



ÓBUDAI EGYETEM
ÓBUDA UNIVERSITY

**NEUMANN JÁNOS
INFORMATIKAI KAR**



SZAKDOLGOZAT

**OE-NIK
2025**

Hallgató neve:
Hallgató törzskönyvi száma:

**Őz Roland Dániel
T009265/F112904/N**

Óbudai Egyetem
Neumann János Informatikai Kar
Szoftvertervezés és -fejlesztés Intézet

SZAKDOLGOZAT FELADATLAP

Hallgató neve:	Őz Roland Dániel
Törzskönyvi száma:	T009265/FI12904/N
Neptun kódja:	H8GXCF

A dolgozat címe:

**Mesterséges Intelligencia Alapú Megoldás Használata a
Növénybetegségek Azonosításában és Kezelésében
Utilization of Artificial Intelligence-Based Solutions in Identifying and
Managing Plant Diseases**

Intézményi konzulens:	Pintér Ádám
Külső konzulens:	

Beadási határidő:	2024. december 15.
-------------------	--------------------

A záróvizsga tárgyai:	Számítógép architektúrák Szoftvertervezés és -fejlesztés specializáció
-----------------------	------------------------------------------------------------------------------

A feladat

Az üvegházi növénybetegségek komoly kihívást jelentenek a mezőgazdaságban, mivel az ilyen környezetben a betegségek könnyebben terjednek és jelentős kárt okozhatnak a terményekben. Ezeknek a betegségeknek a megelőzése és kezelése azonban kulcsfontosságú a hatékony növénytermesztés és a termelési veszteségek minimalizálása szempontjából. Az üvegházi növénybetegségek elleni védekezésben és kezelésben a modern technológiák, például a mesterséges intelligencia alapú megoldások hatékony segítséget nyújthatnak a termelőknek és a mezőgazdasági szakembereknek.

A dolgozat célja mesterséges intelligencia alapú megoldás felhasználása a növénybetegségek felismerésére és kezelésére üvegházakban. A megoldás alapját az üvegházban kiépített kamerarendszer jelenti, melyen keresztül folyamatosan monitorozhatók a növények és azonosíthatók a betegségek, fertőzések. A felhasználó számára egy modern grafikai interfészen keresztül lesz lehetősége a növények megfigyelésére, továbbá a felület részletes leírásokat, hatásbecsléseket nyújt a detektált fertőzésekről, valamint javaslatokat és intézkedési lehetőségeket jelenít meg azok kezelésére és megelőzésére.

A dolgozatnak tartalmaznia kell:

- releváns szakirodalom és a hasonló rendszerek áttekintését,
- kapcsolódó mesterséges intelligencia algoritmusok vizsgálatát,
- detektálható betegségek és fertőzések jellemzőinek összegyűjtését,
- tanításhoz szükséges adatbázis létrehozása és feltöltését,
- mesterséges intelligencia algoritmus kiválasztása, tanítása, paramétereinek finomhangolása,
- automatizált betegség- és fertőzésazonosítás fejlesztését,
- modern felhasználói interfész elkészítését,
- javaslatok és intézkedési lehetőségek kidolgozását,
- az elkészült rendszer tesztelését.



Vámosy Zoltán

Dr. habil. Vámosy Zoltán
intézetigazgató

A szakdolgozat elévülésének határideje: **2026. december 15.**
(ÓE HKR 54.§ (10) bekezdés szerint)

A dolgozatot beadásra alkalmasnak tartom:

.....
külső konzulens

.....
intézményi konzulens



HALLGATÓI NYILATKOZAT

Alulírott hallgató Őz Roland Dániel (H8GXCF) kijelentem, hogy a szakdolgozat / diplomamunka saját munkám eredménye, a felhasznált szakirodalmat és eszközöket azonosíthatóan közöltem. Az elkészült szakdolgozatomban / diplomamunkámban található eredményeket az egyetem és a feladatot kiíró intézmény saját céljára térítés nélkül felhasználhatja.

Budapest, 2025.05.15

.....
hallgató aláírása



KONZULTÁCIÓS NAPLÓ

Hallgató neve:

Őz Roland Dániel

Neptun Kód:

H8GXCF

Tagozat:

nappali

Telefon:

+36-30/601-89-05

Levelezési cím (pl.: lakcím):

5000 Szolnok, Karczag László utca 1 1/13

Szakdolgozat / Diplomamunka¹ címe magyarul:

Mesterséges Intelligencia Alapú Megoldás Használata a Növénybetegségek Azonosításában és Kezelésében

Szakdolgozat / Diplomamunka² címe angolul:

Utilization of Artificial Intelligence-Based Solution in Identifying and Managing Plant Diseases

Intézményi konzulens:

Pintér Ádám

Külső konzulens:

Kérjük, hogy az adatokat nyomtatott nagy betűkkel írja!

Alk.	Dátum	Tartalom	Aláírás
1.	2024.02.28	A szakdolgozat témájának megbeszélése, célkitűzések meghatározása.	
2.	2024.03.27	A téma véglegesítése, a hasonló rendszerek áttekintése, a tartalomjegyzék kialakítása.	
3.	2024.04.24	Eszközök és szabványok áttekintése, a rendszerterv megbeszélése.	
4.	2024.05.08	Elért eredmények értékelése.	

A Konzultációs naplót összesen 4 alkalommal, az egyes konzultációk alkalmával kell láttamoztatni bármelyik konzulenssel.

A hallgató a Szakdolgozat / Szakdolgozat I. / Szakdolgozat II. / Projektlabor 1 / Projektlabor 2 / Projektlabor 3 / Záródolgozati projekt / Diplomamunka I / Diplomamunka II / Diplomamunka III / Diplomamunka IV ³ tantárgy követelményét teljesítette, beszámolóra / védésre ⁴bocsátható.

A konzulens által javasolt érdemjegy: 5

Budapest, 2024.05.10.

Intézményi konzulens

¹ Megfelelő aláhúzendő!

² Megfelelő aláhúzendő!

³ Megfelelő aláhúzendő!

⁴ Megfelelő aláhúzendő!



KONZULTÁCIÓS NAPLÓ

Hallgató neve:

Őz Roland Dániel

Neptun Kód:

H8GXCF

Tagozat:

nappali

Telefon:

+36-30/601-89-05

Levelezési cím (pl.: lakcím):

5000 Szolnok, Karczag László utca 1 1/13

Szakdolgozat / Diplomamunka¹ címe magyarul:

Mesterséges Intelligencia Alapú Megoldás Használata a Növénybetegségek Azonosításában és Kezelésében

Szakdolgozat / Diplomamunka² címe angolul:

Utilization of Artificial Intelligence-Based Solution in Identifying and Managing Plant Diseases

Intézményi konzulens:

Pintér Ádám

Külső konzulens:

Kérjük, hogy az adatokat nyomtatott nagy betűkkel írja!

Alk.	Dátum	Tartalom	Aláírás
1.	2025.02.28	A rendszerterv véglegesítése.	
2.	2025.03.28	A fejlesztés lépéseinek megtervezése.	
3.	2025.04.28	A tesztesetek kialakítása.	
4.	2025.05.12	A dolgozat véglegesítése, formai követelmények áttekintése.	

A Konzultációs naplót összesen 4 alkalommal, az egyes konzultációk alkalmával kell láttamoztatni bármelyik konzulenssel.

A hallgató a Szakdolgozat / Szakdolgozat I. / Szakdolgozat II. / Projektlabor 1 / Projektlabor 2 / Projektlabor 3 / Záródolgozati projekt / Diplomamunka I / Diplomamunka II / Diplomamunka III / Diplomamunka IV ³ tantárgy követelményét teljesítette, beszámolóra / védésre ⁴bocsátható.

A konzulens által javasolt érdemjegy: 5

Budapest, 2025.05.15.

Intézményi konzulens

¹ Megfelelő aláhúzendó!

² Megfelelő aláhúzendó!

³ Megfelelő aláhúzendó!

⁴ Megfelelő aláhúzendó!

ABSZTRAKT

A növénybetegségek jelentős gazdasági terheket rónak a mezőgazdaságra. A korai felismerés és kezelés kulcsfontosságú a termés hozam maximalizálásához. A mesterséges intelligencia (MI) ígéretes terület a növénybetegségek automatizált felismerésében. Ez a szakdolgozat egy webalkalmazás fejlesztését mutatja be Django keretrendszerben, amely mély tanuláson alapuló MI modellt alkalmaz a növénybetegségek felismerésére és a felhasználóknak a kezelésre vonatkozó javaslatok adására. Az irodalomkutatás feltérképezte a növénybetegségek felismerésében használt MI technikákat és a Django keretrendszer releváns aspektusait. A követelmény-specifikáció a rendszer funkcionális és nem funkcionális elvárásait definiálta. A rendszerterv magában foglalta az architektúrát, a modulok közötti interakciókat és az adatbázis-tervet. A fejlesztés során a webalkalmazást Django-ban implementáltam, PyTorch könyvtárat használva az MI modell betanításához. Az egységtesztelés és felhasználói tesztelés validálta a rendszer funkcionalitását és használhatóságát. Sikeresen kifejlesztettem egy webalkalmazást, amely 99.2%-os pontossággal képes 24 különböző növénybetegség felismerésére. A felhasználóbarát felület lehetővé teszi a képek feltöltését és a diagnózis megtekintését. A kutatás eredményei azt igazolják, hogy a mesterséges intelligencia hatékony eszköz a növénybetegségek felismerésére. A kifejlesztett webalkalmazás értékes segédeszköz lehet a gazdák és növénytermesztők számára a termés hozam optimalizálásában.

Kulcsszavak: Növénybetegségek, mesterséges intelligencia, mély tanulás, Django keretrendszer, webalkalmazás.

ABSTRACT

Plant diseases pose a significant economic burden on agriculture. Early detection and treatment are crucial for maximizing crop yield. Artificial intelligence (AI) presents a promising field for automated plant disease recognition. This thesis presents the development of a web application in Django framework that utilizes a deep learning-based AI model to identify plant diseases and provide treatment recommendations to users. Literature review explored AI techniques employed in plant disease recognition and relevant aspects of the Django framework. Requirement specification defined the system's functional and non-functional expectations. System design encompassed the architecture, module interactions, and database plan. Development involved implementing the web application in Django, utilizing the PyTorch library for AI model training. Unit testing and user testing validated the system's functionality and usability. A web application was successfully developed, capable of identifying 24 distinct plant diseases with 98.2% accuracy. The user-friendly interface enables uploading images and viewing diagnoses. Research findings demonstrate that AI is an effective tool for plant disease recognition. The developed web application can serve as a valuable aid for farmers and crop growers in optimizing crop yield.

Keywords: Plant diseases, artificial intelligence, deep learning, Django framework, web application.

TARTALOMJEGYZÉK

1.	BEVEZETÉS	11
2.	IRODALOMKUTATÁS.....	12
2.1.	Növénybetegségek és azok jelentősége a mezőgazdaságban	12
2.1.1.	A legjelentősebb növénybetegségek és kórokozók.....	12
2.1.2.	A legfogékonyabb növénykultúrák	13
2.1.3.	Növénybetegségek gazdasági jelentősége.....	14
2.1.4.	A növénybetegségek megelőzése és kezelése	15
2.2.	Hasonló rendszerek.....	16
2.2.1.	Plantix.....	16
2.2.2.	Plant.id.....	18
2.2.3.	Blossom.....	19
2.2.4.	Összehasonlítás	20
2.3.	ASP.NET	21
2.4.	Django keretrendszer	22
2.4.1.	Django Architektúra	22
2.4.2.	Django URL-ek.....	23
2.4.3.	Django ORM	23
2.5.	Django és ASP.NET összegzés	23
2.6.	PostgreSQL Adatbázis.....	24
2.6.1.	Integrálása a Django keretrendszerrel	24
2.7.	Adatgyűjtés és Adatkészletek	25
2.7.1.	Adatok előkészítése.....	25
2.8.	Konvolúciós neurális hálózatok.....	26
2.8.1.	A konvolúciós réteg és működése	27
2.8.2.	Aktivációs függvények.....	27
2.8.3.	Pooling réteg (leképezés)	27
2.8.4.	Mélyebb rétegek: hierarchikus jellemzőtanulás	27
2.8.5.	Teljes összeköttetésű (fully connected) réteg és osztályozás	28
2.8.6.	Tanítás és optimalizálás	28
2.8.7.	A CNN előnyei a növénybetegség-felismerésben.....	28
2.9.	PyTorch könyvtár	28
2.9.1.	A PyTorch működése	29
2.10.	TensorFlow könyvtár	29
2.10.1.	A TensorFlow működése.....	30
2.10.2.	PyTorch vs TensorFlow.....	30
2.11.	Git Verziókezelő	30
2.11.1.	A Git előnyei	30
2.11.2.	A Git működése	31
3.	KÖVETELMÉNY SPECIFIKÁCIÓ.....	32
4.	TERVEZÉS.....	34
4.1.	Rendszerterv	34
4.1.1.	Authentication alkalmazás végpontjai	35
4.1.2.	Recognition alkalmazás végpontjai.....	36
4.2.	Adatbázis-terv.....	37
4.3.	Funkciólista	38
4.3.1.	Regisztráció és bejelentkezés	39
4.3.2.	Elfelejtett jelszó.....	39
4.3.3.	Képfeltöltés és betegségfelismerés.....	40
4.3.4.	Jelszó módosítás	41

4.4.	Fejlesztés tervezett menete	42
5.	FEJLESZTÉS	44
5.1.	Kezdeti tervezés.....	44
5.2.	Adatbáziskezelés	44
5.2.1.	Migrációk kezelése.....	44
5.2.2.	Teljesítményoptimalizálás.....	44
5.2.3.	Tranzakciókezelés	45
5.3.	Modellarchitektúra-váltás és implementációs szempontok.....	45
5.3.1.	Adathalmaz.....	45
5.3.2.	Modell létrehozása	46
5.3.3.	A modell tanítása.....	48
5.3.4.	Tanítás eredménye.....	48
5.4.	Backend fejlesztés és modellintegráció	49
5.5.	Biztonsági intézkedések	50
5.5.1.	Jelszavak biztonságos kezelése	50
5.5.2.	CSRF védelem.....	50
5.5.3.	Környezeti változók használata.....	50
5.5.4.	Hibakezelés és naplózás	50
5.5.5.	Sentry integráció	51
5.5.6.	Jogosultságkezelés és autentikáció.....	51
5.5.7.	Statikus és médiatartalmak szétválasztása	51
5.6.	Frontend megvalósítása	51
5.6.1.	Don't Repeat Yourself	54
5.6.2.	Üzenetek (Django messages framework).....	55
6.	TESZTELÉS	56
6.1.	Tesztelési eszközök	56
6.2.	Github Actions.....	57
6.3.	Főbb tesztek összefoglalása.....	58
6.4.	AI Modell értékelése	61
6.4.1.	Részletes eredmények betegségtypusonként	62
6.4.2.	Konfúziós mátrix	63
6.4.3.	2D Embedding Vizualizáció	63
7.	ÉRTÉKEKELÉS	65
8.	TOVÁBBFEJLESZTÉSI LEHETŐSÉGEK.....	66
9.	ÖSSZEGZÉS.....	67
10.	SUMMARY	68
	IRODALOMJEGYZÉK.....	69
	ÁBRAJEGYZÉK	72
	TÁBLAJEGYZÉK.....	73

1. BEVEZETÉS

A mezőgazdaság kulcsfontosságú szerepet játszik az emberiség élelmiszerellátásában. Azonban a növénybetegségek jelentős gazdasági károkat okoznak, évente milliárdos veszteségeket eredményezve. A korai felismerés és a hatékony kezelés kulcsfontosságú a termés hozam maximalizálásához és a fenntartható élelmiszertermelés biztosításához.

A hagyományos növénybetegség-felismerési módszerek, mint a vizuális ellenőrzés és a laboratóriumi vizsgálatok, időigényesek, költségesek és szubjektívek lehetnek. Azonban a mesterséges intelligencia (MI) ígéretes új megközelítést kínál a növénybetegségek automatizált felismerésére. A mély tanulási technikák, mint a konvolúciós neurális hálózatok (CNN), lenyűgöző eredményeket értek el a képfeldolgozási feladatokban, beleértve a növénybetegségek azonosítását is.

Ez a szakdolgozat egy webalkalmazás fejlesztését mutatja be Django keretrendszerben, amely mély tanuláson alapuló MI modellt alkalmaz a növénybetegségek felismerésére és a felhasználóknak a kezelésre vonatkozó javaslatok adására. A cél egy hatékony, felhasználóbarát eszköz létrehozása, amely segíti a gazdákat és a növénytermesztőket a termésveszteségek minimalizálásában és a termés hozam optimalizálásában.

A kutatás motivációja a mezőgazdaságban rejlő kihívások megoldása és a fenntartható élelmiszertermelés elősegítése. A növénybetegségek pusztító hatásai elleni küzdelem kulcsfontosságú a globális élelmiszerbiztonság fenntartásához. A mesterséges intelligencia bevetése ebben a küzdelemben ígéretes utat nyithat a hatékonyabb és fenntarthatóbb mezőgazdaság felé.

A következő fejezetekben részletesen bemutatom a kutatás háttérét, a megvalósítás folyamatát, az elért eredményeket, valamint a jövőbeli lehetőségeket. A dolgozat célja, hogy gyakorlati megoldást kínáljon a mezőgazdaság egyik legjelentősebb kihívására, és hozzájáruljon a fenntartható élelmiszertermelés eléréséhez.

2. IRODALOMKUTATÁS

Az alábbi fejezet részletesen tárgyalja a növénybetegségek felismeréséhez és kezeléséhez kapcsolódó témaköröket, valamint bemutatja a rendszerfejlesztéshez használt technológiákat és a jelenleg piacon lévő hasonló megoldásokat. A szakirodalom áttekintése során különös figyelmet fordítottam a növénybetegségek gazdasági hatásaira, a legfogékonyabb növénykultúrákra, valamint a mesterséges intelligencia alapú felismerő rendszerek működési alapelveire. A különböző neurális hálózati megközelítések, valamint a Django és PostgreSQL alapú webes megoldások elemzése lehetővé tette a fejlesztendő rendszer optimális architektúrájának megtervezését. Az adatgyűjtési és adatelőkészítési technikák ismertetése mellett áttekintem a hasonló rendszerek által kínált funkcionalitást és piaci pozíciót is, amely értékes információkkal szolgált a saját rendszerem fejlesztéséhez.

2.1. Növénybetegségek és azok jelentősége a mezőgazdaságban

A növénybetegségek évente mintegy 20-40%-os termésvesztést okoznak világszerte, ami jelentős gazdasági kárt jelent a mezőgazdasági termelésben [1]. A FAO (Food and Agriculture Organization) becslései szerint a fejlődő országokban az élelmiszertermelés akár 50%-a is elveszhet a betakarítás előtt vagy után növényi kórokozók miatt [2]. Ez a fejezet a legjelentősebb növénybetegségeket, azok hatását és a lehetséges megelőzési módszereket mutatja be, különös tekintettel az automatizált korai felismerésre és diagnosztikára alkalmas rendszerekre.

A növénybetegségek nemcsak a terméshozamot csökkentik, hanem rontják a termés minőségét, csökkentik tárolhatóságát, és toxikus vegyületek termelődéséhez vezethetnek, amelyek emberi fogyasztásra alkalmatlanná teszik a növényi terméket [3]. Emellett globális élelmiszerbiztonsági szempontból is jelentős kockázatot jelentenek, különösen a klímaváltozás fényében, amely új kórokozók megjelenéséhez és a meglévők terjedési területének bővüléséhez vezet [4].

2.1.1. A legjelentősebb növénybetegségek és kórokozók

A növénybetegségeket általában négy fő kategóriába sorolhatjuk a kórokozó típusa alapján:

Gombás betegségek: A gombás fertőzések a legelterjedtebb növénybetegségek, a növényi kórokozók közel 85%-át teszik ki [5]. Ezek közül a legjelentősebbek:

- **Lisztharmat (powdery mildew):** Számos növényfajt érint, különösen a gabonákat, szőlőt és zöldségféléket. Az Erysiphaceae családba tartozó gombák okozzák, amelyek fehér, lisztes bevonatot képeznek a levelek felszínén, gátolva a fotoszintézist (Glawe, 2018). Becslések szerint csak búzában és árpyában évente 10-15%-os termésvesztést okoz [6].
- **Rozsdabetegségek (rust diseases):** A Puccinia nemzetségbe tartozó gombák által okozott betegségek, amelyek különösen a gabonafélékre veszélyesek. A búzarozsda (Puccinia graminis f. sp. tritici) Ug99 nevű törzse globális fenyegetést jelent, és akár 100%-os termésvesztést is okozhat fogékony fajtákban [7].
- **Fuzáriumos fertőzések:** A Fusarium nemzetségbe tartozó gombák által okozott betegségek, amelyek különösen a gabonafélékben és kukoricában jelentősek. Nemcsak

termésveszteséget okoznak, hanem mikotoxinokat is termelnek, amelyek súlyos egészségügyi kockázatot jelentenek [8].

- **Szürkepenész (*Botrytis cinerea*):** Rendkívül széles gazdanövény körrel rendelkező gomba, amely több mint 500 növényfajt fertőzhet meg. Különösen jelentős károkat okoz a szőlő-, eper- és paradicsom-termesztésben [9].

Bakteriális betegségek: A bakteriális betegségek különösen nehezen kezelhetők, mivel a jelenleg elérhető baktericidek választéka korlátozott, és gyakran fitotoxikus hatásúak:

- **Tűzelhalás (fire blight):** Az *Erwinia amylovora* baktérium által okozott betegség, amely az almafélék családjába tartozó növényeket, különösen az almát és körtét veszélyezteti. Európában évente körülbelül 100 millió euró kárt okoz [10].
- **Xanthomonas fertőzések:** Többek között a rizs bakteriális levélfoltosságát (*Xanthomonas oryzae* pv. *oryzae*) és a citrusfélék rákosodását (*Xanthomonas citri*) okozzák. A rizs bakteriális levélfoltossága Ázsiában akár 50%-os termésveszteséget is okozhat [1].
- **Burgonya barnarothadása:** A *Ralstonia solanacearum* által okozott betegség, amely a burgonya egyik legpusztítóbb betegsége, és globálisan terjed. Egyes területeken akár 80%-os termésveszteséget is okozhat [11].

Vírusos betegségek: A vírusok által okozott növénybetegségek különösen problémásak, mivel direkt kezelésük jelenleg nem lehetséges, csak a megelőzésre és a vektorok elleni védekezésre lehet összpontosítani:

- **Burgonya Y-vírus (PVY):** A burgonya egyik legjelentősebb vírusos betegsége, amely 40-70%-os termésveszteséget is okozhat. A vírus levéltetvekkel terjed [12].
- **Paradicsom mozaikvírus (ToMV):** Széles gazdanövény körrel rendelkező vírus, amely jelentős károkat okoz a paradicsom- és paprikatermesztésben. Mechanikai úton, érintkezéssel terjed, és akár 20-70%-os termésveszteséget okozhat [13].
- **Kukorica csíkos mozaikvírus (MDMV):** A kukorica jelentős vírusos betegsége, amely levéltetvekkel terjed. Súlyos fertőzés esetén 50%-ot is meghaladó termésveszteséget okozhat [14].

Fitoplazmás és egyéb betegségek:

- **Szőlő aranyszínű sárgasága:** A Flavescence dorée fitoplazma által okozott betegség, amely az európai szőlőtermesztés egyik legveszélyesebb kórokozója. Kabócákkal terjed, és teljes terméselvezítést okozhat [15].
- **Almaproliferáció:** A 'Candidatus Phytoplasma mali' által okozott betegség, amely az almafákat károsítja, csökkentve a termés mennyiségét és minőségét akár 80%-kal is [16].

2.1.2. A legfogékonyabb növénykultúrák

A növénybetegségekkel szembeni fogékonyság számos tényezőtől függ, beleértve a növényfajt, fajtát, környezeti körülményeket és a termesztési gyakorlatokat. Az alábbi kultúrnövények különösen fogékonyak különböző betegségekre:

Burgonya: A burgonya számos betegsége fogékony, többek között a burgonyavészre (*Phytophthora infestans*), amely az írországi éhínség okozója volt a 19. században. A

burgonyavész elleni védekezés ma is jelentős kihívást jelent, és az európai termelők évente mintegy 1 milliárd eurót költenek fungicidekre a betegség elleni védekezésre [17]. Emellett a burgonya fogékony különböző vírusokra, bakteriális és gombás betegségekre is, mint például a burgonya Y-vírus és a burgonya barnarothadása.

Paradicsom: A paradicsom különösen érzékeny a különböző gombás, bakteriális és vírusos betegségekre. A paradicsomvész (*Phytophthora infestans*), a fuzáriumos hervadás (*Fusarium oxysporum* f. sp. *lycopersici*), a paradicsom mozaikvírus (ToMV) és a baktériumos pöttyösség (*Pseudomonas syringae* pv. *tomato*) jelentős termésvesztést okoznak világszerte. Jaarsveld és munkatársai [18] kimutatták, hogy a paradicsom betegségei akár 30-50%-os termésvesztést is okozhatnak megfelelő növényvédelem nélkül.

