Borbély Bálint, VVO4KH 2023

Fábry László, UKG9VP

1. Fejlesztői dokumentáció (PLM)

Tartalomjegyzék

2. Tervezési fázis

A projekt elkészítése során az első lépés, mint minden projekt esetén, a tervezési fázis volt. Itt dőlt el, hogy Bálint foglalkozik az e-mail küldéssel, az adatvizualizációval, és az adatbázisműveletekkel. László dolga volt a kamerás rendszámfelismerés előkészítése, és a LED jelzés megvalósítása.

A projekt maga egy parkolóházi beléptetőrendszer egy szektorának kapujában lenne használatos, és eldöntené, hogy a beparkolni kívánó autónak van-e jogosultsága megtenni ezt, avagy sem.

Már ezen fázis legelején világos volt, hogy valamiféle SQL adatbázis használatával szeretnénk megvalósítani a projektet, de azt, hogy pontosan milyennel csak később döntöttük el.

Az alap ötlet úgy nézett ki, hogy a László által készített rendszámfelismerő scriptet kell majd futtatni a program működéséhez, ez pedig hívja majd az összehasonlító scriptet, és a statisztika készítést is.

A munka ezek mentén kezdődött el, és ment végbe, míg végül a két kész felet olvasztottuk egybe egy közös programozó sessionben.

Az első kérdés az volt, hogy milyen adatbázis fusson a projekt alatt, MySql, vagy MsSql? Rövid érvek és ellenérvek gyűjtögetése után végül a MySql adatbázis mellett döntöttünk, hiszen azt többet használtuk, jobban- és régebb óta is merjük az MsSql-nél.

A következő kérdés ebből kifolyólag az volt, hogy mit tároljunk az adatbázisban? Mindenképp tárolnunk kellett a rendszámokat, hogy milyen szektorba szól a jegyük, illetve a jegy megvételének és lejáratának dátumát, így lejárat esetén értesítést is tudnánk küldeni. Mivel tervben volt egy statisztikai adatvizualizáció megvalósítása is, ezért a belépési kísérleteket logolni is szerettünk volna, mégpedig, hogy ki akar belépni hova, mikor, és sikerült-e neki. Ezek alapján később el is készült az említett statisztikai adatvizualizáció.

3. Felmerülő problémák

A fejlesztés során értelemszerűen több problémába is ütköztünk, ebben a pontban pedig bemutatunk párat.

3.1. Jegyek, bérletek

Az első bökkenő az volt, hogy ha az eredeti jegyalapú belépési rendszert használnánk, akkor nehézkes lenne a tesztelés, hiszen senki nem vesz 3 hétre parkolójegyet, esetleg néhány órára. Nem beszélve arról, hogy ebben az esetben a projekt bemutatása előtt fel kellett volna újra töltenünk az adatbázist friss adatokkal, hogy ne legyen az összes jegy lejárva. Ezért mondtunk le végül a jegyes rendszerről, és ehelyett egy bérletes alternatívát találtunk ki. Itt egyszerűen egy boolean változó reprezentálja, hogy az adott autónak van-e érvényes bérlete, avagy nincs, így az adatok relevanciája nem múlik el idővel.

Ez azt jelentette, hogy az eredeti ötletet, miszerint a jegy lejárata előtt körülbelül 15 perccel küldenénk e-mail értesítést az autó tulajdonosának erről sem volt használható. Ehelyett arról kapnak értesítést a tulajok, hogy ha be szerettek volna parkolni egy szektorba, de nem tudtak bejutni annak két oka lehet. Az első, hogy nincs érvényes bérletük, a második pedig, hogy nem ebbe a szektorba szól a bérlet. Egy erről szóló e-mailt kapnak az autótulajdonosok minden sikertelen belépési kísérletnél. Utólag belegondolva a jegyes rendszer amiatt sem lett volna a legmegfelelőbb, mert ha pont rosszul jönnek ki az időtartamok, akkor meglehet, hogy bemutatás közben 5-10 percet is várni kell, hogy megérkezzen az értesítő e-mail, mert egyszerűen még nem telt el annyi idő, hogy ez megtörténjen.

