ÓBUDAI EGYETEM
ÓBUDA UNIVERSITY

NEUMANN JÁNOS
INFORMATIKAI KAR

SZAKDOLGOZAT

OE-NIK Hallgató neve: **Luxeder Zoltán**

2022 Hallgató törzskönyvi száma: **T008818/FI12904/N**

Óbudai Egyetem
Neumann János Informatikai Kar
Szoftvertervezés és -fejlesztés Intézet

SZAKDOLGOZAT FELADATLAP

Hallgató neve:	**Luxeder Zoltán**
Törzskönyvi száma:	T008818/FI12904/N
Neptun kódja:	[redacted]

A dolgozat címe:

Nyilvános használatú internet terminál infrastruktúra fejlesztése
Development of a public access internet terminal infrastructure

Intézményi konzulens:	Kovács András
Külső konzulens:	
Beadási határidő:	2022. május 15.
A záróvizsga tárgyai:	Szoftvertechnológia és grafikus felhasználói interfész tervezése Szoftvertervezés és -fejlesztés specializáció

A feladat

A feladat egy számítógép fizetős internet terminálként való használatát lehetővé tevő rendszer fejlesztése. A fő komponens egy WPF alkalmazás legyen, ahol a felhasználó számára elérhető funkciók vannak felsorolva. Vagyis az operációs rendszer legtöbb funkciója ne legyen elérhető, csak internetezésre, illetve opcionálisan a kijelölt programok használatára legyen lehetőség kontrollált módon. A felhasználó a megfelelő összeg bedobása után a terminál percalapú használatára legyen jogosult. Ennek megvalósításához tervezzen egy mikrokontroller alapú áramkört, amely interfészként funkcionál az alkalmazás, és egy választott érmeelfogadó egység között. Az egyenleg lejárta után kerüljön leállításra a felhasználó munkamenete. Bejelentkezés után legyen lehetőség a még fel nem használt egyenleg mentésére, mely később bármelyik azonos internet terminálon felhasználható. Ehhez a hátteret egy ASP.NET Core API végpont szolgáltassa. A WPF alkalmazás diagnosztikai/konfigurációs adatokat is küldjön ennek a végpontnak, amelyek alapján egy frontend alkalmazásban összesítve láthatjuk a terminálok jelenlegi alapvető adatait (Pl.: állapot, bevétel stb…)

A dolgozatnak tartalmaznia kell:

- a szakirodalom részletes áttekintését és értékelését,
- a megoldandó feladat pontos leírását és részletes elemzését,
- a meglévő rendszerek elemzését,
- az alkalmazni kívánt technológiák elemzését, döntések indoklását,
- a rendszertervet,
- a megvalósítás pontos leírását,
- a tesztelési tervet, teszteredményeket, a fejlesztés során felmerült problémák elemzését,
- az elkészült rendszer felhasználási és továbbfejlesztési lehetőségeit

ÓBUDAI EGYETEM Neumann János Informatikai Kar 1. Budapest

Ph

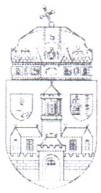
.....
Dr. Vámossy Zoltán
intézetigazgató

A szakdolgozat elévülésének határideje: **2024. május 15.**
(OE TVSz 55.§ szerint)

A dolgozatot beadásra alkalmasnak tartom:

.....
külső konzulens

.....
intézményi konzulens



ÓBUDAI EGYETEM
ÓBUDA UNIVERSITY

Neumann János Informatikai Kar

HALLGATÓI NYILATKOZAT

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

Budapest, 2022. 05. 09.

Lukács Zoltán

hallgató aláírása

ÓBUDAI EGYETEM
ÓBUDA UNIVERSITY

Neumann János Informatikai Kar

KONZULTÁCIÓS NAPLÓ

Hallgató neve: LUXEDER ZOLTÁN

Neptun Kód: [redacted]

Tagozat: NAPPALI

Telefon: [redacted]

Levelezési cím (pl.: lakcím): [redacted]

Szakdolgozat címe magyarul:

NYILVÁNOS HASZNÁLATÚ INTERNET TERMINÁL INFRASTRUKTÚRA FEJLESZTÉSE

Szakdolgozat címe angolul:

DEVELOPMENT OF A PUBLIC ACCESS INTERNET TERMINAL INFRASTRUCTURE

Intézményi konzulens: KOVÁCS ANDRÁS

Külső konzulens:

Kérjük, hogy az adatokat nyomtatott nagy betűkkel írja!

Alk.	Dátum	Tartalom	Aláírás
1.	2022.02.15.	IMPLEMENTÁCIÓ TERVEZÉS	
2.	2022.03.25.	IMPLEMENTÁCIÓ ÁTTEKINTÉSE	
3.	2022.04.10.	TESZTELÉSEK KIÉRTÉKELÉSE	
4.	2022.05.10.	VÉGSŐ ELLENŐRZÉS, LEZÁRÁS	

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

A hallgató a Szakdolgozat II. tantárgy követelményét teljesítette, védésre bocsátható.

.....

Intézményi konzulens

Budapest, 2022. május 10.

TARTALOM

1. TARTALMI ÖSSZEFOGLALÓ

Szakdolgozatom során egy olyan megoldás fejlesztésének menetét fogom végig követni, mely lehetővé teszi, egy hagyományos számítógép publikus, fizetős internet terminálként történő használatát. Áttekintem a már létező megoldásokat, majd az ennek során megszerzett ismeretek fényében dokumentálom egy ilyen infrastruktúra elkészítésének a folyamatát.

2. BEVEZETÉS

21. században internetes tartalmakhoz egyre többen kívánnak hozzájutni.

Egy 2020 januári tanulmány alapján a 15-69 éves magyar lakosság ~87%-a használja az internetet legalább havonta egyszer. A korábbi évek tendenciáit, és a környező országok statisztikáit figyelembe véve, ez a szám feltételezhetően továbbra is emelkedni fog. [1]

Manapság az online tartalmakhoz nagyon sokféleképpen hozzáférhetünk: Okostelefonokon, munkahelyünkön (amennyiben az biztosítja számunka), otthon, netklubokban, szórakozóhelyeken, ingyenesen látogatható könyvtárakban, teleházakban, számos városban közterületi termináloknál, oktatási intézményekben stb. [2]

Internetezési lehetőség biztosításának érdekében, személyes tapasztalatom alapján, a fentebb felsorolt helyeken meglepően sokszor azt a megoldást választják, hogy egy felügyelet nélküli asztali számítógépet biztosítanak a vendégek számara. Ez meglehetősen sok problémával járhat: Nem kívánatos szoftverek/vírusok/kémprogramok telepítése, beállítások nem kívánatos módosítása, bejelentkezési adatok véletlen történő eltárolása, csak hogy egy párat említsek. Ezeknek a kiküszöbölésére felmerülhet az igény, hogy a fentebb említett helyeken is, kontrollált módon, esetlegesen pénzért cserébe szeretné biztosítani a tulajdonos/üzemeltető az internet elérést.

A mobiltelefonok, illetve a megfizethető mobilinternet elterjedésével a fentebb felsorolt felhasználási területek jelentős része meglehetősen háttérbe szorult, de véleményem szerint nem veszítette el teljesen a létjogosultságát. Bármikor érhet minket váratlan helyzet, amikor tegyük fel, hogy lemerült a telefonunk, egy összetettebb dokumentumot kell váratlanul megtekintenünk, vagy esetleg a telefonok relatív kicsi virtuális billentyűzetével nem tudunk elég hatékonyan dolgozni.

Egy internet terminál rendelkezhet olyan hardveres adottságokkal (gondoljunk itt akár fejlett hangtechnikára, vagy általánosságban véve különféle perifériákkal való bővítési lehetőségekre), amivel egy mobil eszköz nem tudja felvenni a versenyt.

Szakdolgozatom témaválasztásának legfőbb motivációja a családi vállalkozásunk tevékenységi köre volt. Vállalkozásunk 2010 óta foglalkozik internet terminálok

gyártásával, illetve üzemeltetésével. Az általunk üzemeltetett gépek túlnyomó többsége szórakozóhelyeken, kávézókban található, ebből adódóan fizetés ellenében lehet használni őket. A rajtuk használt szoftvercsomaghoz, ebben az évben lett megvásárolva a szoftverlicensz, ami akkor még teljeskörűen lefedte az elvárásainkat.

Ezek a termináljaink mára, a 2010-es szoftvercsomagot használva már teljesen elavultnak számítanak. Csak hogy egy példát említsek, ekkoriban még Internet Explorer 8-at használt a szoftver a háttérben, ez a statisztikai adatok alapján a ma használt böngészők **0.04%-**át teszi ki. [3] Ebből adódóan a különféle weboldalak támogatása efelé a böngésző felé elhanyagolható, mondhatni nem létező.

Szakdolgozatom során egy olyan szoftver fejlesztésén fogok dolgozni, mely lehetővé teszi egy asztali számítógép fizetős internet terminálként való használatát, naprakész szoftvereket felhasználva. Megvizsgálom a már meglévő megoldásokat, és megtervezek egy olyan rendszert, ami alkalmas egy internet terminál alapvető feladatainak ellátására, miközben rendelkezik azokkal a számunkra szükséges funkciókkal, amiket ügyfeleink megszoktak a korábbi megoldásnál.

3. A SZAKDOLGOZAT SORÁN ELKÉSZÍTENDŐ SZOFTVER

3.1. Hasonló megoldások vizsgálata

Szakdolgozatom ezen részében már meglévő, hasonló problémakörre megoldást kínáló rendszerek után fogok kutatni. Az elemzésem során elsősorban funkcionalitás szerint fogom áttekinti a rendszereket, de emellett szerepet fog kapni az adott projekt üzleti modellje, flexibilitása, illetve a támogatottsága is.

3.1.1. Sitekiosk Internet terminál szoftvercsomag

A PROVISIO GmbH. által fejlesztett Sitekiosk nevű rendszer egy dedikált, internet terminálok működtetésére tervezett szoftvercsomag. Meglehetősen sok személyes tapasztalatom van a programmal kapcsolatban, mivel internet termináljaink eddig egy korábbi verziójával üzemeltek. Microsoft Windows operációs rendszereken fut, alapvető feladataként felel a számítógép biztonságos használatáért egy nyilvános környezetben is.

Használata során a felhasználó nem férhet hozzá az operációs rendszer semmilyen kezelőfelületéhez, egy korlátozott környezetben használhatja azokat az alkalmazásokat, amiket az üzemeltető a rendelkezésére bocsájt.

Az alábbiakban áttekintem a szoftver főbb, projektem szempontjából releváns funkcióit:

Kontrollált felhasználói környezet kialakítása: A szoftver telepítésekor konfigurálja az operációsrendszert, hogy a felhasználóknak gyakorlatilag semmilyen módosítási lehetősége ne legyen a rendszer beállításain. Ezek közé tartozik a kritikus billentyűkombinációk letiltása, operációs rendszer módosításaihoz lehetőséget nyújtó vizuális elemeinek elrejtése. Weboldalak megjelenítését egy beépített böngészővel teszi, ez korábban Internet Explorer alapú volt, viszont mostanra a cég átváltott Chromium alapúra. Egyedi implementációjuk lehetővé teszi, hogy a meglátogatható weboldalakat korlátozzuk, tiltó, illetve engedélyezési (angol kifejezéssel whitelist) listába szedve őket, egyedi díjazást szabjunk ki rájuk. [4]

Szoftveres működésfelügyelő (másnéven watchdog) alrendszer: Felügyeli a fő alkalmazás állapotát, amennyiben az meghatározott ideig nem válaszol, összeomlott, vagy bármilyen más, nem kívánatos állapotba kerül, helyreállítja azt, ezáltal is garantálva a nagyobb rendelkezésre állást. Ezen felül figyeli a számítógépen futó egyéb alkalmazásokat is egy beépített, de bővíthető tiltólista alapján, és leállítja azokat, ha egyezést talál, vagy beállítástól függően, ha előtérbe kerülnek.[4]

Fizetőeszköz kezelés: Létfontosságú funkció a fizetőeszközök kezelése. A program régebbi verzióiban egy meglehetősen robosztus fizetőeszköz kezelő támogatással rendelkezett, rengeteg féle kereskedelmi forgalomban kapható papír- és fémpénz elfogadó hardvert magába foglalva. [5]

Sajnos egy verziófrissítés után, minden előzetes erre utaló jel nélkül, jelentősen csökkentették a hardveres eszközök támogatását, ezzel ellehetetlenítve a későbbi frissítésékre való átállást az érintetteknek. Újabb verzió használatához, több gép esetén egy meglehetősen nagy beruházásra lett volna szükség, ami során átállnak a vevők egy újabb, érdemi új funkciókkal nem rendelkező hardverre, mindeközben pedig semmi garancia nem lett volna, hogy később, egy újabb frissítésnél nem kerülnek hasonló helyzetbe.[6]

Felhasználókezelés: Bár az alkalmazás modern verzióiból ez a funkció hiányzik, korábbiakban lehetőség volt a felhasználóknak egy központi szerveren fiókokat létrehozni, és ezen keresztül a még rendelkezésre álló egyenlegüket menteni, hogy azt később felhasználhassák bejelentkezés után. Ezen felül lehetőség volt egyszer használatos kódok generálására, mely lehetővé tette, hogy a kód bevitele után, adott összegnek megfelelő egyenleget írjunk jóvá a felhasználónak. Ez jelentős promóciós lehetőségeket hordozott magában, miközben könnyebbé tette, hogy kipróbálási lehetőséget biztosítsunk ügyfeleinknek. A fiókkezelés mellett ez is egy olyan funkció, amit mai napig aktívan használnak gépjeinkben. Ezt a két lehetőséget a Provisio cég Sitecafe szoftverjének integrálásával tették lehetővé, melyhez a szoftveres támogatás 2010 környékén szűnt meg, pár évvel később pedig a Sitekiosk szoftverből el lett távolítva az erre vonatkozó lehetőség. [7]

Választott alkalmazások elérhetővé tétele: A programban lehetőség van bizonyos alkalmazásokat ún. fehérlistára tenni, ezáltal az alapvető funkcionalitás (internet böngészés) mellett, elérhetővé válik ezeknek a használata is a felhasználó számára. Mivel a felhasználó nem fér hozzá semmihez, az operációs rendszer megszokott alkalmazás indító felületei közül, így ezeket az alkalmazásokat a Sitekisok kezdőképernyőjén indíthatjuk el. A terminál program monitorozza az ily módon indított alkalmazásokat is, ezáltal, ha azok nem kívánt műveletet végeznek (pl.: háttérbe kerül, nem megengedett ablakot nyit meg, másik alkalmazást indít), akkor bezárhatja azokat.

Felhasználóbarát UI: A terminál szoftvert rendkívül részletekbe menően személyre tudjuk szabni mind működés, mind pedig megjelenés szempontjából, a mellékelt beállításszerkesztő alkalmazással. Felhasználói interfésze letisztult, egyszerű, hogy a lehető legnagyobb felhasználói kör könnyedén el tudjon igazodni rajta.

3.1. ábra: Sitekiosk kezdőképernyője

Mindezeken felül, a szoftver rengeteg, extra funkcionalitással rendelkezik, a fentebb említettek voltak leginkább relevánsak a projekt szempontjából.

A fejlesztői cég célközönsége leginkább nagyvállalati csoportokból áll, ebből adódóan árazása nem minden esetben felelhet meg kis- és középvállalkozásoknak, magánszemélyeknek. A dolgozat írásakor 200€ volt az egy gépre történő szoftverlicensz vásárlása, melyhez 1 évig biztosítják a szoftverfrissítéseket. [8] Bár az üzleti modelljük egyszeri megvásárláson alapul, a rövid támogatottsági idő miatt bizonyos szemszögből nézve tekinthetjük előfizetéses modellnek is. Ellenkező esetben termináljaink meglehetősen hamar ki lesznek téve az elavult, biztonsági frissítéseket hiányoló böngésző, illetve egyéb szoftverek által nyújtott kockázatoknak. A frissítések telepítése bár nem bonyolult, de nem megy automatikusan végbe, ezt az üzemeltetőnek manuálisan kell elvégeznie.

3.1.2. Porteus Kiosk

Egy kis méretű, minimalista Linux disztribúció, mely a korábban vizsgált szoftverhez hasonlóan, egy zárt környezetet biztosít nyilvános internetezéshez. A felhasználó csak egy böngészőhöz férhet hozzá, egyéb programokat nem használhat, nem módosíthatja az operációs rendszer, illetve a böngésző konfigurációját. Felhasználói adatok, előzmények törlődnek használatok között, ezáltal is biztosítva az adatok biztonságát. [9]

Előnyök:

Mivel alapvetően nyílt forráskódú megoldásokat tartalmaz, így megvan a lehetőség, hogy igény szerint kisebb módosításokat végezzünk a kódban. Alapvetően a rendszer használata is ingyenes, viszont amennyiben szeretnénk, igénybe vehetünk bizonyos fizetős szolgáltatásokat, mint pl. az automatikus frissítések, amiért éves díjat kell fizetnünk. [10]

Jelentős pozitívuma ennek a megoldásnak, hogy mivel az operációs rendszer Linux alapú, így megspórolhatnánk a többi megoldásnál felmerülő Windows licensz vásárlási költségeket, ami nem elhanyagolható, amennyiben több gép üzemeltetésében gondolkozunk.

Hátrányok:

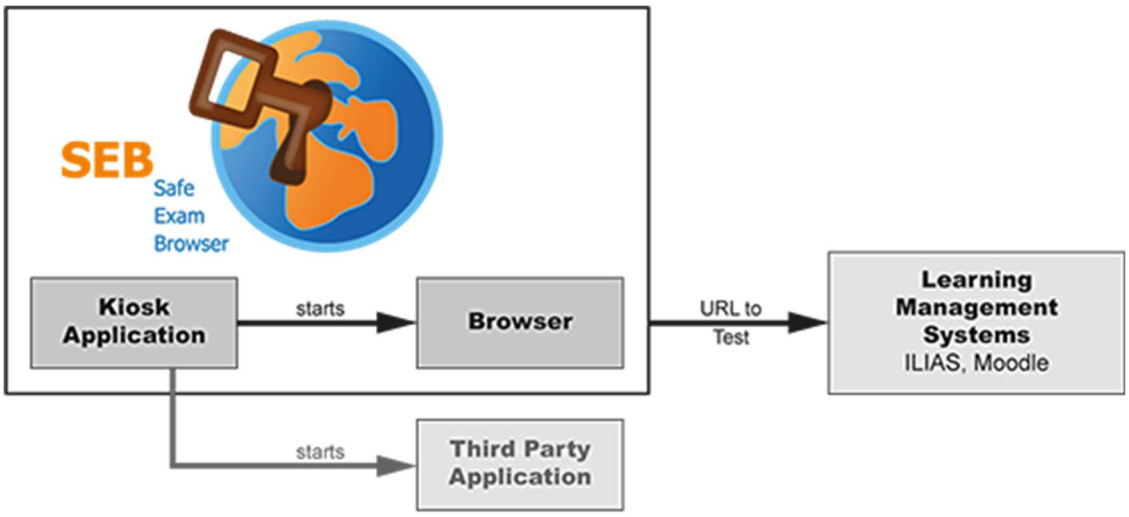
Külső módosítások nélkül meglehetősen limitált a funkcionalitása, böngésző használatán kívül nem sokat nyújt. Ideális lehet például könyvtárakban, információs pultokban, bankokban, bárhol, ahol tájékoztatós vagy promóciós jelleggel szeretnénk biztosítani az internetezés lehetőségét. Viszont bankjegy és érmefelismerő rendszer támogatás hiánya miatt nincs lehetőség fizetősként használni az internet terminált ennek a szoftvernek a használatával.