Gabonafélék: A búza, árpa és egyéb gabonafélék számos gombás betegsége fogékonyak, beleértve a különböző rozsda- és üszögbetegségeket, a fuzáriumos kalászfertőzést és a lisztharmatot. A klímaváltozás következtében a *Puccinia graminis* f. sp. *tritici* Ug99 törzs terjedése különösen aggasztó, mivel a korábban rezisztens fajták is fogékonnyá váltak [7]. A fuzáriumos kalászfertőzés nemcsak termésvesztést okoz, hanem mikotoxin-szennyezést is, ami az élelmiszerbiztonsági szempontokat figyelembe véve még súlyosabb probléma.

Szőlő: A szőlőt számos betegség veszélyezteti, többek között a lisztharmat (*Erysiphe necator*), a peronoszpóra (*Plasmopara viticola*) és a szürkepenész (*Botrytis cinerea*). Ezek a betegségek nemcsak a termés hozamot csökkentik, hanem a bor minőségét is jelentősen befolyásolják. A szőlőtermesztők Európában évente mintegy 500 millió eurót költenek fungicidekre [19].

2.1.3. Növénybetegségek gazdasági jelentősége

A növénybetegségek globális gazdasági hatását nehéz pontosan számszerűsíteni, de becslések szerint az éves veszteség meghaladja a 220 milliárd USD-t [1]. Ez az összeg tartalmazza mind a közvetlen termésvesztést, mind a védekezés költségeit.

Termésvesztés és minőségromlás: A növénybetegségek által okozott termésvesztés régióként és kultúráként jelentősen változik, de általánosságban elmondható, hogy a fejlődő országokban nagyobb problémát jelentenek. Az FAO adatai szerint a fejlődő országokban a betakarítás előtti veszteségek akár 30-40%-ot is elérhetnek, míg a fejlett országokban ez az arány általában 10-20% közé tehető [2].

A mennyiségi veszteségek mellett a minőségromlás is jelentős gazdasági kárt okoz. A fuzáriumos fertőzések által termelt mikotoxinok például nemcsak az emberi és állati egészségre veszélyesek, hanem jelentősen csökkentik a gabona értékét is. A European Food Safety Authority (EFSA) jelentése szerint az EU tagállamaiban évente több mint 3 millió tonna gabona kerül kizárásra az emberi fogyasztásból mikotoxin-szennyezés miatt [20].

Növényvédelmi költségek: A növényvédő szerek használata jelentős költséget jelent a termelők számára. Csak Európában az éves növényvédőszer-felhasználás értéke meghaladja a 10 milliárd eurót, amelynek jelentős részét a fungicidek teszik ki [21]. Az Egyesült Államokban mintegy 14 milliárd dollárt költenek évente növényvédő szerekre [22].

A kémiai növényvédő szerek használata azonban nemcsak költséges, hanem környezeti és egészségügyi kockázatokat is hordoz. Emellett egyre növekvő probléma a kórokozók rezisztenciájának kialakulása a gyakran használt hatóanyagokkal szemben.

2.1.4. A növénybetegségek megelőzése és kezelése

A növénybetegségek elleni védekezés komplex megközelítést igényel, amely magában foglalja a megelőzést, a korai felismerést és a megfelelő kezelést.

Integrált növényvédelem (IPM): Az integrált növényvédelem egy holisztikus megközelítés, amely különböző stratégiák kombinációját alkalmazza a növénybetegségek és kártevők elleni védekezésben, minimalizálva a környezeti hatásokat és a kémiai növényvédő szerek használatát. Az EU 2009/128/EK irányelve kötelezővé tette az IPM alapelveinek alkalmazását az EU tagállamaiban [23].

Az IPM fő komponensei:

- Megelőzés (ellenálló fajták, vetésforgó, megfelelő agrotechnika)
- Rendszeres megfigyelés és korai felismerés
- Közbelépési küszöbértékek alkalmazása
- Nemkémiai módszerek előnyben részesítése
- Célzott növényvédőszer-használat
- Alkalmazott növényvédőszer-használat csökkentése
- Rezisztencia-kezelési stratégiák
- Az alkalmazott módszerek hatékonyságának értékelése

Rezisztens fajták nemesítése és használata: A növénybetegségekkel szemben ellenálló fajták nemesítése és használata az egyik leghatékonyabb és környezetbarát növényvédelmi stratégia. A modern nemesítési technikák, beleértve a molekuláris markereket és a génszerkesztést, felgyorsították az ellenálló fajták fejlesztését [24].

Példaként említhető a burgonyavésszel szemben ellenálló burgonya fajták fejlesztése, amely jelentősen csökkentheti a fungicidhasználatot. A CRISPR-Cas9 technológiával előállított, lisztharmattal szemben ellenálló búzafajták ígéretes eredményeket mutatnak a termésvesztés csökkentésében [6].

Korai diagnózis és monitoring rendszerek: A növénybetegségek korai felismerése kulcsfontosságú a hatékony védekezéshez. A hagyományos vizuális megfigyelésen alapuló módszerek mellett egyre nagyobb szerepet kapnak a modern technológiák:

- **Érzékelők és távérzékelés:** A hiperspektrális és multispektrális képalkotás lehetővé teszi a növényi stressz és betegségek korai, akár tünetmentes fázisban történő felismerését [25].
- **Mesterséges intelligencia és gépi tanulás:** A képfelismerési algoritmusok fejlődése lehetővé teszi a növénybetegségek automatizált diagnózisát. A mélytanuláson alapuló rendszerek akár 98%-os pontossággal képesek azonosítani különböző növényi betegségeket [26].
- **Internet of Things (IoT) alkalmazások:** A szenzorhálózatok real-time adatokat szolgáltatnak a környezeti körülményekről és a növények állapotáról, lehetővé téve a betegségek előrejelzését és a célorientált védekezést [27].

2.2. Hasonló rendszerek

Ebben a fejezetben áttekintem a növénybetegségek felismerésére szolgáló mesterséges intelligencia (MI) alapú rendszereket, a Plantix, Plant.id és Blossom Plant-re összpontosítva. Célom, hogy összehasonlítsam ezeket a megoldásokat, és feltárjam azokat a kulcsfontosságú jellemzőket, amelyek a saját fejlesztésű rendszerem szempontjából relevánsak lehetnek.

A hasonló rendszerek elemzése során a következő szempontokra fogok összpontosítani:

- **Funkcionalitás:** Milyen funkciókat kínál a rendszer? Képes-e felismerni a növénybetegségek széles skáláját? Kínál-e további funkciókat, mint például a kártevők felismerése, a tápanyaghiány diagnosztizálása, vagy a kezelési javaslatok?
- **Pontosság:** Milyen pontos a rendszer a növénybetegségek diagnosztizálásában?
- **Felhasználói felület:** Milyen felhasználóbarát a rendszer felülete? Könnyű-e használni az alkalmazást?
- **Ár:** Ingyenes a rendszer használata, vagy díj ellenében kell előfizetni? Milyen a díjazási struktúra?
- **Technológia:** Milyen technológiát használ a felismeréshez?

A hasonló rendszerek elemzése segít megérteni a piacon elérhető lehetőségeket, és azonosítani azokat a területeket, ahol a saját fejlesztésre szánt rendszerem kiemelkedhet. Ezenkívül segít meghatározni a funkcionalitás, a pontosság, a felhasználói felület és az ár optimális kombinációját a célközönség számára.

2.2.1. Plantix

A Plantix [28] egy népszerű mobilalkalmazás, amely mesterséges intelligenciát (MI) alkalmaz a növénybetegségek, kártevők és tápanyaghiányok felismerésére. A felhasználók fényképet készíthetnek a beteg növényről, és az alkalmazás diagnosztizálja a problémát, és javaslatokat tesz a kezelésre. A Plantix a mezőgazdaságban és a kertészetben egyaránt hasznos eszköz lehet, segítve a gazdáknak és kertészeknek a terméshozam maximalizálásában és a növények egészségének megőrzésében.

Főbb jellemzők:

- **Növénybetegségek felismerése:** A Plantix mély tanulási modellt használ a növénybetegségek széles skálájának felismerésére, beleértve a foltosodást, a rothadást, a hervadást és a gombás fertőzéseket.
- **Kártevők felismerése:** A Plantix nem csak a növénybetegségeket, de a kártevőket is képes felismerni, mint például a levéltetűeket, a pókokat és a hernyókat.
- **Tápanyaghiány diagnosztizálása:** A Plantix segíthet a tápanyaghiány diagnosztizálásában is, mint például a nitrogén-, foszfor- és káliumhiány.
- **Kezelési javaslatok:** A Plantix a diagnózis alapján kezelési javaslatokat tesz.
- **Növényi profilok:** A Plantix lehetővé teszi a növényi profilok létrehozását, amelyben rögzítheti a növény típusát, a telepítési dátumot, a helyet és a kezelési előzményeket.
- **Offline mód:** A Plantix offline módban is használható, így internetkapcsolat nélkül is hozzáférhet a funkcióihoz.
- **Technológia:** Konvolúciós neurális hálózatokat (CNN-eket) használ a képfeldolgozáshoz.

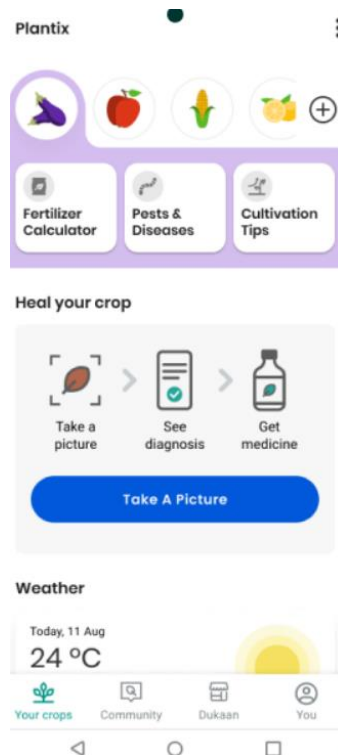
Előnyök:

- **Könnyen használható:** A Plantix intuitív felülettel rendelkezik, amelyet bárki könnyen használhat, szakmai ismeretek nélkül is.
- **Pontos:** A Plantix mély tanulási modellje megbízhatóan pontos diagnózist biztosít.
- **Sokoldalú:** A Plantix a növénybetegségek, kártevők és tápanyaghiányok széles skálájának felismerésére képes.
- **Hasznos:** A Plantix értékes információkat és javaslatokat nyújt a növények kezeléséhez.
- **Ingyenes:** A Plantix alapvető funkciói ingyenesen elérhetők.

Hátrányok:

- **Prémium funkciók:** A Plantix prémium verziója további funkciókat kínál, mint például a növényi profilok mentése és a kezelési előzmények nyomon követése, de ez díj ellenében érhető el.
- **Internetkapcsolat:** A Plantix teljes funkcionalitásának eléréséhez internetkapcsolat szükséges.

A Plantix egy hatékony és felhasználóbarát eszköz a növénybetegségek, kártevők és tápanyaghiányok felismerésére. Az alkalmazás értékes információkat és javaslatokat nyújt a növények kezeléséhez, segítve a gazdáknak és kertészeknek a termés hozam maximalizálásában és a növények egészségének megőrzésében. A Plantix alapvető funkciói ingyenesen elérhetők, de a prémium funkciókért díjat kell fizetni. Az alábbi **2.1. ábra** szemlélteti a Plantix felületét.



2.1. ábra Plantix andorid felülete [28]

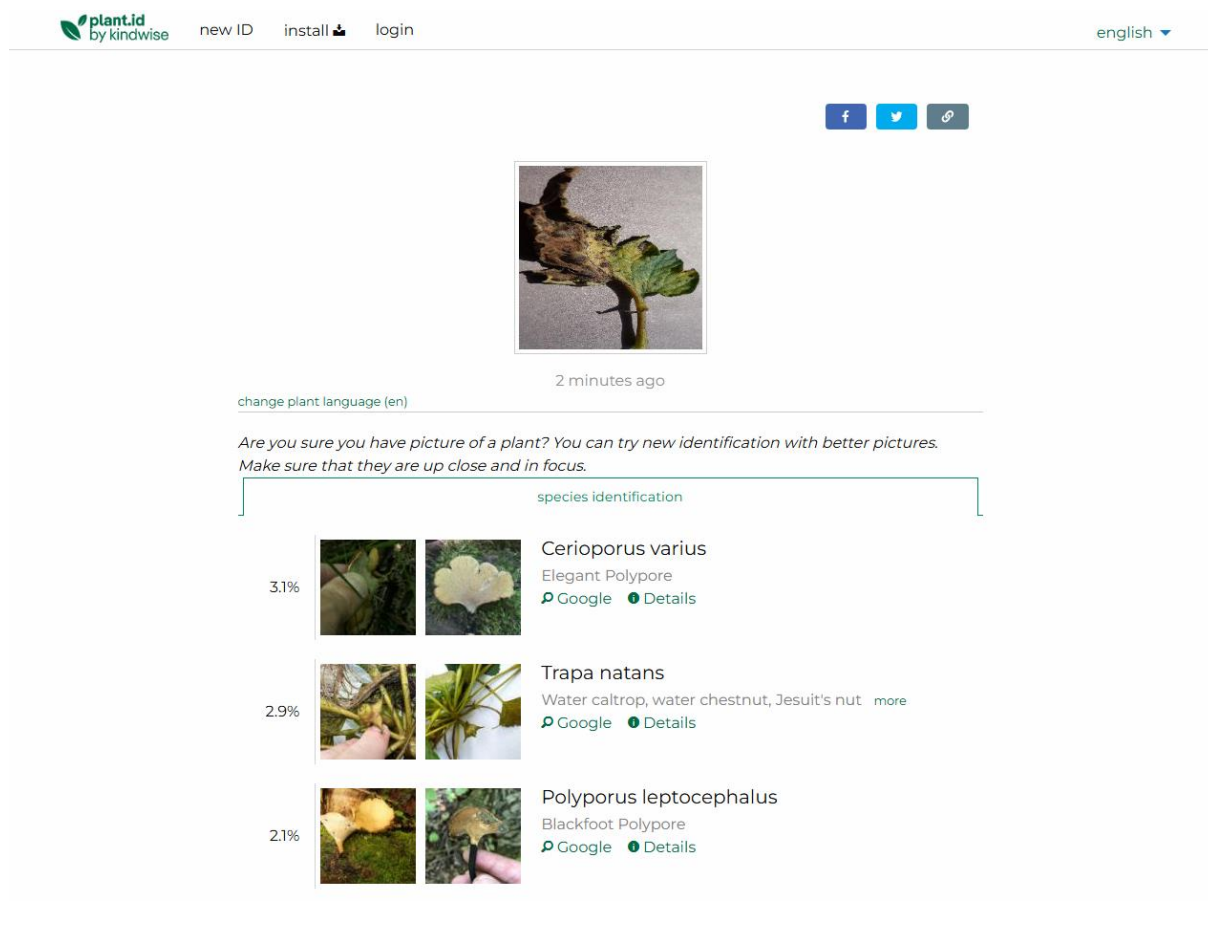
2.2.2. Plant.id

A Plant.id [29] egy online szolgáltatás, amely mesterséges intelligenciát használ növények, virágok és fák azonosítására, valamint növénybetegségek diagnosztizálására.

Főbb jellemzők:

- **Ingyenes növényfelismerés:** A Plant.id weboldala lehetővé teszi akár 10 növény ingyenes azonosítását havonta. Csak töltsd fel a növény képeit, és az oldal azonosítja azt.
- **Széleskörű adatbázis:** A Plant.id több mint 35.000 növényfaj azonosítására képes, beleértve szobanövényeket, kerti növényeket, fákat, gyomokat, gombákat és zuzmókat a világ minden tájáról. Az azonosítás mellett a tudományos (latin) név mellett a növény közönséges nevét, egy rövid leírást és a besorolását is megkapod.
- **Növénybetegségek diagnosztizálása:** Egészségügyi értékelés funkciója segít a különbségtételben a kártevők, a gombás betegségek és az öntözés okozta problémák között. A modelljük 90 különböző betegséget képes diagnosztizálni.
- **Mesterséges intelligencia:** A Plant.id a gépi tanulás legmodernebb módszereit alkalmazva fejleszti mély tanuló neuronhálózatait a lehető legjobb eredmények elérése érdekében. A becslésük szerint 85% -os pontossággal képesek azonosítani a növényeket.
- **API fejlesztőknek:** A Plant.id API-t kínál a mezőgazdaságban, a környezetvédelemben vagy az okoskertek területén dolgozó vállalkozások számára. Az API segítségével integrálhatják a Plant.id növényfelismerési technológiáját saját alkalmazásaikba és weboldalaikba.
- **Technológia:** Konvolúciós neurális hálózatokat (CNN-eket) használ a képfeldolgozáshoz.

A Plant.id egy hasznos online eszköz a növények és a növénybetegségek ingyenes azonosítására. Az alapvető növényfelismerési funkció ingyenes, a betegségdiagnosztizáláshoz viszont prémium tagság szükséges. A Plant.id széleskörű növényadatbázissal rendelkezik, és mesterséges intelligenciát használ a pontos azonosítás érdekében. Fejlesztők számára API-t is kínálnak. Az alábbi **2.2. ábra** mutatja a feltöltés utáni felületet.



2.2. ábra Plant.id felülete

2.2.3. Blossom

A Blossom [30] egy mobilalkalmazás, amely mesterséges intelligencia (MI) segítségével azonosítja a növényeket a felhasználók által készített fényképek alapján. A Bending Spoons S.p.A. fejlesztette ki, és 2020-ban jelent meg a piacon.

Főbb jellemzői:

- **Növényazonosítás:** A Blossom adatbázisa több mint 30.000 növényfajt tartalmaz, beleértve a szobanövényeket, kerti virágokat, fákat, cserjéket, zuzmókat és mohát. A növény azonosítása fénykép alapján történik, a Multisnap módban akár három fotó is feltölthető a pontosság növelése érdekében.
- **Növénygondozási információk:** A növény azonosítása után az alkalmazás kiterjedt **gondozási információkat** nyújt, beleértve az öntözés gyakoriságát, a fény- és hőmérséklet-igényeket, a talajtípust, a szaporítási technikákat, a gyakori betegségeket és kártevőket, valamint a növény általános gondozási tippeket.
- **További funkciók:** A Blossom növénygondozási emlékeztetőket, betegségdiagnosztikát, vízszükséglet-kalkulátort, kertészeti ütemtervet és időjárás-értesítéseket is kínál.
- **Felhasználói felület:** Az alkalmazás felülete egyszerűen használható és vizuálisan vonzó.

- **Technológia:** Nem közölték pontosan, milyen gépi tanulási modellt használnak, de valószínűsíthetően szintén CNN-alapú képosztályozó modellekre épül, esetleg harmadik féltől licencelt technológiára.

Hátrányok:

- Lehet, hogy nem minden növényt ismer fel pontosan
- Az ingyenes verzió korlátozott funkciókkal rendelkezik
- Előfizetés szükséges a prémium funkciókhoz

A Blossom egy hatékony és felhasználóbarát eszköz a növények azonosítására és gondozására. Az alkalmazás széles körű funkcionalitást, magas pontosságot és kiváló felhasználói felületet kínál. A Blossom ideális választás a kezdő és haladó növénykedvelők számára egyaránt, akik szeretnék bővíteni tudásukat a növényvilágról, és hatékonyan gondozni a növényeiket. Az alábbi **2.3. ábra** szemlélteti az azonosított növénybetegéget a telefonos alkalmazásban.



2.3. ábra Blossom IOS alkalmazás detektált betegség felülete

2.2.4. Összehasonlítás

A **2.1. táblázatban** foglalom össze az elemzett rendszereket.

Szempont	Plantix	Plant.id	Blossom
Platform	Mobilalkalmazás (Android)	Weboldal	Mobilalkalmazás (iOS, Android)
Offline funkciók	Korlátozott (néhány betegség azonosítása)	Nincsenek	Korlátozott (néhány betegség azonosítása)

Képkészítés	Fénykép feltöltése a telefon tárhelyéről	Fénykép feltöltése a böngészőből	Fénykép feltöltése a telefon tárhelyéről, kép készítése az alkalmazásból
Funkcionalitás	Növénybetegség-felismerés, kártevő-felismerés, tápanyaghiány-diagnosztizálás, részletes kezelési javaslatok, növényazonosítás, növénygondozási tippek, kártevő- és betegségmegelőzési útmutatók, közösségi fórum	Növénybetegség-felismerés, részletes kezelési javaslatok, növényazonosítás, növénygondozási tippek	Növényazonosítás, részletes növénygondozási információk, növénygondozási emlékeztetők, betegségdiagnosztika, vízszükséglet-kalkulátor, kertészeti ütemterv, időjárás-értesítések
Pontosság	90%	85%	Nem publikált adatok
Technológia	CNN-alapú képfeldolgozás	CNN-alapú képfeldolgozás	Nem publikált adatok
Ár	Ingyenes alapverzió, előfizetés szükséges a prémium funkciókhoz	Ingyenes alapverzió, előfizetés szükséges a prémium funkciókhoz	Ingyenes alapverzió, előfizetés szükséges a prémium funkciókhoz
Támogatás és karbantartás	Aktív felhasználói támogatás, rendszeres frissítések	Aktív felhasználói támogatás, rendszeres frissítések	Aktív felhasználói támogatás, rendszeres frissítések

2.1. táblázat Hasonló rendszerek összehasonlítása

A három bemutatott rendszer mindegyike értékes eszköz lehet a növénybetegségek felismerésében és a növények gondozásában. A Plantix a legösszetettebb funkcionalitást és a legmagasabb pontosságot kínálja, de előfizetés szükséges a teljes körű szolgáltatáshoz. A Plant.id hasonló funkcionalitással rendelkezik, de weboldalas platformja miatt kevésbé felhasználóbarát lehet. A Blossom a legegyszerűbb, de pont ezért pontatlanabb is lehet.

2.3. ASP.NET

Az ASP.NET [31] a Microsoft által fejlesztett, nyílt forráskódú webalkalmazás-fejlesztési keretrendszer, amely lehetővé teszi dinamikus weboldalak, webalkalmazások és webszolgáltatások létrehozását. Az ASP.NET a .NET platform része, és kifejezetten a szerveroldali fejlesztések támogatására szolgál, C# vagy VB.NET nyelvek használatával. Az ASP.NET egyik legfontosabb jellemzője a teljesítményorientált működés, amelyet a .NET futtatókörnyezet optimalizáltsága és a Just-In-Time (JIT) fordítás biztosít. Ennek köszönhetően a vele készült webalkalmazások gyorsak és jól reagálnak a felhasználói műveletekre. A keretrendszer erős modularitást biztosít, mivel lehetővé teszi újrafelhasználható komponensek – például köztes rétegek, vezérlők vagy megjelenítő komponensek – létrehozását, ami elősegíti a tisztább kódbázist és a hatékonyabb fejlesztést.

Biztonság szempontjából az ASP.NET korszerű autentikációs és autorizációs megoldásokat kínál, beleértve az Identity rendszert, az OAuth protokollt vagy a JSON Web Token (JWT)

alapú jogosultságkezelést. A keretrendszer a Model-View-Controller (MVC) architektúrát alkalmazza, amely világosan elkülöníti az adatkezelést, a megjelenítést és az üzleti logikát, így átláthatóbbá és könnyebben karbantarthatóvá válik az alkalmazás szerkezete.

A Razor szintaxis különleges előnyt jelent a fejlesztők számára, mivel lehetővé teszi a C# alapú szerveroldali logika és a HTML keverését egy egységes, letisztult formában. Továbbá az ASP.NET beépített támogatást nyújt a dependency injection (DI) használatához, amely segíti a moduláris fejlesztést és nagymértékben elősegíti a kód tesztelhetőségét.

2.4. Django keretrendszer

A Django [32] egy nyílt forráskódú Python webkeretrendszer, amelyet gyors és hatékony webes alkalmazások fejlesztésére terveztek. Széleskörű funkcionális kínál, beleértve a URL-vezérlést, sablonrendszert, adatbázis-kezelést, hitelesítést és engedélyezést, ezáltal megkönnyítve a komplex webes alkalmazások létrehozását.

Főbb jellemzői:

- **Gyors fejlesztés:** A Django számos beépített funkciót és eszközt kínál, amelyek felgyorsítják a webes alkalmazások fejlesztését.
- **Széleskörű funkcionális:** A Django számos beépített funkciót és keretrendszert tartalmaz a webes alkalmazások gyakori feladatainak elvégzésére, mint például hitelesítés, engedélyezés, URL-vezérlés, sablonrendszer, űrlapok kezelése és adatbázis-kezelés.
- **Skálázhatóság:** A Django nagy terhelésű webes alkalmazások számára is skálázható, köszönhetően rugalmas architektúrájának és optimalizált teljesítményének.
- **Biztonság:** A Django beépített biztonsági funkciókat biztosít, amelyek megvédik a webes alkalmazásokat a gyakori biztonsági fenyegetésektől.
- **Aktív közösség:** A Django-nak nagy és aktív közössége van, amely támogatást és útmutatást nyújt a fejlesztőknek.

