

ETDK-dolgozat

Gozner Máté

Fischer Mátyás-Zsigmond

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

Informatika II.: innovatív számítástechnikai termékek, alkalmazások szekció

Kolozsvár, 2024. május 15–18.

Parkolóhelyek foglaltságának meghatározása és adminisztrációja mesterséges intelligencia segítségével



Szerzők:

Gozner Máté

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

Fischer Mátyás-Zsigmond

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

Témavezetők:

dr. Simon Károly, egyetemi adjunktus

Babeş–Bolyai Tudományegyetem, Matematika és Informatika Kar

Zölde Attila, szoftverfejlesztő,

Codespring

Fazakas Lóránt-Tibor, szoftverfejlesztő,

Codespring





Parkolóhelyek foglaltságának meghatározása és adminisztrációja mesterséges intelligencia segítségével

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

Fischer Mátyás-Zsigmond, Babeş–Bolyai Tudományegyetem, Matematika és Informatika Kar, Informatika szak, III. év

dr. Simon Károly, egyetemi adjunktus
Babeş–Bolyai Tudományegyetem, Matematika és Informatika Kar
Zölde Attila, szoftverfejlesztő,
Codespring
Fazakas Lóránt-Tibor, szoftverfejlesztő,
Codespring

A nagy parkolóházakban komoly problémát jelenthet az autóvezetőknek a szabad helyek megtalálása, és ezt súlyosbítja a városok autóteltettsége. Ha repülőtér, nagy bevásárlóközpont vagy cégek parkolóházaira gondolunk, mindegyiknél van valamilyen megoldás a parkolók adminisztrációjára, viszont a legtöbb esetben ezeknek számos hiányosságuk van, például csak a szabad helyek számát mutatják, a helyüket nem, illetve nem segítik a felhasználókat a helyek megtalálásában. Azokban az esetekben is, amikor a helyek szabad állapotáról van információ, az költséges befektetések árán valósul meg, parkolóhelyenkénti szenzorok használata által.

A Parking Lot Visualizer projekt célja parkolóházak virtuális modellezése, és egy olyan platform kialakítása, illetve adminisztrációja, amely segíti a beérkező autós felhasználókat a szabad helyek megtalálásában, költséghatékony módszerrel. A mesterséges intelligencia által vezérelt alkalmazáslogika, a kamerák képeit felhasználva, meg tudja állapítani az egyes parkolóhelyek állapotát. Jó esetben az ehhez szükséges kamerarendszer nagy része már alaphoz rendelkezésre áll a parkolóházakban. A másik fontos eleme a projektnek a parkolóház adminisztrációja és vizualizációja, így segítve a parkolóház működtetőjét és a felhasználókat is. A parkolóház virtuális modelljének elemei a virtuális parkolóhelyek és az ezeket figyelő virtuális kamerák. Ezek elhelyezése a virtuális modellen belül GPS koordináták segítségével történik, és a kamerák esetében egy látószög meghatározása is szükséges.

A parkolóház virtuális modelljét megfelelően konfigurálva juttatjuk el a felhasználókhoz, egy mobil applikáción belül megjelenített térképre kirajzolva a parkolóhelyeket.

Tartalomjegyzék

Bevezető	1
1. A Parking Lot Visualizer projekt	3
1.1. Aktuális megoldások	3
1.2. Funkcionalitások	3
1.2.1. Adminisztrátori funkcionalitások	3
1.2.2. Felhasználói funkcionalitások	5
2. A mesterséges intelligencia alkalmazása	6
2.1. Miért a YOLO?	6
2.2. A YOLO alkalmazása a rendszerben	7
3. Virtuális parkoló modell	9
3.1. A virtuális világ és a valóság viszonya	9
3.2. Foglaltságot meghatározó logika	10
4. Az alkalmazás felépítése	12
4.1. Admin felhasználói felület	12
4.1.1. Felhasznált technológiák	12
4.1.2. Belső struktúra	13
4.2. Helyi parkolóház szerver	14
4.2.1. Felhasznált technológiák	14
4.2.2. Belső struktúra	14
4.2.3. Adatmodell	15
4.2.4. API	16
4.2.5. Security	16
4.3. Központi szerver	17
4.3.1. Felhasznált technológiák	17
4.3.2. Belső struktúra	17
4.3.3. Adatmodell	18
4.3.4. API	18
4.4. Mobil appikáció	19

4.4.1.	Felhasznált technológiák	19
4.4.2.	Belső struktúra	20
4.4.3.	Kedvencek tárolása	20
5.	Az alkalmazás működése	21
5.1.	Adminisztrátor	21
5.1.1.	Kamerák felvezetése	21
5.1.2.	Felülnézeti mód	22
5.1.3.	Kamera beállítások	23
5.2.	Felhasználó	25
	Következtetések és továbbfejlesztési lehetőségek	28

Bevezető

Minden autós találkozhatott már azzal a problémával, hogy milyen nehezen lehet parkolóhelyet találni, főképp a nagyobb városokban, ahol egyre többen járnak autóval. A megnövekedett forgalomban egyre gyakrabban fordul elő az a bosszantó helyzet, hogy sokszor csak több kör megtétele után lehet találni egy megüresedett helyet. Ilyenkor az idővesztés mellett feleslegesen használunk üzemanyagot is, ami miatt növekszik az autós költsége, a károsanyag kibocsátása, és az amúgy is telített forgalmat növeli új köreivel.

A jelenlegi parkolást segítő megoldások egyik legnagyobb hiányossága, hogy általában csak a parkolóhelyek telítettségéről képesek adatot szolgáltatni. Meg tudják mondani, hogy hány szabad hely van, de azt már nem tudja az autós, hogy a parkolóházon belül ezek pontosan hol vannak, ami nagy parkolóházak esetében szintén problémát jelent. Ezt a hiányosságot ugyan a szenzoros módszer kiküszöböli, de a szenzorok beszerelése és karbantartása is költséges.

A fentebb említett megoldások hiányosságait küszöböli ki a Parking Lot Visualizer projekt, mely pontos adatokkal tud szolgálni a parkolóhelyek állapotáról, anélkül, hogy nagy költségekkel járna a beüzemelése vagy a karbantartása. A rendszer mesterséges intelligencia segítségével képes felismerni valós időben a járműveket egy kamera képén, majd egy 3D matematikai modell segítségével megállapítja, hogy a jármű foglal vagy sem parkolóhelyet. A rendszer fel tudja használni a biztonsági kamerák képeit is, így sok helyen az új kamerák beüzemelését is meg lehet spórolni általa.

A dolgozat négy fejezetből épül fel. Az első fejezet bemutatja a Parking Lot Visualizer projektet, felvázolja a piacon már létező, hasonló célú megoldásokat, majd bemutatja mind az adminisztrátor, mind a felhasználó számára nyújtott funkciókat. A második fejezet a mesterséges intelligencia alkalmazását részletezi, kitérve arra, hogy miért pont arra az algoritmusra esett a választás, amit jelenleg használunk. A harmadik fejezet bemutatja, hogy hogyan épül fel a virtuális modell, ami alapján a felvezetett parkolóhelyeket végül a kamerák képeire rávetíti a rendszer. A negyedik fejezetben a rendszer architektúrájáról lesz szó. Bemutatja a fő elemeit, részletesen, képekkel illusztrálva, valamint ezek egymás közti kapcsolatait. Az utolsó fejezetben a levont következtetések, valamint a továbbfejlesztési lehetőségek szerepelnek.

A projekt fejlesztése 2023 nyarán, a Codespring cégnél végzett szakmai gyakorlat során kezdődött. A nyári gyakorlat során elkészültek a projekt fő részei, a szakmai gyakorlat végére lett egy sikeres predikciót végző helyi parkolóház szerver, valamint elkészült a

felhasználóknak szánt mobil alkalmazás és a központi szerver. Így már használható állapotba került a prototípus.

A következő lépés a projekt történetében a 2023-2024-es tanév első félévében szereplő csoportos projekt tantárgy volt, mely során csatlakozott még a csapathoz további két diáktárs. Ennek keretein belül folytatódott tovább a munka: az adminisztrátori felület felhasználóbarátabbá tételével, a biztonság növelésével, valamint több hiba kijavításával, amitől a rendszer stabilabbá vált. Ugyancsak ebben a fázisban a csapat elérte az egyik legfontosabb mérföldkövet: a rendszer ki lett telepítve a Codespring szerverére és jelenleg is fut, elemezve a cég parkolóját.

Köszönet illeti témavezető tanárunkat Dr. Simon Károly egyetemi adjunktust, Zölde Attila és Fazakas Lóránt-Tibor szoftverfejlesztőket a mentorálásért és a szakmai segítségért, a Codespring céget a szakmai gyakorlat és az azt megelőző szakmai felkészítő biztosításáért, valamint a csoportos projekt során csatlakozott diáktársakat, Czumbil Márkot és Szabó Boldizsárt a hozzájárulásért.

1. A Parking Lot Visualizer projekt

A Parking Lot Visualizer projekt egy parkolást segítő alkalmazás, amely valós időben nyújt információt egy parkolóház állapotáról, megjelenítve a szabad és a foglalt parkolóhelyeket. Célja a parkolás könnyebbé tétele, illetve az, hogy üzemanyagot és időt spóroljon a felhasználónak.

1.1. Aktuális megoldások

Jelenleg a piacon levő parkolást segítő alkalmazások nem nyújtanak teljes körű megoldást a parkolni kívánó felhasználók számára.

Egyes parkolóházak csak a telítettségi szintről képesek információt nyújtani, ami, ha nagy parkolóházzal van szó, még okozhat fejfájást a sofőrnek, hiszen még meg kell találnia a parkolóházon belül a szabad helyet.

Ennél ugyan több információt képes nyújtani a szenzoros alternatíva, mely pontos információt tud adni az egyes parkolóhelyek állapotáról, de ez egy igencsak költséges beruházás, hiszen egyenként le kell helyezni a szenzorokat, amihez az aszfalt feltörése is szükséges lehet. A szenzorok karbantartása is okozhat plusz költséget.

Az említett szempontok figyelembe vételével a Parking Lot Visualizer egy költséghatékony alternatívát nyújthat a piacon már jelen levő alkalmazásokkal szemben.

