Domo
- Teradata
- Talend
- Xplenty
- Greenplum
- Tableau

Alapvető adatbázis kezelőnek relációs, SQL adatbázis kezelőt javaslunk használni a táblák között kialakítható relációi és több soros tranzakció kezelése miatt mely alapvető igénye projekt adatkezelésének továbbá a rendszer-alkalmazás adatkezelési (adatok gyors elérése, komplex

adatkapcsolatok és komplex feltételek mentén) szempontjainak megvalósíthatósága miatt. Ezen szempontok ellenére valószínűleg lesznek olyan adatforrások (különféle dokumentum állományok) amelyek kezelése nem praktikus relációs adatbázis kezelővel ezért NoSQL adatbázis kezelő integrálását is javasoljuk a rendszerbe. Az SQL adatbázis kezelőnek a *PostgreSQL*-t javasoljuk, stabilitása, funkcióinak gazdagsága, nyílt forráskódúsága és szabadon felhasználható licencelése miatt, továbbá pedig, mert teljes mértékben megfelel a tervezett rendszer SQL adatbázis hátterének kiszolgálására.

A NoSQL adatbázis kezelők közül a MongoDB támogatását javasoljuk elsősorban annak rugalmassága, sokoldalúsága és népszerűsége miatt. A MongoDB alkalmas téradatok és idősor adatok kezelésére is. A MongoDB ingyenesen használható, de létezik egy kereskedelmi szolgáltatás változata is (MongoDB Atlas) ami egy multifelhős adatbázis-szolgáltatás.

Data Warehouse (adattárház) platformok használatát illetve adattárház eszközök integrálását is javasoljuk megfontolni mivel ezek lehetővé teszik komplex és diverz adatstruktúrákkal való munkát egyetlen platformon. Továbbá a legtöbb adattárház platform olyan eszközöket is biztosít, amelyek megkönnyítik az adatok elemzését, ábrázolását és riportok készítését. A különféle Warehouse megoldások és szolgáltatások áttekintése nem célja ennek a dokumentumnak mivel az egyes igények prioritása szerint több alternatív javaslat is lehetséges. Több adattárház platform fizetős szolgáltatásként vehető igénybe és a legtöbb a teljes komplex adatstruktúra alternatív kezelési módját jelenti a jelenleg tervezett módhoz képest. Kiemelendő mégis a Greenplum mint nyílt forráskódú párhuzamosítható adatkapcsolatok útján is integrálható rendszer, továbbá a Telend platform amely szintén nyílt forráskódú eszközöket kínál. A fizetős szolgáltatások azok *API* felületén keresztül integrálhatóak illetve kapcsolódhatnak *Business Intelligencia* eszközökön (pl. *Metabase*), vagy az *OpenMetadata*-n keresztül is.

Három adatbáziskezelő rendszer táblázatos összehasonlítása

	PostgreSQL	**MySQL**	**Oracle**
Ár	Ingyenes	Ingyenes, vagy fizetős	Fizetős, egyes kiegészítésekért további licenszeket kell vásárolni
Ismertség	A világ legfejlettebb nyílt forráskódú adatbázis szervere	A világ legnépszerűbb nyílt forráskódú adatbázis szerver	Az egyik legjelentősebb szereplő az adatbázis-kezelők piacán
Fejlesztés	A *PostgreSQL* egy nyílt forráskódú projekt.	A MySQL nyílt forráskódú termék. Az Oracle fejleszti.	Kereskedelmi termék, korlátozott ingyenes változat elérhető

Licenszelés	MIT és BSD licenchez hasonló Nyílt forráskódú PostreSQL licenc.	Kettős licencű, GNU General Public és kereskedelmi. Választható.	Igen szerteágazó licenszelési lehetőségeket kínál.[8] Nem hivatalos tájékoztató oldal: https://www.orafaq.com/wiki/Oracle_licensing
Programozási nyelv	C	C/C++	C/C++
Grafikus felhasználói eszköz	PgAdmin	MySQL Workbench	SQLDeveloper
Tranzakciós koncepciók	ACID	ACID (MyISAM esetében nincs)	ACID (Az izolációs szint paraméterezhető)
Tároló motor	Egyetlen tárolási motor	Többféle tárolási motor, pl.: InnoDB and MyISAM	Egyetlen tárolási motor
Teljes szöveges keresés	Igen	Igen (limitált)	Oracle Text
Ideiglenes tábla törlése	Nincs TEMP vagy TEMPORARY kulcsszó a DROP TABLE utasításban	Támogatja a TEMP vagy TEMPORARY kulcsszót a DROP TABLE utasításban, amely csak az ideiglenes tábla eltávolítását teszi lehetővé.	Kezel session szintű temporary táblát. A DROP TABLE utasítás alapértelmezetten csak a tábla metaadatait törli. A PURGE kitétellel törli fizikailag az adatokat.
Tábla törlése	CASCADE opció támogatása a tábla függő objektumainak, pl. táblák és nézetek törléséhez.	Nem támogatja a CASCADE opciót.	CASCADE opció támogatása a tábla függő objektumainak, pl. táblák és nézetek törléséhez.

[8]. A legújabb verzió licensz opciói: https://docs.oracle.com/en/database/oracle/oracle-database/21/dblic/Licensing-Information.html#GUID-B6113390-9586-46D7-9008-DCC9EDA45AB4

Tábla csonkítás	A *PostgreSQL* TRUNCATE TABLE több funkciót is támogat, mint például CASCADE, RESTART IDENTITY, CONTINUE IDENTITY, tranzakció biztos stb.	A MySQL TRUNCATE TABLE nem támogatja a CASCADE-t és a tranzakció biztonságot, azaz az adatok törlése után nem lehet visszaállítani.	A TRUNCATE TABLE több funkciót is támogat, mint például CASCADE, PRESERVE, DROP, REUSE
Auto increment oszlop	SERIAL	AUTO_INCREMENT	IDENTITY oszlop
Identity Column	Igen	Nem	Igen
Elemző funkciók	Igen	Nem	Igen
Adat típusok	Számos fejlett típus támogatása, mint például tömb, hstore és felhasználó által definiált típusok.	SQL-szabványos típusok	Skalár és összetett adatszerkezetek támogatása, felhasználó által definiálható.
Unsigned integer	Nem	Igen	
Boolean típus	Igen	TINYINT(1) belső használata Boolean értékekhez	
IP cím adat típus	Igen	Nem	
Alapértelmezett érték beállítása egy oszlophoz	Állandó és függvényhívás támogatása	TIMESTAMP vagy DATETIME oszlopok esetén konstansnak vagy	

		CURRENT_TIMESTAMP-nek kell lennie.	
CTE	Igen	Yes (Supported CTE since MySQL 8.0)	
EXPLAIN output	Részletes	Kevésbé részletes	
Materialized views	Igen	No	
CHECK constraint	Igen	Yes (Supported since MySQL 8.0.16, Before that MySQL just ignored the CHECK constraint)	Igen
Tábla öröklés	Igen	Nem	Az Oracle nem támogatja a táblázatok öröklését - helyette TYPE-okat (objektumokat) használ.
Programozási nyelvek tárolt eljárásokhoz	Ruby, Perl, Python, TCL, PL/pgSQL, SQL, *JavaScript*, etc.	SQL:2003	PL/SQL, Java
FULL OUTER JOIN	Igen	Nem	Igen
INTERSECT	Igen	Nem	Igen
EXCEPT	Igen	Nem	Igen (MINUS operátor)
Partial indexes	Igen	Nem	Igen (12c-től kezdve)
Bitmap indexes	Igen	Nem	Igen
Expression indexes	Igen	Nem	Csak az Oracle Database Enterprise Editionben érhető el