2.4.1. Django Architektúra

A Django architektúra [33] az MVT struktúrát követi. Az MVT-ben az M a modellre (Model), a V a nézetekre (View), a T pedig a sablonokra (Template) utal. A modell az adatbázisban tárolt adatok struktúráját határozza meg, a nézet egy Python függvény, amely a webes kéréseket kezeli, a sablon pedig statikus tartalmakat tartalmaz, mint például HTML, CSS és JavaScript.

Modell

A Django alkalmazások Python objektumokat, úgynevezett modelleket használnak az adatok eléréséhez és kezeléséhez. Ezek a modellek definiálják az adattárolás belső struktúráját, például a mezőtípusokat és azok maximális méretét, az alapértelmezett értékeket, az űrlapok címke szövegét stb. A modellek létrehozása független attól, hogy melyik adatbázist használja. Bármely adatbázist használhatja. Nem kell közvetlenül kommunikálnia az adatbázissal. Csak létre kell hoznia a modellstruktúrát és a többi kódot, a Django pedig elvégzi a piszkos munkát az adatbázissal való kommunikáció terén.

Nézet

A Django-ban egy nézetfüggvény egy olyan Python függvény, amely fogad egy webes kérést és továbbít egy webes választ. A Django nézetek válasza bármi lehet, amit egy webböngésző megjeleníthet, beleértve egy weboldal HTML tartalmát, egy átirányítást, egy XML dokumentumot, egy 404-es hibát, egy képet stb. A Django nézetek az alkalmazás felhasználói felületének részét képezik, mivel olyan weboldalakat adnak vissza, amelyek HTML/CSS/JavaScript kódot tartalmaznak a Django sablonjában.

Sablon

A Django sablon a Django projekt statikus tartalmát tartalmazza, mint például a HTML-t, a CSS-t és a JavaScriptet, a projektben használt képekkel együtt. A Django sablonok elérési útját a Django projekt beállítási fájljából (setting.py) állíthatjuk be.

Django Sablon Nyelv (DTL)

Röviden, a Django Sablon Nyelvet DTL-nek (Django Template Language) nevezik. A Django sablonok saját szintaxist használnak az adatok weboldalakon történő megjelenítéséhez. A Django-ban egy kontextust használnak a sablonok rendereléséhez. Amikor egy sablont renderelnek, a benne lévő változókat az értékeikre cseréli a rendszer, amelyeket a kontextusban keres, és a címkéket végrehajtja. A többi tartalom változtatás nélkül kerül megjelenítésre.

2.4.2. Django URL-ek

Az URL-ek definiálják az alkalmazás útvonalát és leképezését. Azok az urls.py fájlban vannak meghatározva, amely az URL minta alapján a felhasználó kérését egy konkrét nézethez társítja. A modell kezeli az adatokat és az üzleti logikát, a nézet a megjelenítési logikát, a sablon a felhasználói felületet, az URL pedig a web alkalmazás útvonalazását és leképezését. Ez jobb kódstruktúrához, karbantarthatósághoz és skálázhatósághoz vezet.

2.4.3. Django ORM

Az ORM [34] a "Object-relational mapper" (Objektum-Relációs Leképező) kifejezés rövidítése. Az ORM a Django egyik leghatékonyabb tulajdonsága, amely lehetővé teszi, hogy az adatbázissal úgy kommunikáljon, mintha SQL-t használna. Valójában a Django ORM egy olyan Python-specifikus technika, amellyel lekérdezéseket és módosításokat végezhet az adatbázisán, és eredményeket kaphat. Ez egy igazán nagyszerű megoldás, amely kihasználja a Python néhány fejlett funkcióját, hogy megkönnyítse a fejlesztők életét. A Django ORM elsősorban alacsony és közepes összetettségű lekérdezések kezelésére lett tervezve.

2.5. Django és ASP.NET összegzés

Mind az ASP.NET, mind a Django modern, professzionális webalkalmazás-fejlesztési keretrendszerek, amelyek képesek komplex rendszerek kiszolgálására. A választás elsősorban a fejlesztési környezet, nyelvi preferenciák és az alkalmazás célja alapján történik.

A szakdolgozat tárgyát képező alkalmazás esetében – amelyben a képfeldolgozás és AI modellek központi szerepet kapnak – a Django előnyösebb választás, mivel:

- Python nyelven működik, amely natív környezetet biztosít az AI modellekhez,

- egyszerű az integráció a PyTorch modellel,
- beépített adminisztrációs felületet kínál,
- gyors prototípus-készítést tesz lehetővé,
- és erős közösségi támogatással rendelkezik.

ASP.NET ugyanakkor ideális lehet azok számára, akik Microsoft-alapú ökoszisztémában dolgoznak, C# nyelvet preferálnak, vagy nagyvállalati, erőforrás-igényes, skálázható rendszert fejlesztenek.

2.6. PostgreSQL Adatbázis

A PostgreSQL [35] egy nyílt forráskódú relációs adatbázis-kezelő rendszer (RDBMS), amely megbízhatóságáról, rugalmasságáról és a nyílt műszaki szabványok támogatásáról ismert. A hagyományos relációs adatbázis-kezelő rendszerektől eltérően a PostgreSQL támogatja mind a relációs, mind a nem relációs adattípusokat. Ez teszi az egyik legmegbízhatóbb, legstabilabb és legfejlettebb relációs adatbázissá a piacon. A PostgreSQL 1986-ban az INGRES (az 1970-es évek elején indult nyílt forráskódú SQL relációs adatbázis-projekt) továbbfejlesztéseként jött létre. Michael Stonebraker, a Berkeley Egyetem informatika professzora volt az ötletgazdája. 1994-ben a projekt kiterjesztette a támogatást az SQL nyelvre, és röviddel ezután született meg a PostgreSQL.

A PostgreSQL folyamatosan fejlődik, egy lelkes fejlesztői közösség gondoskodik a rendszer továbbfejlesztéséről.

Főbb jellemzői:

- **Objektum-relációs:** Az objektum-orientált modellezési paradigmát támogatja, lehetővé téve az adatok hatékony tárolását és lekérdezését komplex objektum hierarchiák esetén.
- **Skálázható:** Nagy terhelésű webes alkalmazások és adatbázisok igényeinek is képes megfelelni.
- **Megbízható:** ACID (Atomicity, Consistency, Isolation, Durability) tranzakciós támogatást nyújt az adatintegritás és a konzisztencia biztosítása érdekében.
- **Nyílt forráskódú:** Ingyenesen használható és módosítható, aktív közösséggel rendelkezik.
- **Széleskörű funkcionalitás:** Támogatja a SQL lekérdezési nyelvet, az adatbázis-felhasználók és jogosultságok kezelését, az adatbetöltést és -kiírást, valamint a különböző adatbázis-kezelési funkciókat.

2.6.1. Integrálása a Django keretrendszerrel

A Django számos beépített funkciót és eszközt kínál az adatbázisokhoz való csatlakozáshoz és a lekérdezések végrehajtásához. A PostgreSQL integrálása a Django-val egyszerű és hatékony. A Django ORM (Object-Relational Mapper) segítségével az objektumorientált kódot leképezi adatbázisbeli táblákra és mezőkre. Ez leegyszerűsíti az adatbázis-műveleteket és megkönnyíti a modellek és az adatbázis közötti interakciót.

A Django beépített adatbázis-konfigurációs beállításai lehetővé teszik a PostgreSQL csatlakozási adatainak definiálását. A Django ORM automatikusan kezeli az adatbázis-

kapcsolatokat és a lekérdezéseket, így a fejlesztőnek nem kell aggódnia az alacsony szintű adatbázis-műveletekkel kapcsolatban.

Előnyök:

- **Hatékony:** A Django ORM gyors és hatékony adatbázis-műveleteket biztosít.
- **Egyszerű:** Az objektumorientált programozási modell megkönnyíti az adatbázis-kezelést.
- **Skálázható:** A Django és a PostgreSQL együttesen nagy terhelésű webes alkalmazások adatbázis-igényeinek is képesek megfelelni.
- **Megbízható:** A PostgreSQL ACID tranzakciós támogatása biztosítja az adatintegritást és a konzisztenciát.

2.7. Adatgyűjtés és Adatkészletek

A gépi tanulási modellek hatékonyak, de csak akkor, ha jó minőségű adatokkal képezik ki őket. A jó adatkészlet reprezentatív, pontos és elegendő mennyiségű adatot tartalmaz a modell hatékony betanításához. A növénybetegségek felismerésének gépi tanulási modelljei esetében az adatkészletnek tartalmaznia kell a növények képeit, valamint a képeken látható betegségek címkéit. Az adatkészletnek a lehető legnagyobb mértékben változatosnak kell lennie, hogy a modell megtanulhassa felismerni a különböző növénybetegségeket különböző körülmények között.

Az alábbiakban néhány népszerű adatkészlet található a növénybetegségek felismeréséhez:

- **PlantVillage:** Ez a GitHubon elérhető adatkészlet [36] több mint 54 000 képet tartalmaz 38 különböző osztály áll rendelkezésre. A képeket különböző kamerákkal és különböző megvilágítási körülmények között készítették.
- **PlantDoc:** Az adathalmaz [37] összesen 2 598 adatpontot tartalmaz, 13 növényfaj és legfeljebb 17 betegségosztály mentén. Az interneten talált képek annotálása körülbelül 300 emberi munkaórát vett igénybe.
- **New Plant Diseases Dataset:** Ez a Kaggle-adatbázis [38] több mint 87 000 képet tartalmaz 38 különböző növénybetegségről. Ez az adathalmaz offline augmentációval lett újragenerálva az eredeti PlantVillage adathalmazból

2.7.1. Adatok előkészítése

Az adatok előkészítése [39] a gépi tanulási munkafolyamat fontos része. Olyan képeket szeretnénk választani, amelyek a legnagyobb értéket nyújtják egy robusztus és pontos modell kiképzéséhez. Íme néhány szempont, amelyeket figyelembe kell venni a legjobb képek kiválasztásakor:

- **Objektumlefedettség:** A képeken a célobjektum (pl. paradicsomlevél) jól kivehető kell legyen. A képmanipulációs technikák például: automatikus cropping segítségével javítható az objektumközpontság. A rosszul levágott vagy részben takart objektumokat tartalmazó képeket automatikus szűréssel például: YOLO-detektor confidence score alapján lehet kiszűrni.

- **Objektumvariáció:** A képadatok változatosságát data augmentation technikákkal növelhetjük, pl. forgatás, tükrözés, fényerő-módosítás, random zoom vagy elforgatás. Ez segíti a modellt a különböző valós helyzetekre való felkészülésben.
- **Képek minősége:** Előnyben kell részesíteni a jó minőségű és tisztaságú képeket. A homályos, zajos vagy alacsony felbontású képek negatívan befolyásolhatják a modell objektumfelismerési pontosságát. A képminőség automatikusan becsülhető blur detection (pl. Laplacian variance) vagy noise level estimation segítségével. Az alacsony kontrasztú, zajos vagy homályos képeket ezek alapján kiszűrhető, vagy denoising algoritmusokkal (pl. Gaussian blur, median filter) javítható.
- **Annotáció pontossága:** Az annotációk ellenőrzéséhez használhatók bounding box consistency ellenőrzések. Automatikus annotálás során aktív tanulás (active learning) segíthet a hibás példák kiszűrésében. A pontos és precíz határolókeret-annotációkkal rendelkező képek jobb tanulási eredményekhez vezetnek.
- **Osztályegyensúly:** Biztosítani kell a különböző objektumkategóriák közötti képek egyensúlyát. Ha a kategóriák nagyjából egyenlő arányban képviseltetik magukat az adatkészletben, az segít megelőzni, hogy a modell a tanulás során bizonyos osztályokat előnyben részesítsen vagy figyelmen kívül hagyjon. Az osztályeloszlás elemzéséhez statisztikai mutatók például: histogrammok használhatók.
- **Képek diverzitása:** Olyan képeket is illesszünk be, amelyeket különböző forrásokból, szögekből, nézőpontokból vagy beállításokból készítettek. Ez a sokféleség segíti a modellt az új és ismeretlen adatokra való általánosításban. A képdiverzitás biztosítható különböző adatforrások integrálásával például: saját gyűjtés vagy nyílt adatbázisok.
- **Redundancia elkerülése:** Távolítsuk el az adatkészletből azokat a képeket, amelyek túlzottan hasonlítanak egymásra, vagy duplikált példányok. Az ilyen redundáns minták torzíthatják a modell tanulását, mivel a gyakran ismétlődő vizuális minták miatt a háló túltanulhat bizonyos jellemzőkre. A hasonlóság mérése érdekében használható előtanított konvolúciós neurális hálózatokból (pl. ResNet) származó deep feature embeddingeket, majd ezek alapján koszinusz-hasonlóságot vagy euklideszi távolság számolható. Az egymáshoz túl közeli embeddingek redundanciára utalnak, és ezek közül elegendő csak egy reprezentatív képet megtartani.
- **Minőségellenőrzés:** Végezhető minőségellenőrzést az adatkészleten, hogy a kiválasztott képek megfeleljenek a kívánt kritériumoknak, és mentesek legyenek anomáliáktól, hibáktól vagy műtermékektől. A minőségellenőrzés történhet automatikus szabályokkal (pl. képfelbontás küszöbértékei, blur-szűrés), valamint vizuális ellenőrzéssel.

2.8. Konvolúciós neurális hálózatok

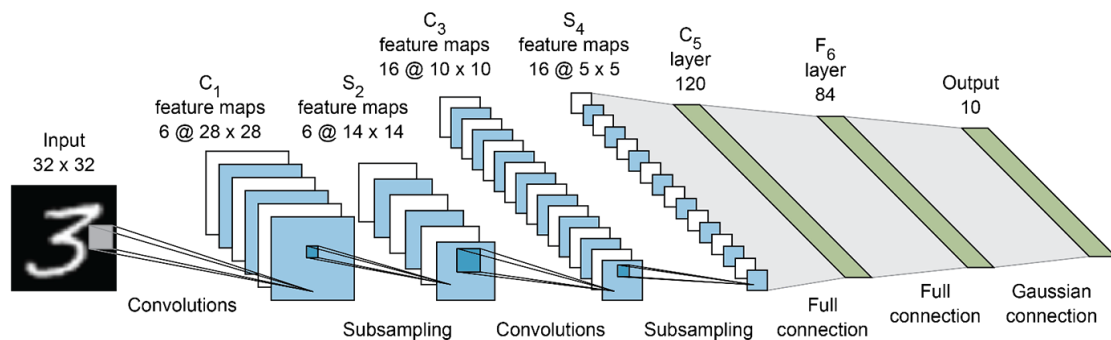
A konvolúciós neurális hálózatok (CNN-ek) a mélytanulás egyik legsikeresebb és legelterjedtebb architektúrái közé tartoznak, különösen képi adatok feldolgozásában [40]. Számos alkalmazásban bizonyították hatékonyságukat, például arcfelismerésben, önvezető autók látásrendszerében, orvosi képalkotásban, valamint a dolgozat témájához közvetlenül kapcsolódóan: növénybetegségek képalapú felismerésében is.

A CNN-ek egyik legnagyobb előnye, hogy képesek automatikusan megtanulni a bemeneti képek legfontosabb jellemzőit, és ezzel elkerülhető a manuális jellemzőképzés. Ez különösen hasznos mezőgazdasági környezetben, ahol a betegségek nagyon változatos módon jelenhetnek meg a leveleken – apró foltok, elszíneződések, deformációk formájában.

2.8.1. A konvolúciós réteg és működése

A CNN-ek legfontosabb építőeleme a konvolúciós réteg, amely a bemeneti képen egy kis méretű (pl. 3x3 vagy 5x5) szűrőt, más néven kernelt futtat végig, és ezáltal egy új képet, az úgynevezett feature mapet (jellemzőtérképet) hoz létre. Ez a művelet azt modellezi, hogyan "érzékeli" a hálózat a képen található alakzatokat és mintázatokat.

Matematikailag a konvolúció során a kernel mátrixát rácsúsztatjuk a bemeneti képre, és minden pozícióban kiszámítjuk a kernel és a képrészlet pontonkénti szorzatának összegét. Ezzel kiemelhetők például élek, textúrák, foltok – amelyek jellemzőek lehetnek egy adott növénybetegségre is. Az alábbi **2.4. ábra** szemlélteti ezt a működést.



2.4. ábra Konvolúciós neurális hálózat működése [41]

2.8.2. Aktivációs függvények

A konvolúciós réteg után általában nemlinearitást vezetünk be egy aktivációs függvény segítségével. Leggyakrabban a ReLU (Rectified Linear Unit) aktivációt használjuk, amely a negatív értékeket nullára csonkolja, a pozitívakat változatlanul hagyja:

$$f(x) = \max(0, x)$$

Ez segít a hálózatnak abban, hogy komplex, nemlineáris összefüggéseket tanuljon meg.

2.8.3. Pooling réteg (leképezés)

A pooling (általában max pooling) réteg célja a jellemzőtérkép méretének csökkentése, miközben a legfontosabb jellemzőket megtartja. Ez csökkenti a számítási igényt és növeli a hálózat robusztusságát a kisebb elmozdulásokkal szemben. Például egy 2x2-es max pooling réteg minden 2x2-es blokkban megtartja a legnagyobb értéket.

2.8.4. Mélyebb rétegek: hierarchikus jellemzőtanulás

A CNN-ek rétegzett struktúrája lehetővé teszi, hogy a hálózat a bemeneti képen egyre összetettebb jellemzőket tanuljon meg:

- Az első rétegek egyszerű éleket, vonalakat, színeket ismernek fel.
- A középső rétegek textúrákat, formákat, szimmetriákat tanulnak.

- A mélyebb rétegek már képesek teljes tárgyak (pl. levélszélek, foltok, sebhelyek) azonosítására.

Ez a hierarchikus tanulás különösen hatékony növénybetegségek felismerésénél, ahol a tünetek gyakran kombinált mintázatként jelennek meg a leveleken.

2.8.5. Teljes összeköttetésű (fully connected) réteg és osztályozás

A konvolúciós és pooling rétegek után a kimeneti jellemzők egy teljesen összekötött (fully connected) rétegre kerülnek, ahol az utolsó réteg általában egy softmax függvényt tartalmaz, amely minden lehetséges osztályhoz (pl. betegségfajta) hozzárendel egy valószínűségi értéket 0 és 1 között. A legnagyobb érték jelzi, hogy a hálózat szerint melyik osztályba tartozik a kép.

2.8.6. Tanítás és optimalizálás

A CNN-ek tanítása során visszaterjesztéssel (backpropagation) és gradiens alapú optimalizálással (pl. SGD vagy Adam algoritmus) a hálózat frissíti a szűrők és kapcsolatok súlyait annak érdekében, hogy minimalizálja az osztályozási hibát. Ehhez nagy mennyiségű címkézett kép szükséges (pl. „Tomato_Leaf_Mold” vagy „Apple_Black_Rot”).

A veszteségfüggvény (loss function) általában keresztentrópia, amely méri a hálózat kimenete és a valós címkék közötti eltérést.

2.8.7. A CNN előnyei a növénybetegség-felismerésben

A konvolúciós neurális hálózatok egyik legnagyobb előnye, hogy képesek automatikusan megtanulni a bemeneti képeken jelen lévő jellemzőket, így nincs szükség arra, hogy a tüneteket manuálisan definiáljuk. Ez a képesség különösen értékes olyan komplex mintázatok esetén, mint amilyenek a növénybetegségek vizuális megnyilvánulásai [42]. Továbbá a CNN-ek jól skálázhatók: ha rendelkezésre áll elegendő mennyiségű és megfelelően címkézett kép, akkor újabb betegség típusok is könnyen bevonhatók a rendszerbe anélkül, hogy a meglévő architektúrát jelentősen módosítani kellene. Emellett ezek a hálózatok robusztusnak bizonyulnak a valós környezetből származó képekkel szemben is, mivel viszonylag jól teljesítenek eltérő fényviszonyok, változó kameraállások vagy részben takart levelek esetén is. Mindezek felül gyakorlati szempontból is előnyös, hogy a megfelelő előfeldolgozás mellett a CNN-alapú rendszerek akár mobiltelefonos képeken is megbízhatóan működhetnek, ami lehetővé teszi a technológia közvetlen mezőgazdasági alkalmazását terepen is.

2.9. PyTorch könyvtár

A PyTorch [43] egy nyílt forráskódú gépi tanulási (ML) keretrendszer, ami Python programozási nyelven és a Torch könyvtáron alapszik. A Torch egy nyílt forráskódú ML könyvtár, melyet mély neurális hálózatok kreálására használnak, és a Lua szkriptnyelven írták. Ez egyike a legkedveltebb platformoknak a mélytanulási kutatásokhoz. A keretrendszert úgyformálták, hogy felgyorsítsa az átmenetet a kutatási prototípusoktól a gyakorlati alkalmazásig. A PyTorch több mint 200 különféle matematikai műveletet támogat. Növekvő népszerűségét annak köszönheti, hogy leegyszerűsíti a mesterséges neurális hálózatok

modelljeinek létrehozását. A PyTorch-ot elsősorban adatkutatók alkalmazzák kutatáshoz és mesterséges intelligencia (AI) alkalmazásokhoz.

2.9.1. A PyTorch működése

A PyTorch egy rugalmas és „pythonic” mélytanulási keretrendszer, amely azt jelenti, hogy szorosan illeszkedik a Python nyelv szintaxisához és filozófiájához. Ez lehetővé teszi a fejlesztők számára, hogy olvasható, természetes stílusú kódot írjanak, amely gyorsan fejleszthető és könnyen hibakereshető. A Python egyik fő előnye — amely a PyTorch népszerűségét is erősíti — a dinamikus számítási gráfok támogatása. Ennek köszönhetően a modellek viselkedése futás közben módosítható, így a kutatók és mérnökök részletekbe menően tesztelhetik és finomíthatják a hálózat egyes részeit anélkül, hogy a teljes programot újra kellene futtatni.

A PyTorch legfontosabb jellemzői:

- **Tenzorszámítás:** A PyTorch tenzorai hasonlóak a NumPy többdimenziós tömbjeihez, de lehetőséget adnak a GPU-alapú gyorsításra. Ezek az n-dimenziós tömbök rendkívül hatékonyan használhatók numerikus műveletekre, és egy könnyen kezelhető API-n keresztül manipulálhatók.
- **TorchScript:** A PyTorch egyik gyártásra szánt komponense, amely lehetővé teszi a fejlesztők számára, hogy dinamikus és statikus végrehajtás között váltsanak. Optimalizálja a kód teljesítményét, skálázhatóságát és használhatóságát éles környezetben.
- **Automatikus differenciálás:** Ez a szolgáltatás a visszaterjesztés (backpropagation) alapját képezi. A PyTorch automatikusan képes kiszámítani a gradiens értékeket, így nagyban megkönnyíti a neurális hálózatok tanítását.
- **Python ökoszisztéma integráció:** A PyTorch zökkenőmentesen működik együtt a Pythonban széles körben használt könyvtárakkal, például a NumPy, SciPy, Cython vagy Numba csomagokkal, így jól illeszkedik a tudományos számítási környezetekbe.
- **Változók:** A tenzorokhoz hasonló objektumok, amelyek automatikusan nyomon követik a számításokat és tárolják a gradiens értékeket. Ezek képviselik a számítási gráf csomópontjait.
- **Paraméterek:** Speciális típusú változók, amelyeket jellemzően a tanulható hálózati súlyok reprezentálására használnak. Ezek lehetővé teszik a PyTorch számára, hogy automatikusan kezelje a tanítás során optimalizálandó értékeket.

2.10. TensorFlow könyvtár

A TensorFlow egy nyílt forráskódú platform machine learninghez, melyet adatfolyam-gráfok segítségével valósít meg. A platformot adattudósok, szoftverfejlesztők és oktatók egyaránt széles körben használják. A gráf csomópontjai matematikai műveleteket ábrázolnak, míg az élek a közöttük áramló, többdimenziós adattömböket (tenzorokat) jelentik. Ez a rugalmas architektúra lehetővé teszi a gépi tanulási algoritmusok összekapcsolt műveletek gráfként történő leírását. A TensorFlow modelleket GPU-kon, CPU-kon és TPU-kon is lehetőség van futtatni és tanítani különböző platformokon anélkül, hogy a kódot újra kellene írni. Ez a skálázhatóság a hordozható eszközöktől az asztali gépeken át a high-end szerverekig terjed. Ez azt jelenti, hogy a különböző háttérű programozók ugyanazokat az eszköztárat használhatják az együttműködésre, jelentősen javítva a hatékonyságot [44].

2.10.1. A TensorFlow működése

A TensorFlow munkafolyamatát három megkülönböztető rész határozza meg: az adatok előkészítése, a modell felépítése és a modell tanítása az előrejelzések készítéséhez. A keretrendszer többdimenziós tömbökként, úgynevezett tenzorokként fogadja az adatokat, és kétféle módon hajtja végre a műveleteket. Az elsődleges módszer egy számítási gráf felépítése, amely meghatározza az adatáramlást a modell tanításához. A második, és gyakran intuitívabb módszer az eager execution használata, amely imperatív programozási elveket követ, és azonnal kiértékeli a műveleteket.

2.10.2. PyTorch vs TensorFlow

A PyTorch-ot gyakran hasonlítják össze a TensorFlow-val, amely egy mélytanulási keretrendszer, amit a Google fejlesztett. Mivel a TensorFlow régebb óta létezik, nagyobb fejlesztői közösséggel és bőségesebb dokumentációval rendelkezik.