1.2. Funkcionalitások

A fejezet bemutatja a projekt funkcionálisait, mind a parkolóház adminisztrátora, mind a felhasználó szemszögéből.

1.2.1. Adminisztrátori funkcionálisok

Egy parkolóház adminisztrátorának egy felhasználói felületen keresztül van lehetősége a lokálisan futó, parkolóhelyeket elemző szerver konfigurálására, így létrehozva a parkolóház virtuális modelljét.

Az adminisztrátornak lehetősége van új kamerát felvezetni a rendszerbe, tud kamerákat törölni, illetve azok pozícióját állítani, mind egy térképen, mind a virtuális modellnek a kamera képre való igazítása során. A virtuális modell beállításai a kamerák pozíciójának megváltoztatási lehetőségén túl, tartalmazzák a dőlésszögek beállítását, mind a függőleges,



1. ábra. Use case diagram.

mind a vízszintes tengely mentén, illetve a kamerák elforgatását. Amennyiben egy kamera képét elfordulva kapja meg a rendszer, ami gyengíti a mesterséges intelligencia becslési sikerességét, az adminisztrátor elforgathatja a felületen belül ezt a képet, így nem szükséges a kamerán fizikailag állítani.

Általában egy kamera több parkolóhelyet is képes megfigyelni. Ugyanakkor egy parkolóhelyet több kamera is figyelhet. Az adminisztrátornak lehetősége van kiválasztani, hogy egy kamera számára melyek legyenek a megfigyelt parkolóhelyek. Ez olyan esetben is fontos, ha egy objektum eltakar a képen bizonyos parkolóhelyeket.

A kamera beállításoknál hasznos a virtuális modell által reprezentált parkolóhelyek burkoló dobozának a megjelenítése. Ezeknek a mesterséges intelligencia által észlelt objektumok burkoló téglalapjaival történő összehasonlítása még finomabb beállítások lehetőségét nyújtja az adminisztrátornak. Ezeket a harmadik fejezet részletesebben kifejti, a 4. ábrával szemléltetve.

A kamerákhoz hasonlóan az adminisztrátor felvezethet parkolóhelyeket is a rendszerbe. A parkolóhelyek pontos pozícióját a térképen tudja beállítani. Itt egy beépített eszköz is segíti ebben, melynek feladata, hogy a kijelölt, parkolóhelyek széleit jelző pontokat egy vonalra igazítsa, ugyanis, ahogy a hétköznapiakban is megfigyelhető, a parkolóhelyek leggyakrabban

egymás mellett, egy vonalban vannak felfestve. A térkép nézetén a beállítási lehetőségek mellett a parkolóhelyek valós idejű állapota is megtekinthető.

1.2.2. Felhasználói funkcionálisok

A felhasználók a mobil applikáció segítségével tudják megtekinteni a parkolóhelyek aktuális állapotát.

Az alkalmazás megnyitása után egy lista fogadja őket, amely a parkolóházakat tartalmazza, megadva azok telítettségét. Kiválasztva egyet, megjelenik a parkolóhelyek listája a helyek állapotával. A listás nézetek mellett lehetőség van a térképen is megtekinteni a parkolóházakat, illetve a parkolóhelyeket, színekkel jelölve azok állapotát. Először csak a parkolóházak látszanak a térképen, a parkolóhelyek akkor válnak láthatóvá, ha belenagyítanak a térkép nézetbe. A térképen lehet a parkolóházak között keresni, így a felhasználók könnyebben megtalálhatják a keresett parkolóházat.

Ugyancsak a keresés egyszerűsítését és a felhasználók kényelmét hivatott segíteni a „Kedvencek” menüpont is. Ide lehet menteni a felhasználó által gyakran használt, vagy ideális helyen levő parkolókat, melyek állapotára gyakrabban kíváncsi, és mielőtt elkezdene más parkolóhelyeket keresni, először ezek állapotát szeretné megnézni.

2. A mesterséges intelligencia alkalmazása

A mesterséges intelligencia által vezérelt alkalmazáslogika felhasználja a parkolóházak kameráinak képeit az egyes parkolóhelyek állapotának rögzítésére és értelmezésére. Ennek eredményeként az autósok könnyen megtalálhatják a szabad parkolóhelyeket az applikáció térképén. A kamerák által rögzített adatok alapján az MI rendszer folyamatosan frissíti a rendelkezésre álló parkolóhelyek állapotát.

A mesterséges intelligencia alkalmazása nemcsak az autósok számára előnyös, hanem a parkolóházak üzemeltetői számára is. Az adminisztrációs felület lehetővé teszi számukra, hogy nyomon kövessék a parkolóház kihasználtságát és optimalizálják az infrastruktúrát a felhasználói igényeknek megfelelően. A virtuális modell segítségével könnyen megtekinthető a parkolóházban lévő parkolóhelyek eloszlása és kihasználtsága.

A mesterséges intelligencia alkalmazása révén a parkolóházak hatékonyabbá válhatnak a parkolóhelyek kihasználtságának maximalizálása és az autósok számára nyújtott kényelem növekedése révén. Ez a technológiai megoldás hozzájárulhat a városi közlekedés hatékonyabbá és zöldebbé tételéhez, csökkentve a parkolóhelykeresésből adódó forgalmi torlódásokat és a környezetszennyezést.

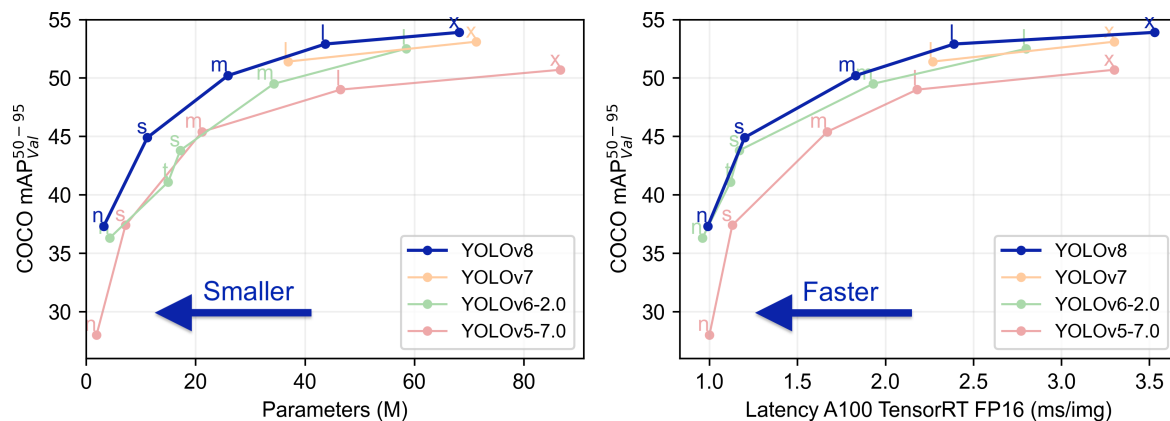
2.1. Miért a YOLO?

A YOLO [7] egy konvolúciós neurális hálózat (Convolutional Neural Network, CNN) alapú objektumdetektáló rendszer, amely a deep learning módszert alkalmazza. A deep learning egy olyan gépi tanulási módszer, amely neurális hálózatokat használ a bonyolult minták felismerésére és tanulására nagy adathalmazokból. A deep learning módszerek, mint például a konvolúciós neurális hálózatok használata, alkalmasak arra, hogy automatikusan és adaptívan tanuljanak a nagy dimenziójú adatokból, mint például a képek vagy a szövegek, anélkül, hogy manuálisan kellene kiválasztani a jellemzőket.

A YOLO egyetlen konvolúciós hálózatot használ, amely a teljes képet átfedi, és ezzel képes az objektumokat és azok helyét egyetlen lépésben azonosítani. Ez a módszer sokkal gyorsabb, mint a hagyományos képfeldolgozási módszerek, amelyek több lépésben végzik el az objektumok azonosítását és lokalizálását. Más módszerek gyakran először egy előfeldolgozási lépést végeznek. Ezt követi a klasszifikációs és lokalizációs lépések elvégzése, mielőtt végül az objektumokat azonosítanák és megjelenítenék. Ez a folyamat jelentősen bonyolultabb

és időigényesebb lehet, különösen nagy méretű parkolóházak esetén, ahol sok parkolóhely állapotát kell figyelni egyszerre.

A Parking Visualizer projekt konkrétan a YOLOv8 verzióját használja. Ez a verzió továbbfejlesztett hálózati architektúrával rendelkezik, amely még pontosabb és hatékonyabb objektumfelismerést tesz lehetővé, mint elődei. Az újításoknak köszönhetően javul az algoritmus általános teljesítménye és pontossága.



2. ábra. A YOLO modellek teljesítménye a COCO adathalmazon mért átlagos pontossági arány (mAP), valamint a modellek paramétereinek száma és a feldolgozási idő alapján. *Forrás: <https://github.com/ultralytics/ultralytics> Letöltés dátuma: 2024. április 23.*

Az 2. ábra segít megérteni, hogy a YOLO algoritmus különböző verziói hogyan javították a pontosságot és a sebességet, miközben kevesebb paramétert használnak, ami gyorsabb válaszidőt eredményez. Ez kritikus tényező a valós idejű alkalmazásokban, mint például a parkolóházakban történő objektumfelismerés, ahol a gyorsaság és a pontosság elengedhetetlen.

Bár más módszereket is kipróbáltunk, mint például a Faster R-CNN-t, ezek általában két lépésben határozzák meg az objektumokat: először az objektumokat lokalizálják, majd azonosítják. Ez a megközelítés nem előnyös a mi esetünkben, mivel gyors és hatékony megoldásra van szükségünk a valós idejű válaszadáshoz. A YOLO, bár nem sokkal pontosabb, sokkal gyorsabb választ tud adni, ami fontos a mi alkalmazásunk számára. [6]

2.2. A YOLO alkalmazása a rendszerben

A YOLO algoritmus a parkolóház kameráinak képeit használja fel, és képes azonosítani az autókat. Ezt az információt felhasználva a rendszer megállapítja, hogy a parkolóhelyek szabadok vagy foglaltak.

