ETDK-dolgozat

Budai Sámuel

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

Kolozsvár, 2021. május 27–30.

Agriary

Agrárgazdálkodási napló

Szerző:

Budai Sámuel

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

Témavezetők:

dr. Sulyok Csaba, egyetemi adjunktus

Babeş–Bolyai Tudományegyetem, Matematika és Informatika Kar

Miklós Botond, szoftverfejlesztő,

Codespring

Molnár Krisztina, szoftverfejlesztő,

Codespring

CODESPRING
OUR CODE YOUR SUCCESS

Kivonat

Valamennyi mezőgazdasági intézet rendszeresen követel részletes jelentést a gazdáktól, melyeknek többek között tartalmazniuk kell a birtokon végzett munkálatokat a vetéstől az aratásig, parcellákra lebontva. A gazdáknak viszont nagy kihívást jelent ezeknek az információknak a számontartása és a jelentések elkészítése.

Az Agriary projekt célja egy olyan felület létrehozása, ahol a mezőgazdászok nyomon követhetik a gazdaságban végzett munkálatokat és az ezekkel járó kiadást és bevételt.

Minden regisztrált felhasználónak lehetősége van bevezetni a saját parcelláit a hozzájuk tartozó információkkal. Ezekhez a már előre bevezetett növények megadásával vetési ciklusokat lehet hozzárendelni. A vetésforgó négy kategóriáját használva az alkalmazás javaslatokat tesz a felhasználónak az optimális vetésforgó elérése érdekében. A parcellákhoz műveleteket is rendelhetünk pl. szántás, permetezés, aratás. Az így bevezetett adatokat összegezve jön létre a jelentés. Ennek birtokában a gazdák sokkal gyorsabban elkészíthetik az éves kimutatásaikat, ezeket elemezve pedig nagyobb rálátást nyernek a befektetett pénz és energia, valamint a haszon kapcsolatára, ezáltal jövedelmezőbbé válhat a vállalkozás.

Tartalomjegyzék

Bevezető **1**

1. Háttér **3**

2. Az Agriary projekt **6**

2.1. Funkcionalitások 6

2.2. Kommunikációs modell 7

3. Az Agriary alkalmazás felépítése **9**

3.1. Az alkalmazásszerver 9

3.1.1. A szerver felépítése 9

3.1.2. Az adatmodellek 10

3.1.3. Vetésforgó 12

3.2. A webkliens 12

3.2.1. A kliens felépítése 12

3.2.2. Egyoldalas alkalmazás 13

3.2.3. A felhasználói felület 14

3.2.4. Útvonalválasztás és hitelesítés 15

4. Felhasznált technológiák **18**

4.1. Szerveroldali technológiák 18

4.2. Kliensoldali technológiák 20

5. Segédeszközök és munkamódszerek **24**

5.1. Segédeszközök 24

5.2. Munkamódszerek 26

6. Az alkalmazás működése **29**

Következtetések és továbbfejlesztési lehetőségek **34**

Bevezető

A mezőgazdászokról az a sztereotípia él, hogy olyan romantikus egyének, akik hajnaltól sötétedésig a friss levegőn dolgoznak, az állataikat és a növényeiket gondozzák, és nem sokat törődnek az élet nagy kérdéseivel. Azonban ez a kép kissé hamis. A nyilvánvaló teendők mellett manapság a mezőgazdászok sem kerülhetik el az adminisztratív munkát. Arra gondolva, hogy mire kell figyelni egy mezőgazdasági vállalkozás vezetésénél, a papírmunka valószínüleg a sokadrangú lenne a sorban. Az állami intézetek és a legtöbb magánszövetség, amellyel egy hétköznapi mezőgazdász együtt dolgozik, rendszeresen követelnek részletes jelentéseket a munkatársaitól. Romániában ilyen a *román állami mezőgazdasági ügynökség* (APIA), vagy a bio élelmiszereket felügyelő *Ecoinspect SRL*. Ahhoz, hogy állami támogatásokban részesüljenek, vagy megkapják a terményeikre a kívánt minősítést, a mezőgazdászoknak rendszeresen átfogó jelentéseket kell készíteniük. Ezeknek részletesen tartalmazniuk kell a mezőgazdaságban történt eseményeket, a növények ültetésétől kezdve a gondozásukon át egészen az aratásig. A jelentések elkészítése általában nagy fejtörést okoz a gazdáknak, mert mire sor kerül rá, az adatok nagy része vagy már nem áll rendelkezésre, vagy napokba telik összegyűjteni.

Ebben nyújt segítséget az Agriary alkalmazás. A rendszeresen vezetett agrárgazdálkodási naplóval a gazdaságban történt munkálatok mindig visszakövethetők. Ezek az információk számontartása nem csupán a bürokráciához tartozó haszontalan adminisztratív munka. A jelentéseket elemezve a gazdák messzemenő következtetéseket vonhatnak le, hogy hogyan vezessék nyereségesen a vállalkozást. A hosszú távú előnyök mellett az alkalmazásnak azonnali haszna is van. A gazdák nyomon követhetik, hogy a különböző parcellákon milyen munkálatokat végeztek, és ezáltal tisztábban átláthatóak a még elvégzésre váró feladatok. Továbbá a rendszer elemzi az elmúlt évek terményeit és a vetésforgó szabályai szerint ajánlatot is tesz a gazdáknak, hogy milyen növény kategóriát ültessenek a következő évben a legbőségesebb termés elérése érdekében.

Az alkalmazás olyan gazdaságok számára ajánlott, akik megélhetésszerűen foglalkoznak mezőgazdasággal, de még nem nőtték ki túlságosan magukat. Néhány hektáron nagyon nehéz jövedelmező gazdaságot kialakítani bármilyen segédeszközzel, a több tízezer hektáros óriásfarmok pedig saját applikációval rendelkeznek és évi szinten milliókat költenek optimalizálásra. Egy közepes (30-40 hektár feletti) gazdaság esetében azonban az Agriary alkalmazással is igen magas hatékonyságot el lehet érni, és rengeteg időt spórol meg az elfoglalt

gazdáknak.

A dolgozat felépítése a következő: az 1. fejezet ismerteti az alapvető mezőgazdasági elveket és kontextust teremt az alkalmazás funkcionalitásai fontosságának megértéséhez. A 2. fejezet a projekt funkcionalitásait és kommunikációs modelljét tárgyalja. A 3. fejezetben az Agriary-t alkotó szerver és kliens felépítése van elemezve. A 4. fejezetben a projekt létrejöttéhez használt technológiákról van szó. Az 5. fejezet tartalmazza a fejlesztés során használt segédeszközöket és ismerteti az alkalmazott munkamódszereket. A 6. fejezetben bemutatásra kerül az alkalmazás működése, az utolsó fejezetben pedig említésre kerül néhány továbbfejlesztési lehetőség.

A projekt fejlesztése 2020 júliusában indult a Codespringnél töltött nyári gyakorlat ideje alatt és 2020 októberében folytatódott a Csoportos projekt [6] tantárgy keretein belül. Köszönet illeti dr. Sulyok Csabát a szakmai irányításért, Molnár Krisztinát és Miklós Botondot a mentorálásért és Borsay Zsuzsannát a projekt ötletéért. Továbbá köszönet Szabó Andreának, aki a nyári gyakorlat alatt vett részt a fejlesztésben, valamint Poszet Kristófnak, Kunc Krisztofernek, Schuller Gergelynek, Pataki Krisztinának és Wegendt Erichnek, akik a Csoportos projekt alatt járultak hozzá az alkalmazás létrejöttéhez.

1. Háttér

Az Agriary fő célja a gazdák segítése abban, hogy a vállalkozásuk a lehető legjövedelmezőbb lehessen. A mezőgazdaság tipikusan olyan terület, ami más iparágakkal ellentétben nagyon nehezen válik jövedelmezővé. A gazdák ki vannak szolgáltatva az időjárás viszontagságainak és a piaci árak ingadozásának. Egy korai fagy, egy szárazabb vagy esősebb időszak, az üzemanyag- és takarmányárak: mind olyan tényezők, amik hatással vannak a profitra. Néhány százalék eltérés az optimális körülményektől könnyen eredményezheti az év veszteségességét.

A mezőgazdaság helyzete

A farmok többsége rá van szorulva az állami támogatásokra. Ez a jelenség világszinten is általánosnak mondható, viszont a román mezőgazdasággal ezen felül is problémák vannak. Románia rendelkezik az Európai Unió össz megművelhető földterületének 9%-ával [40], ezzel az értékkel európa szinten az ötödik helyen áll. A lakosság egyötöde mezőgazdasággal foglalkozik, viszont az ebből származó profit csak 4%-át teszi ki az éves GDP-nek. Ennek az előnytelen aránynak az egyik oka, hogy európában minden harmadik mezőgazdaság román, mi több, a mezőgazdaságok 86%-a főként saját fogyasztásra termel. Az átlag területe egy romániai farmnak mindössze 3,7 hektár, a régió más országaiban (pl. Bulgária) ez az érték ennek az 5-6-szorosa. Ekkora területen a kifizetődő gazdálkodás szinte lehetetlen.

A gazdálkodás másik véglete egészen más problémákkal küzd. A Föld legnagyobb farmja a kínai *Mudanjiang City Mega Farm* [8] a maga 9.1 millió hektárával. Ez az érték megközelítőleg Magyarország teljes területével egyenlő. Már az ennél jóval kisebb, néhány ezer hektárt felölelő farmok is a munkálatok elvégzéséhez saját alkalmazást használnak, közülük a legtöbb saját drón flottával és műholddal is rendelkezik [29]. Ekkora területeken az egészen kicsi eltérések is jelentősen megmutatkoznak a bevételben, például ilyen a traktorok útvonalának (néhány centiméteres) eltérése vetés vagy aratás közben.

A mezőgazdaság nyilvánvaló problémáinak függvényében egyértelmű, hogy szükség van az optimalizálásra. Az Agriary használata az olyan gazdák számára ajánlott, akik már nem esnek bele az említett 86%-ba, de túlságosan nagy gazdaságnak sem nevezhetőek. Területben kifejezve ez az érték 30-500 hektár közé tehető.

Jelentések

A farm helyzetéről készült jelentések elemzése az első lépés a profit maximalizálásának irányába. Rendszeres naplózás által követhető a mezőgazdasági munkák állapota. Néhány konkrét példát említve: számontartható, hogy melyik parcellák szántása vagy vetése van még hátra; az előző műveletek alapján kiszámolható a következő permetezés vagy kaszálás várható ideje; ütemezhetőek az aratások stb. Azáltal, hogy a gazdák idejében elvégzik a munkákat, hozzájárulnak a kiadások csökkentéséhez és a termés növeléséhez.

A mezőgazdászok munkáit gyakran felügyelik bizonyos állami vagy magán szövetségek. Romániában sok más mellett ilyen például a *román állami mezőgazdasági ügynökség* [2] (APIA), a bio élelmiszereket felügyelő *Ecoinspect SRL* [30] vagy a húsmarhák fajvédelméért felelős *Asociația Aberdeen Angus* [3]. Az együttműködés minden esetben anyagi haszonnal is jár. Az APIA az állami támogatásokat csak abban az esetben írja jóvá, ha a földterületeken történő munkálatok jól dokumentáltak. A partnercégek anyagi támogatás helyett bizonyítványokat (certificate) írnak jóvá a gazdák terményeire. Az Ecoinspect SRL felügyeli Romániában a bio vagy részben bio élelmiszereket. A minősítés megszerzéséhez a gazdáknak többek között jelentéseket kell készíteni, melyeknek tartalmazniuk kell a munkálatok pontos dátumát és az ezekhez használt módszereket, vegyszereket, vetőmagokat stb. A termények minőségét ezek a bizonyítványok garantálják, így a gazdák piaci előnyre tehetnek szert. A jelentéseket alkalomadtán a partnercégek szakképzett mérnökei is elemezhetik, és javaslatokat tehetnek a gazdáknak a nagyobb hatékonyság elérésére.

A vetésforgó

A mezőgazdaságban ha ugyanazt a növényt vetik több éven át egymás után egy adott földterületre, negatívan hat a hozamra. Ennek oka az, hogy az ugyanolyan kategóriába tartozó növények felborítják a talaj egyensúlyát és lefárasztják annak erőforrásait. A megoldás erre a problémára a vetésforgó, ami azt biztosítja, hogy egy bizonyos földterületen termesztett növények minél változatosabbak legyenek, ezzel maximalizálva a hozamot és fokozva a termelés hatékonyságát.

A vetésforgó négy növény kategóriát különböztet meg. A sor növények (row crop) csoportjába tartoznak azok a növények, amelyeket szigorú sorokba vetve termesztenek (pl. kukorica, burgonya, hagyma). Általában ezek értékesítése a legjövedelmezőbb, de ez azzal jár, hogy a talaj erőforrásait is nagyon kimerítik. A második kategória a hüvelyes

(legumes) növényeket foglalja magába (pl. lucerna, borsó, lencse). Ezeknek a növényeknek a különlegessége abban rejlik, hogy gyökérzetük megköti a levegőben található tápanyagokat, ezáltal gazdagítva a talaj erőforrásait. A következő kategória a fű- és búzafélék (grasses and cereals). Ide tartozik a közönséges takarmányfű mellett a zab vagy rizs. Az ilyen növények gyökérzete sűrün átszövi a talajt, ezzel dúsítva annak szerkezetét. Az utolsó kategória a biomassza (green manure) ami a talaj pihentetésére szolgál.

