ÓBUDAI EGYETEM
ÓBUDA UNIVERSITY

KANDÓ KÁLMÁN VILLAMOSMÉRNÖKI KAR
ELEKTRONIKAI ÉS KOMMUNIKÁCIÓS RENDSZEREK INTÉZET
MŰSZERTECHNIKAI ÉS AUTOMATIZÁLÁSI TANSZÉK

SZAKDOLGOZAT

OE-KVK
2023.

Orbán Tamás
T008702/FI12904/K

ÓBUDAI EGYETEM
ÓBUDA UNIVERSITY

Óbudai Egyetem
Kandó Kálmán Villamosmérnöki Kar
Műszertechnikai és Automatizálási Intézet

SZAKDOLGOZAT FELADATLAP

Hallgató neve: Orbán Tamás
Szakdolgozat száma: SZD2309040923461336[redacted]
Törzskönyvi száma: T008702/FI12904/K **Neptun kódja:** [redacted]

Szak: villamosmérnöki
Specializáció: Műszer-automatika specializáció

A dolgozat címe: Okos teremgarázs - Automata helykiosztással
A dolgozat címe angolul: Smart car park
A feladat részletezése: Tervezzen és valósítson meg okos teremgarázs rendszert automata helykiosztással.

Intézményi konzulens neve: Borsos Döniz

A kiadott téma elévülési határideje:2025. 12. 31.
Beadási határidő: 2023. 12. 15.

A szakdolgozat: Nem titkos.

Kiadva: Budapest, 2023. 10. 24.

P.H

Intézetigazgató

A dolgozatot beadásra alkalmasnak találom:

belső konzulens

külső konzulens

ÓBUDAI EGYETEM
ÓBUDA UNIVERSITY

KANDÓ KÁLMÁN VILLAMOSMÉRNÖKI KAR
ELEKTRONIKAI ÉS KOMMUNIKÁCIÓS RENDSZEREK INTÉZET
MŰSZERTECHNIKAI ÉS AUTOMATIZÁLÁSI TANSZÉK

HALLGATÓI NYILATKOZAT

Alulírott hallgató kijelentem, hogy a szakdolgozat saját munkám eredménye, a felhasznált szakirodalmat és eszközöket azonosíthatóan közöltem. Az elkészült szakdolgozatban található eredményeket az egyetem és a feladatot kiíró intézmény saját céljára térítés nélkül felhasználhatja, a titkosításra vonatkozó esetleges megkötések mellett.

Budapest, 20.23.12.15.....

.....
hallgató aláírása

TARTALOM

I. BEVEZETÉS

A szakdolgozatom célja egy olyan univerzális parkoló rendszer létrehozása, amelyet bármekkora méretű és elrendezésű helyre telepíteni lehet, használata lerövidítheti a belépés és a parkolóhely tényleges elfoglalása között eltelt időt. Lényege, hogy belépéskor kapott jegy, kimondottan egy parkolóhoz szól, máshová nem engedélyezett állni, azaz nem kell parkolóhelyet keresni.

A téma kiválasztásánál nagyon fontosnak találtam, hogy egy komplexebb feladatot próbáljak meg teljesíteni, mivel nem úgy tekintek a szakdolgozatra, mintha csak magam után szeretném tudni, hanem tanulni, fejlődni szeretnék általa. .A programok megírásához NET keretrendszerre épülő C# programozási nyelvet választottam. Mindig is kíváncsi voltam, milyen egy magasabb szintű nyelv használata, erre a tanulmányain során nagyon kevés lehetőség adatott.

Komoly kihívásokkal kellett szembenéznem, hiszen hat különböző alkalmazás összehangolására volt szükség. A szerver alkalmazás fejlesztésével kezdtem meg a munkálatokat, amely során törekedtem a program ésszerű felépítésére és a megírt fájlok újra hasznosíthatóságára, a többi alkalmazás fejlesztésés megkönnyítve. Ezt sikerült is kivítelezni, hiszem az összes program majdnem teljesen ugyan azokat a fájlokat használja a kommunikácóra. Ezek megírásához már csak a logikát kellett implementálni, hogy melyik ünezet küldheti, és fogadhatja az adott kliens.

Sokkal mélyebb betekintést nyertem az objektum-orientált programfejlesztés világába, úgy érzem sokat fejlődtem az elkészítése alatt

II. SPECIFIKÁCIÓ

Fő funkciói

1. Belépéskor kapott jegy, az egy adott parkolóhelyhez szól, amely garantáltan szabad.

2. Választható pozíció, amelyhez közeli parkolót szeretnénk igényelni

3. Perc alapú díjszámítás

1., Funkció implementálásának feltétele: Tudni kell, hogy egy adott parkoló éppen szabad, vagy sem, ezt szenzorok segítségével egyszerűen implementálhatjuk. Ezt a célt szolgálják a Hub-ok és a hozzájuk csatlakoztatott szenzorok.

2., Funkció implementálásának feltétele: Ismernünk kell minden egyes parkolóhely pontos elhelyezkedését, telepítéskor ezeket pontosan be kell állítani.

3., Funkció implementálásának feltétele: Belépési időpont pontos rögzítése

Hátránya: Aktív monitorozást igényel, mivel előfordulhat, hogy valaki, nem a neki kiosztott helyet foglalta el, ekkor figyelmeztetni kell, ha a probléma továbbra is fennáll, akkor pedig fellépni ellene.

Limitációk: Ha egy olyan parkolóhelyre álltak be illetéktelenül, amelyet szintén kiadtak már, viszont nincs elfoglalva, akkor a rendszer ezt nem tudja érzékelni

Felépítés

Hub

Minden parkolóhelyhez tartozik egy szenzor, amely érzékeli, hogy elfoglalt, vagy szabad státuszú-e. Ezen szenzorok **összefoglalását hálózatra kapcsolását a hub** oldja meg. Ezekből tetszőleges számú eszköz csatlakoztatható a szerverhez (skálázhatóság).

Belépési pont

Belépéskor ezen keresztül igényelhető a jegy, választható rajta előre meghatározott pozícióhoz közeli hely kérvényezése pl.: mozihoz, éttermekhez. A szervernek elküldi a kérelmet, majd ha van szabad parkolóhely, akkor kiadja a generált jegyet és beenged a parkolóház területére.

Kilépési pont

Jegy azonosítóját elküldi a szervernek, amely eldönti, hogy kiléphet-e a parkolóház területéről, vagy sem.

Jegyautomata

Jegy azonosító alapján lekérdezi az árát, majd fizetést kezdeményezhetünk, amelynek sikerességéről tájékoztatja a szervert.

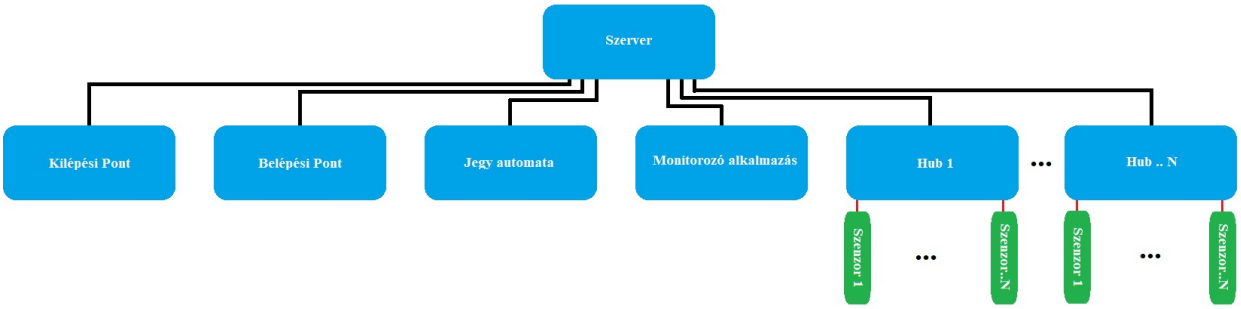
Szerver

Telepítés során konfigurálni kell, minden egyes parkolóhely pozícióját, méretét, valamint hub és a hubon belüli azonosításra használt parkolóhely azonosítóját.

- A szerver feladatai
- Jegyek (parkolóhelyek) kiosztása
- Jegy árának kiszámítása
- Jegyhez tartozó fizetési eredmények tárolása
- Kilépés engedélyezése vagy tiltása
- Monitorozó alkalmazásnak a parkoló rendszer adatainak kiküldése és szinkronban tartása az aktuális állapottal.

Monitorozó rendszer:

Ennek az alkalmazásnak a használatával kapunk élő képet a parkolóház aktuális állapotáról. Észlelhetjük, ha valaki tilos helyre parkolt és felléphetünk ellene. Ezek mellett az adatbázisban szereplő minden jegyre vonatkozó lekérdezést hajthatunk végre.



1. Ábra Rendszer felépítése. Fekete vonalak socket-ek közötti TCP/IP kapcsolatot jelent, míg a piros vonalak fizikálist.

Természetesen ennek fizikai megvalósítása túllépi a szakdolgozatra rendelkezésre álló időmet és költségkeretemet, ezért a hub-ok, szenzorok, belépési pont, jegy automata, kilépési pont csak szimuláció formájában készült el, és az egész rendszer csak **lokális hálózaton működik.**

III. FEJLESZTŐI DOKUMENTÁCIÓ

Smart Parking Server

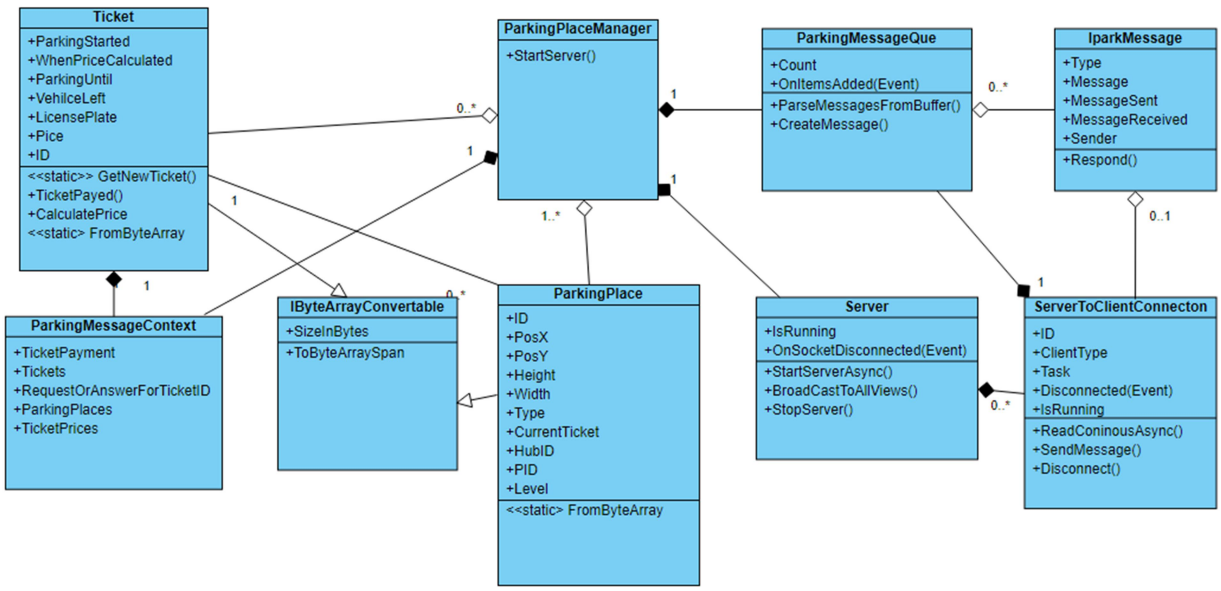
Ez a program felelős minden kliens fogadásáért: hub, jegyautomata, belépési pont, kilépési pont és monitoring alkalmazás. Ezekkel socketen (TCP/IP kapcsolaton) keresztül kommunikál, értelmezi a beérkezett üzeneteket (IParkMessage), végre hajtja az ezekben lévő kérelmeket, majd válaszol rájuk. Ha változás lépett fel a parkolóház állapotában, akkor értesíti a monitoring alkalmazást erről.

TargetFramework: .NET 8.0 C#12

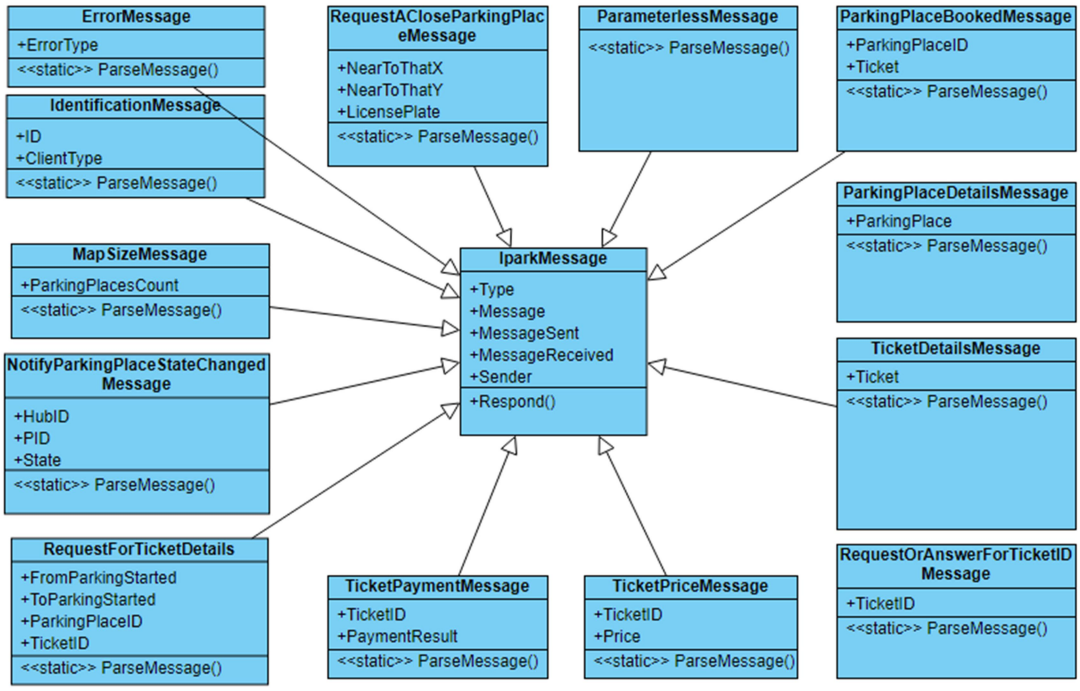
Applikáció típusa: Windows Console Application

Szükséges NuGet Packages:

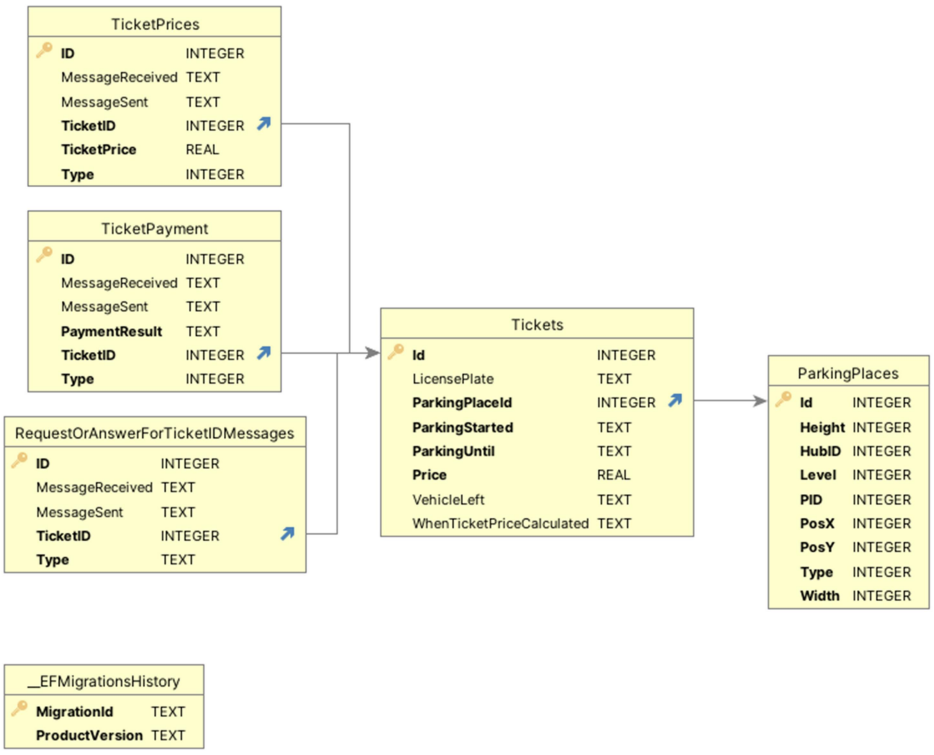
- Microsoft.EntityFrameWorksCore
- Microsoft.EntityFrameWorkCore.Sqlite
- Microsoft.EntityFrameWorkCore.Tools



2. Ábra Szerver UML-Class Diagramja



3. Ábra IParkMessageInterfacet megvalósító üzenetek



4. Ábra Adatbázis séma diagramja

IParkingMessage Interfész

Minden egyes üzenetnek implementálnia kell ezt az interfészt. Ez határozza meg, az felépítését.

```
public ParkingMessageType Type { get; }
```

Egy enumerátor, amelyet a IParkMessage.cs tartalmaz. Ennek segítségével lehetséges az üzenet visszafejtése bájt sorozatból.

```
public byte[] Message { get; }
```

Az üzenetet tartalmazza bájt sorozattá konvertálva

```
public DateTime? MessageSent { get; set; }
```

Megtévesztő lehet az elnevezése, amikor egy **üzenet érkezett**, akkor ennek az értéke **null,** viszont ha mi küldjük el az üzenetet, akkor az elküldés pillanatában beállítjuk az értékét (adatbázisban szerepeljen, hogy mikor lett a válasz elküldve).

```
public DateTime? MessageReceived { get; }
```

Amikor egy üzenetet visszafejtünk, akkor kell beállítani az aktuális időpontra. Lehessen nyomon követni, hogy mikor kaptunk meg egy adott üzenetet.

```
public ServerToClientConnection? Sender { get; }
```

Ha üzenet érkezett, be kell állítani, hogy legyen referenciánk a küldő kapcsolatra, tudjunk válaszolni.

```
public void Respond();
```

Itt lehet definiálni előre meghatározott választ egy üzenetre, jelenleg nem implementálja egyetlen osztály sem.

```
public void Respond(IParkingMessage message);
```

Paraméter: `IParkingMessage message` – A válasz üzenet

Funkciója: Ennek a függvénynek meghívásával lehet válaszolni az adott üzenetre.

Azon osztályok leírásánál, amelyek implementálják ezt az interfészt, ezeket a tulajdonságokat nem fogom részletezni, mivel funkciójuk minden esetben ugyan az. Kivétel `public byte[] Message { get; }`, ahol megtalálható lesz az adott üzenet formátuma byte-sorozatban.

ParkingMessageType Enum

Byte típusból van származtatva, tehát 0..255 közötti értékeket vehet fel. Ezzel az enumerátorral azonosítjuk az üzenetek típusát, ha új üzenetet szeretnénk implementálni, akkor itt regisztrálni kell.

- FullMapRequest = 1,
  Az összes parkolóhely lekérdezése
- AllTicketsHaveBeenSent=2
  Ha el lett küldve az összes lekérdezésnek megfelelt jegy, akkor ezt az üzenetet kell elküldeni lezárásképp.
- RequestACloseParkingPlace = 3
  Beléptető ponttól érkezik, egy parkolóhelyet igényel, adott pozícióhoz
- NotifyParkingPlaceStateChanged = 4
  Értesítés, ha egy parkolóhely állapota megváltozott.
- FirstMessageIdentification = 5
  Minden kliens csatlakozásnál ennek az üzenetnek kell lennie az elsőnek, amelyben azonosítja magát.
- TicketPrice = 6
  Szerver küldi, a jegy automatának, az adott jegyhez tartozó árat
- MapSize=7
  A szerver válasza FullMapRequestre első üzenetként, parkolóhelyek számát tartalmazza.
- ParkingPlaceBooked=8
  Ha egy parkolóhely ki lett adva, akkor annak a részleteit küldi el a monitoring alkalmazásnak.

- Error = 9
  Szerver küldi válaszként, ha valamely kérelem során hiba lépett fel

- ParkingPlaceDetails = 10
  Szerver küldi a monitoring alkalmazásnak, egy parkolóhelyhez köthető minden publikus adatot tartalmaz.

- TicketPayment = 11,
  Jegy automata küldi el a fizetés eredményét a szervernek.

- RequestTicketPrice = 12
  Jegy automata kérvényezi egy jegy árát.

- CanVehicleLeave = 13
  Kilépési pont kérdezi, hogy az adott jeggyel el lehet-e hagyni a parkolóházat

- VehicleCanLeave = 14,
  Szerver válaszol, ha az adott jeggyel ki lehet lépni

- VehicleCannotLeave=15
  Szerver válaszol, ha az adott jeggyel nem lehet kilépni

- TicketDetailsMessage = 16
  Egy jegyhez tartozó minden publikus adat elküldése a monitoring alkalmazásnak

- RequestForTicketDetailsMessage = 17
  Monitoring alkalmazás küldi, jegy adatok lekérdezése, szűrési feltételekkel az adatbázisból,

- UnknownMessage =255
  nem definiált üzenet, kivételkezeléshez használatos

FullMapRequest és **AllTicketsHaveBeenSent** értéke használatban van hiba ellenőrzésre, azaz ParameterlessMessage típusa nem lehet nagyobb mint AllTicketsHaveBeenSent és nem lehet kisebb mint FullMapRequest, tehát ha **ParameterlessMessage-t, használó üzenetet szeretnénk hozzáadni, akkor ezek közé kell azt megtenni**. Hasonlóan a **TicketPriceRequest** és **VehicleCannotLeave** közé kell beilleszteni az olyan üzeneteket, amelyek **RequestOrAnswerForTicketIDMessage**-ként kerülnek implementálásra.