Fedő indexek	Igen (9.2 óta)	Igen. A MySQL támogatja a fedő indexeket, amelyek lehetővé teszik az adatok kinyerését kizárólag az index átvizsgálásával, a tábla adatainak érintése nélkül. Ez előnyös a több millió sorból álló nagy táblák esetében..	Igen
Triggerek	Támogatja a legtöbb típusú parancsra kioldható triggereket, kivéve azokat, amelyek globálisan befolyásolják az adatbázist, például a szerepköröket és a táblaterületeket.	Egyes parancsokra korlátozva	Igen
Particionálás	Partícionálás tartomány, lista és (a *PostgreSQL* 11 óta) hash szerint: RANGE, LIST.	Horizontális particionálás, megosztás MySQL Cluster vagy MySQL Fabric segítségével: RANGE, LIST, HASH, KEY és összetett partícionálás a RANGE vagy LIST HASH vagy KEY alpartíciókkal való kombinációjával.	Megosztás, , horizontális partícionálás
Task Schedule	pgAgent	MySQL Scheduled Event	A DBMS_SCHEDULER PL/SQL csomagban található eljárásokkal és függvényekkel valósul meg.
Kapcsolat skálázhatóság a	Minden új kapcsolat egy OP folyamat	Minden új kapcsolat egy OP szál	ORACLE Connection Manager
API-k és egyéb hozzáférési módszerek	ADO.NET JDBC natív C könyvtár	ADO.NET JDBC ODBC	JDBC ODBC ODP.NET

	ODBC streaming *API* nagy objektumokhoz	Saját natív *API*	Oracle Call Interface (OCI)
Biztonsági mentés	A *PostgreSQL* adatbázis lehetővé teszi a felhasználók számára, hogy a pg_dump segédprogram segítségével biztonsági mentést készítsenek.	Egy MySQL-adatbázisban, a biztonsági mentés segédprogramot használva az adatok SQL-utasításokként hajtódnak végre, ezért a nagyméretű adatok mentése és visszaállítása a különböző SQL utasítások teljesítménye miatt sok időt vehet igénybe.	A biztonsági mentés az Oracle legfontosabb backup segédprogramja. A BACKUP parancsot használhatjuk az adatbázis, a táblatér vagy a vezérlőfájl fő vagy másolatának biztonsági mentésére.
Package	Nincs	Nincs	Az eljáráshoz hasonlóan a csomag is egy sémaobjektum, amely a hozzá kapcsolódó változókat, állandókat, alprogramokat, kurzorokat és kivételeket gyűjti össze. A csomagot összegyűjtik és elmentik az adatbázisban.
Támogatott programozási nyelvek	.Net C C++ Delphi Java (JDBC) *JavaScript* (*Node.JS*) Perl PHP Python Tcl	Ada C C# C++ D Delphi Eiffel Erlang Haskell Java *JavaScript* (*Node.JS*) Objective-C OCaml	C C# C++ Clojure Cobol Delphi Eiffel Erlang Fortran Groovy Haskell Java *JavaScript*

		Perl PHP Python Ruby Scheme Tcl	Lisp Objective C OCaml Perl PHP Python R Ruby Scala Tcl Visual Basic
Térbeli funkciók	Több mint 300 függvény és operátor	*OGC* többnyire, kevés valódi térbeli kapcsolat függvény.	Oracle Locator vagy Oracle Spatial - SF kompatibilis az Oracle Database release 10*g* (version 10.1.0.4) kezdve.
Web Mapping ToolKits	Manifold, MapDotNet, ArcGIS 9.3-tól, *UMN Mapserver*, *GeoServer*, FeatureServer, MapGuide Open Source (beta FDO meghajtóval)	*UMN Mapserver*, *GeoServer*, MapGuide Open Source	Oracle Map Builder Tool *UMN Mapserver*, *Geoserver*

8. Jogosultságkezelés lehetséges megoldásai

A jogosultság kezelés a rendszer működését és a felhasználói élményt is meghatározó funkció. Kiemelten fontos a stabilitása, amit a megfelelően kiválasztott fejlesztői koncepció alapoz meg és fontos, hogy a megvalósítandó céloknak megfelelő léptékű jogosultsági rendszer szolgálja ki az igényeket. A jogosultságkezelés jelentősen lassíthatja az adatok kezelését ezért alacsony szintű funkciókkal kell megvalósítani. Továbbá a jogosultság kezelésnek a lehető legegyszerűbbnek kell lennie a fejlesztési átláthatóság és biztonság miatt. Kerülni kell a több független szinten

megvalósított jogosultság kezelést és nem szabad különböző jogosultságkezelési megoldásokból képzett összetett jogosultság kezelést alkalmazni.

Lehetőség szerint egyetlen jogosultság kezelő *API*-n keresztül kell vezetni minden hozzáférés igény kérés kezelést, habár a különböző integrált rendszerek esetében ez nem feltétlenül megvalósítható, csak a rendszer alacsony szintű funkcióinak jelentős korlátozásával.

Az *OpenBioMaps* egy transzparens jogosultság kezelő réteget biztosít a benne foglalt rendszerek használati jogosultságainak kezeléséhez, amivel lehetővé teszi, hogy például a *PostgreSQL* szerver elérhető legyen egyéni felhasználók számára, úgy, hogy a hozzáféréseket nem *PostgreSQL*, hanem *OpenBioMaps* adminisztratív szinten lehet kezelni. Ez a fajta megoldás nagyon nagy mértékű rugalmasságot tesz lehetővé a felhasználási módok támogatása terén, miközben az eltérő rendszerek jogosultság kezelése egyetlen felületen keresztül kezelhető marad.

Javaslatok a kialakítandó jogosultság szerep szintekre:

- **adatok kezelésének jogosultságai**:
  - adat elérési szintek: érzékeny adat, részlegesen nyílt adat, nyílt adat
  - adatmódosítási szintek: nem módosítható, módosítható
- **felhasználók jogosultságai**: ismeretlen felhasználó, ismert felhasználó
  Az ismert felhasználók jogosultságai különböző szintűek lehetnek:

    - Többféle szabadon konfigurálható adminisztrátor szint és
    - Adatfeltöltés,
    - adatmódosítás,
    - adat elérés,
    - adatmegosztás,
    - külső alkalmazás szintű használati jogosultságok.

A legkomplexebb esetben a fenti két kategória tetszőlegesen kombinálható egymással.

Egyszerűbb esetekben egyik, vagy másik alá van rendelve virtuálisan a másik szintnek. Ilyen például a *PostgreSQL* adatbázis kezelő jogosultság kezelése, ahol az adatstruktúra hierarchikus és szerepekhez van rendelve, a szerepek pedig felhasználókhoz. Ilyen módon az adatoknak nincs önálló jogosultság kezelése.

Az adathozzáférés szempontjából a webes alkalmazással elérhető felhasználói funkciók egy része közvetlenül a kezelt adatokkal kapcsolatos (pl.: adatok olvasása és módosítása), míg mások csak közvetetten (pl.: tartalom megosztás, validálás). Az adatkezeléssel kapcsolatos jogosultságkezelés kialakítása során fontos figyelembe venni, hogy az adatokat kiszolgáló rendszereknek is van egy jogosultság kezelése és felmerülhet az igény ezek közvetlen használatának az engedélyezésére és ennek szabályozására. Ebből a szempontból az *OpenBioMaps* jó példa, mivel kombinálja az adatok és felhasználók jogosultságait az adateltérések meghatározásához, ami részletesebb lehetőségeket ad mint az adatszolgáltatást biztosító *PostgreSQL* adatbázis kezelő, viszont emiatt lényegesen bonyolultabbak is a beállításai is, ami több hibalehetőséget rejt magában.

Amennyiben egy magasabb szintű rendszer transzparens módon épít a kiszolgáló jogosultság kezelésére úgy, a kapcsolódó egyéb alkalmazások is tudják, vagy kénytelenek örökölni annak

jogosultság kezelését. Ilyen öröklött jogosultságkezelés esetén kiszolgáló jogosultság kezeléséhez szükséges magasabb szintű adminisztratív felületet biztosítani.

Gyakorlatban nem ismerünk olyan rendszert ami például az adatbázis kiszolgáló vagy a fájlrendszer jogosultság kezelésére épülne, hanem leginkább olyat ami a kiszolgálóhoz való teljes hozzáférésre építve egyéni jogosultság réteget definiál.

A rendszerbe bekerülő adatok struktúrája bizonyos dimenziókban hierarchikusnak tekinthető pl. a projekt -> mintavétel -> gyűjtő vagy projekt -> gyűjtő -> gyűjtött adatok. Ezek fa struktúrát építenek. Elsősorban tároló rendszerekben (object store, fájlrendszer) találkozhatunk részfákra vonatkozó hozzáférési engedélyekkel, de ebben az esetben is releváns lehet. Meg kell vizsgálni, hogy például szeretnénk-e hozzáférést adni valakinek egy projekten belül a saját adataihoz.

Megfontolandó alternatíva lehet egy hibrid hozzáférés kezelés kialakítása, azaz a kiszolgáló jogosultságkezelő rendszerét is felhasználni és kombinálni egy magasabb szintű új jogosultság kezelő réteggel.