Az algoritmus először a kép minden pixelét átvizsgálja, és meghatározza azoknak a területeknek a helyét, amelyek valószínűleg objektumokat tartalmaznak. Ezután a hálózat megpróbálja azonosítani, hogy milyen objektumok találhatók ezeken a területeken. A mi esetünkben az autó objektumok a relevánsak.

Az algoritmus által előállított adatokat a rendszerünk feldolgozza, és frissíti a parkolóhelyek állapotát az alkalmazásban. Az adminisztrátorok ezután valós időben láthatják a szabad parkolóhelyeket az alkalmazás térképén.

3. Virtuális parkoló modell

A szabad helyek meghatározásához elengedhetetlen, hogy a rendszer tudja, hogy a kamerák képein hol vannak a parkolóhelyek, mivel ezekhez kell viszonyítsa a mesterséges intelligencia által talált objektumokat. Ehhez szükség van egy valóságot modellező virtuális parkolóház felépítésére.

3.1. A virtuális világ és a valóság viszonya

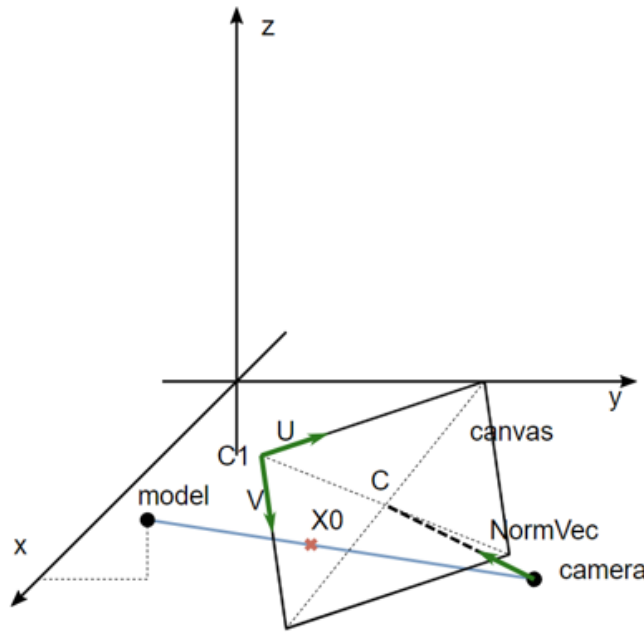
A virtuális modell a rendszer számára releváns fizikai objektumokat (kamerák, parkolóhelyek) képezi le feldolgozható adatokra.

A kamerák a 3 dimenziós világról alkotnak 2 dimenziós képfolyamot. A rendszernek ezeken a képeken kell felismernie az objektumokat, illetve ezek alapján kell megállapítania, hogy azok parkolóhelyeket foglalnak el vagy sem. Ehhez viszont tudnia kell, hogy hol vannak a parkolóhelyek határai a képen. A rendszerbe már fel lettek vezetve a kamerák, illetve a parkolóhelyek koordinátái, ugyanakkor ennyi adat még nem elég ahhoz, hogy a rendszer pontosan a valós parkolóhelyek határoló vonalaira vetítse a virtuális modellt. Szükséges a kamerák képalkotásának a modellezése is.

A kamerák beállítás szempontjából három féle képpen mozgíthatóak: egy függőleges tengely körül, egy vízszintes tengely körül, valamint meg lehet őket dönteni jobbra vagy balra. Ezen kívül azt is be kell állítani, hogy a kamera által alkotott képbe a valóság mekkora szelete fér bele, mekkora a kamera nagyítása.

Ahhoz, hogy a 3D-ben értelmezett virtuális modellt a 2D-s kamera képre vetítse, a rendszer perspektivikus vetítést alkalmaz. A vetítési síkot többféleképpen is meg lehet határozni, az alkalmazásban, ahogy a 3. ábrán is látható, ez két vektor segítségével van megoldva. Ennek köszönhetően egyszerűbben lehet a kamera beállítások változtatásai nyomán módosítani a hozzá tartozó vetítési síkon, hiszen csupán a változtatásban érintett vektoron kell állítani, így megvalósítva a virtuális kamera „mozgatását”. A függőleges tengely körüli mozgatás során az ábrán látható U vektor változik, a vízszintes mozgatás során a V , míg döntés során mind a kettő. A vetítési sík mérete a nagyítási beállítások függvényében változik. A perspektivikus vetítés értelmében egy 3D-s pont ott fog megjelenni a 2D-s képen, ahol a pont és a vetítési középpont (esetünkben a kamera pontja) között húzott egyenes metszi a vetítési síkot.

Így a rendszer képes meghatározni a virtuális parkolóhelyek pontjainak a perspektivikus



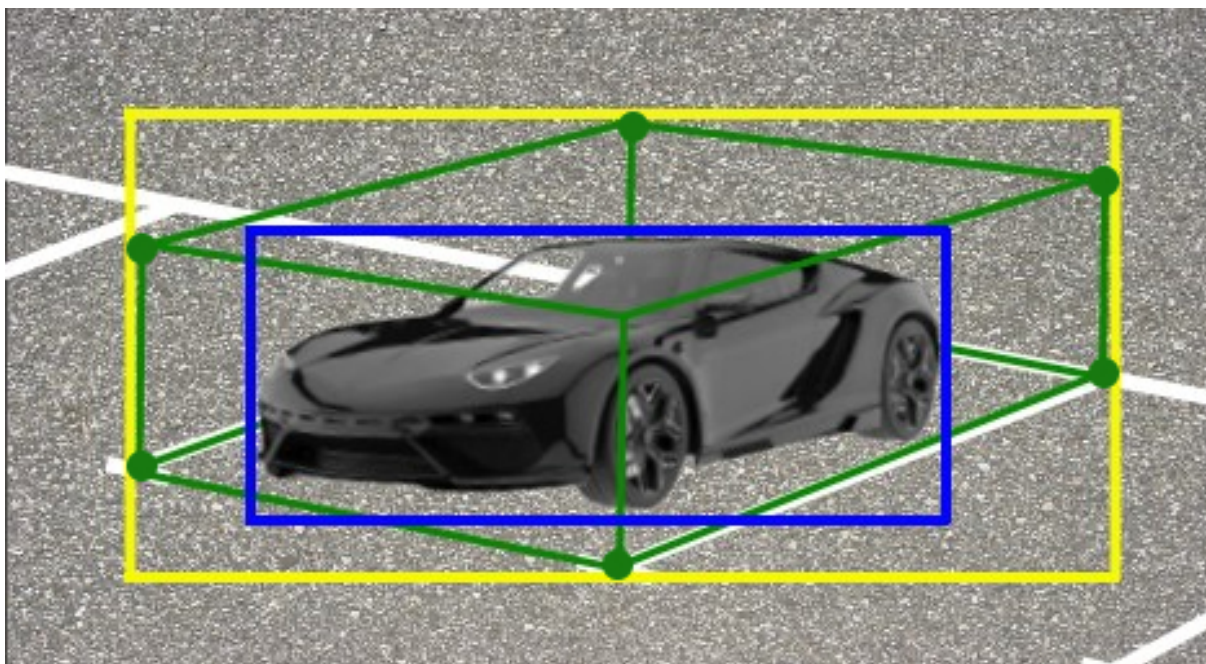
3. ábra. A perspektivikus vetítés ábrázolása.

vetítését a kamera képre, amik megfelelő beállítások mellett a kamera képen szereplő valós parkolóhelyekre fognak illeszkedni.

3.2. Foglaltságot meghatározó logika

A rendszer a parkolóhelyek állapotát egy foglaltságot meghatározó logika alapján állapítja meg. Az előbbi alfejezetben leírt módszer segítségével meghatározott virtuális parkolóhelyet a képre vetíti, majd az ezt közrefogó burkoló téglalapot hasonlítja a YOLO által visszatérített objektumot közrefogó téglalappal. A rendszer annak függvényében határozza meg a parkolóhely állapotát, hogy a parkolóhelyhez tartozó burkoló téglalap milyen arányban foglal magába egy objektumot közrefogó téglalapot.

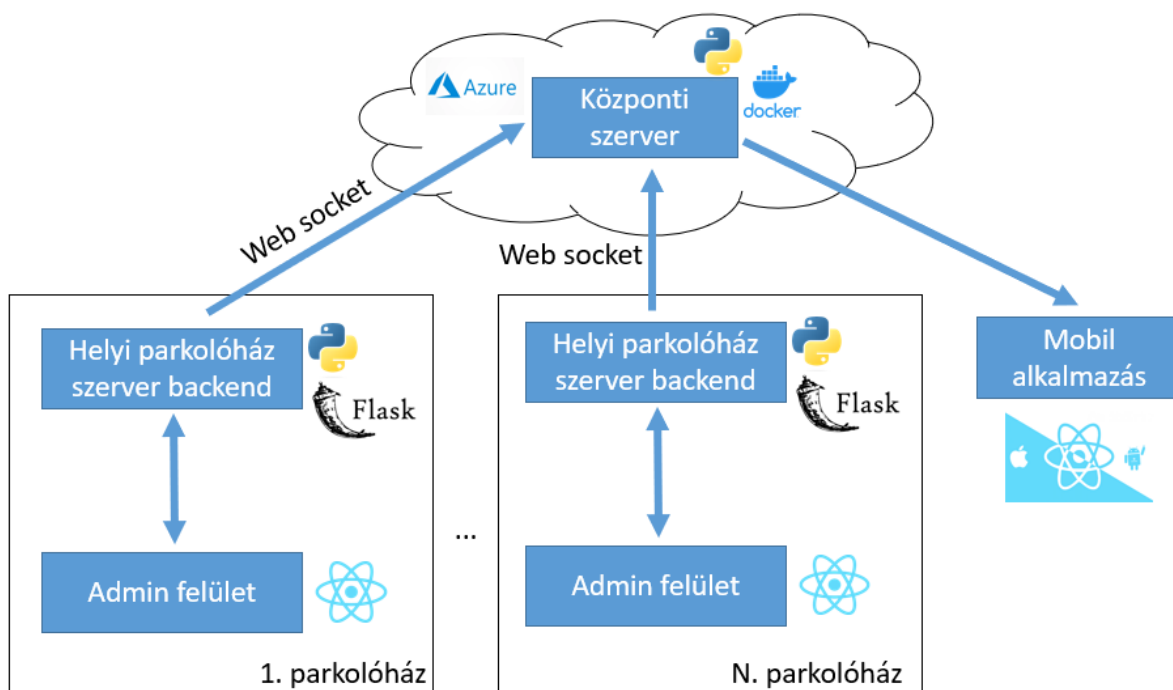
A 4. ábrán láthatunk egy példát ezekre. Zölddel van jelölve a kiegészített virtuális parkolóhely. Alapértelmezetten csak a virtuális parkolóhely alsó négyzete látszik, hiszen ennyi elegendő ahhoz, hogy rá lehessen igazítani a valóságban felfestett parkolóhelyekre. Ugyanakkor ahhoz, hogy egy parkolóhelyet elemezzünk, nem elegendő csupán az alsó négyzete, szükség van a térre, ahova az objektumok parkolhatnak. Alapértelmezetten egy virtuális parkolóhely magasságának 2 méter van beállítva, így a virtuális parkolóhely koordinátáiból és a virtuális kamera beállításáiból könnyen ki lehet számítani a kiegészített virtuális parkolóhely pozícióját a képen. Az így kapott sokszögnek a rendszer megkeresi a határoló pontjait, az ábrán ezek zöld