Emellett figyelembe kell vennünk, a Linux alapú operációs rendszerek alacsony elterjedtségét a Windowszal szemben. Amennyiben a webböngészés mellett semmilyen hozzáadott funkcionalitással nem bővítjük ki a rendszert, abban az esetben a végfelhasználóknak túl sok kényelmetlenséget nem okozna az idegen operációs rendszer, tekintve, hogy a böngésző megjelenése meglehetősen hasonló operációs rendszerek között. Viszont, a felhasználók mellett figyelembe kell vennünk a terminálok üzemeltetőit is, akiknek a telepítés, konfigurálás, felmerülő hibák kezelése/elhárítása kihívást okozhat, egy ismeretlen operációs rendszer környezetében.

3.1.3. Safe Exam Browser

A szoftvercsomag eredeti célja szerint lehetővé teszi, hogy tanulók/hallgatók nem ismert eszközökön (pl.: saját laptop, tablet), kontrollált körülmények között tehessenek iskolai/egyetemi számonkéréseket. [11]

Bár alapvetően egyáltalán nem internet terminál szoftverről beszélhetünk, mégis, kutatásom során annyi párhuzamot találtam ennek a szoftvernek a belső felépítése, és itt a dolgozatban megfogalmazott megoldandó problémák között, hogy érdemesnek találtam bevonni az elemzésbe.



3.2. ábra: Safe Exam Browser komponens diagram

Működését tekintve, ez a rendszer is egy kioszk alkalmazás. Többnyire ugyanazokat mondhatjuk el, mint amiket a korábbi rendszereknél is megállapítottunk. Korlátozza a hozzáférést az operációs rendszer kezelőfelületeihez, felügyeli és menedzseli a futó/előtérbe kerülő alkalmazásokat. Korlátozza a kritikus billentyűkombinációkat, kontextus menüket. A beépített böngészője szűri a megnyitható weboldalakat, és korlátozza a felhasználó navigálási lehetőségeit, amik egy hagyományos böngészőben rendelkezésére állnak. [11]

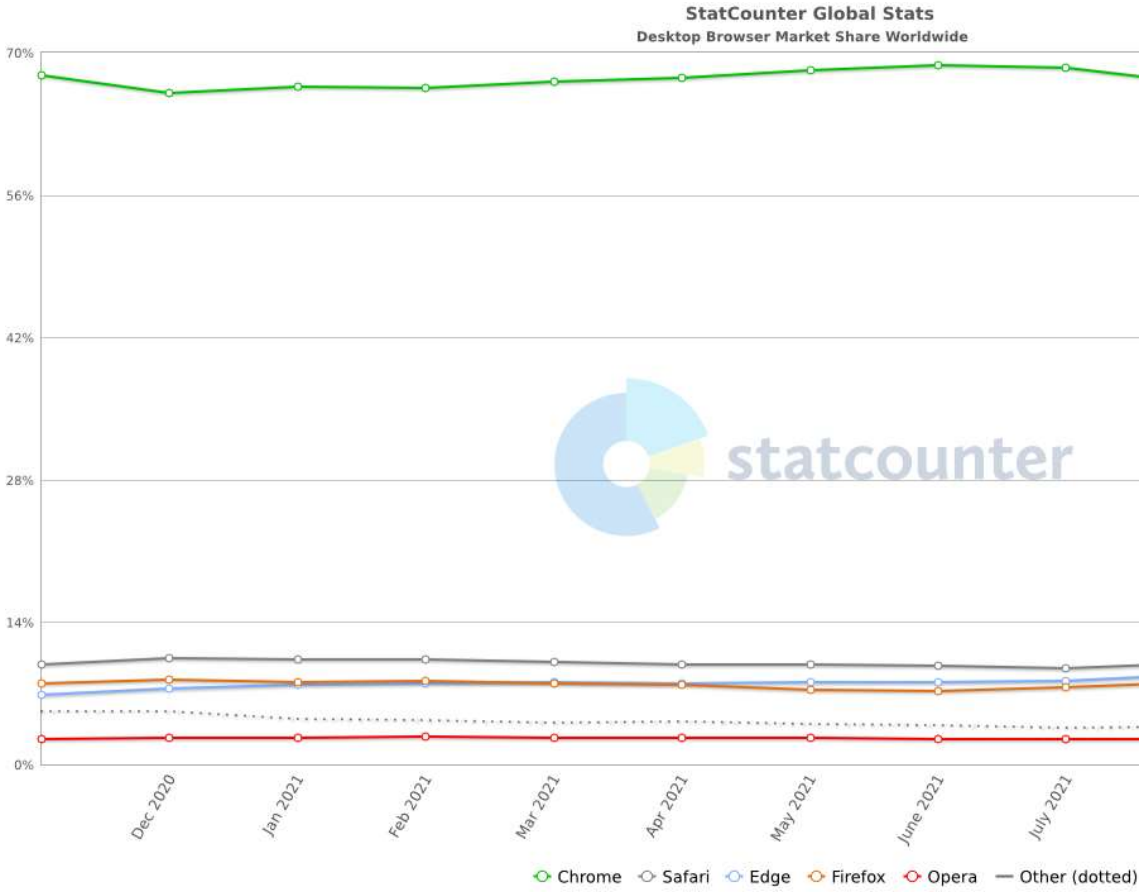
Természetesen internet terminál szoftvernek teljesen alkalmatlan, nem erre lett tervezve, de ettől függetlenül, a fentebb felsorolt funkciók mindegyike olyan, amire szükségünk van egy működő internet terminál szoftverben.

Tekintettel arra, hogy nyílt forráskódú szoftver, és az asztali számítógépekre szánt kiadása (SEB for Windows) C#-ban íródott, így fejlesztésem során egy nagyszerű ihletként, illetve referenciaként fog szolgálni.

3.2. Kioszk böngésző megvalósítás választása

A szakdolgozat kiválasztásakor az eredeti elgondolásom az volt, hogy az internet terminál szoftver webböngésző részét egy hagyományos, önálló webböngésző program futtatásával fogom elérni, az általam fejlesztett alkalmazás – a részleteket most hanyagolva – csak megnyitná, illetve bezárná azt. A Safe Exam Browser elemzése közben viszont felkeltette az érdeklődésem az általuk használt megoldás, a CefSharp könyvtár, mely potenciális alternatív megoldásként szolgálhat, ezért a dolgozat ezen részében összehasonlítanám a két megközelítést.

3.2.1. Chromium böngésző



3.3. ábra: Böngésző használati statisztika

Ahogy a grafikonból is látható, napjainkban népszerűségben fölényesen vezet a Google Chrome böngészője, így kézenfekvő választás lett volna egy Chromiumon alapuló önálló böngésző. Mivel népszerű, így a végfelhasználók jelentős része ismerősként tekintene rá, szép, letisztult felhasználói felülettel rendelkezik, így ennek az elkészítésével nem kellett volna a projekt során foglalkozni.

A felhasználók használhatták volna a rendelkezésre álló bejelentkezési lehetőségeket, így munkamenetük szinkronizálódhatódott volna eszközeik között, illetve élvezhették volna a bővítmények nyújtotta lehetőségeket. Amennyiben az internet terminál fizetősként lett volna használva, abban az esetben rendelkezésre álló egyenleg esetén a terminál program elindította volna a böngészőt, majd az egyenleg lejártával/a használat végeztével bezárta volna azt, törölte volna a felhasználó munkamenetét, visszaállított volna minden beállítást az alapállapotba, várva a következő felhasználót.

Ennek a megoldásnak az esetén a terminál a terminál programban fel kell állítani egy listát, minden előre láthatóan problémás programablakról, amit a felhasználó elérhet a böngészőből, és indokolt esetben kell zárnunk őket. Ezen felül korlátoznunk kell a böngésző beállításaihoz, konfigurációs oldalaihoz, illetve az URL sávból a helyi fájlrendszerhez való hozzáférést.

3.2.2. CefSharp

A CefSharp egy .NET könyvtár, mely többek között lehetővé teszi, hogy WPF alkalmazásokba beágyazzunk egy böngésző felületet, a Chromium Embedded Framework (CEF) keretrendszerre építkezve. [12]

Ennek a megoldásnak mentén tovább haladva jelentősen nagyobb flexibilitást érhetünk el, hiszen itt igényeimre alakíthatom a program működését, míg az előző esetben egy már kész böngésző nem kívánatos funkcióit igyekeztem elérhetetlenné tenni.

3.4. ábra: Cefsharp példa WPF böngésző implementáció

3.3. Összegzés

A vizsgált kioszk szoftverek közül egyértelműen a Provisio Sitekiosk nevű megoldása közelíti meg legjobban az általam elvártakat, azonban az elemzés során felmerült több olyan hiányosság, ami alkalmatlanná teszi, hogy potenciális megoldásként tekinthessek rá. Ezek közül a fontosabbak a megfelelő érme- és bankjegyelfogadó támogatás, a fiók és egyszerhasználatos kód kezelés hiánya, illetve az üzleti modelljük azok, amik kiemelkednek a többi közül.

A kontrollált böngésző környezet létrehozásáért vizsgált két megoldást áttekintve, egyértelműen a második, a CefSharp könyvtárat használó bizonyul jobbnak számomra.

Mindamellett, hogy a Google Chrome mellékelése egy potenciális helyreállító képfájlba jogi kérdéseket vethet fel (így a nyílt forráskódú Chromiumnál kellene, hogy maradjunk), véleményem szerint túl sok a lehetségesen előforduló hibák/mulasztások száma.

Még ha feltételezném is, hogy tudnám azonosítani az összes lehetséges módot arra, ahogy egy felhasználó hozzáférhet nem kívánt ablakokhoz/beállításokhoz a böngészőből, akkor se lehetünk biztosak abban, hogy egy későbbi böngészőfrissítés nem ad hozzá egy olyan új funkciót, amit kezelnem kellene.

4. SPECIFIKÁCIÓ

Az előző bekezdésekben elvégeztem a hasonló rendszerek áttekintését. Mivel maradéktalanul egyik sem teljesítette a projekttel szemben támasztott elvárásokat, ezért az alábbiakban összeállítom az elkészítendő rendszer specifikációját.

- A programcsomag tegye lehetővé, hogy egy nyilvános használatra kihelyezett számítógép internet terminálként legyen használható
- A számítógép lehetőségekhez mérten legyen védett a rosszindulatú felhasználóktól. Elsősorban ne lehessen:
    - Rendszerbeállításokat módosítani
    - Bezárni az internet terminál alkalmazást
    - Számítógép fájlrendszeréhez hozzáférni
    - Nem kívánatos alkalmazásokat megnyitni
    - Kritikus billentyűkombinációkat használni (ALT+F4, CTRL+ALT+DEL stb.)
- A terminálnál az üzemeltetői beavatkozások számát minimalizálni kell, ami csak lehet, az legyen automatizálva. Amennyiben a böngésző/terminál szoftver nem válaszol, nem fut, abban az esetben egy szoftvernek gondoskodnia kell ezek újraindításáról.
- Felügyelni kell a futó alkalmazásokat. Mikor egy nem engedélyezett alkalmazás beállítástól függően fut, előtérbe, vagy fókuszba kerül, a programnak azt le kell tudni állítani.
- Legyen lehetőség fizetősen, percalapon működtetni a terminált. Ennek preferált módja érmeelfogadókkal való működés. Ezen felül legyen lehetőség előre generált, promóciós jellegű ajándék kódokkal való egyenleg hozzáadásra.
- Felhasználóknak legyen lehetősége egyenleg elmentésére felhasználói fiók létrehozásával. Később a felhasználó ebbe bejelentkezve, akár egy másik azonos rendszerű internet terminálon tudja felhasználni ezt.
- Legyen lehetőség automatikus frissítések telepítésére a szoftvercsomaghoz
- Lehetőségekhez mérten legyen automatizálható egy internet terminál szoftveres telepítése. Ideális megoldás egy helyreállító képfájl, vagy egy telepítő csomag.
- Szükség van egy központi menedzser alkalmazásra, melyen kezelhetjük a terminálokat. Az üzemeltető lássa, a bevételi adatokat, hogy jelenleg mely gépek vannak bekapcsolva, utolsó IP címeket, tudjon új promóciós kódokat generálni, illetve módosítani egy adott terminál beállításait.
- Ennek megvalósításához szükség van egy központi szerverre, melyhez kapcsolódnak a terminálok, és az admin alkalmazás is. Ugyanez a szerver felel a felhasználói fiókok kezeléséért is.

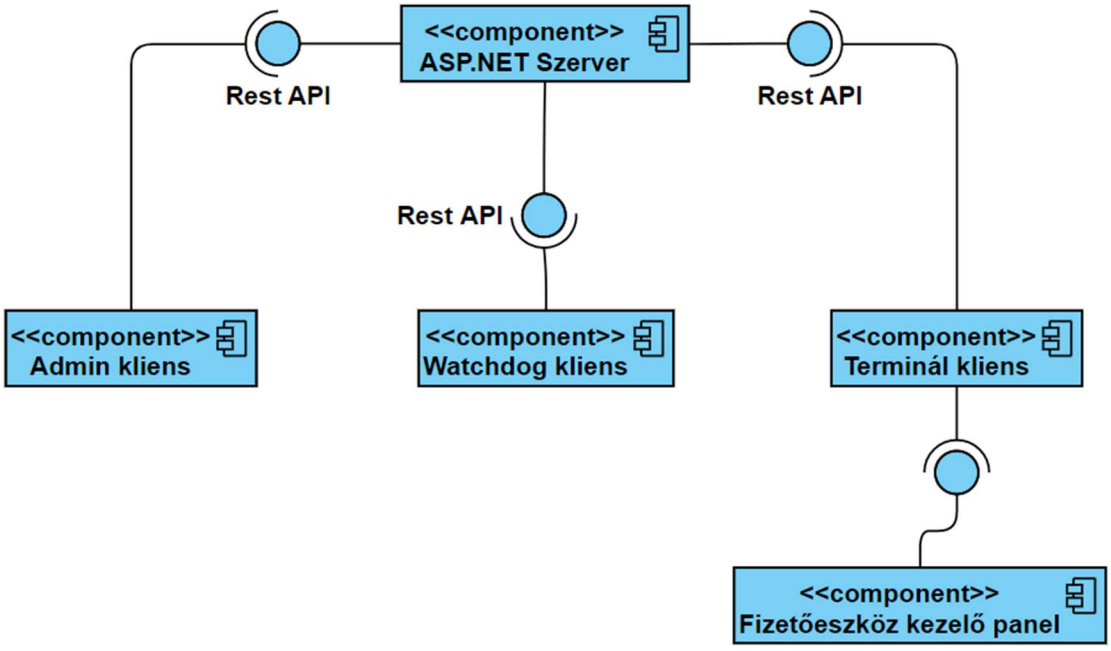
- A terminálóknak a szerver elérhetetlensége esetén is működniük kell, az egyértelműen szerver kapcsolatot igénylő funkciók kivételével (pl.: felhasználókezelés)
- Legyen lehetőség alapvető beállítások üzemeltető általi módosítására terminálonként. (Pl.: árazás, feketelistás tiltott oldalak, fehér listás engedélyezett oldalak szerkesztése, engedélyezett alkalmazások szerkesztése, stb.)
- A konfigurációs fájl legyen eltárolva lokálisan, viszont, ha a szerveren az adott géphez egy frissebb beállítás fájl van hozzárendelve, akkor a terminál bekapcsolásakor az kerüljön letöltésre.

5. TERVEZÉS

A specifikációs szegmensben támasztott elvárások alapján összeállítottam a teljes infrastruktúra megalkotásához szükséges részmegoldások listáját, melyet az áttekinthetőség kedvéért alábbiakban felsorolnék, rövid magyarázattal kiegészítve.

- **Fő terminál szoftver**: A megvalósítás alapjául szolgáló grafikus alkalmazás, mely lehetővé teszi a számítógép internet terminálként való használatát.
- **Működésfigyelő (watchdog) alkalmazás**: Menedzseli a fő terminál alkalmazás (továbbiakban kliens) működését.
- **Terminál szerver**: Központi szerver, diagnosztikai és felhasználói adatokat közvetít a kliensek felé.
- **Adminisztrátori alkalmazás**: A szerver által szolgáltatott adatok menedzselésére szolgál, felügyeli a terminálok működését, konfigurációs módosításokat tesz lehetővé.
- **Fizetőeszköz kezelő interfész**: Lehetővé teszi, különféle fizetőeszközök használatát a fő terminál alkalmazással.

A fentebbi komponensek együttműködését az alábbi komponens diagram hivatott ábrázolni.



5.1. ábra: Terminál rendszer komponens diagram

5.1. Alkalmazás környezeti terv

A hasonló rendszerek elemzése fejezetben taglaltokat is figyelembe véve, az internet terminál szoftver megvalósításához Microsoft Windows 10 operációs rendszert találtam ideálisnak. Az ott említetteken előnyökön felül, az operációs rendszer népszerűségéből és elterjedtségéből adódóan, egy-egy részproblémára sokkal nagyobb eséllyel tudunk már kész, létező megoldást találni online, illetve szakirodalmakban.

Az internet terminál hatékony, megbízható működéséhez elengedhetetlen, hogy gyökeres változtatásokat eszközöljünk az operációs rendszer alap beállításaihoz, megjelenéséhez képest.