Adatbázisra épülő webes alkalmazásokban leggyakrabban önálló jogosultság kezelést szoktak létrehozni, azaz az SQL adatbázisból egy meghatározott hozzáférési szintű felhasználó adja ki az adatokat a webes alkalmazásnak és a webes alkalmazás végzi el a további jogosultság kezelést. Fejlesztési szempontból ez a legegyszerűbb ugyan, de nem teszi lehetővé az alacsony szintű adathozzáférés szabályozását. A hibrid rendszer nagyobb stabilitását támogatja, ha a magasabb szintű jogosultság szintek virtuális rétegként megjelennek az alacsonyabb szintű rendszerben.

Az alacsony szintű funkciók külső alkalmazásokból való elérésének biztosítása érdekében egy hibrid jogosultság kezelés megvalósítását javasoljuk, olyan módon, hogy a webes alkalmazásból, egy felületen keresztül kezelhető legyen az integrált rendszerek tovább osztott jogosultság kezelése a webes alkalmazás belső jogosultság kezelésével. Továbbá (amennyiben erre van lehetőség), az alacsonyabb szintű kiszolgáló jogosultság rendszerében egy automatikusan kezelt virtuális rétegként jelenjen meg a webes alkalmazás jogosultság kezelése. Például a SQL adatbázis kezelőben jöjjenek létre a webes alkalmazásban létrehozott szerepek is és a felhasználók adat elérését ne egy magasabb jogosultsági szintű központi felhasználó szabályozza, hanem minden felhasználó a saját (webes rendszerben meghatározott) jogosultsági szintjének megfelelő legalacsonyabb jogosultságokkal rendelkező szereptől örökölt jogosultságokkal férjen az adatokhoz. Ilyen módon a webes alkalmazás már csak a finom skálázását fogja végezni az adatok hozzáférés kezelésének és az adatbázis szintjéről is követhető lesz az egyes felhasználók adathasználata.

A bejelentkezések kezelésének is több megvalósítási iránya lehetséges, illetve ezek kombinációja. Lehet kötni személyekhez (jelszavas, vagy kulcsos), lehet kötni eszközökhöz (MAC address, vagy kulcs). A személyes bejelentkezések intézményi szintű kezelése is lehetséges amelyre jó példák vannak egyetemi környezetben (EduRoam), vagy a kormányzati ügyfélkapu rendszer. Megfontolandó az EduRoam és akár az ügyfélkapu támogatása is fejlesztendő rendszerben. Ezen túl érdemes további bejelentkezési módokat is támogatni (erre számos EU-s rendszerben láthatunk példát (pl. *EOSC*)). A bejelentkeztetés és a bejelentkezési állapot egymástól független rétegekben megvalósítandó feladat a fentiek miatt, hogy kellően rugalmas és kiegészíthető legyen.

Megfontolható a kétlépcsős azonosítás (kétényezős hitelesítés, avagy Two Factor Authentication (2FA)) támogatása is.

A tartalom megosztások kezelése is jogosultság függő funkció, amit egyszerű megvalósítani amennyiben egyetlen tartalmi elemre vonatkozik, amely egy permanens azonosítóval azonosítható (Handle, DOI). Jó példákat lehet látni tartalmi megosztásra a Nextcloud rendszerben, vagy a Google Drive rendszerben. Amennyiben viszont komplex tartalmat szeretne valaki megosztani, például egy lekérdezés összes sorát kapcsolódó információkkal együtt, azaz sok rekordból álló adatállományt, akkor ennek a jogosultságkezelése bonyolulttá válhat, ha az egyes rekordoknak, vagy tartalmaknak eltérő hozzáférés jogosultságai vannak, amit meg szeretnénk őrizni. További komplex eset, ha olyan felhasználó oszt meg adatokat aki nem fér hozzá a megosztandó rekordok teljes mélységéhez, mert ilyen esetben a megosztásánál, a megosztó jogosultság szintjének öröklésére is oda kell figyelni. Amennyiben viszont a megosztás nem az eredeti rekordokra hivatkozik, hanem azokról egy másolat készül, akkor a jogosultság örökítés kezelésének problémája kiküszöbölhető.

Ezen okok miatt azt javasoljuk, hogy az adatmegosztások miden esetben olyan önálló új adatcsomagok formájában készüljenek el, amelyekhez permanens azonosítót lehet rendelni (pl. Handle, vagy DOI) és metaadatokkal lehet ellátni. Ilyen módon az adatmegosztásoknál nem kell figyelembe venni a megosztó hozzáférési jogosultságait, hanem elegendő azt szabályozni, hogy ki oszthat meg tartalmakat. További lehetőségek vannak a hozzáférés finom szabályozására itt is az adatbázis kezelő jogosultság kezelő rétegének felhasználásával.

Külső alkalmazások számára hozzáférési jogosultság adására a legjobb példa az *API* token regisztráció, amely legegyszerűbb esetben személyhez kötődik. Ilyen esetben az egyes felhasználók tudnak *API* token-t létrehozni és egyes alkalmazások számára regisztrálni azt. Továbbá pedig tudnak a token-hez feladatköröket rendelni (a saját jogosultsági körökön belül). Így működik a legtöbb nagy webes szolgáltató (például a Google API használata). *API* token létrehozást és a tokenhez saját jogosultsági szinten belüli feladatkörök kiosztásának engedélyezést javasoljuk beépíteni a rendszerbe minden felhasználó számára elérhető szolgáltatásként. A bejelentkezéseket és az alkalmazás hozzáféréseket *OAuth2* használatával javasoljuk megvalósítani.

9. Összefoglalás és javaslattétel

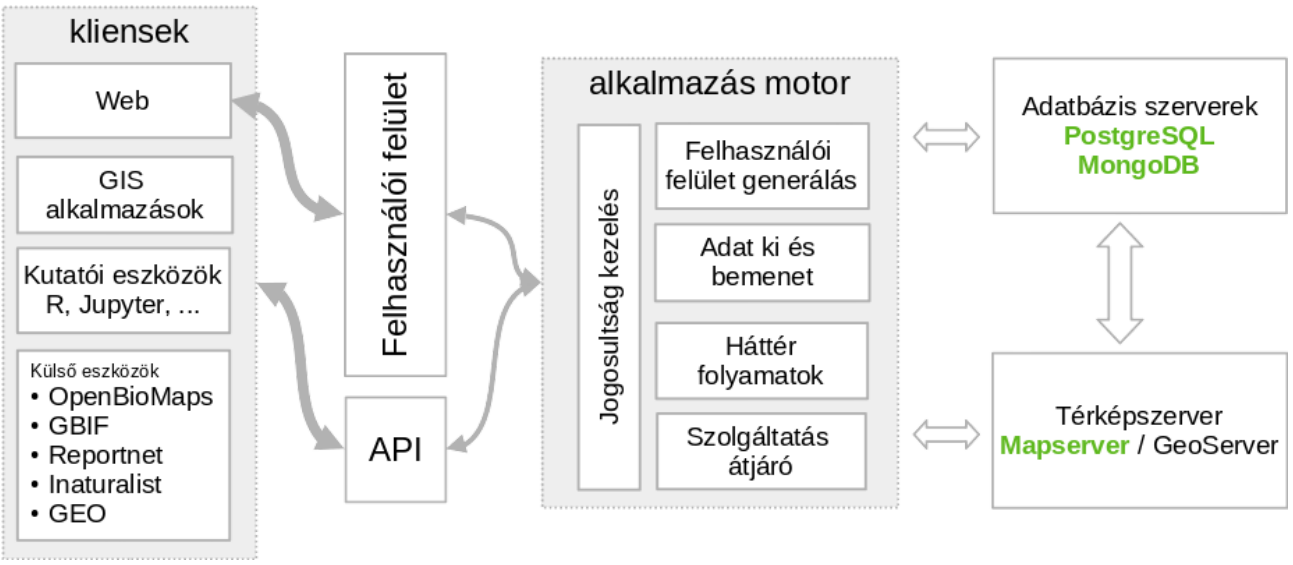
A megvalósítandó rendszer korábban tárgyalt önálló komponensekből felépülő struktúrájú felépítésének véleményünk szerint nincs észszerű alternatívája úgyhogy megvalósítási alternatívák szempontjából inkább csak az a kérdés, hogy melyek legyenek ezek a komponensek és milyen kapcsolati módokat támogassanak.