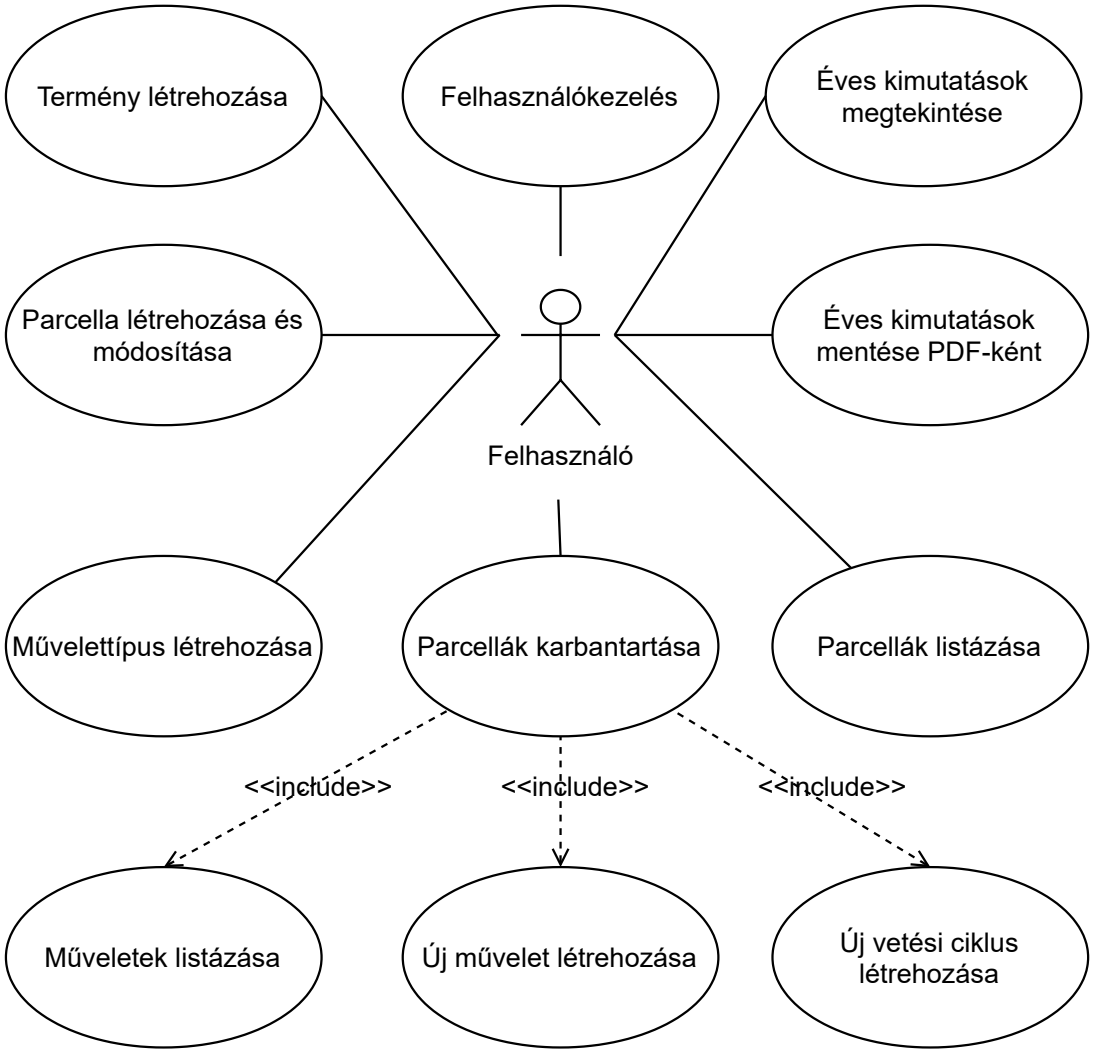
2. Az Agriary projekt

Ez a fejezet bemutatja az Agriary alkalmazást, a platform funkcionalitásait, valamint ismerteti a rendszer kommunikációs modelljét.

2.1. Funkcionalitások

Az Agriary rendszerben minden felhasználó egy farmot kezelhet. Vendégek belépése nem engedélyezett, így regisztrálás és bejelentkezés után minden felhasználó számára ugyanazok a funkcionalitások érhetőek el (ld. 1 ábra).

A bejelentkezett felhasználó hozzáfér a korábban mentett adataihoz, létrehozhat új adatokat, karbantarthatja a parcelláit, vagy szerkesztheti a profilját. Lehetőség van új parcella létrehozására ami a gazdák által megadott földterületre vonatkozik. Emellett be lehet vezetni



1. ábra. A felhasználó számára elérhető funkcionalitások az Agriary rendszerben.

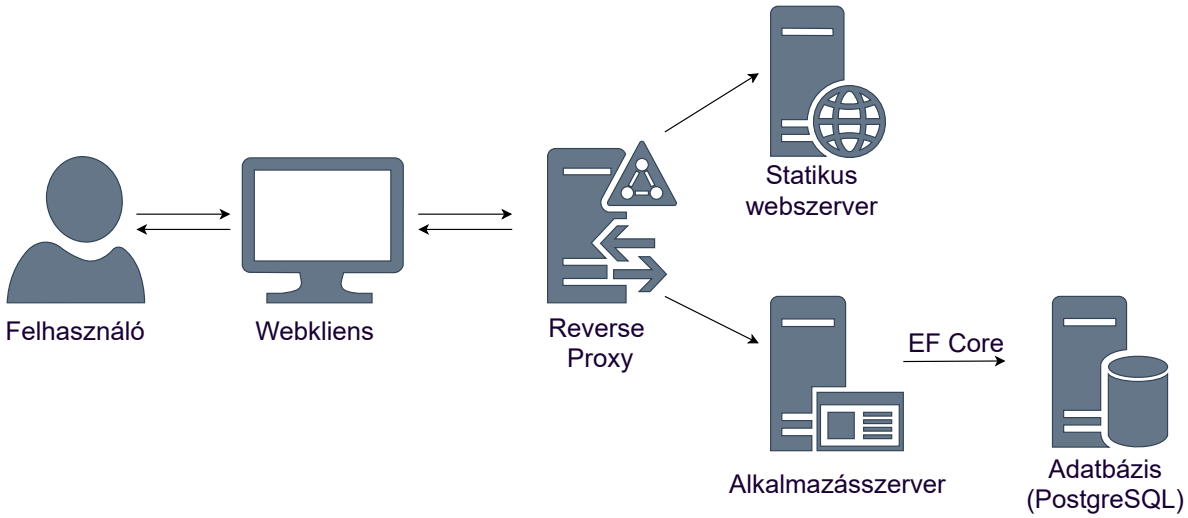
új terményt, ami a gazdaságban ültetett növények jellemzőit tárolja (pl. kukorica, zab) és új művelettípust, amiben a gazdaságban végzett munkálatok adatai vannak tárolva (pl. aratás, szántás). A parcellák listázásával lehetőség nyílik ezek törlésére, szerkesztésére és karbantartására. Karbantartás során először a parcella általános információit láthatjuk, mint a hozzátartozó kép, a parcella neve és más adatok, az elmúlt évek vetésforgójára vonatkozó diagram, valamint a parcellán végzett műveletek listája és a hozzájuk tartozó adatok.

Minden parcellához új vetési ciklust lehet hozzárendelni. Ez fogja jelezni, hogy mi van ültetve abba a földbe az adott évben. Itt a rendszer elemzi az elmúlt évek terményeit az adott parcellán és ennek alapján javaslatot tesz a felhasználónak, hogy az optimális vetésforgó és a maximális termés elérése érdekében milyen kategóriájú növény ültetése ajánlott. Így elindul egy vetési ciklus, amit aratáskor le lehet zárni. A vetési ciklusokhoz mezőgazdasági műveleteket lehet hozzáadni, megadva a hozzájuk tartozó adatokat. Ezek szerkesztésére és törlésére utólag is lehetőség van.

Az alkalmazás fő célja az év végi jelentések készítése. Itt a felhasználó kilistázhatja a gazdaságban történt összes műveletet, de lehetőség van ezek szűrésére parcella vagy év szerint. Az összegző táblázatok tartalmazzák a bevételekből és kiadásokból származó egyenleget és a parcellákhoz tartozó műveleteket.

2.2. Kommunikációs modell

A rendszer jól elkülöníthető komponensekből épül fel (ld. 2. ábra). A felhasználó közvetlenül a frontenddel áll kapcsolatban, ami egy React keretrendszerre épülő,



2. ábra. Az Agriary rendszer kommunikációs modellje

TypeScript-ben megírt webkliens (bővebben a 4.2. fejezetben). A frontend egy reverse-proxy-n [42] keresztül vagy a statikus webszerverhez, vagy az éles alkalmazásszerverhez irányítja a kéréseket. A lényeges különbség a reverse-proxy és a hagyományos proxy szerver között (forward-proxy) az, hogy míg a forward-proxy azt biztosítja, hogy a kliensek csak egy ellenőrző köztes rétegen át érjenek el bizonyos szervereket, úgy a reverse-proxy feladata az, hogy a konkrét alkalmazásszervert egy kliens se érhesse el közvetlenül. Ez a köztes réteg (middleware) olyan előnyökkel járhat, mint a bejövő adatforgalom egyenletes elosztása a különböző szerverek között (load balancing), támadásoktól való védelem (pl. DDoS[1]), vagy adatok tárolása (cache-elés). Ezek mind hozzájárulnak az alkalmazásszerver zökkenőmentes működéséhez, de az Agriary projekt esetében ez még fejlesztésre vár (6. fejezet).

Az alkalmazásszerver RESTful[2] API[3]-on [28] keresztül kommunikál a webklienssel, HTTP[4] protokollt használva. A REST egy olyan architekturális minta, ami bizonyos standardok betartása által egységesíti és könnyebbé teszi a kommunikációt a különböző rétegek között. RESTful minősítést akkor ér el egy (nem feltétlenül webalkalmazás specifikus) szolgáltatás, ha implementálja a REST standardjait. Ezek a következők: jól meghatározott URI[5] konvenciórendszer; két jól elkülönített, egymástól függetlenül fejleszthető oldal (kliens-szerver) amit a felhasználó nem érzékel; cache-elés lehetőségének biztosítása; állapotmentesség. Ezeket az elveket az Agriary rendszer megvalósítja. Például a rendszerben adott a következő útvonal:

```
/api/lands/19/operations
```

GET kérést küldve erre az útvonalra az alkalmazás visszatéríti JSON[6] formátumban a 19-es azonosítójú parcellához rendelt műveleteket.

Az alkalmazásszerver .NET Core-ban lett implementálva, ami az Entity Framework Core segítségével kommunikál az adatok tárolásáért felelős adatbázissal. Ez képezi a rendszer legalsó rétegét, ami egy PostgreSQL relációs adatbázis (további részletek a 4.1. fejezetben).

[1] Distributed Denial-of-Service
[2] REpresentational State Transfer
[3] Application Programming Interface
[4] HyperText Transfer Protocol
[5] Uniform Resource Identifier
[6] JavaScript Object Notation

3. Az Agriary alkalmazás felépítése

Az 2.2. fejezetben ismertetett topológia alapján látható, hogy az Agriary rendszer két főbb, jól elkülöníthető komponensből épül fel. A felhasználó kizárólag a frontenddel áll kapcsolatban. A megjelenítés teljes egészében itt valósul meg, valamint az adatfeldolgozás nagyrésze is. Ezzel szemben a backend réteg feladata elsősorban az adatok tárolása és mozgatása az adatbázis és a kliens között.

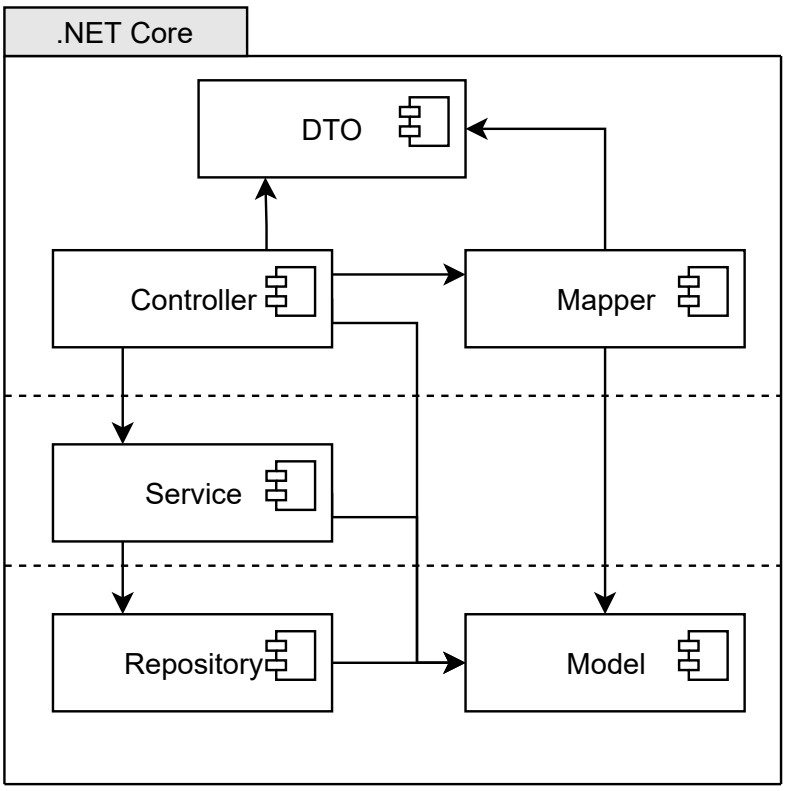
A fejezet részletezi a kliens és a szerver főbb komponenseit és felépítését.