3.2. E-mail küldés

A következő, szorosan az előzőhöz kapcsolódó nehézség maga az e-mail küldése volt. Készítettem egy Gmail-es e-mail címet az üzenetküldő botnak, mert emlékeztem, hogy a Google fiók beállításai közt volt egy checkbox, amivel meg lehetett engedni, hogy nem hitelesített alkalmazások (ilyenek például az otthon fejlesztett appok is, mint ez a projekt) is hozzáférhessenek, és használhassák azt a fiókot.

Amit nem tudtam viszont, hogy ez a jelölőnégyzet, biztonsági sérülékenységre hivatkozva a közelmúltban eltávolításra került, így egy kissé bonyolultabb, verifikációs folyamaton kell időnként átesnie a hitelesítetlen alkalmazásoknak egy Google fiók használatához, amely a felhasználói mappán belüli ’.credentials’ mappába tölt le egy ’.json’ fájlt a hitelesítéshez. A fejlesztési idő alatt háromszor is kellett hitelesítenem az appot, hogy képes legyen e-mail-eket küldeni, így abban tudunk reménykedni, hogy nem épp a bemutató közben történik a negyedik.

A működőképesség eléréséhez létre kell hozni egy projektet Google-ön belül, ezen belül megadni, hogy pontosan mihez férhessen hozzá az alkalmazás, majd le

kell tölteni a ’client_secret.json’ megnevezésű és kiterjesztésű fájlt, amely minden alkalmazás esetén egyedi, és ezt a projekt mappájába kell helyezni. Innentől kezdve történhet meg az e-mail-ek küldése egy Gmail címen keresztül. Az Általam használt scripthez három Google library-t kellett telepíteni, és kell frissen tartani, hiszen, ha úgy szeretnénk használni ezt a funkciót, hogy nem a legfrissebb verzió van telepítve a Google autentikációs library-kből, akkor egyszerűen nem fog működni. Ez is egy olyan dolog, amiről reménykedni lehet, hogy nem pont a bemutató előtt jön ki egy új frissítés.

3.3. Pi problémák

A harmadik nehézség, amibe ütköztünk a projekt végén, hogy a Raspberry-re való másolás után könyvtárak közötti kompatibilitási problémák merültek fel. Két különböző library ugyan azon harmadik könyvtárra épített, de a kettő különböző verziószámmal nem volt hajlandó együttműködni, így hosszú bogarászás után találtunk egy olyan verziót, amelyet mindkettő elfogad.

A másik probléma a pi-vel az volt, hogy hordozható kijelző hiányában a mappába lementett diagramokat (adatvizualizáció) csak egy, a pi vezérlésére használt laptopon tudjuk megjeleníteni, viszont mivel a MacOSX nem engedélyez külső forrásból SSH-n, vagy FTP-n keresztül való fájltovábbítást, ezért azt a megoldást találtuk, hogy a diagramokat elkészülés után egy pendrive-ra másoljuk, amit ezután a Mac-re csatlakoztatva megtekinthetjük a képeket.

3.4. Rendszámfelismerő nehézségek

Kezdetekben a Tesseract OCR nevezetű ingyenes elérhető offline szövegfelismerő került felhasználásra, viszont ez nem bizonyult megfelelőnek. Nagyon érzékeny volt a beviteli képre, mivel, ha nem megfelelő volt a felbontása vagy a színe, akkor nem ismerte fel. Ezért használni kellett mellé egy opencv nevezetű computer vision library. Ez azt a célt szolgálta, hogy a képet átalakította szürke árnyalatosra és kivágta csak a rendszámtáblát. Ez probléma mentesen működött, de a Tesseract OCR ugyan arra a rendszámra különböző felismeréseket eredményezett. Következő lépcsőfok az Easy OCR volt ami hasonlít a Tesseracthoz, de a használatát nem tudtam megoldani, mivel videókártya szükséges a használatához. Torchvision és Pytorch, ami az alappilére a felismerőnek kompatibilitási és dependency (avagy ráépülési) gondok miatt nem volt használható. A harmadik lehetőség, amit megláttam az az OpenALPR nevezetű online rendszámfelismerő keretrendszer volt. Ennek hártánya az, hogy internet szükséges a használatához, valamint a közelmúltban egy nagyobb cég

felvásárolta ezért havi előfizetéshez kötött a használata. Ennek egy alternatívája a PlateRecognizer.com oldal, ami ugyan azt valósítja meg csak 2500 lekérdezés ingyenes és nem igényel semmiféle előfizetést.