Szükségessé válik:

- A kritikus billentyűkombinációk letiltása (Pl.: ALT +F4, CTRL+ALT+DEL stb.)
- Operációs rendszer néhány alapvető felhasználói felületéhez való hozzáférés tiltása, mint például a start menü, tálca, asztal stb.
- Programok telepítésének és törlésének a tiltása
- Külső alkalmazások futtatásának tiltása (Az internet terminál rendszerben potenciálisan engedélyezett, felhasználó által indítható alkalmazások a terminál kliens alkalmazásból lesznek indíthatóak, felügyelt módon)
- Operációs rendszer előugró értesítéseinek a tiltása
- Számítógép fájlrendszeréhez való hozzáférés korlátozása

5.2. Internet terminál kliens alkalmazás

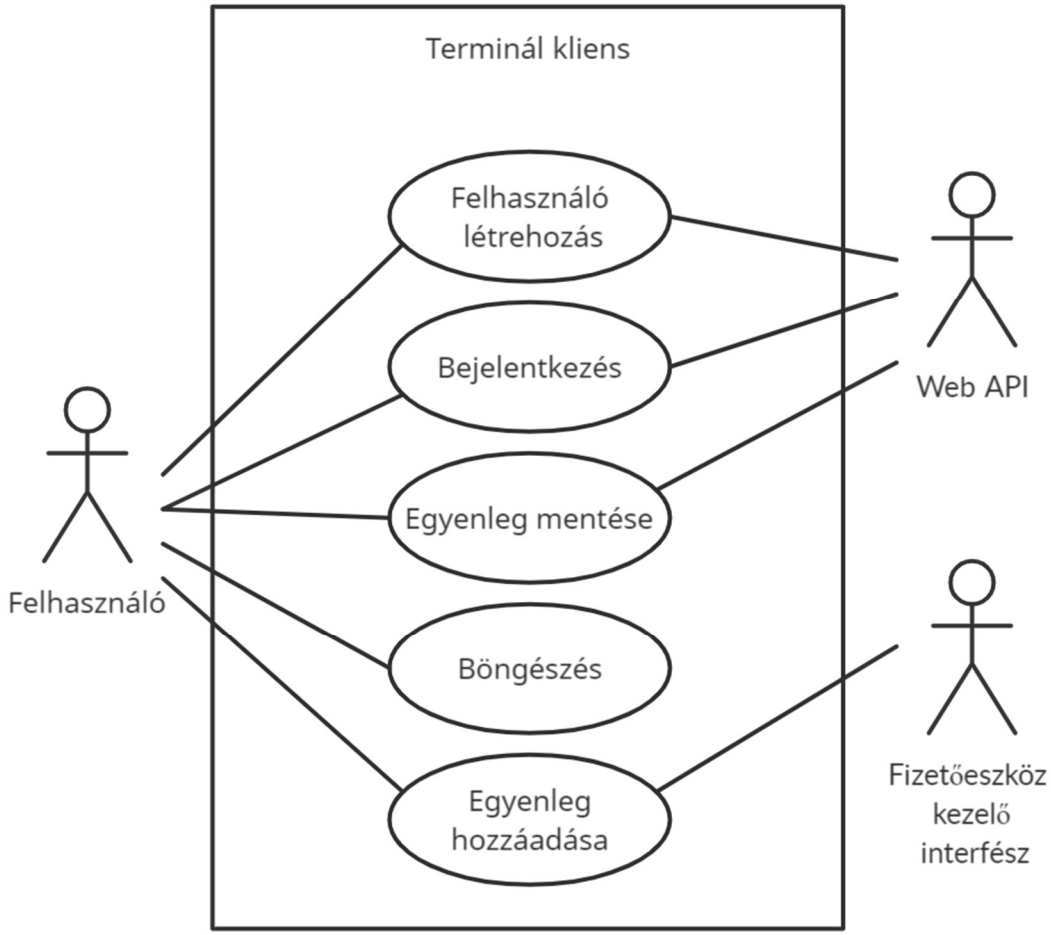
A szoftvercsomag ezen komponense fogja ellátni, egy internet terminálától elvárt, korábban taglalt funkciók jelentős részét, miközben építkezik a többi komponens által biztosított szolgáltatásokra.

Az alkalmazás egy .NET Core WPF (Windows Presentation Foundation) alkalmazás, mely a felhasználók számára a fő interakciós felület lesz a rendszerrel. Egy felhasználó átlagos munkamenete a következőből lépésekből fog állni:

Az internet terminál alapállapotban, egy kezdőképernyőt fog mutatni, ahol a felhasználók láthatják a rendelkezésre álló lehetőségeket, alkalmazásokat. Fizetés nélkül, az egyik opciót kiválasztva, a kliens egy információs ablakot jelenít meg, amely tartalmazza a díjazáshoz kapcsolódó információkat. (elfogadott fizetőezközök, percdíj stb.). Amennyiben egyenleg kerül hozzáadásra, lehetősége nyílik a korábban ismertetett funkciók használatára (elsősorban webböngészés). használat befejeztével, illetve az egyenleg lejártával a felhasználó munkamenete véget ér, a használat közben eltárolt

adatok törlésre kerülnek, a terminál visszatér a kezdőképernyőre, készen állva a következő használatra.

Rendelkezésre álló egyenleggel, a felhasználónak használat közben bármikor lehetősége van bejelentkezni/fiókot létrehozni a szerverünkön, ez után pedig dönthet úgy, hogy eltárolja a még hátralévő egyenleget, későbbi használatra. Mikor a rendelkezésre álló egyenleg hamarosan el fog fogyni, a felhasználó értesítve lesz erről, elkerülve, hogy váratlanul véget érjen a munkamenete.



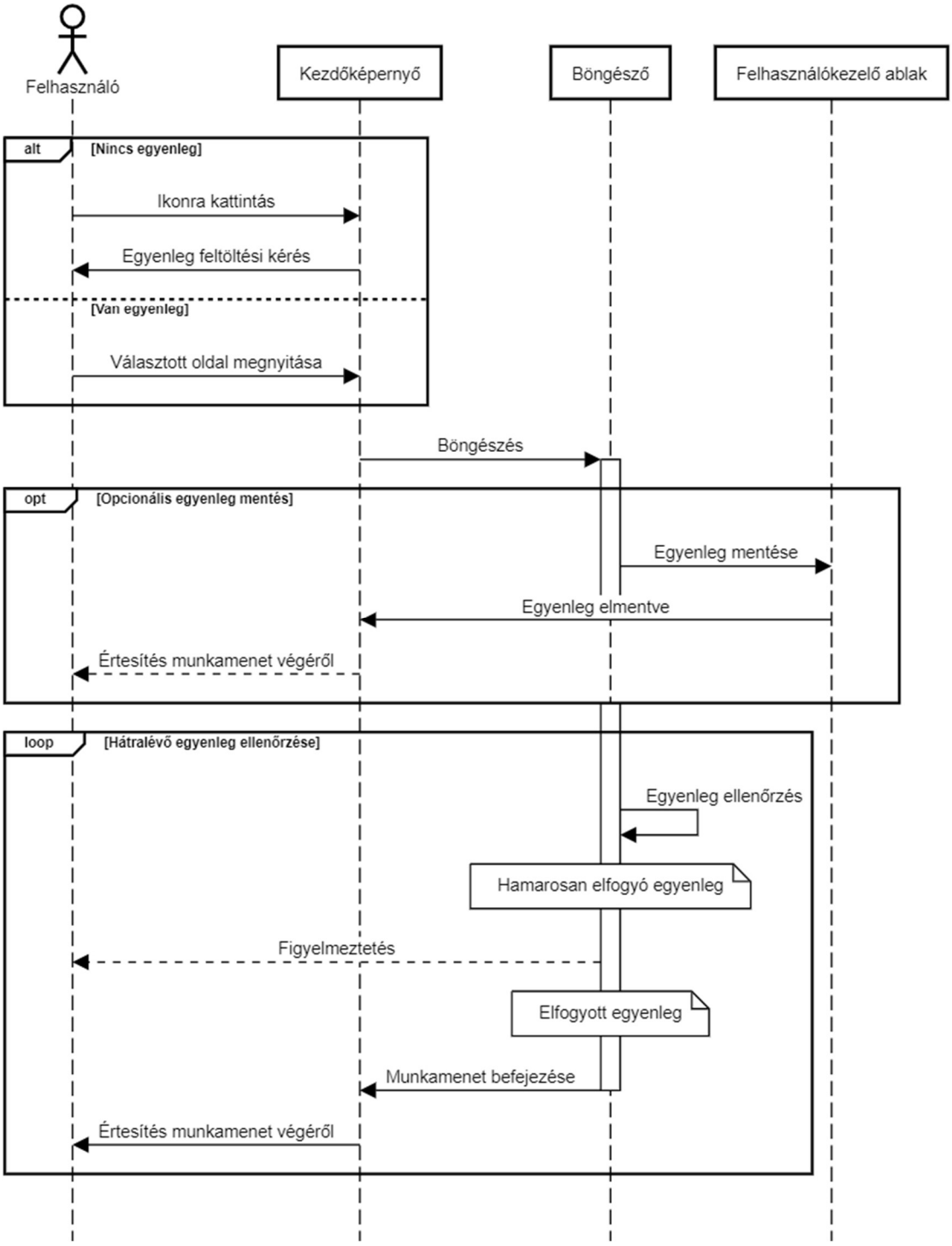
5.2. ábra: Felhasználói munkamenet használati eset diagramja

Az alkalmazás felhasználói felülete a következő főbb ablakokból fog állni:

- **Kezdőképernyő**: Egy letisztult felhasználói felület, amely ikonokat tartalmaz a böngészőhöz, az opcionálisan elérhető alkalmazásokhoz, illetve néhány népszerűbb weboldalhoz, a gyors elérés érdekében
- **Böngésző**: A korábbi elemzésben említettek alapján, az internet terminál böngészőjét a CefSharp könyvtár használatával fogom megoldani. Megjelenés szempontjából rendelkezni fog egy általános böngészőtől megszokott és elvárt vizuális elemekkel (Vissza, előre, frissítés stb. gombok, navigációs sáv), egy munkamenet befejezés/kijelentkezés gombbal, és egy felhasználó kezelő ablak megnyitására való hivatkozással. A böngésző szűrni fogja a megjeleníthető weboldalakat a terminál konfigurációs fájljában megadottak alapján, illetve lesz lehetőség bizonyos eltérő weboldalakhoz egyedi díjszabást rendelni. (pl.: egy üres Google kereső eshet az ingyenes kategóriába, így az egyenleg nem fog fogyni, ameddig nincs semmilyen érdemi tartalom megjelenítve)
- **Bejelentkezési felület**: A felhasználónak itt nyílik lehetőségük felhasználói fiókot létrehozni, bejelentkezni, illetve eltárolni az egyenlegüket.

Minden ablakra általánosságban jellemző, hogy lesz egy állandóan látható szegmens, mely a hátralévő egyenleget fogja kijelezni, perc, forint, és folyamatábra formájában.

Egy felhasználó átlagos munkamenetét a következő szekvencia diagram hivatott ábrázolni:



5.3. ábra: Felhasználói munkamenet szekvencia diagram

5.3. Működésfelügyelő alkalmazás

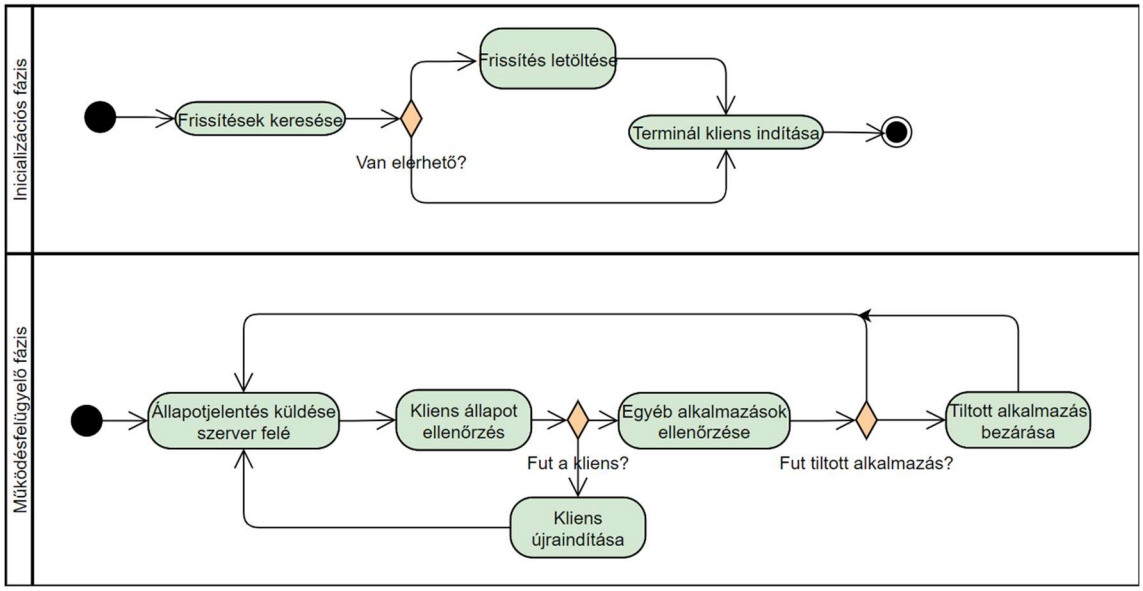
A számítógép indulásakor, ez a szintén .NET Core-ban íródott, grafikus felülettel nem felétlenül rendelkező alkalmazás indulna el. Fő feladata, az alap terminál szoftver (kliens)

működésének menedzselése. Az olyan alkalmazásokat, melyek elsődleges feladata egy másik szoftver működésének felügyelete, illetve hibák esetén helyreállítása, angol kifejezéssel szokás watchdog alkalmazásnak nevezni.

Induláskor az alkalmazás csatlakozik a szerverhez, és ellenőrzi, hogy az adott számítógéphez áll-e rendelkezésre újabb verziójú szoftver a terminál alkalmazáshoz. Amennyiben igen, ebben az esetben a program letölti az újabb verzió binárisait tartalmazó archívumot, és ennek a tartalmával felülírja a régi alkalmazást. Ezt követően elindítja a terminál szoftvert, ami hatására a gép készen áll a használatra. Ellenkező esetben, ha nem áll rendelkezésre frissítés, egyszerűen futtatja a klienst.

Ezzel lezárul a program inicializálási szakasza, innentől pedig periodikusan ellenőrzi az alkalmazás működését. Hibás működés esetén ennek az alkalmazásnak a feladata annak az elhárítása. Amennyiben az alkalmazás nem fut, háttérbe kerül, nem válaszol, abban az esetben bezárja, majd megkísérli újraindítani azt. Ezen felül az alkalmazás felügyeli a számítógépen futó többi programot is. A terminál konfigurációjában tudunk tiltott folyamat neveket felvenni. Ha egyezést talál a szoftver, abban az esetben bezárhatja, háttérbe kényszerítheti az érintett alkalmazásokat beállítástól függően.

Amennyiben minden rendeltetésszerűen működik, ez az alkalmazás szolgáltatja a diagnosztikai, üzemeltetési adatokat a szerver felé. Ezek közül a fontosabbak a gép IP címe, online állapota stb.



5.4. ábra: Program indulásának aktivitás diagramja

5.4. Terminál szerver

A terminálok működésük során állandó kapcsolatban lesznek egy központi szerverrel. Ez a szerver fogja biztosítani a felhasználókezelést, üzemeltetési és diagnosztikai adatok gyűjtését, illetve a konfigurációs fájlok terminál felé történő biztosítását.

A szerver egy ASP.NET Core web API alkalmazás, melyhez a kliensek, illetve a központi terminál menedzselő szoftver REST API kérésekkel fognak csatlakozni.

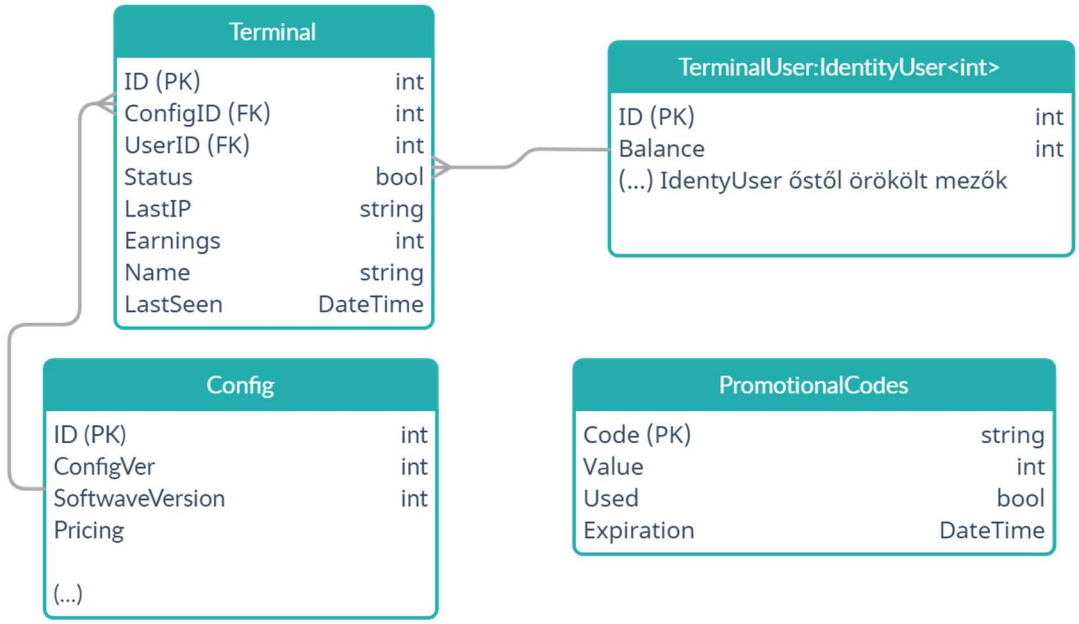
Az adatok egy adatbázisban lesznek tárolva, melyhez kódból az Entity Framework Core objektum-relációs leképező keretrendszer segítségével fogok hozzáférni.

A kliensek hitelesítését JSON Web Token standardnak megfelelően fogom biztosítani. Ennek megvalósításához a Microsoft.AspNetCore.Authentication.JwtBearer, és a Microsoft.AspNetCore.Identity.EntityFrameworkCore nevű NuGet csomagok nyújtotta megoldásokat használom fel.

A szerver elsődleges funkciói a következők:

- **Felhasználókezelés**: A terminál felhasználóinak lehetősége van fiókokat létrehozni, és ezekbe a hátralévő egyenlegük elmentésére.
- **Üzemeltetési adatok gyűjtése, és szolgáltatása**: A terminálok periodikusan aktivitást jelző kéréseket küldenek a szerver felé, ezzel jelezve az online állapotukat. Ezen felül a gépek bevétel adatokat, is szolgáltatnak a szerver felé. A menedzser alkalmazás a szerverhez csatlakozva összegzett formában hozzáférhet ezekhez az adatokhoz
- **Konfigurációs adatok szolgáltatása**: A terminálok indításukkor lekést küldenek a szervernek, melyre válaszként megkapják, az adott terminálhoz rendelt konfigurációs fájlt szerializált formában. Amennyiben ez újabb, mint a lokális változat, abban az esetben ez kerül eltárolásra. Ez alapvető beállításokat tartalmaz a terminál működésére vonatkozóan (pl.: árazás, verziószám stb.).