3.1. Az alkalmazásszerver

3.1.1. A szerver felépítése

Az alkalmazásszerver három főbb, egymástól jól elkülöníthető gondolati egységből áll: Controller, Service és Repository (ld. 3. ábra).

A *Controller* réteg áll az alkalmazás legfelső szintjén, frontend kérései ide érkeznek. Ez a réteg nem tartalmaz vállalati logikát, vagy adatelérést. Elsődleges szerepe annak a biztosítása,



3. ábra. A szerver architektúrája. A szaggatott vonal jelzi a három jól elkülöníthető réteg egymástól való függetlenségét.

hogy a beérkező adatok helyesek legyenek, majd a bejövő kérések továbbítása az alsóbb rétegek felé, és a megfelelő API *responseok* visszatérítése.

A *Model* osztályokban van tárolva az alkalmazásban előforduló adatok sematikus ábrázolása. Az üzleti logika megvalósítása és az adatbáziselérés is a Model osztályokat használva történik.

A *DTO*-k[7] a kliensek és a szerver közötti kommunikációban használt adatszerkezeteket modellezik. Ennek szükségességét az indokolja, hogy bizonyos CRUD[8] műveleteknél az adott adatszerkezet tartalmazhat más (több vagy kevesebb) adattagot. Így az alkalmazásszerver megkülönböztet két csoporthoz tartozó DTO-t: bejövő (incoming) és kimenő (outgoing).

Egy *Mapper* osztály végzi a DTO-k és a Modell osztályok közötti oda-vissza történő átalakítást. Ez az átalakítás kizárólag a Controller rétegben szükséges, mivel itt történik a kommunikáció.

Az üzleti logikát a *Service* réteg tartalmazza. Ide érkeznek be a Controller kérései, és itt találhatóak azok a műveletek, amelyek az adatok feldolgozásáért felelősek. A Service mindig a kész választ téríti vissza a Controllernek, ami lehet a kért adatszerkezet vagy a művelet sikerességére utaló visszajelzés.

A *Repository* réteg tartalmazza az adatok eléréséhez szükséges logikát. Ide érkeznek be a Service kérései, és itt kerülnek felhasználásra a szűrő paraméterek (bővebben a 4.1. fejezetben). Ez a réteg közvetlenül az adatbázissal kommunikál.

3.1.2. Az adatmodellek

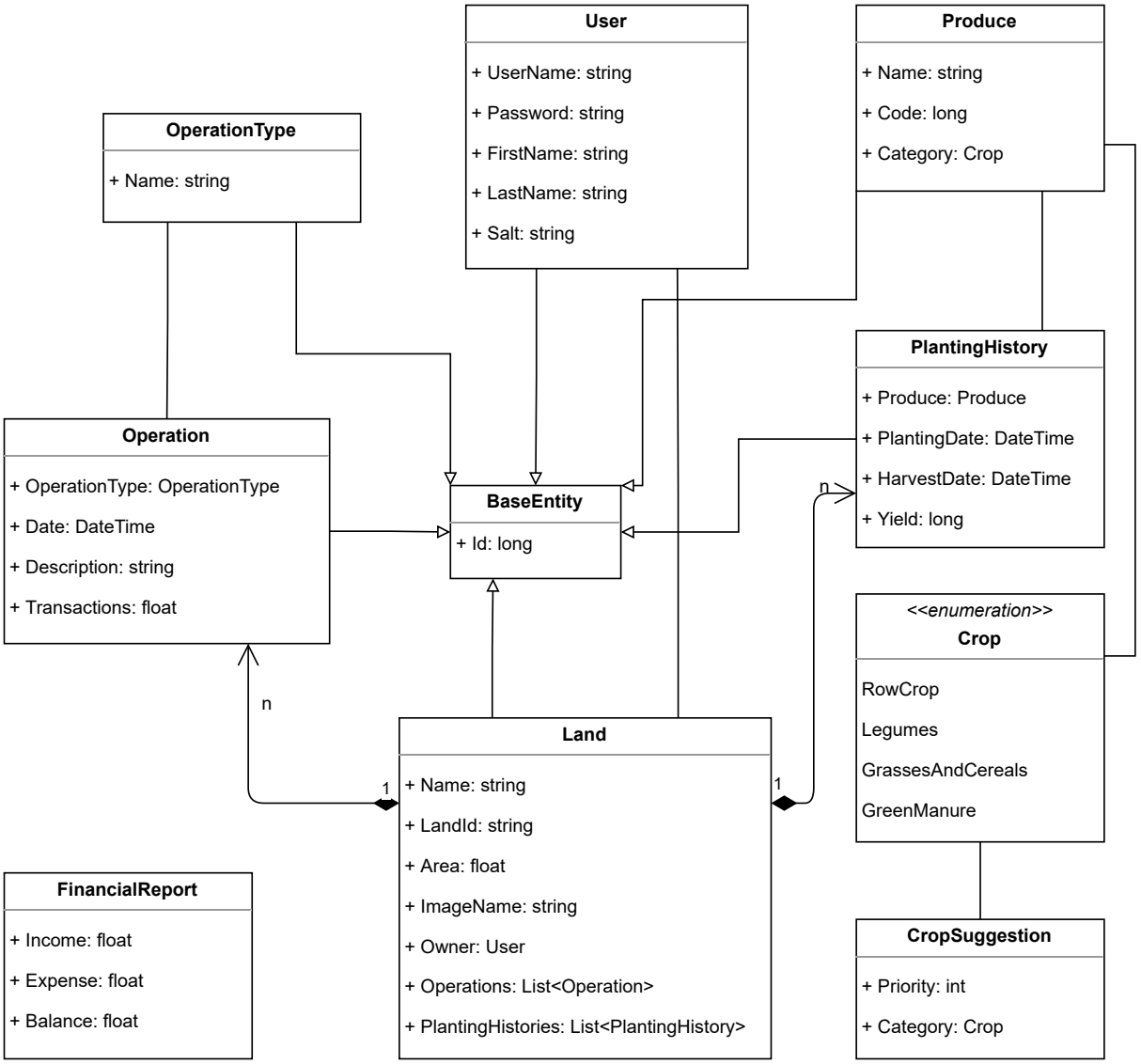
Az Agriary rendszerben a 4. ábrán látható adatmodellek vannak jelen. Ez az ábra tükrözi a különböző adatszerkezetek közötti kapcsolatot és azok tartalmát. Ez a fejezet csak a Model osztályokhoz tartozó reprezentációt tárgyalja, vagyis azt a sémát amit az üzleti logika során használ az alkalmazás, viszont az ezekre vonatkozó DTO osztályok sem tartalmaznak lényeges eltérést.

A rendszerben lévő valamennyi osztály ősosztálya a *BaseEntity*, amely egyetlen long típusú adattagot tartalmaz. Minden osztály ebből örökli a beazonosításához szükséges Id adattagot.

Az alkalmazás felhasználói egy *User* típusú táblában vannak tárolva. Ez tartalmazza a felhasználók személyes adatait (felhasználónév, vezetéknév, keresztnév) valamint az enkriptált jelszót és az ehhez felhasznált sót (bővebben a 3.2.4. fejezetben).

[7] DTO - Data Transfer Object

[8] CRUD - Create Read Update Delete. A perzisztens adatok kezelésére szolgáló négy alapművelet.



4. ábra. A vállalati logikában használt adatmodellek struktúrája

Az alkalmazás központi entitása a *Land*, ami a parcellák adatait tartalmazza, a többi entitás nagy része valamilyen szinten ettől függ. Minden parcella tartalmazza azt a felhasználót, akihez tartozik. Egy parcellának kötelezően tartalmaznia kell annak nevét, valamint területét, ami négyzetméterben van számontartva. Opcionálisan megadható a parcella kataszteri száma (*LandId*), ami egy hivatalos, országosan egyedi, a parcellákra vonatkozó azonosító. Az *ImageName* adattag a földterület beállított borítóképének azonosítóját tartalmazza. Minden parcellához több vetési ciklust (*PlantingHistory*) vagy műveletet (*Operation*) lehet hozzárendelni.

Az *OperationType* a rendszerben használt művelettípusokat képezi, eltárolva ezek nevét. Az *Operation* egy konkrét mezőgazdasági művelet adatait tartalmazza. Itt szerepel a művelettípus, a művelet elvégzésének dátuma, egy rövid leírás és a művelettel járó kiadás vagy bevétel. A

FinancialReport modell ezeket a kiadásokat és bevételeket összegzi és a végső egyenleget is tartalmazza (bővebben az 6. fejezetben).

A *PlantingHistory* a vetési ciklusokat tárolja. Egy vetési ciklus tartalmazza a rá vonatkozó növényt, a vetés és az aratás dátumát, és a hozamot kg-ban tárolva. Utóbbit csak aratás után lehet megadni.

A *Produce* a rendszerben használatos terményeket tárolja. Minden termény tartalmazza annak nevét és kategóriáját, valamint meg lehet adni a termény kódját is, ami egy hivatalos, a mezőgazdaságilag termeszthető növények azonosítására szolgáló számsorozat. A növények kategóriái egy *Crop* nevű enumerációban vannak tárolva. A *CropSuggestion* modellt használja a rendszer a vetésforgó ajánlásaira. Ebben a modellben a *Priority* jelenti azt, hogy hányadik ajánlás (1=legjobb, 2=második legjobb stb.) és a *Category* maga a terménykategória.

3.1.3. Vetésforgó

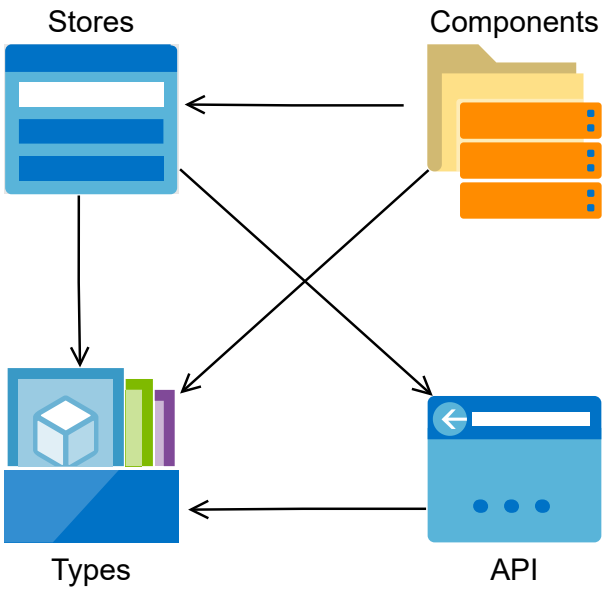
Az alkalmazás elemzi az elmúlt évek alatt egy adott földterületen termesztett növények kategóriáit, és egy algoritmus segítségével kiszámolja, hogy a vetésforgó betartásához milyen kategóriájú növény ültetése ajánlott a következő évben. Például, ha már két éve sor növények voltak termesztve, akkor az elsődleges ajánlás a hüvelyesek, másodlagos ajánlás a fű- és búzafélék lesz. Ha egy éve hüvelyesek voltak ültetve, de két éve fű- és búzafélék, akkor sor növények ültetése ajánlott. A cél a lehető legnagyobb változatosság elérése. Ezt a javaslatot a kliens egy *CropSuggestion* típusú tömbben kapja meg. Az ajánlás mellett az alkalmazás azt is elemzi, hogy egy földterület hogyan tartotta be a vetésforgó szabályait az elmúlt években. Erről egy statisztikát állít elő, amiből látható a parcella optimális kihasználtságának minősége (bővebben az 6. fejezetben).

3.2. A webkliens

3.2.1. A kliens felépítése

A webkliens négy fontosabb, jól elkülöníthető egységből épül fel, melyeknek kapcsolata az 5. ábrán látható.

A *Types* tárolja a rendszerben használt entitások kliens oldalra vetített modelljeit. Ezek megegyeznek a szerveroldali modellekkel, és itt is megkülönböztetünk vállalati logikában használt modelleket és DTO-kat.



5. ábra. A frontend architektúrája

A *Components* réteg tárolja a megjelenítéshez szükséges összes webes komponenst. Ez magába foglalja a navigációhoz tartozó elemeket és a stílusleíró fájlokat is. A komponensek Store-okat tarthatnak számon, amelynek változása maga után vonja az érintett komponens újratöltését (ld. 3.2.2. fejezet).

A *Stores* modul tartalmazza a komponensek állapot leírásához szükséges adattagokat és függvényeket. A Store az API-val áll kapcsolatban, így az állapota az API hívások után dinamikusan frissül, ha megváltozik a forrás tartalma.

Az *API* modul felelős a kommunikációért, itt jön létre a kapcsolat az alkalmazásszerver Controller rétegével. Az API aszinkron függvényeit a Store hívja meg a már előre elkészített model vagy DTO példányok átadásával. Az aszinkron függvények eredményeit vagy hibaüzeneteit pedig egy *Promise* [12] objektumból lehet kinyerni a visszahívó (*callback*) függvények segítségével (*then* és *catch*).

Az webkliens ezen kívül tartalmaz *resource* fileokat, melyek a nemzetköziesítésnél és a statikus borítóképeknél kerülnek felhasználásra, valamint gyakran használt *util* függvényeket.