4. Felhasznált elemek és könyvtárak

A projekt elkészítése során több külső kódrészletet, és könyvtárat is használtunk.

A használt könyvtárak a következőek: MariaDB az adatbázishoz való csatlakozás, és az azzal való kommunikáció működéséhez, Dateutil a statisztikák időben való lebontásához, Beautifulsoup az értesítő e-mailek szerkesztéséhez, valamit a Maptlotlib a statisztikai diagramok készítéséhez.

Ezek mellett a G-mail működéséhez említett három Google library pedig a Google API Python Client, a google-auth-httplib2, ami egy httplib2 transport a Google Auth-hoz, illetve a google-auth-oauthlib, ami pedig egy oauthlib integráció a Google Auth-hoz, és még egy ’oauth2client’ library-t is telepíteni kellett.

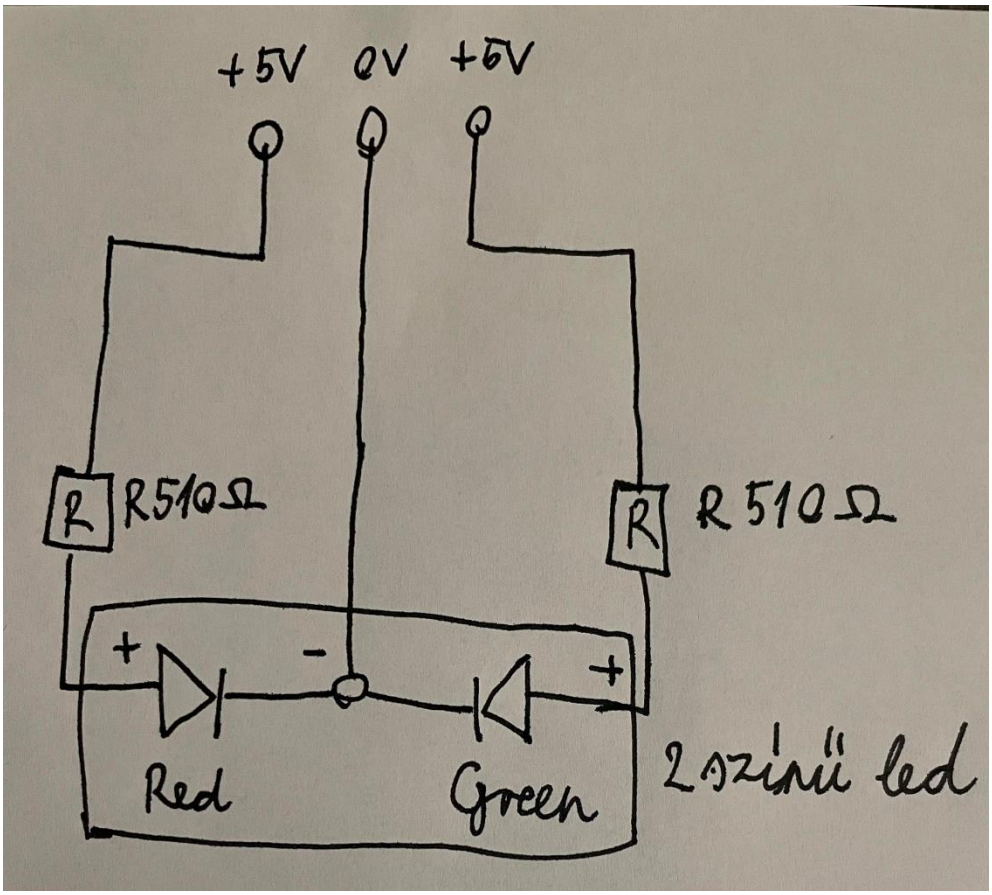
A kamera használatához szükséges telepíteni a ’fswebcam’ libraryt, valamint a weboldallal való kommunikációhoz a ’requests’ libraryt.