4. ábra. Példa a foglalásmeghatározó logika szemléltetésére.

ponttal vannak jelölve, és meghatározza az ezeket befogó téglalapot, ami az ábrán sárga színnel van jelölve. Mivel a YOLO is hasonló módon határozza meg a felismert objektumok burkoló téglalapjait, azaz az adott objektumot négy irányból közrefogó legkisebb téglalapot (az ábrán kék színnel jelölve), könnyen össze lehet hasonlítani a kettőt.

A logika szerint azt vizsgáljuk, hogy egy parkolóhely magába foglal-e egy felismert objektumot. Mivel az objektum kisebb, mint a parkolóhely, azt kell vizsgálnunk, hogy a parkolóhely burkoló téglalapja magába foglalja vagy sem a YOLO által visszatérített objektum burkoló téglalapját. Amennyiben egy objektumot burkoló téglalap felületének 90%-a benne van egy parkolóhely burkoló téglalapjában, akkor a rendszer megjelöli az illető parkolóhelyet foglalként, különben a jelölése szabad marad.



5. ábra. A rendszer architektúrája.

4. Az alkalmazás felépítése

Ahogy a 5. ábrán is látszik, az alkalmazás négy fő részből áll: az adminisztrációs felületből, a helyi parkolóház szerverből, a központi szerverből és a mobil applikációból. Ebben a fejezetben részletesen bemutatjuk az egyes részek felépítését és működését.

4.1. Admin felhasználói felület

Minden helyi parkolóház szerverhez kapcsolódik egy adminisztrációs felület, ahol a parkolóházak és parkolóhelyek adatait lehet kezelni. Az adminisztrációs felület három fő részből áll: a parkolóházakhoz tartozó parkolóhelyek listájából, a térkép nézetből, valamint a parkolóházhoz tartozó kamerák beállításait kezelő felületből.

4.1.1. Felhasznált technológiák

Az adminisztrációs felület a React [3] keretrendszerrel készült Single Page Application (SPA). A React lehetővé teszi a komponens-alapú UI fejlesztést, ami nagyban megkönnyíti a fejlesztési folyamatot és javítja a felhasználói élményt. A projekt a react-scripts csomaggal készült, amely számos hasznos eszközt és konfigurációt tartalmaz a React alkalmazások

fejlesztéséhez.

Az adminisztrációs felület előtt egy NGINX statikus szerver található, amely gyors és megbízható szolgáltatást nyújt. Az NGINX továbbá lehetővé teszi a HTTPS kommunikációt a helyi parkolóház felé, ami biztosítja az adatok biztonságát.

A felület stílusát a NextUI [9] 2.0 biztosítja, amely a Tailwind [5] CSS-re épül. A NextUI gyors betöltést és egyszerű stílusalkotást tesz lehetővé, ami javítja a felhasználói élményt.

A kommunikáció a helyi parkolóházzal a Fetch API segítségével történik, amely egy modern, promise-alapú interfész a hálózati kérésekhez.

Az adminisztrációs felület, a helyi parkolóház és az adatbázis mind Docker környezetben futnak. A Docker lehetővé teszi az alkalmazások izolálását és egyszerű telepítését, ami nagyban megkönnyíti a fejlesztési és üzemeltetési folyamatokat. Az adatbázis MySQL [2], amely egy megbízható és széles körben használt relációs adatbázis-kezelő rendszer.

4.1.2. Belső struktúra

Az adminisztrációs felület belső architektúrája a React komponens-hierarchián alapul. A gyökérkomponens felelős a navigációs útvonalak és az alkalmazás állapotának kezeléséért, valamint az aktuális parkolóházhoz kapcsolódó információk, mint az összfoglaltság és a legutóbbi predikciók megjelenítéséért. A parkolóhelyek komponens a parkolóhelyek listáját tartalmazza NextUI kártya komponensek formájában, lehetővé téve a felhasználók számára, hogy megtekinthessék és módosíthassák a kiválasztott parkolóház kameráinak adatait. Egy szerkesztő és adatmegjelenítő komponens biztosítja a kamerák adatainak módosítását és megjelenítését, valamint a kamerák képeinek valós idejű megjelenítését. Ezen a felületen az adminisztrátor meg tudja tekinteni a parkolóhelyeket a kamerák képein, a virtuális vetítés után. Egy térkép nézetet megjelenítő komponens lehetővé teszi az adminisztrátor számára, hogy megtekinthesse a parkolóházban a parkolóhelyek elhelyezkedését és állapotát. A térkép nézetben az adminisztrátor a parkolóhelyeket mozgatni tudja és átméretezheti, valamint új parkolóhelyeket tud hozzáadni és törölni.

Az összes komponens a Fetch API segítségével kommunikál a helyi parkolóház szerverével, hogy frissítse és lekérdezze az adatokat. Az adatok állapotát a React állapotkezelő funkciói kezelik.

4.2. Helyi parkolóház szerver

A helyi parkolóház szerver felelős a parkolóházakban található parkolóhelyek állapotának érzékeléséért. Lehetőség van akár több parkolóház kezelésére is. Minden parkolóházhoz tartozó parkolóhely adatait továbbítjuk a központi szerver felé. Ez a kommunikáció egyirányú, ugyanis a helyi parkolóház szerver csak a központi szerverrel tud kommunikálni, és fordítva nem történik kommunikáció. Biztonsági okok miatt nem feltétlenül szeretnénk publikussá tenni a helyi parkolóház szerverét, így a kommunikációt mindig a helyi parkolóház szerver kezdeményezi a központi szerver felé.

4.2.1. Felhasznált technológiák

A helyi parkolóház szerver Python nyelvben van megírva, és a Flask [8] keretrendszerre épül.

A Flask és az SQLAlchemy használatának számos előnye van. A Flask egy pehelysúlyú és rugalmas Python webkeretrendszer, amely az alapvető funkciókat nyújtja a webalkalmazások létrehozásához. Egyszerűsége és rugalmassága miatt ideális választás a parkolóház szerver applikációhoz, ahol gyors és pontos válaszadásra van szükség.

Az SQLAlchemy egy erőteljes SQL eszközkészlet és Objektum-Relációs Leképezés (ORM) keretrendszer Pythonhoz. Az SQLAlchemy lehetővé teszi a fejlesztők számára, hogy SQL adatbázisokkal dolgozzanak Pythonban, jelentősen leegyszerűsítve az adatbázis-műveleteket, és csökkentve a hibák előfordulási valószínűségét.

4.2.2. Belső struktúra

A helyi parkolóház szerver egy háromrétegű architektúrát alkalmaz. A rendszer három fő rétege: a prezentációs réteg, az üzleti logika réteg és az adathozzáférési réteg. A Prezentációs réteg feladata, hogy kiszolgálja a helyi parkolóház adminisztrációs felületét. Az Üzleti logika rétegnek az alábbiak a feladatai:

- A szabad parkolóhelyek érzékelése a mesterséges intelligencia algoritmusok segítségével, a kamerák által rögzített képek feldolgozásával.
- A virtuális modell perspektívikus vetítésének alkalmazása a parkolóhelyek 2D koordinátáinak valós világbeli koordinátákra való leképezéséhez.

- A perzisztencia rétegben az adatokat egy MySQL adatbázisban tárolja, amelyhez csatlakozik egy SQLAlchemy ORM keretrendszer segítségével.

4.2.3. Adatmodell

Az adatmodell négy fő entitást tartalmaz: a 'Camera', 'IsVisible', 'CarParkTable' és 'ParkingSpace' osztályokat.

A 'Camera' osztály reprezentálja a parkolóházban található kamerákat. Minden kamera rendelkezik egy egyedi azonosítóval ('id'), névvel ('name'), URL-lel ('url'), felhasználónévvel ('username'), jelszóval ('password'), koordinátával ('positionLat', 'positionLong'), magassággal ('height'), és a parkolóház azonosítójával ('carParkID'). Továbbá, a kamera beállításait is tárolja: 'Alfa', 'Beta', 'Zoom', 'SlantAngle', 'ImageRotationAngle'. Az 'Alfa' a kamera vízszintes forgatását, a 'Beta' a kamera függőleges forgatását, a 'Zoom' a kamera zoomolását, a 'SlantAngle' a kamera dőlését, és az 'ImageRotationAngle' magának a kamera képének a forgatását jelöli. Lehetőség van a kamera dőlésszögének beállítására, ugyanakkor a kamera képét is elforgathatjuk, mivel a képfelismerő hatékonyabban működik, ha a képen az objektumok közel vízszintes helyzetben vannak.

Az 'IsVisible' osztály a kamera és a parkolóhely közötti kapcsolatot képviseli. Minden kapcsolat rendelkezik egy egyedi azonosítóval ('id'), a kamera azonosítójával ('cameraID') és a parkolóhely azonosítójával ('parkingSpaceID').

A 'CarParkTable' osztály a parkolóházakat képviseli. Minden parkolóház rendelkezik egy egyedi azonosítóval ('id'), névvel ('name'), koordinátával ('positionLat', 'positionLong'), magassággal ('height'), és rendelkezésre állással ('availability').