3.2.2. Egyoldalas alkalmazás

A webfejlesztés korai fázisában a webalkalmazások *thin clientként* működtek. Ez azt jelenti, hogy az alkalmazásszerver a teljes weboldalt elküldte hálózaton keresztül a kliensnek, akinek feladata csak ezeknek a megjelenítése volt. A legkisebb adat változása maga után vonta a teljes weboldal újraépítését, amit az érintetlen komponensekkel együtt újra át kellett küldeni

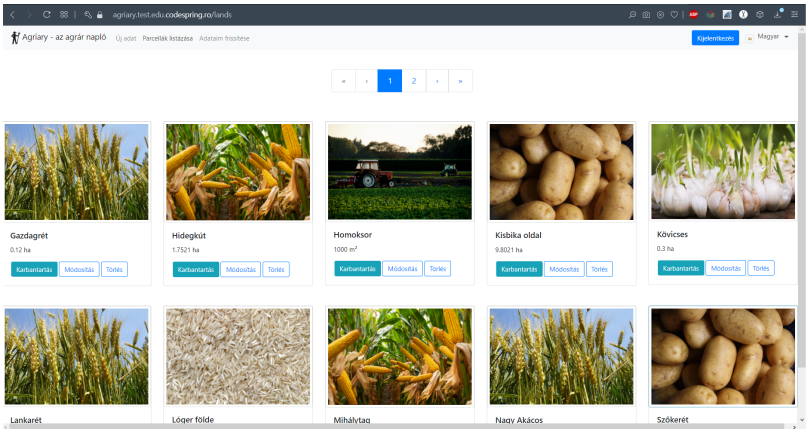
a kliensnek. Ezzel szemben az Agriary alkalmazás *rich clientnek* [35] tekinthető. Ennek az a jellegzetessége, hogy a megjelenítés a kliens oldalon történik, a böngészőben, a kliens számítógépének erőforrásait használva.

Emellett a webalkalmazás Single-page applicationként (SPA) [36] is kategorizálható. Az egész alkalmazás egyetlen oldalként fogható fel. A JSX[9] szintaxisú fájlok HTML, CSS és JavaScript fájlokká lesznek átalakítva. Ezt a böngésző az alkalmazás első betöltésekor betölti. Ezt követően a böngésző csak azt a komponenst rendereli újra a weboldalon, ahol változás történt.

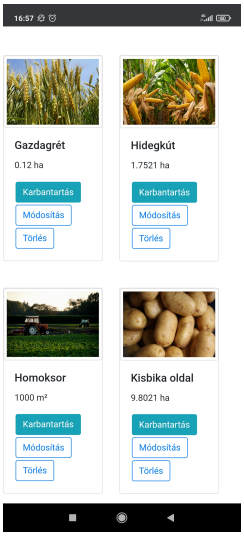
Ezen megközelítés által a frontend sokkal inkább hasonlít egy desktop alkalmazásra. Mivel csak a frissíteni kívánt komponenseket szükséges újratölteni, ezért mind a hálózati adatforgalom, mind az oldal frissítéséhez szükséges idő nagymértékben csökken, ezáltal jobb felhasználói élményt biztosítva.

3.2.3. A felhasználói felület

Az Agriary alkalmazás hatékony működése érdekében elengedhetetlenül fontos a mezőgazdasági adatok rendszeres és folyamatos naplózása. Mivel az alkalmazáshoz nem terveztünk mobil applikációt, ezért a fejlesztés során ügyeltünk arra, hogy az alkalmazás reszponzív legyen az asztali számítógépektől eltérő eszközökön is, hogy akár a traktoron ülve, egy mobilos böngészőből is be lehessen vezetni az adatokat.



(a) A kártyák elrendezése desktop felületen



(b) Mobil böngészőben

6. ábra. A parcellákat tartalmazó kártyák elrendezése különböző méretű képernyőkön

[9] JSX - JavaScript XML

```
<Row lg={5} md={3} xs={2}>
    state.lands.map((land: Land) => (
        <div key={land.id}>
            <LandCard land={land} />
        </div>
    ))}
</Row>
```

1. kódrészlet. Példakódrészlet az alkalmazás reszponzívitásának biztosítására. A Row tagnek megadható, hogy különböző méretű képernyőkön hány elem jelenhet meg egy sorban

Az SPA princípium, miszerint a komponenseket a böngésző építi fel a saját erőforrásait felhasználva, megkönnyíti a reszponzivitás biztosítását is. A 6. ábrán egy példa látható, ahol ez alkalmazásra került. Az parcellák kártyák formájában vannak megjelenítve a felhasználó számára (bővebben az 6. fejezetben), de azt, hogy egy sor hány kártyát tartalmaz az eltérő méretű képernyőkön (laptop, táblagép, telefon, stb.) külön biztosítani kell (ld. 1. kódrészlet).

Annak érdekében, hogy az alkalmazás egy szélesebb felhasználói réteg számára használható legyen, a felület nemzetköziesítése is biztosított. Az Agriary a *react-intl* [11] könyvtár segítségével tárolja és jeleníti meg a szövegeket, egyelőre angol vagy magyar nyelven.

Az alkalmazás szülőobjektuma egy *IntlProvider* komponens, amely saját storeban tárolja az aktuálisan kiválasztott nyelvet. A legördülő listából kiválasztott egyéb nyelv esetén az oldal szövegmezői újratöltődnek az újonnan kiválasztott nyelven.

A szövegek locale fileokban, kulcs-érték párokban vannak tárolva, minden nyelvnek külön JSON állományban. Egy új nyelv bevezetése esetén egy npm script létrehozza az új locale fájlt, ahol csak le kell fordítani a szövegeket. A komponensekben egy-egy szövegrészletre a következő módokon lehet hivatkozni:

```
<FormattedMessage {...messages.hello} />
placeholder={formatMessage(messages.hello)}
```

Az első módszer olyan esetekben használatos, amikor egy kívánt szöveget komponensként akarunk kezelni (pl. egy címke tartalma), a második módszer pedig akkor, amikor a szövegre kifejezetten stringként van szükség (pl. egy függvény paramétere).

3.2.4. Útvonalválasztás és hitelesítés

Mivel az Agriary egyoldalas webalkalmazás, ezért szükség van egy routing (útvonalválasztó) mechanizmusra annak érdekében, hogy követni lehessen, a felhasználó

```
if (!state.loggedIn) {
    return (
        <Switch>
            <Route component={RegisterPage} path="/register" />
            <Route component={LoginPage} path="/login" />
            <Redirect to="/login" />;
        </Switch>
    );
} else {
    return (
        <Switch>
            <Route component={Home} exact path="/" />
            <Route component={NewProduce} path="/produce" />
            <Route component={NewLand} path="/newland" />
            <Route component={Lands} path="/lands" />
            {...}
        </Switch>
    );
}
```

2. kódrészlet. A *Routes* állomány egy részlete. A nem bejelentkezett felhasználók csak bizonyos oldalakat érhetnek el

aktuálisan a frontend melyik részét használja. Ezt az útvonal dinamikus változásának követésével lehet elérni.

Az alkalmazás a *react-router-dom* [27] könyvtár segítségével valósítja ezt meg. A webkliens (az előző fejezetben említett *IntlProvider* komponenst is beleértve) egy *BrowserRouter* komponens gyereke. Ezáltal a böngésző előzményekben tárolja az útvonal változását, és visszalépés esetén a felhasználó mindig az előzőleg látogatott oldalra lesz irányítva, annak eredeti komponenseivel. Az alkalmazásban elérhető útvonalak a *Routes* állományban vannak tárolva, az alkalmazás ezek alapján választja ki a betöltésre kerülő oldalt (ld. 2. kódrészlet).

Az Agriary rendszerbe csak regisztrált felhasználók léphetnek be, ezért szükség van a weboldal látogatóinak hitelesítésére is. Ez a mechanizmus biztosítja, hogy minden felhasználó csak a saját adataihoz férhessen hozzá. Az 2. kódrészleten megfigyelhető, hogy az alkalmazás ellenőrzi a kliens bejelentkezésre vonatkozó állapotát, és az alapján dönti el, hogy milyen útvonal érhető el számára.

Regisztrációkor az alkalmazás a felhasználó jelszavát *hash*elt formában menti el az adatbázisba. Ehhez a jelszó mellett felhasznál egy random módon generált karakterláncot ("*só*"). Bejelentkezéskor a rendszer a felhasználó által megadott jelszót és a beírt felhasználónévhez tartozó, adatbázisba mentett *sót* ugyanezzel a módszerrel *hash*eli el, és azt vizsgálja hogy

az így kapott eredmény egyezik-e az adatbázisban szereplő hashelt jelszóval. Ezt követően a szerver JWT[10] tokent [5] generál, amit a kliens sütiben tárol, így az oldalon történő minden navigálásnál a szerver tudni fogja, hogy hiteles-e az aktuális felhasználó. Egy JWT három részből áll: base64-el kódolt fejlécből (*header*), a felhasználó base64-el kódolt személyes adataiból (*payload*) és az aláírásból (*signature*). A *signature* a tokennek az a része, amely azt biztosítja, hogy illetéktelenül ne lehessen módosítani, ugyanis ez egy olyan kriptográfiai függvény eredménye, amihez felhasználásra kerül a kódolt header és payload, valamint az alkalmazásszerveren tárolt titok (*secret*) is.

A hitelesítést biztosító sütikhez csak az alkalmazászerver férhet hozzá. A kliens az *api/introspect* útvonalra küldött API hívás által tudhat meg információkat a bejelentkezés állapotáról.

[10]JWT - JSON Web Token

4. Felhasznált technológiák

Az Agriary alkalmazás backend része C#-ban íródott a .NET Core keretrendszer segítségével, a frontend pedig a React könyvtárral TypeScript-ben. Ez a fejezet részletesen ismerteti az alkalmazáshoz használt technológiákat.

4.1. Szerveroldali technológiák

ASP.NET Core

Az ASP.NET Core [7] egy rendszeresen karbantartott, népszerű, ingyenes és nyílt forráskódú szoftver keretrendszer webes alkalmazások fejlesztésére, amely használható Windows, Linux és macOS operációs rendszereken, valamint virtuális Docker környezetben.

A keretrendszer legfőbb erőssége a teljesítményben rejlik. Ez elsősorban annak köszönhető, hogy a nyelv támogatja az aszinkron hívásokat és a funkcionális programozási paradigmákat (ld. 3. kódrészlet). A független *TechEmpower benchmarks* [41] felmérése szerint a .NET alatt futó szerverek sebessége a sokszorosa az olyan népszerű webes keretrendszerekéhez képest, mint a Spring vagy a Node.js. Az ASP.NET Core rendelkezik saját beépített függőségkezelő rendszerrel (*NuGet*). Emellett lehetőséget ad a különböző környezetek közötti váltakozásra (*development, production*, stb...).

PostgreSQL

A PostgeSQL [22] egy több operációs rendszeren is elérhető relációs adatbázis-kezelő rendszer, amit többek között gyakran használnak webalkalmazások fejlesztésére. Ez több mint 30 éve egy kaliforniai egyetemi projektként indult és nincs irányítva egy cég által sem, a rendszer fejlesztését és karbantartását az open-source közösség biztosítja. Az alkalmazás nyílt forráskódú és rendszeresen karbantartott, a 13-as verzió megfelel szinte az összes SQL:2016 Core követelménynek [34]. A nyílt forráskód mellett az adatbázis-kezelő rendszer erőssége a robosztusságban, a megbízhatóságban, az egyszerű használatában és a skálázhatóságban rejlik, viszont nagyobb alkalmazásokhoz használva a teljesítménye gyengébb lehet, mint más, népszerűbb rendszereknek (pl. MySQL). Emellett a PostgreSQL a széles körben ismert SQL[11] nyelvet használja kombinálva saját fejlesztésekkel, lehetőség van benne saját adat- és indextípusok létrehozására és saját bővítmények írására.

[11] SQL - Structured Query Language

Entity Framework Core

Az Entity Framework (EF) Core [18] a cross-platform változata a csak Windows-on elérhető Entity Framework adatelérési technológiának. Ez egy *ORM*[12] rendszer, amely lehetőséget biztosít arra, hogy .NET környezetben az adatbázissal C# objektumok segítségével lehessen dolgozni. Az EF Core többfajta modellfejlesztési megközelítést is támogat. Lehetőség van a modelleket kézzel kódolni, hogy megfeleljenek a szintén kézzel kódolt adatbázisnak. Kigenerálhatóak a modellek egy létező adatbázisnak megfelelően, ezt *database-first* eljárásnak nevezzük. Az Agriary fejlesztése során viszont a *code-first* eljárás volt követve. Ez azt jelenti, hogy a modellek implementálása után létrehozhatóak az adatbázis táblái úgynevezett *migrációk* [20] segítségével. Ezzel a módszerrel az adatbázis karbantartása is megoldott, ugyanis a modellek változása után a migration frissíti az adatbázis szerkezetét is.