Egyes komponensek kiválasztásánál figyelembe lehet venni, ha a minisztérium a komponens szolgáltatását megvalósító szerver vagy szolgáltatás licenccel rendelkezik, habár a korábbi fejezetekben tárgyalt összehasonlítások során minden felhasználandó szerver szolgáltatás esetén a nyílt forráskódú eszközök használatát javasoltuk az esetleges kereskedelmi alternatívákkal szemben és összességében egyetlen olyan esetet sem tudunk mondani, ahol érdemes lenne

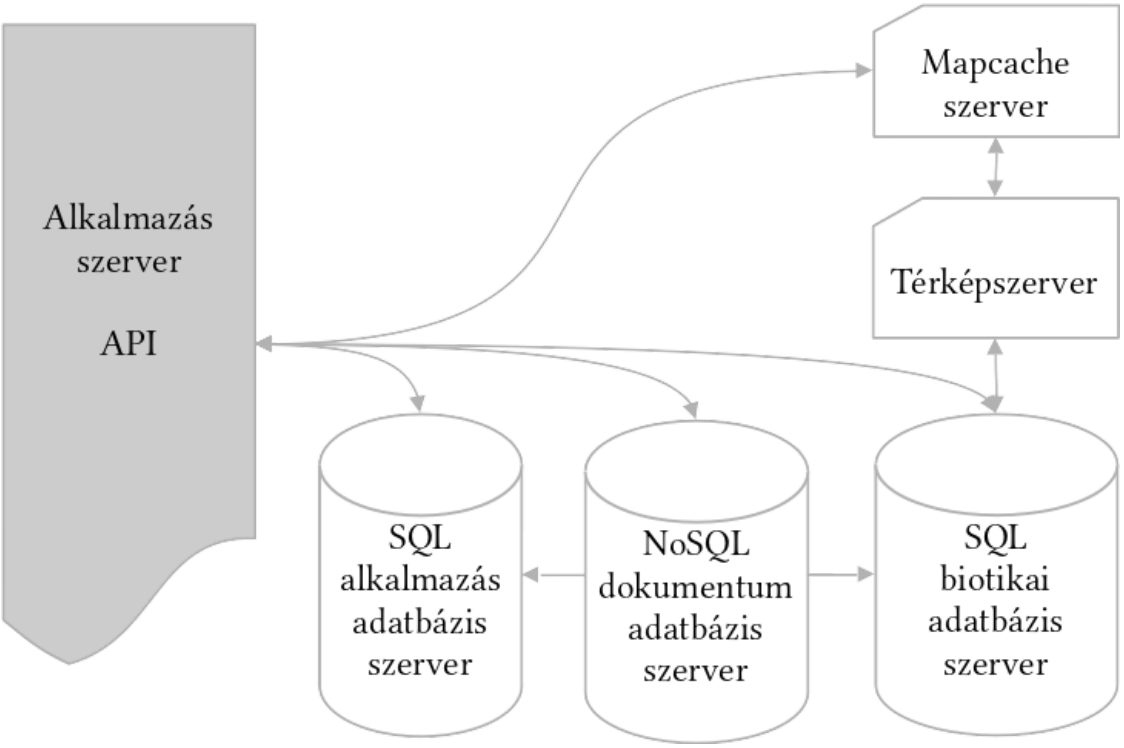
bármilyen meglévő licenc felhasználása miatt egy kereskedelmi szoftvert előnyben részesíteni a nyílt forráskódú alternatívák helyett (különös tekintettel adatbázis és térkép szerverre).

A következőkben a rendszer javasolt strukturális felépítésének összefoglalását és az egyes komponensek felépülésének néhány alternatív felépülését mutatjuk be kiemelve az általunk javasolt irányokat. Habár minden komponens felépítésével kapcsolatban megfogalmaztunk javaslatokat, mégis több olyan eset is van, ahol a fejlesztői preferenciákat is érdemes figyelembe venni. Ezeket zöld színnel jelöljük.

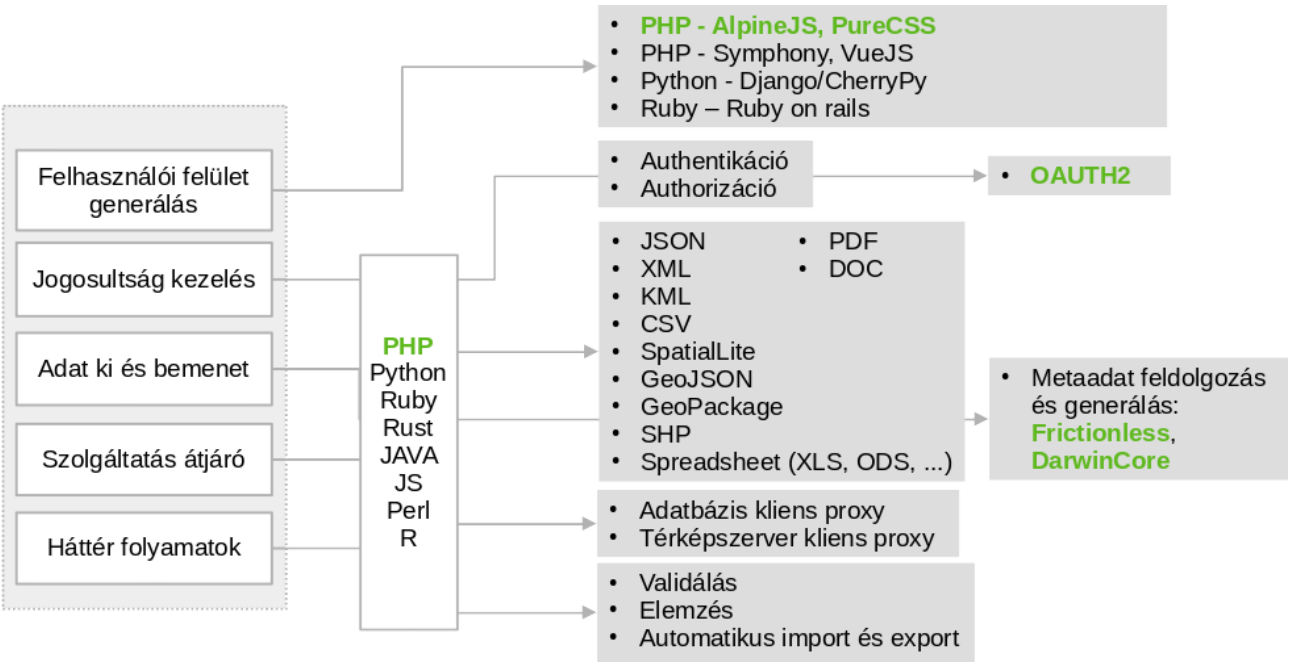
1. ábra. A szerver felépítésének és a kliensek kapcsolódásának sematikus ábrázolása.



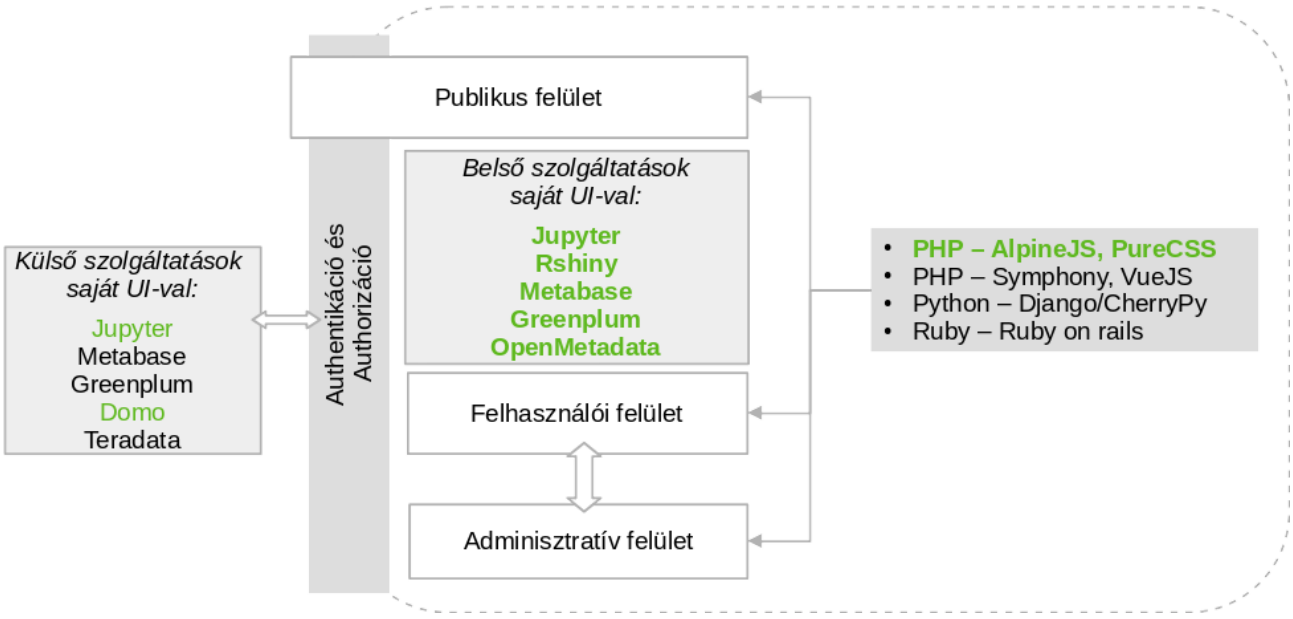
2. ábra. A kiszolgáló szerverek horizontális skálázásának egy lehetséges felépítése.