Ugyanakkor a PyTorch több előnnyel is bír a TensorFlow-val szemben. A PyTorch dinamikusan definiálja a számítási gráfokat, szemben a TensorFlow statikus megközelítésével. A dinamikus gráfok valós időben módosíthatók. Emellett a TensorFlow tanulási görbéje meredekebb, mivel a PyTorch egy intuitív Python-alapú rendszer.

A TensorFlow jobban megfelelhet olyan projektekhez, amelyeknél fontos a gyártási környezet és a skálázhatóság, mivel eleve ilyen célokra fejlesztették ki. A PyTorch ezzel szemben egyszerűbb és könnyebben kezelhető, így ideális gyors prototípus-készítéshez és kutatáshoz.

2.11. Git Verziókezelő

Git egy népszerű verziókövető rendszer szoftverfejlesztéshez. Azzal segít a fejlesztőknek, hogy nyomon kövessék a kód változásait, együttműködjenek a csapat többi tagjával, és visszalépjenek a projekt előző verzióira, ha szükséges. A Git eltér a hagyományos verziókövető rendszerektől, mivel minden fejlesztőnek rendelkeznie kell a projekt teljes másolatával a saját gépén. Ez lehetővé teszi az offline munkát és a párhuzamos fejlesztést. A változtatások megosztása a csapat többi tagjával egyszerű, mivel a Git könnyen kezelhető eszköztár air rendelkezik a különböző ágazatok (branches) kezelésére és a módosítások összevonására (merging) [45].

2.11.1. A Git előnyei

- **Offline munka:** A Git lehetővé teszi az offline munkát, mivel a fejlesztőknek rendelkezniük kell a projekt teljes másolatával a saját gépükön.
- **Párhuzamos fejlesztés:** A fejlesztők egyidejűleg dolgozhatnak a projekten anélkül, hogy egymás munkáját zavarnák. Ezt a különálló ágazatok (branches) támogatása teszi lehetővé.
- **Korábbi verziókhoz való visszatérés:** A Git segítségével könnyen visszatérhetsz a projekt előző verzió
- **Verziókövetés:** A Git nyomon követi a projekt összes változását, így áttekintheted a kód módosításainak történetét, és megtudhatod, ki, mikor és mit változtatott. Ez segít a hibák forrásának azonosításában és a projekt korábbi, stabil állapotához való visszatérésében.

- **Összevonás (Merging):** A Git lehetővé teszi a különböző ágazatok módosításainak összevonását a projekt fő ágazatába. Ez megkönnyíti a csapatmunka összehangolását és a különböző fejlesztések integrálását.
- **Pull requestek:** A pull requestek segítségével a fejlesztők megoszthatják a kódváltoztatásaikat a csapattal, mielőtt azokat egyesítenék a fő ágazattal. Ez lehetővé teszi a kód áttekintését és a hibák kiszűrését a projektbe való betörés előtt.
- **Közösségi támogatás:** A Git egy nyílt forráskódú projekt, hatalmas közösséggel és számos tanulási eszközzel. Ez azt jelenti, hogy könnyen találsz segítséget, ha elakadsz valamiben.
- **Integráció más eszközökkel:** A Git integrálható a legtöbb fejlesztői eszközzel és platformmal, beleértve az IDE-ket (Integrated Development Environment), a folyamatos integrációs/folyamatos szállítási (CI/CD) eszközöket és a feladatkövető rendszereket. Ez egyszerűsíti a munkafolyamatot és javítja a hatékonyságot.

2.11.2. A Git működése

A 2.5. ábra a Git verziókövető rendszer alapvető munkafolyamatát mutatja be. A munkafolyamat három fő lépésből áll:

1. Working Directory

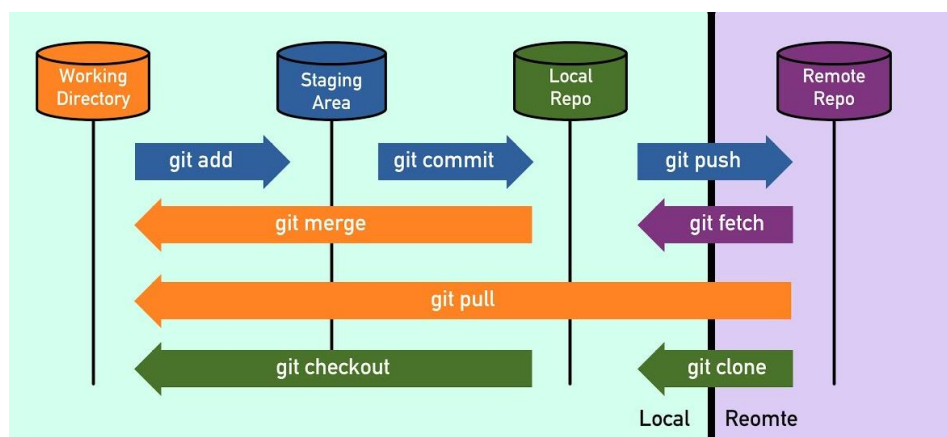
- A bal oldalon, azaz "Working Directory" mappában tárolódnak a projekt fájllai.
- A fájlok módosításakor a változások nem kerülnek automatikusan a Git adatbázisba.
- A fájlok állapotát a „. git” mappában tárolt metaadatok mutatják.

2. Staging Area

- A "Staging Area" egy virtuális terület, ahová a fájlokat manuálisan kell felvenni a "git add" paranccsal.
- Csak a Staging Areában lévő fájlok kerülnek be a következő commitba.
- A "git status" paranccsal ellenőrizhetjük, hogy mely fájlok módosultak és melyek vannak a Staging Areában.

3. Git Repository

- A "Local Repo" a Git adatbázis helyi másolata.
- A "Remote Repo" a Git adatbázis távoli másolata, amely egy szerveren tárolódik.
- A "git commit" paranccsal menthetjük a Staging Areában lévő fájlokat a helyi adatbázisba.
- A "git push" paranccsal szinkronizálhatjuk a helyi adatbázist a távoli adatbázissal.



2.5. ábra A git működése [46]

3. KÖVETELMÉNY SPECIFIKÁCIÓ

Funkcionális követelmények

- **Felhasználói felület:**
 - Intuitív és könnyen használható felület.
 - Képfeltöltési lehetőség a növényekről.
 - Növénybetegségekre és kezelési lehetőségekre vonatkozó információk elérhetősége.
 - Feltöltött képek böngészése
 - Mobilbarát felület
- **Felhasználói fiók kezelés**
 - Regisztrációs lehetőség új felhasználók számára
 - Bejelentkezés és kijelentkezés lehetősége hitelesítési folyamaton keresztül.
 - Jelszómódosítás lehetősége a bejelentkezett felhasználók számára.
 - Elfelejtett jelszó esetén jelszó-visszaállítási folyamat, amely e-mailben küldött hivatkozással történik.
 - Felhasználói profil megtekintése, aktivitási előzmények megtekintése.
- **Növénybetegség-felismerés:**
 - Magas pontosságú felismerés a különböző növénybetegségek tekintetében.
 - Model bizalmi szint megjelenítése a felhasználónak.
- **Növényfelismerés:**
 - A feltöltött képek alapján növényfajta felismerése.
 - A felismerési eredmény jelenjen meg közérthető névvel és bizalmi szinttel.
- **Adatbázis:**
 - A rendszer PostgreSQL adatbázist használ, amely ACID-kompatibilis, transzparensszen kezeli a konkurens tranzakciókat.
 - A képek nem közvetlenül az adatbázisban kerülnek tárolásra, hanem fájlrendszerben, és csak a metaadatok (útvonal, típus, státusz, tulajdonos) kerülnek az adatbázisba. Ez biztosítja a gyorsabb válaszidőt és a tárhely hatékony kezelését.
 - Az adatbázisséma normalizált, a képek, felhasználók, eredmények és betegségleírások külön táblákban szerepelnek idegen kulcsos kapcsolatokkal.
 - A rendszer támogatja a többfelhasználós működést, a felhasználókhoz tartozó képfeltöltések és eredmények izoláltan kezelhetők.

Teljesítményi követelmények

- **Adatbázis indexelés:** A PostgreSQL teljesítményét egyedi indexek definiálásával lehet optimalizálni a Django Meta.indexes segítségével. Ezek az indexek a gyakran keresett mezőkre kerülhetnek beállításra, ami jelentősen csökkenti a lekérdezések válaszidejét.
- **Backend válaszidő minimalizálása:** A predikciós endpointok és a képfeltöltés aszinkron módon kerülhetnek feldolgozásra, ahol lehetséges, így nem blokkolják a fő folyamatokat.
- **Naprakész naplózás és hibakövetés:** A Sentry és a lokális logrendszer segítségével azonnali információt kaphatunk az esetleges teljesítményproblémákról, lehetővé téve az időben történő optimalizálást.

Megbízhatósági követelmények

- **Tranzakciókezelés:** A Django `transaction.atomic` dekorátorával biztosítható, hogy a kapcsolódó adatbázis-műveletek egy tranzakciós blokkban történjenek, így hiba esetén minden visszavonható. Ez garantálhatja az adatintegritást, például: ha a predikció mentése vagy kép feltöltése megszakad, nem marad vissza félkész adat.
- **Rendszeres naplózás:** A rendszer minden fontos eseményét érdemes naplózni – hibák, felhasználói aktivitás, rendszerfolyamatok – konzolra, fájlba, adatbázisba és Sentry-be. Ez segítheti a hibák visszakövetését és a problémák gyors javítását.

Biztonsági követelmények

- **Hitelesítés és jogosultságkezelés:** Csak bejelentkezett, hitelesített felhasználók férjenek hozzá a képfeltöltéshez és a predikcióhoz. Az adatokhoz való hozzáférés jogosultságalapú, és minden művelethez szükséges a megfelelő azonosítás.
- **CSRF védelem:** A Django `CsrfViewMiddleware` automatikusan védi az összes POST alapú műveletet, így a rendszer védett az oldal-közi kérés hamisítás ellen.
- **Környezeti változók használata:** Az érzékeny adatok (pl. adatbázis elérési adatok, API kulcsok) nem kerülnek be a forráskódba, hanem `.env` fájlban keresztül töltődnek be az `environ` csomag segítségével.
- **Naplózás érzékeny adatokkal:** A `send_default_pii=True` beállítással a Sentry automatikusan csatolja a hibához tartozó felhasználói adatokat is, így segíti a célzott hibakezelést, de ezek nem kerülhetnek nyilvános naplózásra.
- **Statikus és médiafájlok szétválasztása:** A statikus fájlokat (pl. CSS, JS) és a felhasználók által feltöltött képeket (MEDIA) külön-kezelendő, ez megakadályozza, hogy rosszindulatú fájlok közvetlenül elérhetővé váljanak. A médiához szükséges jogosultságokat csak a Django biztosítja.

4. TERVEZÉS

Ebben a fejezetben a rendszer architektúrájának részletes leírását mutatom be, beleértve az egyes komponenseket, azok kapcsolatait és a tervezési döntéseket. A cél egy átfogó kép biztosítása a rendszerről és annak működéséről, amely mind a fejlesztők, mind a nem technikai érdeklődők számára hasznos.

A tervezési folyamat során a következő szempontokat tartottam szem előtt:

- **Moduláris felépítés:** A rendszert független modulokra bontottam, amelyek jól definiált interfészekkel kommunikálnak egymással. Ez elősegíti a karbantartást, a skálázhatóságot és a tesztelhetőséget.
- **Skálázhatóság:** A rendszert úgy terveztem, hogy képes legyen kezelni a növekvő terhelést és a felhasználói bázist. Ehhez horizontális skálázási technikákat, például terheléelosztókat és cache-t fogok alkalmazni.
- **Megbízhatóság:** A rendszert úgy terveztem, hogy nagy rendelkezésre állást és adatintegritást biztosítson. Ehhez redundáns komponenseket, hibakeresési mechanizmusokat.
- **Biztonság:** A rendszert úgy terveztem, hogy megvédje az adatokat a jogosulatlan hozzáféréstől, a módosítástól és a törléstől. Ehhez titkosítási technikákat és hitelesítési mechanizmusokat fogok alkalmazni.
- **Teljesítmény:** A rendszert úgy terveztem, hogy hatékony legyen és gyors válaszidőket biztosítson. Ehhez optimalizálási technikákat, cache-t és hardveres gyorsítást fogok alkalmazni.

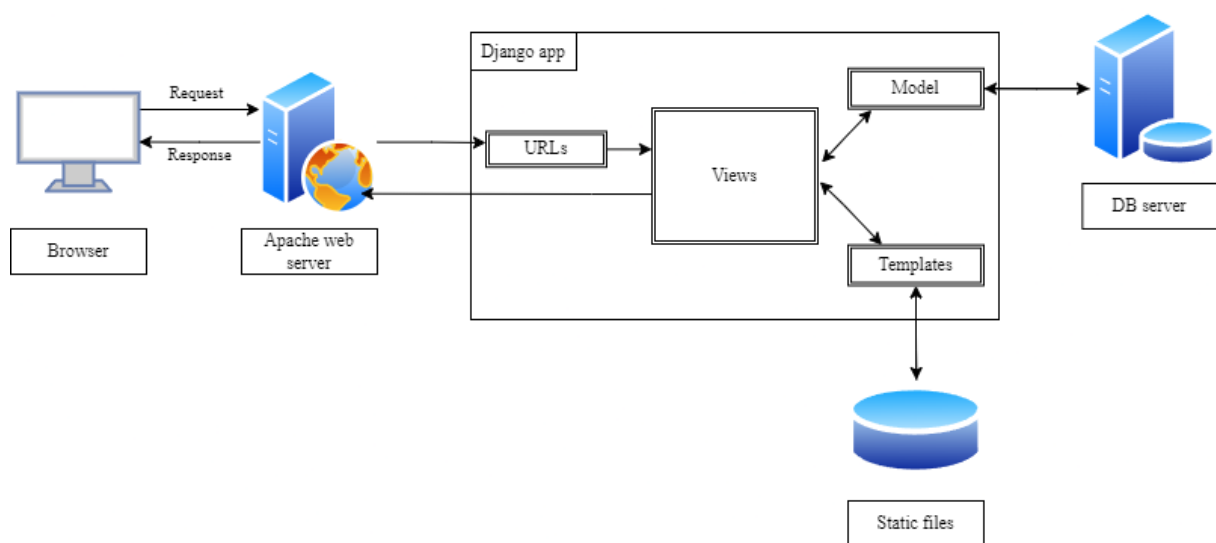
4.1. Rendszerterv

A Django architektúra működésének folyamata a következő lépésekből áll:

1. **Böngésző kérés:** A felhasználó böngészője HTTP kérést küld a webalkalmazás számára a Django által definiált URL-eken keresztül. A kérés tartalmazza az URL-t és a kéréstípust (pl. GET, POST stb.).
2. **Web szerver (Apache):** Az Apache web szerver fogadja a HTTP kérést, és továbbítja azt a Django alkalmazásnak.
3. **URL továbbítása a Django alkalmazásnak:** A Django alkalmazás definiál URL-mintákat és a hozzájuk tartozó nézet függvényeket. Az Apache továbbítja a kapott kérést a Django alkalmazásnak az URL-minták alapján.
4. **Nézet függvények (Views):** Az URL-mintához rendelt nézet függvények felelősek a kérés feldolgozásáért. Ezek a függvények végzik a szükséges adatok lekérdezését az adatbázisból (modellek segítségével), feldolgozzák a kérést, és visszaadják a megfelelő választ.
5. **Kommunikáció a modellel:** A nézet függvények kommunikálnak a modellekkel, amelyek a Django alkalmazás által definiált adatmodelljeit képviselik. Ezek a modellek felelősek az adatbázis tábláinak kezeléséért, az adatok lekérdezéséért, módosításáért és törléséért.
6. **Válasz előkészítése és renderelése:** A nézet függvények válaszokat készítenek elő a kérés alapján. Ezek lehetnek HTML oldalak, JSON adatok vagy más típusú válaszok. Az előkészített válaszokat a nézet függvények visszatérési értéként adják vissza.

7. **Sablonok (Templates):** Az előkészített válaszok tartalmazhatnak sablonokat, amelyek HTML kódokat tartalmaznak. Ezek a sablonok tartalmazhatnak dinamikus adatokat, amelyeket a nézet függvények adják át nekik.
8. **Kommunikáció a statikus fájlokkal:** A sablonok gyakran hivatkoznak statikus fájlokra, például CSS, JavaScript vagy képek. Ezeket a statikus fájlokat a Django szolgáltatja a böngészőnek a megfelelő URL-eken keresztül.
9. **Válasz visszaküldése a böngészőnek:** Az előkészített válaszokat visszaküldik az Apache web szervernek, amely továbbítja azokat a böngészőnek. A böngésző megjeleníti a kapott választ a felhasználó számára.

A **4.1. ábra** ábrázolja a Django architektúra működését, amelynek során a kérések és válaszok közvetlenül vagy közvetve haladnak át a böngészőtől a nézet függvényeken és a modelleken keresztül a sablonokig és vissza.



4.1. ábra Django Architektúra

4.1.1. Authentication alkalmazás végpontjai

A **4.1. táblázat** az authentication alkalmazás végpontjait összesíti, feltüntetve az URL-t, a HTTP-módszert, a nézet nevét vagy funkcióját, a kapcsolódó modellt, valamint a végpont rövid leírását.

URL	Típus	Nézet neve	Modell	Leírás
/	GET	home	-	Ha a felhasználó be van jelentkezve, automatikusan átirányítja a /recognition/disease/ oldalra. Ha nincs bejelentkezve, megjeleníti a nyitólapot.

/signup/	GET/POST	signup	User	GET: megjeleníti a regisztrációs űrlapot. POST: feldolgozza a regisztrációt. Ellenőrzi a jelszavakat, menti a felhasználót, inaktíválja, majd aktiváló e-mailt küld.
/activate/<uidb64>/<token>/	GET	activate	User	A felhasználói fiók aktiválását végzi a token alapján.
/login/	GET/POST	login	-	GET: belépési oldal megjelenítése. POST: bejelentkezteti a felhasználót, ha az adatok helyesek.
/logout/	GET	signout	-	Kijelentkezteti a felhasználót és Redirect: vissza a login oldalra.
/profile/	GET	profile	Image, Result	Megjeleníti a felhasználó profilját, illetve aktivitását

4.1. táblázat Authentication alkalmazás főbb végpontjai

4.1.2. Recognition alkalmazás végpontjai

A **4.2. táblázat** a recogniton alkalmazás végpontjait összesíti, feltüntetve az URL-t, a HTTP-módszert, a nézet nevét vagy funkcióját, a kapcsolódó modellt, valamint a végpont rövid leírását.

URL	Típus	Nézet neve	Modell	Leírás
/recognition/<image_type>/	GET/POST	recognition	Image	GET: megjeleníti a felhasználó által feltöltött képeket. POST: több kép feltöltését kezeli image_type szerint
/evaluate-disease/	POST	evaluate_disease	Image, Result	Kiválasztott képek alapján

				betegségfelismerés történik
/evaluate-plant/	POST	evaulate_plant	Image, Result	Kiválasztott képek alapján PlantNet külső API hívás
/delete_images/	POST	delete_images	Image, Result	Kiválasztott képek törlése fájlrendszerből, adatbázisból

4.2. táblázat Recognition alkalmazás főbb végpontjai

4.2. Adatbázisterv

A tervezés során a Django keretrendszer és annak beépített táblái kerülnek felhasználásra a felhasználói hitelesítéshez, engedélykezeléshez, naplózáshoz, valamint a növényekre, betegségekre, felismerésekre és képekre vonatkozó adatok tárolásához.

Fájltárolás a Szerveren:

A képek tárolására a Django ImageField mezőt használok, ami a fájlokat a projekt média könyvtárába menti a szerveren. Ez a beállítás egyszerű és könnyen implementálható, azonban nagyobb mennyiségű kép esetén érdemes lehet megfontolni a következőket:

- **Saját fájlrendszer:** A Django modelljeimtől függetlenül a szerver saját fájlrendszere is használható a képek tárolására. Ez nagyobb rugalmasságot biztosít a fájlkezelésben, de a fájl elérési útvonalak kezelését és a biztonsági szempontokat külön kell megvalósítani.

Django Beépített Táblák:

A Django számos beépített táblát biztosít a felhasználói hitelesítéshez, engedélyek kezeléséhez és naplózáshoz. A rendszerben az AbstractUser modellt használok, ezt a modellt kibővítettem egy email mezővel, amely egyedi azonosítónak (username) szolgál.

- **User:** Tárolja a felhasználók alapvető adatait, mint a név, jelszó, e-mail cím stb.
- **Permission:** Tárolja a rendszerben elérhető engedélyeket.
- **ContentType:** Tárolja az engedélyek típusait (pl. app_label, model).
- **LogEntry:** Tárolja az admin felületen végrehajtott műveletek naplóbejegyzéseit.
- **Group:** Tárolja a felhasználói csoportokat.
- **Sessions:** Tárolja az aktív felhasználói munkamenetek adatait.
- **AbstractBaseSession:** Az Sessions tábla absztrakt bázis osztálya.

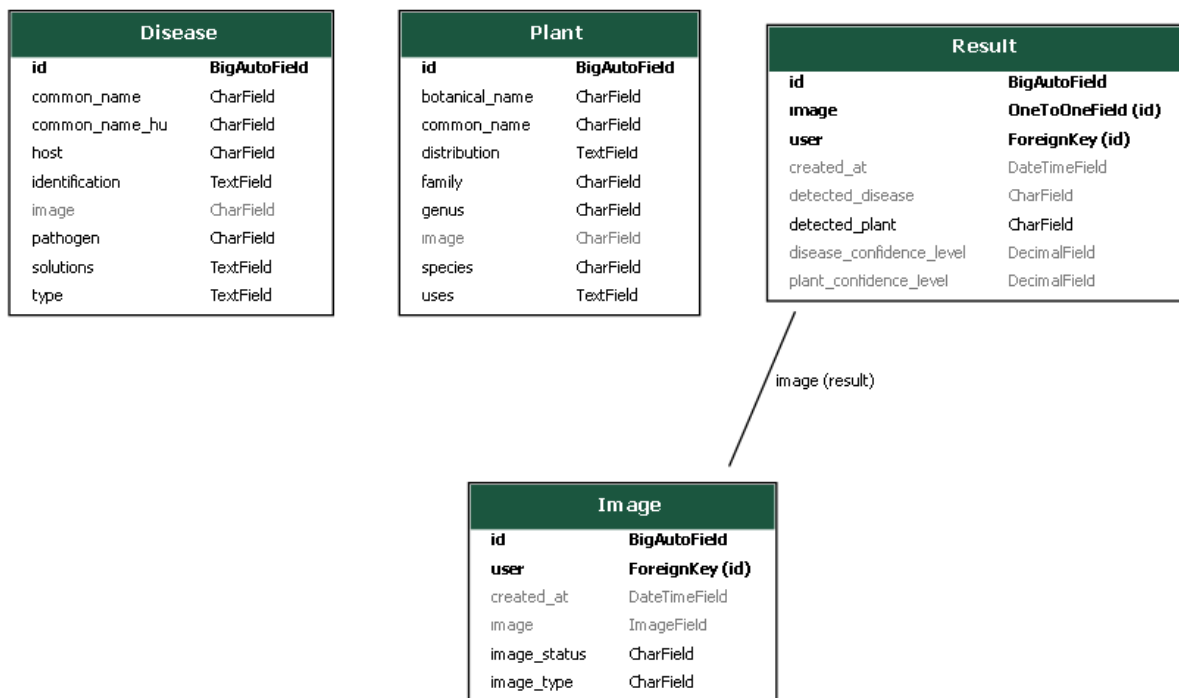
Specifikus Táblák:

A Django beépített táblái mellett a következő táblákat definiáltam a rendszer adatainak tárolására:

- **Növények (Plant):** Tárolja a növények nevét, típusát és egyéb releváns információkat.
- **Betegségek (Disease):** Tárolja a betegségek nevét, leírását, tüneteit, kezelési javaslatait és egyéb releváns információkat.

- **Felismerések (Result):** Tárolja a felismert betegségeket, a növényeket, a felhasználókat, a bizalmi szintjekeket a detektálás pontosságának meghatározására, a dátumot és a képekre vonatkozó hivatkozást.
- **Képek (Image):** Tárolja a képek fájlneveit, feltöltési dátumait, valamint a feldolgozási állapotot.