ErrorMessage osztály

Implementálja az IParkMessage interfészt, hiba üzenetek küldésére-fogadására használatos.

```
public required byte[] Message { get;  init; }
```

```
Message[0] MessageType
Message[1] ErrorType
```

```
private static readonly int MessageSize = 2;
```

Az üzenet méretét tartalmazó konstans (bájtban):

```
public required ErrorType ErrorType { get;  init; }
```

A hibát definiáló tulajdonság, értékei a következő oldalon olvashatóak.

```
public static ErrorMessage? ParseMessage(byte[] buffer, int length, int startIndex,
ServerToClientConnection Sender)
```

Paraméterek:

- `byte[] buffer` – socket által beolvasott bájt sorozat.
- `int length` beolvasott bájtok száma
- `int startIndex` ettől az indextől kezdve kell értelmezni a buffer tartalmát.
- `ServerToClientConnection Sender` Az üzenetet küldő kapcsolat referenciája, válaszadásra legyen lehetőség.

Visszatérési érték: Ha az üzenetet sikerül rekonstruálni, akkor egy új ErrorMessage példánnyal térünk vissza, ha nem (bufferbe nem fért bele a teljes üzenet), akkor null.

Kivételek: WrongMessageFormatException –ha az ErrorType egyenlő ErrorTypeMinNotIncluded, vagy nagyobb-egyenlő ErrorTypeMaxNotIncluded

Funkció: Üzenet rekonstrukciója bájt tömbből. Üzenet fogadáskor kerül meghívásra.

ErrorType Enum

Byte típusból van származtatva, tehát 0..255 közötti értékeket vehet fel. Ez az enumerátor hordozza az ErrorMessage tartalmát, azaz milyen hiba történt.

- ErrorTypeMinNotIncluded=0,
  Hibadetektálásra használatos, ezt az értéket nem veheti fel az ErrorMessageType

- NoFreeparkingPlace = 1,
  Belépési pontnak adott válasz, ha a parkolóház tele van.

- NoTicketForTicketID=2,
  Bármely jeggyel kapcsolatos kérelemre adott válasz, ha az adott jegy nem szerepel az aktív jegyek között.

- NoTicketFoundForThatRequest=3,
  Monitorozó alkalmazásnak küldött válasz, ha az adott lekérdezési feltételeknek egyetlen jegy sem felel meg.

- ThisHubIDAlreadyConnected=4,
  IdentificationMessage-re adott válasz, ha egy hub olyan ID-vel szeretne csatlakozni, amely már kapcsolódott.

- ThisHubIsNotRegistered=5,
  IdentificationMessage-re adott válasz, ha egy hub olyan ID-vel szeretne csatlakozni, amely nem szerepel az adatbázisban

- PIDOutOfRange = 6,
  NotifyParkingPlaceStateChangedMessage-re adott válasz, ha PID nem szerepel az adatbázisban.

- SenderIDandHubIDNotTheSame=7,
  NotifyParkingPlaceStateChangedMessage adott válasz, ha a küldő ID és a változott állapotú parkolóhely HubID nem egyezik.

- TicketPriceNotBeenCalculated=8,
  Jegy automatának küldött üzenet, ha olyan jegyet szeretne kifizetni, amelyhez árat még nem kérdeztek le.

- ErrorTypeMaxNotIncluded =9
  Hibadetektálásra használatos, ezt az értéket nem veheti fel az ErrorMessageType

RequestForTicketDetials osztály

Implementálja az IParkMessage interfészt, ezek adattagok tulajdonságait a hozzátartozó fejezetben olvashatja.

Monitoring alkalmazás küldheti ezt az üzenetet, amennyiben jeggyel kapcsolatos információt szeretne kapni az adatbázisból.

Háromféle lekérdezés hajtható végre:

1. Csak jegy kiadásának időpontjára
2. Jegy kiadásának időpontjára és egy adott parkolóhelyre
3. Jegy ID-re

```
private const int MessageSize = 25;
```

Az üzenet méretét tartalmazó konstans (bájtban):

```
public required DateTime FromParkingStarted { get; init; }
```

Ha egy adott intervallumban szeretnénk keresni, akkor ez az érték, tartalmazza a jegy kiadási időpontjának az alsó határát.

```
public required DateTime ToParkingStarted { get; init; }
```

Ha egy adott intervallumban szeretnénk keresni, akkor ez az érték, tartalmazza a jegy kiadási időpontjának az felső határát.

```
public required int ParkingPlaceID { get; init; }
```

Ha egy adott parkolóhelyhez kiadott jegyekre szeretnénk keresni, akkor ez az értékét tartalmazza a parkolóhely ID-jét. A dátum határok ugyanúgy befolyásolják a keresést. Ha nem szeretnénk ID szerint szűrni, akkor ezt az értéket 1-nél kisebb értékre kell állítani.

```
public required int TicketID { get; init; }
```

Ha egy adott jegyre ID szerint szeretnénk keresni, akkor ennek az értékét állítsuk 1-nél nagyobb értékre, ilyenkor a dátumhatárok nincsenek figyelembe véve.

```
public required byte[] Message { get; init; }

Message[0] MesageType

Message[1..8] FromParkingStarted (long)

Message[9..16] ToParkingStarted (long)

Message[17..20] ParkingPlaceID (int)

Message[21..24] TicketID (int)
```

```
private void GenerateMessage()

{
  Message[0] = (byte)Type;
  Span<byte> msg = new(Message);
  int indexer = 1;
  BitConverter.TryWriteBytes(msg[indexer..], FromParkingStarted.Ticks);
  indexer += 8;
  BitConverter.TryWriteBytes(msg[indexer..], ToParkingStarted.Ticks);
  indexer += 8;
  BitConverter.TryWriteBytes(msg[indexer..], ParkingPlaceID);
  indexer += 4;
  BitConverter.TryWriteBytes(msg[indexer..], TicketID);
}
```

Funkciója: Az üzenet bájt sorozatba való konverzióját végzi.

Működése: Konstruktorban már a tömböt létrehoztuk és a tulajdonságokat beállítottuk. Mivel C#-ban biztonságos környezetben nem használhatóak a mutatók (és pointer aritmetika), ezért kénytelenek vagyunk képezni egy Span<byte> -ot a Message-ből, amely segítségével létre tudunk hozni egy memória szeletet, a range operátor használatával: **msg[indexer..]**. Ezt alkalmazva végighaladunk az összes tulajdonságon és a BitConverter.TryWriteBytes függvénnyel indexer-től kezdődő pozícióba betöltjük az adattag bájt sorozatát, figyelve, hogy az indexer értékét mindig helyesen növeljük.

```
public static RequestForTicketDetails? ParseMessage(byte[] buffer, int length, int
startIndex, ServerToClientConnection Sender)
{
if (startIndex + MessageSize > length)
    return null;
byte[] msg = new byte[MessageSize];
Array.Copy(buffer, startIndex, msg, 0, MessageSize);
int indexer = 1;
long date_from = BitConverter.ToInt64(msg, indexer);
indexer += 8;
long date_to = BitConverter.ToInt64(msg, indexer);
indexer += 8;
int parkingID = BitConverter.ToInt32(msg, indexer);
```

```
indexer += 4;
int ticketID = BitConverter.ToInt32(msg, indexer);
RequestForTicketDetails Request = new()
{
  FromParkingStarted = new DateTime(date_from),
  ToParkingStarted = new DateTime(date_to),
  TicketID = ticketID,
  ParkingPlaceID = parkingID,
  Type = ParkingMessageType.RequestForTicketDetailsMessage,
  Sender = Sender,
  Message = msg,
  MessageReceived = DateTime.Now
};
    return Request;
}
```

Paraméterek:

- `byte[] buffer` – socket által beolvasott bájt sorozat.
- `int length` beolvasott bájtok száma
- `int startIndex` ettől az indextől kezdve kell értelmezni a buffer tartalmát.
- `ServerToClientConnection Sender` Az üzenetet küldő kapcsolat referenciája, válaszadásra legyen lehetőség.

Visszatérési érték: Ha az üzenetet sikerül rekonstruálni, akkor egy új RequestForTicketDetails példánnyal térünk vissza, ha nem (bufferbe nem fért bele a teljes üzenet), akkor null.

Funkció: Üzenet rekonstrukciója bájt tömbből.

Működése: Ellenőrizzük, hogy az adott üzenet belefér-e a bufferbe, ha nem, akkor null értékkel visszatér a függvény. Belemásoljuk a Message-be az üzenetet, majd egyesével BitConvert adott típushoz tartozó függvényeit meghívva kiolvassuk ezeket, utána pedig létrehozzunk egy példányt a RequestForTicketDetails osztályból, amellyel visszatérünk.

```
public RequestForTicketDetails(int ticketID)
```

Akkor használjuk ezt a konstruktort, ha csak ticketID szerint szeretnénk lekérdezést kérvényezni

```
public RequestForTicketDetails(DateTime fromParkingStarted, DateTime
toParkingStarted)
```

Akkor használjuk ezt a konstruktort, ha egy adott időintervallumra szeretnénk lekérdezést kérvényezni

```
public RequestForTicketDetails(DateTime fromParkingStarted, DateTime
toParkingStarted, int parkingPlaceID)
```

Akkor használjuk ezt a konstruktort, ha egy adott időintervallumra és parkolóhelyre szeretnénk lekérdezést kérvényezni

A konverzió menete minden osztálynál ugyan az, ezért azt többet nem fogom megemlíteni.

IdentificationMessage Osztály

Implementálja az IParkMessageInterface-t, ezek adattagok tulajdonságait a hozzátartozó fejezetben olvashatja.

Minden eszköznek csatlakozás után egy IdentificationMessage-t kell küldenie, **ha ez nem történik meg, akkor a szerver bontja a kapcsolatot**. Ennek segítségével lehet megkülönböztetni a klienseket.

```
public required int ID { get; init; }
```

Csatlakozott eszköz ID-je

```
public required ClientType ClientType
```

Csatlakozott kliens típusa, ennek definíciója a ServerToClientConnection-ben található

```
public required byte[] Message { get; init; }
```

```
Message[0] MessageType
```

```
Message[1] ClientType
```

```
Message[2..5] ClientID (int)
```

```
public static IdentificationMessage? ParseMessage(byte[] buffer, int length, int
startIndex, ServerToClientConnection Sender)
```

Funkciója és működése hasonló a RequestForTicketDetails ParseMessage függvényéhez, ezekről információ ott található.

Kivételek: WrongMessageFormatException –ha ClientType egyenlő ClientTypeMinNotIncluded, vagy nagyobb-egyenlő ClientTypeMaxNotIncluded.

MapSizeMessage Osztály:

Implementálja az IParkMessageInterface-t, ezek adattagok tulajdonságait a hozzátartozó fejezetben olvashatja.

FullMapRequest-re küldött válasz, csak a parkolóhelyek számát tartalmazza, ezáltal tudja majd a monitorozó szoftver, ha megérkezett az összes elem.

```
public required int ParkingPlacesCount { get; init; }
```

Parkolóhelyek száma

```
public required byte[] Message { get; init; }
```

```
Message[0] MessageType
```

```
Message[1..4] ParkingPlacesCount (int)
```

```
  public static MapSizeMessage? ParseMessage(byte[] buffer, int length, int
startIndex, ServerToClientConnection Sender)
```

Funkciója és működése hasonló a RequestForTicketDetails ParseMessage függvényéhez, további információ ott található.

Kivételek: WrongMessageFormatException ha a ParkingPlacesCount kisebb mint 0.

NotifyParkingPlaceStateChangedMessage Osztály:

Implementálja az IParkMessageInterface-t, ezek adattagok tulajdonságait a hozzátartozó fejezetben olvashatja.

Egy parkolóhely állapotának megváltozásának jelzésére szolgáló üzenet.

```
public required int HubID { get; init; }
```

Megváltozott státuszú parkolóhelyhez tartozó HubID

```
public required int PID { get; init; }
```

Megváltozott státuszú parkolóhelyhez tartozó PID-

```
public required ParkingPlaceState ParkingPlaceState { get; init; }
```

A parkolóhely új állapota

```
public required byte[] Message { get; init; }
```

```
Message[0] MessageType
```

```
Message[1..4] HubID (int)
```

```
Message[5..8] PID (int)
```

```
Message[9] ParkingPlaceState
```

```
public static NotifyParkingPlaceStateChangedMessage? ParseMessage(byte[] buffer, int length, int startIndex, ServerToClientConnection Sender)
```

Funkciója és működése hasonló a RequestForTicketDetails ParseMessage függvényéhez, további információ ott található.

Kivételek: WrongMessageFormatException –ha ParkingPlaceState egyenlő ParkingPlaceState-MinNotIncluded vagy nagyobb-egyenlő ParkingPlaceStateMaxNotInlcuded. Ezeket az értékeket nem veheti fel az üzenet.

ParameterlessMessage Osztály

Implementálja az IParkMessageInterface-t, ezek adattagok tulajdonságait a hozzátartozó fejezetben olvashatja.

Ez az osztály a típusán kívül más érdemi információt nem hordoz, paraméterek nélküli kérések vagy parancsok küldésére szolgál. Ezek lehetnek:

- FullMapRequest
- AllTicektsHasBeenSent

```
public required byte[] Message { get; init; }
```

```
Message[0] MessageType
```

```
public ParameterlessMesage(ParkingMessageType type)
```

Kivételek : Mivel létrehozásnál meg kell adni az üzenet típusát, ezért annak helyes értékére kivételesen figyelni kell. Ha kisebb, mint FullMapRequest vagy nagyobb, mint AllTicketsHaveBeenSent, akkor WrongFormatException keletkezik.

ParkingPlaceBookedMessage Osztály

Implementálja az IParkMessageInterface-t, ezek adattagok tulajdonságait a hozzátartozó fejezetben olvashatja.

Ha egy jegyet kiadott a rendszer, akkor ez az üzenet kerül elküldésre a monitoring programnak.

```
public required int ParkingPlaceID { get; init; }
```

Kibérelt parkolóhely ID

```
public required Ticket Ticket { get; init; }
```

Kibérelt parkolóhelyhez tartozó jegy.

```
public required byte[] Message { get; init; }
```

```
Message[0] MessageType
```

```
Message[1..4] ParkingPlaceID (int)
```

```
Message[5..TicketSize-1] Ticket
```

```
public static ParkingPlaceBookedMessage? ParseMessage(byte[] buffer, int length, int startIndex, ServerToClientConnection Sender)
```

Funkciója és működése hasonló a RequestForTicketDetails ParseMessage függvényéhez, további információ ott található.

ParkingPlaceDetailsMessage Osztály

Implementálja az IParkMessageInterface-t, ezek adattagok tulajdonságait a hozzátartozó fejezetben olvashatja.

Egy parkolóhelyhez tartozó publikus adatok elküldésére szolgál.

```
public required ParkingPlace ParkingPlace { get; init; }
```

A megváltozott parkolóhely

```
public required byte[] Message { get; init; }
```

```
Message[0] MessageType
```

```
Message[1..ParkingPlaceSize] ParkingPlace
```

Fontos megjegyezni, hogy a kibérelt parkolóhely mérete és a szabadé különböző, ezért a Message mérete fordítási időben nem meghatározható.

```
public static ParkingPlaceDetailsMessage? ParseMessage(byte[] buffer, int length, int startIndex, ServerToClientConnection Sender)
```

Funkciója és működése hasonló a RequestForTicketDetails ParseMessage függvényéhez, további információ ott található.

RequestOrAnswerForTicketIDMessage Osztály

Implementálja az IParkMessageInterface-t, ezek adattagok tulajdonságait a hozzátartozó fejezetben olvashatja.

Ez az üzenet akkor használatos, ha jegy azonosítón és üzenet típuson kívül semmilyen más információt nem kívánunk átküldeni. Ilyen üzenetek a : **RequestTicketPrice, CanVehicleLeave, VehicleCanLeave, VehicleCannotLeave**

```
public int ID { get; set; }
```

Ezt az üzenet elmentésre kerül az adatbázisban, ez a hozzá tartozó elsődleges kulcs, automatikusan generált.

```
public  Ticket? Ticket { get;  set; }
```

EntityFrameWork-höz szükséges, kapcsolatot definiálja az adatbázisban.

```
public required int TicketID { get; init; }
```

TicketID amihez a parancs vagy kérés tartozik.

```
[NotMapped]
public required byte[] Message { get; init; }

Message[0] ParkingessageType

Message[1..4] TicketID
```

Ebben az osztályban minden olyan tulajdonságot fel kell tüntenti [NotMapped] „DataAnnotations”-el, amit nem tárolunk az adatbázisban.

```
public RequestOrAnswerForTicketIDMessage(ParkingMessageType _type, Ticket _ticket)
```

Kivétel : Mivel létrehozásnál meg kell adni az üzenet típusát, ezért erre kivételesen figyelni kell. Ha kisebb, mint RequestTicketPrice vagy nagyobb mint VehicleCannotLeave, akkor Wrong-FormatException-t keletkezik.

```
public static RequestOrAnswerForTicketIDMessage? ParseMessage(byte[] buffer, int length, int startIndex, ServerToClientConnection Sender)
```

Funkciója és működése hasonló a RequestForTicketDetails ParseMessage függvényéhez, további információ ott található.

RequestACloseParkingPlace Osztály

Implementálja az IParkMessageInterface-t, ezek adattagok tulajdonságait a hozzátartozó fejezetben olvashatja.

Ezzel az üzenettel lehet közeli parkolóhelyet kérvényezni a szervertől.

```
public required int NearToThatX { get; init; }
```

Az a X dimenzióban lévő pozíció, amelyhez közeli parkolóhelyet szeretnének kérvényezni.

```
public required int NearToThatY { get; init; }
```

Az a Y dimenzióban lévő pozíció, amelyhez közeli parkolóhelyet szeretnének kérvényezni.

```
public required string? LicensePlate { get; init; }
```

Ha sikerült beazonosítani a rendszámot, akkor a kérvényező autó rendszáma (max 10 karakter)

```
public required byte[] Message { get; init ; }
```

```
Message[0] MessageType

Message[1..4] NearToThatX (int)

Mesasge[5..8] NearToThatY (int)

Message[9..12] LicensePlateLength(int)

Message[13..32] LicensePlate (max 10 char, 1 char == 2 bájt)
```

```
public RequestACloseParkingPlaceMessage(string? licensePlate, int posX, int posY)
```

Kivételek : ArgumentOutOfRangeException ha a megadott rendszám karaktereinek száma meghaladja a 10-et.

```
public static RequestACloseParkingPlaceMessage? ParseMessage(byte[] buffer, int
length, int startIndex, ServerToClientConnection Sender)
```

Funkciója és működése hasonló a RequestForTicketDetails ParseMessage függvényéhez, további információ ott található.

TicketDetailsMessage Osztály

Implementálja az IParkMessageInterface-t, ezek adattagok tulajdonságait a hozzátartozó fejezetben olvashatja.

Hasonlóan a ParkingPlaceDetailsMessage osztályhoz, egy adott jegy összes publikus tagját lehet elküldeni vele.

```
public required Ticket Ticket { get; init; }
```

A megváltozott jegyet tartalmazza

```
public required byte[] Message { get; init; }
```

```
Message[0] ParkingMessageType
```

```
Message[1..TicketSize-1] Ticket
```

```
public static TicketDetailsMessage? ParseMessage(byte[] buffer, int length, int startIndex, ServerToClientConnection Sender)
```

Funkciója és működése hasonló a RequestForTicketDetails ParseMessage függvényéhez, további információ ott található.

TicketPaymentMessage Osztály

Implementálja az IParkMessageInterface-t, ezek adattagok tulajdonságait a hozzátartozó fejezetben olvashatja.

Egy jegyhez tartozó tranzakció fizetési tranzakció eredményét tartalmazza.

```
public int ID { get; set; }
```

EntityFrameWork-höz létrehozott tulajdonság, elsődleges kulcs az adatbázisban, automatikusan generált.

```
public  Ticket? Ticket { get; set; }
```

EntityFrameWork-höz létrehozott tulajdonság, hogy megteremtse a kapcsolatot az adatbázisban a jegy és TicketPayMentMessage között

```
[NotMapped]
public required byte[] Message { get; init;}

Message[0] ParkingMessageType

Message[1..4] TicketID (int)

Message[5] PaymentResult (byte)
```

```
public required int TicketID { get; init; }
```

A kifizetni kívánt jegyhez tartozó ID

```
public required PaymentResult PaymentResult { get; init; }
```

A fizetés eredményét tartalmazza, az enumerátor definíciója a következő fejezetben található.

```
public static TicketPaymentMessage? ParseMessage(byte[] buffer, int length, int
startIndex, ServerToClientConnection Sender)
```

Kivételek : WrongMessageFormatException ha a beolvasott PaymentResult értékei nem esnek a PaymentResultMinNotIncluded és PaymentResultMaxNotIncluded közé.

Funkciója és működése hasonló a RequestForTicketDetails ParseMessage függvényéhez, további információ ott található.

Ebben az osztályban minden olyan tulajdonságot fel kell tüntetni [NotMapped] „DataAnnotations”-el, amit nem tárolunk az adatbázisban.

PaymentResult Enumerátor

Szabályos működés során két értéket vehet fel, Succesful vagy Denied.

```
public enum PaymentResult : byte
  {
    PaymentResultMinNotIncluded = 0,
    Succesful = 1,
    Denied = 2,
    PaymentResultMaxNotIncluded = 3
    }
```

TicketPriceMessage osztály

Implementálja az IParkMessageInterface-t, ezek adattagok tulajdonságait a hozzátartozó fejezetben olvashatja.