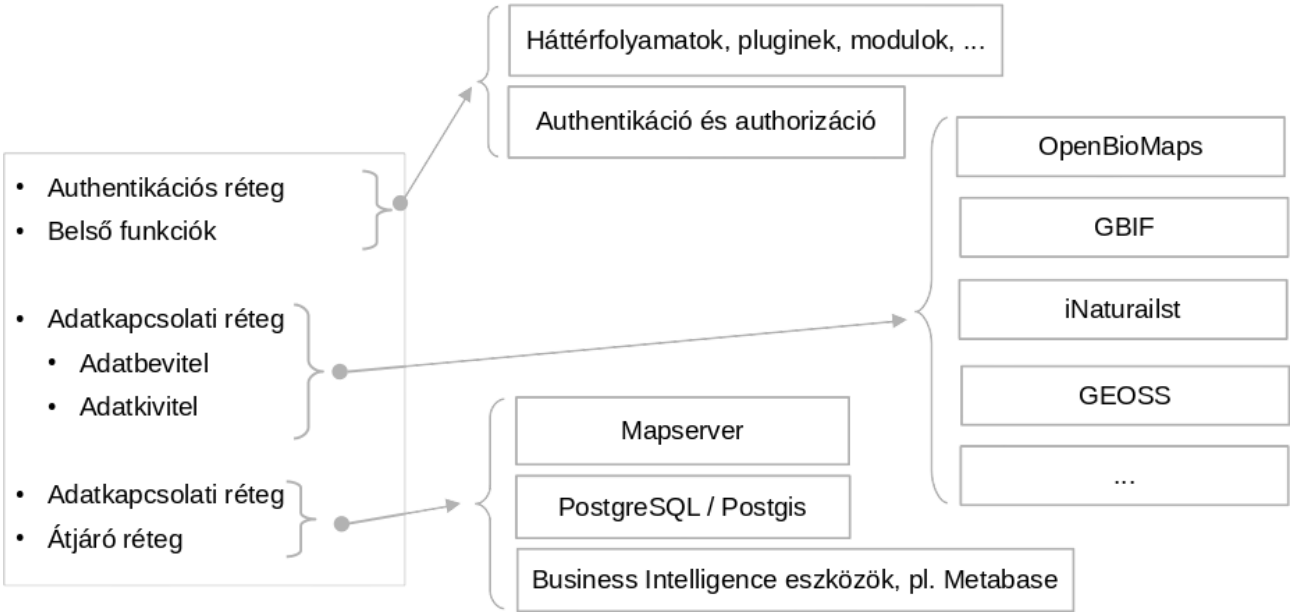
3. ábra Az alkalmazás motor felépülésének sematikus ábrázolása.



4. ábra. felhasználói felületek és fejlesztői platformok sematikus ábrázolása.



5. ábra *API* kapcsolatok sematikus ábrázolása.



9.1. Az operációs környezet kiválasztása

A megvalósítandó feladat hosszútávon fenntartható kezelése és a megfelelő rugalmasság elérése érdekében egy olyan szerviz komponensekből álló elosztott rendszer fejlesztését javasoljuk, amely alapvetően nem operációs rendszer, hanem operációs környezet függő. Azaz, a fejlesztett szerviz komponensek egy integrált alkalmazás környezetben futnak, amelyet többféle operációs rendszer (illetve ezek együttese) is kiszolgálhat. Ilyen módon a szolgáltatás réteg nem operációs rendszer függő, mert a hardver és az alkalmazás között egy köztes alkalmazás kiszolgáló réteg helyezkedik el. Az operációs környezet optimális esetben nem jelent extra terhet, vagy legalábbis mérhető terhet a kiszolgáló szerver számára, viszont jelentősen növeli az alkalmazás réteg rugalmasságát és stabilitását (nem kell az alacsony szintű függőségek kezelését nyomon követni). Operációs környezetet konténerizált alapon érdemes felépíteni. Ebben az esetben izolált környezetben futnak az egyes alkalmazás komponensek úgy, hogy az operációs rendszer kerneljére közvetlenül támaszkodnak (azaz nem virtualizáció!). A legnépszerűbb és legnagyobb tudású konténerizáló platform - aminek a használatát javasoljuk - jelenleg a *Docker*, amely Linux kernelre épül. A konténer alapú alkalmazáskezelők nagy előnye az üzemeltetés során jelentkezik. A kezelő képes a rendelkezésére álló hardver erőforrások szolgáltatások közötti elosztására. Ez megjelenik a skálázhatóság képességben illetve a rendelkezésre állás biztosításában. Ezt a redundáns rendszer felépítéssel éri el. A skálázhatóság lehetővé teszi, hogy dinamikusan reagáljon a megnövekedett terhelésre pl. egyszerre több felhasználói kérés kiszolgálása. A rendelkezésre állást növeli a hardver hibák ill. a karbantartás miatti leállások során a rendszer erőforrásainak újra elosztásával.

Komplexebb szolgáltatás hálózat felépítése esetén a konténer alapú rendszert *Kubernetes* alapú alkalmazáskezelő rendszerbe lehet szervezni, amelynek platformja az aktuálisan elérhető bármelyik nagyobb rendszer lehet (pl. OpenShift, Tanzu, Google Kubernetes Engine), de ennek eldöntése nem célja ennek a dokumentációnak.

A konténerizált operációs környezetben futtatott szerver alkalmazás vertikálisan könnyen skálázható az egyes egységeinek nagyobb kapacitású szerverre való költöztetéssel. A *Docker*izált

környezetet számos szolgáltató is támogatja, mint például az *EOSC*, AWS, Google amelynek köszönhetően egyes komponensek saját szerver nélküli (serverless) használata is megvalósítható, amellyel egy hibrid rendszer jön létre. A komponensek önálló működésének további előnye a magas hibatűrés és így a magasabb költséghatékonyság.

9.2. Javaslatok a hardver környezet kialakításával kapcsolatban

Egy olyan szerviz komponensekből álló elosztott rendszer fejlesztését javasoljuk, amely technikai üzemeltetése elsősorban csak a komponensek közötti kapcsolati rendszer karbantartásából áll és a szerviz komponensek hardver igénye egymástól függetlenül szabályozható.

Figyelembe véve, hogy ezen a területen jelentős fejlődés várható mind a hazai mind Európai szinten is, a fő komponensek kapcsolódásait úgy kell megvalósítani, hogy azok igény szerint külső más rendszerben üzemeltetett (gyűjtő-szervíz) szolgáltatásba is átvihetők legyenek.

A fejlesztendő komponensek és becsült minimális hardverigény átlagosan 20-50 aktív felhasználóval számolva:

- *UI* szerver
  - 4 CPU mag / 4Gb memória / 100Gb Hdd
- *API* szerver
  - 4 CPU mag / 4Gb memória / 100Gb Hdd
- Adatbázis szerver
  - 16 CPU mag / 16Gb memória / 1000Gb Hdd
- Térkép szerver
  - 8 CPU mag / 16Gb memória / 6000 Gb Hdd
- Alkalmazás szerver (Számítások, riportok, felhasználó kezelés, import, export, megosztás)
  - 8 CPU mag / 8Gb memória / 1000Gb Hdd

Az egyes komponensek szervereit nem praktikus egyetlen fizikai szerveren felépíteni, hanem legalább két külön szerveren elhelyezni amivel növelhető a rendszer hibatűrése és rendelkezésre állása.