Az alábbi **4.2. ábra** szemléletesen ábrázolja az adatbázis fontos tábláinak a struktúráját, beleértve a képek és a felismerések közötti kapcsolatot.:



4.2. ábra Adatbázis ER modell

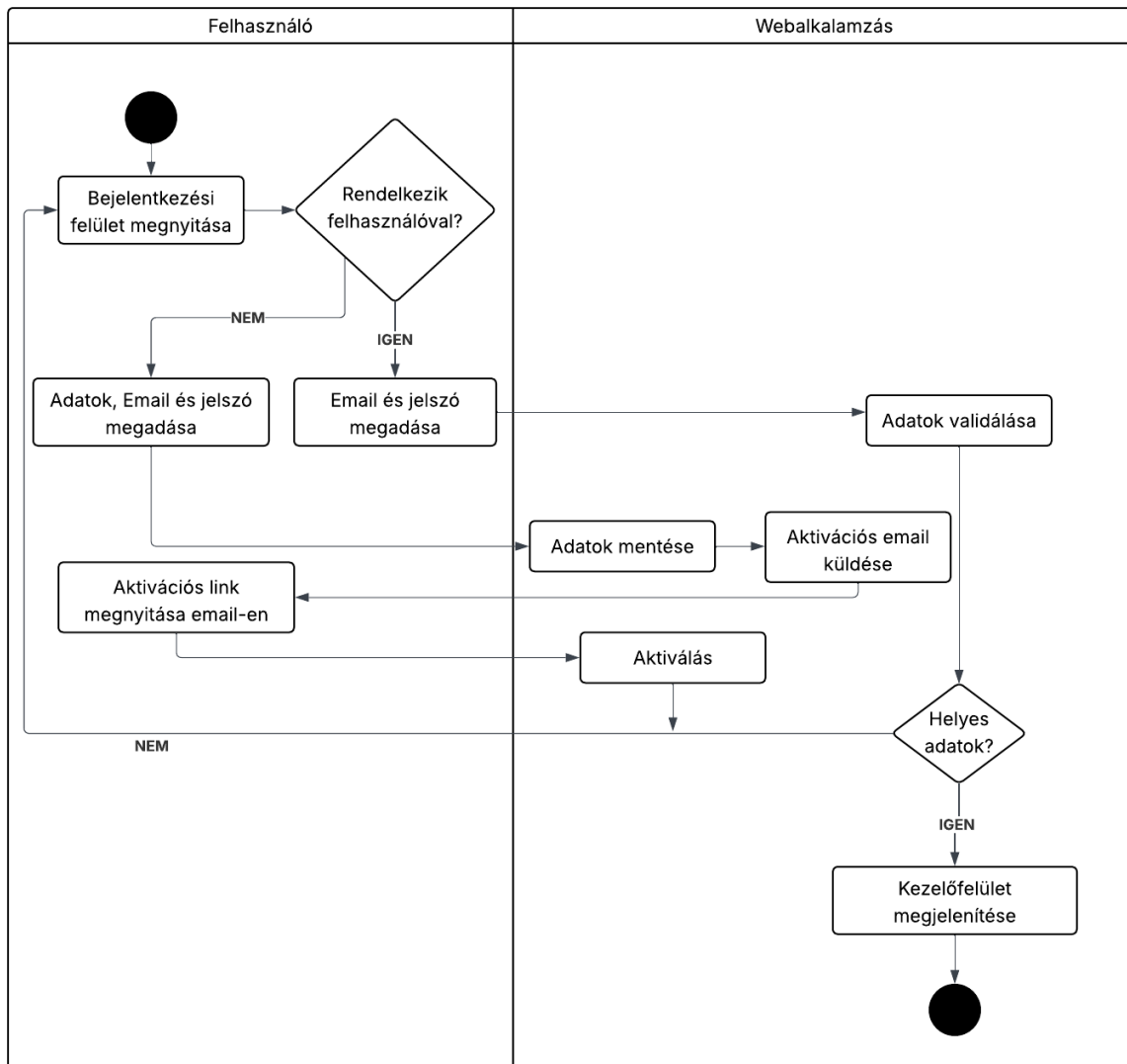
4.3. Funkciólista

- **Regisztráció és bejelentkezés:** A felhasználók regisztrálhatnak és bejelentkezhetnek a rendszerbe.
- **Elfelejtett jelszó:** A felhasználók igényelhetnek új jelszót, email-ben kap jelszó visszaállítási linket a felhasználó.
- **Képfeltöltés:** A felhasználók feltölthetik képeket a növényeikről a rendszerbe.
- **Betegségfelismerés:** A felhasználók igényelhetik a betegség kiértékelést a kiválasztott képekhez.
- **Növényfelismerés:** A felhasználók igényelhetik a növény kiértékelést a kiválasztott képekhez.
- **Korábban feltöltött képek böngészése:** A felhasználók böngészhetik a rendszerben tárolt képeiket és azok adatait.
- **Növény/betegség böngészése:** A felhasználók böngészhetik, a betegségeket ezek, leírását, tüneteit, kezelési javaslatait és a növényeknek a hasznos információit.
- **Jelszómódosítás:** A felhasználók módosíthatják a jelszavukat.
- **Aktivitási előzmények:** A felhasználók láthatják az aktivitási előzményeket.

- **Adminisztrációs felület:** Az adminisztrátorok kezelhetik a felhasználókat, a jogosultságokat, a csoportokat és a beállításokat.

4.3.1. Regisztráció és bejelentkezés

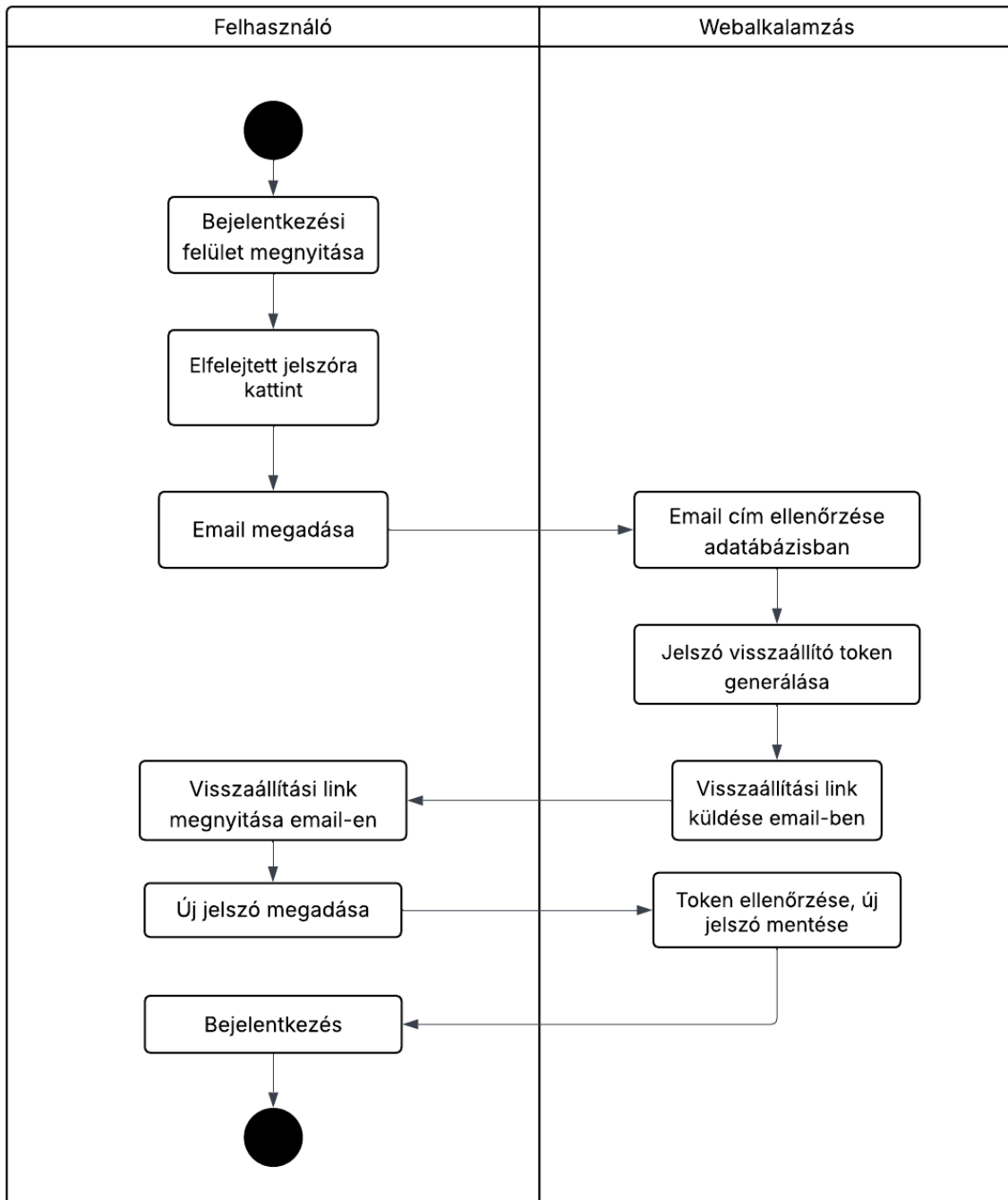
A regisztrálásnál a felhasználónak meg kell adni az email címét, a teljes nevét és egy jelszót, sikeres regisztráció esetén a felhasználó egy emailt kap, amivel tudja aktiválni a fiókját, ezután be is tud lépni a felületre. Az alábbi **4.3. ábra** ennek a folyamatát szemlélteti.



4.3. ábra Regisztráció és bejelentkezés Activity Diagram

4.3.2. Elfelejtett jelszó

Elfelejtett jelszó esetében a felhasználónak van lehetősége visszaállítási linket igényelni, miután megadta az email címét a rendszer generál egy egyszer használatos linket, amit elküld az email címre, ahol a felhasználó meg tudja változtatni a jelszavát. Az alábbi **4.4. ábra** ennek a folyamatát szemlélteti.



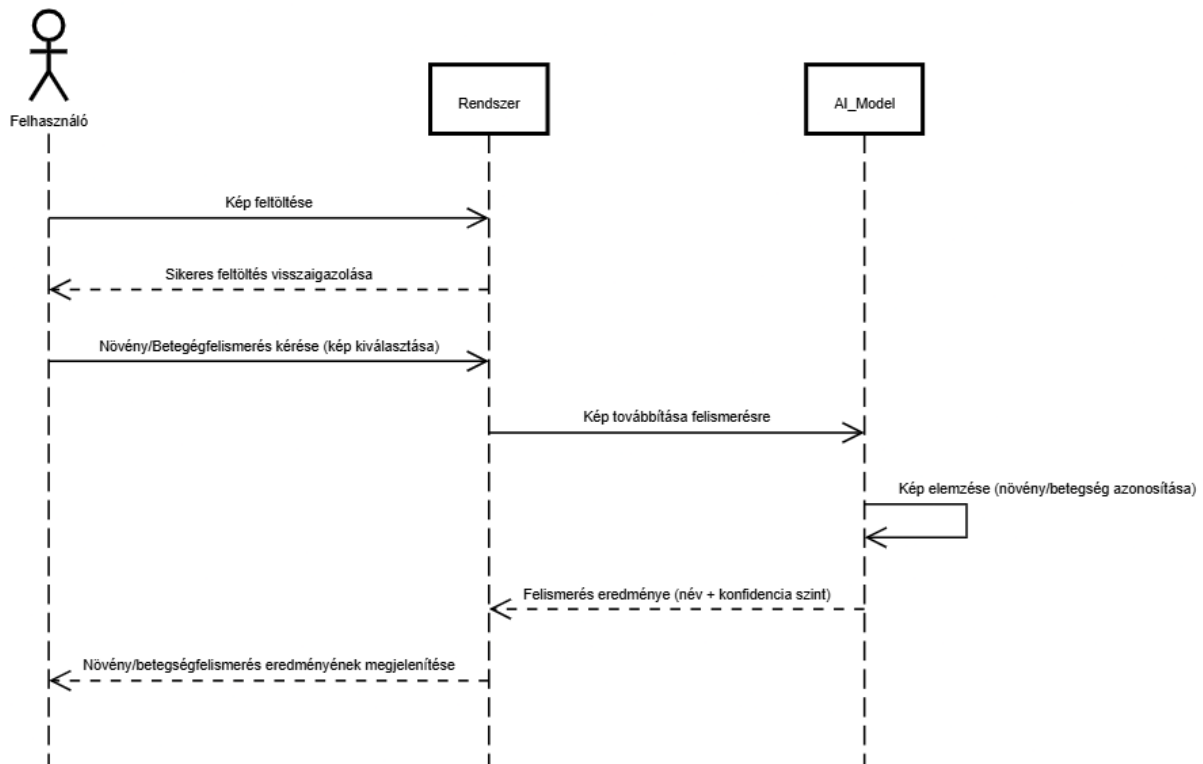
4.4. ábra Elfelejtett jelszó Activity Diagram

4.3.3. Képfeltöltés és betegségfelismerés

A rendszerben kizárólag bejelentkezett felhasználók férhetnek hozzá a kép alapú növény- és betegségfelismerési funkciókhoz. A betegségfelismerés esetén a felhasználó először képeket tölt fel a rendszerbe, majd a feltöltést követően kiválaszthatja, mely képeket kíván kiértékelteni. A kiválasztott képeket a rendszer automatikusan előfeldolgozza: azokat egységes méretre átméretezi (256x256 pixel), normalizálja, tenzorral alakítja, és a PyTorch-alapú, előre betanított ResNet9 neurális hálózat bemenetére továbbítja. A modell ezt követően meghatározza, hogy a képen milyen betegség látható, és a felismeréshez egy konfidenciaszintet

(valószínűségi érték) rendel. Az eredményeket a rendszer az adatbázisba menti, összekapcsolva a megfelelő kép- és felhasználói bejegyzésekkel.

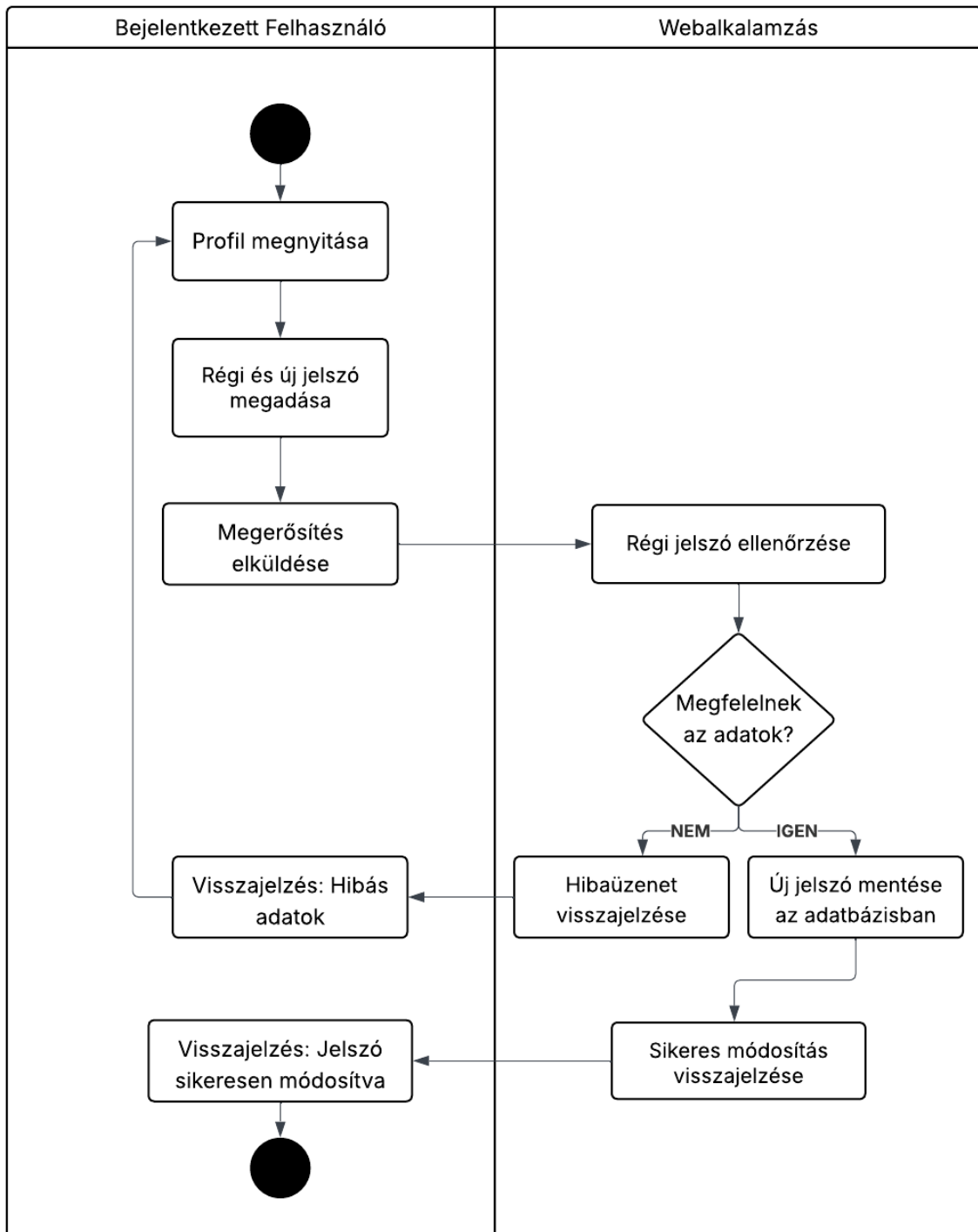
A növényfelismerés folyamata hasonlóképpen zajlik, azzal a különbséggel, hogy ebben az esetben nem saját AI modell végzi az azonosítást, hanem a rendszer a PlantNet szolgáltatás API-ján keresztül küld HTTP POST kérést a kiválasztott képekkel. A kérés tartalmazza az API kulcsot, a nyelvi beállítást (magyar), valamint azt, hogy csak a legjobb találatot kérjük vissza. A válaszból a rendszer kiolvassa a legvalószínűbb növény nevét és a hozzá tartozó pontossági értéket. Ezután az eredményt hasonló módon eltárolja az adatbázisban. Az alábbi **4.5. ábra** ennek a folyamatát szemlélteti.



4.5. ábra Képfeltöltés és felismerés Sequence Diagram

4.3.4. Jelszó módosítás

Bejelentkezett felhasználóknak jelszó módosításra is van lehetőségük, ezt a profil oldalon tudják megtenni, miután megadát a régi és új jelszavukat. Az alábbi **4.6. ábra** ennek a folyamatát szemlélteti.



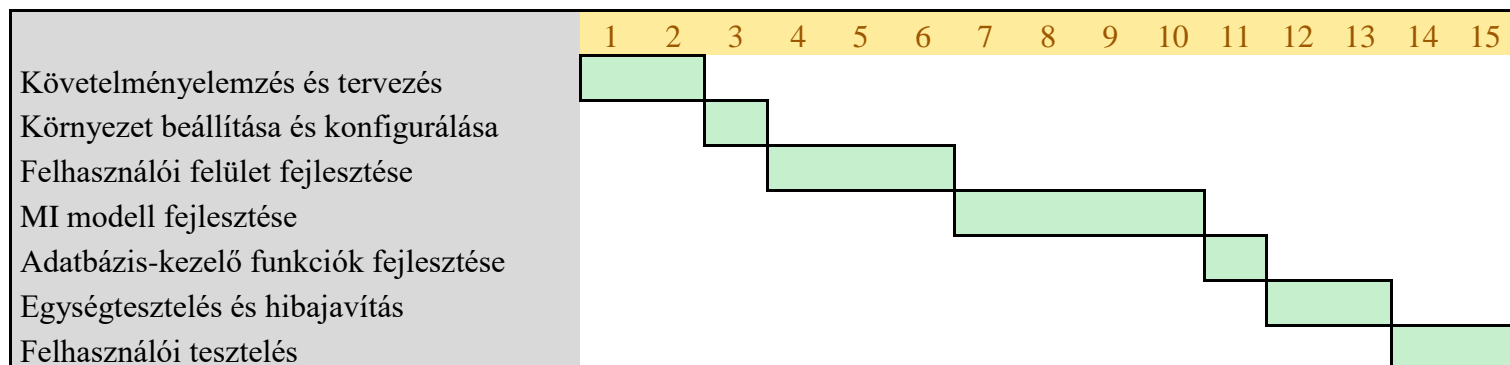
4.6. ábra Jelszó módosítás Activity Diagram

4.4. Fejlesztés tervezett menete

1. Követelményelemzés és tervezés (2 hét):
 - Követelmények felülvizsgálata és pontosítása
2. Környezet beállítása és projekt konfigurálása (1 hét):
 - Django projekt inicializálása
 - Adatbázis konfigurációja

- Környezet beállítása (pl. virtuális környezet létrehozása)
3. Felhasználói felület fejlesztése (3 hét):
 - Felhasználói felület tervezése és felépítése Django-ban
 - Felhasználói regisztráció és bejelentkezés funkciók implementálása
 - Képfeltöltés és kezelés funkcionalitás kialakítása
 - Felhasználói felület tesztelése és finomítása
 4. Mesterséges intelligencia modell fejlesztése (4 hét):
 - Tanító adatok összegyűjtése és előkészítése
 - PyTorch-ban alapvető modellarchitektúra kialakítása
 - Modell tanítása és finomhangolása
 - Modell tesztelése és pontosság validálása
 5. Adatbázis-kezelő funkciók implementálása (1 hét):
 - Adatbázis kapcsolatok kezelése Django ORM segítségével
 - Felhasználói adatok kezelése (regisztráció, bejelentkezés)
 - Kép- és eredményadatok kezelése
 6. Egységtesztelés és hibajavítás (2 hét):
 - Egységtesztok írása a különböző komponensekhez
 - Hibajavítás és funkciók finomhangolása
 7. Felhasználói tesztelés és visszajelzések alapján finomítás (2 hét):
 - Felhasználók általi tesztelés, visszajelzések gyűjtése
 - Visszajelzések alapján módosítások végrehajtása és finomítások elvégzése

A **4.3. táblázat** ezt a fejlesztési menetet mutatja be.



4.3. táblázat Fejlesztés tervezett menete lebontva hetekre

5. FEJLESZTÉS

A rendszer fejlesztése során a cél egy olyan webalkalmazás létrehozása volt, amely képes növényi képek alapján felismerni az adott növényfajt vagy az azon megjelenő betegséget. A fejlesztés főbb szakaszai a modell kiválasztása és betanítása, a webes backend és frontend kialakítása, valamint a modell integrálása a rendszerbe. Az alábbiakban bemutatásra kerülnek azok a fontosabb fejlesztési lépések, döntések és problémák, amelyek a megvalósítás során felmerültek.

5.1. Kezdeti tervezés

A projekt indulásakor célként tűztem ki, hogy a felhasználók képfeltöltéssel felismerhessék a növények fajtáját, illetve az esetleges betegségeket, valamint betekintést kapjanak ezek leírásába és kezelésébe. A fejlesztéshez a Django keretrendszert választottam, mivel gyors prototípus-készítést tesz lehetővé, és jól támogatja a REST API-k és felhasználókezelés kiépítését. A képosztályozó modellhez kezdetben TensorFlow és a DenseNet121 architektúra mellett döntöttem, azonban a későbbi tapasztalatok alapján ezt lecseréltem egy PyTorch-alapú ResNet9 modellre.

5.2. Adatbáziskezelés

A projekt során az adatok tárolására és kezelésére PostgreSQL adatbázist alkalmaztam. A PostgreSQL egy nyílt forráskódú, objektum-relációs adatbázis-kezelő rendszer, amely magas teljesítményt és megbízhatóságot biztosít. A Django és PostgreSQL közötti integráció zökkenőmentes, mivel a Django natívan támogatja a PostgreSQL-t, így az adatbázis-beállítások és migrációk kezelése egyszerű és hatékony. Az alkalmazás kezdetben SQLite adatbázison futott, amelyet a fejlesztési fázis elején használtam. Azonban a projekt növekedésével és az adatkezelési igények bővülésével átálltam a PostgreSQL-re, hogy biztosítsam a nagyobb adatbázis-kezelési teljesítményt és a jobb skálázhatóságot.

5.2.1. Migrációk kezelése

A Django adatbázis-migrációs rendszere lehetővé teszi a modellekben végrehajtott változtatások egyszerű átültetését az adatbázisba. A makemigrations és migrate parancsok segítségével a módosításokat alkalmazhatjuk, és biztosíthatjuk, hogy az adatbázis mindig naprakész legyen a modellek állapotával. A migrációk automatikusan létrehozzák az adatbázis táblákat és azok kapcsolatait, így a fejlesztési folyamat során nem kell kézzel kezelnem az adatbázis struktúráját.

5.2.2. Teljesítményoptimalizálás

PostgreSQL teljesítményének optimalizálásához használtam a beépített indexelési lehetőségeket. Django automatikusan létrehozza az indexeket a modellek mezőire, azonban további egyedi indexeket is definiáltam a Meta.indexes segítségével, a gyakran keresett mezőkhöz. Ez a megoldás segített gyorsabbá tenni a lekérdezéseket és biztosította a nagy adatbázisok kezelését anélkül, hogy azok jelentősen lelassították volna a rendszert.

5.2.3. Tranzakciókezelés

A PostgreSQL erőteljes tranzakciókezelést kínál, amely lehetővé teszi, hogy több adatbázis műveletet egy tranzakcióban hajtsunk végre. Ez különösen fontos volt a projekt során, mivel biztosítani kellett, hogy minden adatbázis művelet (pl. adatok beillesztése, frissítése, törlése) atomikusan történjen, és ne sérüljön az adatbázis integritása. Django automatikusan biztosítja a tranzakciók kezelését, és a `transaction.atomic` dekorátorral a kód olyan részeit, ahol több, egymással összefüggő adatbázisműveletet végzek tranzakciós blokkokba foglaltam.