Egy jegyhez azonosító és a hozzá tartozó ár elküldésére szolgál

```
public int ID { get; set; }
```

EntityFrameWork-höz létrehozott tulajdonság, elsődleges kulcs az adatbázisban, automatikusan generált.

```
public  Ticket? Ticket { get; set; }
```

EntityFrameWork-höz létrehozott tulajdonság, hogy megteremtse a kapcsolatot az adatbázisban.

```
public required int TicketID { get; init; }
```

Kalkulált árhoz tartozó TicketID

```
public required double TicketPrice { get; init; }
```

Kiszámított ár

```
[NotMapped]
public required byte[] Message { get; init; }

Message[0] ParkingMessageType

Message[1..4] TicketID

Message[5..12] TicketPrice (double)
```

```
public static TicketPriceMessage? ParseMessage(byte[] buffer, int length, int
startIndex, ServerToClientConnection Sender)
```

Funkciója és működése hasonló a RequestForTicketDetails ParseMessage függvényéhez, további információ ott található.

Ebben az osztályban minden olyan tulajdonságot fel kell tüntetni [NotMapped] „DataAnnotations”-el, amit nem tárolunk az adatbázisban.

ParkingMessageQue osztály

Ez az osztály hivatott biztosítani a kapcsolatot ServerToClient és a ParkingPlaceManager között, oly módon, hogy a ServerToClient által fogadott bájt sorozatot (buffer) tartalmát értelmezve létrehozza a megfelelő IParkingMessage-t implementáló üzenetet, amit aztán egy FIFO-ba helyez el és meghívja az lefuttatja az OnItemsAddedEventhandlerhez hozzáadott függvény(eke)t, azaz értesíti a Parking-PlaceManager-t, ha feliratkozott rá.

```
private readonly Queue<IParkingMessage> messages;
```

Ez a privát adattag tárolja a beérkezett üzeneteket.

```
private readonly object locker;
```

Ennek az adattagnak az a célja, hogy a program szál kritikus részeit biztonságossá tudjuk tenni, amelyet a C# natívan támogat a lock(locker){ } keyword segítségével.

```
public event EventHandler? ItemsAdded;
```

Erre feliratkozott függvények meghívása, ha a FIFO-ba üzenet került

```
public int Count { get { lock (locker) { return messages.Count; } } }
```

A sorban lévő elemek számát lehet lekérdezni.

```
public int ParseMessagesFromBuffer(byte[] buffer, int startIndex, int length,
ServerToClientConnection sender)
```

Visszatérési érték: Ha a bufferbe nem fért bele az egész üzenet, akkor visszatér az utolsó üzenet első bájtjának az indexével, ha sikerült mindent kiolvasni, akkor 0

Paraméterek:

- `byte[] buffer`– Beolvasott bájt halmaz, amit üzenetekké próbál alakítani.
- `int startIndex` Bufferben lévő bájtokat, innentől kezdve értelmezzük
- `ServerToClientConnection sender` Az adatot küldő klienshez kapcsolat

Funkciója: Bufferben található bájt sorozat üzenetekké alakítása, majd a FIFO-ba helyezése OnItemsAdded esemény lefuttatása.

Kivételek: WrongMessageException hogy ha valamely konverzió során értelmezhetetlen értékekkel szembesült.

Mellékhatások: Que-ba értékeket helyez.

```
private static IParkingMessage? CreateMessage(byte[] buffer, int startIndex, int
length, ServerToClientConnection sender);
```

Visszatérési érték: Ha sikeres volt az üzenet létrehozása, akkor visszatér az üzenettel, egyébként null

Paraméterek:

- `byte[] buffer`– Beolvasott bájt halmaz, amiből egy üzenetet próbál létrehozni
- `int startIndex` Bufferben lévő bájtokat, innentől kezdve értelmezzük
- `ServerToClientConnection sender` Az adatot küldő klienshez kapcsolat

Funkciója: Bufferben található bájt sorozatból megpróbálja létrehozni a következő üzenetet.

Kivételek: WrongMessageException hogy ha valamely konverzió során értelmezhetetlen értékekkel szembesült.

WrongMessageFormatException Osztály

Ha valamely IParkMessage-t implementáló osztály létrehozása során probléma merült fel, akkor ezt a típusú kivételt kapjuk. Örökli az Exception osztályt.

```
public ServerToClientConnection? Sender { get; }
```

Kapcsolat a küldőhöz, akitől az üzenet érkezett

```
public ParkingMessageType MessageType { get; }
```

Az üzenet típusa, amelyik a problémát okozta.

Server Osztály

Feladata szerverhez csatlakozni kívánt kliensek fogadása, majd ezeket kiszolgáló ServerToClient létrehozása és elindítása, külön szálon.

```
readonly IPAddress ipAddress;
```

Szerver socket ezen az IP-címen fog klienseket fogadni

```
const int Socket_Port = 55000;
```

Szerver socket ezen a port-on fog klienseket fogadni

```
readonly IPEndPoint ipEndPoint;
```

ipAddres és ipEndPoint kombinálva

```
readonly List<ServerToClientConnection> connectedSockets;
```

Szerverhez csatlakozott aktív kliensek listája

```
readonly Socket server;
```

Maga a Socket, ami fogadja a klienseket

```
readonly object locker = new ();
```

locker, hogy szál-biztos legyen az alkalmazás

```
readonly CancellationTokenSource cts;
```

CancellationToken-t ebből a TokenSource-ból lehet igényelni.

```
readonly CancellationToken cancellationToken;
```

Ennek segítségével tudjuk leállítani a szerver aszinkron funkcióit

```
readonly ParkingMessageQueue inComingMessageQueue;
```

ParkingMessageQueue amely a ServerToClientConnection-öknek lesz átadva paramtérként.

```
public bool IsRunning { get; set; }
```

Ez a tulajdonság mutatja, hogy éppen fut e a szerver.

```
public EventHandler<ServerToClientConnection>? ClientDisconnected;
```

Ha egy kliens lecsatlakozott, akkor ez az event fog lefutni.

```
private void OnSocketDisconnected(ServerToClientConnection connection)
```

Paraméter: ServerToClientConnection connection lekapcsolódott ServerToClientConnection

Funkció: ClientDisconnected esemény meghívása, ha a hozzá tartozó handler tartalmaz függvényeket.

```
private static IPAddress GetIPAddres()
```

Visszatérési érték: Kiválasztott IP cím, amely a kapcsolat létesítéséhez szükséges.

Funkció: Jelen esetben csak a localhost IP-címével tér vissza.

```
public async Task StartServerAsync()
```

Funkció: Szerver elindítása aszinkron módon, kliensek csatlakoztatása. Csatlakozott kliens esetén ServerToClient létrehozása, SocketDisconnected eseményre feliratkozás, majd a ServerToClientConnection ReadContinouslyAsync függvényének elindítása egy új szálon.

```
public void BroadCoastToAllViews(IParkingMessage msg)
```

Paraméter: `IParkingMessage msg` üzenet amit továbbítani szeretnénk.

Funkció: Paraméterként megkapott üzenetet továbbítja minden csatlakozott monitoring applikáció számára.

```
private void SocketDisconnected(object? sender, Socket disconnectedSocket)
```

Paraméterek:

- `object? Sender` Függvényt meghívó objektum
- `Socket disconnectedSocket` Lekapcsolódott Socket

Funkció: Ezt a függvényt adjuk át a ServerToClientConnection `ClientDiscConnected` eseményéhez, a connectedSockets listából töröljük az adott ServerToClientConnection-t, majd meghívjuk a Server `OnSocketDisconnected` függvényt, ez azért fontos, mivel a ParkingPlaceManager nem tud feliratkozni a ServerToClient connection eseményére.

```
public void StopServer()
```

Funckió: Szerver leállítása

ServerToClientConnection

Ha kliens csatlakozott onnantól kezdve ezen az osztályon keresztül folyik vele a kommunikáció.

```
readonly Socket client;
```

Kapcsolattartásra használt Socket

```
readonly CancellationTokenSource cts;
```

CancellationToken-t ebből a TokenSource-ból lehet igényelni.

```
readonly CancellationToken cancellationToken;
```

Ennek segítségével tudjuk leállítani a szerver aszinkron funkcióit

```
readonly ParkingMessageQueue pMessageQueue;
```

ParkingMessageQueue amely majd értelmezi a bufferban lévő bájt sorozatot

```
public Task? Task { get; set; }
```

A ServerToClientConnectionhoz tartozó Task.

```
public ClientType ClientType { get; private set; }
```

Csatlakozott kliens típusa

```
public int ID { get; private set; }
```

Csatlakozott kliens ID-je

```
public bool ISRunning { get; private set; }
```

Fogadunk-e üzeneteket a kliensről

```
public event EventHandler<Socket>? ClientDiscConnected;
```

Ha egy kliens lecsatlakozott, akkor ezt az eseményt hívjuk meg.

```
private void OnClientDisconnected(object sender,Socket disconnectedSocket)
```

Ez csak egy segítő funkció az esemény hívásához.

```
private async Task<int> GetFirstMesage(byte[] buffer)
```

Visszatérési érték: Ofszet, ha az utolsó üzenet nem fér bele a bufferbe, akkor ennek az üzenetnek a kezdeti indexével egyenlő, egyébként nulla.

Paraméter: `byte[] buffer` Ebbe a bufferbe olvassuk be a kliens által küldött bájt sorozatot

Funkció: Első üzenet értelmezése, hogy azonosítani tudjuk a csatlakozni kívánt klienst, ha nem IdentificationMessage-et kapunk, akkor a klienst azonnal leválasztjuk, egyébként megprobáljuk feldolgozni az üzenetet pMessageQueue használatával.

Mellékhatás: pMessageQueue ParseMessagesFromBuffer használata.

```
private async Task<int> GetMessageToQueueAsync(byte[] buffer, int offset)
```

Visszatérési érték: Ofszet, ha az utolsó üzenet nem fér bele a bufferbe, akkor ennek az üzenetnek a kezdeti indexével egyenlő, egyébként nulla.

Paraméterek:

- `byte[] buffer` Ebbe a bufferbe olvassuk be a kliens által küldött bájt sorozatot
- `int offset` Az előző beolvasások során kapott ofszet értéke.

Funkció: Ha ofszet értéke nem nulla, akkor ettől az indextől kezdve, a buffer tartalmát át kell helyezni a buffer elejébe, és az adatokat beolvasni csak ez után szabad. Beolvasott bájt sorozat továbbítása a ParseMessageQueue-nak, ha probléma lép fel az üzenetek feldolgozása során, akkor leválasztja a klienst.

Mellékhatás: pMessageQueue ParseMessagesFromBuffer használata.

```
public async Task ReadContinouslyAsync()
```

Funkció: Ezzel a függvénnyel indítható a buffer folyamatos olvasása.

```
public void SendMessage(IParkingMessage message)
```

Paraméter: `IParkingMessage message` elküldeni kívánt üzenet

Funkció: IParkingMessage elküldése a kliensek.

```
public void Disconnect()
```

Funkció: ServerToClientConnection leállítsa, kapcsolat bontása.

ClientType enumerátor

Kliensek megkülönböztetésére használt enumerátor.

View: Monitoring applikáció

Hub: Hubok

Barrier: Kiléptetési – Beléptetési pontok

TicketMachine Jegyautomata

```
public enum ClientType : byte
{
    ClientTypeMinNotIncluded = 0,
    View = 1,
    Hub = 2,
    Barrier = 3,
    TicketMachine = 4,
    ClientTypeMaxNotIncluded=5
}
```

ParkingMessageContext Osztály

Ez az osztály hivatott létrehozni a kapcsolatot az SQLite adatbázissal, majd biztosítja a lekérdezéseket LINQ segítségével, tehát úgy hajthatunk végre SQL utasításokat, hogy közben C# objektumokon dolgozunk. **Ehhez szükséges telepíteni a Microsoft.EntityFramework.Core** –t. Örökli a DbContext osztályt

```
public DbSet<TicketPaymentMessage> TicketPayment { get; set; }

public DbSet<Ticket> Tickets { get; set; }

public DbSet<RequestOrAnswerForTicketIDMessage> RequestOrAnswerForTicketIDMessages {
get; set; }

public DbSet<ParkingPlace> ParkingPlaces { get; set; }

public DbSet<TicketPriceMessage> TicketPrices { get; set; }
```

Mindegyik fenti tulajdonság egy táblát jelent az adatbázisban.

```
protected override void OnModelCreating(ModelBuilder modelBuilder)
{
modelBuilder.Entity<ParkingPlace>()
                .HasMany(e => e.Tickets)
                .WithOne(e => e.BookedParkingPlace)
                .HasForeignKey(e => e.ParkingPlaceId);
            modelBuilder.Entity<Ticket>()
                .HasMany(s => s.TicketPriceMessages)
                .WithOne(s => s.Ticket)
                .HasForeignKey(e => e.TicketID);
            modelBuilder.Entity<Ticket>()
                .HasMany(ticket => ticket.TicketPaymentMessages)
                .WithOne(msg => msg.Ticket)
                .HasForeignKey(ticket => ticket.TicketID);
            modelBuilder.Entity<Ticket>()
                .HasMany(ticket => ticket.RequestOrAnswerForTicketIDMessages)
                .WithOne(msg => msg.Ticket)
                .HasForeignKey(msg => msg.TicketID);
            modelBuilder.Entity<TicketPaymentMessage>()
                .Property(t => t.PaymentResult)
                .HasConversion(
                t => t.ToString(),
                t => (PaymentResult)byte.Parse(t)
                );
            modelBuilder.Entity<RequestOrAnswerForTicketIDMessage>()
                .Property(msg => msg.Type)
                .HasConversion(
                _type => _type.ToString(),
```

```
                _type => (ParkingMessageType)byte.Parse(_type)
                );
}
```

Az onModelCreating függvényt felülírva tudjuk manuálisan definiálni az adatok közötti relációkat és esetleges konverziókat. Ezek alapján, már képesek vagyunk létrehozni az adatbázist.

Fontos megjegyezni, hogy konverzió esetén komplex átalakítás nem valósítható meg ezzel a megoldással, mivel HasConversion függvény nem tartalmazhat delegáltakat.

```
protected override void OnConfiguring(DbContextOptionsBuilder optionsBuilder)
=>optionsBuilder.UseSqlite("DataSource=sqlite.db").EnableSensitiveDataLogging();
```

onConfiguring függvény felülírásával tudjuk meghatározni, hogy milyen adatbázis szolgáltatást szeretnénk alkalmazni, és a hozzátartozó ConnectionString-et is itt kell megadnunk.

IByteArrayConvertableInterface

Interfészt biztosít osztályok bájt halmazzá való alakításához.

```
int ByteArraySize { get;}
```

Az adott objektum méretét tartalmazza bájtokban.

```
int ToByteArraySpan(Span<byte> Array);
```

Visszatérési érték: **Lényegében pedig a ByteArraySize**

Funkció: Implementáló osztály átkonvertálása bájt sorozattá.

ParkingPlace Osztály

Ez az osztály a modellez le egy parkolóhelyet a rendszerben. Minden hozzá köthető információt itt definiálunk. Implementálja az IByteArrayConvertable interfészt.

```
[Key]
public int Id { get; set; }
```

Entity Framewok által generált érték, elsődleges kulcs a ParkingPlaces táblázatban

```
public static readonly int SizeInBytes = 36;
```

Osztály mérete bájtokban

```
public required int PosX { get;  init; }
```

Parkolóhely pozíciója az X tengelyen

```
public required int PosY { get;  init; }
```

Parkolóhely pozíciója az Y tengelyen

```
public required int Height { get; init; }
```

Parkolóhely mérete (magassága)

```
public required int Width { get; init; }
```

Parkolóhely mérete (szélessége)

```
public  ParkingPlaceType Type { get; set; }
```

Parkolóhely típusa, Lásd ParkingPlaceType enumerátor

```
public  virtual ICollection<Ticket>? Tickets { get; set; }
```

Entity framework-höz szükséges a reláció definiálásához.A parkolóhelyhez tartozó minden jegy betölthető ide

```
public Ticket? CurrentTicket { get;  set; }
```

Ha kiadott parkolóhely, akkor ez tartalmazza a hozzá tartozó jegyet.

```
public required int PID { get; init; }
```

Parkolóhely PID a hub-on belüli sorszámát jelöli a parkolóhelynek.

```
public required int HubID { get; init; }
```

Melyik hub-hoz tartozik

```
public required int Level { get; init; }
```

Melyik szinten helyezkedik el a parkolóhely, lényegében Z tengely.

```
public bool DisableWhenFree
```

Ha le szeretnénk tiltani a parkolóhely kiadását, ennek a tulajdonságnak a beállításával tehetjük meg. Amennyiben szabad a parkolóhely, akkor azonnal, egyébként a következő státuszváltozásnál, amikor a parkolóhely szabad lesz

```
public bool FreeToBook
```

Ha bérelhető egy parkolóhely, akkor igaz,ha a parkolóhely típusa általánosparkolóhely és a státusza Unkown vagy Free

```
public ParkingPlaceState State
```

A parkoló aktuális állapotát reprezentálja.

```
private ParkingPlaceState _stateDruingDisable;
```

Ha le van tiltva a parkolóhely kiadása, ez a változó tárolja a Hub-tól kapott állapotot.

```
public int ToByteArraySpan(Span<byte> Message)
```

Visszatérési érték: A konverzió utáni utolsó elemre mutató indexer. Lényegében SizeInBytes

Paraméter: `Span<byte> Message` Memória ablak, amely kezdő elemétől szeretnénk föltölteni bájtokkal

Funkciója: Parkolóhely bájt tömbbé való konverzióját látja el, a paraméterként kapott `Span<byte> Message` változóba.

```
public static ParkingPlace? FromByteArray(byte[] data, int index)
```

Visszatérési érték: Ha sikeres volt a művelet, új ParkingPlace példány, egyébként null.

Paraméterek:

- `byte[] data` Bájt tömb, amelyből a konverziót végre szeretnénk hajtani
- `int index` A bájt többöt ettől az indextől kezdve értékeljük ki.

Funkciója: Egy parkolóhely rekonstrukciója bájt sorozatból.

```
public override bool Equals(object? obj)
```

Equals metódus felülírása. Két parkolóhely akkor egyenlő, ha HubID és PID is egyenlő.

```
public override int GetHashCode()
```

Mivel az Equals metódust felülírtuk ezért muszáj definiálnunk a HashCode generáláshoz szükséges változót, ID.

```
public override string ToString()
```

Összes publikus tulajdonságát összefűzi egy stringbe, beleértve a jegyet is, hogyha kiadott parkolóhely.

ParkingPlaceState Enumerátor

Bájt tömbből származtatott enumerátor, a parkolóhely állapotát tükrözi, amelyek lehetnek:

- Unkown – Hub nincs csatlakoztatva
- Occupied Elfoglalták a parkolóhelyet
- Free Szabad a parkolóhely
- Disabled Ha ki van kapcsolva a parkolóhely kiadása.

```
public enum ParkingPlaceState : byte
{
```

```
    ParkingPlaceStateMinNotIncluded=0,
    Unkown=1,
    Occupied=2,
    Free=3,
    Disabled =4,
    ParkingPlaceStateMaxNotIncluded=5
}
```

ParkingPlaceType Enumeráor

Bájt tömbből származtatott enumerátor, a parkolóhely típusát tükrözi, amelyek lehetnek:

- RegularParkingPlace – Bérelhető
- BookedParkingPlace - Bérelt

```
public enum ParkingPlaceType : byte
{
    RegularParkingPlace = 1,
    BookedParkingPlace = 2,
}
```

Ticket Osztály

Ez az osztály modellezi le a jegyet, tartalmaz minden hozzá tartozó információt és rajta elvégezhető műveleteket itt kell definiálnunk.

```
public  const int SizeInBytes = 72
```

A Ticket méretét tárolja bájtokban.

```
private const int MINUTESTOLEAVE = 5
```

Fizetés után hány percünk van elhagyni a parkolóházat

```
private const double PRICEPERMINUTE = 4.5;
```

Percenkénti parkolási ár

```
public const int Max_Chars_In_License_Plate = 10;
```

Maximum hossza a rendszám változónak

```
[Key]
public int Id { get; private set; }
```

Entity Framewok által generált érték, elsődleges kulcs a Tickets táblázatban

```
public DateTime ParkingStarted { get; private set; }
```

Időpont, amikor a jegyet kiadták

```
public DateTime ParkingUntil { get; set; }
```

Eddig az időpontig van kifizetve a jegy.

```
public DateTime? WhenTicketPriceCalculated { get; set; }
```

Az az időpont, amikor utoljára ki lett számítva a jegy ára.

```
public DateTime? VehicleLeft { get; private set; }
```

Az az időpont, amikor a bérlés befejeződött, jármű kilépett.

```
public ParkingPlace? BookedParkingPlace { get; set; }
```

EntityFrameWork relációhoz definiálásához szükséges adattag, segítségével betölthető adatbázisból a jegyhez tartozó parkolóhely.

```
public int ParkingPlaceId { get; set; }
```

Jegyhez kiosztott parkoló azonosítója.

```
public  ICollection<TicketPriceMessage>? TicketPriceMessages { get; set; }
```

EntityFrameWork relációhoz definiálásához szükséges adattag, segítségével betölthető adatbázisból a jegyhez tartozó összes lekérdezett ár.

```
public  ICollection<TicketPaymentMessage>? TicketPaymentMessages { get; set; }
```

EntityFrameWork relációhoz definiálásához szükséges adattag, segítségével betölthető adatbázisból a jegyhez összes fizetési tranazkció eredménye

```
public ICollection<RequestOrAnswerForTicketIDMessage>?
RequestOrAnswerForTicketIDMessages { get; set; }
```

EntityFrameWork relációhoz definiálásához szükséges adattag, segítségével betölthető adatbázisból a jegyhez RequestOrAnswerForTicketIDMessage

```
public string? LicensePlate { get; private set; }
```

A jegyhez tartozó rendszámtábla, ha van.

```
public double Price{ get; private set; }
```

Utoljára kiszámolt jegy árat tartalmazza.

```
public int ByteArraySize{ get;}
```
A jegy méretét adja vissza bájtokban.

```
public static Ticket GetNewTicket(string? licensePlate, ParkingPlace newly_Booked)
```

Visszatérési érték: Újonnan generált jegy példánya

Paraméterek:

- `string? licensePlate` Generálni kívánt jegyhez tartozó rendszám
- `ParkingPlace newly_Booked` A parkolóhely, amelyhez hozzá szeretnénk rendelni az új jegyet.