Az adatbázis és térkép szervert érdemes egy fizikai szerveren két önálló virtuális szerverként telepíteni, míg másik három szervert (*API*, *UI*, Alkalmazás) egy független fizikai szerver önálló virtuális szervereire.

Alapvetően nem javasoljuk saját fizikai szerverek használatát, hanem külső független szolgáltatótól való szerver bérlést, mivel ennek fenntartási költsége kedvezőbb a saját szerver üzemeltetésével szemben, hacsak nem áll már amúgy is rendelkezésre üzemeltetett szerver kapacitás.

Amennyiben saját szerver üzemeltetése mellett döntenek akkor minimális igényeket figyelembe véve 2 db Dell PowerEdge R740xd szerver (2x INTEL XEON DODECA CORE SILVER 4116 2,1GHZ 12CORE 24THREADS / 128GB DDR4 RDIMM ECC RAM / 6TB NL SAS HDD) beszerzését javasoljuk, amely hardver konfiguráció teljesítményében meghaladja az ajánlott minimum szintet, hozzávetőlegesen 150 felhasználó egyidejű kiszolgálására lehet alkalmas, de skálázhatósága is hibatűrése tekintetében csak a minimum követelményeket teljesíti. Optimális stabilitást és rendelkezésre állást négy önálló szerverrel lehet megvalósítani. Ebben az esetben a vertikális skálázhatóság a szolgáltatások közötti erőforrás elosztás szabályozásával részben megoldható, míg a horizontális skálázhatóságra továbbra sincs lehetőség.

Saját hardver, bérelt erőforrások és külső szolgáltatók szolgáltatásaival felépített hibrid rendszerrel lehet a maximális rugalmasságot és skálázhatóságot elérni.

9.3. Javaslatok a fejlesztés megvalósításával kapcsolatban

A fejlesztést konzorciális formában javasoljuk megvalósítani, olyan módon, hogy legyen egy vezető fejlesztő, és az egyes fejlesztési egységeknek önálló egymástól független fejlesztői. Ennek a megoldásnak az az előnye, hogy az egyes egységek fejlesztése párhuzamosan történhet meg és így egymástól független lesz, ami a későbbi fenntartási stabilitáshoz is hozzájárulhat. Továbbá az egymástól független, több résztvevőből álló fejlesztői konzorcium várhatólag egy megbízhatóbban működő alkalmazást tud prezentálni, mivel a független komponenseknek már a fejlesztés során jól kell kommunikálnia egymással. Ennek a megoldásnak a hátránya, hogy valószínűleg költségesebb mint egy fejlesztővel megoldani a teljes fejlesztést.

Ezek az önállóan, különálló fejlesztőkkel megvalósítandó részek legalább a következők legyenek, de ennél több alrészre is bontható (alpontokkal jelölve)

- *API* (I. fejlesztő)
    a. belső API
    b. külső API
- *UI* (II. fejlesztő)
    a. webes felhasználói felület, parancssori felület, elemző felületek, riportok,
    b. adminisztratív *UI*: térkép konfigurálás, adatkezelő, jogosultság kezelő, fájlkezelő, …
- Háttér alkalmazás (III. fejlesztő)
    a. háttér folyamatok,
    b. jogosultsági rendszer,
    c. térkép kezelés, adatbázis kezelés,
    d. plugin rendszer, modul rendszer,
    e. fájlkezelés, importálás, exportálás, megosztás

A fejlesztendő alkalmazás fenntarthatósága szempontjából egy sztenderd verziókövető rendszer és ismert szolgáltatás kiválasztását és használatát javasoljuk. A verziókövető rendszer a *GIT* legyen, mivel jelenleg ez a legismertebb, nyílt forráskódú és rendkívül széleskörűen támogatott.

A legismertebb webes *GIT* szolgáltatás habár a *GitHub*, de majdnem hasonlóan népszerű *GitLab* a nyílt forráskódja és a beépített *CI/CD* eszköze miatt előnyben részesítendő. Sőt a *GitLab* használatával saját *GitLab CI/CD* szervert is fenn lehet tartani amely egy nagyobb projekt esetén hasznos és költséghatékony megoldás.

9.4 Javaslatok a rendszer fenntartásával kapcsolatban

A rendszer üzemeltetése és fenntartása szempontjából több szempontot is érdemes figyelembe venni, mivel hosszú távon ezek a rendszer kapcsán felmerülő legnagyobb költségek. Ezek egyrészt a kód fenntartás költségei másrészt a használat kapcsán felmerülő támogatás költségei.

Általában elmondható, az üzemeltetés és fenntartás a szoftver teljes költségének 40-80%[9]-át teszik ki és annál magasabb ez az arány minél komplexebb szoftverről van szó. Egy szoftver átlagos életideje körülbelül 5 év, ami azt jelenti, hogy öt évenként jelentősebben meg kell újítani ahhoz, hogy a technikai változásokat és az ennek kapcsán megjelenő felhasználási igény változásokat követni tudja. Ezért egy 5 éves fenntartási időszakra gondolva ennek a meglehetősen komplex igényeket kiszolgáló szoftvernek a fejlesztésre és fenntartásra szánt keretét úgy kell

[9]. Frequently Forgotten Fundamental Facts about Software Engineering. Robert L. Glass, IEEE Software May/June 2001.

beosztani, hogy körülbelül a teljes költség 75%-át a fenntartási költség fogja kitenni, amit nagyjából egyenletesen elosztva folyamatosan kell megfizetni. A kód fenntartási költségek egyrészt csökkenthetők *DevOps* szakemberek és technológiák alkalmazásával másrészt a közösségi fejlesztésű eszközök használatával is (ilyen esetben a fejlesztő közösség közösen viseli a hibajavítások és a továbbfejlesztések költségét). A fenntartási költségek további jelentős összetevője a rendszergazdai költségek (monitoring, rendszer hangolások) és a használati támogatás költsége, amely egy komplex rendszer esetében jelentős lehet (pl. “data scientist” alkalmazása, oktatási költségek). Ez utóbbi típusú - a fejlesztéstől független üzemeltetési költségek csökkentésére nincsenek általánosan javasolható receptek és biztosan lehet rá számítani, hogy a rendszer komplexitásának növekedésével ezek is növekednek. Például az *OpenBioMaps* rendszerek fenntartása kapcsán elmondható tapasztalat, hogy kutató csoportonként, vagy intézményenként (kb. 5-50 fő felhasználó között) egy fő el tudja látni a rendszer fenntartásával kapcsolatos feladatokat, de szerverenként az alacsonyabb szintű rendszerüzemeltetési feladatok elvégzése további egy fő rendszergazdai ismeretekkel rendelkező alkalmi foglalkoztatását is megköveteli. Az adatbázisok körüli adattervezési és elemzés tervezési feladatok szintén további egy fő alkalmi foglalkoztatását igényli. Elméletben lehet ugyan találni olyan informatikus szakembert aki képes egy személyben ellátni az adat-kurátor, rendszergazda és data-scientist feladatokat is de valójában sokkal reálisabb három különböző szakemberrel megoldani ezeket a feladatokat. Viszont három szakember főállású foglalkoztatása valószínűleg feleslegesen nagy költség teher lenne. Ezért véleményünk szerint a leghatékonyabb és legkevésbé költséges megoldás külső szakértő céggel megoldani az üzemeltetési és fenntartási feladatokat.

A rendszer ökológiai lábnyomának csökkentése érdekében javasoljuk hogy a rendszerben kezelt adatok teljes mértékben legyenek metaadatolva és elsődlegesen a metaadatok legyenek kereshetők és megjeleníthetők a rendszerben különös tekintettel azokra az adatokra, amelyek nincsenek aktuális használatban (például archív adatok, amelyeknek csak a publikus megjelenés szempontjából vagy alkalmi kutatói igények kielégítése szempontjából érdekesek). Azok az adatok pedig, amelyek nincsenek aktuális használatban kerüljenek ki az online adatbázisból és esetleg kerüljenek „offline storage” rendszerbe (pl. szalagos tároló), vagy alacsonyabb fenntartási költségű tároló rendszerbe. Ezek a módszerek nyilván csak nagy adattömegek esetén térülnek meg, de például ilyen nagy adattömeg lehet egy órási kép adatbázis (légifotók).