Ezek alapján elkészítettem a szerver adatbázistervét, mellőzve a felhasznált könyvtárak által automatikusan generált táblákat:

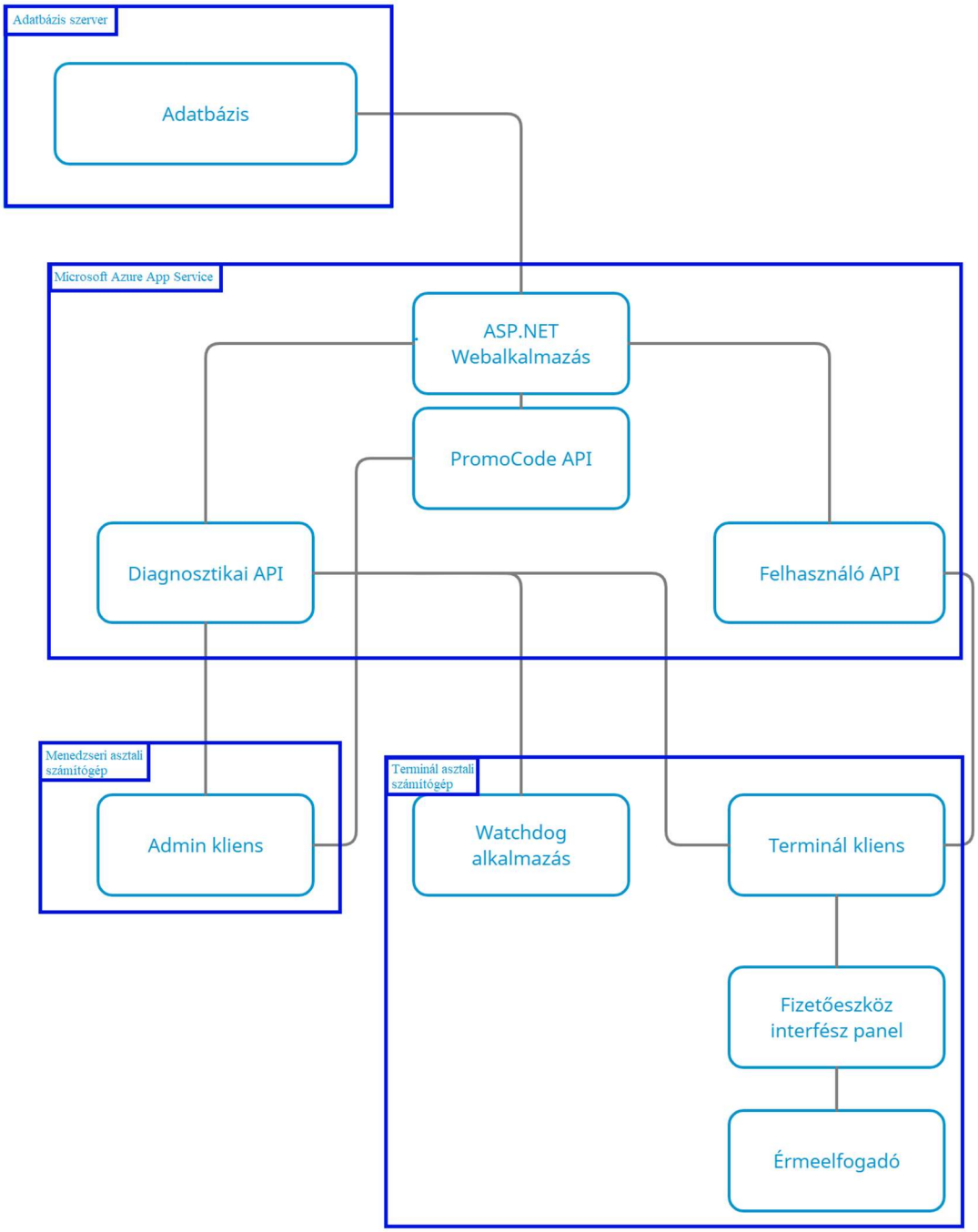


5.5. ábra: A szerverhez kapcsolódó adatbázis-diagram

Az adatbázisterv-diagram néhány kevésbé magától értetődő részét az alábbiakban részletezném:

- **Terminal tábla**:
  - Status: A gép jelenlegi állapota. Igaz érték esetén a gép jelenleg online, ellenkező esetben offline
  - LastIP: Az kliens IP címe, mikor legutoljára online volt
  - Earnings: Az utolsó nullázás óta összegyűjtött bevétel
  - Name: Egy tetszőlegesen hozzárendelhető név a terminálhoz, amely a menedzser alkalmazásban jelenik meg.
- **Config tábla:**
  - ConfigVer: Az adott beállításjegyzék verziószáma. Amennyiben nagyobb, mint a lokálisan tárolt, abban az esetben ez kerül letöltésre.
  - SoftwareVersion: Az internet terminál indításakor az indítóprogram összeveti a telepített kliens verziószámát az ebben az értékben találhatóéval. Amennyiben ez nagyobb, abban az esetben megkísérli a kliens legfrissebb verziójának letöltését és telepítését.
  - Ezeken felül, ebben a táblában tárolódik el minden gépspecifikus beállítás (pl.: pricing – árazás). Ez a tábla még bővülni fog, az implementáció során felmerülő igényekkel.
- **TerminalUser**:
  - Felhasználók tárolására, mely örökli az ASP.NET Core Identity csomag IdentityUser mezőit.

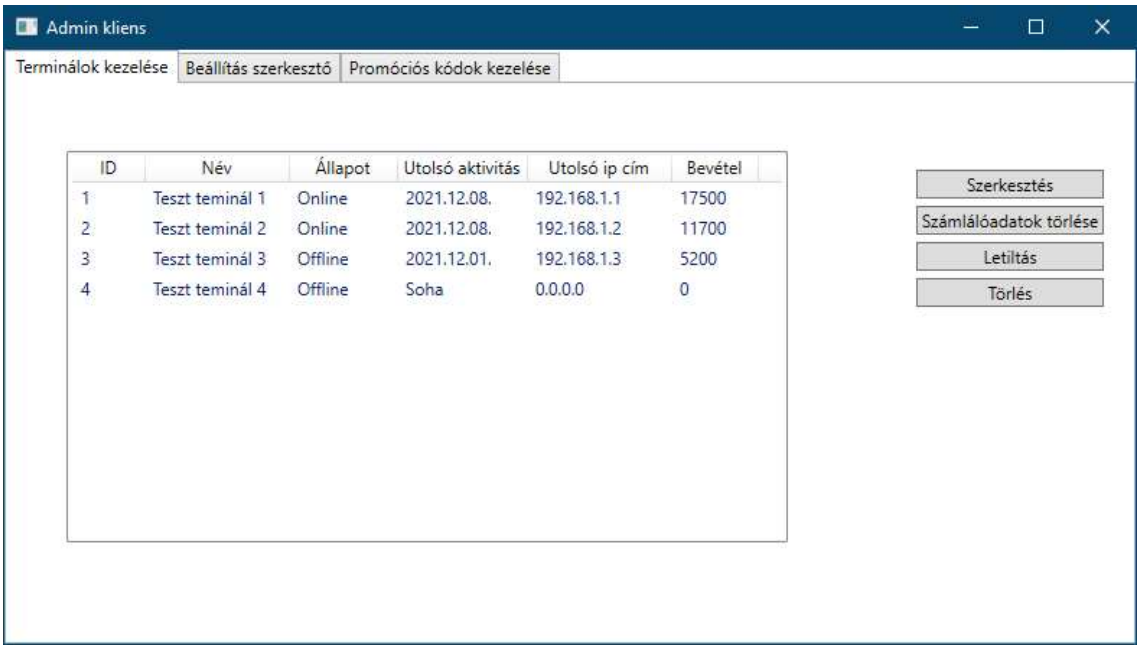
    - ID: elsődleges kulcs, a táblák közötti egységesség kedvéért integer típusú.
    - Balance: A regisztrált felhasználó által elmentett hátralevő egyenleg.
- **PromotionalCodes tábla**:
    - Előre legenerált véletlen karakterláncból álló kódok, melyek beütés után a Value mezőben található értéket jóváírják a felhasználó egyenlegéhez.
    - Code: Egyedi, meghatározatlan hosszúságú karakterlánc, mely elsődleges kulcsként is funkcionál.
    - Used: Amennyiben igaz az értéke, abban az esetben a kód már el lett használva, így érvénytelen.
    - Expiration: A létrehozott kódok ebben a mezőben meghatározott dátumig maradnak érvényesek, ez után nem lesz lehetőség felhasználni őket.



5.6. ábra: Az infrastruktúra architekturális terve

5.5. Adminisztrátori alkalmazás

Egy meglehetősen egyszerű .NET Core WPF alkalmazás, melyben menedzselhetjük a jelenleg online internet termináljainkat.

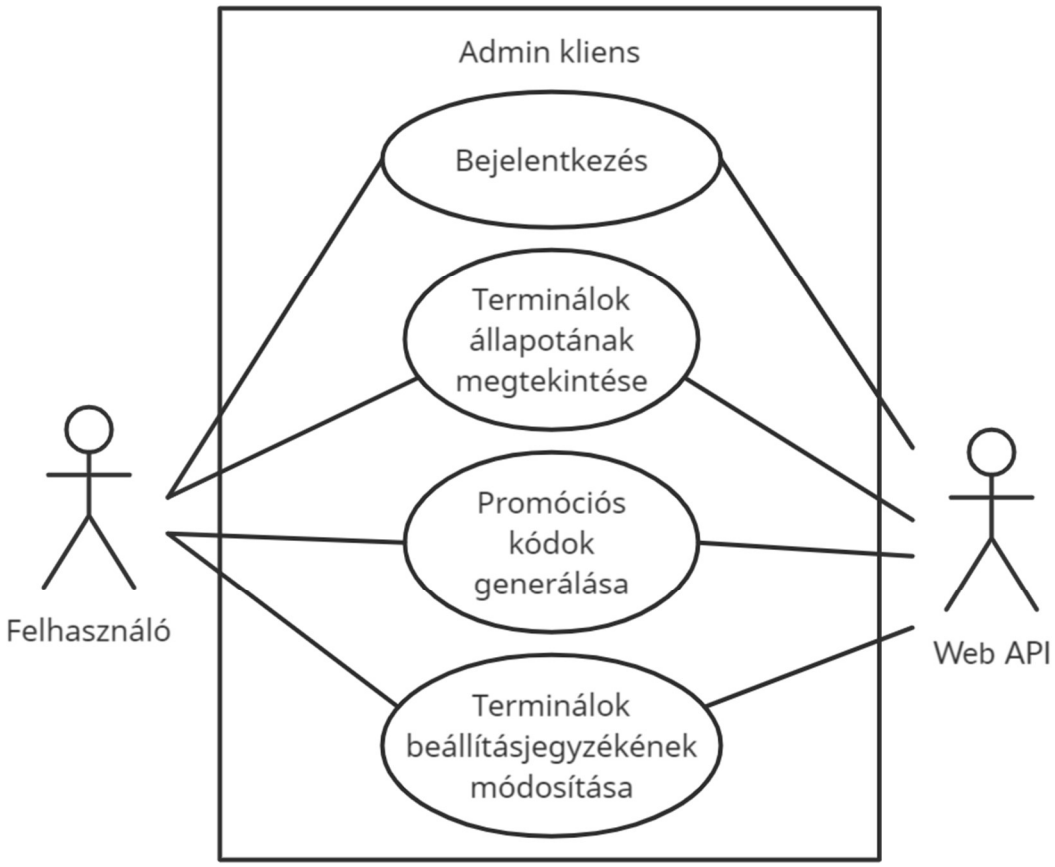


5.7. ábra: Az admin alkalmazás látványterve

Az alkalmazás szintén a korábban taglalt szerveralkalmazástól nyeri az adatait, JWT hitelesítést követően. Lista nézetben fogjuk tudni áttekinteni a szerverre már legalább egyszer bejelentkezett terminálokat, illetve megtekinteni a hozzájuk tartozó fontosabb információkat (Utoljára online, bevétel, IP cím stb.)

Ezen felül ebben az alkalmazásban lesz lehetőség grafikus felületen szerkeszteni egy terminálhoz tartozó konfigurációs beállításokat. Az így létrehozott beállításjegyzék feltöltődik a szerverre, amit ezután hozzárendelhetünk az általunk kiválasztott internet terminálokhoz.

Végezetül pedig, ebben az alkalmazásban nyílik lehetőség a promóciós kódok generálására is. Megadhatjuk, hogy hány karakterből álljon a kódot alkotó véletlen karakterlánc, hány kódot szeretnénk generálni egyszerre, mennyit érjenek a generált kódok, illetve meddig legyenek érvényesek. A generált kódlista exportálható lesz .CSV formátumban.



5.8. ábra: Adminisztrátori alkalmazás használati eset diagram

5.6. Fizetőeszköz kezelő interfész

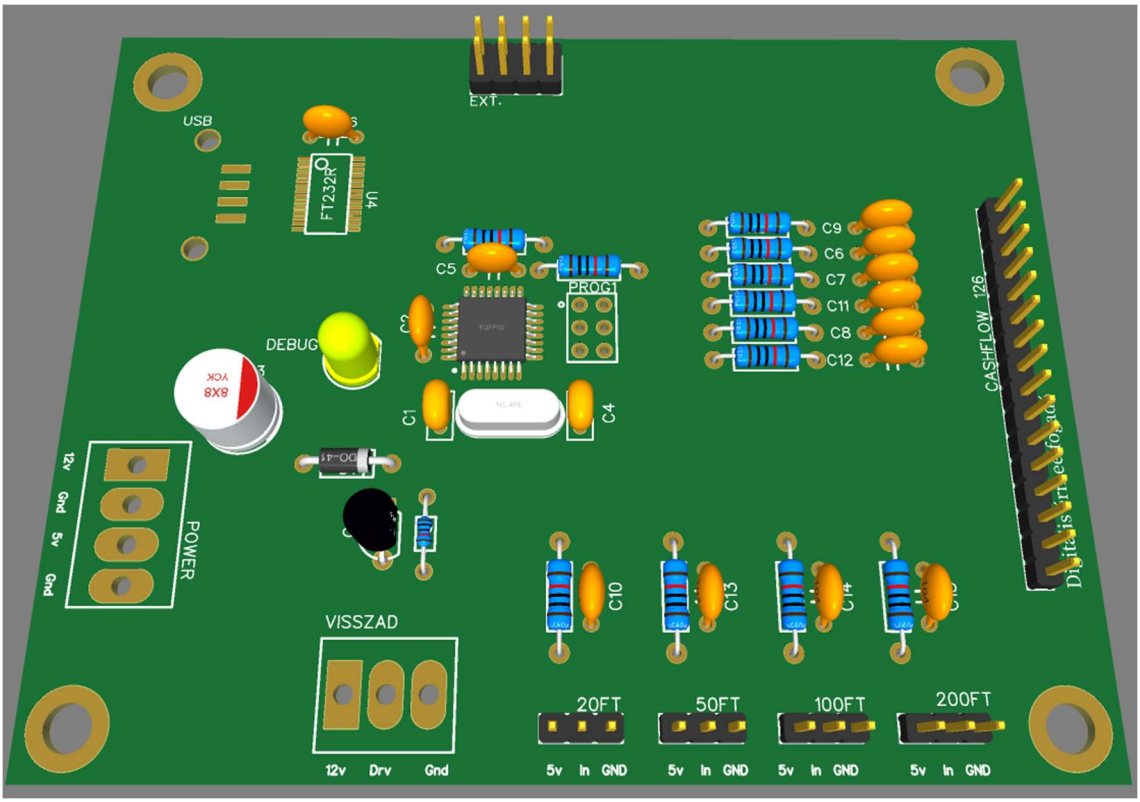
Az előző fejezetben, a Sitekiosk nevű internet terminál szoftver vizsgálatánál áttekintettem a rendelkezésre álló fizetőeszköz támogatást. Bár régebben valóban számos fizetőeszközt támogatott, ezeknek a száma frissítésekkel folyamatosan csökkent, jelentős nehézségeket okozva ezzel azoknak, akik egy olyan hardverrel építették ki a rendszerüket, aminek váratlanul megszűnt a támogatása.[5] Tekintve, hogy a kereskedelmi forgalomban kapható érme- és bankjegyelfogadók jelentős része nem alkalmaz egy egységes, gyártókon átívelő kommunikációs formát, így egyszerre számos fizetőeszköz támogatása indokolatlanul nagy feladattá válik.

Ennek a problémának a megoldására, az általam tervezett megoldásban, elhelyeznék egy mikrokontroller vezérelt áramkört a számítógép, és a különféle fizetőeszközök közé. Ez az eszköz, kommunikálna a különféle fizetőeszközökkel, majd amennyiben sikeres volt a fizetés, a kliens alkalmazás felé már csak a tranzakció eredményét, és értékét kommunikálná absztrakt módon.

Ennek a megközelítésnek az előnye, hogy egy új fizetőeszközre való átálláskor a számítógépen szoftveres oldalról nem is kell tudnunk a bekövetkezett változásról, csak az interfész panelt kell alkalmaztatni az új hardverhez.

5.6.1. Vezérlő tervezése

Az áramkör alapjául az Atmel cég ATmega328P mikrovezérlője fog szolgálni. A tervezés során referenciául az Arduino Uno fejlesztői panel kapcsolási rajzát fogom venni, tekintve, hogy ez is ezt a mikrovezérlőt használja. [13] A mikrovezérlő kódját C++-ban fogom írni, többek között az Arduino könyvtár felhasználásával.



5.9. ábra: Interfész panel látványterve

5.6.2. Kommunikáció

A mikrokontroller rendelkezik egy UART kommunikációs modullal. Az UART (Universal Asynchronous Reciever&Transmitter) egy kétirányú, aszinkron kommunikációs forma [14]. Sajnos, ez nem ideális számítógépekkel való kommunikációra, így közbe kell iktatni egy integrált áramkört, mely az USB, és az aszinkron kommunikáció közötti kapcsolat létesítésére lett kifejlesztve. A választásom korábbi tapasztalataim alapján a FTDI Chip cég FT232R integrált áramkörére esett, hiszen ez pontosan ezt a funkcionalitást nyújtja az adatlapja alapján. [15]

A mikrovezérlő és a terminál kliensalkalmazás között két fajta üzenet kerül küldésre.