5. Rendszerkövetelmények

A rendszer működéséhez nem kell nagy számítási kapacitás, hiszen egy Raspberry PI 4-en is elfut, viszont szükség van hozzá internetelérésre az e-mail-ek küldéséhez, és a rendszámfelismeréshez, futtatni kell egy MySQL szervert a rendszámok, bérletek és logok tárolásához, továbbá kell egy kamera, amellyel a rendszámokat felismerhetjük. A kamerának USB-vel kell rendelkeznie és driver nélkül is működőképesnek kell lennie. A Raspberry pi rendszerének telepítéséhez elengedhetetlen egy SD kártya is, ami minimum Application Speed rating A1-el rendelkezik. Az áramkör megépítéséhez szükséges egy háromlábú kétszínű led, vezeték, 2 darab 510 ohm ellenállás és próbapanel.

6. Hardver specifikáció és áramköri rajz

Raspberry-pi : CPU: 1.5 GHz quad-core A72 64-bit 2Gb memória. 2db USB 2.0 2db USB 3.0 2db micro hdmi, 1 db USB-C, 1 db ethernet RJ45 1db 3.5mm jack. Ezek közül szükséges 2 db USB 2.0 és 1db USB-c Drivermentes kamera, 720p felbontással.



7. Szoftver specifikáció

A szoftvert kétféleképpen lehet működtetni, amelyek különböző scriptek elindítását igénylik.

Az egyik a rendszámfelismerő mód, amelyhez a ’plate_main.py’ nevű scriptet kell elindítani, ami képet készít a csatlakoztatott kamerával, felismeri a rendszámot, eldönti bejöhet-e az autó, és logolja a kísérletet, valamit jogosulatlan kísérlet esetén értesítő e-mailt küld, ha a rendszám megtalálható a rendszerben.

A második móddal a statisztikákat tudjuk elkészíteni, mert gondoltam, hogy minden kísérlet után újra generálni a diagramokat csak feleslegesen von el erőforrásokat a rendszertől. A ’stats_main.py’ scriptet futtatva egy kérdést kapunk, miszerint az elmúlt hétre, hónapra, vagy évre levetítve szeretnénk statisztikát kapni a parkolási kísérletekről. Az opció kiválasztása után a kód négy diagramot generál a ’charts’ mappába. Ezek közül kettő a sikertelen, és kettő a sikeres belépéseket reprezentálja, amik közül egy-egy dátum alapján bontott vonaldiagram, a maradék kettő pedig oszlopdiagram a szektorokba való sikeres, illetve sikertelen belépési kísérletek bemutatására.

8. Fejlesztett kód részletezése

Ha már nagy vonalakban ismerjük a működést, nézzük meg ezt kódra lebontva is. A következő néhány pontban a projektben található scriptek működésébe tekintünk bele mind alapvető cél, mint működési mechanizmus szintjén. Jellemző az egész projektre, hogy igyekeztünk úgy gyűjteni külön a kódrészleteket, hogy az egy fájlban lévők ugyan arra vonatkozzanak, pl. a ’dbconnect.py’-ban az adatbázishoz való kapcsolódás valósul meg.

8.1. plate_main.py

Az első funkcionalitás használata érdekében kell ezt a scriptet futtatni. Itt a ’plate.py’ adja át a felismert rendszámot ennek a scriptnek, majd az adatbázishoz való csatlakozás után ellenőrzi, hogy az autó bejöhet-e. Különböző okok miatt is megtagadhatjuk a belépést, így ezek külön is vannak választva. Először megnézi, hogy megtalálható-e a rendszerben a rendszám, ha nem, rögtön kivétel történik, és nem engedjük be az illetőt, és a LED pirosan villog, nagy sebességgel.

Ha benn van, akkor következő lépésként megnézzünk, hogy van-e érvényes bérlete. Ha nincs, az előzőhöz hasonlóan kivétel történik, az autó nem jöhet be, de mivel a rendszerben megtalálható, ezért a tulajdonos e-mail címe is benne van. Erre a címre küldünk egy e-mail a megtagadás okáról (érvénytelen bérlet), és a LED pirosan villog, az előzőnél lassabban.

Ha rendelkezik érvényes bérlettel, lekérdezzük, hogy a bérlet arra a szektorra szól-e, amelyiken ez a kamera van (ez a scriptben állítható). Ha nem, ismét egy kivétel keletkezik, nem engedjük be az autót, egy a tulaj e-mailt kap arról, hogy nincs erre a szektorra érvényes bérlete, és a LED pirosan világít.