9.5. Szószedet

- ***AWS*** - Az Amazon Web Services, Inc. az Amazon leányvállalata, amely igény szerinti felhő alapú számítástechnikai platformokat és *API*-kat kínál magánszemélyek, vállalatok és kormányzatok számára, mérőszámlák szerint fizetős alapon.
- ***API*** - Application Programming Interface, azaz alkalmazásprogramozási felületet, amely egy eszköz arra, hogy programozóknak alkalmazások fejlesztéséhez hozzáférést biztosítson egy alkalmazás utasításkészletéhez és eszközeihez.
- ***Business Intelligence*** - Üzleti intelligencia - Az üzleti intelligencia (BI) azt az eljárási és technikai infrastruktúrát jelenti, amely összegyűjti, tárolja és elemzi a vállalat tevékenységei által előállított adatokat. A BI tág fogalom, amely magában foglalja az adatbányászatot, a folyamatelemzést, a teljesítmény-összehasonlítást és a leíró elemzést.
- ***CI/CD*** - A folyamatos integráció (CI) és a folyamatos szállítás (CD) olyan kultúrát, működési elveket és gyakorlatok gyűjteményét testesíti meg, amelyek lehetővé teszik az alkalmazásfejlesztő csapatok számára, hogy gyakrabban és megbízhatóbban szállítsák a kódváltozásokat. A megvalósítást CI/CD-csatornának is nevezik. Röviden összefoglalva, a telepítési lépések automatizálására szolgál.

- ***CMS*** - Content Management System, azaz webes tartalomkezelő rendszer
- ***CNTK*** - Microsoft Cognitive Toolkit a Microsoft Research által kifejlesztett, (de tovább már nem fejlesztett) nyílt forráskódú mélytanulási ONNX kompatibilis AI keretrendszer.
- ***CVAT*** - (https://cvat.org/) A Computer Vision Annotation Tool egy nagy tudású, ingyenes, nyílt forráskódú, webalapú kép- és videomegjelölő eszköz, amely a ML tanulási algoritmusok adatainak címkézésére szolgál.
- ***Data Warehouse*** - Azaz adattárház egy olyan információs rendszer, amely több forrásból származó múltbeli és kommutatív adatokat tárol. A különböző forrásokból származó tranzakciós adatok elemzésére, jelentésére, integrálására szolgál. Elsősorban riportok és adat analízisek kiszolgálására való.
- ***DBT*** - (https://www.getdbt.com/) - (data build tool) lehetővé teszi elemző fejlesztők (és adattudosók) számára, hogy az adattáraikban (pl. CSV forrásfájlok) lévő adatokat egyszerűen select utasítások írásával átalakítsák. A dbt alapvetően egy parancssori eszköz, de az integrált fejlesztői környezet (IDE) megjelenése a dbt Cloudban, lehetővé teszi az elemzők számára, hogy a teljes analitikai tervezési munkafolyamatot a böngészőjükben építsék fel.
- ***DevOps*** - a fejlesztés (Dev) és az üzemeltetés (Ops) összetétele, a DevOps az emberek, a folyamatok és a technológia egyesítése az ügyfelek számára történő folyamatos értékteremtés érdekében. A DevOps lehetővé teszi a korábban elszigetelt szerepkörök - fejlesztés, IT-üzemeltetés, minőségfejlesztés és biztonság - összehangolását és együttműködését a jobb és megbízhatóbb termékek előállítása érdekében. A DevOps-kultúra, valamint a DevOps gyakorlatok és eszközök elfogadásával a fejlesztők képessé válnak arra, hogy jobban reagáljanak az ügyfelek igényeire, növeljék az általuk készített alkalmazásokba vetett bizalmat, és gyorsabban érjék el az üzleti célokat.
- ***Docker*** - egy olyan platform mint szolgáltatás termékcsalád, amely operációs rendszer szintű virtualizációt használ a szoftverek konténereknek nevezett csomagokban történő szállítására. A konténerizáció a szoftverkód és az összes szükséges komponens, például könyvtárak, keretrendszerek és egyéb függőségek csomagolása úgy, hogy azok a saját "konténerükben" vannak elszigetelve.
- ***EOSC*** - *European Open Science Cloud* (https://eosc-portal.eu/about/eosc) azaz Európai Nyílt Tudományos Felhő az Európai Bizottság kezdeményezése, amelynek célja egy olyan infrastruktúra kialakítása, amely a felhasználók számára a nyílt tudományos gyakorlatokat támogató szolgáltatásokat nyújt.
- ***GeoServer*** - (http://geoserver.org/) - *lásd még 6. fejezetben.*
- ***GIS*** - térinformatikai rendszer és egy olyan adatbázis-típus, amely földrajzi adatokat tartalmaz, az adatok kezelésére, elemzésére és megjelenítésére szolgáló szoftver eszközökkel kombinálva.
- ***Git, Gitlab, Github*** - A Git egy olyan szoftver, amely bármilyen fájlkészletben bekövetkezett változások nyomon követésére szolgál, és általában a szoftverfejlesztés során a forráskódot közösen fejlesztő programozók munkájának koordinálására szolgál. Céljai közé tartozik a sebesség, az adatintegritás és az elosztott, nem lineáris munkafolyamatok (több ezer párhuzamos ág különböző rendszereken futó ágai) támogatása. A Git nyílt forráskodú szoftver, szabadon felhasználható és számos kereskedelmi és szabadon használható szolgáltatás épül rá, amelyek további eszközöket biztosítanak szoftverek fejlesztéséhez. A legtöbb webes verziókövető szolgáltatás ma már vagy GIT alapú, vagy támogatja a Git-et. A legismertebb ilyen szolgáltatások a GitHub és a GitLab.
- ***GBIF*** - azaz Global Biodiversity Information Facility - a világ kormányai által finanszírozott nemzetközi hálózat és adat-infrastruktúra, amelynek célja, hogy bárkinek, bárhol, nyílt hozzáférést biztosítson a földi élet minden típusára vonatkozó adatokhoz.
- ***GUI*** - Graphical User Interface - grafikus (nem csak karakteres) felhasználói felület.
- ***JS*** - ***JavaScript*** - egy programozási nyelv, amely a HTML és a CSS mellett a World Wide Web egyik alapvető technológiája.
- ***Jupyter, Jupyter Nootebook*** - A Jupyter Notebook a Jupyter projekt (egy olyan projekt és közösség, amelynek célja "nyílt forráskódú szoftverek, nyílt szabványok és szolgáltatások fejlesztése az interaktív számítástechnikához számos programozási nyelven) része, egy nyílt forráskódú webes alkalmazás, amely lehetővé teszi a felhasználói számára olyan

dokumentumok létrehozását és megosztását, amelyek egyetlen dokumentumban integrálják az élő kódot, egyenleteket, számítási eredményeket, vizualizációkat és egyéb multimédiás forrásokat, valamint magyarázó szöveget.

- ***JSON*** - egy nyílt szabványos fájlformátum és adatcsere-formátum, amely ember által olvasható szöveget használ attribútum-érték párokból és tömbökből (vagy más szerializálható értékekből) álló adatobjektumok tárolására és továbbítására. Ez egy általános adatformátum, amelyet az elektronikus adatcserében sokféleképpen használnak, beleértve a webes alkalmazások és a kiszolgálók közötti adatcserét is.
- ***Kubernetes*** - egy nyílt forráskódú konténer-orchestrációs rendszer a szoftverek telepítésének, skálázásának és kezelésének automatizálására.
- ***Leaflet*** - (https://leafletjs.com/) egy nyílt forráskódú *JavaScript* könyvtár, amelyet webes térképes alkalmazások készítésére használnak. Először 2011-ben adták ki, a legtöbb mobil és asztali platformot támogatja, és támogatja a HTML5-öt és a CSS3-at.
- ***Library*** - A számítástechnikában a könyvtár (library) a számítógépes programok által használt tartós "eszköz" források gyűjteménye, amelyeket számítógép programok használnak. Ezek lehetnek konfigurációs adatok, dokumentáció, súgó adatok, üzenetsablonok, előre megírt kód és alprogramok, osztályok, értékek vagy típusspecifikációk is.
- ***MapServer*** - (https://mapserver.org) - *lásd még 6. fejezetben.*
- ***Manifest*** - a webalkalmazás-manifest egy W3C-specifikáció, amely egy JSON-alapú manifestet határoz meg, hogy a fejlesztők számára egy központi helyet biztosítson a webalkalmazáshoz kapcsolódó metaadatok elhelyezésére, beleértve:
  - A webalkalmazás neve
  - A webalkalmazás ikonjaira vagy képi objektumaira mutató linkek.
  - A webalkalmazás elindításához vagy megnyitásához használt URL-cím.
  - A webalkalmazás konfigurációs adatai
  - A webalkalmazás alapértelmezett tájolása
  - A megjelenítési mód beállításának lehetősége, pl. teljes képernyő

  Ezek a metaadatok elengedhetetlenek ahhoz, hogy egy alkalmazás hozzáadható legyen a kezdőképernyőhöz, vagy más módon a natív alkalmazások mellé kerüljön.