A mikrokontroller üzenhet az alkalmazásnak, hogy egy adott összegű egyenleg kerüljön jóváírásra, vagy amennyiben valamilyen hiba történt, akkor elküldheti annak a leírását. Emellett a kliens alkalmazás periodikusan engedélyező üzenetet küldhet a mikrovezérlőnek. Az interfész panel, ennek az engedélyező jelnek a hatására engedélyezi a hozzá csatlakoztatott fizetőeszközök bemeneteit. Erre akkor lehet szükség, ha esetleg a terminálon megszűnik az internetkapcsolat vagy valamilyen más okból üzemképtelenné válik a kliens. Terv szerint a kliensalkalmazás minden 10. másodpercben fog küldeni egy ilyen engedélyező üzenetet a mikrokontrollernek, és amennyiben az 20 másodpercig nem észlelt ilyet (tehát legalább 2 üzenet kimaradt), letiltja a bemeneteket. Ebben az esetben, amennyiben a fizetőeszköz támogatja, a bedobott érmék visszaadásra kerülnek.

5.6.3. Támogatott fizetőeszközök

A specifikáció alapján elsősorban érmeelfogadókkal fog együttműködni az interfész.

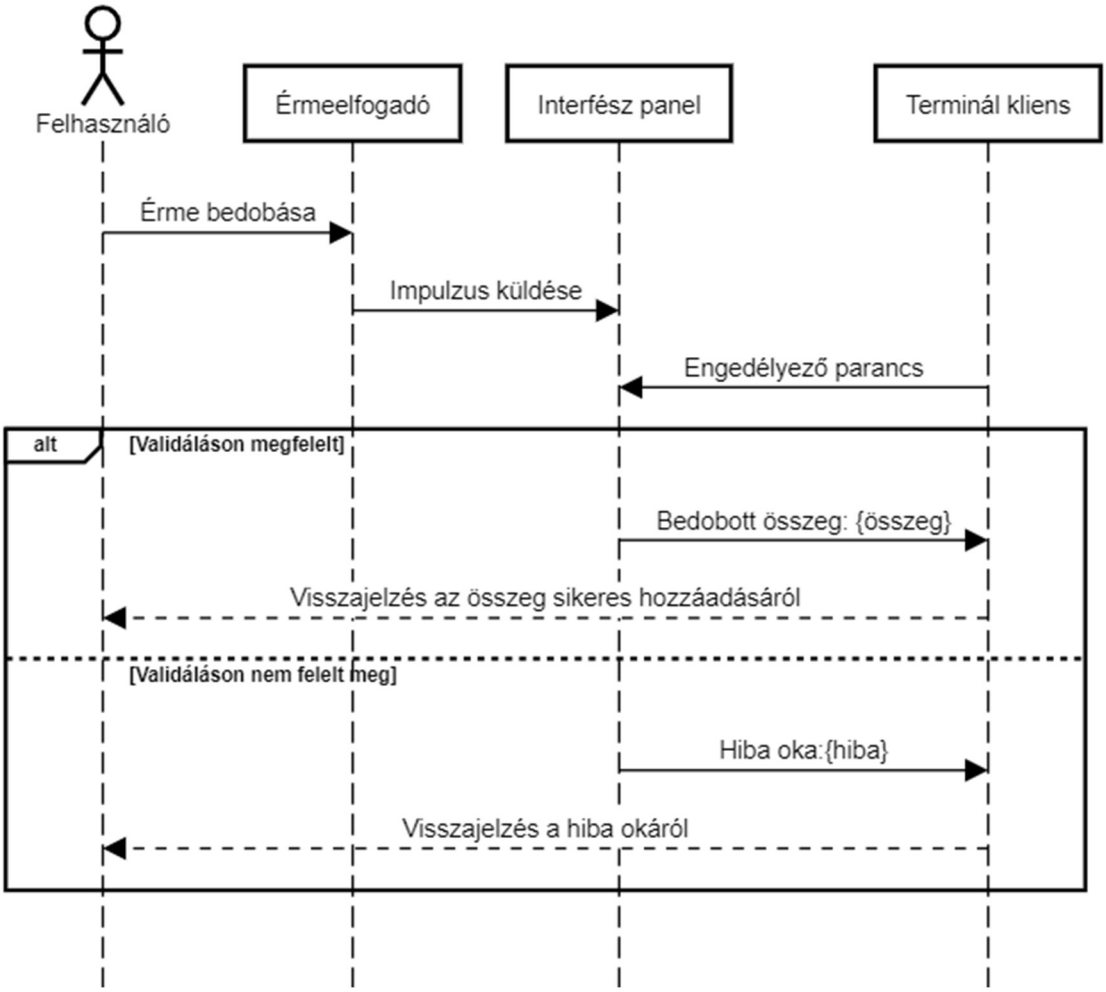
Ezeknek az eszközöknek a legegyszerűbb fajtája a passzív érmeelfogadó, mely egyszerűen az érme fizikai tulajdonságai alapján választja szét az érméket. Megfelelő érme bedobásakor mechanikus elven az érme egy olyan csatornába lesz irányítva, melynek az alján egy mikrokapcsoló található, ellenkező esetben pedig az érme abban a csatornában halad tovább, amely visszavezet a felhasználóhoz. Ennek a fajtának hátránya, hogy minden támogatott érméhez külön csatornára van szükség, ami rendkívül helyigényes.

Ennek egy modernebb alternatívája a digitális érmeosztályozó, mely a benne levő áramkörrel figyeli a leeső érme sebességét, súlyát, fémszenzorokra gyakorolt hatását, és ezek alapján ítéli meg, hogy milyen érme került bedobásra. Előnye, hogy egy csatornán többféle, a készülék által ismert érmét is bedobhatunk, és az elfogadó külön kimeneteken jelzi, hogy milyen fémpénz került detektálásra.

Dolgozatom során a számomra rendelkezésre álló internet terminálokban a Mars Electronics cég Cashflow 126 modellű érmeelfogadója áll rendelkezésre, így az interfész panelt ehhez fogom tervezni. [15]

Emellett lehetőség lesz még a Magyarországon forgalomban lévő fémpénzek közül 4 elfogadására hagyományos, mechanikus elven működő passzív érmeelfogadókkal (20, 50, 100, 200 Ft). A mikrovezérlő szabadon maradt bemenetei ki lesznek vezetve egy bővítésre szánt csatlakozóra, így későbbiekben, egy új, eltérő kimenetekkel rendelkező fizetőeszköz hozzáadása is megoldható, az interfész panel újra tervezésé nélkül is.

A fizetőeszköz kezelő modul működését az alábbi szekvencia diagram mutatja be:



5.10. ábra: Érmeelfogadó modul szekvencia diagram

6. FEJLESZTÉS

Az alábbi fejezetben taglalnám, a projekt főbb elemeinek megvalósításának lényegesebb lépéseit. Gondolataimat a rendszer moduljainak mentén igyekeztem pontokba szedni, kitérve az egy adott területet érintő, megvalósítás során felmerülő problémákra, nehézségekre.

A fizetőeszköz kezelő panel mikrovezérlőjére írt programon kívül az összesről elmondható, hogy Visual Studio 2022-ben kerültek fejlesztésre, a .NET 6-os keretrendszer verziót célozva.

6.1. Szerver webalkalmazás

A terminálok központi szerveréül szolgáló alkalmazást egy ASP.NET Core Web alkalmazásként valósítottam meg. A kliensek és a szerver közötti kommunikációt JWT-vel (JSON Web Token) ellátott RESTful API hívások szolgáltatják.

A szerveralkalmazás útvonalválasztását vezérlők (idegen szóval Controllerek) mentén konfiguráltam. Az webes API végpontjai 4 többnyire elkülönülő üzleti logika mentén lettek szétcsoportosítva különböző kontrollerekbe.

Az alkalmazás végpontjainak dokumentálására a Swashbuckle.AspNetCore NuGet csomag által nyújtott Swagger kiegészítőt használtam. A szerveralkalmazást Debug módban futtatva, a **/swagger/** végponton elérhető Swagger UI segítségével áttekinthetjük a rendelkezésre álló végpontok listáját, a végpontok konkrét URL címét és a hozzá tartozó REST API hívás típusát is.

Post, illetve **Put** kérések esetén szintén itt tekinthetjük át, a kérés törzsét alkotó JSON objektum elvárt formátumát is, ami eredetileg a Models projektben került definiálásra.

A következőkben áttekinteném az alkalmazás végpontjait, az itt említett csoportosítás szerint. A standard Létrehozás/Lekérés/Frissítés/Törlés feladatokat (továbbiakban CRUD műveleteket) ellátó, illetve az egyéb, kisebb jelentőséggel bíró végpontok részletezését mellőzöm, ezeket a fentebb említett módon elérhető dokumentációban lehet áttekinteni részletesen.

6.1.1. Auth Controller

Az ebbe a csoportba tartozó végpontok felelősek a felhasználó kezeléssel, hitelesítés, illetve jogosultság kezeléssel kapcsolatos kérések kiszolgálására.

Mint ahogy fentebb már említettem, az alkalmazás JWT hitelesítést alkalmaz. Ez lényegében abból áll, hogy a szerver, egy sikeres bejelentkezés után generál egy egyedi karakterláncból, illetve lejárati dátumból álló objektumot, melyet eltárol, illetve továbbít

a bejelentkező kliens felé. Ezt a kliens is eltárolja a munkamenete során, és csatolja minden elkövetkezendő http kérés fejlécének hitelesítés szegmenséhez.

Miután a szerver ezt összeveti az általa eltárolttal, egyértelműen meg tudja állapítani, hogy a beérkező kérés melyik felhasználói munkamenettől érkezett.

A webalkalmazásban a fiókokhoz 3 lényeges jogosultsági kört definiáltam:

- **Admin**: Kiterjedt hozzáférése van a legtöbb funkcióhoz, beleértve a kliensek, illetve a felhasználók kezelését is.
- **Terminal**: Minden internet terminál kliens alkalmazás tagja ennek a körnek, a kliensek menedzseléséhez köthető funkciókhoz biztosít hozzáférést.
- **User**: Felhasználó által létrehozott fiókok tartoznak ide, minimális jogosultságokat biztosít.

A kontroller összesen 10 végponttal rendelkezik, melyből az alábbiakat részletezném:

- **HTTPPUT** /auth/login: Nem igényel hitelesítést, bejelentkezés után válaszban szolgáltatja a tokent.
- **HTTPPOST** /auth/register: Új felhasználó létrehozása. Szintén elérhető JWT nélkül is.
- **HTTPGET** /auth/setrole: Beállítja egy tetszőleges felhasználó jogosultsági szintjét a kiválasztottra.
- **HTTPPOST** /auth/setbalance: Egy kiválasztott felhasználóhoz tartozó egyenleg módosítására szolgál.

6.1.2. Client Controller

Ezen kontroller alá besorolt végpontok szolgálnak a kliensalkalmazások által automatikusan létesített műveletek kiszolgálására. Lényegesnek tartom kiemelni, hogy ebben a kontrollerben a CRUD műveletek közül nem találkozhatunk konkrét létrehozás végponttal, mivel itt ennek a szerepét a **/client/heartbeat/** végpont kiváltja.

A kontroller összesen 8 végponttal rendelkezik, melyek közül az fontosabbak az alábbiak:

- **HTTPGET** /client/heartbeat: Minden kliens periodikus kéréseket küld ennek a végpontnak, mellyel jelzi a jelenlegi online állapotát.
- **HTTPGET** /client/addbalance: A kliensalkalmazásokhoz történő egyenleg hozzáadása után hívódik meg, a számlálóállások követéséhez.
- **HTTPGET** /client/getupdateurl: A frissítő alkalmazás számára létrehozott végpont, megadja az egy adott klienshez tartozó frissítési fájl letöltési címét.

6.1.3. Codes Controller

Felelevenítésképpen, a terminál használata érmeelfogadón keresztüli egyenleg hozzáadásán kívül lehetséges még egyszer használatos kódok felhasználásával is. Ez a kontroller felel az ehhez a funkcionalitáshoz szükséges elérési pontok biztosításáért. Jellegéből adódóan itt nem találkozhatunk a teljes CRUD felhozatallal.

Itt az alábbi végpontok találhatóak meg:

- **HTTPGET** /codes/generate: Ez a végpont szolgál az egyedi kódok generálására. 3 paraméterrel rendelkezik, melyek megszabják a generálandó véletlen karakterlánc hosszát, az egyszerre generálandó kódok mennyiségét, illetve a kódok értékét.
- **HTTPGET** /codes/use: Kliensen keresztül, felhasználói interakcióra meghívott végpont, mely felhasznált státuszra állítja a paraméterként megadott kódot.
- **HTTPDELETE** /codes/deleteused: Töröl minden már elhasznált, és/vagy lejárt kódot az adatbázisból.
- **HTTPUT** /codes/delete: Törli a kérés törzsében megadott kódokat az adatbázisból

A dolgozat elején, eredeti terveim szerint, terveztem még a kliensbeállítások kezelése logika mentén létrehozni egy konfigurációért felelős csoportot is, megfelelő osztályokkal a Data, Repo, Logic, Api rétegekben is. Erre kitértem a tervezés fejezet során, viszont az implementáció során szembesültem két olyan tényezővel is, ami alapján ezt már nem találom ideális megközelítésnek.

Ezek közöl az első, hogy megállapítottam, hogy a már létező „TerminalClient” adatbázis táblám gyakorlatilag ugyanazt a feladatot látja el. Mind a két táblában 1-1 bejegyzés egyértelműen hozzá lett volna rendelve egy-egy üzemelő internet terminálhoz.

Ezen felül beláttam, hogy a konfigurációs beállítások állandó online lekérése miatt a klienseim nem tudnának tovább üzemelni, egy esetleges szerver kimaradás esetében.

Mindezeket figyelembe véve, a konfig logikához tartozó adattagokat, metódusokat (pl.: Árazás, frissítési URL stb.) átszerveztem a már meglévő „TerminalClient” logikai egységbe.

6.1.4. Log Controller

Eredetileg, projekt tervezése során ez a logika nem képezte az alkalmazás részét, fejlesztés során viszont felmerült bennem az igény, hogy szükség adódhat bizonyos esetekben a klienseken váratlanul bekövetkező események naplózására. Erre egy lehetőség lokálisan, az adott gépeken naplófájlokban eltárolni ezeket, de az üzemeltető szempontjából előnyös lehet ezeket központilag, szerveren tárolni.

Ennek megvalósítására hoztam létre ezt a kontrollert/logikai csoportot, melynek segítségével eltárolhatunk naplóbejegyzéseket. Egy bejegyzéshez tartozik egy cím, üzenet, dátum, küldő, illetve egy bejegyzés súlyossága mező.

A jelenlegi verzióban a kliensekben a naplózás implementációja minimális, de a webalkalmazás, illetve az adatbázis fel van készítve ennek a támogatására.

6.1.5. Rétegzés

Az alkalmazás struktúráját tekintve az alábbi projektekre van felbontva:

- **Data**: Adatbázis kapcsolatot szolgáltató könyvtár
- **Repository**: Alapvető adatbázis műveleteket tartalmazó könyvtárok
- **Logic**: Üzleti logikai réteg, feladatkörök szerint osztályokba csoportosítva
- **Tests**: Üzleti logika Unit tesztjeit tartalmazó réteg
- **terminalBackend**: ASP.NET Core Web API réteg
- **Models**: Adatbázis, illetve kliens-szerver közti kommunikáció alapjául szolgáló JSON objektumok modell osztályait tartalmazó projekt. Ez a projekt meg van osztva a többi dolgozat részét képező megoldás között.

6.1.6. Futtatáshoz szükséges információk

A szerveralkalmazás .NET 6-os futtatási környezetet igényel. Az adatbázis elérési útvonalának beállításához a projekt könyvtárában található appsettings.json megfelelő részét kell módosítanunk.

Az adatbázis kontextus alapértelmezett konfigurációjában úgy állítottam be, hogy az adatbázis táblák generálásakor automatikusan jöjjön létre egy admin jogosultsági körrel rendelkező felhasználó, „root” felhasználói névvel, illetve „abcd123!” hozzá tartozó jelszóval. Ennek elsődleges célja a tesztelés, illetve a szerver első üzembehelyezésének megkönnyítése volt. Ezek végeztével, mindenképpen javallott ennek a hozzáférésnek a törlése/módosítása, biztonsági okokból.

A szerver alapértelmezett beállításai alapján egy Microsoft Azure-ben üzemelő SQL adatbázist használ, de a Connection String, és esetlegesen a Data réteg megfelelő módosítása után bármilyen Entity Framework Core által támogatott lokális, vagy távoli adatbázissal együtt tud működni.

Tesztelés megkönnyítése érdekében a szerveralkalmazás jelenlegi verziója egy Microsoft Azure App Service-ben van üzemeltetve. Az összes kliens alapértelmezett beállításokkal ehhez a szerverhez próbál elsődlegesen csatlakozni, de ez igény esetén módosítható. Az elérési címe a következő: "https://terminalbackend20220427091120.azurewebsites.net".

6.2. Terminál böngésző kliens

A programcsomag elsődleges, felhasználók által használható kliens alkalmazása egy .NET 6-os WPF alkalmazás. Az alkalmazás lényegében egy percdíjjas alapokon működő böngésző implementáció, mely kontrollált környezetet biztosít a felhasználók általi használathoz.

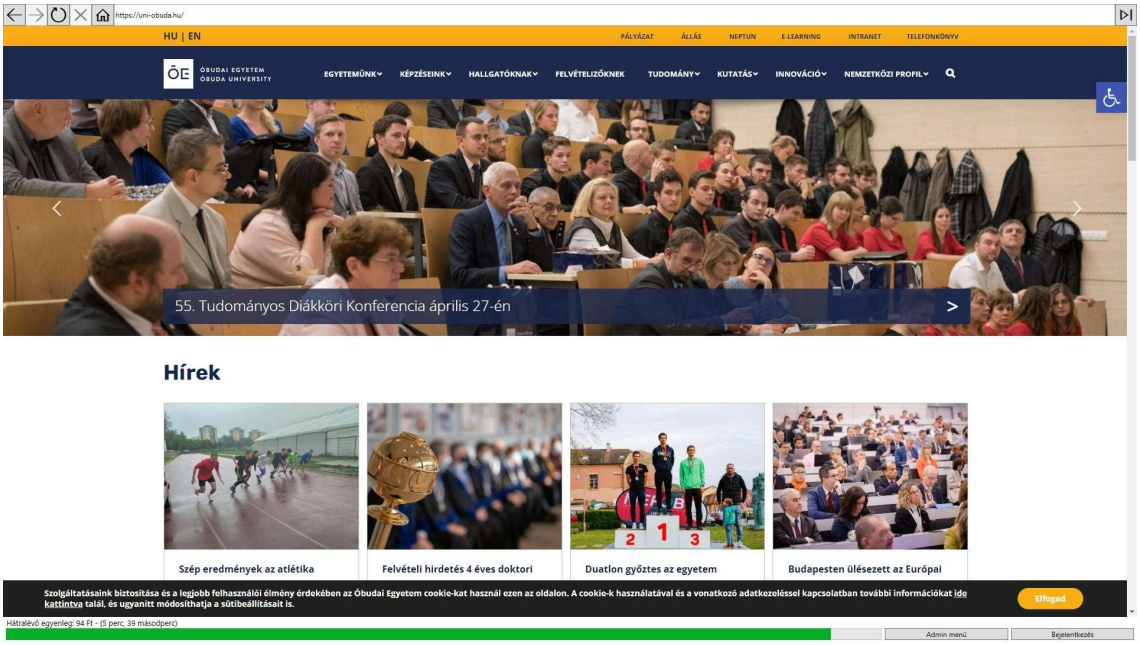
Dolgozatom tervezése során, eredetileg a CEFSharp szoftvercsomagot jelöltem meg, mint az általam választott böngészőmotor implementáció. Bár a tervezés során kipróbált, minimális funkcionalitással lefejlesztett teszt megvalósításom során nem találkoztam semmi előrehaladást blokkoló hiányossággal, a fejlesztés során több olyan problémát tapasztaltam, melyek alapján megkérdőjeleztem, hogy valóban ez-e az ideális választás számomra.