A mindhárom próbán átment a rendszám, a képzeletbeli sorompó felnyílik, és a LED zölden világít; az autó bemehet.

8.2. dbconnect.py

Ahogy a bevezetőben is szó volt róla, a ’dbconnect.py’-ban az csak adatbázishoz való kapcsolódás valósul meg, így ez összesen nyolc sor hosszú csak.

Az első sorban a MaraDB szerverhez való csatlakozáshoz szükséges MariaDB-t importáljuk, amelyet a ’connecttodb’ függvény használ fel a csatlakozásra.

8.3. plate_actions.py

Ebben a scriptben találhatóak azok a függvények, amelyek használatával a ’plate_main.py’ eldönti, hogy az autó benn van-e a rendszerben, érvényes bérlete van-e, és jó szektorban van-e az érvényes bérletével. Ezen kívül egy negyedik függvény is található itt, ami visszaadja a rendszámhoz tartozó e-mail címet az e-mail-es értesítések küldéséhez.

8.4. plate.py

Ez a kód a rendszámfelismerő weboldallal létesít kapcsolatot, amely során átad a szervernek egy rendszámot tartalmazó képet és a kiegészítő paramétereket. A felhasznál paraméter jelen esetben a rendszám nemzetiségének kiválasztása (Hungary). A válaszul kapott json állományból levágja a felismert rendszámot tartalmazó stringet, amit továbbad a meghívó plate_main.py scriptnek.

8.5. led.py

Ez a kódrészlet tartalmaz 4 darab függvény: green_light, red_light, red_slow, red_fast. Nevükből adódóan világításra bírják a külső ledet a rendszám hitelesítése alapján, ami a plate_main.py-ban történik meg.

8.6. send_email.py

Itt található az e-mail küldés maga, amit szintén a ’plate_main.py’-ból hívunk meg. Minden e-mail küldéssel kapcsolatos függvény itt található, beleértve az e-mail-ek megtagadási ok miatti személyre szabását is. A felmerülő problémák, é-mail küldés alpontjában erről a scriptről is volt szó, hiszen itt történik az autentikáció is a Gmail használatához.

8.7. detection_actions.py

Ez a script felelős a logok felvételéért az adatbázisba, emiatt a ’connecttodb.py’-hoz hasonlóan itt is csak egy függvény található, amely az adatbázis második ’detections’ táblájába szúr be adatokat minden parkolási kísérletnél.

8.8. stats_main.py

Ez a második funkcionalitás megvalósításának, a statisztika készítésnek a main scriptje. Először itt is adatbázishoz való kapcsolódás történik, majd a program megkérdezi, hogy a statisztikát az elmúlt egy hét, egy hónap, vagy egy év

adataiból szeretnénk készíteni, majd, ha ezt megadtuk meghívja a következő pontban kifejtett fájlban található függvényeket, amik segítségével generálja a diagramokat.

8.9. stat_actions.py

Ahogy az előző pontban is írva volt, ebben a fájlban találhatóak azok a függvények, amelyek a statisztikát valóban elkészítik matplotlib segítségével, és lementik a ’charts’ mappába egy ’.png’ kiterjesztésű képként, így azok később megtekinthetőek. Összesen négy diagramot készít, kettőt-kettőt sikeres és sikertelen belépési kísérletekről, és ezeken belül egyet-egyet szektor és dátum szerint lebontva. Figyelni kell viszont, hogy ha egy verziót meg szeretnénk tartani, akkor ki kell másolni ebből a mappából a következő statisztika lefuttatása előtt, mivel helytakarékossági okok miatt egy lefutáskor az előző futás eredményei (a diagramok) felülíródnak az újakkal. Ez a veszély nem áll fenn, ha a szükséges képeke kimásoljuk egy másik mappába futtatás előtt.

9. A kész projekt

Összességében a projekt tökéletes volt arra, hogy megismerjük a python működését egy fizikai összeköttetést megvalósító közegben. Sokat tanultunk a Raspberry külső könyvtárak és a python működéséről és úgy gondoljuk, ha az élet egy hasonló kihívás elé állít, akkor könnyebben fogjuk tudni teljesíteni azt.