- ***MetaBase*** - (https://www.metabase.com/) egy nyílt forráskódú *üzleti intelligencia* eszköz, amely lehetővé teszi, hogy kérdéseket tegyünk fel az adatokról, és a válaszokat könnyen értelmezhető formátumban jeleníti meg, legyen az egy oszlopdiagram vagy egy részletes táblázat. A kérdések lementhetőek, megismételhetőek (riportfunkciók alapja) és megoszthatóak másokkal.
- ***MIT*** licenc - Kifejezetten megengedő licenc, semmi nem korlátozza sem a származékos mű kereskedelmi értékesítését, sem pedig zárt forráskódú programként való licencelését. A szoftverrel kapcsolatban a fejlesztő felé nem támaszthatunk garanciális követelést.
  A szoftver szabadon újra licencelhető (beépíthető saját licences termék alá), módosítható, megosztható, eladható, al-licencebe adható, összevonható más szoftverrel korlátozás nélkül.

- ***MVC*** - A modell-nézet-vezérlő (MVC) egy olyan szoftvertervezési megoldás, amelyet általában olyan felhasználói felületek fejlesztésére használnak, amelyek a kapcsolódó program logikát három, egymással összekapcsolt elemre osztják. Ezt azért teszik, hogy elválasszák az információk belső reprezentációit attól, ahogyan az információkat a felhasználónak átadják és fogadják.
- ***OAuth2*** - Az OAuth2 a hozzáférés delegálásának nyílt szabványa (az OAuth továbbfejlesztése), amelyet általában arra használnak, hogy az internetfelhasználók más weboldalakon lévő webhelyeknek vagy alkalmazásoknak hozzáférést biztosítsanak az adataikhoz anélkül, hogy megadnák a jelszavakat.
- ***OGC*** - (https://www.ogc.org/) Open Geospatial Consortium amely 1994 óta létezik és a térbeli adatok kezelésére és megosztására vonatkozó szabványokat hoznak létre.
- ***OGC API*** - (http://opengeospatial.github.io/e-learning/ogcapi-features/text/basic-main.html) egy több részből álló szabvány, amely lehetővé teszi a térbeli adatok létrehozását, módosítását és lekérdezését a weben, és meghatározza a követelményeket és ajánlásokat azon *API*-k

számára, amelyek a térbeli adatok megosztásának szabványos módját kívánják követni.

- ***OpenBioMaps*** - *OBM* - (https://openbiomaps.org) - nyílt forráskódú biodiverzitási adatkezelő platform - *lásd még 3. fejezetben.*
- ***ONNX*** - Az Open Neural Network Exchange egy nyílt forráskódú mesterséges intelligencia ökoszisztéma, amely nyílt szabványokat hoz létre a gépi tanulási algoritmusok és szoftvereszközök fejlesztéséhez, hogy elősegítse az innovációt és az együttműködést az AI-szektorban.
- ***OpenAPI*** - (https://www.openapis.org/) - Az OpenAPI specifikáció (OAS) egy szabványos, nyelv-agnosztikus interfészt határoz meg a *REST*ful *API*-khoz, amely lehetővé teszi, hogy emberek és számítógépek egyaránt felfedezzék és megértsék a szolgáltatás képességeit anélkül, hogy hozzáférnének a forráskódhoz, a dokumentációhoz vagy a hálózati forgalom vizsgálatához - *lásd még 3. fejezetben.*
- ***OpenLayers*** - (https://openlayers.org/) egy nyílt forráskódú *JavaScript* könyvtár, amely térképadatokat - bármilyen forrásból betöltött térképcsempéket, vektoros adatokat és jelöléseket képes megjeleníteni webböngészőkben.
- ***Node.JS*** - A Node.js egy nyílt forráskódú, többplatformos, back-end JavaScript futtatókörnyezet, amely a Google által fejlesztett V8 motoron fut, és a JavaScript kódot a webböngészőn kívül hajtja végre.
- ***PostGIS*** - (https://postgis.net/) egy nyílt forráskódú szoftver, amely a geográfiai objektumok támogatásával bővíti a *PostgreSQL* objektum-relációs adatbázist. A PostGIS az Open Geospatial Consortium Simple Features for SQL specifikációját követi. Technikailag a PostGIS a *PostgreSQL* külső kiterjesztéseként került megvalósításra.
- ***PostgreSQL*** - (https://postgresql.org) egy nagy teljesítményű, nyílt forráskódú objektum-relációs adatbázis-rendszer, amely több mint 30 éve aktív fejlesztéssel rendelkezik, és amely a megbízhatóság, a funkciók robusztussága és a teljesítmény tekintetében komoly hírnevet szerzett magának.
- ***QGIS*** - (https://www.qgis.org/hu/site/) egy ingyenes és nyílt forráskódú, több-platformos asztali gográfiai információs rendszer alkalmazás, amely támogatja a térbeli adatok megtekintését, szerkesztését és elemzését.
- ***QGIS Server*** - A QGIS Server a QGIS asztali alkalmazás könyvtárain alapuló webes térképszolgáltatásokat biztosító szerver alkalmazás (https://docs.qgis.org/2.14/en/docs/user_manual/working_with_ogc/ogc_server_support.html) - *Lásd még 6. fejezetben*.
- ***R*** - (https://www.r-project.org/) egy ingyenes szoftverkörnyezet statisztikai számításokhoz és grafikus adat ábrázoláshoz. Sokféle UNIX-platformon, Windowson és MacOS-en fordítható és futtatható.
- ***R Shiny*** - (https://shiny.rstudio.com/) egy olyan R csomag, amely megkönnyíti az interaktív webes alkalmazások készítését közvetlenül az R-ből.
- ***REST API*** - Egy állapotmentes (stateless) szoftver architektúra megoldás, ahol a meglévő webes standardokat használjuk ki az alkalmazás-alkalmazás kommunikációban a szerveren vagy kliensen tárolt objektumok állapotának átvitelére. http://www.opengov.hu/tervezesi-elvek/restapi.html.
- ***SDK*** - szoftverfejlesztői készlet, szoftverfejlesztési eszközök gyűjteménye egy telepíthető csomagban. Ezek megkönnyítik az alkalmazások létrehozását, mivel rendelkeznek fordítóval, hibakeresővel és esetleg szoftverkerettel. Ezek általában egy adott hardverplatform és operációs rendszer kombinációjára jellemzőek.
- A ***Service Worker***ek virtuális proxy-t jelentenek a böngésző és a hálózat között. Megoldják azokat a problémákat, hogy hogyan lehet megfelelően gyorsítótárba helyezni a webhely eszközeit, és hogyan lehet azokat elérhetővé tenni, amikor a felhasználó eszköze offline állapotban van.
- ***Symfony*** - egy PHP webes alkalmazás keretrendszer és egy újrafelhasználható PHP komponensekből/könyvtárakból álló készlet. Szabad szoftver és a MIT licenc alatt adták ki.
- ***UI*** - User Interface - Felhasználói felület, *lásd még GUI*.
- ***Web worker*** - A World Wide Web Consortium (W3C) és a Web Hypertext Application Technology Working Group (WHATWG) meghatározása szerint a web worker egy HTML

oldalról futtatott *JavaScript* szkript, amely a háttérben fut, függetlenül az ugyanarról a HTML oldalról esetleg szintén futtatott szkriptektől. A web workerek gyakran képesek a többmagos CPU-kat hatékonyabban kihasználni.
A web worker specifikáció a HTML Living Standard része.

- ***WebAssembly*** - lehetővé teszi az előre lefordított kód futtatását a webböngészőben, közel natív sebességgel. Így az olyan nyelveken írt könyvtárak, mint például a C, C++, hozzáadhatók a webes alkalmazásokhoz.