Kivétel: ArgumentOutOfRangeException ha a rendszám hossza meghaladja az előre definiált hosszt.

Funkciója: Egy új jegy generálása egyedi azonosítóval és ennek hozzárendelése a parkolóhelyhez. **Fontos**, hogy ez **adatbázis lekérdezést tartalmaz, ezért ezek után nem szabad hozzáadni a jegyet az adatbázishoz, mivel már szerepel benne.** Erre azért van szükség, hogy a jegy azonosítóját az adatbázis generálja, ne pedig mi.

```
public bool CanVehicleLeave()
```

Visszatérési érték: Ha a jármű elhagyhatja a parkolóhelyet, akkor igaz

Funkció: Eldönteni, hogy az adott jármű elhagyhatja e a parkolóhelyet.

```
public double CalculatePrice()
```

Visszatérési érték: Jegy aktuális árának kiszámítása

Funkció: Jegy aktuális árát kiszámolni

Mellékhatás: WhenTicketPriceCalculated frissítése

```
public void TicketPayed()
```

Funkció: Ha egy jegyet kifizettek, akkor ezt a metódust hívjuk meg, amely frissíti a ParkingUntil mezőt, intuitívebb megoldásnak láttam, mint a Ticket osztályon kívülről, beállítani a ParkingUntil mezőt.

Mellékhatás: ParkingUntil frissítése

```
public int ToByteArraySpan(Span<byte> Message)
```

Visszatérési érték: A konverzió utáni utolsó elemre mutató indexer. Lényegében SizeInBytes

Paraméter: `Span<byte> Message` Memória ablak, amely kezdő elemétől szeretnénk feltölteni bájtokkal

Funkciója: Jegy bájt tömbbé való konverzióját hivatott ellátni, a paraméterként kapott `Span<byte> Message` változóba.

```
public static Ticket? FromByteArray(byte[] data, int indexer)
```

Visszatérési érték: Ha sikerült a konverzió, akkor egy új Ticket példánnyal tér vissza, egyébként null.

Paraméterek:

- `byte[] data` Bájt tömb, amelyből a konverziót végre szeretnénk hajtani
- `int index` A bájt többöt ettől az indextől kezdve értékeljük ki.

Funkciója: Ticket rekonstrukciójára használatos egy bájt folyamból

```
public override string ToString()
```

Összes publikus tulajdonságát összefűzi egy string-be.

ParkingPlaceManager Osztály

Ez modellezi le az egész parkolóház rendszer működését. Kapcsolódik az adatbázishoz, betölti a parkolóhelyeket és a nem teljesített jegyeket. Létrehozza és elindítja a szervert, majd feldolgozza a kliensek által kapott üzeneteket és ennek megfelelően változtat a parkolóház állapotán.

```
readonly private Dictionary<int, ParkingPlace> parkingPlaces;
```

Az aktív jegyeket tartalmazó könyvtár, kulcsa jegy azonosító

```
readonly private Dictionary<int,Ticket> tickets;
```

Parkolóhelyeket tartalmazó könyvtár, kulcsa a parkolóhely azonosító

```
readonly Dictionary<int, bool> ConnectedHubs;
```

Keys: Adatbázisban szereplő hub-ok azonosítója. Values: Ha csatlakozott az adott hub, akkor igaz.

```
readonly Server server;
```

A kommunikációért felelős Server osztály példánya.

```
private readonly ParkingMessageContext Db;
```

Az adatbázis kapcsolatot megteremtő osztály példánya.

```
public ParkingPlaceManager()
```

Kivételek:

ParkingPlacesHaveNotBeenAddedException, ha az adatbázisban nem szerepelnek parkolóhelyek, akkor a program nem megfelelően lett üzembe helyezve.

ArgumentException, ha két parkolóhely között átfedés van vagy két parkolóhely Hub és PID-je azonos.

Ha **nincs adatbázis fájl** a ParkingMonitoring.exe könyvtárában, **akkor létrehoz egyet**, majd ParkingPlaceHaveNotBeenAddedException-t dob.

```
public async Task StartServerAsync()
```

Funkció: Server.StartServerAsync() metódusának meghívása, innentől kezdve tudunk küldeni és fogadni üzeneteket.

```
private Dictionary<int, Ticket> LoadPendingTickets()=> Db.Tickets.Where((x) => !x.VehicleLeft.HasValue).ToDictionary<Ticket, int>(ticket => ticket.Id);
```

Visszatérési érték: Az adatbázisban szereplő aktív jegyek.

Funkció: Az adatbázisból aktív, lezáratlan státuszú jegyek betöltése.

```
private Dictionary<int, ParkingPlace> GetParkingPlaces(Dictionary<int, Ticket> pending_tickets)
```

Visszatérési érték: Adatbázisból betöltött parkolóhelyek.

Kivételek:

- ParkingPlacesHaveNotBeenAddedException Ha az adatbázisban nem szerepelnek parkolóhelyek
- ArgumentException: Ha két parkolóhely területén átfedés van vagy két parkolóhely Hub és PID-je azonos

Funkció: Adatbázisból az összes parkolóhely lekérdezése, ha van hozzátartozó aktív jegy, akkor összerendelése. Majd minden egyes parkolóhely állapotát ismeretlenre állítunk, hiszen még nem csatlakozhatott Hub, hogy frissítse ezek értékeit.

Mellékhatás: ConnectedHubs frissítése

```
private void MessageRecieved(object? sender, EventArgs e)
```

Paraméterek:

- `object? sender` függvényt meghívó objektum.
- `EventArgs e` Esemény argumentumok, jelen esetben nem használtak EventHandler használata miatt közeletű.

Kivétel: ArgumentException ha véletlenül rossz eseményre iratkoztunk fel, és a küldő típusa nem ParkingMessageQueue

Funkció: messageQueueba érkezett összes számunkra releváns üzenet kiolvasásra és az üzenetekhez tartozó „reakció" függvények meghívása. Például: Ha RequestACloseParkingPlaceMessage üzenetet kaptunk, akkor meghívjuk a RequestACloseParkingPlaceMessageReceived metódust.

```
private void
ParkingPlaceStateChangedMessageReceived(NotifyParkingPlaceStateChangedMessage msg)
```

Paraméter: A beérkezett üzenet

Funkció: Ha egy parkolóhely állapota megváltozott, akkor erről értesítjük az monitoring rendszert. Ha az üzenet tartalma nem értelmezhető, pl.: megváltozott parkolóhely HubID-je nem egyezik meg a küldő azonosítójával, akkor a klienst leválasztjuk. Egyébként továbbítjuk az üzenetet a monitoring alkalmazásnak.

Mellékhatás: parkingPlaces frissítése.

```
private void IdentificationMessageReceived(IdentificationMessage msg)
```

Paraméter: A beérkezett üzenet

Funkció: Hub-ok csatlakozásának regisztrálása. connectedHubs könvytárba. Ha egy olyan azonosítóval próbálnak meg csatlakozni, ami nem szerepel a connectedHubs-ban, vagy már csatlakozott, akkor az a kliens kapcsolatbontással jár.

Mellékhatás: connectedHubs frissítése

```
private void ClientDisconnected(object? Sender, ServerToClientConnection
Disconnected)
```

Paraméterek:

- `object? sender` függvényt meghívó objektum. (Server)
- `ServerToClientConnection Disconnected` A lecsatlakzotatott klienshez tartozó ServerToClientConnection példánya.

Funckió: Ha egy kliens lecsatlakozott, akkor meg kell vizsgálni, hogy Hub volt-e. Ha igen, akkor a connectedHubs-ból eltávolítjuk a csatlakozott státuszt és az összes hozzá tartozó parkolóhelynek az állapotát ismeretlenre állítjuk, majd erről értesítjük a monitoring alkalmazást.

Mellékhatás:

- connectedHubs frissítése
- parkingPlaces hubhoz tartozó státuszok frissítése

```
private void RequestForTicketDetailsMessageReceived(RequestForTicketDetails msg)
```

Paraméter: A beérkezett üzenet

Funkció: A üzenetben meghatározott feltételek szerinti adatbázis lekérdezés végrehajtása, majd ha talált ennek megfelelő jegyeket, akkor ezeket egyesével elküldi TicketDetailsMessage formájában, ha az összessel végzett, akkor elküld még egy AllTicketsHaveBeenSent üzenetet. Ha nem talált jegyet, akkor egy ErrorMessage-et küldd vissza, amely hibájának típusa: NoTicketFoundForThatRequest

```
private void
RequestACloseParkingPlaceMessageRecieved(RequestACloseParkingPlaceMessage msg)
```

Paraméter: A beérkezett üzenet

Funkció: Az üzenetben kapott pozícióhoz tartozó legközelebbi parkolóhely kiválasztása, jegy generálása, majd a parkolóhelyhez hozzárendeli a jegyet, ennek megfelelően frissíti az adatbázist és tájékoztatja a monitoring alkalmazást a változásról. Ha nincs szabadhely, akkor hibaüzenet kerül kiküldésre.

Mellékhatás: parkingPlaces és tickets frissítése

```
private ParkingPlace? FindTheClosestFreeParkingPlace(int PosX, int Posy)
```

Visszatérési érték: A legközelebbi parkolóhely az adott pozícióhoz, ha nincs parkolóhely, null.

Paraméterek:

- `int PosX` A keresendő közeli helyhez tartozó X koordináta.
- `int Posy` A keresendő közeli helyhez tartozó Y koordináta.

Funkció: Minimumkeresést végez az adott pozíció és a parkoló helyek távolságai között.

```
private ParkingPlace? GetFirstFreeParkingPlace()
```

Visszatérési érték: Az első szabad parkolóhely, ha van, egyébként null.

Funkció: Első szabad parkolóhely megkeresése.

```
private static double ParkingPlaceMinDistanceFromPoint(ParkingPlace a, int PosX, int PosY)
```

Visszatérési érték: Megadott parkolóhely és pozíció közötti távolság.

Paraméterek:

- `ParkingPlace` a parkolóhely, amelynek a távolságát szeretnénk kalkulálni.
- `int PosX` Pozíció X tengelyen, amelyhez tartozó távolságra szükség van.
- `int PosY` Pozíció Y tengelyen, amelyhez tartozó távolságra szükség van.

Funkció: Meg keresi azt a pontot, amely a legközelebb helyezkedik el a keresendő pozicóhoz (mivel a parkolóhely egy téglalap), majd kiszámítja a kettő közötti távolságot.

```
private void GetTicketPriceMessageRecieved(RequestOrAnswerForTicketIDMessage msg)
```

Paraméter: A beérkezett üzenet

Funkció: Az üzenetben található jegyhez tartozó azonosító alapján kiszámítja a jegy árát és visszaküldi a kérvényezőnek (jegyautomata). Jegyhez tartozó rekordot frissíti az adatbázisban és a monitoring rendszernek elküldi a megváltozott jegyet.

Mellékhatás: tickets frissítése

```
private void TicketPaymentMessageReceived(TicketPaymentMessage msg)
```

Paraméter: A beérkezett üzenet

Funkció: Jeggyel kapcsolatos fizetés eredményének feldolgozása, ami ha sikeres volt, akkor frissítjük az adatbázist, és a megváltozott jegyet átküldjük a monitoring alkalmazásnak.

Mellékhatás: tickets frissítése

```
private void FullMapRequestMessageRecieved(ParameterlessMesage msg)
```

Paraméter: A beérkezett üzenet

Funkció: A teljes parkolóház térképének lekérdezése. Elsőnek elküld egy MapSizeMessage üzenetet, majd pedig egyesével minden parkolóhelyet elküld ParkingPlaceDetailsMessage formájában.

```
private void CanVehicleLeaveMessageReceived(RequestOrAnswerForTicketIDMessage msg)
```

Paraméter: A beérkezett üzenet

Funkció: Kilépési pont által küldött üzenet, amelyben eldöntjük, hogy az adott azonosítóval rendelkező jegy kilépésre fel jogosít-e. Ha nincs ilyen jegy az tickets könyvtárban (nem aktív), akkor hibaüzenetet küldünk, aminek típusa NoTicketForTicketID. Amennyiben nem jogosult kilépésre, akkor VehicleCannotLeave , egyébként VehicleCanLeave válasz üzenetet küld. Ezeket mentjük az adatbázisban, ha a jármű kilépett a parkolóházból, akkor a tickets könyvtárból eltávolítjuk, és a parkolóhelyet is visszaállítjuk általános típusúra, majd a monitorozó alkalmazásnak elküldjük a megváltozott parkolóhelyet.

Mellékhatások:

- tickets könyvtárból jegy eltávolítása
- parkingPlaces frissítése

```
private bool TwoParkingParkingPlacesOverlap(ParkingPlace a, ParkingPlace b)
```

Visszatérési érték: Ha két megadott parkolóhely fedi egymást, akkor igaz

Paraméter: Két parkolóhely, amelyekről el kell dönteni, hogy fedik e egymást

```
private bool AnyParkingPlacesOverLap(Dictionary<int, ParkingPlace> places)
```

Visszatérési érték: Bármely parkolóhelyek között van átfedés.

Paraméter: Collection, amin a keresést végezzük.

```
private bool AnyHubidAndPidMatch(Dictionary<int, ParkingPlace> places)
```

Visszatérési érték: Igaz, bármely parkolóhely PID- és HubID-je is azonos.

Paraméter: Collection, amin a keresést végezzük.

ParkingPlacesHaveNotBeenAddedException Osztály

```
public class ParkingPlacesHaveNotBeenAddedException(string message) :
Exception(message)
```

Exception osztályból származik. Csak string üzenettel lehet létre hozni, egyébként semmilyen extra információt nem tartalmaz. Akkor keletkezik, ha az adatbázisban nem szerepelnek parkolóhelyek.

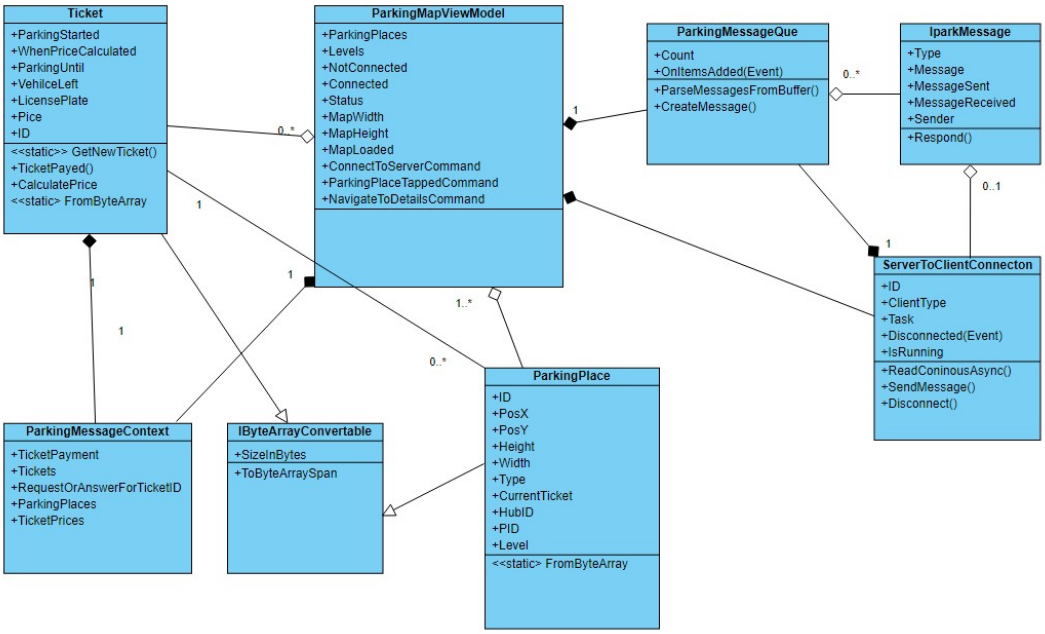
Smart Parking Monitoring

Ezzel az alkalmazással lehet, a parkolóház vizuális, táv-monitorozását elvégezni. Szerverhez való csatlakozás után, lekérdezi az összes parkolóhely és jegy aktuális állapotát. Ha megérkezett az összes információ, akkor létrehozza a parkolóház grafikus leképezését. Innentől kezdve látjuk élőben a parkolóház működését. Lehetőséget biztosít egy adott parkoló összes aktuális tulajdonságának megtekintésére, és jegyek lekérdezésére az adatbázisból.

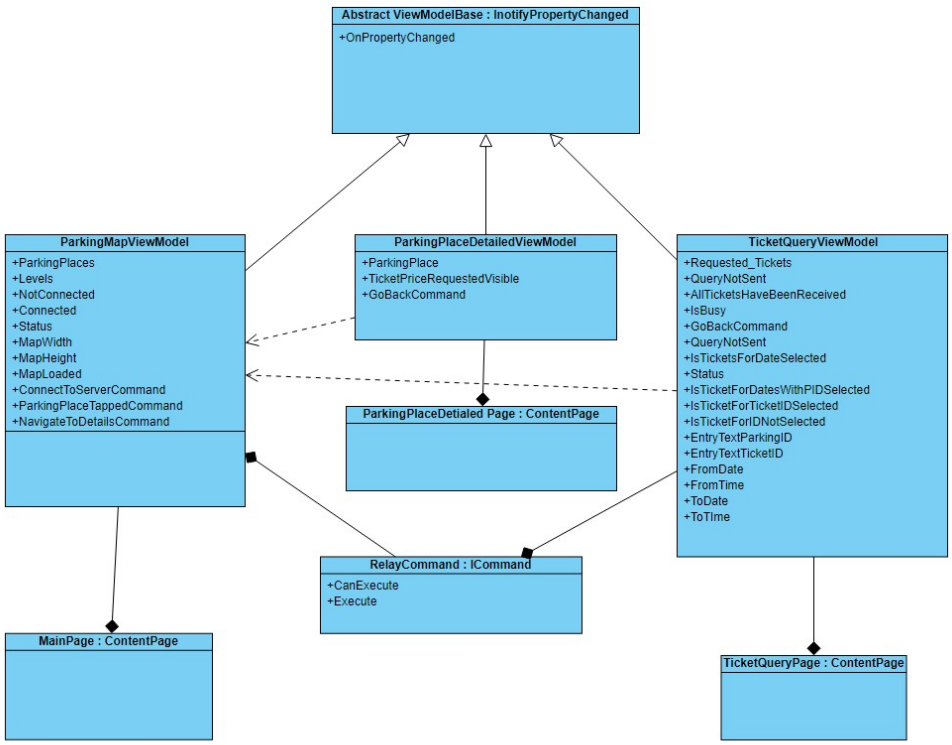
TargetFramework: .NET 8.0 C#12

Applikáció típusa: .NET MAUI

Modell-nézet-nézetmodell architekturális minta alapján készült el. A kinézet leírását a hozzá tartozó .xaml fájl tartalmazza.



5. Ábra A kommunikációért felelős UML Class diagramja majdnem teljesen megegyezik a szerverével.



6. Ábra Az nézetek és nézetmodellek közötti UML-Class diagram

A következő osztályok leírásai a szerver fejezetben olvashatóak:

- ErrorMessage
- IdentificationMessage
- IparkMessage
- MapSizeMessage
- NotifyParkingPlaceStateChangedMessage
- ParameterlessMessage
- ParkingMessageQueue
- ParkingPlaceBookedMessage
- ParkingPlaceDetailsMessage
- RequestACloseParkingPlaceMessage
- RequestForTicketDetails
- RequestOranswerForTicketIDMessage
- TicketDetialsMessage
- TicketPriceMessage
- ServerToClientConnection
- IByteArrayConvertable
- ParkingPlace
- Ticket

Kiegészítés, eltérés található a ServerToClientConnection fájlban:

`private async Task<int> GetFirstMesage(byte[] buffer)` függvény nem szerepel, mivel a kliens nem azonosítja a szervert..

ParkingMapViewModel Osztály:

Ez az osztály felelős a szerverrel való kapcsolattartásért és az élő monitorozás kivitelezéséért.

```
private int ParkingPlacesCount = 0;
```

Az parkolóhelyek száma, amelyet egy MapSizeMessage érkezése során állítunk be

```
readonly ServerToClientConnection toServer;
```

Kapcsolat a szerverrel.

```
readonly ParkingMessageQueue pmq;
```

ParkingMessageQueue, üzenetek érkezéséről általa kapunk értesítést

```
public ObservableCollection<ParkingPlace> ParkingPlaces { get; private set; }
```

Ez a lista tartalmazza a parkolóhelyek aktuális állapotát, csatlakozás után.

```
public ObservableCollection<int> Levels { get; private set; }
```

Ez tartalmazza, hogy a parkolóház hány emeletet tartalmaz (lényegében Z tengelyen felvett érékek)

```
public bool MapLoaded { get { return _mapLoaded; } set { _mapLoaded = value;
OnPropertyChanged(); } }
```

Megérkezett-e az összes parkolóhely.

```
public bool NotConnected
```

Ha nincs csatlakozva a szerverhez, akkor igaz.

```
public bool Connected
```

Ha csatlakozva van a szerverhez, akkor igaz.

```
public string Status
```

Státusz üzenet megjelenítésére használt tulajdonság.

```
public int MapHeigth
```

A parkolóház hosszúságát tartalmazza (Y tengely)

```
public int Map Width
```

A parkolóház szélességét tartalmazza (X tengely)

```
public ICommand ConnectToServerCommand { get; }
```

A nézet ezzel a paranccsal tudja meghívni a ConnectClicked függvényt.

```
public ICommand ParkingPlaceTappedCommand { get; }
```

A nézet ezzel a paranccsal tudja meghívni a ParkinPlaceTapped függvényt.

```
public ICommand NavigateToTicketDetailsCommand { get; }
```

A nézet ezzel a paranccsal tudja meghívni a NavigateToTicketDetails függvényt.

```
public ParkingMapViewModel(ServerToClientConnection conn, ParkingMessageQueue _pmq)
```

Konstruktorban lévő paraméterek a Maui keretrendszerbe beépített Dependency Injection segítségével kerülnek átadásra.

```
private void ToServer_ClientDiscConnected(object? sender, Socket e)
```

Paraméterek:

- `object? sender,` Esemény kiváltó objektum (ServerToClientConnection)
- `Socket e` Megszakadt kapcsolathoz tartozó socket.