Ezek közül jelentős volt az a tény, hogy a CefSharp egy nyílt forráskódú, közösség által fejlesztett/karbantartott projekt. Életciklusa során több jelentős refaktoráláson, alternatív verzió kiadásokon esett át. Bár a hivatalos dokumentáció naprakész, implementálása során elengedhetetlen volt, hogy különféle harmadik féltől származó leírásokra, dokumentációkra támaszkodjak. A fentebb említettek miatt viszont ezeknél gyakran előfordul, hogy az abban leírtak nem mindig felelnek meg az általam használt megoldásnak, és ez jelentős többletmunkát okozhat.

A szakdolgozat első felének, időközi értékelésén az oktatók által javaslatként elhangzott, hogy a Microsoft WebView2 nevű implementációja is életképes megoldás lehet a CEFSharp helyett.

Így a fentebb említett okok miatt megvizsgáltam ennek a megvalósíthatóságát, és arra a megállapításra jutottam, hogy megfelelő alternatíva lehet, így a szakdolgozat megvalósítása során a „Microsoft.Web.WebView2” NuGet csomagra építkeztem az eredetileg eltervezettel ellentétben. Mivel a WebView2 Microsoft Edge-et használ, és az mára Chromium böngészőmotoron alapul, így ez a változás nem jár jelentős módosulásokkal a projekt funkcionalitását tekintve. [16]

Bár a CefSharp jelenleg több funkcióval, személyre szabási lehetőséggel rendelkezik, a WebView2 is szorosan követi, és mivel az utóbbi mögött ott van egy meglehetősen nagy szoftverfejlesztői cég, mint a Microsoft támogatása, így a jövőbeli frissítéseket tekintve pozitívok a kilátások.



6.1. ábra: Terminál kliens böngésző felülete

6.2.1. Egyenleg kezelés megvalósítása:

A terminál alapvetően fizetős használattal történő használatra készült. Ennek érdekében, a kliens alkalmazás soros porton kommunikál a fizetőeszköz kezelő panellal.

Amennyiben a fizetőeszköztől soros porton érkezik egy üzenet, abban az esetben ez a kliens soros pufferébe kerül. A kliens alkalmazás, egy külön szálon, végtelen ciklusban futó metódusban időközönként (jelenlegi beállítás szerint másodpercenként) megvizsgálja a puffer tartalmát. Amennyiben a benne található adat sikeres validáláson esik át, abban az esetben egy esemény segítségével értesíti a fő szálat ennek a bekövetkezéséről, illetve a hozzáadott egyenleg mennyiségéről.

A kliens ebben az esetben tájékoztatja a szervert a számlálóállás változásáról, majd hozzáadja az egyenleget a felhasználóéhoz.

Érmeelfogadón felül, lehetőségünk van egyszer használatos kódokkal is egyenleget hozzáadni. Amennyiben érvényes kódot visz be a felhasználó, abban az esetben ez a kód a szerveren elhasznált státuszra kerül állításra, majd hozzáadódik az egyenleghez.

Amennyiben egy felhasználó bejelentkezik az általa létrehozott fiókba, abban az esetben a kliens egyenlegéhez hozzáadódik a felhasználó fiókjában tárolt egyenleg.

Kijelentkezéskor ennek a fordítottja megy végbe, amennyiben még van hátralevő egyenleg a kliensen, az a felhasználó fiókjához kerül hozzá a szerveren.

Hátralevő egyenleget a képernyő alján található folyamatjelző sávon követheti nyomon a felhasználó. Amennyiben ez lejár, a kliens visszanavigál a kezdőoldalra, töröl minden adatot, mely a jelenlegi felhasználói munkamenethez köthető, és ezután készen áll egy következő ügyfél fogadására.

6.2.2. Felhasználói felület korlátozása

A kliens egyik legfontosabb feladata, hogy úgy biztosítson böngészési lehetőséget a felhasználó felé, hogy ő ne tudja elhagyni a számára szánt felületeket, és hozzáférni egyéb programokhoz, vagy az operációs rendszer többi felületéhez.

Ennek elérését több lépésben valósítottam meg, melyek közül a lényegesebbek:

Kritikus billentyűzet kombinációk tiltása:

Ennek a részproblémának a megoldására az elsődleges megközelítésem, az alkalmazás „KeyDown” eseményére történő feliratkozás, majd eseménykezelő metódusban a kritikus billentyű kombinációk elkapása, majd ezek lekezelése volt.

Meglehetősen hamar szembesültem vele, hogy bár ez a módszer jól működik, az alkalmazások szintjén kezelt kombinációk semlegesítésére (pl.: CTRL+N – új böngészőlap megnyitása), számos Windows szintű kombinációt nem lehetett kezelni ezzel a módszerrel.

Ennek megoldására, az alkalmazás belépési pontján, néhány a Windows részét képező, user32.dll könyvtár által szolgáltatott metódus segítségével vizsgálok minden billentyűzet leütést, a rendszer üzenet kezelési láncolatába való becsatlakozással. Amennyiben ez egy általam kritikusnak vélt kombinációnak a része, abban az esetben ezt lekezelem, ellenkező esetben továbbítom. [17][18]

Ezzel a megoldással már meglehetősen hatékonyan tudom szűrni a nem kívánt bemeneteket, de még mindig van egy kombináció, a CTRL+ALT+DEL, amit ezzel a módszerrel se lehet kiszűrni, feltehetőleg biztonsági okokból.

Korábbi tapasztalataimból emlékeztem, hogy a Sharpkeys nevezetű szoftver képes volt ezt a kombinációt is megakadályozni. Mivel ez szerencsére egy nyílt forráskódú szoftver, így a kód vizsgálata után megállapítottam, hogy ezt a

„SYSTEM\CurrentControlSet\Control\Keyboard Layout”

útvonal alatt található „Scancode Map” Windows Registry kulcs értékének a szerkesztésével éri el. A fentebbi program segítségével generáltam egy olyan értéket, ami eléri a CTRL+ALT+DEL kombináció letiltását, majd a programomhoz hozzáadtam egy részt, amely hozzáadja ezt a Windows beállításjegyzékhez. [19]

Letöltések tiltása:

A fejlesztés kezdetekor a WebView2 csomag legfrissebb stabil verziója még nem rendelkezett letöltéskezelő API-val, melyben manipulálhattam volna a letöltéseket, azonban a már ekkor elérhető „1.0.1189-prerelease” verzió már tartalmazta ezt.

Így fejlesztés során ezt használtam, és ebben a „DownloadStarting” eseményre feliratkozva már meg tudtam szakítani az éppen kezdődő letöltéseket.

Navigálás felülvizsgálata:

A WebView „NavigationStarting” eseményére feliratkozva visszavonom a navigációt, amennyiben az nem webes tartalomra irányul, vagy a felhasználó nem rendelkezik a szükséges egyenleggel.

Explorer.exe folyamat leállítása:

Eredeti terveim szerint a kontrollált környezet integritásának biztosítása érdekében terveztem az operációs rendszer részét képező, explorer.exe nevű folyamat kliens általi leállítását.

Korábbi tapasztalataim alapján, ennek hatására több, Windows felhasználói felület elem elérhetetlenné válik, biztosítva ezáltal, hogy a felhasználó nem szerez hozzáférést például a tálcához, start menühöz. Sajnos, mint ahogy az a fejlesztés során kiderült, a WebView2 megfelelő működésének érdekében elengedhetetlen az explorer.exe folyamat futása, így más megoldások után kellett néznem.

A korábban említett user32.dll által szolgáltatott FindWindow() és ShowWindow() metódusok használatával, sikeresen láthatatlanra tudtam állítani a tálcát és az elemeit. Bár az alkalmazás alapértelmezetten teljes képernyős módban fut, de ennek ellenére ez egy másodlagos védelemként szolgálhat illetéktelen hozzáférés ellen.

További módosítások:

A zökkenőmentes működés érdekében érdemes lehet, a gazda operációs rendszer megfelelő beállításainak módosítására, hogy a felhasználói munkamenetet minél kevesebb külső tényező zavarhassa meg. Ilyen lehet például a Windows 10 részét képező fókuszsegéd használata, a megjelenő értesítések lenémítása érdekében.

A kellő biztonság érdekében, a billentyűkombinációkon túl a feladatkezelő a Windows beállításjegyzék szintjén is letiltásra kerül, amennyiben a kliens az „unsafeChanges” opció engedélyezésével van futtatva.

6.2.3. Lokális beállítások kezelése:

A program tervezésekor célkitűzések között szerepelt, hogy legyen lehetőség lokálisan, illetve opcionálisan online is kezelni egy-egy kliens beállításait. Több iteráció után, az alábbi megoldást találtam ideálisnak. A kliensalkalmazásban megtalálható egy Config osztály, mely Singleton tervezési mintát követve, a benne található adattagokkal meghatározza a beállítás fájl struktúráját.

A kliens minden indulásnál megkísérli egy .JSON fájlból deszerializálni a beállításokat, majd lekérést küld a szerver felé, bizonyos online beállítás értékek iránt. Amennyiben innen kap érvényes választ, így ezeket beleírja a konfig Singleton osztály megfelelő adattagjaiba.

Amennyiben indulásnál nem található érvényes konfigurációs fájl, vagy annak a verziója régebbi az alkalmazás által elvárttól, új fájl generálódik a konfig osztály konstruktora alapján.

Az alkalmazás leállításakor a konfigurációs osztály, ami már esetlegesen tartalmazhatja a szervertől kapott módosított értékeket is, kiírásra kerül a .JSON fájlba, mely az alkalmazás futtatható binárisának mappájában található, „Settings_Client.json” néven (ennek a kliensnek az esetében). Így amennyiben a következő futtatásnál a szerver nem elérhető, a már frissített beállítások elérhetőek lesznek lokálisan is.

A konfigurációs fájlok kezelése gyakorlatilag ugyanígy került implementálásra a másik két kliens alkalmazásban is, az online komponenstől eltekintve, így ezt már azokban a bekezdésekben nem részletezem ismételten.

6.2.4. Futtatáshoz szükséges információk:

A valódi környezetben történő futtatáshoz javasolt a programcsomag részét képező frissítő alkalmazás használata, de teszt jelleggel futtatható enélkül is. A konfigurációs fájl alapértelmezett adatai megfelelnek egy kiindulási pontnak, de érdemes módosítani a kliens bejelentkezési adatokat arra, amit létrehoztunk a szerveren, kifejezetten ennek a terminálnak. Alapértelmezés képpen a terminal00 fiók kerül beállításra, ami mindig létezik a szerveren. Amennyiben lokális szervert szeretnénk használni, akkor azt is érdemes itt átállítani.

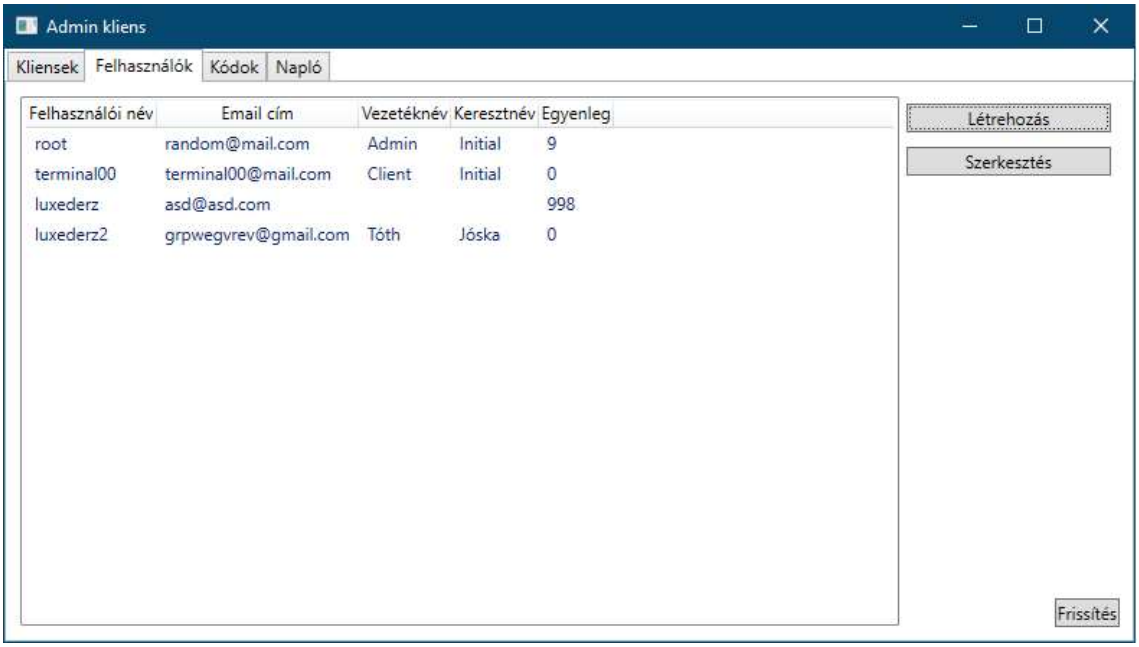
Ezen felül, a kliens a megfelelő környezet kialakítása érdekében végez olyan, maradandó módosításokat is a gazda számítógépen, ami nem feltétlenül előnyös teszt környezetben. Ezt elkerülendő, alapértelmezés képpen a konfigurációs fájlban le van tiltva az „unsafeChanges” opció, amit éles környezetben erősen ajánlott bekapcsolni.

Ennek hiányában többek között nem kerül letiltásra néhány billentyűzet kombináció, nem kerül módosításra a gazda operációs rendszer alapértelmezett DNS szervereinek a címe stb.

Végezetül pedig, szintén ebben a konfigurációs fájlban szükséges a fizetőeszköz panelhoz hozzárendelt soros port kiválasztása, a „ComPort” opció megfelelő értékre való állításával.

Vegyük figyelembe, hogy a kliens úgy lett tervezve, hogy a felhasználók ne tudjanak kilépni belőle. A program „Debug” módban futtatva az ALT+W kombinációval bezárható, de „Release” módban csak úgy tudjuk elhagyni, hogy bejelentkezünk egy „admin” jogosultsági körrel rendelkező fiókkal. Ekkor a képernyő alján megjelenik egy Admin menü feliratú gomb, amivel elhagyhatjuk az alkalmazást.

6.3. Admin kliens



6.2. ábra: Az elkészült admin kliens

Ennek a WPF kliensalkalmazásnak segítségével menedzselhetjük a szerveren tárolt adatokat. Az elkészült klienst a szerveralkalmazás végpontjainak csoportosítása szerint bontottam fülekre. Az adott füleken lehetőségünk van a releváns adatbázis táblában tárolt rekordok grafikus felületen történő kezelésére.

A kliensek fülön van lehetőségünk módosítani, törölni meglévő klienseket, online konfigurációs adatokat megadni, ami a következő indításnál kerül letöltésre, vagy áttekinteni a számláló állásokat. Ezen a fülön új bejegyzéseket létrehozni nincs lehetőségünk, mivel ezek automatikusan jönnek létre, ahogy egy adott kliens alkalmazás periodikus állapotjelentést küld a szervernek.

A kódok fülön, a már a szerveralkalmazásnál taglalt opciókon kívül lehetőségünk van exportálni a még érvényes kódokat .csv formátumban, egy általunk választott helyre.

Futtatáshoz szükséges információk:

A használat szerveres bejelentkezéshez kötött, bármely fiókot használhatjuk, mely rendelkezik az „admin” jogosultsági körrel.

Az admin kliens is az előző szegmensben taglalt konfigurációs fájl sémát követi, de itt nem volt indokolt túlságosan sok beállítási lehetőséget biztosítani, jelenleg csak a szerver elérési útvonalát van lehetőségünk beállítani.

Ennek ellenére, érdemesnek találtam a korábban használt beállítás kezelő módszert itt is alkalmazni, az esetleges későbbi bővítési lehetőségek érdekében.

6.4. Frissítő/Watchdog alkalmazás

Ebben a konzolos projektben, a rendszerv fejezetben leírtak mentén haladva megvalósítottam a fő kliens frissítéséért, és üzemeltetéséért felelős alkalmazást.

A frissítő program indításakor ellenőrzi, hogy a beállításaiban megadott elérési útvonalon megtalálható-e egy terminál kliens, illetve, hogy a szerver által az adott klienshez rendelt verzió megegyezik-e a jelenleg telepítettel. Ha a szerver új verzióról tájékoztat, annak a webes elérési útvonala felülírja az eddig lokális beállításban tároltat. Ha a fentebbi két kitétel közül bármely teljesül, a frissítő alkalmazás kitörli az eddigi kliens mappáját, letölti a friss verziót tartalmazó archívumot, majd kicsomagolja azt.

Ezt megelőzően egy ideiglenes mentést készít a kliens beállításairól, amit a művelet végeztével visszaállít, annak érdekében, hogy a kliens beállításai megmaradjanak.

Ezzel lezárul a frissítő alkalmazás frissítési fázisa, innentől a fő kliens leállásáig felügyeleti feladatokat lát el. Beállításban megadható időközönként ellenőrzi, hogy fut-és válaszol-e a terminál kliens, illetve, hogy a folyamathoz tartozó ablak előtérben van-e. Amennyiben nem, akkor elindítja, újraindítja, vagy előtérbe kényszeríti azt.