A 'ParkingSpace' osztály a parkolóhelyeket képviseli. Minden parkolóhely rendelkezik egy egyedi azonosítóval ('id'), a parkolóház azonosítójával ('carParkID'), a parkolóhely négy sarkának koordinátaival ('topRightLat', 'topRightLon', 'topLeftLat', 'topLeftLon', 'bottomRightLat', 'bottomRightLon', 'bottomLeftLat', 'bottomLeftLon'), és a parkolóhely állapotával ('parkingStatus').

Az adatmodell a kapcsolatokat is kezeli. A 'Camera' és 'IsVisible' osztályok között, valamint a 'ParkingSpace' és 'IsVisible' osztályok között egy-egy kapcsolat található. A 'CarParkTable' és 'ParkingSpace' osztályok között pedig egy "egy-sok" kapcsolat van, ahol egy parkolóházhoz több parkolóhely is tartozhat.

4.2.4. API

Az alkalmazásunk a Flask webkeretrendszer segítségével RESTful API-t biztosít. A REST (Representational State Transfer) architektúra egy széles körben elfogadott szabvány az API-k tervezéséhez, amely lehetővé teszi a kliensek számára, hogy interakcióba lépjenek az alkalmazásunkkal a HTTPS protokollon keresztül.

Az API lehetővé teszi az adminisztrátor számára, hogy lekérdezze a kamerák, parkolóházak és parkolóhelyek adatait, új entitásokat adjon hozzá, frissítse a meglévő entitások adatait, és törölje a meglévő entitásokat. Az adatokat JSON formátumban küldjük és fogadjuk.

Az API a CRUD (Create, Read, Update, Delete) műveleteket támogatja, amelyek a leggyakoribb műveletek az adatkezelésben. Ez lehetővé teszi a adminisztrátorok számára, hogy teljesen a saját igényeik szerint kezeljék az adatokat.

4.2.5. Security

A biztonság kulcsfontosságú szempont a rendszerünkben, különösen mivel a kamerák jelszavait tároljuk, amelyeket vissza kell tudnunk szerezni az RTSP (Real-Time Streaming Protocol) protokoll számára. Az RTSP protokoll lehetővé teszi a kamerák által rögzített képek feldolgozását, továbbá lehetőséget biztosít az adminisztrátorok számára, hogy valós időben megtekintsék a kamerák képeit. Egy parkolóházban a legtöbb kamera RTSP protokollt használ.

A kommunikáció során HTTPS-t (Hyper Text Transfer Protocol Secure) használunk, amely a HTTP biztonságos változata. A HTTPS titkosítja az adatokat, amelyeket a kliens és a szerver között küldenek, így megakadályozza a harmadik felek számára, hogy hozzáférjenek az érzékeny információkhoz. Ez különösen fontos, mivel jelszavakat és más érzékeny adatokat továbbítunk.

A jelszavak adatbázisban történő tárolásához a Fernet titkosítási könyvtárat használjuk, amely biztonságos és könnyen használható módot biztosít a jelszavak titkosítására és visszafejtésére. Az RTSP protokoll számára a konkrét jelszó szükséges, így szimmetrikus kulcsú titkosításra van szükség.

Végül, fontos megjegyezni, hogy ezek a parkolóházak helyi jellegűek, és nem nyilvánosak a nagyközönség számára. Ez további védelmi szintet biztosít, mivel korlátozza a potenciális támadási lehetőségeket.

4.3. Központi szerver

A központi szerver a rendszer központi eleme, amely az összes adatot kezeli és szolgáltatja a kliensek számára. Ez a szerver felelős a lokális parkolóház szerverektől kapott adatok fogadásáért, feldolgozásáért és tárolásáért. A szerver Flask keretrendszer segítségével van megírva, futtatása Azure App Service Plan-on [1] történik, ami egy felhőalapú szolgáltatás, amely lehetővé teszi a webalkalmazások futtatását magas rendelkezésreállás (uptime) mellett.

4.3.1. Felhasznált technológiák

A központi szerver alapját a Flask keretrendszer adja, az SQLAlchemy keretrendszert használjuk az adatok kezelésére.

A szerveren futó alkalmazás a Gunicorn WSGI HTTP szerver segítségével képes egyszerre több felhasználót kiszolgálni. A Gunicorn egy Python alapú szerver, amely képes egyszerre több beérkező kérést is kezelni, és így nagyobb terhelés mellett is stabilan működni.

A SocketIO technológiát használjuk a valós idejű kommunikációra a szerver és a helyi parkolóház között. A SocketIO lehetővé teszi a valós idejű, kétirányú kommunikációt a webkliensek és a szerverek között.

Az alkalmazásunk egy Azure App Service Plan-on fut. Az Azure App Service egy olyan felhőszolgáltatás, amely lehetővé teszi a webalkalmazások, mobilalkalmazások háttérszolgáltatásainak, RESTful API-knak a futtatását. Az Azure App Service előnyei közé tartozik a skálázhatóság, a nagy rendelkezésre állás, az automatikus menedzsment, a DevOps támogatás és a globális skálázhatóság. Az Azure App Service segítségével könnyen skálázhatjuk az alkalmazásunkat a felhasználói igényeknek megfelelően, és biztosíthatjuk, hogy az alkalmazás mindig elérhető legyen a felhasználók számára.

4.3.2. Belső struktúra

A központi szerver belső struktúrája három rétegből áll: a prezentációs rétegből, az üzleti logika rétegből és az adathozzáférési rétegből.

A prezentációs réteg felelős a mobil kliensek kiszolgálásáért, és a kliensek által küldött kérések feldolgozásáért. Ez a réteg RESTful kommunikációt használ a mobil kliensekkel, és a kliensek által küldött kéréseket továbbítja az üzleti logika rétegnek.

Az üzleti logika réteg felelős az adatok feldolgozásáért, amelyeket a helyi parkolóház szerverektől kap, SocketIO segítségével. Ez a réteg kezeli a parkolóházak és parkolóhelyek

adatait, és felelős a beérkező adatok validálásáért, feldolgozásáért és továbbításáért a perzisztencia rétegnek.

A perzisztencia rétegben az adatokat egy MySQL adatbázisban tároljuk, amelyhez csatlakozik egy Flask-SQLAlchemy ORM. Az adatokat a perzisztencia réteg kezeli, és a perzisztencia réteg felelős az adatok tárolásáért és lekérdezéséért. Ez a réteg biztosítja, hogy az adatok hosszú távon is elérhetőek legyenek, és felelős az adatok integritásának és biztonságának fenntartásáért.

4.3.3. Adatmodell

Az adatmodell a rendszerben tárolt adatok struktúráját írja le. A modell két fő entitást tartalmaz: a Parkolóházat és a Parkolóhelyet.

A Parkolóház entitás reprezentálja a parkolóházakat. Minden parkolóház rendelkezik egy egyedi azonosítóval, névvel, címmel és földrajzi koordinátákkal. Az azonosító a parkolóház egyedi azonosítója a rendszerben. A név és cím a parkolóház fizikai helyét jelöli, míg a földrajzi koordináták GPS koordináták formájában tárolják a parkolóház pontos helyét.

A Parkolóhely entitás reprezentálja a parkolóházakban található parkolóhelyeket. Minden parkolóhely rendelkezik egy egyedi azonosítóval, állapottal és a hozzá tartozó parkolóház azonosítójával. Az azonosító a parkolóhely egyedi azonosítója a rendszerben. Az állapot jelzi, hogy a parkolóhely szabad-e vagy foglalt. A parkolóház azonosítója a parkolóhelyhez tartozó parkolóházat jelöli.

Az adatmodellben a Parkolóház és Parkolóhely entitások között egy a többhöz kapcsolat van, azaz egy parkolóházhoz több parkolóhely is tartozhat.

4.3.4. API

Az API (Application Programming Interface) a központi szerver és a kliensek közötti kommunikációt biztosítja. Az API két fő részből áll: a HTTPS végpontokból és a WebSocket eseményekből.

A HTTPS végpontok a RESTful API részét képezik, amelyek a HTTPS protokollon keresztül kommunikálnak a mobil kliensekkel. A végpontok a következők:

- /: Az alapértelmezett végpont, amely a szerver állapotát jelzi.
- /car-parks: Az összes parkolóház adatait adja vissza.

- `/car-parks/<int:id>`: Egy adott azonosítójú parkolóház adatait adja vissza.

A végpontok GET metódust használnak az adatok lekérdezésére. A válaszok JSON formátumúak, amelyek tartalmazzák a lekérdezett adatokat.

A WebSocket események a valós idejű kommunikációt biztosítják a központi szerver és a helyi parkolóház szerverek között. A szerver a SocketIO technológiát használja a WebSocket kommunikáció kezelésére.

Az események JSON formátumú adatokat fogadnak és küldenek. Az adatokat a szerver feldolgozza, és a megfelelő módon tárolja.

4.4. Mobil appikáció

A mobil applikáció biztosítja a felhasználóknak a parkolóházak adataihoz való hozzáférést, felhasználóbarát módon megjelenítve ezeket az adatokat.

4.4.1. Felhasznált technológiák

Az alkalmazás React Native [4] keretrendszerrel készült, ami egy mobil applikációk fejlesztéséhez használt nyílt forráskódú keretrendszer. Egyik legfontosabb tulajdonsága, hogy lehetővé teszi a cross-platform fejlesztést, egyszerre lehet fejleszteni Android és iOS rendszerekre egyaránt. Ugyancsak nagy előnye, hogy a React-hez hasonlóan komponensalapú megközelítést alkalmaz, így megkönnyíti a React-ben már jártas fejlesztőknek a mobil applikációk készítését. Ezen kívül a React Native, bár a fejlesztés JavaScriptben történik, natív elemeket használ, ennek köszönhetően a natív alkalmazásokra hasonló teljesítményre képes.

Az alkalmazás fejlesztése során használtuk továbbá az Expo keretrendszert, melynek célja, hogy leegyszerűsítse a mobilalkalmazások telepítését, felgyorsítva így a fejlesztési folyamatot. Az Expo használatával a fejlesztő könnyen tudja tesztelni a React Native-ben írt alkalmazását, köszönhetően a Live Reload és Hot Reload funkcióknak, amik lehetővé teszik, hogy a fejlesztők azonnal lássák változtatásaikat, újrafordítás vagy újratelepítés nélkül. Ezek mellett az Expo megkönnyíti olyan funkciók integrálását, mint a térkép, GPS stb. Az Expo Go nevű mobil alkalmazással pedig a fejlesztők a saját mobiljukon is tesztelni tudják az alkalmazásukat, kényelmesebbé téve a fejlesztést.