A Model osztályokat használva az adatokat natív SQL kód nélkül is el lehet érni, így könnyen elkerülhető az inkompatibilitás a különböző SQL "dialektusok" között. Az EF Core által támogatott adatbázisok adatait *LINQ*[13] [19] segítségével lehet elérni, amely funkcionális programozási stílusban alkalmazza a táblára a szűrőfeltételeket (ld. 3. kódrészlet).

```
public async Task<IEnumerable<Land>> FindAllByOwnerIdAsync(long ownerId,
int page, int pageSize)
{
    return await _context.Lands.AsNoTracking()
        .Where(l => l.OwnerId == ownerId)
        .OrderBy(s => s.Name)
        .Skip((page-1)* pageSize)
        .Take(pageSize)
        .ToListAsync();
}
```

3. kódrészlet. A LandRepository osztály egyik függvénye. Az alkalmazás aszinkron módon kommunikál az adatbázissal. A tábla szűrve lesz a felhasználó szerint, rendezve név szerint, majd a *pagination*-re vonatkozó logika is elvégzésre kerül.

[12] ORM - Object-Relational Mapping
[13] LINQ - Language INtegrated Query

4.2. Kliensoldali technológiák

TypeScript

A TypeScript (TS) [1] egy kiterjesztett változata a JavaScript (JS) nyelvnek, mely lehetővé teszi dinamikus weboldalak létrehozását. A TypeScriptben lehetőség van típusok, osztályok és interfészek használatára. Az erős típusozásnak köszönhetően a fejlesztői környezetek statikus kódellenőrzői észlelik az esetleges hibaforrásokat, így azok javíthatók még azelőtt, hogy futtatáskor előkerülnének, ezáltal a szoftverfejlesztés is lényegesen hatékonyabbá válik.

Fordításkor a TypeScript kód JavaScript kóddá alakul át a *transpiler* által, tehát a helyes JS kód helyes TS kódnak is számít. Ha JS alapú könyvtárakkal akarunk dolgozni TS környezetben, a nyelv az adatszerkezeteket az általános any típussal fogja kezelni. A pontos típusdefiníció meghatározása érdekében szükség van az ezekre vonatkozó *npm* csomagra, amely leírja a könyvtár szerkezetét és típusait.

React

Eleinte a Facebook által fejlesztett, majd nyílt forráskóduvá tett React [10] az egyik legnépszerűbb JavaScript alapú könyvtár interaktív webes felhasználói felületek készítésére. A felhasználói felület alkotóelemei a komponensek. Ezek a kisebb, többször felhasználható, egymásba ágyazható kódrészletek tartalmazzák a VDOM[14] szerkezetére vonatkozó információkat, és általuk az UI[15] különböző részei dinamikusan frissíthetők az alkalmazás belső információinak változása esetén.

A React alkalmazások készítéséhez a hagyományos JavaScript kód helyett ajánlott az úgynevezett JSX[16] szintaxis használata, mivel az utóbbival a komponensek kezelése egyszerűbbé válik (4. kódrészlet[17]). Az XML típusú stílus a HTML[18] tag-ekre emlékeztet, viszont a leíró nyelvvel ellentétben ez teljes értékű programkódnak számít. Ezeknek a komponenseknek bármilyen adatszerkezet átadható paraméterként, így a kommunikáció is strukturáltabbá válik. Fordításkor a JSX kód is JavaScript kóddá alakul, így teljesítménybeli különbség nincs.

[14] VDOM - Virtual Document Object Model
[15] UI - User Interface
[16] JSX - JavaScript XML
[17] Forrás: `https://reactjs.org/docs/components-and-props.html`, utolsó elérés dáruma: 2021-04-15
[18] HTML - HyperText Markup Language

```
class Welcome extends React.Component {
  render() {
    return <h1>Hello, {this.props.name}</h1>;
  }
}
```

4. kódrészlet. JSX kód szerkezete

MobX

A webes alkalmazások állapotának karbantartása kiemelt fontosságú, mivel az alkalmazásban jelen levő adatok nagyban befolyásolják a felhasználói felületet. A MobX [21] egy népszerű, skálázható, főként React alkalmazásokhoz használt állapotkezelő rendszer. Az szoftver alapgondolata szerint *"Mindent, amit az alkalmazás állapotából származtatni lehet, származtatni is kell. Automatikusan"*. Tehát a MobX állapotkezelő rendszert használva az állapot és a felhasználói felület közötti szinkronitás mindig biztosított.

Az Agriary a MobX által javasolt *observer-observable* tervezési mintát használja. Egy store a különböző entitásokhoz tartozó adattagokat és műveleteket csoportosítja. Ezek a 3.2. fejezetben említett Stores rétegben tárolt fájlokban vannak számon tartva. A komponenseket egy *observer* dekorátor-függvénybe kell burkolni. Ez gondoskodik arról, hogy a state-ben történő bármilyen változás maga után vonja a komponens újra renderelését. A MobX-nek a *useContext()* [23] *hook*-kal jelezzük, hogy a komponens melyik store-t használja állapotnak, ahogy az az alábbi kódrészleten látszik.

```
export const LandForm: React.FC<Props> = observer(({ land }) => {
    const state = useContext(LandStoreContext);
    {...}}
```

A store elemeit az *@observable* [26] dekorátorral kell ellátni, ha azt szeretnénk, hogy az őt használó komponensek észleljék a változását. Ezeket az elemeket nem lehet a store-on kívül változtatni. Az *@action* dekorátort a store függvényeire kell használni. Ezeket a függvényeket lehet meghívni a komponensekből. Az action-ök változtathatják a store elemeit indirekten API hívások által, vagy a komponensekben megadott értékre is be lehet állítani (ld. 5. kódrészlet).

A MobX-től függetlenül lehetőség van komponens szintű, egyszer használatos állapotok létrehozására is a *useState()* React hook által.

```
export class LanguageStore {
        @observable
        language: Locales = Locales.HUNGARIAN;

        @action.bound
        setLanguage(lang: Locales) {
                this.language = lang;
        }
}
```

5. kódrészlet. A nemzetköziesítéshez használt store. A felületen a legördülő listából egy elemre kattintás maga után vonja a *setLanguage* függvény meghívását, ami beállítja a *language* elemet. Ha ez változott az előzőhöz képest, az maga után vonja a komponens újra renderelését

Bootstrap

Webalkalmazások fejlesztése során a teljesítmény mellett nagy hangsúlyt kell fektetni a felület kinézetére is, hogy annak használata kényelmes legyen a lehető legtöbb eszközön. A Bootstrap [38] a legnépszerűbb, ingyenes, open-source CSS[19] keretrendszer reszponzív webes felületek készítésére, amit kezdetben a Twitter részére fejlesztettek.

Az erős CSS keretrendszerek megjelenése előtt az oldalon elhelyezni kívánt elemeket táblázatokba kellett rendezni, a különböző komponensek pedig a táblázat celláiban kaptak helyet. A Bootstrap egyik erőssége, hogy saját rács rendszerrel rendelkezik [4]. Az egyetlen megkötés ezzel kapcsolatban az, hogy az oszlopok a sorok "gyerekei" kell legyenek és összesen 12 oszlop lehet egy sorban, amiket tetszés szerint lehet kiosztani a komponenseknek. Tartalommal feltöltve az így felépített rácsszerkezetet átlátható, barátságos, és főként reszponzív UI-t lehet kiépíteni.

A Bootstrap több előre definiált stílusú komponenst is magával hoz. Megadva egy komponens *class* attribútumának a megfelelő értéket, aktiválni lehet rá a kívánt stílust. Reacttal használva szükség van a *react-bootstrap* [24] könyvtárra, amely HTML tagek helyett valódi React komponenseket használ.

Formik

Webes alkalmazásoknál szinte elengedhetetlen az űrlapok (*formok*) használata. A Formik [16] a legnépszerűbb, open-source, Reactra fejlesztett keretrendszer az űrlapok kezelésére. A Formik erőssége, hogy megoldja az űrlapokkal járó problémák többségét: számon

[19] CSS - Cascading Style Sheets

tartja a bevezetett értékeket, megjegyzi a látogatott mezőket, hibakezelést végez és kezeli az adatok beküldését (*submit*). Egy ilyen formot kezdeti értékekkel kell ellátni, és a Formik ezeket kezeli egészen a beküldésig. Ez azt jelenti, hogy az űrlapokon belül nincs szükség külső állapotkezelő rendszerre (MobX), hanem elegendőek az alap React state-ek és props-ok.

5. Segédeszközök és munkamódszerek

5.1. Segédeszközök

A programozói munka a megfelelő eszközök kiválasztásánál kezdődik. A jól működő, sokak által ismert és széles körben támogatott rendszerek nagymértékben megnövelik a munka hatékonyságát és lényegesen csökkentik a fejlesztésre szánt időt. Az alfejezet ezeket a segédeszközöket mutatja be.

Fejlesztői környezetek

A webkliens fejlesztéséhez a Visual Studio Code (VSCode) volt használva. Ez egy kis memóriaigényű, könnyen kezelhető, nagymértékben személyre szabható szövegszerkesztő. Képes az intelligens kód kiegészítésre, a statikus kódellenőrzésre és rendelkezik beépített Git támogatással.

Az Agriary backend része a Microsoft által fejlesztett Visual Studio segítségével volt fejlesztve. Ez az IDE[20] többek között olyan alkalmazások fejlesztésére használatos, melyek C#, C++ és F# nyelvekben íródtak, de ezeken kívül egyéb nyelveket is támogat. A szövegszerkesztő egyebek mellett rendelkezik beépített intelligens kódkiegészítő és -refaktoroló rendszerrel.

A legtöbb applikáció esetében a backend és a frontend két különálló, egymástól független egységként van fejlesztve, amit az API (jelen projekt esetén REST API) köt össze. Ennek okán a backend endpointjait külön is lehet tesztelni. A Postman segítségével egy grafikus felületen REST hívásokat lehet szimulálni, amelyekkel egyszerűbben tesztelhetőek az alkalmazásszerver funkcionalitásai.

Verziókövetés

A nagy vállalati alkalmazásoktól az egészen kicsi projektekig mindenhol szükség van a kódbázis fejlődésének követésére. A Git [17] napjaink legnépszerűbb osztott verziókövető rendszere, amelyet olyan óriáscégek is használnak, mint a Google, a Facebook vagy a Microsoft. A lényeges különbség a Git és más verziókövető rendszerek között (pl. Mercurial) a repositoryk (tárolók) osztott jellegéből fakad. Minden felhasználó gépén megtalálható a teljes repository historyja, beleértve minden ágat és a hozzájuk tartozó commitokat is. Ennek előnye, hogy a legtöbb művelet elvégezhető lokálisan is és nem igényel internetkapcsolatot.

[20] IDE - Integrated Development Environment

A Git másik sajátossága adatok változásának kezelésében rejlik. A régebbi verziókövető rendszerek zöme (pl. Subversion, Perforce) a változásokat tárolták, és ez alapján építették fel a különböző verziókat. Ezzel ellentétben a Git a teljes fájlrendszerről készült pillanatképet (snapshot) tárolja. A hatékonyság növelése érdekében az olyan fájlokról, amelyekben nem történt változás, egy referenciát tárol, ami az eredeti fájlra mutat.

A GitLab [43] egy Git alapú tárolók kezelésére használt webes alkalmazás és projektmenedzsment eszköz, továbbá grafikus felületet biztosít az alkalmazás *DevOps* életciklusaihoz. A hagyományos Git funcionalitásokon felül lehetőség van wiki oldalak készítésére, folyamatos integrációra és kitelepítésre (CI/CD), taszkok menedzselésére, a kódbázis felülvizsgálatára, a csapattagok közötti kommunikációra, stb.

Docker

Komplex alkalmazások telepítésénél sok probléma merülhet fel, többek között az operációs rendszer inkompatibilitása, eszközök pontos verziója, környezeti változók és admin jogok. Erre szolgáltat megoldást a Docker [14], ami egy 2013-ban megjelent virtualizációs eszköz alkalmazások kitelepítéséhez és futtatásához. A Docker operációsrendszer-szintű virtualizációt biztosít, ezáltal megoldva a kitelepítéssel járó hagyományos problémákat. Az alkalmazásokat Docker konténerekben lehet futtatni. Minden konténer egy létező imageből indul ki, ami egy konténer lefagyasztott állapota. Használni lehet saját, vagy mások által létrehozott imageket, utóbbit az online image tárolókról lehet letölteni, melyek közül a legnépszerűbb a DockerHub [9]. A perzisztens adatok tárolására volumeok használhatók, így a fontos adatok (pl. adatbázis) nem vesznek el a konténer újraindítása vagy törlése esetén.

Nagyobb projekteknél több különálló konténer futtatása szükséges. A Docker Compose [15] egy olyan eszköz, amivel megoldható ezeknek a konténereknek a beállítása és karbantartása. A konténerek a szervergép virtuális hálózata által állnak kapcsolatban, és ezek között a Docker DNS-e[21] biztosítja a kommunikációt. Ezzel a hálózattal a külvilág egyetlen porton keresztül kommunikálhat (bővebben a 5.2. fejezetben).

A Docker használata Linuxon ajánlott, Windowson és Macen a Docker Desktop [13] alkalmazás biztosítja a támogatást, ahol a Docker egy silent virtuális gépen fut. A Docker Desktop emellett tartalmazza a Docker-Compose és a Kubernetes legújabb verzióját, valamint lehetőséget biztosít natív Linux környezetben dolgozni a WSL2[22] által.

[21] DNS - Domain Name System
[22] WSL - Windows Subsystem for Linux

5.2. Munkamódszerek

A hatékony fejlesztői munka elérése érdekében elengedhetetlenül fontos a jól bevált és standardizált praktikák alkalmazása. Az alfejezet ismerteti az Agriary fejlesztése alatt használt módszereket.