Funkció: Ha megszakadt a kapcsolat a szerverrel, akkor az élő monitorozás nem működik, meg kell próbálni újra csatlakozni. Ilyenkor az összes adat újraküldésre kerül.

Mellékhatások: MapLoaded, Status, NotConnected paraméterek beállítása

```
private async Task<bool> ConnectToServerAndStartListening()
```

Visszatérési érték: Sikerült-e a csatlakozni a szerverhez.

Funkció: Szerverhez való csatlakozás és az üzenetek fogadásának elindítása egy új szálon. Majd az IdentificationMessage elküldése a szerver számára.

```
private async Task ConnectClicked()
```

Funkció: Ha a felhasználó csatlakozni szeretne a szerverhez, ez a metódus kerül meghívásra, megpróbál csatlakozni a szerverhez (ConnectToServerAndStartListenging függvény segítségével), ha sikerült, akkor küld egy FullMapRequestet, egyébként visszatér.

```
private async Task ParkingPlaceTapped(object? args)
```

Paraméter: Az a parkolóhely, amelyre kattintottak, típus ellenőrzés szükséges.

Funkció : Ha egy parkolóhelyre kattintottak, akkor a beépített Shell segítségével elnavigálunk a ParkingPlaceDetailed oldalra, amelynek paraméterként átadjuk a megjeleníteni kívánt parkolóhelyet.

```
private async Task NavigateToTicketDetails()
```

Funkció: Ha a TicketQueryPage –et szeretnénk megnyitni, akkor az ez a függvény kerül meghívásra, ami beépített Shell navigációt használva megjeleníti azt.

```
private static void UpdateParkingPlaceDetailsIfOpen(ParkingPlace updated)
```

Paraméter: `ParkingPlace updated` frissíteni kívánt parkolóhely

Funkció: Ellenőrizzük, hogy ParkingPlaceDetails oldal van-e megnyitva, ha igen, akkor frissítjük a nézetmodelljében található ParkingPlace objektumot.

```
private void OnItemsAddedToTheQue(object? Sender, EventArgs e)
```

Paraméterek:

- `object? sender` függvényt meghívó objektum.
- `EventArgs e` Esemény argumentumok, jelen esetben nem használtak EventHandler használata miatt közeletű.

Kivétel: ArgumentException ha véletlenül rossz eseményre iratkoztunk fel, küldő típusa nem ParkingMessageQueue

Funkció: messageQueueba érkezett összes számunkra releváns üzenet kiolvasása és az üzenetekhez tartozó „reakció” függvények meghívása.

```
private void ParkingPlaceStateChangedMessage(NotifyParkingPlaceStateChangedMessage
msg)
```

Paraméter: Fogadott üzenet.

Funkció: Megváltozott állapotú parkolóhely kikeresése, majd annak frissítése. Részletes nézetben is.

Mellékhatások: ParkingPlaces kollekció frissítése, TicketQueryViewModel-hez tartozó parkolóhely frissítése.

```
private void MapSizeMessageReceived(MapSizeMessage msg)
```

Paraméter: Fogadott üzenet.

Funkció: Eltárolni, hogy hány darab parkolóhelyet várunk a szervertől.

Mellékhatások: ParkingPlaces, ParkingPlacesCount, Maploaded

```
private static void ErrorMessageReceived(ErrorMessage msg)
```

Paraméter: Fogadott üzenet.

Funkció: Hiba üzenet esetén annak feldolgozása. Jegy lekérdezéshez használt nézet frissítése, ha nem talált jegyet a megadott szűrési feltételeknek.

```
private static void AllTicketsHaveBeenSent()
```

Paraméter: Fogadott üzenet.

Funkció: Ha a szerver átküldte a lekérdezésnek megfelelő összes üzenetet, akkor ezzel nyugtázza, ha nyitva van még a részletes jegy lekérdezés oldal, akkor ezt az üzenetet jelezzük.

```
private  void TicketDetailsMessageReceived(TicketDetailsMessage msg)
```

Paraméter: Fogadott üzenet.

Funkció: Ha egy parkolóhelyet kibéreltek, akkor átállítja a típusát és hozzárendeli a jegyet. Frissíti a részletes parkolóhely néző oldalt.

Mellékhatások: ParkingPlaces kollekció, ParkingPlaceDetaliedViewModel.Parkingplace

```
private void ParkingPlaceBookedMessageReceived(ParkingPlaceBookedMessage msg)
```

Paraméter: Fogadott üzenet.

Funkció: Eldönti, hogy változás történt e valamelyik aktuális jegy állapotában, ha igen, akkor frissíti, ha nem, akkor részletes jegy lekérdezéshez érkezett az üzenet, hozzá kell adni a TicketQueryViewModel.Requested_Tickets kollekciójához.

```
private void ParkingPlaceDetailsMessageReceived(ParkingPlaceDetailsMessage msg)
```

Paraméter: Fogadott üzenet.

Funkció: Ezt az üzenetet FullMapRequest után kaphatjuk, vagy ha egy parkolóhely bérelt státuszból szabaddá változott, ennek megfelelően kerül frissítésre a ParkingPlaces kollekció, ha az utóbbi eset következett be, akkor frissíti a részletesen parkolóhely nézetmodelljét is.

Mellékhatások: ParkingPlaces kollekció, ParkingPlaceDetaliedViewModel.ParkingPlace

ParkingPlaceDetailedViewModel Osztály

Ez az osztály kizárólagosan azért létezik, hogy biztosítsa a megjelenítendő információkat az adott parkolóhelyről a nézetnek. Érdemi műveleteket nem végez. ViewModelBase-t örökli.

```
public bool TicketPriceRequestedVisible
```

Ez a tulajdonság azt tükrözi, hogy a parkolóhelyhez tartozó jegynek a WhenTicketPriceCalculated mezője értelmezhető értéket tartalmaz-e ? Ha még nem lett beállítva, akkor ne mutassa

```
public ICommand GoBackCommand { get;}
```

A nézet ezzel a tulajdonsággal tudja meghívni a GoBack függvényt.

```
private static async Task GoBack()
```

Funkciója: Beépített Shell navigáció segítségével visszatérünk a fő oldalra

```
public ParkingPlace? ParkingPlace
```

Ez az tulajdonság a beépített Shell navigáció során kerül beállításra és a megjeleníteni kívánt parkolóhelyet tartalmazza. Beállítása során kap értéket a TicketPriceRequestedVisible változó is.

TicketQueryViewModel Osztály

Ez az osztály biztosítja az információkat a TicketQueryPage-nek. Üzenetet küldhet a szervernek, amelyben adatbázis lekérdezést kérvényez, szűrési feltételek mellet a Jegyek táblára. Sikeres a lekérdezés, esetén megjeleníti a kapott értékeket.

```
readonly ServerToClientConnection Connection;
```

Referencia a szervehez, hogy RequestForTicketDetails üzenetet küldhessen.

```
public ObservableCollection<Ticket> Requested_Tickets
```

Ha szűrési feltételeknek megfelelő ticketeket tartalmazza

```
public bool IsTicketForDateSelected
```

Érétkét nézetből állítjuk be, ha a csak időintervallumra szeretnénk szűrni

```
public bool IsTicketForDatesWithPIDSelected
```

Értékét neztből állítjuk be, ha időintervallumra és parkolóhely azonosítóra is szeretnénk szürni.

```
public bool IsTicketForTicketIDSelected
```

Értékét nézetből állítjuk be, ha csak jegy azonosítóra szeretnénk szűrni.

```
public bool IsTicketForIDNotSelected
```

IsTicketForIDNotSelected negáltja, nézetnek szükséges, egyes vezérlők elrejtéséhez

```
public string Status
```

Az adott lekérdezés státuszát ezzel tudjuk megjeleníteni.

```
public string EntryTextParkingID
```

Ha parkolóhely azonosítóra is szűrünk, akkor ez tartalmazza a megadott értéket.

```
public string EntryTextTicketID
```

Ha jegy azonosítóra szűrünk, akkor ez tartalmazza a megadott értéket

```
public DateTime FromDate
```

Ha dátumra szűrűnk, ez tartalmazza az intervallum alsó határát

```
public TimeSpan FromTime
```

Ha dátumra szűrűnk ez tartalmazza az intervallum alsó határához tartozó óra- perc étéketeket

```
public TimeSpan ToTime
```

Ha dátumra szűrűnk ez tartalmazza az intervallum alsó határához tartozó óra- perc étéketeket

```
public DateTime ToDate
```

Ha dátumra szűrűnk, ez tartalmazza az intervallum felső határát

```
public bool QueryNotSent
```

Ha a lekérdezés nem lett elküldve, vagy már megérkezett az összes válaszüzenet, akkor igaz

```
public bool AllTicketsHasBeenReceived
```

Ha megérkezett az összes válasz üzenet, akkor igaz

```
public ICommand GoBackCommand
```

Ennek a parancsnak a segítségével tudja a nézet meghívni a GoBack() metódust.

```
public ICommand ExecuteQueryCommand { get; }
```

Ennek a parancsnak a segítségével tudja a nézet meghívni az ExecuteQuery() metódust.

```
public TicketQueryViewModel(ServerToClientConnection Conn)
```

Conn az dependency injection által kapott konstrukot paraméter.

```
private static async Task GoBack()
```

Funckiója: Shell navigáció segítségével visszatérünk a kezdő oldalra.

```
private async Task ExecuteQuery()
```

Funkciója: Felhasználó által beállított adatok szerinti adatabázis lekérdezési kérelem elküldése a szervernek.

BooleanToOpacityConverter Osztály

Ez osztály örökli az IValueConverter interfészt, ezeknek metódusait implementálja. Ha egy komplex layout struktúrát láthatatlanná állítok isVisible tulajdonságával, akkor a keretrendszer úgy kezeli, mintha nem is lenne ott, ennek következményében elcsúszhatnak a relatív viszonyban álló további objektumok is ezért az áttetszőségét állítjuk át.

```
public object Convert(object? value, Type targetType, object? parameter, CultureInfo culture)
```

Visszatérési érték: ha bool tipusú volt a változó és az értéke hamis értékekre, akkor 0, egyébként 100

Paraméterek: Csak a value paraméter hordoz számunkra használatos információt.

```
public object ConvertBack(object? value, Type targetType, object? parameter, CultureInfo culture)
```

Kivétel: NotImplementedException, mivel egyirányű adatkötést haszálunk csak, ezért nem szükséges Implementálni.

LevelToOpacityConverter Osztály

Implementálja az IMultiValueConverter interfészt. Kiválasztott szint és az adott parkolóhely szinte között végez láthatósági konverziót.

```
public object Convert(object[] values, Type targetType, object parameter, CultureInfo culture)
```

Visszatérési érték: ha value[0] int és value[1] int, értékük megegyezik, akkor 100, egyébként 0.

Paraméterek: Csak a values paraméter hordoz számunkra használatos információt,

```
public object Convert(object[] values, Type targetType, object parameter, CultureInfo culture)
```

Kivétel: NotImplementedException, mivel egyirányű adatkötést haszálunk csak, ezért nem szükséges.

ParkingPlaceStateConverter Osztály

Implementálja az IValueConverter interfészt. Adott parkolóhely állapotát és tipusát figyelembe véve előállít egy színt, ezáltal megkülönböztethetővé válnak ezek.

```
public object Convert(object? value, Type targetType, object? parameter, CultureInfo
culture)
```

Visszatérési érték: Az adott parkolóhely tulajdonságaihoz hozzárendelt szin.

Paraméterek: Csak a values paraméter hordoz számunkra használatos információt, ennek értéke minden esetben egy ParkingPlace.

```
public object Convert(object[] values, Type targetType, object parameter,
CultureInfo culture)
```

Kivétel: NotImplementedException, mivel egyirányű adatkötést haszálunk csak, ezért nem szükséges.

MauiProgram.cs

Az applikáció eppen a fájlban kerül felépítésre, itt tudjuk a MauiAuppBuilder segítségével regisztrálni azokat az osztályokat, amelyeket depedency injection segítségével szeretnénk átadni konstruktor paraméteknek.

```
public static MauiApp CreateMauiApp()
{
    var builder = MauiApp.CreateBuilder();
    builder
        .UseMauiApp<App>()
        .ConfigureFonts(fonts =>
        {
            fonts.AddFont("OpenSans-Regular.ttf", "OpenSansRegular");
            fonts.AddFont("OpenSans-Semibold.ttf", "OpenSansSemibold");
        })
        .UseMauiCommunityToolkit();
    builder.Services.AddSingleton<ParkingMessageQueue>();
    builder.Services.AddSingleton<ServerToClientConnection>();
    builder.Services.AddSingleton<ParkingMapViewModel>();
    builder.Services.AddSingleton<MainPage>();
```

```
            builder.Services.AddTransient ParkingPlaceDetailed>();
            builder.Services.AddTransient<TicketQueryViewModel>();
            builder.Services.AddTransient< TicketQueryPage>();
            return builder.Build();
        }
```

Ebből látható, hogy Singleton ként regisztráltuk a fő oldalt, ParkingMessageQueue-t , ServerToClientConnection-t és a MainPage-et, ennek következtében, bármelyik konstruktor paraméterben, ha szerepelnek, akkor ugyan azt a példányt fogják megkapni autómatikusan, míg a TicketQueryViewModel ParkingPlaceDetailted és TicketQueryPage mindig új példányként kerül átadásra.

AppShell.xaml.cs

Ebben a fájlban kell regisztrálnunk azokat az oldalakat, amelyeket a beépített Shell navigációval szeretnénk használni

```
public AppShell()
{
    InitializeComponent();
    Routing.RegisterRoute(nameof(ParkingPlaceDetailed),
typeof(ParkingPlaceDetailed));
    Routing.RegisterRoute(nameof(TicketQueryPage), typeof(TicketQueryPage));
}
```

Entry Point Parking Simulator

Ez a program szimulálja a beléptető eszköz szerepét, hogy tesztelhető legyen a rendszer. Csatlakozni tud a szerverhez és parkoló helyet kérni egy megadott pozicióhoz. Ha tele van a parkolóház, akkor kijelzi a hibát.

A Smart Parking Monitoring fájljait használja a szerverhez kapcsolodó kommunikációhoz, egyben ugyan az a felépítése az x ábrán látottakhoz, kivétel a ParkingMapView helyén a Form1 foglal helyet ennek attribútumai, metódusai mások, viszont a kapcsolat rendszer ugyan az.

TargetFramework: .NET 8.0 C#12

Applikáció típusa: Winforms Application

```
private async void button1_Click(object sender, EventArgs e)
```

Paraméterek: ButtonClickEventHandlerhez köthetőek, jelen esetben nincsenek használva.

Funkció: Szerverhez csatlakozás aszinkron módon, és üzenetek fogadásának elindítása új szálon.

```
private void Client_ClientDiscConnected(object? sender, Socket e)
```

Paraméterek:

- `object? Sender` Függvényt meghívó objektum (ServerToClientConnection)
- `Socket disconnectedSocket` Lekapcsolódott Socket

Funkció: Ha a szerver valamilyen oknál fogva lecsatlakoztatott minket, akkor új ServerToClientConnection-t kell létrehozni, majd engedélyezni az újrakapcsolódás lehetőségét.

```
private void MessageRecieved(object? sender, EventArgs e)
```

Paraméterek:

- `object? sender` függvényt meghívó objektum.(ParkingMessageQueue)
- `EventArgs e` Esemény argumentumok, jelen esetben nem használtak EventHandler használata miatt közeletű.

Kivétel: ArgumentException ha véletlenül rossz eseményre iratkoztunk fel, és a küldő típusa nem ParkingMessageQueue

Funkció: messageQueueba érkezett összes számunkra releváns üzenet kiolvasásra és az üzenetek megjelenítése label2-n. Két üzenet érkezését kell csak lekezelni, ErrorMessage (akkor kaphatjuk, ha nincs szabad hely a parkolóházban), és TicketDetailsMessage, ha sikerült parkolóhelyet kapnunk.

```
private void button2_Click(object sender, EventArgs e)
```

Paraméterek: ButtonClickEventHandlerhez köthetőek, jelen esetben nincsenek használva.

Funkció: Szerverheznek RequestACloseParkingPlaceMessage küldése, a textbox1 és textbox2 mezőben megadott pozició X és Y argumentummal.

ParkingPlace Hub Simulator

Ez a program szolgálja a hub szerepét, hogy tesztelhető legyen a rendszer. Csatlakozni tud a szerverhez egy megadott Hub azonosítóval és megadott számú parkolóhellyel, amelyek státuszait változtatni tudjuk.

A Smart Parking Monitoring fájljait használja a szerverhez kapcsolodó kommunikációhoz, egyben ugyan az a felépítése az x ábrán látottakhoz, kivétel a ParkingMapView helyén a Form1 foglal helyet ennek attribútumai, metódusai mások, viszont a kapcsolat rendszer ugyan az.

TargetFramework: .NET 8.0 C#12

Applikáció típusa: Winforms Application

```
private async void button1_Click(object sender, EventArgs e)
```

Paraméterek: ButtonClickEventHandlerhez köthetőek, jelen esetben nincsenek használva.

Funkció: Szerverhez csatlakozás aszinkron módon, és üzenetek fogadásának elindítása új szálon

```
private void Client_ClientDiscConnected(object? sender, Socket e)
```

Paraméterek:

- `object? Sender` Függvényt meghívó objektum (ServerToClientConnection)
- `Socket disconnectedSocket` Lekapcsolódott Socket

Funkció: Ha a szerver valamilyen oknál fogva lecsatlakoztatott minket, akkor új ServerToClientConnection-t kell létrehozni, majd engedélyezni az újrakapcsolódás lehetőségét.

```
private void button2_Click(object sender, EventArgs e)
```

Paraméterek: ButtonClickEventHandlerhez köthetőek, jelen esetben nincsenek használva.

Funkció: Feltölti a listbox1 elemeit parkingplace állapotokkal 1-től a textbox1 ben megadott értékig.

```
private void MessageRecieved(object? sender, EventArgs e)
```

Paraméterek:

- `object? sender` függvényt meghívó objektum.(ParkingMessageQueue)
- `EventArgs e` Esemény argumentumok, jelen esetben nem használtak, EventHandler használata miatt közeletű.

Kivétel: ArgumentException ha véletlenül rossz eseményre iratkoztunk fel, és a küldő típusa nem ParkingMessageQueue

Funkció: messageQueueba érkezett összes számunkra releváns üzenet kiolvasásra és az üzenetek megjelenítése label2-n. Egy üzenet releváns csak számunkra, mégpedig az ErrorMessage. Ha valami hibás adatot közöltünk.

```
private void listBox1_MouseDoubleClick(object sender, MouseEventArgs e)
```

Paraméterek:

- `object? sender` függvényt meghívó objektum.(listBox1)
- `MouseEventArgs e` Az egérhez köthető esemény argumentumok, jelenleg nem használtak

Funkció: listbox1 elemére duplán kattintva fut le ez az esemény. Ha a szerverhez kapcsolódva vagyunk, akkor a kiválasztott elem értékét szabadról –elfolgaltra állítjuk vagy elfolgaltról-szabadra, majd ezt a kapott értéket elküldjük a szervernek.

Exit Point Simulator

Ez a program szolgálja a kiléptető kapu szerepét, hogy tesztelhető legyen a rendszer. Csatlakozni tud a szerverhez és FullMapRequest üzenettel lekérdezi az összes parkolóhelyet, ezekhez tartozó jegyeket elmenti és a továbbiakban bármiféle jeggyel kapcsolatos változást figyel. Erre azért van szükség, hogy ne kézzel keljen jegy azonosítókat beirogatni, hanem létező jegyekkel probáljunk meg kilépni. A kilépés sikerességéről tájékoztat.

A Smart Parking Monitoring fájljait használja a szerverhez kapcsolodó kommunikációhoz, egyben ugyan az a felépítése az x ábrán látottakhoz, kivétel a ParkingMapView helyén a Form1 foglal helyet ennek attribútumai, metódusai mások, viszont a kapcsolat rendszer ugyan az.

TargetFramework: .NET 8.0 C#12

Applikáció típusa: Winforms Application

```
private void Client_ClientDiscConnected(object? sender, Socket e)
```

Paraméterek:

- `object? Sender` Függvényt meghívó objektum (ServerToClientConnection)
- `Socket disconnectedSocket` Lekapcsolódott Socket

Funkció: Ha a szerver valamilyen oknál fogva lecsatlakoztatott minket, akkor új ServerToClientConnection-t kell létrehozni, majd engedélyezni az újrakapcsolódás lehetőségét.

```
private async void button1_Click(object sender, EventArgs e)
```

Paraméterek: ButtonClickEventHandlerhez köthetőek, jelen esetben nincsenek használva.

Funkció: Szerverhez csatlakozás aszinkron módon, és üzenetek fogadásának elindítása új szálon

```
private void MessageRecieved(object? sender, EventArgs e)
```

Paraméterek:

- `object? sender` függvényt meghívó objektum.(ParkingMessageQueue)
- `EventArgs e` Esemény argumentumok, jelen esetben nem használtak, EventHandler használata miatt közeletű.