4.4.2. Belső struktúra

A mobil applikáció belső architektúrája, az adminisztrációs felülethez hasonlóan React komponens-hierarchián alapul.

Az alkalmazás gyökérkomponense teszi lehetővé a komponensek közötti navigációt. A parkolóház lista komponens megjeleníti a parkolóházak listáját, feltüntetve telítettségi szintjüket, százalékban kifejezve.

A parkolóház komponens megjeleníti a kiválasztott parkolóházhoz tartozó parkolóhelyek listáját. A parkolóhely részletek komponens megjeleníti egy parkolóhely adatait, itt lehetőség van a kiválasztott parkolóhelyet a kedvencekhez adni.

Másik fontos komponens a térkép komponens, a parkolóhelyeket, foglaltságuk függvényében, piros vagy zöld színnel kirajzolja a térképre. A térkép komponens tartalmaz egy keresőmezőt, amivel rá lehet keresni a parkolóházakra. Amint a felhasználó kiválaszt egyet, a térkép a megfelelő parkolóházra navigál.

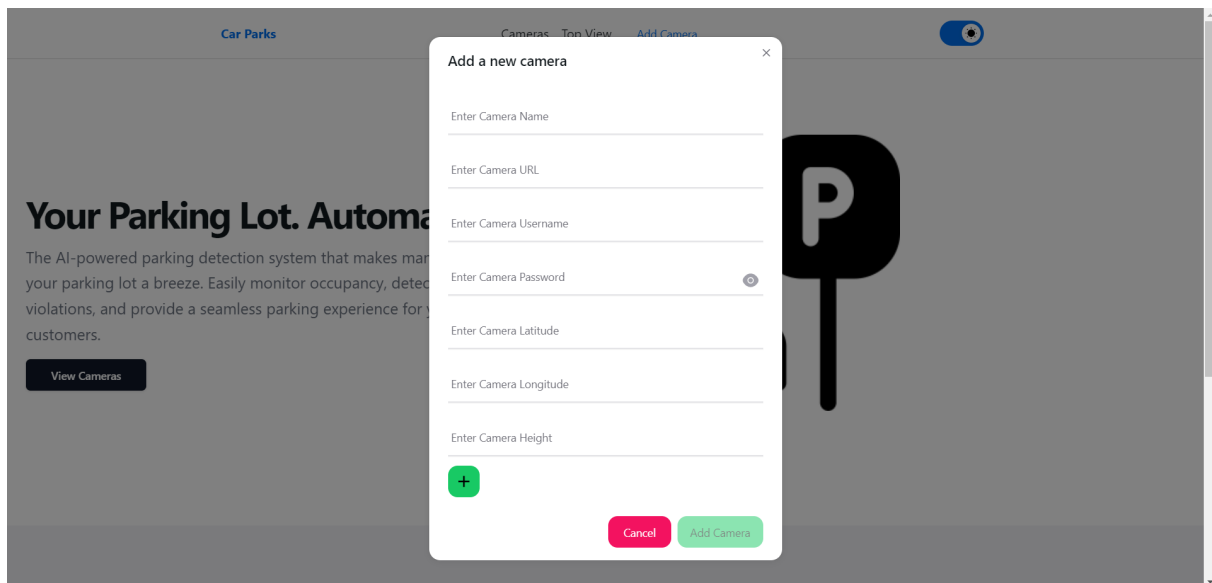
Végül pedig az utolsó fontosabb komponens a kedvencek komponens, mely a kedvencek listáját jeleníti meg.

A mobil applikáció komponensei Fetch API segítségével kéri le a központi szerverről a parkolóházak adatait.

4.4.3. Kedvencek tárolása

A mobil applikáció a kedvencek tárolásához az AsyncStorage React Native könyvtárat használja. Ez a könyvtár az adatokat kulcs-érték párok formájában tárolja. Ahogy a neve is mutatja az AsyncStorage aszinkron módon működik, a tárolási és lekérési műveletek nem blokkolják a felhasználói felületet. Az AsyncStorage-el tárolt adatok az eszközön kerülnek tárolásra, vagyis az alkalmazás újraindítása esetén sem vesznek el.

Az alkalmazás megnyitása során létrejön egy FavoritesProvider nevű komponens, ami betölti az AsyncStorage segítségével a kedvencek adatait. Ezen a komponensen keresztül történik a kedvencek hozzáadása, törlése, illetve a kedvencek lekérése a többi komponens számára.



6. ábra. Új kamera hozzáadása.

5. Az alkalmazás működése

Az alkalmazás működését két nézőpontból lehet vizsgálni: adminisztrátori és felhasználói funkcionálisok. Ez a fejezet részletesen, ábrákkal szemléltetve bemutatja a rendszer használatát.

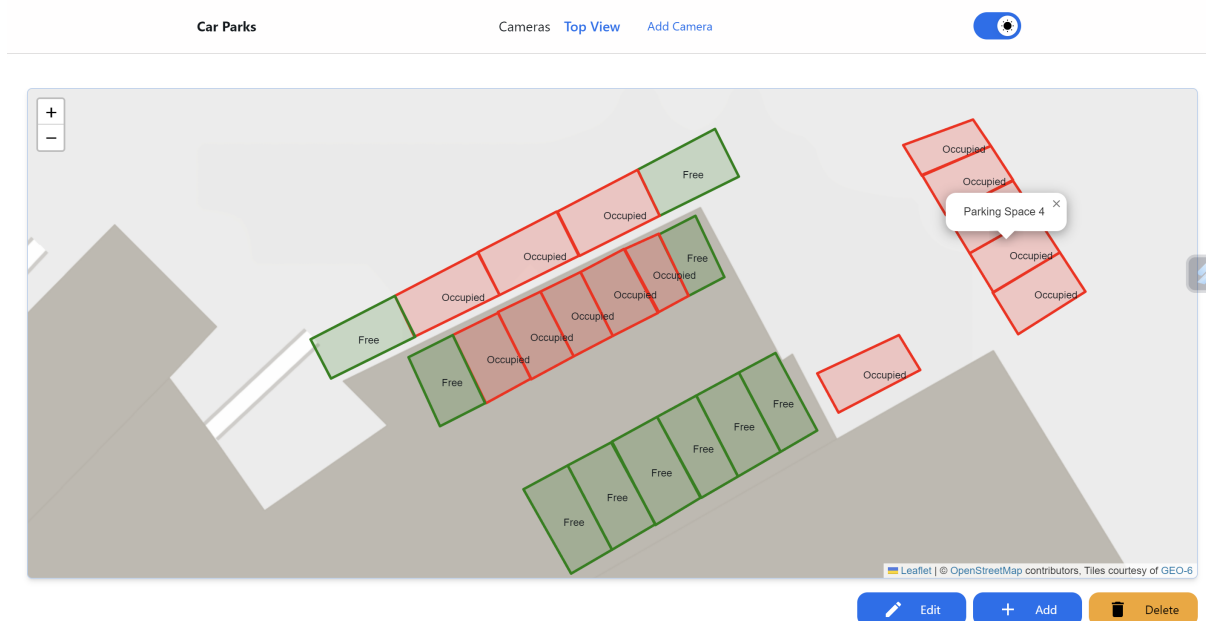
5.1. Adminisztrátor

Az adminisztrátornak az adminisztrátor felhasználói felületen keresztül van lehetősége a rendszer beállítására: új kamerák, parkolóhelyek felvezetésére, ezek törlésére, pozícióik beállítására és a virtuális modell beállítására.

5.1.1. Kamerák felvezetése

A helyi parkolóház szerver beüzemelése a kamerák felvezetésével kezdődik. A szerver teljes mértékben használni tudja a már felszerelt biztonsági kamerákat, így azoknak csak az URL-jét, a felhasználónevét, illetve jelszavát szükséges megadni, és ezek alapján a szerver már hozzá is tud férni az általuk közvetített streamhez. Új kamerákat csak arra a részre kell felszerelni, amit nem lát egyetlen biztonsági kamera sem, így költséget és energiát lehet spórolni.

Új kamera felvezetésekor, a felhasználónév és jelszó mellett szükséges még megadni a kamera koordinátáit, illetve a talajtól mért magasságát, de ezek akár megközelítő értékek is lehetnek, hiszen mindkettő pontosítására lesz még lehetőség a későbbiekben is.



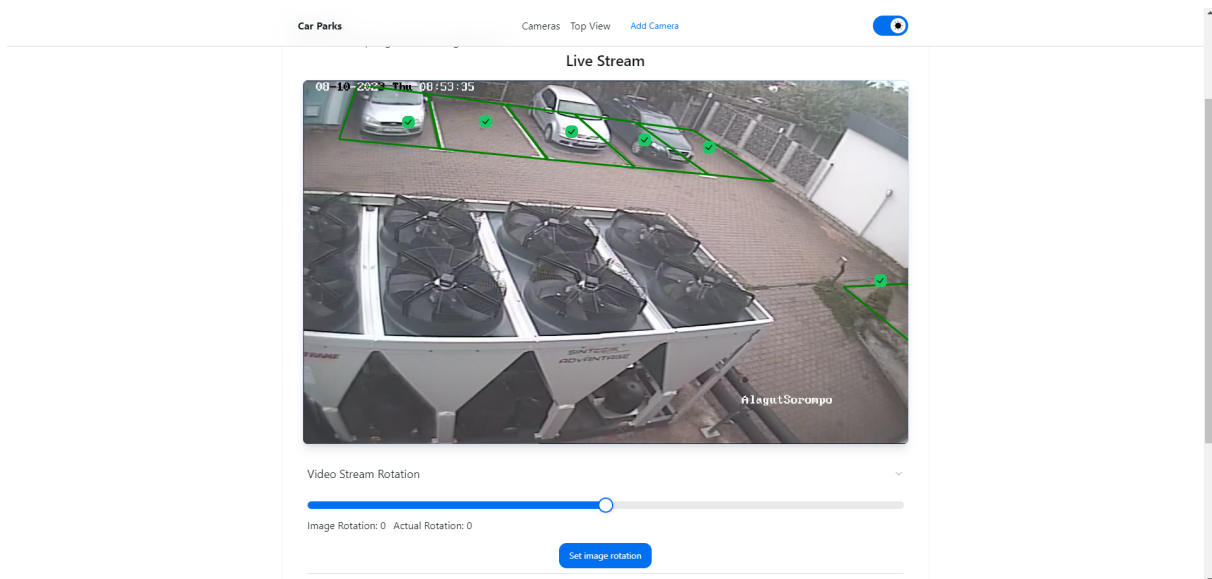
7. ábra. A felülnézeti mód (Top View).