Ezekkel az ellenőrzési ciklus iterációkkal egyidőben vizsgálja a gazda gépen futó folyamatokat. A „Settings_Launcher.json” fájlban lehetőségünk adódik definiálni egy tiltott folyamatokat tartalmazó listát. Amennyiben a futó folyamatok között szerepel bármely ezen listán szereplő folyamat, úgy az azonnal leállításra kerül, minden iterációban.

Új frissítés előkészítése:

Egy frissítési fájl a fő terminál kliens lefordított bináris állományait, és a hozzá referenciaként tartozó könyvtárakat tartalmazó .ZIP fájlból áll. Ennek az előállításának a legegyszerűbb módja az egész Release módban fordított kimeneti mappa .ZIP tömörítése. Opcionálisan használhatjuk a Visual Studio Publish varázslóját erre, amiben lehetőségünk van egy minden szükséges referenciát magában hordozó futtatható állományt fordítanunk. Az archívum igény szerint tartalmazhatja a kliens konfigurációs fájlt, de amennyien az nincs jelen, első indításnál generálódik egy új az alapértelmezett értékekkel. Amennyiben azt szeretnénk, hogy a kliens mindenképpen generáljon egy új konfigurációt frissítés után, az első futtatás során, a fő kliens alkalmazás kódjában, ClientConfig osztály konstruktorában növeljük a ConfigVersion adattag értékét fordítás előtt, így a régi .JSON fájl eldobásra fog kerülni.

Az így elkészült. .ZIP fájlt fel kell tölteni egy tetszőleges web tárhelyre, amely biztosít direkt fájl elérést.

A frissítés kijelölésére két lehetőségünk van.

Az első, hogy hozzáadhatjuk a frissítő alkalmazás konfigurációs fájljához az új archívum URLjét, a második pedig, hogy az admin alkalmazás kliens fülén hozzárendeljük az új URL-t egy tetszőleges klienshez.

Fontos megjegyezni, hogy amennyiben a szerveren egy adott klienshez van beállított frissítés fájl URL, az minden esetben felül fogja írni a frissítő alkalmazás beállításaiban találtató lokális értéket, függetlenül, hogy mi található benne.

Futtatáshoz szükséges információk:

A kliens működéséhez rendszergazdai jogosultságokra van szükség, melyet az „app.manifest” fájl módosításának köszönhetően automatikusan megigényel az operációs rendszertől futtatáskor.

A korábban leírtak jelentős része itt és érvényes, a legtöbb alapértelmezett beállítás megfelel egy kiindulási alapnak. Éles környezetben való üzemelésnél szükséges átírni a szerver URL-en kívül a felhasználónév jelszó párost egy újra, ami szintén rendelkezik a „Terminal” szerveroldali jogosultsági körrel. A frissítő alkalmazás sikeres bejelentkezést követően továbbadja a saját bejelentkezési adatait a fő kliensnek, annak a beállítás .JSON fájlját módosítva, annak érdekében, hogy ez nehogy feledésbe merüljön.

Fontos megjegyezni, hogy amennyiben a frissítő alkalmazással futtatjuk a klienst, bezárásához a fő kliens admin menüjéből kell a kilépést kezdeményezni, mely leállítja ezt a watchdog alkalmazást is. Ellenkező esetben, pár másodpercen belül újraindításra kerülne a terminál kliens, a fentebb taglalt ellenőrzési ciklus miatt.

6.5. Fizetőeszköz kezelő panel

A fizetőeszköz kezelő panel kapcsolási rajzát, illetve nyomtatott áramköri tervét az EasyEda elnevezésű tervezőprogramban hoztam létre.

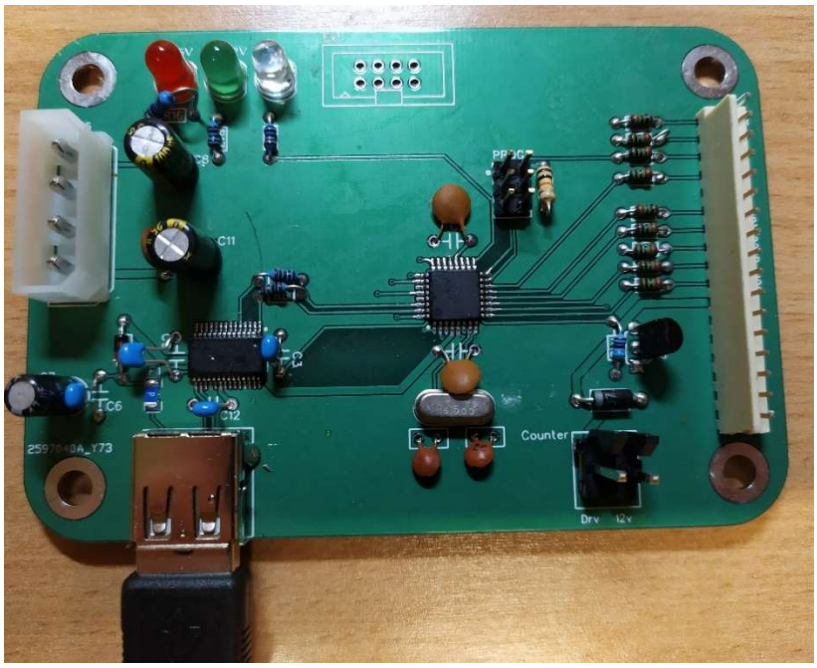
A kapcsolási rajz és a nyomtatott áramkör terv páros megtalálható a dolgozat függelékei között könnyen áttekinthető kép formátumban (Függelék 1 és Függelék 2), illetve a forrásállományok között exportált .json formátumban, ami visszaimportálható az EasyEda kliensbe további szerkesztésre, elemzésre.

6.5.1. Áramkör megvalósítása

A kapcsolási rajt megtervezése után, a nyákterv megvalósítása következett. A megfelelő komponensek elhelyezése után, a tervezőprogram „autorouting" funkcióját használva automatikusan legeneráltam a vezetősávokat, a kapcsolási rajzban meghatározottak alapján.

Tapasztalatom szerint az automatikus vezetősáv generálás többnyire használható végeredményt ad, de valamennyi korrigálásra, finomításra mindig szükség van, általában egyre több, ahogy nő egy projekt komplexitása. Jelenlegi esetben elsősorban a vezetősáv vastagságok finomhangolására, illetve utólagos manuális vezetősáv útvonal optimalizálásra került sor.

Az így elkészült áramkör tervet egy kínai áramkörgyártó céggel gyártattam le, a forrásfájlokból előállított gyártási utasításfájlok alapján, majd az elkészült panelba magam forrasztottam be a komponenseket.



6.3. ábra: Az elkészült fizetőeszköz kezelő panel

A 6.1-es táblázatban definiálom a tervezett áramkör csatlakozóinak a kiosztását.

Megnevezés a panelon	Feladat	Megjegyzés
USB csatlakozó	Usb kapcsolat a mikrovezérlő és a kliens között	
Molex.1	5v tápfeszültség	
Molex.2	Föld	
Molex.3	Föld	
Molex.4	12v tápfeszültség	
Coin.1-2	Érmeelfogadó A, B csatorna bemenet	
Coin.3	Használaton kívül	
Coin.4	Érmeelfogadó F csatorna bemenet	
Coin.5	Kulcs	Fordított polaritás elleni védelem
Coin.6-7	Érmeelfogadó E, D csatorna bemenet	
Coin.8	Használaton kívül	
Coin.9	Érmeelfogadó C csatorna bemenet	
Coin.10	Érmeelfogadó C csatorna tiltás	
Coin.11	12v tápfeszültség	
Coin.12	Föld	
Coin.13-17	Érmeelfogadó D, E, F, B, A csatorna tiltás	
Ext.1-8	Bővítési lehetőseg	Mikrovezérlő D2, D3, A1, A2, A3, A4, A5 portjai
Counter.12v	Számláló 12v kimenet	
Counter.drv	Számláló vezérelt láb	
ICSP.1-6	Programozási csatlakozó	

6.1. táblázat: Nyomtatott áramkör csatlakozó kiosztása

6.5.2. Hardver

A tervezett panel „Coin acceptor” csatlakozásának a kiosztása választott érmeelfogadó egység útmutatója alapján készült.[15]

Ezen a csatlakozón található használaton kívüli tüskék az érmeelfogadóhoz tartozó, opcionális érme szétválogató kiegészítő vezérlésére szolgálnak, mely nem képezi részét a dolgozatnak.

A nyomtatott áramkörön a 12v-os tápfeszültségre egyedül az érmeelfogadónak, illetve a mechanikus számláló kimenetnek van szüksége, az elektronika többi része működik ennek a hiányában is.

A mikrovezérlő adatlapja alapján 2 érték van, amit mindenképpen figyelembe kell vennünk a portok használata során. Az egy porton folyó maximális áram nem haladhatja meg a 40 mA-t, illetve a teljes mikrovezérlő áramfelvétele nem lehet több mint 200 mA. [20]

Ezek abszolút maximum értékek, tehát egy megfelelően tervezett rendszerben soha nem is közelíthetjük meg ezeket. Az általam választott Cashflow 126-os érmeelfogadó egység kimenetei a mikrovezérlő bemenetként konfigurált portjaira vannak csatlakoztatva. Az így beállított portok magas impedanciás módban üzemelnek, tehát a rajtuk keresztülfolyó áram elhanyagolható, csak bináris logikai jelszinteket figyelnek.

Az érmevizsgáló (letiltó/engedélyező) bemenetei a mikrovezérlő kimenetként konfigurált portjaira kerültek csatlakoztatásra. Viszont, az érmevizsgáló bemenetei szintén csak 5v-os logikai jelszinteket vizsgálnak, így itt sem beszélhetünk érdemleges árammennyiségek folyásáról. Méréseim során semelyik nem haladta meg az 1 mA-t.

Ennek ellenére, az esetleges balesetek elkerülése érdekében indokoltnak láttam korlátozó ellenállásokkal ellátni az erre a csatlakozóra tartó mikrovezérlő portokat, így egy esetleges rövidzár esetén se károsodik a mikrovezérlő.

Mivel a rendelkezésre álló pinek száma véges volt a mikrovezérlőn, így arra a döntésre jutottam, hogy az érmeelfogadó csatornánkként különbontott engedélyező/tiltó bemeneteit egyként kezelem. Így elvész a lehetőség arra, hogy egy adott érmét szoftveresen el tudjak utasítani, de ilyen funkcionalitás nem is képezte a projekt terveinek a részét.

A panel rendelkezik még egy kimenettel is, mechanikus számláló vezérléséhez, mellyel számon lehet tartani a bevételt. Mivel az ilyen eszközökben egy apró elektromágnesnek kell mechanikus elven elmozdítani a számlálót, így az ezáltal felvett áram már meghaladná a mikrokontroller által toleráltakat.

Ennek fényében a számláló kimenetet egy, 2n2222 típusú NPN tranzisztor segítségével kapcsolom, amely már elbírja a terhelést.

6.5.3. Szoftver

A mikrovezérlő programját az Arduino könyvtárjainak felhasználásával készítettem el.

Fejlesztéshez a Visual Studio Code-ot használtam, a PlatformIO kiegészítő segítségével. Ez egy kifejezettem beágyazott eszközökre való fejlesztésre specializálódott fejlesztői környezet, melyben véleményem szerint a sok kényelmi funkciónak köszönhetően sokkal hatékonyabban lehet dolgozni, mint például az Arduino IDE-ben. Az írott kódállomány is kompatibilis a két program között.

A mikrovezérlőre írt kód meglehetősen egyszerű, a megfelelő portok ki- és bemenetként való konfigurálása után folyamatosan figyeli a megfelelő csatorna bemeneteket az érmeelfogadó felől, és amennyiben valamelyiken alacsony logikai értéket észlel, értesíti a terminál kliens alkalmazást a soros kimeneten keresztül, majd lépteti a mechanikus számláló kimenetet is a megfelelő értékkel.

A mikrovezérlőt alapvetően a panelon található ICSP csatlakozón keresztül lehet programozni. Az ICSP az In-Circuit Serial Programming kifejezést, tehát áramkörön belüli soros programozást rövidíti. Ennek használatához léteznek dedikált programozó elektronikák, de én a szakdolgozat során egy különálló, Arduino Uno fejlesztői panelt használtam programozóként, a fizetőeszköz kezelő panel inicializálására.

Az első alkalommal ennek segítségével feltöltöttem az ATmega328 flash memóriájába egy bootloader programot, az Arduino IDE segítségével. Ez a mikrovezérlő indulásakor, még a programozó által írt kód előtt fut, és lehetővé teszi, hogy az UART bemeneten programozzuk a mikrovezérlőt. És mivel a dolgozat részét képező áramkörön az UART kommunikáció ki van vezetve az USB-re egy FT232RL integrált áramkörön keresztül, így lehetőség adódik a mikrokontroller USB-n keresztüli programozására, amely jelentősen meggyorsíthatja a folyamatot.

7. TESZTELÉS

Az előző fejezetben taglaltak mentén elkészült rendszer, specifikációk szerinti implementálása után az infrastruktúrát alkotó modulok tesztelése következett. A következő bekezdésekben végig haladok ezeken, és a tesztelés módszerén, lépésein felül kitérek a tesztelés során felmerülő problémákra, és azoknak a megoldására.

Mivel a dolgozatom megvalósítását meglehetősen sok, eltérő technológián alapuló komponens adja, így mindegyik szegmensben igyekszem az adott környezethez illő tesztelési technikákat alkalmazni. Az asztali alkalmazások Windows 10-es környezetben, rendszergazdai jogosultságokkal rendelkező fiók alatt, x64-es architektúrán és rendszeren kerültek tesztelésre, de nem látom okát, hogy x86-on ne működne.

7.1. Szerver tesztelése

A webalkalmazás tesztelését automatizált unit tesztekkel, illetve a Postman elnevezésű program segítségével, az API megfelelő végpontjaira küldött http kérésekkel végeztem el.

A tesztelés eredményeit a 7.1-es táblázat tartalmazza.

Teszt tárgya	Tesztelés módja	Eredmény
Felhasználói fiókok létrehozása, bejelentkezés	Adatbázis elemzése, Webes kérések válaszai, Unit tesztek	✔
Alapvető fiókkezelési CRUD műveletek tesztelése	Adatbázis elemzése, Webes kérések válaszai, Unit tesztek	✔
Státuszjelentés küldése a szervernek, klienstől	Adatbázis elemzése, Webes kérések válaszai, Unit tesztek	✔
Egyszer használatos kódok menedzselése	Adatbázis elemzése, Webes kérések válaszai, Unit tesztek	✔
Kliensek szerkesztésének, törlésének, lekérésének tesztelése	Adatbázis elemzése, Webes kérések válaszai, Unit tesztek	✔
Naplófájlokhoz kötődő CRUD műveletek tesztelése	Adatbázis elemzése, Webes kérések válaszai, Unit tesztek	✔
Hitelesítés/jogosultság nélküli erőforrás hozzáférés megtagadásának tesztelése	Webes kérések válaszai, Http hibakódok elemzése	✔

7.1. táblázat: Szerver tesztek

7.2. Terminál kliens tesztelése

A terminál kliensen végzett funkcionalitási és rendszer tesztek eredményét a 7.2-es táblázat tartalmazza.

Teszt tárgya	Tesztelés módja	Eredmény
Weboldalak megjelenítése, navigálás a beépített böngészővel	Manuális tesztelés	☑
Fiókkezelés, egyenleg mentése/visszaállítása	Adatbázis elemzése, Szerveroldali töréspontok használata, manuális tesztelés	☑
Egyszer használatos kódok kezelése	Adatbázis elemzése, Szerveroldali töréspontok használata, manuális tesztelés	☑
Munkamenet visszaállítása az egyenleg lejártával	Manuális tesztelés, lokális könyvtárak vizsgálata	☒
Fizetőeszközök kezelése	Manuális tesztelés	☑
Konfigurációs fájlok kezelése	Manuális tesztelés, lokális könyvtárak vizsgálata	☑
Szerver kommunikáció	Szerveroldali töréspontok használata	☑
Felhasználói felület elhagyásának megakadályozása	Manuális tesztelés	☑

7.2. táblázat: Terminál kliens tesztjei

7.2.1. Teszt során felmerülő hibák, és megoldásuk

A felhasználói adatok törlésekor (sütik, ideiglenes fájlok stb.) egy futásidejű kivétel került eldobásra, miszerint az ehhez szükséges WebView funkciót csak a felhasználói felületet futtató szálról lehet meghívni. A hibának a tünetét viszonylag egyszerűen lehetne kezelni, egy Threading.Dispatcher.Invoke() hívással, de mivel biztos voltam benne, hogy ez a funkció korábban megfelelően működött, így inkább a regresszió okának a felkutatására összpontosítottam.

Miután a hiba továbbra is fennált egy régi Git commitra visszaállva is, több órányi hibakeresés után sikeresen megtaláltam az okát. A WebView2 explicit konfiguráció hiányában alapértelmezetten a gazda számítógépre telepített WebView2 futtatási környezet Evergreen (örökzöld) verzióját használja. Ez a futtatási környezet gyakorlatilag a Microsoft Edge böngésző binárisainak egy módosított változata, mely beágyazásra alkalmassá teszi. [21]

Ez az alkalmazás háttérben automatikusan, minden figyelmeztetés nélkül frissül, mely sajnos bekövetkezett a szakdolgozat írása alatt is, egy olyan módosítást hozva magával,

mely a fentebbi hibához vezetett. Ennek kiküszöbölésére, módosítottam a kliens alkalmazásomat, hogy a WebView2 futtatási környezet egy fixált (v100.0.1185), jelenleg második legfrissebb verzióját használja, amivel az alkalmazás ismét helyesen működött.

Az Evergreen futtatási környezet előnye, hogy automatikusan rendelkezésre állnak a biztonsági és funkció frissítések, illetve, hogy nem kell mellékelnünk a közel 200 MB-os alkalmazást a klienshez. Viszont, mint ahogy tapasztalhattuk is, olyan váratlan hátrányokkal jártat, mely egy üzemelési környezetben elfogadhatatlan.

A módosítások után egy esetleges kliens frissítésekhez, megfelelő tesztelés után, a fejlesztőnek kell mellékelnie a legfrissebb változatot.

Az itt taglalt lépések után a problémát megoldottnak, a tesztet pedig sikeresnek tekintem.

7.3. Frissítő alkalmazás tesztelése

Mivel ennek az alkalmazásnak a logikája kellőképpen lekövethető szekvenciálisan, így a fejlesztés során iteratív fejlesztést/tesztelést alkalmaztam. Egy részprobléma megoldása után egyből teszteltem azt, és reflektáltam a potenciálisan felmerülő hibákra. Ennek ellenére a dolgozat véglegesítése során a 7.3-as táblázatban található funkcionalitási teszteket elvégeztem.