Kivétel: ArgumentException ha véletlenül rossz eseményre iratkoztunk fel, és a küldő típusa nem ParkingMessageQueue

Funkció: messageQueueba érkezett összes számunkra releváns üzenet kiolvasásra és az üzenetekhez tartozó „reakció" függvények meghívása. Jelen esetben a MapSizeMessage, ParkingPlaceDetailsMessage, ErrorMessage, TicketDetailsMessage, RequestOrAnwerForTicketIDMessage.

```
private void MapSizeMsgReceived(MapSizeMessage msg)
```

Paraméter: Fogadott üzenet.

Funkció: Eltárolni, hogy hány darab parkolóhelyet várunk a szervertől.

```
private void TicketDetailsMessageReceived(TicketDetailsMessage msg)
```

Paraméter: Fogadott üzenet.

Funkció: Ha bármiféle változást történt egy jeggyel kapcsopcsolatban, akkor frissíti őket és a listboxot.

```
private void ParkingPlaceBookedMsg(ParkingPlaceBookedMessage msg)
```

Paraméter: Fogadott üzenet.

Funkció: Ha egy parkolóhelyet kibéreltek, akkor a hozzá tartozó jegyet elmenti és frissiti a listboxot.

```
private void ParkingPalaceDetailsMessageReceived(ParkingPlaceDetailsMessage msg)
```

Paraméter: Fogadott üzenet.

Funkció: Aktív ticketek feltöltésére és meglévő ticketeket frissítését itt tudjuk elvégezni.

```
private void button2_Click(object sender, EventArgs e)
```

Paraméterek: ButtonClickEventHandlerhez köthetőek, jelen esetben nincsenek használva.

Funkció: A listboxon kiválasztott jegyet lehet megprobálni kiléptetni a rendszerből.

Ticket Machine Simulator

Ez a program szolgálja a jegyautómata szerepét, hogy tesztelhető legyen a rendszer. Csatlakozni tud a szerverhez és FullMapRequest üzenettel lekérdezi az összes parkolóhelyet, ezekhez tartozó jegyeket elmenti és a továbbiakban bármiféle jeggyel kapcsolatos változást figyel. Erre azért van szükség, hogy ne kézzel keljen jegy azonosítókat beirogatni, hanem létező jegyekkel probáljunk meg műveleteket végezni. Képes lekérdezn egy kiválasztott jegynek az árát, majd ki tudja fizetni.

TargetFramework: .NET 8.0 C#12

Applikáció típusa: Winforms Application

A Smart Parking Monitoring fájljait használja a szerverhez kapcsolodó kommunikációhoz, egyben ugyan az a felépítése az x ábrán látottakhoz, kivétel a ParkingMapView helyén a Form1 foglal helyet ennek attribútumai, metódusai mások, viszont a kapcsolat rendszer ugyan az.

```
private void Client_ClientDiscConnected(object? sender, Socket e)
```

Paraméterek:

- `object? Sender` Függvényt meghívó objektum (ServerToClientConnection)
- `Socket disconnectedSocket` Lekapcsolódott Socket

Funkció: Ha a szerver valamilyen oknál fogva lecsatlakoztatott minket, akkor új ServerToClientConnection-t kell létrehozni, majd engedélyezni az újrakapcsolódás lehetőségét.

```
private void MessageRecieved(object? sender, EventArgs e)
```

Paraméterek:

- `object? sender` függvényt meghívó objektum.(ParkingMessageQueue)
- `EventArgs e` Esemény argumentumok, jelen esetben nem használtak, EventHandler használata miatt közelető.

Kivétel: ArgumentException ha véletlenül rossz eseményre iratkoztunk fel, és a küldő típusa nem ParkingMessageQueue

Funkció: messageQueueba érkezett összes számunkra releváns üzenet kiolvasásra és az üzenetekhez tartozó „reakció” függvények meghívása. Jelen esetben számunkra releváns üzzenetek: ErrorMessage, MapSizeMessage, ParkingPlaceDetailsMessage, ParkingPlaceBookedMessage, TicketPriceMessage, TicketDetailsMessage

```
private void TicketDetailsMessageReceived(TicketDetailsMessage msg)
```

Paraméter: Fogadott üzenet.

Funkció: Ha egy jegy tulajdonságai megváltoztak, akkor a tickets listát és listbox-ot ennek megfelelően frissíteni kell.

```
private void MapSizeMsgReceived(MapSizeMessage msg)
```

Paraméter: Fogadott üzenet.

Funkció: Eltárolni, hogy hány darab parkolóhelyet várunk a szervertől.

```
private void ParkingPlaceBookedMsg(ParkingPlaceBookedMessage msg)
```

Paraméter: Fogadott üzenet.

Funkció: Ha egy parkolóhelyet kibéreltek, akkor a hozzá tartozó jegyet elmenti és frissiti a listbox-ot.

```
private void ParkingPalaceDetailsMessageReceived(ParkingPlaceDetailsMessage msg)
```

Paraméter: Fogadott üzenet.

Funkció: Aktív ticketek feltöltésére és meglévő ticketeket frissítését itt tudjuk elvégezni.

```
private void TicketPriceMessageReceived(TicketPriceMessage msg)
```

Paraméter: Fogadott üzenet.

Funkció: A szerver válasza a jegy automatának TicketPriceRequest után, ennek tartalmát a label2-n jelenítjük meg.

```
private async void button1_Click(object sender, EventArgs e)
```

Paraméterek: ButtonClickEventHandlerhez köthetőek, jelen esetben nincsenek használva.

Funkció: Szerverhez csatlakozás aszinkron módon, és üzenetek fogadásának elindítása új szálon

```
private void button2_Click(object sender, EventArgs e)
```

Paraméterek: ButtonClickEventHandlerhez köthetőek, jelen esetben nincsenek használva.

Funkció: A listbox-ból kiválasztott jegyhez kérvényezünk a szervertől egy árat.

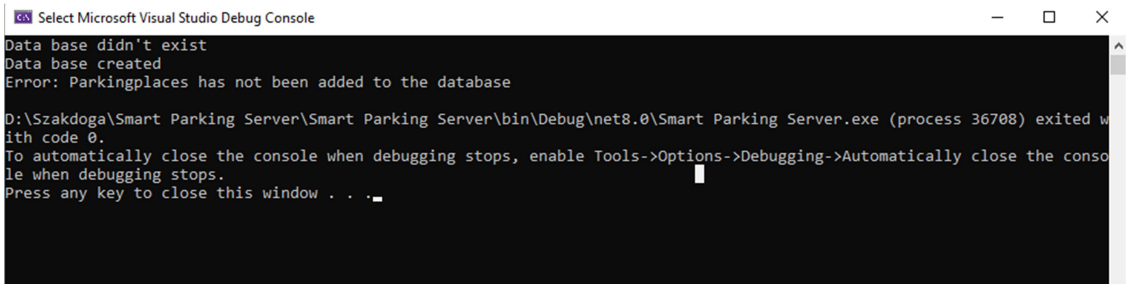
```
private void button2_Click(object sender, EventArgs e)
```

Paraméterek: ButtonClickEventHandlerhez köthetőek, jelen esetben nincsenek használva.

Funkció: A listbox-ból kiválasztott jegyhez küldünk a szervernek egy TicketPaymentMessage-t PaymentResult.Succesful paraméterrel, azaz kifizetjük a jegyet.

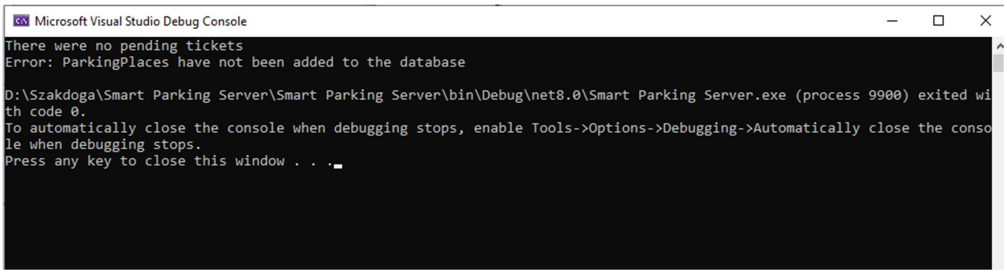
Tesztelés

Szerver elindítása adatbázis fájl nélkül



7. Ábra A szerver nem indul el, és tájékoztat minket a problémáról.

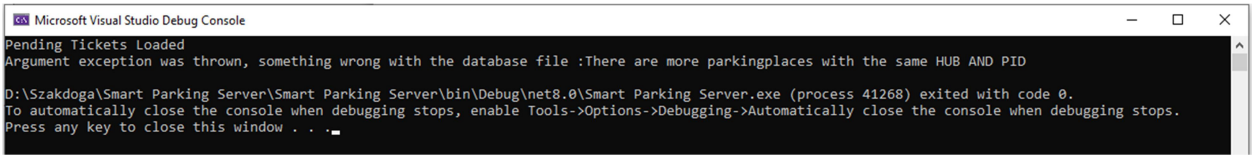
Szerver elindítása létező adatbázis fájllal, viszont Parkolóhelyek nincsenek definiálva



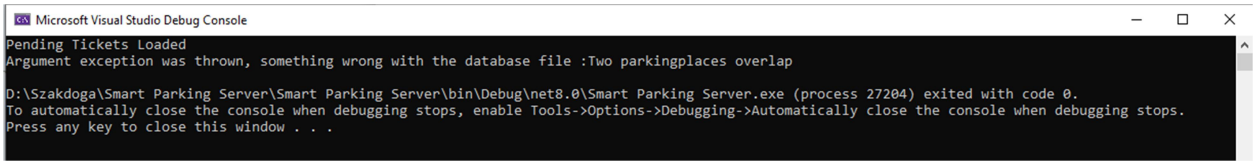
8. Ábra A szerver nem indul el és tájékoztat minket a problémáról

Szerver elindítása korrupt adatbázis fájllal

	Id	Height	HubID	Level	PID	PosX	PosY	Type	Width
	Filter	Filter	Filter	Filter	Filter	Filter	Filter	Filter	Filter
1	1	2	1	0	1	0	0	2	2
2	2	2	1	1	1	3	0	2	2
3	3	2	1	0	3	6	0	1	2

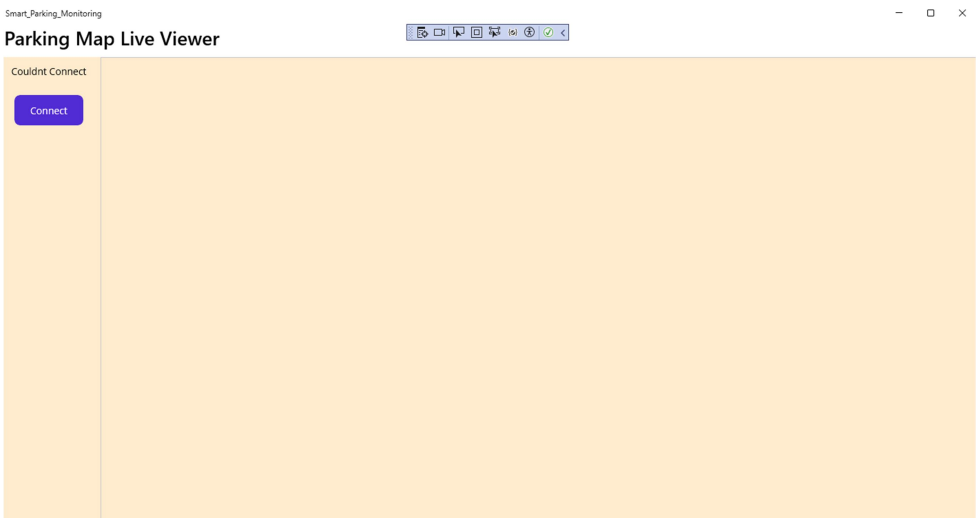


9. Ábra Valamely PID ÉS HUBID duplikálva szerepel, hibát jelez



10. Ábra Valamely parkolóhelyek között átfedés van, a szerver nem indul el, tájékoztat a problémáról

Motirozó alkalmazás csatlakozni próbál, amikor a szerver offline



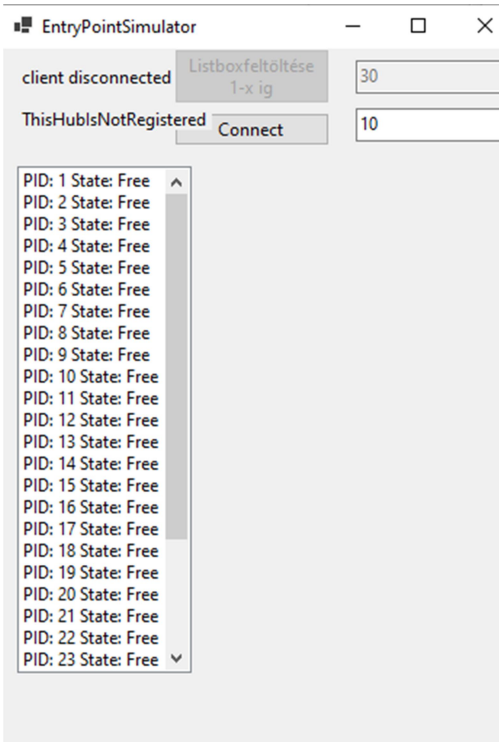
11. Ábra Csatlakozás sikertelen, az applikáció nem omlik össze

A Szerverrel a kapcsolat megszakad

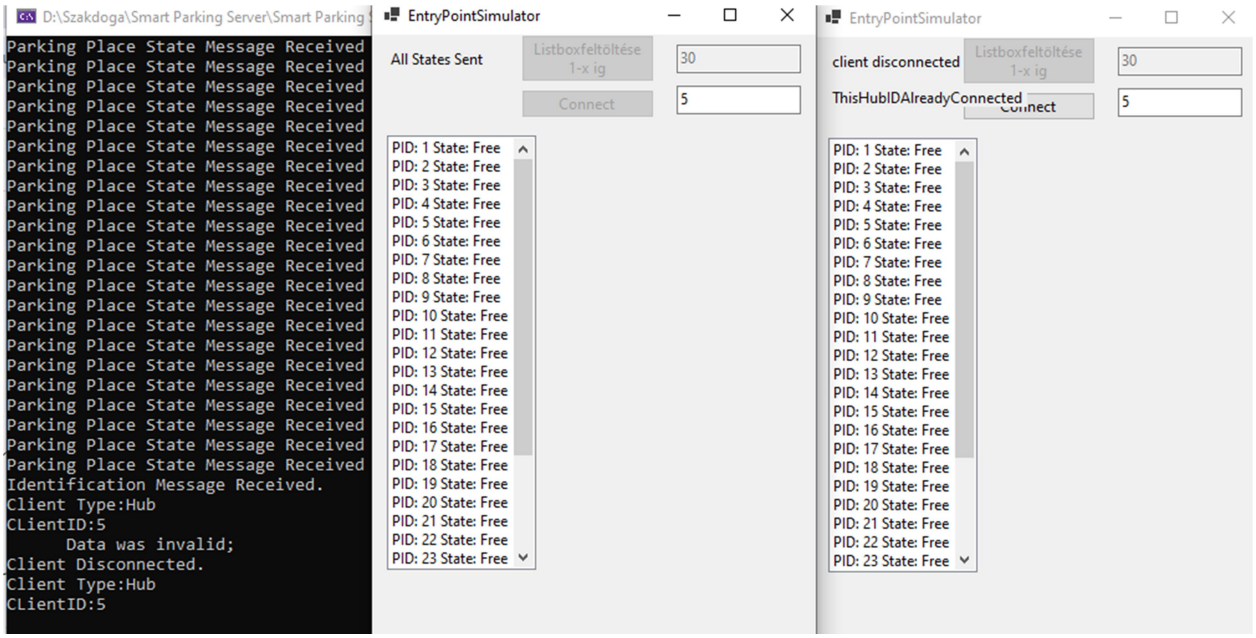
12. Ábra Csatlakozás megszakadt, az applikáció nem omlik össze

Megjegyzés: Ezeket a műveletek nem fogom elvégezni a szimulátor programokon, mivel ugyan azokat a fájlokat használja, mint a monitorozó alkalmazás.

ParkingHub szimulátorral nem megfelelő csatlakozási feltételek

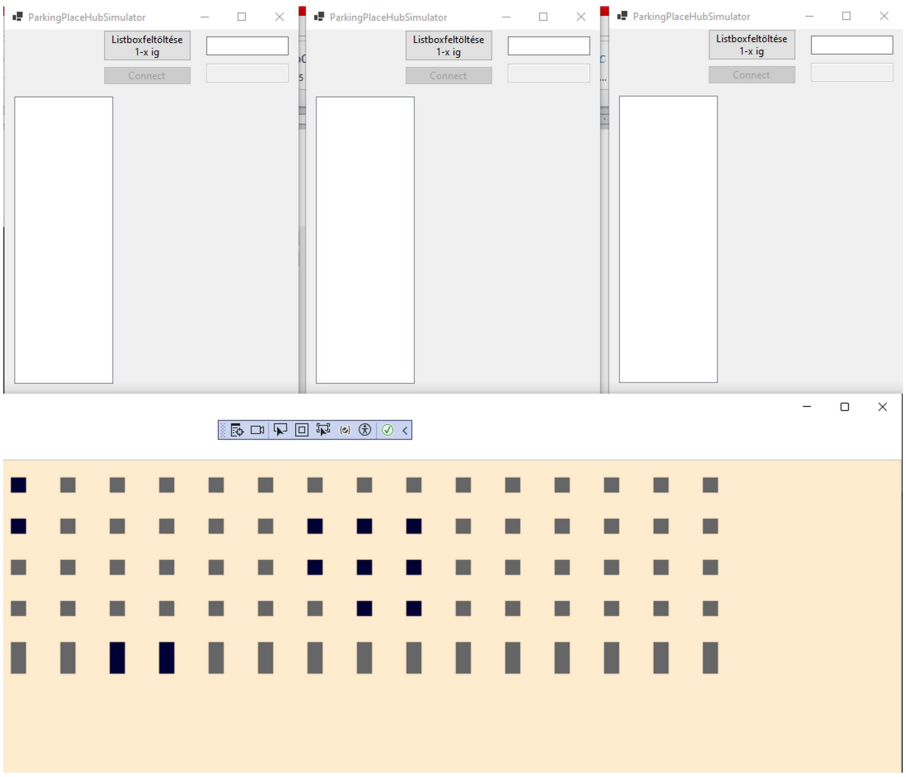


13. Ábra Adott HubID hez nem tartozik parkolóhely az adatbázisban



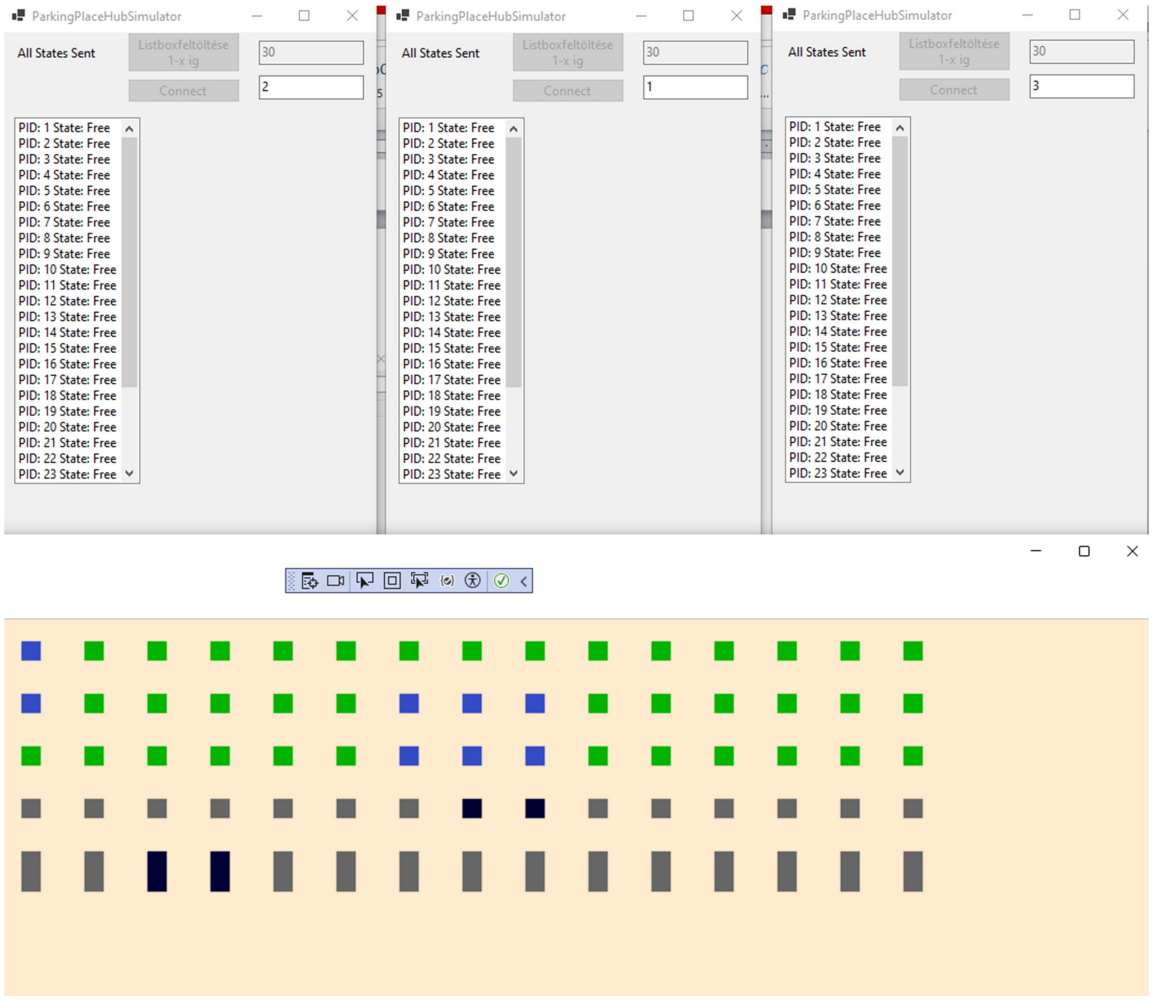
14. Ábra Olyan ID-vel szeretnénk csatlakozni, amely már használatban van.