5.1.2. Felülnézeti mód

A helyi parkolóház szerver felhasználói felülete biztosít, a „Top View” fül alatt, egy felülnézetet, ahol meg lehet tekinteni a rendszer kameráinak, illetve a parkolóhelyeknek az elhelyezkedését a térképen.

Ezeket van lehetőség egyszerűen, húzással változtatni az „Edit” módban. A parkolóhelyek négyszögek, a négy csúcsuk mozgatásával lehet módosítani őket. Előfordulhat, hogy két parkolóhely csúcsa egymásra esik, ilyenkor a hozzájuk tartozó markerek takarják egymást. Ebben az esetben a jobbklikk segítségével lehet azt elérni, hogy a felső marker az alatta levő marker alá kerüljön. Az „Edit” mód még biztosít egy hasznos eszközt, ami több parkolóhely csúcsát egy vonalra helyezi. Általában a parkolóhelyek egymás mellett vannak, és csúcsaik egymással szemben, két vonalon. Ezt nehéz és időigényes lenne kézzel kivitelezni, emellett nem is lenne teljesen pontos. Ennek a funkciónak az alkalmazásához az adminisztrátor duplaklikkel kijelöli a kiválasztott markereket. Ilyenkor az amúgy kék marker zöldre vált. A „Merge” gomb, amint van kijelölve változtatás, elérhetővé válik, és rákattintás esetén, lineáris regresszió segítségével egy vonalra helyezi a kijelölt pontokat. Ezt a többi változtatáshoz hasonlóan a „Save” gombbal tudjuk elmenteni.

Ugyancsak a „Top View” fül alatt van lehetőség új parkolóhelyeket felvezetni. Az „Add” mód kiválasztása után klikkeléssel lehelyezhetjük a térképre az új parkolóhely négy csúcsát. Ekkor megjelenik a parkolóhely négyszöge a térképen. Itt még lehetőség van a csúcsokon



8. ábra. Az élő stream a virtuális modellel, illetve a "Video Stream Rotation" beállító csúszka. A képen látható zöld mezők jelölik azokat a parkolóhelyeket, amiket az adminisztrátor kiválasztott, mert láthatóak a kamera számára.

állítani, ha szükséges, majd az „Add” gombra kattintva lehet elmenteni a változtatásokat.

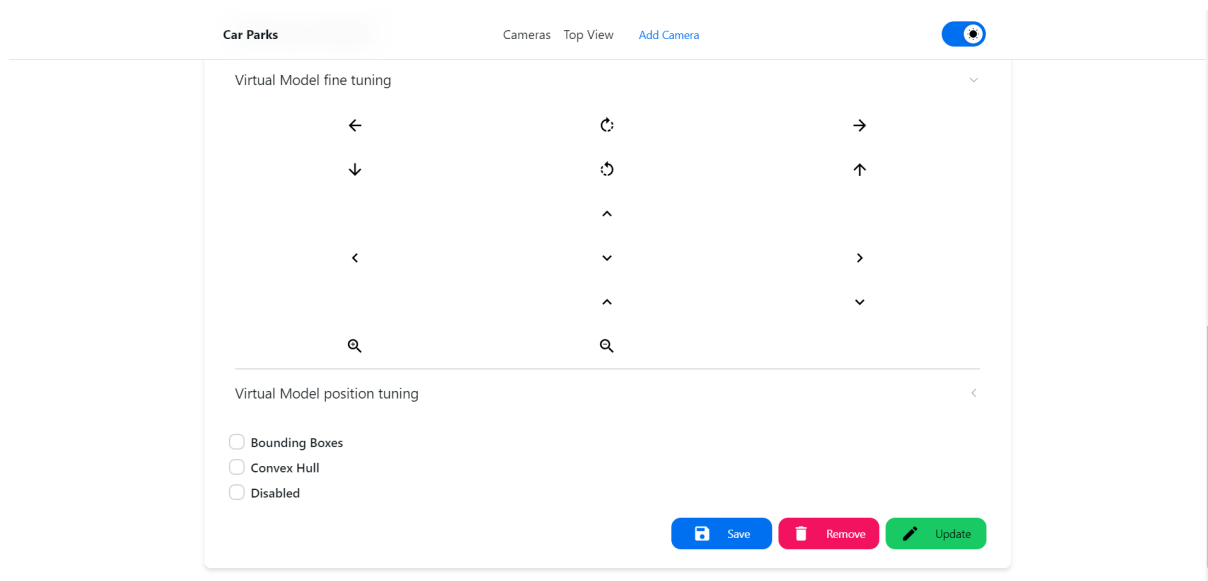
Parkolóhelyek törlésére a „Delete” módban van lehetőség. A térképen kattintással ki lehet jelölni a törölni kívánt parkolóhelyeket, a „Delete” gombra kattintva ezek törlődnek a rendszerből.

5.1.3. Kamera beállítások

A rendszer beállítását a „Cameras” fül alatt lehet folytatni. Itt a felhasználói felület megjeleníti a rendszerben levő kamerák listáját. Az adminisztrátor ebből választhatja ki azt a kamerát, amelyikkel foglalkozni szeretne. A kamera kiválasztása után megjelenik a kamerához tartozó felület: a kamera adatai, élő streamje, a virtuális modellel kiegészítve, valamint a beállító gombok.

A stream alatt a „Video Stream Rotation” menüpontot lenyitva egy csúszka segítségével lehet fordítani a kamera streamjén. Erre akkor lehet szükség, ha a kamera rosszul lett felszerelve, vagy elfordult, ugyanis a mesterséges intelligencia nagyobb hatékonysággal ismeri fel az autókat a valóságnak megfelelő képeken, mint az eldőlteken.

A következő menüpont a „Virtual Model fine tuning”, amely segítségével változtatni lehet a virtuális kamera beállításain, hogy azok minél pontosabban modellezzék a valóságot. Lehetőség van forgatni a virtuális kamerát a vízszintes és függőleges tengely mentén mindkét irányban, el lehet dönteni saját maga körül, lehet változtatni a virtuális kamera pozícióján mind a négy



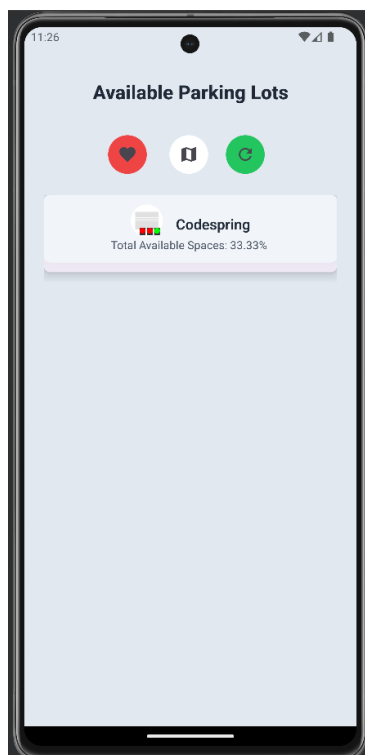
9. ábra. Kamera beállítások: a virtuális kamera beállításait módosító gombok.

irányban (előre, hátra, jobbra, balra), valamint növelni vagy csökkenteni lehet a virtuális kamera magasságát.

Az utolsó lenyitható menüpont a „Virtual Model position tuning”, ami az előző menüpont által beállított értékeket jeleníti meg. Amennyiben az adminisztrátor meg tudja becsülni az értékeket, itt lehetősége van manuálisan beállítani a modellt, amivel időt spórolhat, hiszen ezután már csak kevés finomhangolásra van szükség, gyorsabb mintha mindent a gombokkal állított volna be.

A virtuális modell beállítása után az adminisztrátor kijelöli azokat a parkolóhelyeket, amik láthatóak a kamera számára. Erre azért van szükség, mert bár a virtuális modell lemodellezi a valóságot, előfordulhat olyan eset, hogy a kamera látómezejében van egy fal vagy másféle akadály, ami miatt nem tud becslést adni egy parkolóhely állapotáról. Fontos megjegyezni, hogy a 8. ábrán is látható, zölddel jelölt parkolóhelyek ebben a nézetben nem azt jelentik, hogy szabadok, hanem azt, hogy olyan parkolóhelyek, amik előtt nincs akadály, tehát a rendszer megfelelően képes elemezni őket. Például, ha egy parkolóhely lenne a képen látható fűtőrendszer doboz mögött, akkor az pirossal lenne jelölve, hiszen az adminisztrátor nem választja ki, mivel egy akadály miatt nem látható. Amennyiben az adminisztrátor kiválasztotta azokat, az érvényes parkolóhelyek zölddel vannak jelölve, ellenkező esetben pirossal.

A három lenyitható menüpont alatt található három checkbox, melyek további segítséget nyújtanak a beállításhoz. A „Convex Hull” megjeleníti a kamera által látható parkolóhelyek burkoló dobozait, a „Bounding Boxes” az ezeket határoló négyzetet sárga színnel, valamint



10. ábra. A parkolóházak listája és azok telítettsége.