Agilis szoftverfejlesztés

Az informatikai projektek sikere nagymértékben függ a fejlesztői csapat és a megrendelő közötti gördülékeny együttműködéstől és a jó kommunikációtól. 2001-ben 17 szoftvermérnök egy konferencián létrehozta a *Manifesto for Agile Software Development* [37] című kiáltványt, melyben ismertették a hatékonyabb szoftverfejlesztéshez szükséges alapelveket. Ez az új gondolkodásmód a megrendelők elégedettségét és a fejlesztői csapat motiváltságát helyezte előtérbe, olyan gyakorlatok bevezetése által, mint a szoftver új, működő verziójának szállítása rövid időközönként, vagy a fejlesztők közötti folyamatos kommunikáció. Manapság már a legtöbb szoftverfejlesztő vállalat alkalmazza valamilyen formában az agilis módszereket.

A *Scrum* [25] egy agilis keretrendszer amely a vízesés modell lineáris menedzsmentjével ellentétben iteratív fejlesztést használ. Elsősorban ez a módszer határozta meg az Agriary fejlesztése során a csapat munkamenetének felépítését. A Scrum csapat kulcsszereplői a *Product Owner* (PO), aki a terméket megrendelők érdekeit képviseli; a *Scrum Master*, akinek a Scrum ceremóniák vezénylésében van fontos szerepe; valamint a fejlesztői csapat tagjai. A folyamat első lépése a *user story*k létrehozása, amely a szoftverrel szemben támasztott megrendelői igényeket tartalmazza, hétköznapi nyelven megfogalmazva. Ezeket a feladatokat a csapattagok közös megegyezés alapján *story point*okkal látják el, amely a feladat komplexitására vonatkozó érték. Ezt a műveletet *esztimálásnak* nevezzük.

A munkafolyamat kettő-három hetes szakaszokba, úgynevezett *sprintekbe* rendeződik. Minden ilyen ciklus a *sprint planning*el kezdődik amit a scrum master irányít. A csapattagok eldönthetik, hogy milyen user storykat foglalnak bele a sprintbe, figyelembe véve azok story pontjait, és a storyk fontossági sorrendjét, amit a PO-val közösen határoztak meg. A ciklus a *demo*val zárul, ahol a csapattagok bemutatják az elkészült funkcionalitásokat. Egy sprint akkor számít sikeresnek, ha minden tervezett user story megvalósításra kerül, és kivétel nélkül minden funkcionalitás elfogadásra került a PO által. Ezután kerül sor a *retrospective*re, ahol a Scrum csapat minden tagja megoszthatja a pozitív és negatív élményeit a sprinttel kapcsolatban, és javaslatokat fogalmazhat meg a következő sprintre vonatkozóan.

A Scrum keretrendszerrel jól használható agilis segédeszköz a *kanban board* [39], amely a szoftverfejlesztéstől független, egyéb vállalatok esetében is hozzájárulhat a munka irányításához. A módszer lényege a teendőkre vonatkozó folyamatos vizuális visszajelzés. Az Agriary fejlesztése alatt a feladatok a kanban tábla oszlopaiba voltak rendezve, ezzel elősegítve az átláthatóságot. Az *Open* oszlopba kerülnek azok a user storyk, amelyek elkezdésére még nem fog sor kerülni a következő sprintben. A *To do* oszlopban azok a feladatok vannak, amelyeket az aktuális sprint alatt végre kell hajtani. A csapat tagjai innen választhatnak maguknak feladatot. Az elkezdett user storyk a *Doing* oszlopba kerülnek azok befejezéséig. Ha egy feladat kész, az átkerül a *Review* oszlopba. Ezeket a csapat egy vagy több olyan tagja, aki nem vett részt a feature fejlesztésében (jelen projekt esetében a mentorok) felülvizsgálja, és miután mindent megfelelőnek talált, a task átkerül a *Merged* oszlopba. Azok a taskok, amelyek a demo során elfogadásra kerültek, a *Closed* oszlopban kapnak helyet véglegesen, és ezzel a user story lezártnak tekinthető.

GitFlow

A GitFlow [31] keretrendszer egy Gites munkafolyamatot takar, amelynek lényege egy robusztus elágazási (branching) stratégia kialakítása. Ezzel a módszerrel egyszerűbbé lehet tenni a nagyobb projektek esetében a kódbázis fejlődésének követését. Az alkalmazás aktuális és kitelepítésre kész verziója a master ágon van tárolva, hozzáadni kívánt új funkcionalitások pedig külön ágon. A hagyományos módszerrel ellentétben, miszerint az ágak a masterből indulnak ki, a GitFlow két állnadó ágat tart számon: master és develop. A master ágon továbbra is a piackész verziót tárolja, míg a develop egy integrációs ágnak tekinthető a többi ág számára.

A nagyobb átláthatóság érdekében az ágakat érdemes az ág rendeltetésének megfelelő prefixszel ellátni (pl. feature, bug, improvement stb.). Ezek az ágak mindig a developból indulnak ki és oda is lesznek fésülve (*merge*) egy *merge request* (GitHub esetén *pull request*) által. Egy verzió vagy nagyobb egység elkészültével a develop a masterbe lesz mergelve.

Folyamatos integráció és kitelepítés

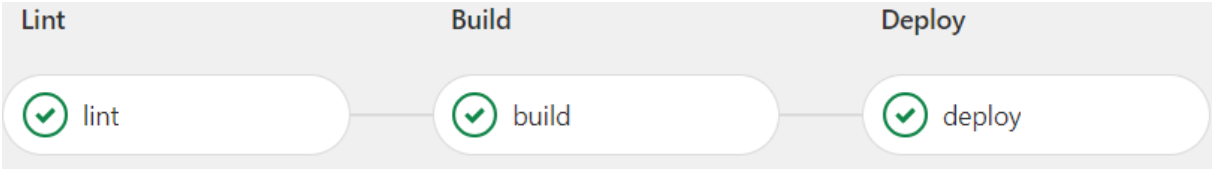
Ha új funkcionalitások kerülnek bevezetésre az alkalmazásba, garantálni kell, hogy a meglévő rendszer futása továbbra is az elvárt legyen. A manuális integrációs műveletek azonban nagyon időigényesek, főleg komplexebb alkalmazásoknál. Erre kínál megoldást a folyamatos

integráció (CI[23]) [32] amelyet egy verziókövető eszközzel használva a legtöbb integrációra vonatkozó ellenőrzés automatizálható.

A GitLab CI/CD esetén az integrációs ellenőrzések (*job*-ok) *pipeline*okba vannak csoportosítva. Egy *pipeline* tartalmazhat több *job*ot. Minden *job* egy jól meghatározott sorrendben fut le, és csak akkor kezdődik el, ha az azt megelőző sikeresen lefutott. Ezek a pipelineok egy, a repositoryn bekövetkezett esemény hatására futnak le. A *feature branch*ek esetén, úgy a frontend mint a backend repositoryjában, minden pusholáskor végrehajtódik két job. Az első a statikus kódellenőrzést végzi el, és ha ez sikeres volt, a következő job egy *build*et hajt végre, ami az alkalmazás kompilálhatóságát teszteli.

Az automatikus ellenőrzések után következik a felülvizsgálat manuális része. Egy új funkcionalitás elkészültével a GitLab felületén *merge request*et lehet indítani, vagyis egy arra irányuló kérést, hogy a feature ág legyen mergelve a developba. A fejlesztés során a merge request sablonok [33] (*template*) is segítettek az integrációban. A *template*ekben számon lehet tartani például azt, hogy az új funkcionalitásokhoz tartozó nemzetköziesítés is el lett-e végezve, illetve, hogy a grafikus felületen történt-e változás. Így a kód felülvizsgálója átfogó képet kap a snapshot teljes állapotáról. Ha minden megfelel az elvárásoknak, a branch csak akkor kerül mergelésre a developba. A develop és master branchek esetében mergekor is végrehajtódnak a pipelineok.

A CI-t követő folyamat az automatikus kitelepités (CD[24]). A master ágra történő mergekor egy új job is elérhetővé válik, ami az alkalmazás automatikus kitelepítését végzi (ld. 7. ábra). A build job következtében létrejött *Docker image* a GitLab privát konténer tárolójába (*Container Registry*) lesz feltöltve. Az alkalmazás különálló részeinek (backend, frontend, adatbázis, reverse-proxy) összekötéséért egy külön repository, az *agriary-compose* felelős. Itt kerül buildelésre a *reverse-proxy*, de az alkalmazásra vonatkozó docker-compose állomány is itt fut le.



7. ábra. A frontendhez tartozó tartozó pipeline. A Deploy stagehez tartozó job kizárólag a master ágon fut le.

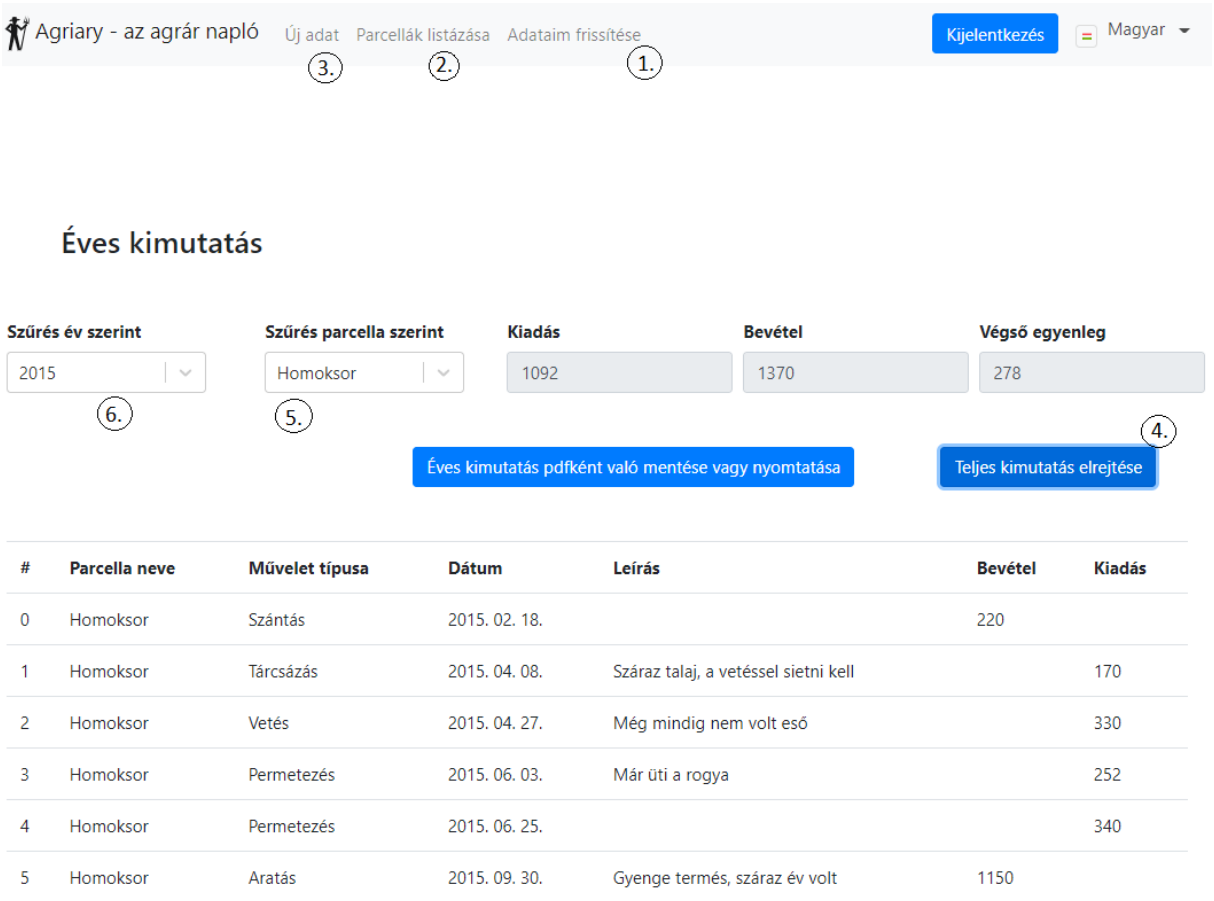
[23]CI - Continuous Integration
[24]CD - Continuous Delivery, Continuous Deployment