5.3. Modellarchitektúra-váltás és implementációs szempontok

A fejlesztés elején a TensorFlow-t alkalmaztam a modell tanításához, DenseNet121 architektúrával. Ez a modell jól teljesített pontosság szempontjából, azonban a betanított modell mérete nagy volt, és a predikció sebessége több kép kértékelése esetén több perc volt (20 kép esetében a predikció sebessége 3 perc volt). Ezen kívül a TensorFlow integrálása Django alá nehezkesebb volt, különösen fájlfeltöltés és dinamikus modellbetöltés esetén.

Ezért döntöttem a váltás mellett: és egy ResNet9 architektúrát tanítottam PyTorch környezetben. A ResNet9 modell előnye, hogy kompaktabb, gyorsabban képes predikciót végezni a 20 kép kiértékelése összesen 4 másodpercet vett igénybe, és könnyebben kezelhető fájlformátumban (.pth) menthető. A döntést a gyakorlati szempontok vezérelték: fontosabb volt a gyors és megbízható működés, mint a pár százalékos pontosságbeli különbség.

A PyTorch modell köré külön előfeldolgozó és predikciós pipeline-t fejlesztettem, amely képes az előfeldolgozást (áthelyezés, átméretezés, normalizálás), a modell betöltését és a kimenetek értelmezését kezelni. Ezeket a függvényeket a `model_loader.py` fájlba szerveztem, így a Django rendszerből is könnyen elérhetők.

5.3.1. Adathalmaz

A Plant Disease Classification Dataset egy offline augmentációval bővített változata az eredeti PlantVillage Dataset-nek, és körülbelül 87 000 RGB képből áll, amelyek különböző növények egészséges és beteg leveleit ábrázolják. A képek 33 különböző osztályba vannak kategorizálva ezt jól mutatja a **5.1. táblázat Osztályok eloszlása**. A teljes adathalmaz 80/20 arányban van felosztva edzésre és validációra, miközben megőrzi a könyvtárstruktúrát. Az adathalmazban még található 33 tesztkép előrejelzés céljából.

Osztály neve	Képek száma
Apple_Apple_scab	2016
Apple_Black_rot	1987
Apple_Cedar_apple_rust	1760
Apple_Healthy	2008
Cherry_Healthy	1826
Cherry_Powdery_mildew	1683
Corn_Common_rust	1907
Corn_Gray_leaf_spot	1642
Corn_Healthy	1859
Corn_Northern_leaf_blight	1908
Grape_Black_rot	1888

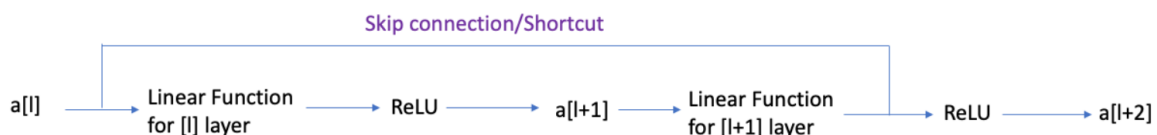
Grape_Esca	1920
Grape_Healthy	1692
Grape_Leaf_blight	1722
Peach_Bacterial_spot	1838
Peach_Healthy	1728
Pepper_Bacterial_spot	1913
Pepper_Healthy	1988
Potato_Early_blight	1939
Potato_Healthy	1824
Potato_Late_blight	1939
Strawberry_Healthy	1824
Strawberry_Leaf_scorch	1774
Tomato_Bacterial_spot	1702
Tomato_Early_blight	1920
Tomato_Healthy	1926
Tomato_Late_blight	1851
Tomato_Leaf_Mold	1882
Tomato_Septoria_leaf_spot	1745
Tomato_Spider_mites	1741
Tomato_Target_Spot	1827
Tomato_Tomato_mosaic_virus	1790
Tomato_Tomato_Yellow_leaf_curl_virus	1961

5.1. táblázat Osztályok eloszlása

5.3.2. Modell létrehozása

A növénybetegségek automatikus felismeréséhez egy ResNet9-alapú konvolúciós neurális hálózat került implementálásra. A modell kialakítása során cél volt a tanulási stabilitás, a mélyebb hálózati struktúra lehetővé tétele, valamint a túlilleszkedés minimalizálása.

A ResNet hálózatokban – a hagyományos neurális hálózatokkal ellentétben – nemcsak az egyik réteg adja tovább az eredményét a következő rétegnek, hanem ún. reziduális blokkokat használtam, amelyekben az egyes rétegek nemcsak a közvetlenül következő, hanem 2–3 lépéssel későbbi rétegekhez is közvetlen kapcsolatot létesítenek. Ez segít elkerülni az overfitting (túlilleszkedés) jelenségét, amikor a validációs veszteség egy ponton már nem csökken tovább, sőt emelkedni kezd, miközben a tanítási veszteség még mindig csökken. Emellett ez a megközelítés megelőzi az eltűnő gradiens problémát is, és lehetővé teszi, hogy mély neurális hálózatokat is hatékonyan tanítsam. Az alábbi **5.1. ábra** egy réteget mutat be a ResNet-ben.



5.1. ábra Egy réteg a ResNet-ben [47]

A maradék blokk nemcsak az előző réteg aktivációját használja, hanem a korábbi például az l-edik réteg kimenetét is átugró kapcsolaton (skip connection) keresztül hozzáadja a következő réteg bemenetéhez.

A folyamat a következőképpen írató le:

$$a^{[l+2]} = \text{ReLU}(z(a^{[l+1]}) + a^{[l]})$$

ahol:

- $a^{[l]}$: a l-edik réteg aktivációja (kimenete),
- $a^{[l+1]}$: a következő réteg aktivációja, amire újabb transzformációt végzünk,
- $z(\cdot)$: a lineáris transzformáció (pl. $W a + b$)
- $\text{ReLU}(x) = \max(0, x)$: a ReLU aktivációs függvény

Ez azt jelenti, hogy az eredeti információ ($a^{[0]}$) gyorsított úton továbbjut a mélyebb rétegekhez. Ha például a köztes súlymátrix nullára tanulna (tehát a köztes réteg semmit sem tanulna), a l+2-edik réteg még mindig képes tanulni a l-edik réteg kimenetéből.

A modell egy klasszikus konvolúciós neurális hálózat (CNN) architektúrát követ kimondottan képosztályozási feladatokra optimalizált. A hálózat szerkezete fokozatosan növekvő feature térképekkel és csökkenő térbeli dimenziókkal rendelkezik, ahogyan azt a **5.2. ábra** is szemlélteti.

Layer (type)	Output Shape	Param #
Conv2d-1	[-1, 64, 256, 256]	1,792
BatchNorm2d-2	[-1, 64, 256, 256]	128
ReLU-3	[-1, 64, 256, 256]	0
Conv2d-4	[-1, 128, 256, 256]	73,856
BatchNorm2d-5	[-1, 128, 256, 256]	256
ReLU-6	[-1, 128, 256, 256]	0
MaxPool2d-7	[-1, 128, 64, 64]	0
Conv2d-8	[-1, 128, 64, 64]	147,584
BatchNorm2d-9	[-1, 128, 64, 64]	256
ReLU-10	[-1, 128, 64, 64]	0
Conv2d-11	[-1, 128, 64, 64]	147,584
BatchNorm2d-12	[-1, 128, 64, 64]	256
ReLU-13	[-1, 128, 64, 64]	0
Conv2d-14	[-1, 256, 64, 64]	295,168
BatchNorm2d-15	[-1, 256, 64, 64]	512
ReLU-16	[-1, 256, 64, 64]	0
MaxPool2d-17	[-1, 256, 16, 16]	0
Conv2d-18	[-1, 512, 16, 16]	1,180,160
BatchNorm2d-19	[-1, 512, 16, 16]	1,024
ReLU-20	[-1, 512, 16, 16]	0
MaxPool2d-21	[-1, 512, 4, 4]	0
Conv2d-22	[-1, 512, 4, 4]	2,359,808
BatchNorm2d-23	[-1, 512, 4, 4]	1,024
ReLU-24	[-1, 512, 4, 4]	0
Conv2d-25	[-1, 512, 4, 4]	2,359,808
BatchNorm2d-26	[-1, 512, 4, 4]	1,024
ReLU-27	[-1, 512, 4, 4]	0
MaxPool2d-28	[-1, 512, 1, 1]	0
Flatten-29	[-1, 512]	0
Linear-30	[-1, 33]	16,929

5.2. ábra Modell architektúra

5.3.3. A modell tanítása

Mivel a saját gépemen nem áll rendelkezésre CUDA-kompatibilis videokártya, a modell tanítását a Google Colab környezetében végeztem el. A Colab lehetőséget biztosít NVIDIA T4-es GPU használatára, amely jelentősen meggyorsítja a mélytanulási modellek tréningjét CPU-hoz képest. A tanítás során több modern technikát alkalmaztam a tanulási folyamat optimalizálása érdekében:

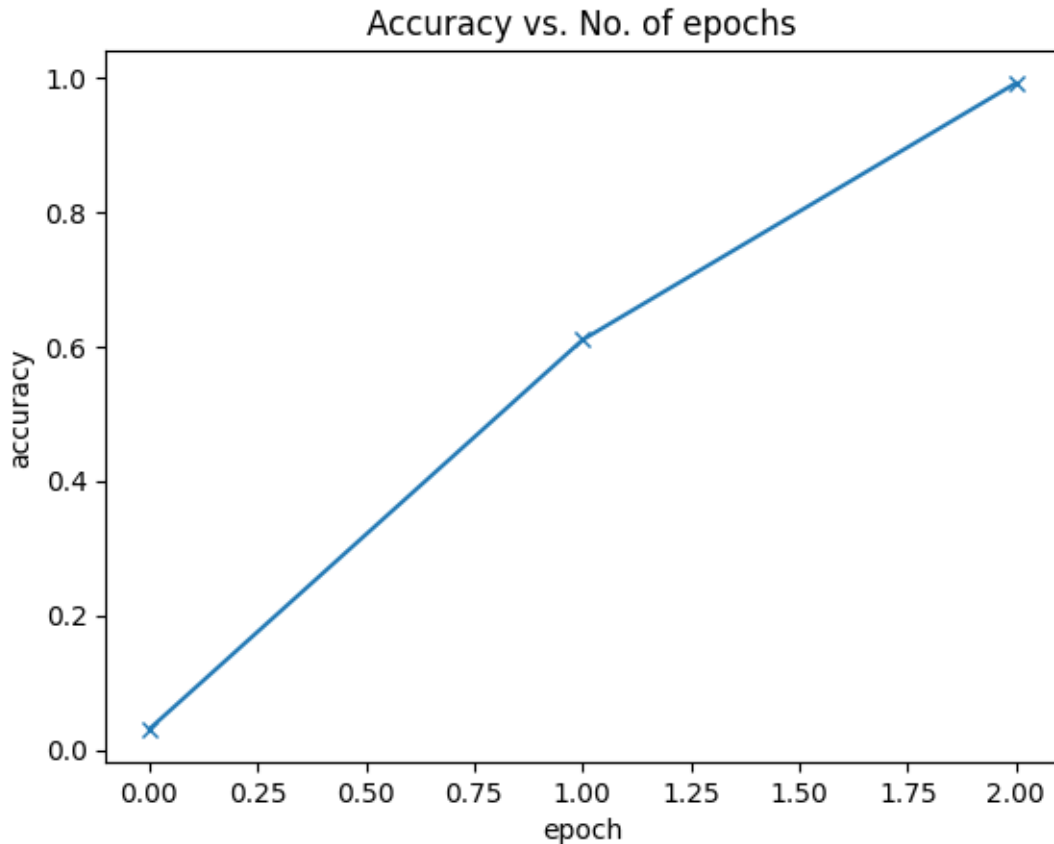
- **One Cycle Learning Rate Policy** (egyciklusos tanulási ráta ütemezés): Ahelyett, hogy fix tanulási rátát használnám, egy dinamikusan változó tanulási rátát alkalmaztam. Az első 30%-ban fokozatosan növeltem a tanulási rátát egy maximum értékig, majd fokozatosan csökkentettem azt a hátralévő epoch-ok során. Ez a módszer segít gyorsabb és stabilabb konvergenciát elérni.
- **Weight Decay** (súlycsökkenéses regulárizáció): Az overfitting elkerülése érdekében bevezettem egy extra regulárizációs tagot a veszteségfüggvénybe, amely megakadályozza, hogy a súlyok túlságosan nagy értékre nőjenek.
- **Gradient Clipping** (gradiens-levágás): A túl nagy gradiensértékek néha instabil tanításhoz vezethetnek. Ennek megelőzésére korlátoztam a gradiens értékeket egy meghatározott maximum értékre.
- **Adam optimalizáló**: A tanítás során az Adam (Adaptive Moment Estimation) optimalizálót használtam, amely a klasszikus SGD-nél gyorsabban és stabilabban konvergál, mivel adaptívan állítja a tanulási rátát minden paraméterre. Ez különösen hasznos lehet kisebb epochszám és összetettebb hálózat esetén.

Ez a kombinált megközelítés lehetővé tette, hogy rövid számítási idő és korlátozott erőforrások mellett is hatékonyan tanítsam be a modellt.

5.3.4. Tanítás eredménye

Az alábbi **5.3. ábra** szemlélteti, hogyan változott a modell validációs pontossága az első két tanítási ciklus (epoch) során. A modell kezdetben alacsony pontosságot ért el (kb. 0.03, vagyis ~3%), ami azt mutatja, hogy a kiindulási állapotban még nem volt képes megbízható

osztályozásra. Az első epoch végére azonban a pontosság 60.98%-ra nőtt, majd a második epochra drámaian megnőtt: elérte a 99.19%-ot.



5.3. ábra Pontosság értékének változása az epochok számának függvényében

5.4. Backend fejlesztés és modellintegráció

A Django backend kialakításakor külön alkalmazásokat hoztam létre az alábbi funkciókhoz:

- recognition: képfeldolgozás és predikció
- plant: adatlapok, leírások megjelenítése
- authentication: felhasználókezelés, saját adatok elérése
- core: gyökér funkcionalitás

A képek feltöltését az Image modell segítségével oldottam meg, amely tárolja a kép típusát (Plant vagy Disease), a feltöltés időpontját, valamint a fájlt. A felismerést követően a Result modell rögzíti a predikció eredményét: a detektált osztálynevet (pl. "Tomato Early Blight"), a bizalmi szintet, és kapcsolódik a feltöltött képhez, illetve a felhasználóhoz. A predikció során a rendszer egy képfeldolgozó pipeline-t használ, amely, betölti a képet, átméretezi a modellspecifikus bemeneti méretre, normalizálja a pixelértékeket, majd torch.Tensor típusra konvertálja.

5.5. Biztonsági intézkedések

A rendszer tervezése során külön figyelmet fordítottam az alkalmazás biztonságára, mind a felhasználói adatok védelme, mind a predikciós szolgáltatások megbízható működése érdekében. Az alábbi biztonsági elemeket építettem be a Django backendbe:

5.5.1. Jelszavak biztonságos kezelése

A Django beépítve támogatja a jelszavak bcrypt-alapú hashelését, így a felhasználói jelszavak nem kerülnek tárolásra tiszta szöveggént. A settings.py fájlban engedélyeztem a AUTH_PASSWORD_VALIDATORS beállításokat, amelyek automatikusan érvényesítik a jelszavak erősségét (minimális hossz, numerikus jelszavak tiltása stb.). Az egyedi User modellt az authentication alkalmazásban valósítottam meg, amely lehetőséget ad további testreszabásra, például e-mail alapú bejelentkezésre vagy extra biztonsági mezőkre.

5.5.2. CSRF védelem

A django.middleware.csrf.CsrfViewMiddleware automatikusan be van töltve a middleware láncban, ami minden állapotváltoztató (POST, PUT, DELETE) kérést véd Cross-Site Request Forgery (CSRF) támadások ellen. A frontend oldalon a CSRF token minden űrlappal együtt elküldésre kerül, így a rendszer megbízhatóan ellenőrzi a kérések forrását.

5.5.3. Környezeti változók használata

Az érzékeny adatok (pl. SECRET_KEY, adatbázis hozzáférés, e-mail jelszavak, PlantNet API kulcs, Sentry DSN) nem szerepelnek közvetlenül a forráskódban, hanem .env fájlban keresztül töltődnek be az environ csomag segítségével. Ez megakadályozza, hogy bizalmas információk szivárognak ki verziókezelő rendszeren keresztül.

5.5.4. Hibakezelés és naplózás

A backend részletes naplózási mechanizmust tartalmaz, amely négy szinten gyűjti a rendszerüzeneteket: fájlba, konzolra, adatbázisba és a Sentry hibakövető szolgáltatásba. A LOGGING szekcióban több naplókezelő (handler) került beállításra:

- **file:** a rendszer súlyosabb hibáit (ERROR szint felett) egy django_error.log nevű fájlba rögzíti. Ez különösen hasznos, ha a Sentry szolgáltatás nem elérhető vagy helyi hibakeresésre van szükség.
- **console:** a fejlesztés alatt álló alkalmazás azonnali hibáit jeleníti meg a terminálban, így gyors hibakeresést tesz lehetővé.
- **db:** egy egyedi, DatabaseLogHandler nevű logoló osztály segítségével nemcsak a hibák (error), hanem az általános információs (info) szintű események is mentésre kerülnek az adatbázisba. Ez lehetővé teszi a teljes rendszerfolyamatok naplózását, így nemcsak hibák, hanem műveleti nyomvonalak és felhasználói aktivitások is visszakereshetők.
- **sentry:** a rendszer legfontosabb hibáit automatikusan küldi a Sentry felhőalapú szolgáltatásba.

A naplózás során külön figyelmet fordítottam a formátumok konzisztenciájára, amely biztosítja a naplófájlok jól strukturált megjelenését (verbose és simple formátumokkal).

5.5.5. Sentry integráció

A rendszerhez beépítésre került a Sentry SDK, amely automatikusan gyűjti a kivételeket, hibás válaszokat és egyéb fontos eseményeket. A következő kulcsfontosságú előnyöket biztosítja:

- **Valós idejű hibaértesítés:** Amint egy kivétel történik a szerveren (pl. predikciós hiba, nem kezelt adatbázis-hiba stb.), azonnali riasztást kapok a Sentry-n keresztül e-mailben.
- **Felhasználói kontextus:** A `send_default_pii=True` beállításnak köszönhetően a Sentry a hiba mellé társítja a kéréshez tartozó felhasználói adatokat is (pl. azonosító, e-mail cím), így gyorsabban beazonosítható, kinél történt a probléma.
- **Trace és Stack Trace:** A rendszer hívásláncolata és teljes stack trace automatikusan naplózásra kerül, megkönnyítve a hibák reprodukálását.
- **Integráció Django-val:** A `DjangoIntegration` automatikusan figyeli a Django specifikus hibákat, így külön beállítás nélkül is támogatja a nézetekben, middleware-ekben vagy ORM-ben fellépő hibák monitorozását.

A Sentry kiválóan használható nemcsak hibák, hanem figyelmeztetések és teljesítményproblémák elemzésére is. Segítségével pontosan követni tudom, mikor, hol és miért omlik össze egy adott folyamat, és ez lehetővé teszi a gyors beavatkozást.

5.5.6. Jogosultságkezelés és autentikáció

A predikciós és képfeltöltési végpontokat kizárólag bejelentkezett, hitelesített felhasználók érhetik el. Ehhez a Django beépített autentikációs rendszerét használok, és minden kérést ellenőrzök, mielőtt érzékeny művelethez engednék hozzáférést. A személyes adatokhoz való hozzáférés kizárólag az adott felhasználó számára engedélyezett.

5.5.7. Statikus és médiatartalmak szétválasztása

A felhasználók által feltöltött tartalmak (`MEDIA_ROOT`) elkülönülnek a statikus fájloktól (`STATIC_ROOT`). Ez különösen fontos, ha a statikus fájlokat CDN-en vagy külön webszerveren keresztül szolgáljuk ki, míg a médiatartalomhoz szükséges jogosultságokat és hozzáférési szabályokat a Django biztosítja.

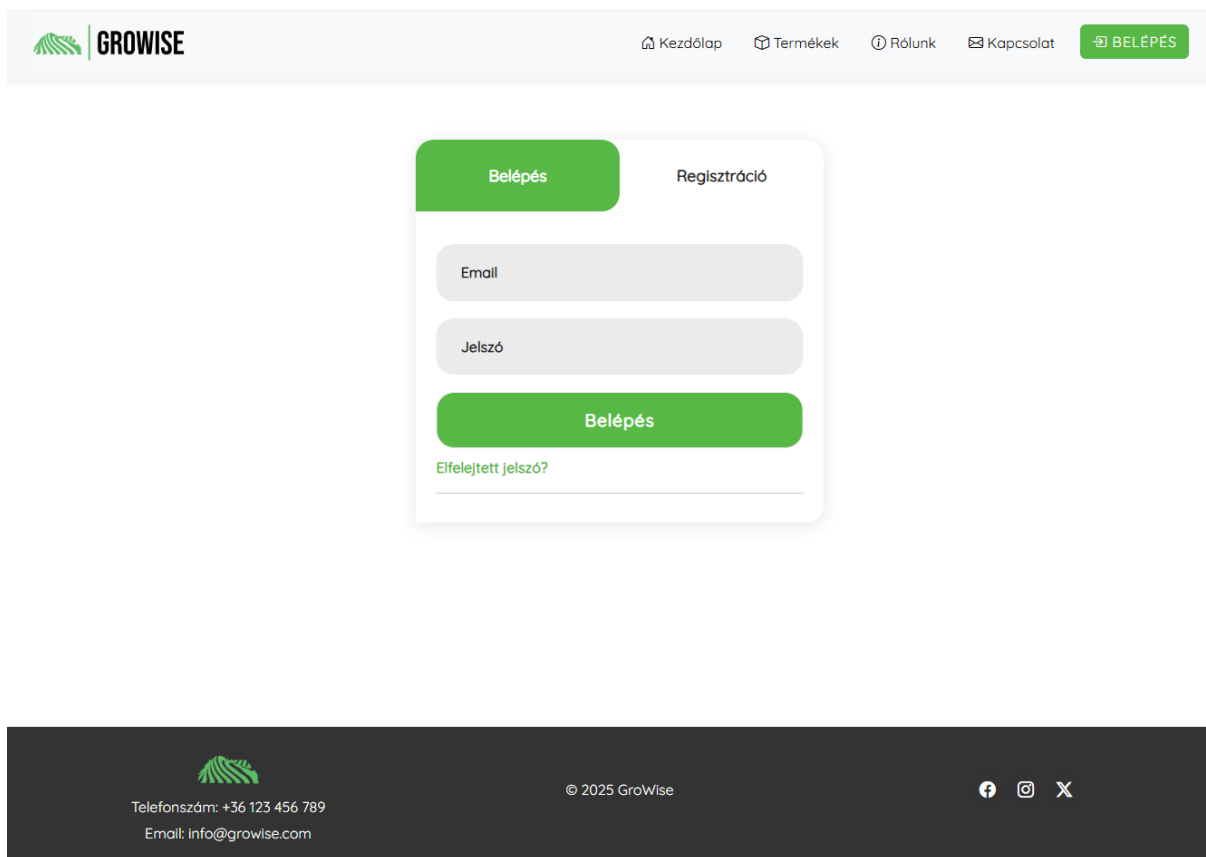
5.6. Frontend megvalósítása

Az alkalmazás frontendje a Django sablonrendszerére épül, amely dinamikusan generált HTML oldalak készítését teszi lehetővé Python kódból. A központi alapját egy `base.html` nevű sablon képezi, amely örökléses mechanizmussal szolgál az összes többi aloldal számára. A sablon a modern webfejlesztési irányelveket követve Bootstrap 5 keretrendszerre és egyéni CSS/JavaScript fájlokra épül.

A frontend megvalósítása során HTML5 és CSS3 technológiákat alkalmaztam az oldalak alapvető szerkezetének és stílusának kialakításához. A modern, reszponzív megjelenést a Bootstrap 5.3 keretrendszer biztosítja, amely megkönnyíti az egységes és mobilbarát elrendezések létrehozását. Az ikonok egységes megjelenéséről a Bootstrap Icons könyvtár gondoskodik.

A projekt vizuális arculatát egyedi stíluslapok (pl. base.css, dashboard.css, login.css) határozzák meg, míg az interaktív elemek működését saját JavaScript fájlok (pl. base.js, login.js) vezérlik. A statikus erőforrások (képek, CSS, JS) kezelésére a Django beépített {% static %} sablon tagjait használok, így biztosítva, hogy a fájlok a megfelelő útvonalakon töltsődjenek be. A következő ábrák bemutatják az elkészült felületet.