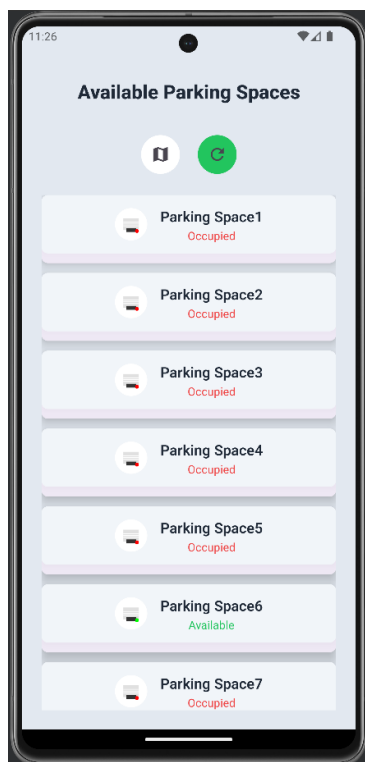
a mesterséges intelligencia által felismert autók határoló négyzeteit, illetve a „Disabled” kikapcsolja a virtuális modell láthatóságát.

5.2. Felhasználó

A felhasználók a mobil applikáción keresztül érik el a parkolóházak adatait.

Az applikáció megnyitáskor lekéri a központi szervertől a parkolóházak adatait: a parkolóház pozícióját, koordinátáit, valamint a szabad helyek arányát százalékban kifejezve. A felhasználónak a 10. ábrán is látható kezdő oldalon több lehetősége van. Kiválaszthat egy parkolóházat, megnyithatja a térkép üzemmódot, vagy megtekintheti kedvenc parkolóhelyeit.

Egy parkolóház kiválasztása után megjelennek egy lista nézetben (a 11. ábrán látható módon) a parkolóház parkolóhelyei. Egy parkolóhelyet kiválasztva megnyílik a parkolóhely részletes adatlapja (12. ábra). Itt a felhasználó hozzá tudja adni a kiválasztott parkolóhelyet a kedvencekhez, illetve tud navigációt kérni a parkolóhelyhez. Navigáció esetében megnyílik a Térkép alkalmazás és elindul egy navigáció a parkolóhely GPS koordinátái alapján. Ez a funkció a jelen formájában még elég kezdetleges, mivel csak egy koordináta szerinti navigációt tartalmaz, így gyakran rossz irányból közelíti meg a parkolóházat. Ugyanakkor, ahogy a Következtetések és továbbfejlesztések fejezetben is ki lesz fejtve, a jövőbeli célok között

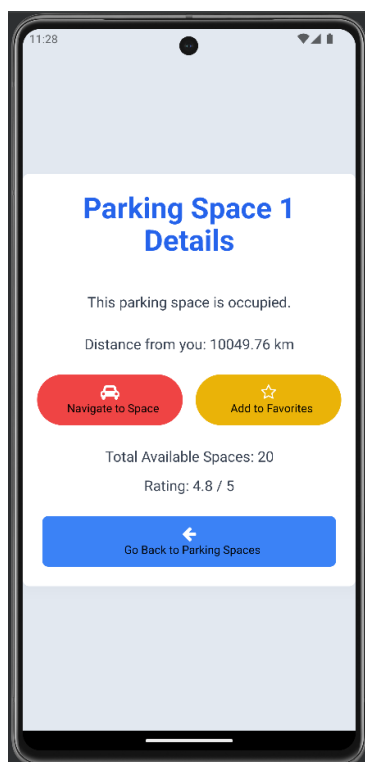


11. ábra. Egy parkolóházhoz tartozó parkolóhelyek listája.

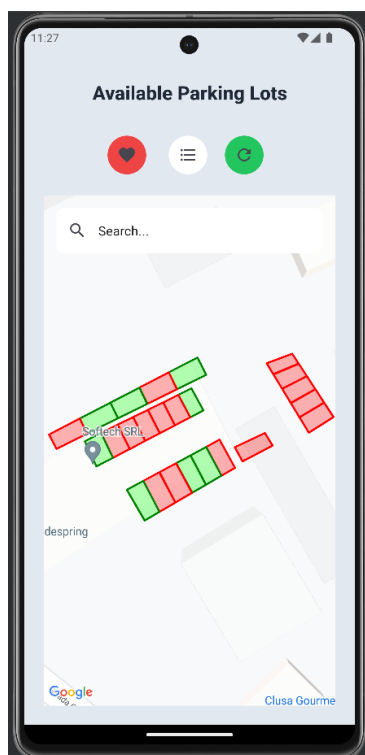
szerepel egy alkalmazásba integrált navigációs funkció, ami segítségével a felhasználó el lesz vezetve a parkolóház bejáratához, majd azon belül a parkolóhelyhez.

A jobb láthatóság érdekében a felhasználó választhatja a 13. ábrán is látható térkép módot. Erre a parkolóházak listájában és egy parkolóház helyeinek listájában van lehetősége. Az előbbinél az összes parkolóház megjelenik a térképen, markerekkel jelölve, és amennyiben ráközelít egyre, akkor a marker helyett megjelennek a parkolóhelyek, pirossal jelölve a foglalt, zölddel a szabad helyeket. Amennyiben a felhasználó kiválasztott egy parkolóházat, és utána vált a térkép nézetre, a térképen csak a parkolóházhoz tartozó parkolóhelyek fognak megjelenni.

A kedvencekre nyomva megjelennek a kedvencekhez adott parkolóhelyek lista nézetben. Itt is van lehetőség térképes nézetre, ebben az esetben a térképen kizárólag a kedvenc parkolóhelyek jelennek meg.



12. ábra. Egy parkolóhely részletes adatlapja.



13. ábra. Térkép nézet.

Következtetések és továbbfejlesztési lehetőségek

A projekt sikeresen megvalósította a parkolóházakban lévő szabad parkolóhelyek detektálását és megjelenítését egy mesterséges intelligencia által vezérelt alkalmazással. Az alkalmazás gyors és hatékony megoldást kínál a felhasználóknak a valós idejű parkolóhely-keresésre, jelentős előnyt biztosítva az autósoknak és a parkolóházak üzemeltetőinek egyaránt. Emellett ez egy költséghatékony megoldás is lehetne, hiszen az alkalmazás általános rendelkezésre állása és könnyű integrációja révén csökkenthetők lennének a telepítés és üzemeltetés költségei, beleértve a már meglévő rendszerekbe való beépítést is. Az alkalmazás jelenlegi kihívása azonban a parkolóhelyek manuális felviteléből adódik, ami időigényes feladat lehet, különösen sok parkolóhely esetén.

A jelenlegi állapot alapján több továbbfejlesztési lehetőség is felvetődik, amelyekkel a rendszer funkcionalitása és hatékonysága tovább javítható.

Először is, egy lehetséges továbbfejlesztési irány az lenne, hogy a Codespring parkolójában teszteljük a rendszert, és az alkalmazottaknak átadjuk tesztelésre. Ezáltal valós visszajelzéseket kapnánk arról, hogy a rendszer milyen pontos és megbízható valós környezetben. Az ilyen tesztek segítségével lehetőségünk lenne finomhangolni és optimalizálni az algoritmusokat a valós kihívásoknak megfelelően.

Második lépésként célszerű lenne fejleszteni a rendszert úgy, hogy a parkolóházon belül az autósokat navigálni tudja a szabad parkolóhelyekhez különböző technikák alkalmazásával. Ennek köszönhetően a felhasználók még könnyebben és gyorsabban találnának parkolóhelyet, ami növelné az alkalmazás használatát és a parkolóházak kihasználtságát.

További hasznos fejlesztés lehet a kamera beállításokat egy új funkcióval kibővíteni, ami a virtuális modell kamera képre való igazítását könnyítené meg. Vannak ugyanis kamerák, amik, hogy nagyobb teret tudjanak befogni, úgynevezett halszemoptikát használnak, így nem lehet pontosan ráigazítani a virtuális modellt a képen látható parkolóhelyekre. Az új funkció segítségével lehetne állítani, hogy mekkora mértékű a halszem effektus, a rendszer pedig ennek megfelelően görbíti meg a virtuális modellt. Így bármilyen típusú kamerával használható lenne, nem csak a klasszikus lencsés kamerákkal.

Végül, de nem utolsósorban, érdemes lenne kiterjeszteni a tesztelést más parkolóházakra is. Ez lehetővé tenné számunkra, hogy teszteljük a rendszer teljesítményét és megbízhatóságát több különböző környezetben. Emellett lehetőségünk nyílna arra is, hogy megvizsgáljuk a központi szerver működését, amikor több helyi szerver csatlakozik hozzá egyszerre.

Ezáltal képet kaphatnánk arról, hogy hogyan skálázódik a rendszer nagyobb terhelés esetén. Ezenkívül megvizsgálhatnánk, hogy milyen felhasználói élményt tudunk biztosítani a mobilalkalmazásban, és hogyan lehetne azt tovább fejleszteni a felhasználói igényeknek megfelelően.

Hivatkozások

- [1] Microsoft Corporation. *Azure App Service - Web App Hosting | Microsoft Azure*. 2024. URL: <https://azure.microsoft.com/en-us/services/app-service/> (elérés dátuma: 2024. máj. 4.)
- [2] Oracle Corporation. *MySQL :: The world's most popular open source database*. 2024. URL: <https://www.mysql.com/> (elérés dátuma: 2024. máj. 4.)
- [3] Facebook Inc. *React - A JavaScript library for building user interfaces*. 2024. URL: <https://reactjs.org/> (elérés dátuma: 2024. máj. 4.)
- [4] Facebook Inc. *React Native - A framework for building native apps using React*. 2024. URL: <https://reactnative.dev/> (elérés dátuma: 2024. máj. 4.)
- [5] Tailwind Labs Inc. *Tailwind CSS - Rapidly build modern websites without ever leaving your HTML*. 2024. URL: <https://tailwindcss.com/> (elérés dátuma: 2024. máj. 4.)
- [6] Fiza Joiya. *Object detection: Yolo vs Faster R-CNN*. 2022. URL: <https://www.doi.org/10.56726/IRJMETs30226> (elérés dátuma: 2024. máj. 4.)
- [7] Ultralytics LLC. *YOLOv8: An Incremental Improvement to Object Detection Algorithms*. 2024. URL: <https://docs.ultralytics.com/models/yolov8/> (elérés dátuma: 2024. máj. 4.)
- [8] Pallets Projects. *Flask (A Python Microframework)*. 2023. URL: <https://flask.palletsprojects.com/> (elérés dátuma: 2024. máj. 4.)
- [9] NextUI Team. *NextUI - A React UI library*. 2024. URL: <https://nextui.org/docs/guide/introduction> (elérés dátuma: 2024. máj. 4.)