6. Az alkalmazás működése

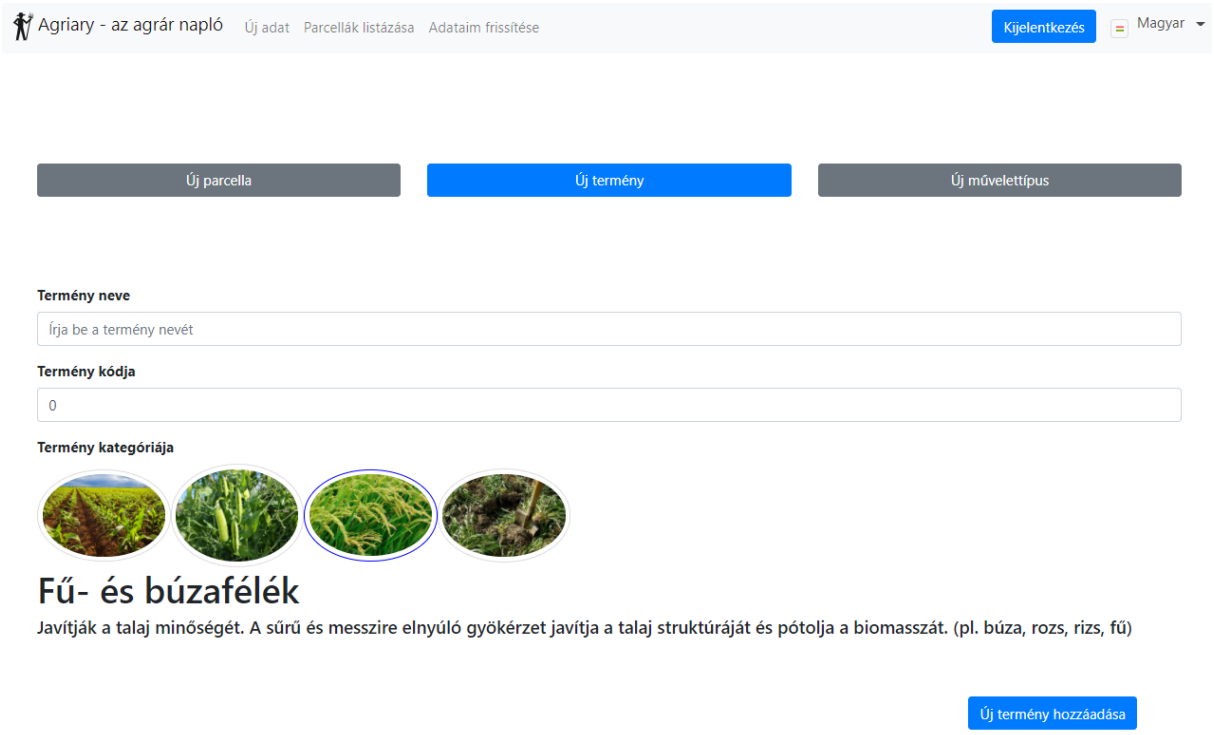
Az alkalmazásba vendég felhasználók belépése nem engedélyezett. Regisztrálni egy űrlap kitöltése által lehetséges, ezt követően be lehet jelentkezni az alkalmazásba. Az első oldal, ami fogadja a bejelentkezett felhasználót, az Éves kimutatás (8. ábra), ami a gazdaság eseményeit listázza. Ennek az oldalnak a funkcionalitásai az alkalmazás többi részének bemutatása után lesznek bővebben tárgyalva.

A 8. ábrán megfigyelhető fejléc navigációs funkciót tölt be. Az alkalmazás nevére vagy logójára kattintva a kezdőoldalra, vagyis az éves kimutatáshoz lesz navigálva a felhasználó. Jobb oldalon kapott helyet a kijelentkezést szolgáló gomb, ami a bejelentkezési oldalra navigál, mellette pedig az alkalmazás nyelvét lehet kiválasztani, jelenleg az angolt vagy a magyart. Az 1-es számmal jelölt gombra kattintva a regisztrációkor megadott személyes információkat lehet megváltoztatni, mint a név vagy a jelszó. A 2-es gomb a felhasználó parcelláit listázza ki, végül pedig a 3-as gombra kattintva lehet új adatokat bevezetni a rendszerbe.

Az Agriary három fő adattípust használ a mezőgazdasági munkák naplózásához: parcella,



8. ábra. Az Agriary alkalmazás kezdőoldala. Itt lehet a jelentéseket létrehozni.

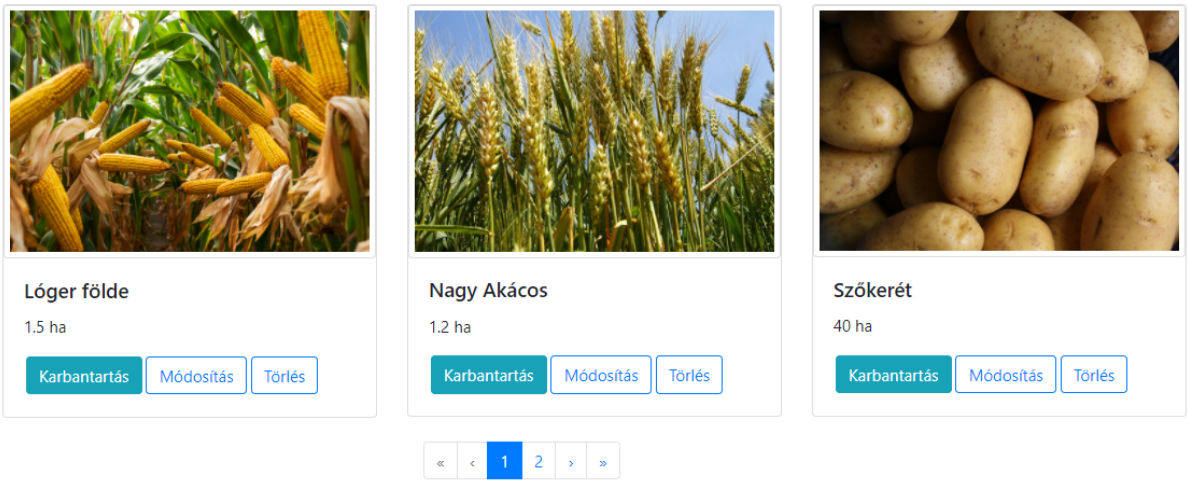


9. ábra. A terményeket létrehozó űrlap kinézete. Megfigyelhető a fejléc alatti navigációs elem, ahonnan ki lehet választani a létrehozni kívánt adattípust.

termény, művelettípus. Új művelettípust egyszerűen a munkálat nevének megadásával lehet hozzáadni, például: "Szántás négyes ekével", "Vetés hatsoros vetőgéppel", "Aratás 3 méteres asztallal".

Új parcella létrehozásához kötelező módon be kell vezetni a földterület nevét, és területét. A területet mértékegységét egy kapcsoló segítségével lehet beállítani hektárra vagy négyzetméterre. Opcionálisan, néhány előre feltöltött kép közül beállítható a parcellának egy borítókép, ami a parcellán termesztett növényre utal, ezzel is hozzájárulva a nagyobb átláthatósághoz a parcellák listázásánál. Továbbá szintén opcionálisan megadható a parcella fizikai blokkszáma. A román közigazgatás a mezőgazdasági területeket ez alapján azonosítja (Număr bloc fizic), ami nagyon hasznos lehet a jelentések készítésénél. Ha egy gazda parcellái nem csak egy város közigazgatási területén helyezkednek el, megadható a település kódja is (SIRUTA) kötőjellel elválasztva, ennek a mezőnek a használata a felhasználóra van bízva.

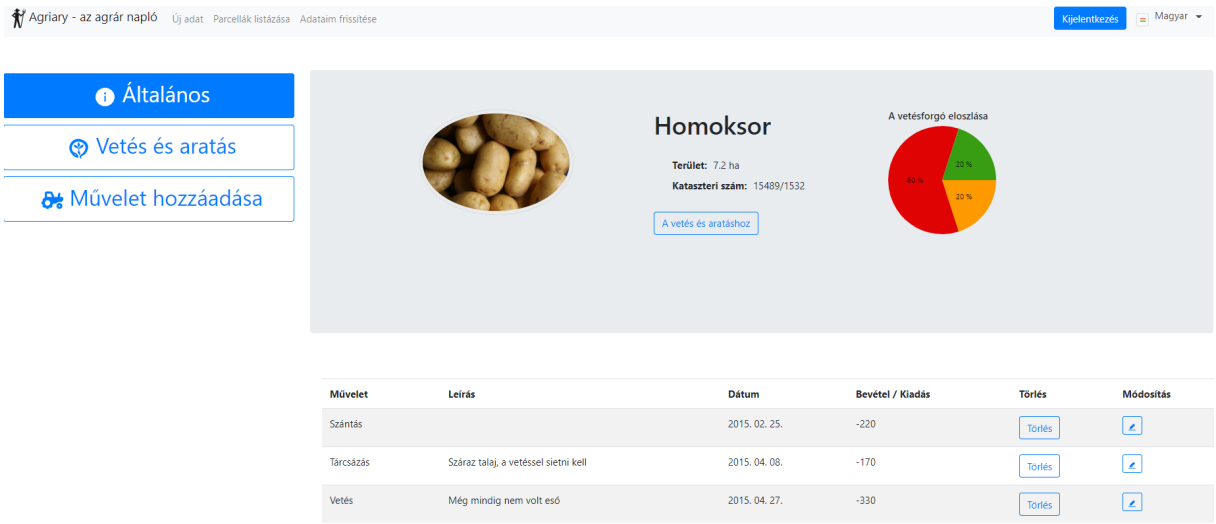
Egy új termény bevezetéséhez kötelező módon meg kell adni annak nevét és ki kell választani a kategóriáját (ld. 9. ábra). A 3.1.3. fejezetben ismertetett vetésforgó kategóriái a következők: sornövények, hüvelyesek, fű- és búzafélék, biotrágya. A felhasználó feladata eldönteni, hogy egy terményt milyen kategóriába sorol, például a napraforgót a sornövényekhez, a lucernát a hüvelyesekhez. A vetésforgó hatékony működéséhez a termények helyes



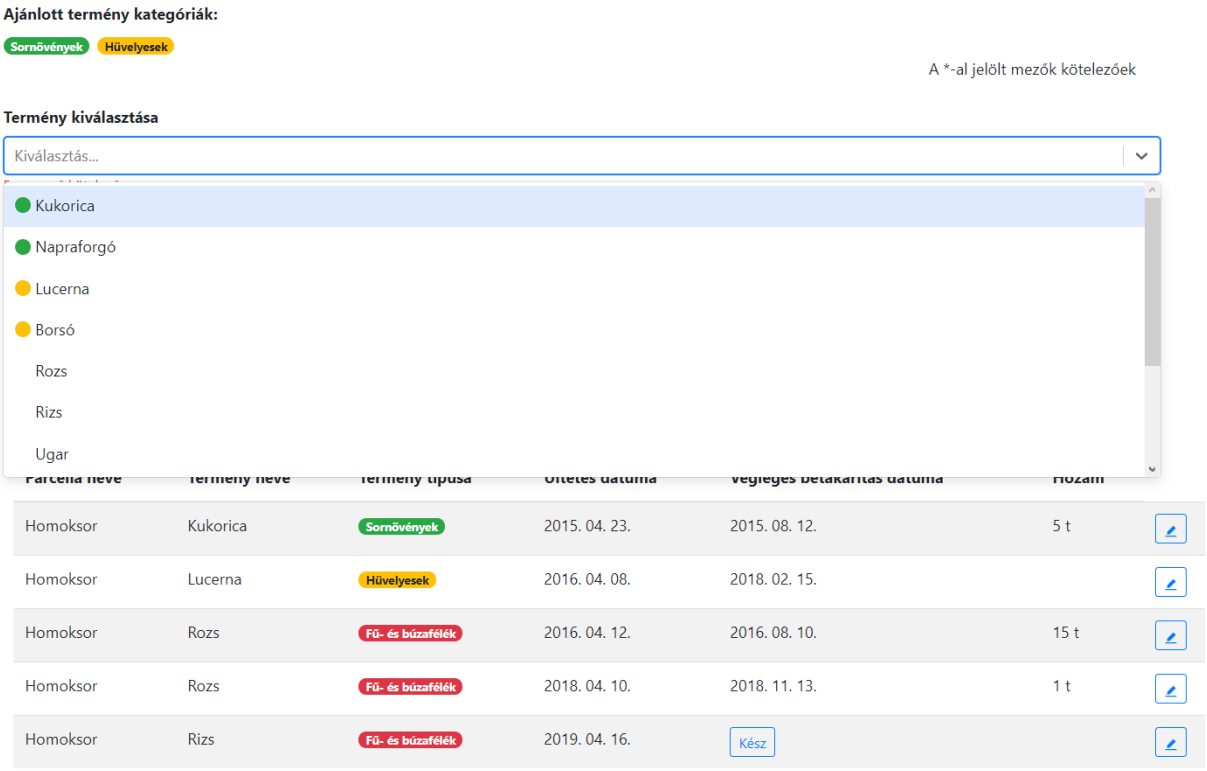
10. ábra. Közeli részlet a parcellákat listázó oldalról. Az alsó navigációs gombokkal lehet az oldalak között váltani.

típus besorolása kulcsfontosságú. Továbbá megadható még a termény kódja. Az APIA [2] rendszerében használt jelölés alapján például a kukorica kódja 108, az újburgonyáé pedig 253. Ennek a mezőnek a megadása szintén opcionális, és ugyancsak a jelentések elkészítésének megkönnyítését szolgálja.

A *Parcellák kilistázása* gombra kattintva a földterületeket jelképező kártyák megjelenítése látható (ld. 10. ábra). A kártyák parcellanév szerint vannak rendezve, és egy oldalon maximum tíz darab listázható ki belőlük (3.2.3. fejezet). A parcellákat itt lehet törölni, vagy az adatait szerkeszteni. A *Karbantartás* gombra kattintva a felhasználó a parcellák belső kezelőfelületére lesz irányítva. Itt (11. ábra) megfigyelhető az oldalsó navigációs mező ami három elemet tartalmaz. Az *Általános* fül tartalmazza a parcellára vonatkozó információkat. Itt vannak



11. ábra. A *Karbantartás* gombra kattintva a parcella általános jellemzői láthatóak.