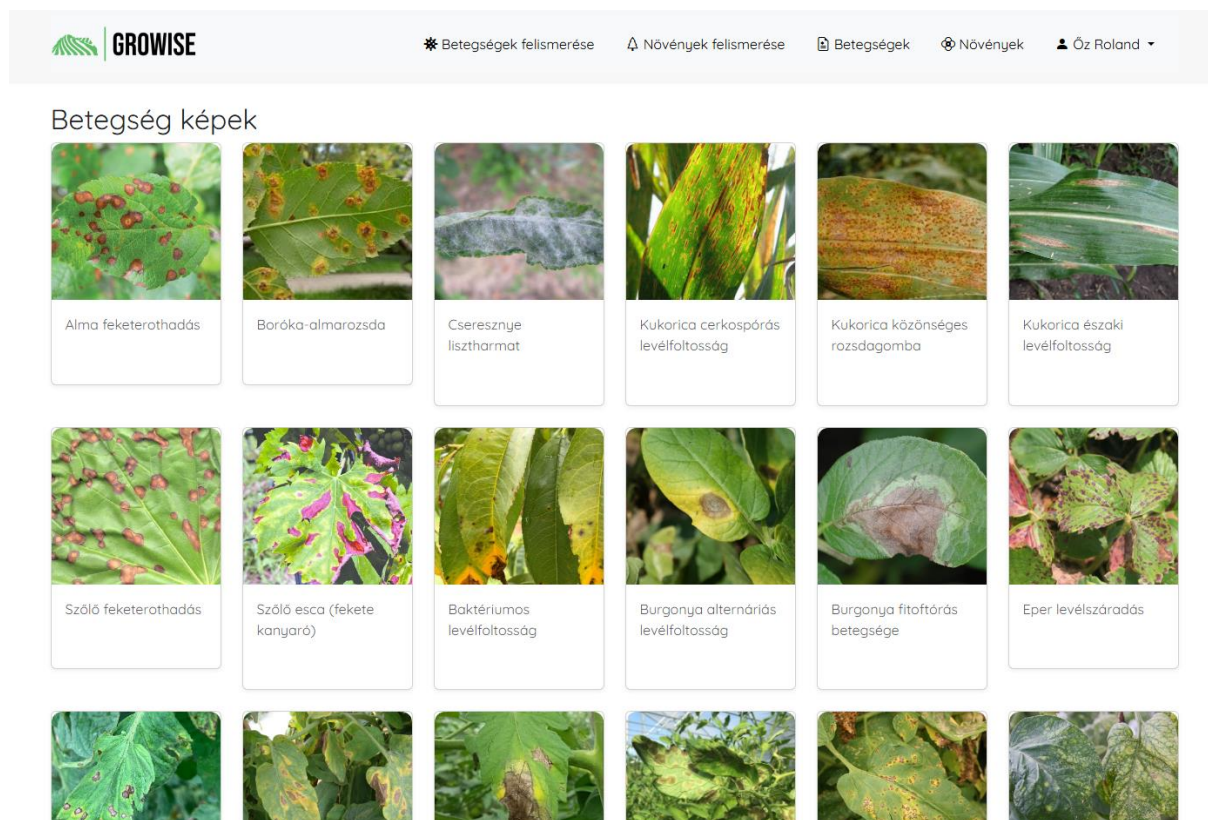
Az alábbi **5.4. ábra** mutatja a bejelentkező felületet, ugyanitt tud a felhasználó regisztrálni is.



The screenshot displays the GroWise website's login and registration interface. At the top, a navigation bar features the GroWise logo on the left and links for 'Kezdőlap', 'Termékek', 'Rólunk', and 'Kapcsolat' on the right, alongside a green 'BELEPÉS' button. The main content area is a white card with a green header containing 'Belépés' and 'Regisztráció' tabs. Below the tabs are input fields for 'Email' and 'Jelszó', followed by a green 'Belépés' button. A link for 'Elfelejtett jelszó?' is positioned below the login button. The footer is a dark grey bar containing the GroWise logo, contact information (phone number +36 123 456 789 and email info@growise.com), the copyright notice '© 2025 GroWise', and social media icons for Facebook, Instagram, and Twitter.


5.4. ábra Bejelentkezési felület

Az alábbi **5.5. ábra** mutatja a betegségek listáját, rájuk kattintva meglehet tekinteni az adott betegség adatait, kezelését és tünetét.



5.5. ábra Betegségek listája felület

Az alábbi **5.6. ábra** mutatja a betegségfelismerés felületet, amit a bejelentkezett felhasználók érnek el.



* Betegségek felismerése
Növények felismerése
Betegségek
Növények
Öz Roland


Kép feltöltése

Fájlok kiválasztása
Nincs fájl kiválasztva

FELTÖLTÉS

KIÉRTÉKEL
TÖRÖL

Eddigi feltöltött betegség képek




Dátum: 2025-05-07

Feldolgozva

Paradicsom - alternáriás levélfoltosság

Pontosság: 97,33%




Dátum: 2025-05-07

Feldolgozva

Paradicsom - alternáriás levélfoltosság

Pontosság: 94,81%




Dátum: 2025-05-07

Feldolgozva

Paradicsom - alternáriás levélfoltosság

Pontosság: 99,95%




Dátum: 2025-05-07

Feldolgozva

Paradicsom - alternáriás levélfoltosság

Pontosság: 83,72%




Dátum: 2025-05-07

Feldolgozva

Paradicsom - alternáriás levélfoltosság

Pontosság: 85,29%




Dátum: 2025-05-07

Feldolgozva

Paradicsom - alternáriás levélfoltosság

Pontosság: 94,96%




Dátum: 2025-05-07

Feldolgozva

Paradicsom - alternáriás levélfoltosság

Pontosság: 97,33%




Dátum: 2025-05-07

Feldolgozva

Paradicsom - alternáriás levélfoltosság

Pontosság: 94,81%




Dátum: 2025-05-07

Feldolgozva

Paradicsom - alternáriás levélfoltosság

Pontosság: 99,95%




Dátum: 2025-05-07

Feldolgozva

Paradicsom - alternáriás levélfoltosság

Pontosság: 83,72%




Dátum: 2025-05-07

Feldolgozva

Paradicsom - alternáriás levélfoltosság

Pontosság: 85,29%



Dátum: 2025-05-07

Feldolgozva

Paradicsom - alternáriás levélfoltosság

Pontosság: 94,96%

5.6. ábra Betegségek felismerése felület

5.6.1. Don't Repeat Yourself

A sablonrendszer kulcsfontosságú elemei a `{% block %}` címkék, amelyek lehetővé teszik, hogy az egyes aloldalak a központi sablonba dinamikusan illesszék be saját tartalmukat, ezáltal elősegítve az újrafelhasználhatóságot és az átlátható kódstruktúrát.

A fejléc dinamikusan alkalmazkodik a felhasználó bejelentkezett állapotához. Bejelentkezett felhasználók számára elérhetőek a növény- és betegségfelismerés, valamint a növény- és betegségeírások oldalai. A jobb felső sarokban egy legördülő menü segítségével hozzáférhetők a felhasználói profil és a kijelentkezés lehetőségek. Ha a felhasználó nincs bejelentkezve, a navigáció korlátozott, és csak a promóciós szekciók (például Termékek, Rólunk, Kapcsolat) jelennek meg, valamint egy jól látható "Belépés" gomb. A navigációs sáv rezponzív, mobilon összecsukszható (navbar-toggler), a Bootstrap osztályainak köszönhetően pedig felhasználóbarát módon működik.

A `{% block content %}` blokk tartalmazza az aktuális oldal tartalmát, amit a különböző sablonok töltenek fel, mint például a főoldal, a betegségfelismerő felület, vagy a profiloldal. Ez biztosítja a DRY (Don't Repeat Yourself) elvet, mivel a közös struktúrák nem szükségesek minden oldalon külön-külön.

A lábléc három fő részből áll: a céglogó és elérhetőségek, a copyright információk és a közösségi média linkek. A stílus egységes, és a közösségi média ikonok a Bootstrap segítségével jelennek meg.

5.6.2. Üzenetek (Django messages framework)

A sablon tartalmazza a Django beépített messages rendszerének megjelenítését is. A hibák, figyelmeztetések vagy sikeres műveletek különböző színekkel ellátott Bootstrap "toast" komponenseken keresztül kerülnek megjelenítésre. A háttérszín dinamikusán változik az üzenet típusának megfelelően (success, error, warning stb.).

6. TESZTELÉS

A fejlesztési folyamat során a megfelelő tesztelés kulcsfontosságú szerepet játszott a rendszer megbízhatóságának biztosításában. A tesztelés során a Django beépített TestCase osztályait alkalmaztam, amelyek lehetővé tették a funkcionális tesztek írását az alkalmazás különböző részein. A projekt során összesen 59 tesztet készítettem, amelyek a rendszer minden funkcióját lefedték, beleértve a növény- és betegségfelismerést, a képek feltöltését és törlését, valamint az ezekkel kapcsolatos hibakezelést. A tesztelés során kiemelt figyelmet fordítottam arra, hogy minden alkalmazásnak legyen külön tests mappája, és minden egyes nézet és funkció külön tesztelve legyen.

6.1. Tesztelési eszközök

A tesztelési környezet kialakításához a következő fontos eszközöket használtam:

1. **Django TestCase osztályok:** A Django beépített tesztelési keretrendszere lehetővé tette az egyes funkciók és nézetek részletes tesztelését. Külön teszteltem a képfeltöltést, a betegség- és növényfelismerést, valamint a hibák kezelését.
2. **Coverage könyvtár:** A tesztelés során alkalmazott coverage eszköz segítségével biztosítottam, hogy a kód 100%-os lefedettséget biztosítson, ahogy ezt a **6.1. ábra** is mutatja. Ez a mérőszám lehetővé tette annak ellenőrzését, hogy minden kódrészlet és funkció tesztelve legyen, így minimalizálva a nem észlelt hibák kockázatát.

File ▲	statements	missing	excluded	branches	partial	coverage
apps\authentication\models.py	7	0	0	0	0	100%
apps\authentication\tokens.py	5	0	0	0	0	100%
apps\authentication\urls.py	6	0	0	0	0	100%
apps\authentication\views.py	133	0	0	22	0	100%
apps\core\handlers.py	7	0	0	0	0	100%
apps\core\models.py	6	0	0	0	0	100%
apps\core\urls.py	3	0	0	0	0	100%
apps\core\views.py	30	0	0	4	0	100%
apps\plant\models.py	23	0	0	0	0	100%
apps\plant\urls.py	3	0	0	0	0	100%
apps\plant\views.py	19	0	0	0	0	100%
apps\recognition\models.py	22	0	0	0	0	100%
apps\recognition\urls.py	3	0	0	0	0	100%
apps\recognition\views.py	114	0	0	30	0	100%
Total	381	0	0	56	0	100%

6.1. ábra Tesztelési lefedettség allományokra bontva

3. **Mocking és Patching:** A külső könyvtárakkal való integráció, mint például a requests API vagy a PyTorch modell, mockolva lett, hogy elkerüljem a valós API hívásoktól és modellektől való függőséget. Ezáltal gyorsabban futtathatóvá váltak a tesztek, és pontosan kontrollálható volt az, hogy a rendszer hogyan reagál különböző szituációkban.

4. Főbb tesztelt funkciók:

- Regisztráció
- Kijelentkezés
- Bejelentkezés
- Betegségfelismerés
- Növényfelismerés
- Profiladatok megtekintése
- TokenGenerátor hash érték létrehozása
- Jelszó Visszaállítás
- Felhasználási feltételek letöltése PDF-ben
- Képek törlése
- Kapcsolat felvétele weboldalról
- Jelszó módosítása
- Felhasználói fiók aktiválása
- AI Modell

5. **Hibakezelés tesztelése:** A hibák és kivételek megfelelő kezelése kulcsfontosságú volt, különösen az olyan kritikus funkciók esetében, mint a képfeldolgozás és az API kommunikáció. Mockoltam azokat a helyzeteket, ahol a rendszer hibát dobhatott, és ellenőriztem, hogy ezek a hibák megfelelően naplózódjanak, és a felhasználó számára érthető hibaüzenetek jelenjenek meg.

A tesztelés során több hibát is felfedeztem, amelyeket a tesztelési folyamat nélkül nehéz lett volna észrevenni. Például olyan esetekben, amikor az API nem válaszolt megfelelően, vagy ha a képfeldolgozás során hiba történt. A tesztelés során történő hibák felderítése segített finomítani a rendszert, és javítani a felhasználói élményt. Ezek a hibák a fejlesztés korai szakaszában történő felfedezése jelentősen csökkentette a későbbi hibák javítására fordított időt.

A Test-Driven Development (TDD) módszertan alkalmazása révén már a fejlesztés során előre definiáltam, hogy a különböző funkcióknak milyen elvárt eredményeket kell produkálniuk. Ennek köszönhetően a tesztek folyamatosan biztosították, hogy a rendszer minden egyes része megfelelően működjön, és minden új fejlesztés során biztos lehettem abban, hogy a meglévő funkciók nem sérültek. A TDD alkalmazása különösen hasznos volt, mivel elősegítette a kód minőségének folyamatos javítását, és a hibák gyors felismerését.

6.2. Github Actions

A GitHub Actions workflow-t használok a Django alkalmazás folyamatos integrációjának és automatikus tesztelésének megvalósítására. A cél az, hogy minden egyes kódváltoztatás után automatikusan végrehajtódjanak a tesztek, ezáltal biztosítva a kód minőségét. A workflow a main branch-re történt push vagy pull request eseményekre aktiválódik, és az alábbi lépéseket hajtja végre:

Elsőként a környezeti változókat, például a titkos adatokat és konfigurációkat, mint az adatbázis elérhetősége, email beállítások, és API kulcsok, a GitHub Secrets segítségével töltöm be, így biztosítva a szükséges információk biztonságos hozzáférhetőségét. Ezt követően a workflow egy PostgreSQL szolgáltatást indít Dockerben, majd telepíti a szükséges Python függőségeket

a requirements.txt fájlból. A coverage eszköz, amely nemcsak a tesztelés lefedettségét méri, hanem HTML alapú tesztlefedettségi jelentést is generál. Végül a tesztlefedettségi jelentést feltöltöm, hogy a tesztelési eredmények könnyen hozzáférhetők legyenek és elemezhetők, segítve a kód minőségének folyamatos nyomon követését.

Ez a GitHub Actions workflow biztosítja, hogy a projekt minden változtatása tesztelve és validálva legyen, így garantálva a megbízható működést és a kód minőségének fenntartását.

6.3. Főbb tesztek összefoglalása

Az alábbi **6.1. táblázat** a RecognitonView fontosabb teszteit mutatja be.

Teszt neve	Leírás	Elvart eredmény
test_recognition_post_view_upload_image	Kép feltöltési funkció helyes működése	HTTP 302 átirányítás sikeres feltöltés után
test_recognition_exception_handling	Hibakezelés kép feltöltés közben	Hiba megfelelő naplózása, hibaüzenet megjelenítése, HTTP 302 átirányítás

6.1. táblázat RecognitionViewsTests

Az alábbi **6.2. táblázat** a EvaulateDisease fontosabb teszteit mutatja be.

Teszt neve	Leírás	Elvart eredmény
test_evaluate_disease	A betegség felismerés funkció helyes működése	HTTP 302 átirányítás, Result objektum létrehozása a kiválasztott képhez
test_evaluate_disease_post_without_selected_images	Viselkedés kiválasztott képek nélkül	HTTP 302 átirányítás a recognition oldalra
test_evaluate_disease_skips_already_processed_images	Már feldolgozott képek kezelése	Már feldolgozott képek kihagyása, ne hozzon létre új Result objektumot
test_disease_recognition_exception_handling	Hibakezelés a felismerési folyamat közben	Hiba megfelelő naplózása, hibaüzenet megjelenítése, HTTP 302 átirányítás, kép státusz változatlan marad

--

6.2. táblázat EvaluateDiseaseViewTests

Az alábbi **6.3. táblázat** a DeleteImageView fontosabb teszteit mutatja be.

Teszt neve	Leírás	Elvárt eredmény
test_delete_images	Képek törlési funkciójának helyes működése	HTTP 302 átirányítás, kiválasztott kép és hozzá tartozó eredmény törlése
test_delete_image_file_does_not_exist	Kezelés, ha a fájl nem létezik	HTTP 302 átirányítás, adatbázis rekord törlése
test_delete_images_exception_handling	Hibakezelés törlés közben	Hiba megfelelő naplózása, hibaüzenet megjelenítése, HTTP 302 átirányítás

6.3. táblázat DeleteImageViewTests

Az alábbi **6.4. táblázat** a EvaulatePlantView fontosabb teszteit mutatja be.

Teszt neve	Leírás	Elvárt eredmény
test_successful_evaluation_creates_result	Sikeres növény felismerés	HTTP 302 átirányítás, Result objektum létrehozása, kép státusz frissítése
test_api_returns_no_results	API válasz eredmények nélkül	HTTP 302 átirányítás, kép státusz nem változik, nem jön létre Result objektum
test_evaluate_plant_api_failure	API hiba kezelése	Hiba megfelelő naplózása, hibaüzenet megjelenítése, HTTP 302 átirányítás

6.4. táblázat EvaluatePlantViewTests

Az alábbi **6.5. táblázat** a SignUpView fontosabb teszteit mutatja be.

Teszt neve	Leírás	Elvárt eredmény
test_successful_signup	Sikeres regisztráció folyamata	HTTP 302 átirányítás, felhasználó létrehozása az adatbázisban, megerősítő email küldése

test_signup_with_existing_email	Viselkedés már létező email címmel történő regisztráció esetén	HTTP 302 átirányítás, új felhasználó nem jön létre
test_signup_with_mismatched_passwords	Viselkedés nem egyező jelszavak esetén	HTTP 302 átirányítás, felhasználó nem jön létre
test_signup_with_weak_password	Viselkedés gyenge jelszó esetén	HTTP 302 átirányítás, felhasználó nem jön létre

6.5. táblázat SignupViewTests

Az alábbi **6.6. táblázat** a LoginView fontosabb teszteit mutatja be.

Teszt neve	Leírás	Elvárt eredmény
test_successful_login	Sikeres bejelentkezés folyamata	HTTP 302 átirányítás, felhasználó hitelesítése
test_login_with_incorrect_credentials	Viselkedés helytelen bejelentkezési adatok esetén	HTTP 302 átirányítás, felhasználó nincs hitelesítve
test_login_with_inactive_user	Viselkedés inaktív felhasználói fiókkal történő bejelentkezés esetén	HTTP 302 átirányítás, felhasználó nincs hitelesítve

6.6. táblázat LoginViewTests

Az alábbi **6.7. táblázat** a ProfileView fontosabb teszteit mutatja be.

Teszt neve	Leírás	Elvárt eredmény
test_profile_redirects_if_not_logged_in	A profil oldal megfelelően átirányítja-e a nem bejelentkezett felhasználókat	HTTP 302 átirányítás
test_profile_loads_if_logged_in	A profil oldal megfelelően	HTTP 200 állapotkód,

	betöltődik-e bejelentkezett felhasználóknak	felhasználói adatok megjelenítése
--	---------------------------------------------------	--------------------------------------

6.7. táblázat ProfileViewTests

Az alábbi **6.8. táblázat** a ChangePasswordView fontosabb teszteit mutatja be.

Teszt neve	Leírás	Elvárt eredmény
test_change_password_redirects_if_not_logged_in	A jelszóváltoztatási oldal megfelelően átirányítja-e a nem bejelentkezett felhasználókat	HTTP 302 átirányítás
test_successful_password_change	Sikeres jelszóváltoztatás folyamata	HTTP 302 átirányítás, jelszó módosítása az adatbázisban
test_change_password_invalid_data_shows_errors	Viselkedés érvénytelen jelszóváltoztatási adatok esetén	HTTP 200 állapotkód, hibaüzenetek megjelenítése

6.8. táblázat ChangePasswordViewTests

6.4. AI Modell értékelése

Az AI modell tesztelése során kiváló eredményeket értem el, amint azt az alábbi **6.9. táblázat** is szemléltet. A rendszer helyesen azonosította a 33 tesztképet.

Növény	Betegség	Tesztképek száma	Helyes azonosítások	Pontosság
Alma	Cédrus-almarozsda	4	4	100%
Alma	Varasodás	3	3	100%
Kukorica	Közönséges rozsdagomba	3	3	100%
Burgonya	Alternáriás levélfoltosság	5	5	100%
Burgonya	Egészséges	2	2	100%
Paradicsom	Alternáriás levélfoltosság	6	6	100%
Paradicsom	Egészséges	4	4	100%
Paradicsom	Sárga levélkunkorodás vírus	6	6	100%
Összesen		33	33	100%

6.9. táblázat Tesztképek szerinti predikciók

A teszt során a modell minden egyes képet helyesen osztályozott, ami rendkívül biztató eredmény. A tökéletes felismerési arány jól mutatja a betanítás során alkalmazott mély tanulási technikák hatékonyságát és a modell általánosító képességét. Különösen figyelemreméltó, hogy

a rendszer nemcsak a betegségek jelenlétét ismerte fel, hanem pontosan meghatározta a betegség típusát is, ami lényegesen összetettebb feladat.

6.4.1. Részletes eredmények betegségtípusonként

Alma betegségei: Az alma esetében két különböző betegséget teszteltem: a cédrus-almarozsdát és a varasodást. Mindkét betegség esetében a modell 100%-os pontossággal működött. A cédrus-almarozsda jellegzetes narancsszínű foltjai és a varasodás barna, érdes felületű elváltozásai egyaránt egyértelműen felismerhetők voltak a rendszer számára.

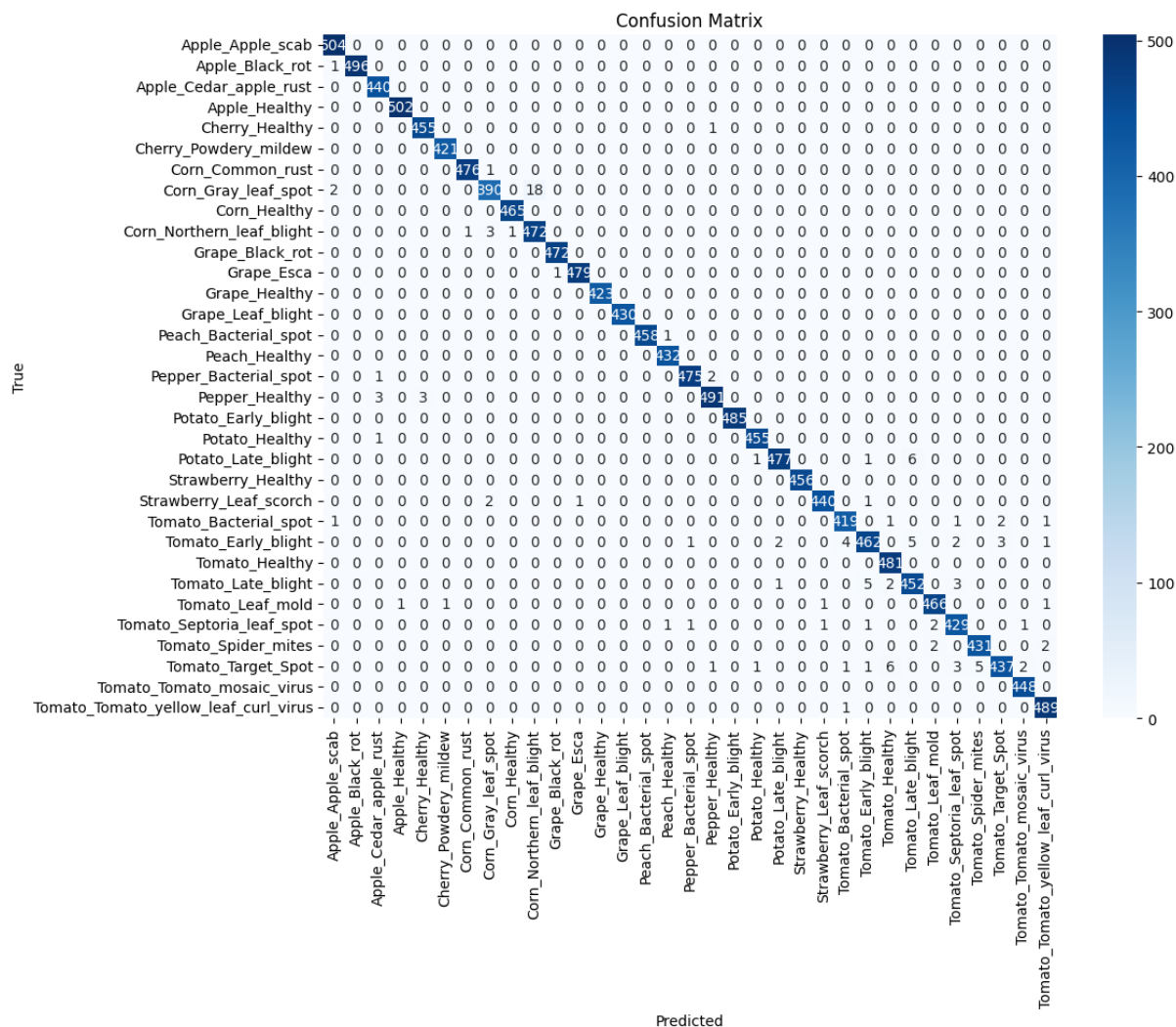
Kukorica betegségei: A kukorica esetében a közönséges rozsdagomba fertőzését vizsgáltam. A modell mind a három tesztképen helyesen azonosította a betegséget, annak ellenére, hogy a rozsdá tünetei a levelek különböző részein jelentkeztek és eltérő fejlődési szakaszban voltak láthatók.