Unkown állapot, ha Hubok nincsenek csatlakozva



15. Ábra Egyik parkolóhely állapota se elérhető

Aktuális állapot kijelzése, ha a hub csatlakoztatva van

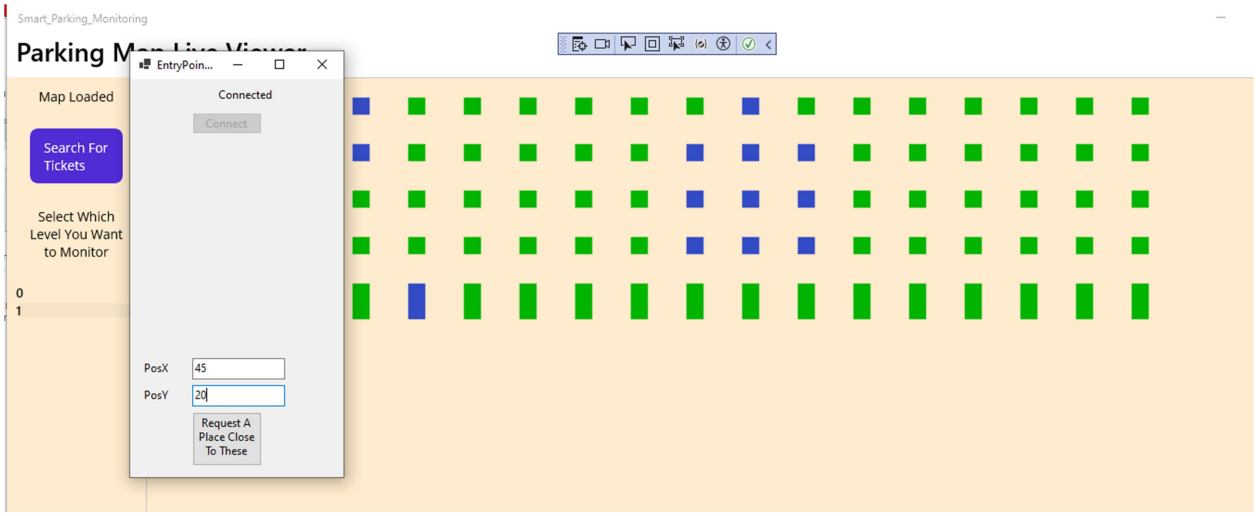


16. Ábra az első három sorhoz csatlakoztatott Hubok elérhetőek, ezért állapotuk látható

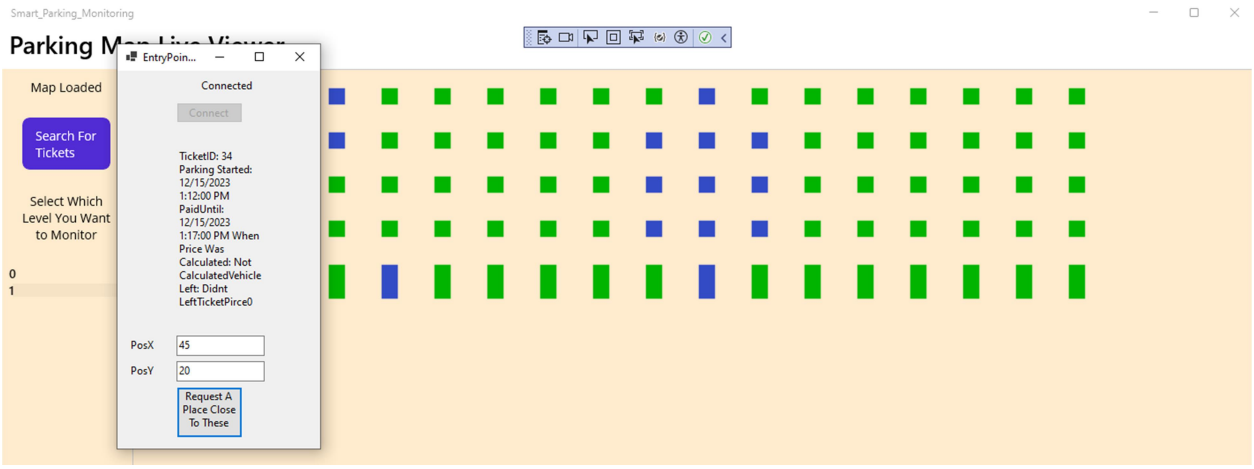
Hub-bal való kapcsolat vesztés

17. Ábra A kettes Hub lecsatlakozott, hozzá tartozó parkolóhelyek nem elérhetőek

Legközelebbi parkolóhely igénylése egy adott pozicióhoz

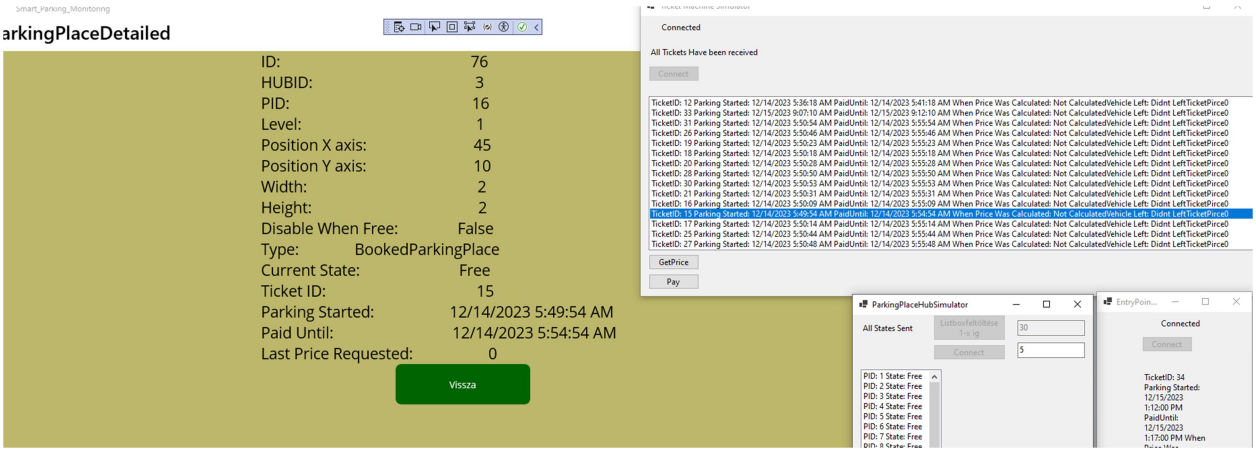


18. Ábra Erre a lekérdezésre az ötödik sor 8 elemét kell lefoglalni a szervernek

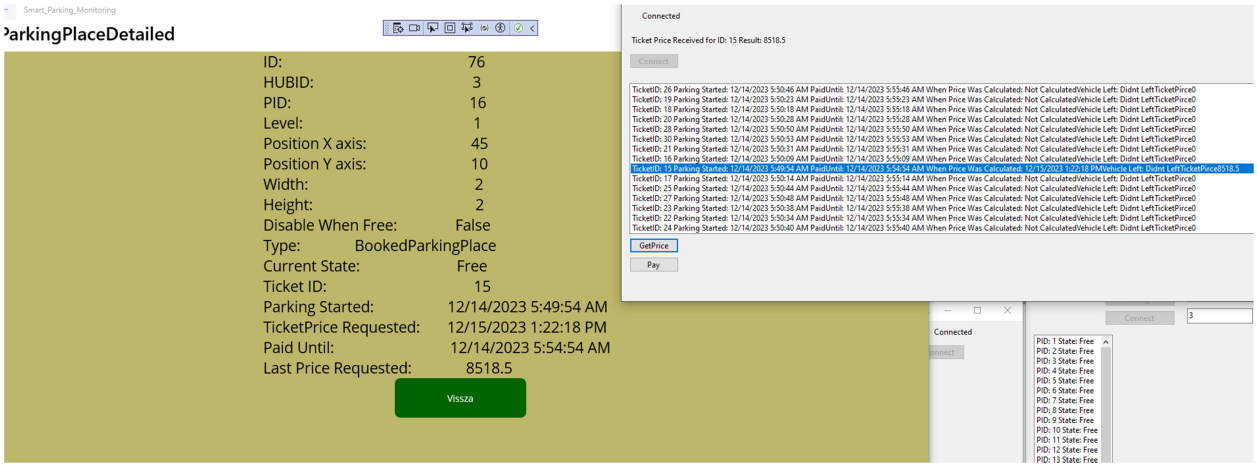


19. Ábra A szerver megfelelő parkolóhelyet adott vissza

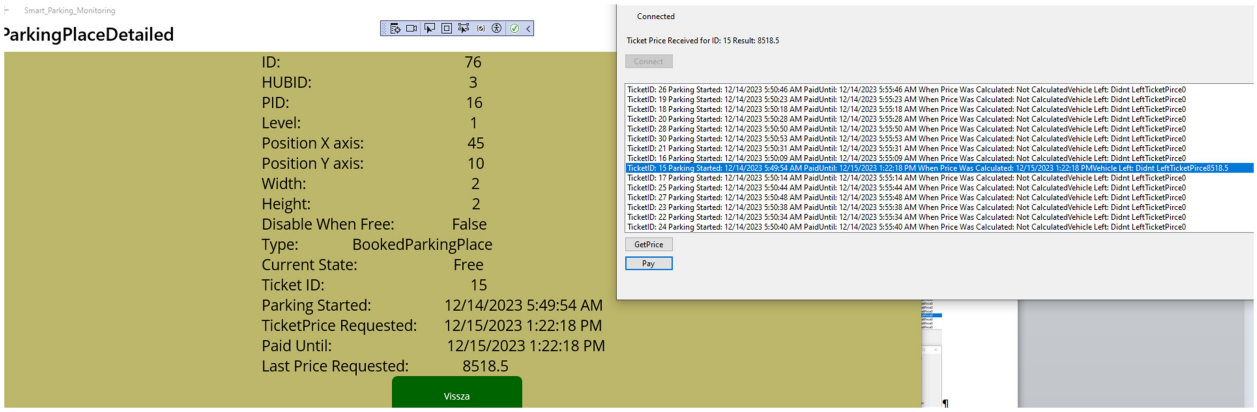
Jegy árának lekérdezése, majd kifizetése



20. Ábra 15-ös azonosítóval rendelkező jegy és parkolóhely értékei lekérdezés elött

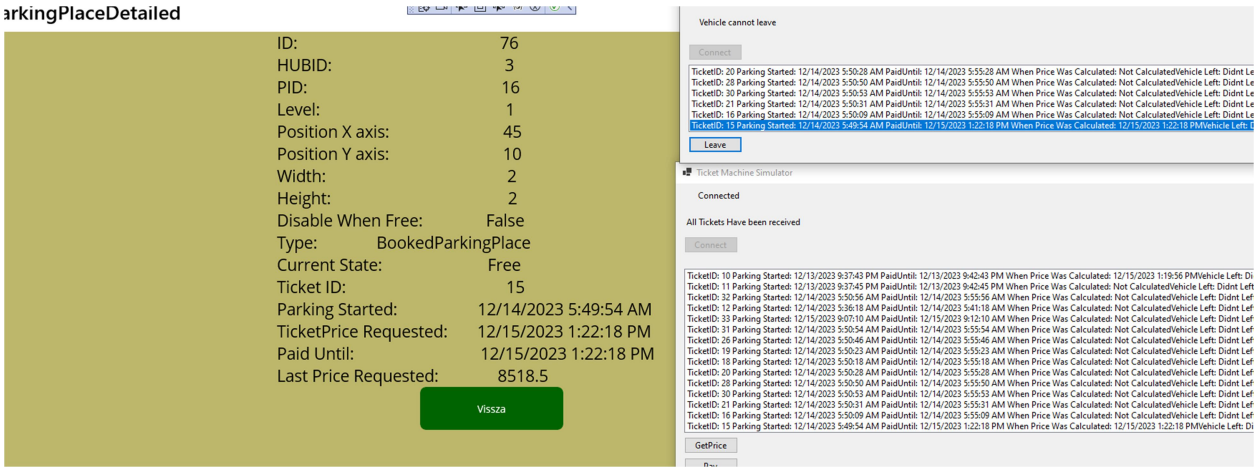


21. Ábra 15-ös azonosítóval rendelkező jegy és parkolóhely értékei lekérdezés ár lekérdezése után



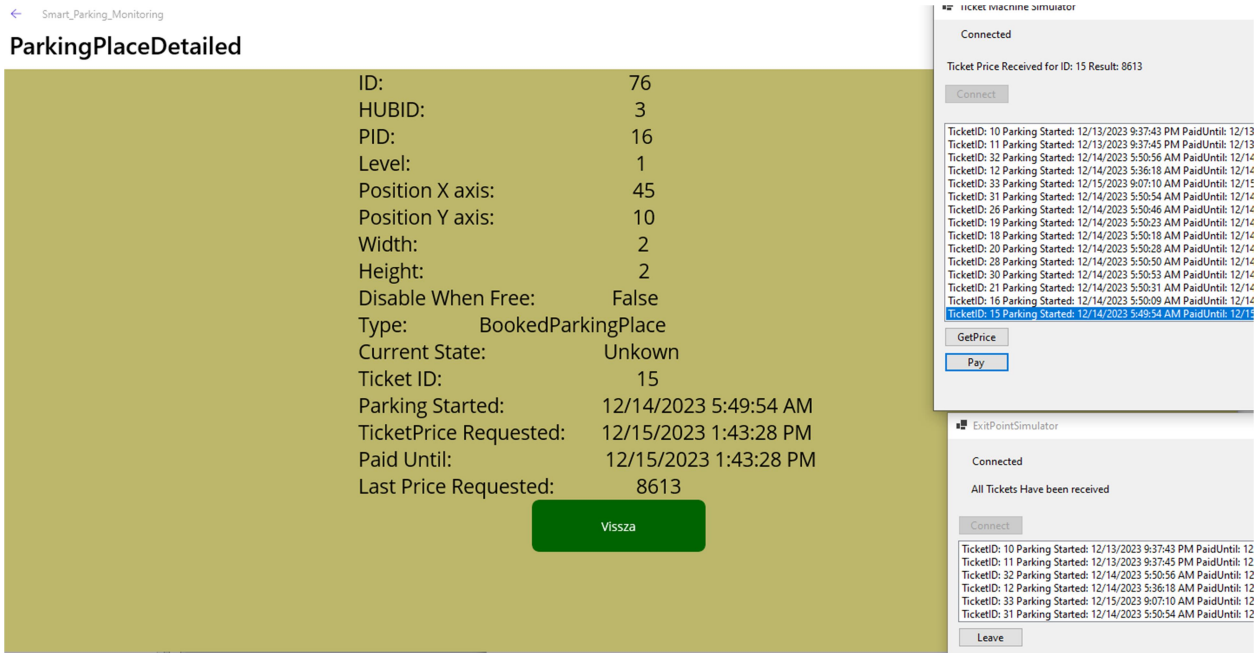
22. Ábra 15-ös azonosítóval rendelkező jegy és parkolóhely értékei kifizetése után

Kiléptetés sikertelen

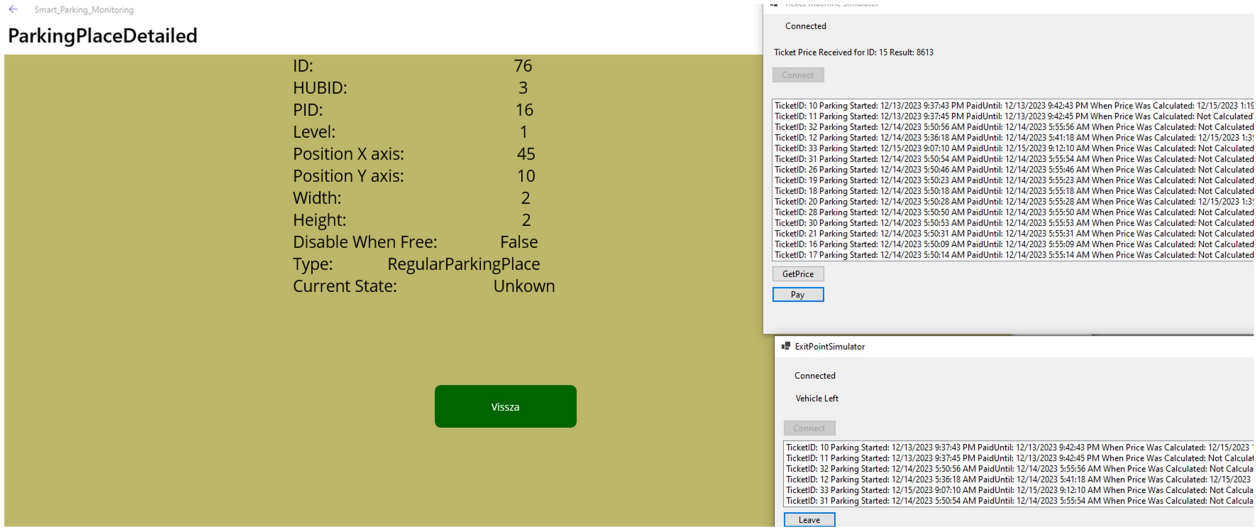


23. Ábra Több mint öt perce fizette ki a parkolóhelyet, ezért sikertelen

Kiléptetés sikeres

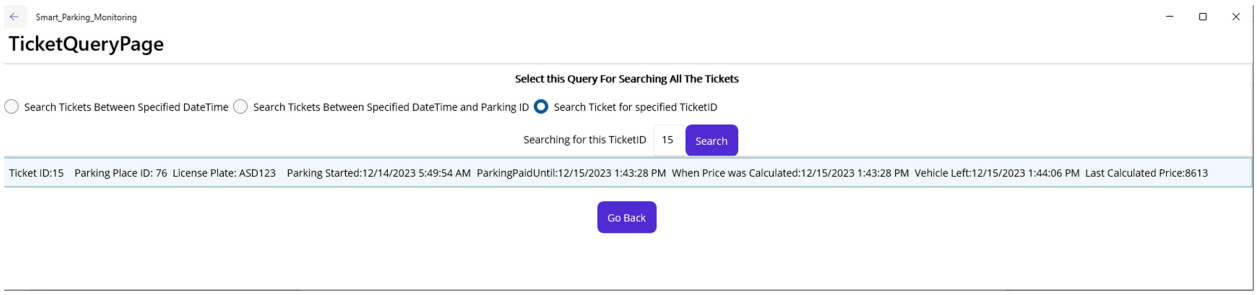


24. Ábra Az ár kifizetve

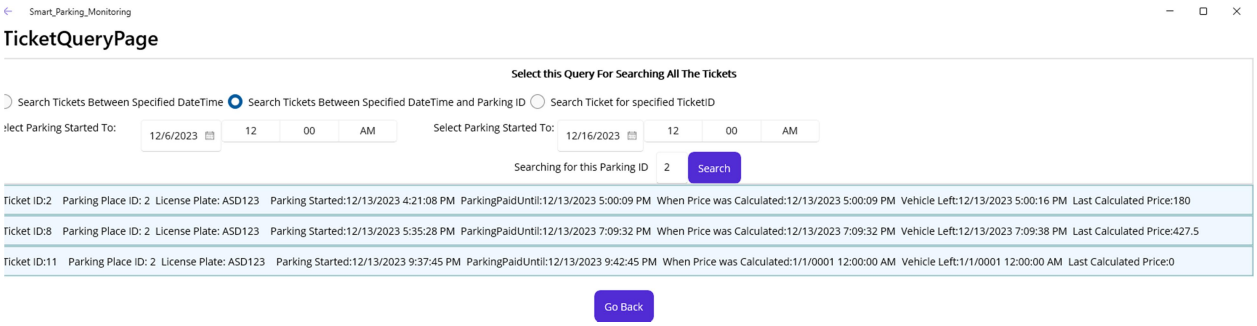


25. Ábra Jármű kliléptetve, újból kiadható a parkolóhely

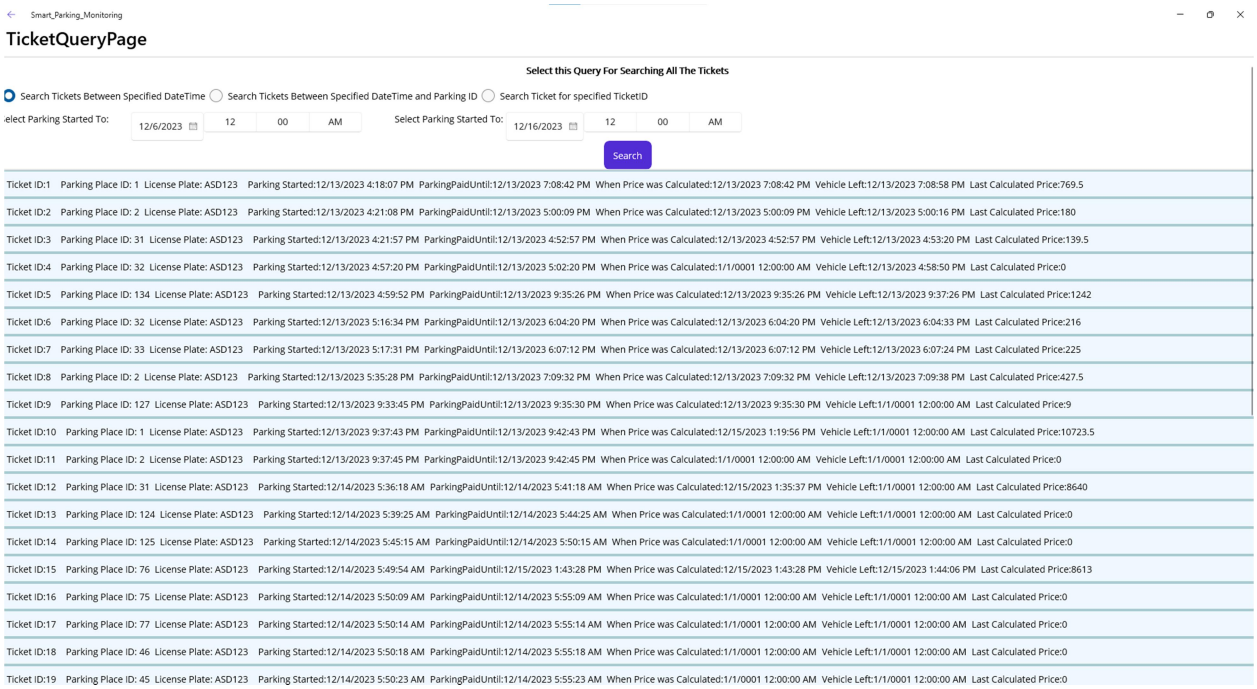
Monitoring alkalmazással jegyek lekérdezése



26. Ábra Csak jegy azanosítóra való keresés



27. Ábra Adott intervallumra keresés, parkoló azonosítóval



28. Ábra Adott intervallumra keresés parkoló azonosító nélkül

Fejlesztési lehetőségek:

A dokumentáció olvasása során nyilvánvalóvá válhatott, hogy az alkalmazás semmiféle titkosítást nem használ kommunikáció során. Ezen mindenképp változtatni kellene, hiszen így könnyűszerrel lehallgatható és manipulálható. Hiába zárt hálózaton működne, hálózati megoldásokkal nem garantálható a biztonságossága (pl. MAC-cím szűrés, stb). Egy ilyen titkosítási protokoll implementálása megoldható lenne, nagy strukturális változtatások nélkül. Mindössze a ParkingMessageQueue üzenet feldolgozása (ParseMessage) és Üzenet létrehozása (CreateMessage) közé kellene egy extra réteget létrehozni.