12. ábra. Részlet a *Vetés és aratás* oldalról. Az ábrán látható esetben az ajánlott termény kategóriák a *sornövények* (zöld) és a *hüvelyesek* (sárga). A legördülő lista is jelzi a felhasználónak, hogy milyen növények ültetése ajánlott. Az alatta lévő listában az utolsó vetési ciklus még nincs lezárva, ezt a *Kész* gombra kattintva lehet megtenni.

kilistázva a parcellán végzett műveletek, de a vetésforgó hatékonyságát jelző diagram is látható már ezen az oldalon. Ez az ábra a parcellába vetett növény kategóriák arányát mutatja meg az elmúlt 10 évben. Minél kisebb a különbség az egyes szeletek között, annál optimálisabban volt alkalmazva a vetésforgó.

Az oldalmenü utolsó elemére kattintva lehet a parcellához műveleteket hozzáadni. Ehhez, az előzőleg bevezetett értékek közül kiválasztva, meg kell adni a művelet típusát. Továbbá kötelező megadni a művelet végzésének dátumát, amelyet egy naptár segítségével lehet kiválasztani. Opcionálisan megadható még egy rövid leírás, valamint a bevétel vagy kiadás *RON*-ban kifejezve.

A *Vetés és aratás* fülön lehet új vetési ciklust rendelni a parcellához (12. ábra). A felhasználónak kötelező megadnia a termény nevét és az ültetés dátumát, opcionálisan megadható a betakarítás dátuma is. A mezőgazdaságban egy hagyományos vetési ciklus ősztől őszig tart, viszont évelő növények termesztésénél ez a periódus változhat, mitöbb, egyszerre akár két típusú növény termesztése is bevett gyakorlat. Ennek okán a naplózási adatok helyes kezelése a felhasználó felelőssége, de utólagos módosításra szinte minden adattag esetében

lehetőség van.

Az alkalmazás ezen az oldalon tesz javaslatot a következő év ültetésére vonatkozóan, a javaslatot azonban nem kötelező betartani. Egy vetési ciklust le lehet zárni a *Kész* gombra kattintva. Ilyenkor meg kell adni a betakarítás dátumát, és megadható a hozam is tonnában, vagy kilogrammban kifejezve. Az alkalmazás a mértékegységeket minden esetben *kilogrammban* vagy *négyzetméterben* tárolja, de 1000 egység fölött már *tonnában* vagy *hektárban* jeleníti meg.

Így, hogy kontextust nyert az alkalmazás naplózási stratégiája, rá lehet térni a főoldal funkcionalitásaira (8. ábra). Ezen az oldalon a mezőgazdasági munkálatok átfogó jelentése található meg. Baloldalt a gazdaság pénzügyi egyenlege látható, a kiadások és bevételek összevont mérlege. A 4-es gombra kattintva megtekinthető a parcellákhoz tartozó műveletek, a részletekig lebontva. Az 5-ös és 6-os legördülő listából kiválasztható, hogy a felhasználó melyik évhez, vagy megyik parcellához tartozó jelentést szeretné elérni, viszont ez mindkét helyen szűrés nélkül is használható. A kimutatás tartalma frissül az új szűrőfeltételek függvényében és a jelentéseket *PDF*-ként is le lehet tölteni a későbbi felhasználhatóság érdekében.

Következtetések és továbbfejlesztési lehetőségek

Az Agriary fejlesztése során sikerült egy olyan webes felületet létrehozni mezőgazdászok számára, ami többnyire kielégíti azok adminisztrációs igényeit. A könnyen használható és átlátható felület segítségével a mezőgazdasági műveletek számontartása egyszerűbbé válik. A jól strukturált szerkezetnek köszönhetően ezek év vagy parcella szerinti szűrése is megoldott. A gazdák számára az éves kimutatások készítését az alkalmazás által generált átfogó jelentések könnyítik meg. A jegyzett műveleteket elemezve optimalizálhatóak és ütemezhetőek a gazdaságban végzett műveletek, ezen felül pedig a következő év vetéseire az alkalmazás is javaslatot tesz a vetésforgó függvényében. Összességében az Agriary minden gazdálkodó ember számára nagy segítség lehet úgy a nyári földművelési kampány, mint a projektek és jelentések készítésétől sűrű téli időszak levezénylésében

A fejlesztés alatt a mentorok, a csapattagok, az ötletgazda és az érdeklődő mezőgazdászok javaslatai alapján megfogalmazódott néhány továbbfejlesztési lehetőség.

Örökös cél a felhasználói felület (UI) továbbfejlesztése és a felhasználói élmény (UX) fokozása, ami többek között a következő fejlesztések által érhető el:

- grafikus komponensek és navigációs elemek újratervezése a látványosság és könnyen használhatóság érdekében
- alkalmazás testreszabhatósága: profilkép beállításának lehetősége, mértékegység megjelenítésére vonatkozó beállítások, saját fénykép feltöltése a parcellákhoz, stb.
- értesítések küldése emailben a előre beállított teendőkről vagy a szélsőséges időjárásváltozásról
- jelentések kibővítése, több fajta és eltérő részletességű jelentés generálásának lehetősége
- váratlan kiadások és bevételek bevezetésének lehetősége
- még több hasznos grafikon és statisztika

A tervek között szerepel egy fórumfelület létrehozása, ahol bármelyik felhasználó témát indíthat el, vagy hozzászólhat más felhasználók topic-jához. A gazdák megvitathatnák a növénytermesztésben szerzett tapasztalataikat, bevált praktikákat, megoszthatnák másokkal a vetőmagokról és vegyszerekről szerzett tudásukat, ezáltal egy Agriary-közösséget létrehozva.

Hivatkozások

[1] *A TypeScript hivatalos dokumentációja*. URL: https://www.typescriptlang.org/docs/ (elérés dátuma: 2021. márc. 16.)

[2] *APIA hivalatos weboldala*. URL: http://www.apia.org.ro/ (elérés dátuma: 2021. márc. 16.)

[3] *Asociația Aberdeen Angus România*. URL: https://aberdeenangus.ro (elérés dátuma: 2021. ápr. 15.)

[4] Tomislav Bacinger. *What is Bootstrap? A Short Bootstrap Tutorial on the What, Why, and How*. URL: https://www.toptal.com/front-end/what-is-bootstrap-a-short-tutorial-on-the-what-why-and-how (elérés dátuma: 2021. ápr. 15.)

[5] Flavio Copes. *JSON Web Token (JWT) explained*. URL: https://flaviocopes.com/jwt/ (elérés dátuma: 2021. ápr. 8.)

[6] *Csoportos projekt tantárgy syllabusza*. URL: https://www.cs.ubbcluj.ro/apps/fise/viewSyllabi.php?an=2020&lang=ro&specializare=IR (elérés dátuma: 2021. ápr. 15.)

[7] Shaun Luttin Daniel Roth Rick Anderson. *Introduction to ASP.NET Core*. URL: https://docs.microsoft.com/en-us/aspnet/core/introduction-to-aspnet-core?view=aspnetcore-3.0 (elérés dátuma: 2021. ápr. 15.)

[8] *Discover Mudanjiang City Mega Farm*. URL: https://pleasantstonefarm.com/discover-mudanjiang-city-mega-farm/ (elérés dátuma: 2021. ápr. 15.)

[9] *DockerHub*. URL: https://hub.docker.com (elérés dátuma: 2021. ápr. 15.)

[10] Hivalalos React dokumentáció. *Getting started*. URL: https://reactjs.org/docs/getting-started.html (elérés dátuma: 2021. márc. 22.)

[11] Hivalatos Formal.JS dokumentáció. URL: https://formatjs.io/docs/react-intl/ (elérés dátuma: 2021. ápr. 8.)

[12] Hivatalos w3schools dokumentáció. *JavaScript Promise Objectd*. URL: https://www.w3schools.com/js/js_promise.asp (elérés dátuma: 2021. ápr. 15.)

[13] Hivatalos Docker dokumentáció. *Docker Desktop overview*. URL: https://docs.docker.com/desktop/ (elérés dátuma: 2021. ápr. 15.)

[14] Hivatalos Docker dokumentáció. *Docker Engine overview*. URL: https://www.docker.com (elérés dátuma: 2021. ápr. 15.)

[15] Hivatalos Docker dokumentáció. *Overview of Docker Compose*. URL: https://docs.docker.com/compose/ (elérés dátuma: 2021. ápr. 15.)

[16] Hivatalos Formik dokumentáció. *Build forms in React, without the tears*. URL: https://formik.org (elérés dátuma: 2021. ápr. 15.)

[17] Hivatalos Git dokumentáció. *Getting Started - What is Git?* URL: https://git-scm.com/book/en/v2/Getting-Started-What-is-Git%3F (elérés dátuma: 2021. ápr. 15.)

[18] Hivatalos Microsoft dokumentáció. *Entity Framework Core*. URL: https://docs.microsoft.com/en-us/ef/core/ (elérés dátuma: 2021. ápr. 15.)

[19] Hivatalos Microsoft dokumentáció. *Language Integrated Query (LINQ)*. URL: https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/concepts/linq/ (elérés dátuma: 2021. ápr. 15.)

[20] Hivatalos Microsoft dokumentáció. *Migrations Overview*. URL: https://docs.microsoft.com/en-us/ef/core/managing-schemas/migrations/?tabs=dotnet-core-cli (elérés dátuma: 2021. ápr. 15.)

[21] Hivatalos MobX dokumentáció. URL: https://mobx.js.org/README.html (elérés dátuma: 2021. ápr. 15.)

[22] Hivatalos PstgreSQL dokumentáció. *What is PostgreSQL?* URL: https://www.postgresql.org/about/ (elérés dátuma: 2021. ápr. 15.)

[23] Hivatalos React dokumentáció. URL: https://hu.reactjs.org/docs/context.html (elérés dátuma: 2021. ápr. 15.)

[24] Hivatalos React Bootstrap dokumentáció. *React Bootstrap The most popular front-end framework Rebuilt for React*. URL: https://react-bootstrap.github.io (elérés dátuma: 2021. ápr. 15.)

[25] Hivatalos Scrum dokumentáció. *WHAT IS SCRUM?* URL: https://www.scrum.org/resources/what-is-scrum (elérés dátuma: 2021. ápr. 15.)

[26] MobX dokumentáció. URL: https://doc.ebichu.cc/mobx/refguide/observer-component.html (elérés dátuma: 2021. ápr. 15.)

[27] React Router dokumentáció. URL: https://reactrouter.com/web/guides/quick-start (elérés dátuma: 2021. ápr. 8.)

[28] Oracle dokumentció. *What Are RESTful Web Services?* URL: https://docs.oracle.com/javaee/6/tutorial/doc/gijqy.html (elérés dátuma: 2021. márc. 16.)

[29] *Drones In Agriculture Make Way For Satellite Monitoring*. URL: https://eos.com/blog/drones-in-agriculture-make-way-for-satellite-monitoring/ (elérés dátuma: 2021. ápr. 15.)

[30] *Ecoinspect SRL hivatalos weboldala*. URL: https://www.ecoinspect.ro/ (elérés dátuma: 2021. márc. 16.)

[31] *Gitflow Workflow*. URL: https://www.atlassian.com/git/tutorials/comparing-workflows/gitflow-workflow (elérés dátuma: 2021. ápr. 15.)

[32] *Hivatalos GitLab dokumentáció*. URL: https://docs.gitlab.com/ee/ci/ (elérés dátuma: 2021. ápr. 15.)

[33] *Hivatalos GitLab dokumentáció*. URL: https://docs.gitlab.com/ee/user/project/description_templates.html (elérés dátuma: 2021. ápr. 15.)

[34] *Hivatalos PostgreSQL dokumentáció*. URL: https://www.postgresql.org/docs/current/features-sql-standard.html (elérés dátuma: 2021. ápr. 15.)

[35] Fabian Lange. *Eclipse Rich Ajax Platform: Bringing Rich Clients to the Web*. Apress, 2009.

[36] Katie Lawson. *What Is a Single Page Application and Why Do People Like Them so Much?* URL: https://www.bloomreach.com/en/blog/2018/07/what-is-a-single-page-application.html (elérés dátuma: 2021. márc. 22.)

[37] *Manifesto for Agile Software Development*. URL: http://agilemanifesto.org (elérés dátuma: 2021. ápr. 15.)

[38] Alexandre Ouellette. *What is Bootstrap: A Beginner's Guide*. URL: https://careerfoundry.com/en/blog/web-development/what-is-bootstrap-a-beginners-guide/ (elérés dátuma: 2021. ápr. 15.)

[39] Max Rehkopf. *What is a kanban board?* URL: https://www.atlassian.com/agile/kanban/boards (elérés dátuma: 2021. ápr. 15.)

[40] *Vezi cele mai grave probleme din agricultura noastră*. URL: https://economie.hotnews.ro/stiri-finante_banci-24662094-desi-suntem-cincea-tara-din-suprafata-agricola-romania-are-doar-4-din-productia-mere-iar-zootehnie-nici-nu-existam-vezi-text-care-sunt-cele-mai-grave-probleme-din-agricultura-noastra.htm (elérés dátuma: 2021. ápr. 15.)

[41] *Web Framework Benchmarks*. URL: https://www.techempower.com/benchmarks/#section=data-r20&hw=ph&test=plaintext (elérés dátuma: 2021. ápr. 15.)

[42] *What Is A Reverse Proxy?* URL: https://www.cloudflare.com/learning/cdn/glossary/reverse-proxy/ (elérés dátuma: 2021. márc. 16.)

[43] *What is GitLab?* URL: https://about.gitlab.com/what-is-gitlab/ (elérés dátuma: 2021. ápr. 15.)