Burgonya betegségei: A burgonya alternáriás levélfoltosságának felismerésében is tökéletes eredményt mutatott a rendszer. Az öt különböző tesztkép mindegyikén helyesen diagnosztizálta a betegséget, függetlenül a foltok méretétől és a fertőzés előrehaladottságától. Emellett az egészséges burgonyaleveleket is 100%-os biztonsággal azonosította.

Paradicsom betegségei: A paradicsom esetében három különböző állapotot teszteltem: az egészséges növényt, az alternáriás levélfoltosságot és a sárga levélkunkorodás vírust. Mindhárom kategóriában hibátlan felismerést tapasztaltam. Különösen jelentős eredmény, hogy a rendszer képes volt megkülönböztetni az alternáriás levélfoltosságot a paradicsomnál és a burgonyánál, annak ellenére, hogy hasonló tüneteket okoznak, ami a modell magas szintű diszkriminációs képességét bizonyítja.

6.4.2. Konfúziós mátrix

A **6.2. ábra** mutatja a konfúziós mátrixot, ami a modell teljesítményének egyik legfontosabb vizualizációs eszköze felügyelt gépi tanulási problémáknál.

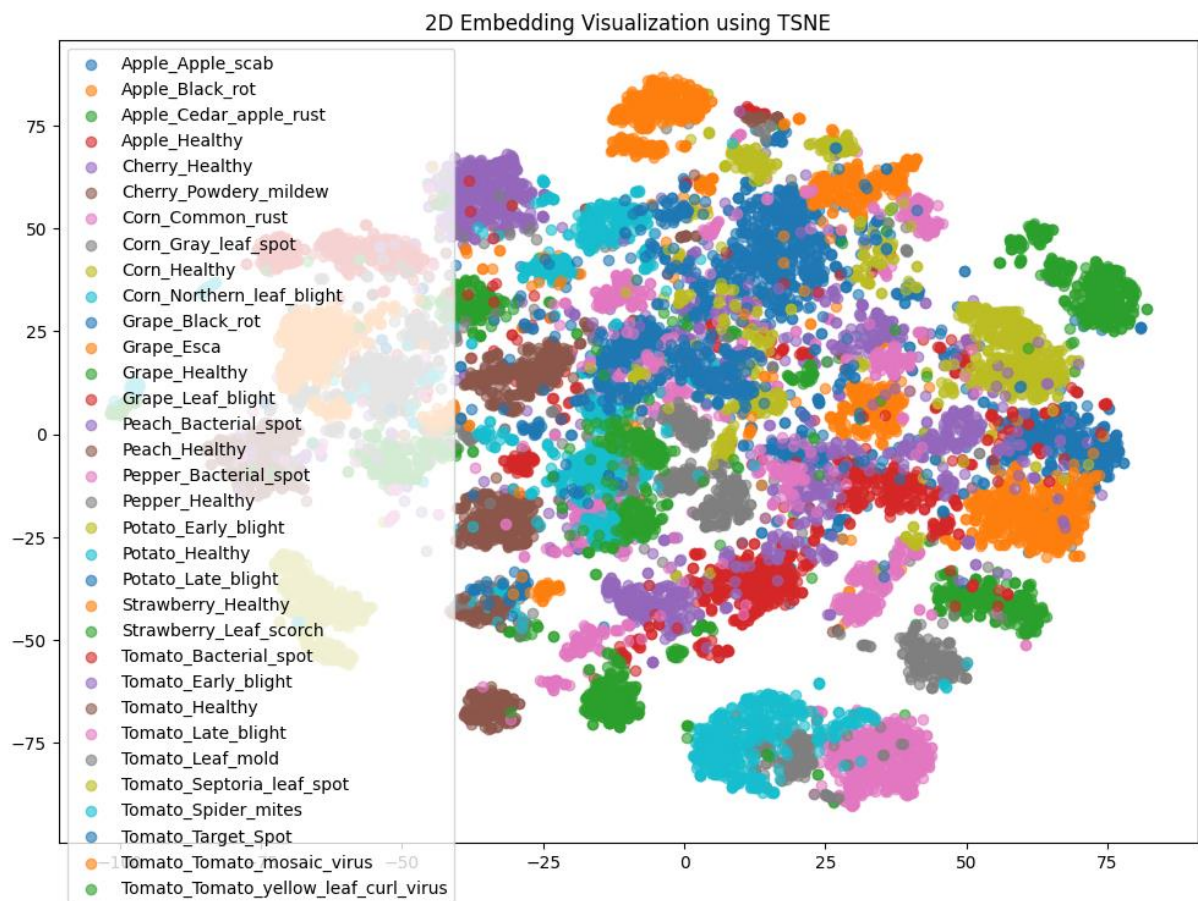


6.2. ábra Konfúziós mátrix

A mátrix sorai a valós osztályokat (tényleges betegségeket), míg az oszlopai a prediktált osztályokat (a modell által meghatározott betegségeket) jelölik. A mátrix átlójában lévő értékek a helyes osztályozások számát (igaz pozitív esetek) mutatják, míg az átlón kívüli elemek a tévesztéseket reprezentálják. Az átló magas értékei és az átlón kívüli alacsony értékek jó modell teljesítményt jeleznek.

6.4.3. 2D Embedding Vizualizáció

A t-SNE vizualizáció, amit a **6.3. ábra** szemléltet elsődleges célja, hogy megmutassa, hogyan csoportosulnak a különböző növénybetegségek példányai a feature térben. Ideális esetben az azonos osztályba tartozó minták közel helyezkednek el egymáshoz, míg a különböző osztályok elkülönülnek. Ez a vizualizáció tehát betekintést nyújt abba, hogy a modell hogyan "látja" és különbözteti meg az egyes betegségosztályokat.



6.3. ábra 2D Embedding Vizualizáció

7. ÉRTÉKELÉS

Az értékelés során a fejlesztett rendszer teljesítményét és a kitűzött célokhoz való illeszkedését vizsgálom. A megvalósított funkciók és a hozzájuk tartozó tesztek lefuttatásával megerősítettem, hogy az alkalmazás az elvárásoknak megfelelően működik. Miután lefejlesztettem az alkalmazás minden szükséges funkcióját, mindegyiket különböző szempontok szerint teszteltem. A tesztek során biztosítottam, hogy a rendszer minden esetben a kívánt eredményt adja.

A fejlesztés során külön figyelmet fordítottam a kód modularitására és átláthatóságára is. A Django keretrendszer előnyeit kihasználva strukturált, jól elkülöníthető komponensekből álló rendszert hoztam létre, amely támogatja a későbbi bővíthetőséget és karbantarthatóságot. A különböző alkalmazások világosan elkülönülnek, mégis egységes felhasználói élményt nyújtanak.

Az AI modell integrációja során különösen fontos szempont volt a megfelelő adatkezelés, valamint a felhasználók által feltöltött képek biztonságos kezelése. A képek alapján történő osztályozás során a rendszer megbízható predikciókat adott, amit a fájlnevek alapján elvárt eredményekkel való összevetés is igazolt. A felismerési eredmények valós idejűek és felhasználóbarát módon jelennek meg, ezzel is támogatva a könnyű kezelhetőséget.

A rendszer validálása során nemcsak a funkcionális helyességet, hanem a hibakezelési mechanizmusokat is alaposan teszteltem. A tesztek során szándékosan előidézett kivételhelyzetek esetén a rendszer megfelelően reagált, és nem fordult elő alkalmazásleállás vagy nem várt működés.

A felhasználói élmény javítása érdekében modern, reszponzív felületet alakítottam ki, amely tartalmaz egy dinamikus menürendszert is. Ez alkalmazkodik a felhasználó jogosultságaihoz és bejelentkezési állapotához, ezzel is növelve az alkalmazás használhatóságát és átláthatóságát.

Az automatikus tesztelés segített a folyamatos validálásban, és a GitHub Actions workflow megbízhatóan biztosította a kód minőségének fenntartását. A tesztlefedettség javítása érdekében a workflow-ban a legfontosabb funkciókat és hibakezeléseket is lefedtem, így biztosítva, hogy a rendszer robusztus és hibamentes legyen.

Az alkalmazás tehát jól teljesít az elvárt funkciók terén, de a jövőben még lehetőség van a rendszer további finomhangolására. A betegségfelismerés pontossága, valamint a felhasználói felület egyszerűsítése még tovább javítható, hogy még szélesebb körben és könnyebben használható legyen.

Összességében a rendszer sikeresen megfelelt az elvárásoknak, és képes az alkalmazás összes funkcióját zökkenőmentesen végrehajtani.

8. TOVÁBBFEJLESZTÉSI LEHETŐSÉGEK

A projekt során számos hasznos funkció és technológia került implementálásra, de van még néhány potenciális fejlesztési lehetőség, amelyek hozzájárulhatnak a rendszer további javításához és bővítéséhez. Ezek a lehetőségek nemcsak a rendszer funkcionalitásának növelésére, hanem a felhasználói élmény javítására is irányulnak. Az alábbiakban néhány konkrét fejlesztési irányt említek:

1. **Bővített diagnosztikai funkciók:** A rendszer jelenleg a betegségek felismerésére összpontosít, de a diagnosztikai folyamat továbbfejleszthető lenne olyan funkciókkal, amelyek nemcsak a betegségeket, hanem a növények egyéb problémáit (például tápanyaghiányok, kártevők) is felismerik. Ehhez további képosztályozó modellek beépítése szükséges, amelyek képesek kezelni ezeket az új problémákat.
2. **Interaktív oktató modulok:** A felhasználók számára hasznos lehetne egy oktató modul, amely bemutatja a növények gondozásának alapjait és a leggyakoribb betegségek megelőzésére vonatkozó ajánlásokat. Ez különösen hasznos lenne kezdő kertészek és gazdák számára, akik szeretnék jobban megérteni a betegségek megelőzését és kezelését.
3. **Mobilalkalmazás fejlesztése:** Bár a jelenlegi rendszer webalapú, hasznos lenne egy mobilalkalmazás kifejlesztése is, amely lehetővé tenné a felhasználók számára, hogy bárhol és bármikor könnyen hozzáférjenek a növényfelismerési és betegségdiagnosztikai funkciókhoz. A mobil alkalmazás további előnye, hogy a képek közvetlenül az okostelefonról tölthetők fel, így gyorsabbá válhat a folyamat.
4. **Valós idejű adatgyűjtés és visszajelzés:** A rendszer képes lenne valós idejű visszajelzést adni a felhasználóknak a növények állapotáról és a megfelelő kezelési javaslatokról. Az automatizált adatgyűjtés segítene nyomon követni a növények fejlődését, valamint a kezelésükhöz szükséges módosításokat, például időpontokat a tápanyagok vagy növényvédő szerek alkalmazására.
5. **Automatizált kamerarendszer üvegházakban:** Egy jövőbeli fejlesztési irány lehetne egy automatizált kamerarendszer kiépítése üvegházakban vagy beltéri termesztő rendszerekben, amely folyamatosan figyeli a növényeket, észleli a betegségek korai jeleit, és automatikusan riasztást küld vagy elemzést készít. A képek alapján a rendszer előfeldolgozást is végezhetne, majd gépi tanulási modellek segítségével diagnosztizálná a problémákat – emberi beavatkozás nélkül.

9. ÖSSZEGZÉS

A dolgozatom célja a mesterséges intelligencia alapú megoldások alkalmazása volt a növénybetegségek azonosításában és kezelésében, különös figyelmet fordítva a képfelismerő modellek használatára a betegségdetektálásban. A projekt során különböző AI modelleket és technológiákat alkalmaztam, hogy fejlesszem a rendszert, amely képes betegségek felismerésére, valamint a kapcsolódó információk kezelésére. Az alkalmazás Django keretrendszerben készült, és Python-alapú képfelismerést használva biztosítja a pontos és gyors eredményeket.

A dolgozatban kifejtett részletes elemzések és a kód megvalósítása után sikerült elérni a célokat: a rendszer képes felismerni a növényeket és a betegségeket, kezelni a felhasználói adatokat, valamint folyamatos integrációval biztosítani a megbízhatóságot és a kód minőségét. Az alkalmazott AI modellek és a folyamatos tesztelés révén a rendszer stabil működése biztosított, és az automatikus tesztelési megoldás segítette a rendszer hibamentes működésének ellenőrzését.

A dolgozat során számos új kihívással és problémával találkoztam, melyek megoldása során rengeteget tanultam a mesterséges intelligenciáról, a képfeldolgozásról és a szoftverfejlesztésről. A rendszer fejlesztése és tesztelése során az egyes funkciók és az automatikus tesztelés megoldása egy olyan stabil rendszert eredményezett, amely az elvárásoknak megfelelően működik, és képes a jövőbeli fejlesztésekhez is alapot adni.

Ezúton szeretnék köszönetet mondani a témavezetőmnek a folyamatos támogatásért és iránymutatásért. A dolgozat elkészítése egy remek tanulási folyamat volt, amely során sikerült elsajátítanom a mesterséges intelligencia alkalmazásának gyakorlati oldalát, és hozzájárultam a növénybetegségek kezelésének modernizálásához, remélem, hogy a kutatásom hozzájárulhat a fenntartható mezőgazdaság és a digitális technológia fejlődéséhez.

10. SUMMARY

The goal of my thesis was to apply artificial intelligence-based solutions for the identification and treatment of plant diseases, with particular emphasis on the use of image recognition models for disease detection. During the project, I applied various AI models and technologies to develop a system capable of recognizing diseases, as well as managing related information. The application was built using the Django framework, and by utilizing Python-based image recognition, it ensures accurate and fast results.

After conducting detailed analyses and implementing the code discussed in the thesis, I was able to achieve the objectives: the system is capable of recognizing plants and diseases, managing user data, and ensuring reliability and code quality through continuous integration. The AI models used and the ongoing testing ensured the stable operation of the system, and the automated testing solution facilitated the verification of the system's flawless functionality.

Throughout the thesis, I encountered several new challenges and problems, the solutions to which allowed me to learn a great deal about artificial intelligence, image processing, and software development. The development and testing of the system, along with the solutions to various functions and automated testing, resulted in a stable system that operates according to expectations and provides a foundation for future developments.

I would like to express my gratitude to my supervisor for their continuous support and guidance. Preparing this thesis has been a great learning experience, during which I have mastered the practical aspects of applying artificial intelligence, and contributed to modernizing the treatment of plant diseases. I hope that my research will contribute to the development of sustainable agriculture and digital technologies.

IRODALOMJEGYZÉK

- [1] S. Savary, L. Willocquet, S. J. Pethybridge, P. Esker, N. McRoberts és A. Nelson, „The global burden of pathogens and pests on major food crops,” pp. 430-439, 2019.
- [2] Food and Agriculture Organization of the United Nations, „Saving plants, saving lives: The International Plant Protection Convention turns 70,” 2021.
- [3] R. N. Strange és P. R. Scott, „Plant disease: a threat to global food security.,” pp. 83-116, 2015.
- [4] D. P. Bebber, M. A. Ramotowski és S. J. Gurr, „Crop pests and pathogens move polewards in a warming world.,” pp. 177-183, 2023.
- [5] M. C. Fisher, S. J. Gurr, C. A. Cuomo, D. S. Blehert, H. Jin, E. H. Stukenbrock és L. E. Cowen, „Threats posed by the fungal kingdom to humans, wildlife, and agriculture,” 2020.
- [6] Y. Wang, X. Cheng, Q. Shan, Y. Zhang, J. Liu, C. Gao és J. L. Qiu, „Simultaneous editing of three homoeoalleles in hexaploid bread wheat confers heritable resistance to powdery mildew,” pp. 947-951, 2021.
- [7] R. P. Singh, P. K. Singh, J. Rutkoski, D. P. Hodson, X. He, L. N. Jørgensen és J. Huerta-Espino, „Disease impact on wheat yield potential and prospects of genetic control,” pp. 303-322, 2022.
- [8] I. S. Hofgaard, H. U. Aamot, T. Torp, M. Jestoi, V. M. Lattanzio, S. S. Klemsdal és G. Brodal, „Associations between Fusarium species and mycotoxins in oats and spring wheat from farmers' fields in Norway over a six-year period,” pp. 365-378, 2019.
- [9] Y. Fillinger és S. Elad, „Botrytis—the fungus, the pathogen and its management in agricultural systems,” 2016.
- [10] Y. Born, L. Fieseler, V. Thöny, N. Leimer, B. Duffy és M. J. Loessner, „Engineering of Bacteriophages Y2::dpoL1-C and Y2::luxAB for Efficient Control and Rapid Detection of the Fire Blight Pathogen,” 2021.
- [11] N. Yuliar és K. Toyota, „Recent trends in control methods for bacterial wilt diseases caused by Ralstonia solanacearum,” pp. 1-11, 2018.
- [12] A. V. Karasev és S. M. Gray, „Continuous and emerging challenges of Potato virus Y in potato,” pp. 571-586, 2019.
- [13] I. M. Hanssen, M. Lapidot és B. P. Thomma, „Emerging viral diseases of tomato crops,” pp. 539-548, 2021.
- [14] M. G. Redinbaugh és L. R. Stewart, „Maize lethal necrosis: an emerging, synergistic viral disease,” pp. 301-322, 2018.
- [15] G. Belli, P. A. Bianco és M. Conti, „Flavescence dorée in Italy: An overview of past and current management,” pp. 233-239, 2018.
- [16] T. Cieślińska és Smolarek, „Detection and molecular characterization of 'Candidatus Phytoplasma mali' in apple trees in Poland,” p. 1606, 2020.
- [17] A. J. Haverkort, P. M. Boonekamp, R. Hutten, E. Jacobsen, L. A. P. Lotz, G. J. T. Kessel, J. H. Vossen és R. G. F. Visser, „Durable late blight resistance in potato through dynamic varieties obtained by cisgenesis,” pp. 225-253, 2022.
- [18] P. Jaarsveld, M. Faber, I. van Heerden, F. Wenhold, W. J. van Rensburg és W. van Averbek, „Nutrient content of eight African leafy vegetables and their potential contribution to dietary reference intakes,” pp. 77-84, 2021.

- [19] I. Pertot, T. Caffi, V. Rossi, L. Mugnai, C. Hoffmann, M. S. Grando és Y. Elad, „A critical review of plant protection tools for reducing pesticide use on grapevine and new perspectives for the implementation of IPM in viticulture,” pp. 70-84, 2021.
- [20] J. Kos, J. Levic, O. Duragic, B. Kokic és I. Miladinovic, „Occurrence and estimation of aflatoxin M1 exposure in milk in Serbia,” pp. 41-46, 2020.
- [21] Eurostat, „Agri-environmental indicator - consumption of pesticides,” 2021.
- [22] USDA, „Agricultural Resource Management Survey (ARMS),” 2020.
- [23] M. Barzman, J. R. Lamichhane, K. Booij, P. Boonekamp, N. Desneux, L. Huber és A. Messéan, „Research and development priorities in the implementation of the European Union's integrated pest management principles,” pp. 14-23, 2020.
- [24] M. Ghislain, A. A. Byarugaba, E. Magembe, A. Njoroge, C. Rivera, M. L. Román és J. F. Kreuze, „Stacking three late blight resistance genes from wild species directly into African highland potato varieties confers complete field resistance to local blight races,” pp. 1119-1129, 2019.
- [25] A. K. Mahlein, M. T. Kuska, J. Behmann, G. Polder és A. Walter, „Hyperspectral sensors and imaging technologies in phytopathology,” pp. 535-558, 2019.
- [26] S. P. Mohanty, D. P. Hughes és M. Salathé, „Using deep learning for image-based plant disease detection,” p. 1419, 2022.
- [27] S. Khanna és A. Kaur, „Internet of Things (IoT), applications and challenges: A comprehensive review. Wireless Personal Communications,” pp. 1687-1762, 2019.
- [28] „Plantix,” [Online]. Available: <https://plantix.net/en/>. [Hozzáférés dátuma: 12 05 2024].
- [29] „Plant.id,” [Online]. Available: <https://plant.id/>. [Hozzáférés dátuma: 12 05 2024].
- [30] „Blossom,” [Online]. Available: <https://blossomplant.com/>. [Hozzáférés dátuma: 12 05 2024].
- [31] „Microsoft,” [Online]. Available: <https://dotnet.microsoft.com/en-us/apps/aspnet>. [Hozzáférés dátuma: 12 05 2024].
- [32] „IBM,” [Online]. Available: <https://www.ibm.com/think/topics/django>. [Hozzáférés dátuma: 12 05 2024].
- [33] „Scaler,” [Online]. Available: <https://www.scaler.com/topics/django/django-architecture/>. [Hozzáférés dátuma: 12 05 2024].
- [34] „OpenSource,” [Online]. Available: <https://opensource.com/article/17/11/django-orm>. [Hozzáférés dátuma: 12 05 2024].
- [35] „IBM,” [Online]. Available: <https://www.ibm.com/think/topics/postgresql>. [Hozzáférés dátuma: 12 05 2024].
- [36] M. H. Salathé, „Using deep learning for image-based plant disease detection,” 2016.
- [37] D. Singh, N. Jain, P. Jain, P. Kayal, S. Kumawat és N. Batra, „PlantDoc: A Dataset for Visual Plant Disease Detection,” 2020.
- [38] „Kaggle,” [Online]. Available: <https://www.kaggle.com/datasets/vipooooool/new-plant-diseases-dataset>. [Hozzáférés dátuma: 12 05 2024].
- [39] „Medium,” [Online]. Available: <https://sumitkrsharma-ai.medium.com/preprocessing-data-for-object-detection-in-computer-vision-e659c18b2dc3>. [Hozzáférés dátuma: 12 05 2024].

- [40] „AnalyticsVidhya,” [Online]. Available: <https://www.analyticsvidhya.com/blog/2021/05/convolutional-neural-networks-cnn/>. [Hozzáférés dátuma: 12 05 2024].
- [41] „YouTube,” [Online]. Available: <https://www.youtube.com/watch?v=Us0xAZUiYTE>. [Hozzáférés dátuma: 12 05 2024].
- [42] T. Bulent, E. Elhoucine és E. Recep, „Convolutional Neural Networks in Detection of Plant Leaf Diseases: A Review”.
- [43] „TechTarget,” [Online]. Available: <https://www.techtarget.com/searchenterpriseai/definition/PyTorch>. [Hozzáférés dátuma: 12 05 2024].
- [44] „NVIDIA,” [Online]. Available: <https://www.nvidia.com/en-us/glossary/tensorflow/>. [Hozzáférés dátuma: 12 05 2024].
- [45] „Microsoft,” [Online]. Available: <https://learn.microsoft.com/en-us/devops/develop/git/what-is-git>. [Hozzáférés dátuma: 12 05 2024].
- [46] „YouTube,” [Online]. Available: https://www.youtube.com/watch?v=e9lnsKot_SQ... [Hozzáférés dátuma: 12 05 2024].
- [47] „Towards Datascience,” [Online]. Available: <https://towardsdatascience.com/resnets-why-do-they-perform-better-than-classic-convnets-conceptual-analysis-6a9c82e06e53/#:~:text=ResNet%20Layers,layers%20remains%20the%20same%20%E2%80%94%204>. [Hozzáférés dátuma: 05 05 2025].

ÁBRAJEGYZÉK

2.1. ábra Plantix andorid felülete [28].....	17
2.2. ábra Plant.id felülete.....	19
2.3. ábra Blossom IOS alkalmazás detektált betegség felülete	20
2.4. ábra Konvolúciós neurális hálózat működése [41]	27
2.5. ábra A git működése [46]	31
4.1. ábra Django Architektúra	35
4.2. ábra Adatbázis ER modell.....	38
4.3. ábra Regisztráció és bejelentkezés Activity Diagram	39
4.4. ábra Elfelejtett jelszó Activity Diagram.....	40
4.5. ábra Képfeltöltés és felismerés Sequence Diagram	41
4.6. ábra Jelszó módosítás Activity Diagram	42
5.1. ábra Egy réteg a ResNet-ben [47]	46
5.2. ábra Modell architektúra	47
5.3. ábra Pontosság értékének változása az epochok számának függvényében.....	49
5.4. ábra Bejelentkezési felület	52
5.5. ábra Betegségek listája felület.....	53
5.6. ábra Betegségek felismerése felület	54
6.1. ábra Tesztelési lefedettség allományokra bontva.....	56
6.2. ábra Konfúziós mátrix	63
6.3. ábra 2D Embedding Vizualizáció	64

TÁBLAJEGYZÉK

2.1. táblázat Hasonló rendszerek összehasonlítása	21
4.1. táblázat Authentication alkalmazás főbb végpontjai.....	36
4.2. táblázat Recognition alkalmazás főbb végpontjai.....	37
4.3. táblázat Fejlesztés tervezett menete lebontva hetekre	43
5.1. táblázat Osztályok eloszlása.....	46
6.1. táblázat RecognitionViewsTests	58
6.2. táblázat EvaluateDiseaseViewTests	59
6.3. táblázat DeleteImagesViewTests	59
6.4. táblázat EvaluatePlantViewTests	59
6.5. táblázat SignupViewTests	60
6.6. táblázat LoginViewTests	60
6.7. táblázat ProfileViewTests	61
6.8. táblázat ChangePasswordViewTests	61
6.9. táblázat Tesztképek szerinti predikciók	61