Ideális lenne még a monitoring szolgáltatást web applikációként tovább fejleszteni, mivel észrevételem szerint a mai irányvonal e felé mozog, egyre kevésbé szeretik az emberek a telepíthető alkalmazásokat.

Parkoló típusokat kibővíteni, úgy, hogy tartós bérleti szolgáltatásokat is lehessen nyújtani és erre külön árat kínálni, továbbá mozgássérült parkolóhelyek létrehozására is implementálni.

Monitoring rendszer grafikus felületét átalakítani, jelenleg a színek nem éppen ergonomikusan vannak kiválasztva.

Létrehozni egy grafikus parkolóház tervező szoftvert, amelynek segítségével konfigurálni lehet a parkolóhelyeket. Jelenleg kézzel szükséges bevinni a parkolóhely adatokat az adatbázisba.

Valamilyen megoldást találni arra, ha egy éppen kibérelt, de nem elfoglalt parkolóhelyre áll be egy illetéktelen, akkor azt is jelezni tudjuk.

Végezetül természetesen belépési, kilépési pontok, valamit a jegy automata és Hub-szenzor rendszer fizikai megvalósítása.

IV. HASZNÁLATI ÚTMUTATÓ

Szerver

Első futtatás során még nincs adatbázisunk, ezért létrehozza azt, majd leáll a program. Ezek után nekünk kell kézzel, egy adatbázis szerkesztő programmal a ParkingPlaces táblázathoz feltölteni parkolóhelyekkel. Ügyelni kell arra, hogy HubID ÉS PID kombinációja másodlagos kulcsként funkcionál, tehet ezek nem duplikálódhatnak, valamint a parkolóhelyek ne lógjanak rá egymásra. Ha mégis elrontanánk, akkor a indításakor ezt jelzi nekünk, és leáll a program.

	Id	Height	HubID	Level	PID	PosX	PosY	Type	Width
	Filter	Filter	Filter	Filter	Filter	Filter	Filter	Filter	Filter
1	1	2	1	0	1	0	0	2	2
2	2	2	1	1	2	3	0	2	2
3	3	2	1	0	3	6	0	1	2
4	4	2	1	1	4	9	0	1	2
5	5	2	1	0	5	12	0	1	2
6	6	2	1	1	6	15	0	1	2
7	7	2	1	0	7	18	0	1	2
8	8	2	1	1	8	21	0	1	2

29. Ábra ParkingPlaces tábla adatai

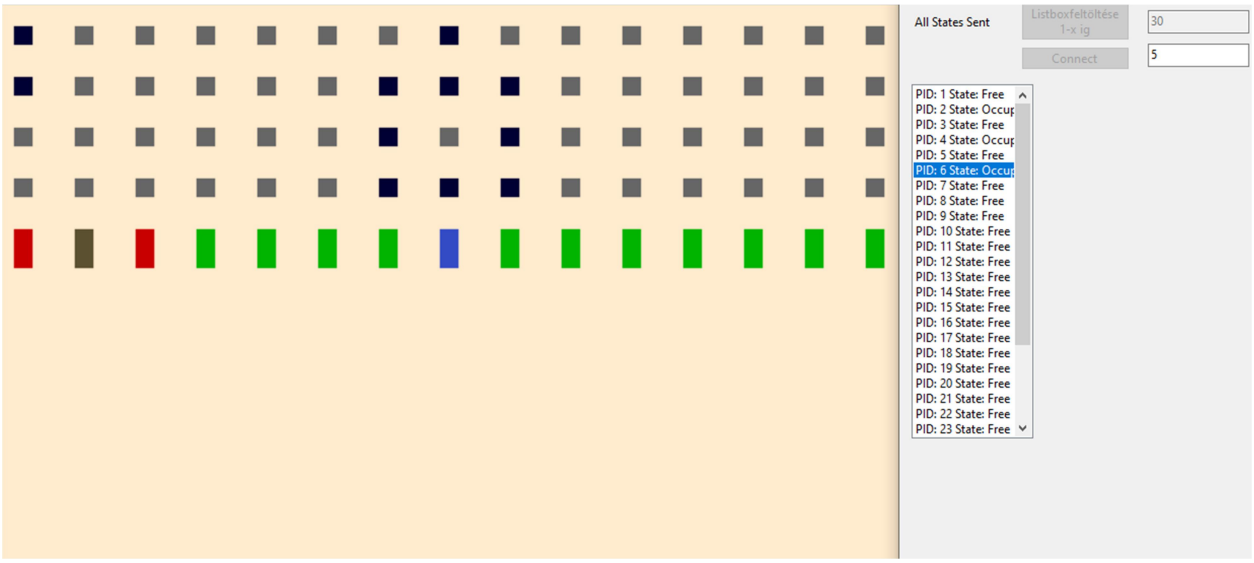
- ID: autómatikusan generált elsődleges kulcs
- Height: a parkalóhely hosszúsága
- HubID: melyik hubhoz tartozik
- Level: melyik szinten található
- PID: hanyadik parkolóhely az adott Hub-on
- PosX: elhelyezkedése az X tengelyen
- PosY: elhelyezkedése az Y tengelyen
- Type: 1, általános parkolóhely
- Width: szélessége

Ha ezek konfigurálásával megvagyunk, akkor elindíthatjuk a szerver programot, amely ha nem talált hibát, akkor kíirja, hogy “Server Started”, innentől kezdve fogad információkat a kliensektől.

Hub emulator

Csatlakozás előtt meg kell adnunk, hogy hány darab parkolóhelyet szimulál, ezt a job felső sarokban lévő szöveges mezőbe kell beírni, és 0-nál nagyobb egész számot kell megadni, különben nem tudunk csatlakozni a szerverhez.

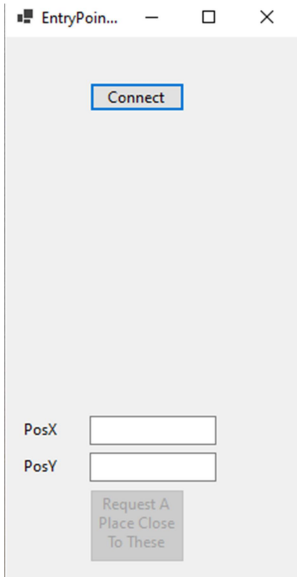
Ha ez megtörtént, akkor a jobb felső sarokban elérhetővé vált egy másik szöveges mező, ide kell beírni a Hub azonosítóját, amelynek szintén 0-nál nagyobb, pozitív egész számnak kell lennie. Most már megpróbálhatunk csatlakozni a szerverhez a Connect gombra kattintva. Kezdő állapotként minden parkolóhely státusza szabad, azonban, ha kétszer rákattintunk, akkor foglalt helyzetbe tehetjük őket, újbóli dupla kattintásra pedig megint szabaddá.



30. Ábra Csak az 5-ös hub van csatlakoztatva jelenleg, látható, hogy a benne lévő állapotok megfelelnek a monitorozó alkalmazásban szereplőknek

Belépési pont szimulátor

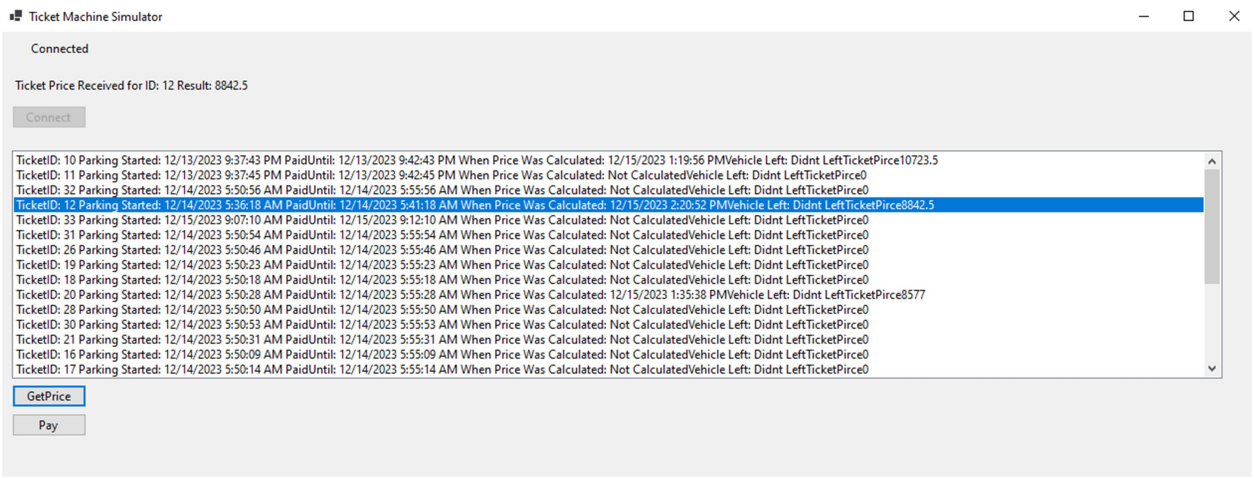
Program elindítása után, bármiféle információ megadása nélkül csatlakozni tudunk a szerverhez, amelynek sikerességéről tájékoztat minket. Ezek után lehetőségünk van a PosX és PosY-hoz tarozó szöveges mezőkkel, ehhez a lokációhoz közeli parkolóhelyet igényelni.



31. Ábra Belépési pont szimulátor

Jegy kiadó automata

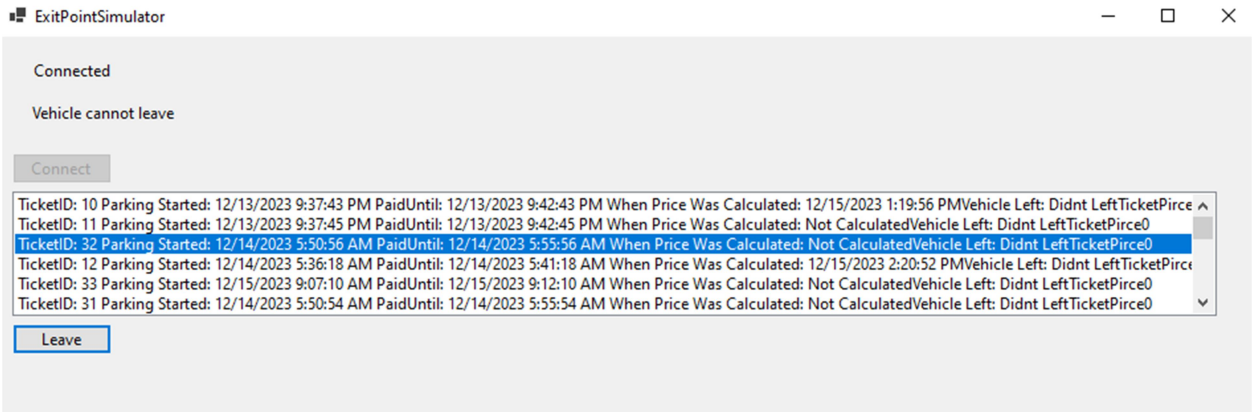
Csatlakozás után, egyből megjelennek a jelenleg "élő" jegyek, amelynek lekérdezhetjük az árát és kifizethetjük. Ennek menete a következő: listbox elemei közül kiválasztjuk azt a jegyet, amin a műveletet szeretnénk végezni majd megnyomjuk a művelethez tartozó gombot



32. Ábra Jegy Kiadó automata

Kilépési pont:

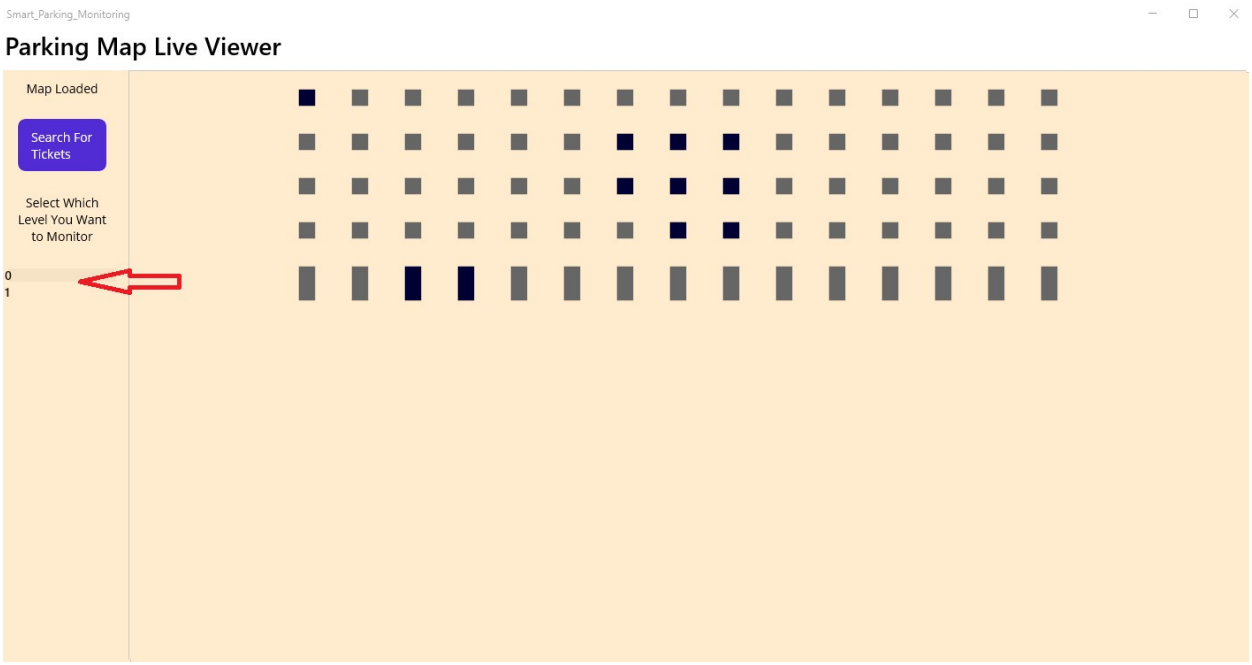
Csatlakozás után, egyből megjelennek a jelenleg “élő” jegyek, amelyeket megpróbálhatunk kiléptetni a rendszerből. Ha sikerült akkor törlődik a lista elemei közül az adott jegy, és a a bal felső sarokban kapunk értesítést arról, hogy az adott jármű elhagyhatta-e a garázst.



33. Ábra Kilépési pont

Monitorozó rendszer

A Connect gombbra kattintva tudunk csatlakozni a szerverhez, a csatlakozás állapotáról a bal felső sarokban kapunk információkat, amennyiben sikerült, ki kell választanunk, hogy melyik szintet szeretnénk megjeleníteni. Innentől kezdve láthatjuk a parkolóház aktuális állapotát és annak változásait.



34. Ábra Szint választó

Általános parkolóhely, amely nincs elfoglalva

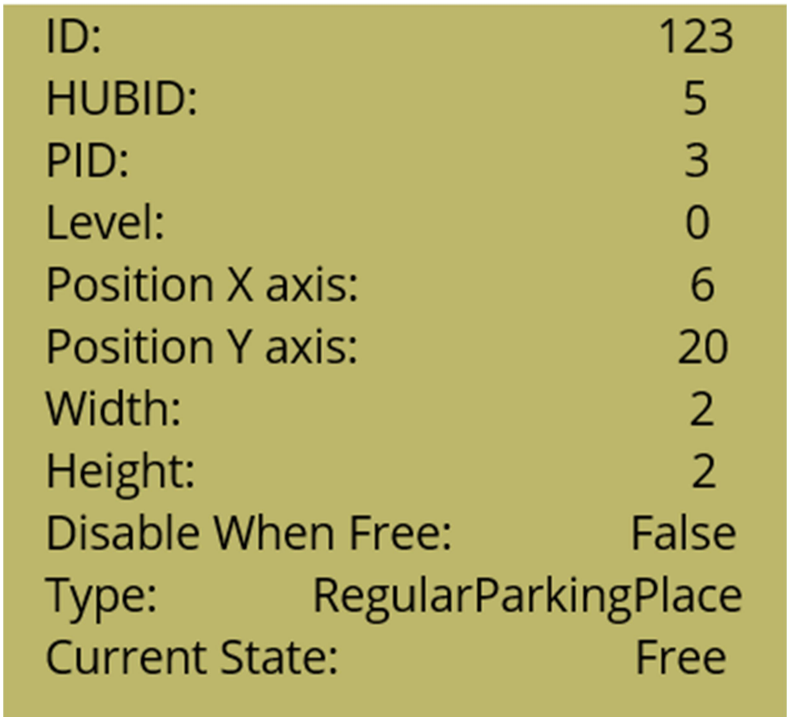
Általános parkolóhely, amelynek foglalt az állapota (tilos)

Általános parkolóhely, amelynek ismeretlen a státusza

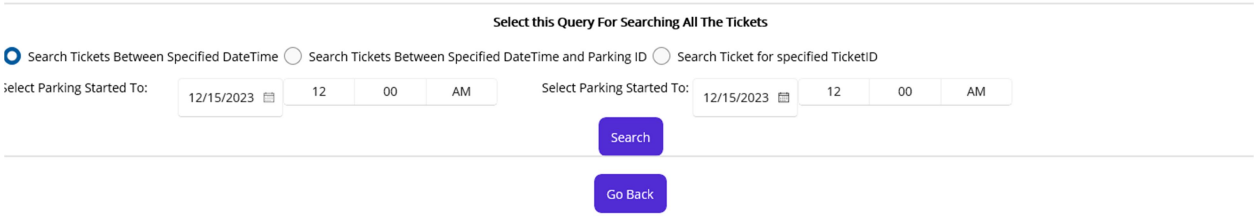
Kiadott parkolóhely, ahol jelenleg nem áll senki.

Kiadott parkolóhely, amelynek állapota ismeretlen

Kiadott parkolóhely, amelyet elfoglaltak.



35. Ábra Parkolóhelyre kattintva minden hozzá tartozó információ részletesen megjelenik



36. Ábra Search For Tickets Gombra kattintva végezhetünk lekérdezéseket jegyekkel kapcsolatban

V. ÖSSZEGZÉS

Habár az adott rendszer specifikációnak megfelelőnek működik, gyakorlati alkalmazását nem feltétlen látom működőképesnek, vagy csak hosszú próbaszakasz után vezetném be élesbe. Ez egy teljesen új megközelítés, amely merően eltér a parkolóházak általános szabályaitól, véleményem szerint nagyon sok tiltott helyre történő parkolás következhetne be.

Ha valaki mégis próbálkozna vele, akkor szerintem a próba szakasz alatt, ***jutalmazni*** kellene azokat, akik megfelelően használják a parkolóházat (árengedmény), ezzel motiválva az ügyfeleket a szabályok betartására, de **nem** megbüntetni, akik nem tartják be. Csak egy ilyen több hónapos teszt szakasz után döntenék arról, hogy alkalmazható-e ez a konstrukció, ha igen, akkor már nem jutalmazást alkalmaznék, hanem büntetést vetnék ki a szabálytalankodókra.

VI. SUMMARY

Although the given system works in accordance with the specifications, I do not necessarily see its practical application as functional, or I would introduce it into production only after a long trial period. This is a completely new approach, which is quite different from the general rules of parking garages, and in my opinion, there could be a lot of parking in prohibited places.

If someone were to try it, I think that during the trial period, those who use the parking garage properly should be rewarded (price discount), thereby motivating customers to follow the rules, but those who do not should not be punished. Only after such a test period of several months would I decide whether this construction could be used, if so, I would no longer apply rewards, but would impose a penalty on violators.

VII IRODALOMJEGYZÉK

(1) Packt Publishing - Software Architecture with C# 9 and .NET 5 második kiadás
(írta: Gabriels Baptista, Francesco Abbruzzese Megjelenés: 2020.December)

Megjegyzés: A szakdolgozatomban semmilyen forrásmegjelölés nincs, ezt a könyvet arra használtam, a C# és .Net fejlesztői környezetet megismerjem.