Teszt tárgya	Eredmény
Frissítés URL lekérése a szervertől	✓
Frissítés letöltése, telepítése	✓
Szerveres hitelesítés	✓
Konfiguráció átvitele a régi kliensből a frissítettbe	✓
Terminál kliens felügyeleti funkciók működése	✓
Tiltott folyamatok leállítása	✓

7.3. táblázat: Frissítő alkalmazás tesztjei

7.4. Admin kliens tesztelése

Mivel az admin kliens relatíve kevés egyedi üzleti logikát tartalmaz, túlnyomó részt csak CRUD kéréseket küld a szerver felé, illetve vizualizálja az adatokat, így a többihez képest ennek az alkalmazásnak a tesztelése kevésbé volt részletekbe menő. Az API kérésekért felelős osztályok a terminál kliensből lettek újrahasznosítva, így azok már ott egyszer tesztelésre kerültek. A manuális tesztelés eredményét a 7.4-es táblázat tartalmazza.

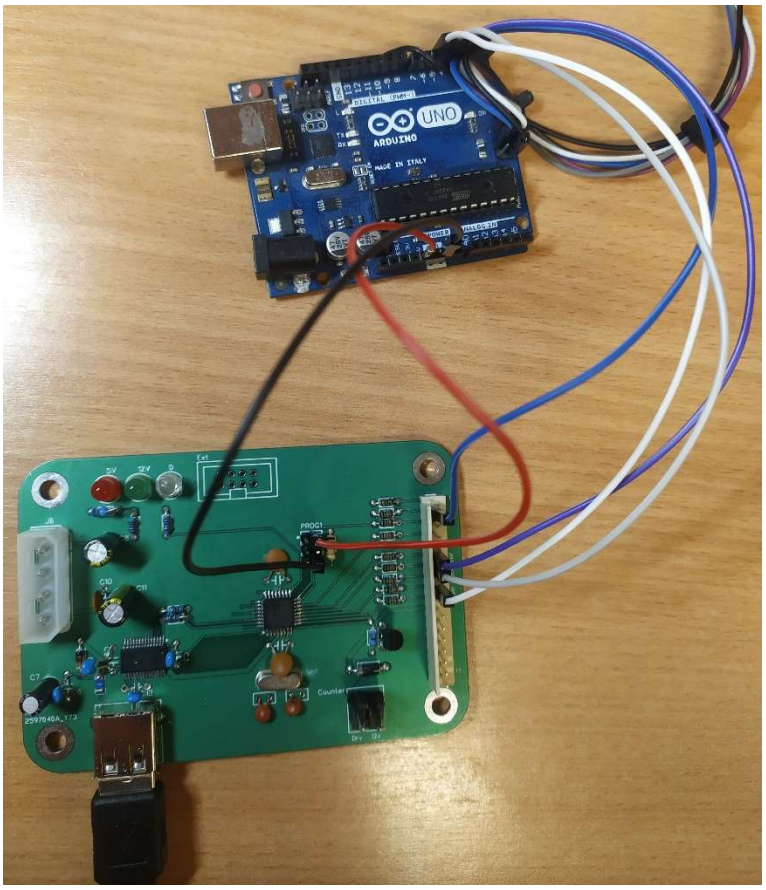
Teszt tárgya	Eredmény
Szerver által küldött adatok megjelenítése	✅
Megfelelő szerver végpontok meghívása hitelesítéssel	✅
Egyszer használatos kódok .CSV exportálása	✅

7.4. táblázat: Admin kliens tesztek

7.5. Fizetőeszköz kezelő panel tesztelése

A panel teszteléséhez bevontam egy különálló Arduino Uno fejlesztőpanelt, melyet felprogramoztam, hogy szimulálja az érmeelfogadótól érkező jeleket, 200, 100, 50, 20 Ft sorrendben, 10 másodperces időközökkel.

Ezt csatlakoztatva a saját panelomhoz, a tesztelés idejéig módosítottam a kliensalkalmazást, hogy ebben a sorrendben várja a bemeneteket, és jelezze a hibakeresési konzolra, amennyiben ettől eltérőt tapasztal.



7.1. ábra: Fizető panel tesztelési konfiguráció

Az fentebbi konfigurációban elindítottam a programot, majd futni hagytam megközelítőleg 4 óráig. 10 másodperces időközökkel számolva ez idő alatt ~1440-szer küldött a panel üzenetet a kliens felé. A teszt végén 0 elvárt szekvenciától való eltérést tapasztaltam a konzolon, így a tesztet sikeresnek tekintem.

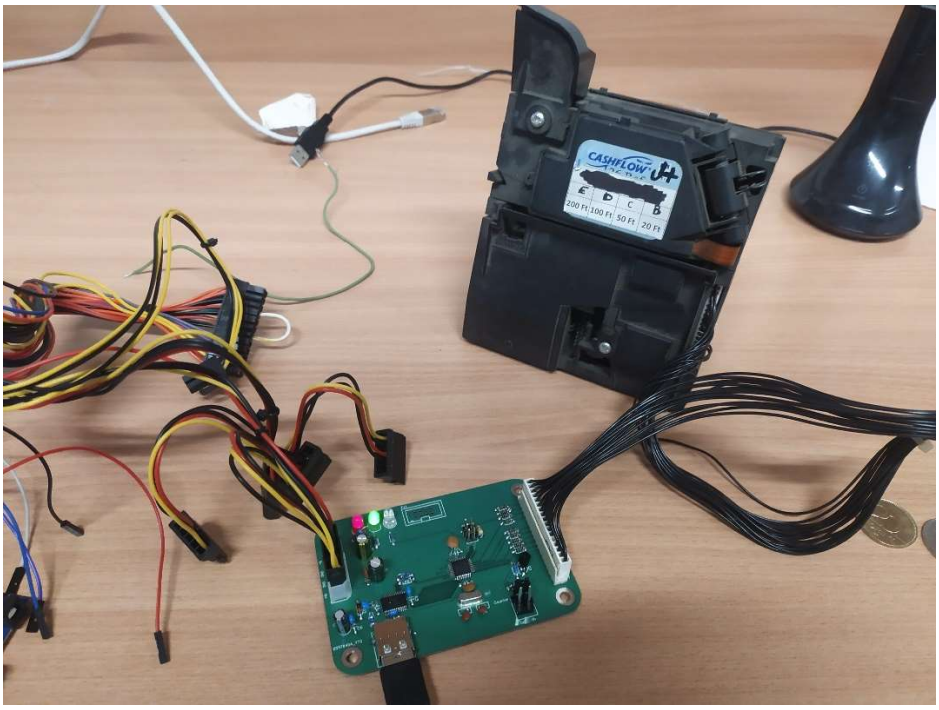
7.6. Integrációs teszt

Befejezésül, az elkészült infrastruktúrát alávetettem egy végső integrációs tesztnek. A különböző tesztlépések során a dolgozat részeit képező programok külön-külön kerültek tesztelésre (kliensek lokálisan futtatott szerverekkel, frissítő nélkül stb.), így indokoltnak találtam, hogy egy végső ellenőrzés formájában kipróbáljam a modulokat egy valós, üzemelő környezetet szimulálva is. A szerver az Azure App Service-ben fut, a kliens Release paraméterekkel került fordításra, (elrejtve így a fejlesztéshez szánt hibaüzeneteket), és most a frissítő alkalmazás futtatja azt. Valamint csatlakoztatva és tesztelve lett az érmeelfogadó is.

A tesztelés eseménymentesen lezajlott, egyetlen észrevételem volt, hogy az érmevizsgáló akkor is elfogadja a pénzt, amikor a kliens alkalmazás nem fut, így az nem is kerül jóváírásra.

Ilyesmi elsősorban rendszerindításkor, vagy egy esetleges alkalmazás összeomláskor fordulhat elő, amikor még éppen a frissítő alkalmazás nem indította újra a terminál klienst.

Ennek korrigálására módosítottam a mikrovezérlőn futó kódot, hogy induláskor az érmeelfogadót tartsa tiltott állapotban, és csak akkor engedélyezze azt, amikor a klienstől induláskor kap egy engedélyező jelet. Így ennek bekövetkeztéig az érmevizsgáló a kassza helyett a pénzvisszaadó nyíláshoz tereli az érméket.



7.2. ábra: Érmeelfogadó tesztelése az integrációs teszt során

8. ELKÉSZÜLT PROJEKT ÉRTÉKELÉSE

A tesztelés eredményeit is figyelembe véve, a projektet elkészültnek tekintem. A tervezés fejezetben taglalt megvalósítási lépésektől csak minimálisan eltérve, a dolgozat összeállított specifikációiban megkövetelteknek teljes mértékben megfelelve, dokumentáltam a fejlesztés menetét az implementációs fejezetben.

Az elkészült infrastruktúra alkalmas egy internet terminál rendszer működtetésére, és rendelkezik azokkal a funkciókkal, melyeket hiányosságként tártam fel a hasonló, már létező megoldások elemzése során.

9. TOVÁBBFEJLESZTÉSI LEHETŐSÉGEK

Ebben a fejezetben áttekintem azokat a pontokat, melyeket a tesztelés során potenciális fejlesztési lehetőségként azonosítottam.

9.1. Mikrovezérlő szoftverének automatikus frissítése

Jelenleg, amennyiben módosítani szeretnénk a fizetőeszköz kezelő panelon futó szoftver verzióját, erre csak helyben, az ICSP, vagy az USB csatlakozón adódik lehetőségünk. Ez alapvetően nem jelent problémát, mivel ritkán kerülhet sor ennek a módosítására, de tekintettel az egyik, lentebb taglalt esetleges fejlesztésre, hasznunkra válhat ennek az újra tervezése.

Ennek a megvalósítására felhasználható a már meglévő frissítés kézbesítési megoldást, kibővítve azt. Mivel a mikrovezérlő már tartalmazza a bootloader programot, amely lehetővé teszi az UART interfészen keresztüli szoftver frissítést, így ez a megoldás nem igényel semmilyen hardveres módosítást.

A mikrovezérlő forráskódjából fordított .HEX kiterjesztésű fájlt mellékelnénk egy kliensfrissítést tartalmazó .ZIP fájlba, melyet a kliens program feltöltene a mikrovezérlőre az „avrdude” nevezetű szoftverrel, melyet szintén mellékelnénk. Az Arduino IDE, vagy a PlatformIO feljesztői környezet is ezt alkalmazza a háttérben a kód feltöltésekor. Ennek a megoldásnak a megvalósításához referenciaként tekinthetünk az alábbi két Github repositoryra, melyek hasonló problémát oldanak meg. [22] [23]

9.2. Felhasználó felület megjelenésének a fejlesztése

A kliens jelenleg egy minimalista megjelenéssel rendelkezik, a jövőben esetlegesen érdemes lehet ennek a továbbfejlesztése, a felhasználói élmény fokozásának érdekében.

Mivel a kliens rendelkezik egy beépített böngészővel, így bizonyos menüelemek tervezése során HTML-en alapuló megvalósításban is gondolkozhatunk.

9.3. További fizetési módok hozzáadása

A tervezett áramkör alkalmas eltérő fizetési módok támogatására is, a megvalósítás nem igényel újabb áramkör tervezését. Ezen a területen érdemes lehet például bankjegy elfogadók támogathatóságának a vizsgálata.

9.4. Külső alkalmazások futtatása

Felmerülhet az igény, hogy az internet terminálon böngészésen kívül lehessen bizonyos, az üzemeltető által engedélyezett, harmadik féltől származó alkalmazások kontrollált körülmények között való futtatására.

Ennek elérése érdekében a kliens módosításán túl fejlesztenünk kell a frissítő/watchdog alkalmazáson is, hogy biztosítva legyen, hogy a külsős alkalmazásból se lehessen elhagyni a kontrollált környezetet. Szükséges megvalósítandó lépések közül a jelentőségteljesebbek közé tartozik az előtérben tartás, az alkalmazásból megnyitható új ablakok korlátozása, billentyűkombinációk kezelése stb.

9.5. Naplózás támogatása

Ahogy az már a korábbi fejezetekben érintve lett, implementáció során, utólagos elgondolásként a szerver alkalmazás el lett látva egy naplóbejegyzések kezelésére szolgáló üzleti logikával. Bár ez tesztelt, és megfelelően működik, a kliensekbe még nem került bele ennek a támogatása, mely hasznos lehet üzemelési környezetben működő terminálok esetében.

Az itt felsorolt továbbfejlesztési lehetőségek közül mindegyik (a fizetőeszköz támogatást bővítőn kívül, mivel az fizikai módosítást igényel) célba juttatása megoldható a dolgozat részét képező, frissítésekért felelős alkalmazás segítségével.

10.ÖSSZEFOGLALÓ

Lezárásképpen, az alábbi fejezetben összegezném a dolgozat során kitűzött célokat, és azoknak a megvalósítását. A szakdolgozat elején meghatároztam, hogy egy asztali számítógépen alapuló, percdíjjas, nyilvános internet terminál megoldást kell megvalósítanom.

Ezt követően átvizsgáltam több, már létező megoldást, melyek közül a Sitekiosk szoftvercsomaggal kapcsolatban már jelentős, évekre visszanyúló tapasztalatom volt. Az összes lehetséges alternatíva rendelkezett olyan negatívumokkal, mely nem tette ideálissá az általam vázolt probléma megoldására. Az itt összegyűjtött információk összegzése után részletesen kidolgoztam az elvárt rendszer specifikációit.

Az itt leírtakat végig szem előtt tartva összeállítottam a rendszertervet, melyet a lehetőségekhez mérten igyekeztem követni a megvalósítás során. Az implementálás során felmerülő problémák sikeres elhárítása után tesztelésnek vetettem alá az elkészült infrastruktúrát.

A tesztek eredményeinek összegzése, és az ezek során felmerülő hiányosságok korrigálása után áttekintettem az elkészült munkát, és megállapítottam, hogy az megfelel a dolgozat elején támasztott követelményeknek, és rendelkezik azokkal a tulajdonságokkal, melyeknek a hiánya ellehetetlenítette az alternatív, már meglévő megoldások használatát. A dolgozat végén megvizsgáltam néhány későbbi fejlesztési lehetőséget, mely hasznára lehet, szerves részét képezheti a projektnek.

11.SUMMARY

To summarize, in this following paragraph I would like to go over the goals set during this thesis, and the ways I have managed to achieve them.

At the beginning of this document, I have determined that I must create a solution for a desktop pc based, public use internet terminal system, which can be accessed on a pay per minute basis.

After this declaration, I have reviewed several pre-existing solutions, one of them being the software called Sitekiosk, whereby I had multiple years of previous experience. All possible alternatives had drawbacks, that resulted in none of them being viable solutions to my problem.

Based on the information gathered in this paragraph, I have set out to create a detailed specification for the system to be created. I have created the design parameters for the infrastructure, while keeping the specifications in mind all the way through.

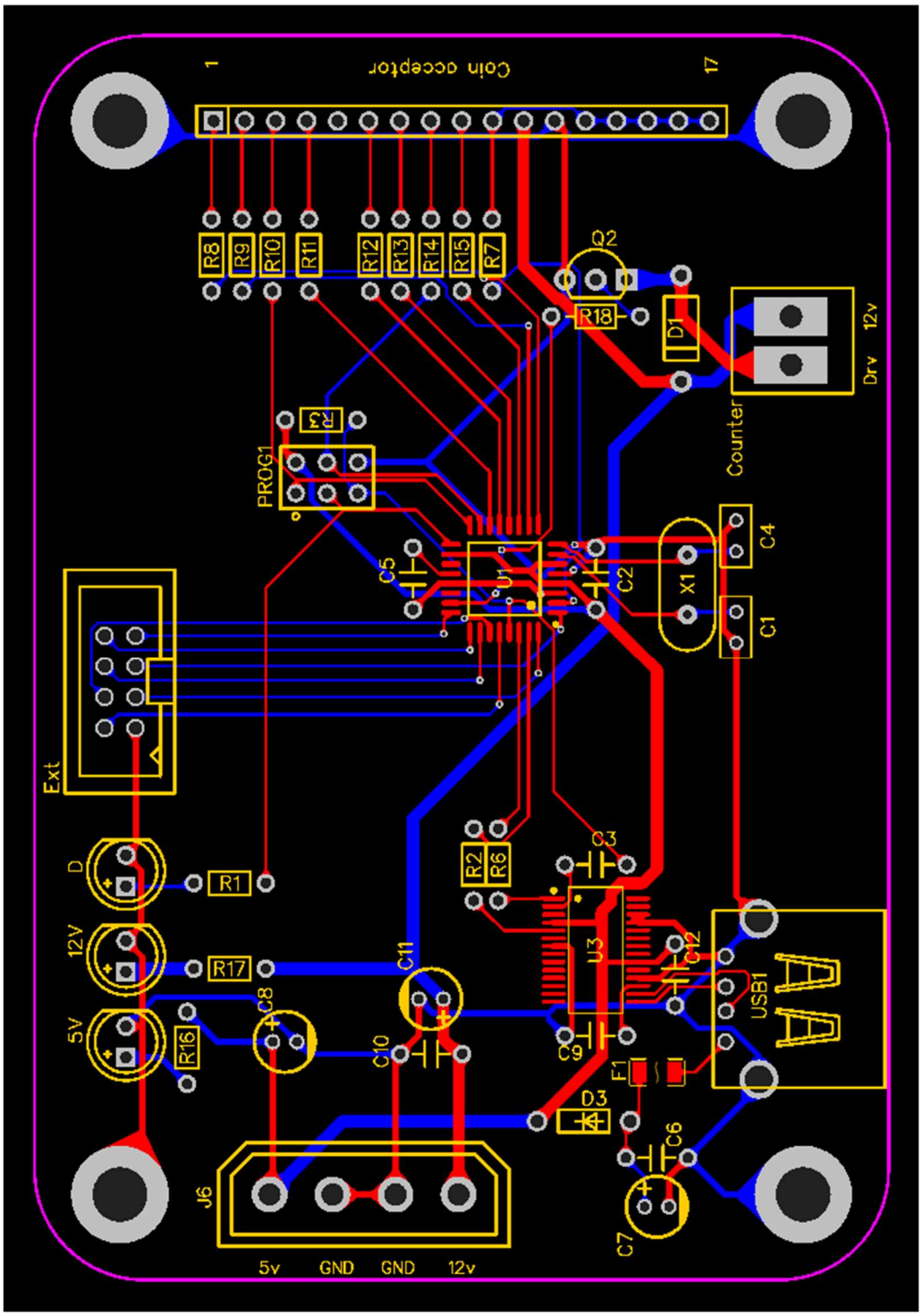
After successfully implementing my solution (and overcoming the obstacles present on the way), I have put the infrastructure though a series of tests.

After observing the results of these, and resolving the problems that have arisen, I have reviewed my work, and determined that it managed to meet the requirements detailed in the specifications, and it had the functions, properties, which were missing from the aforementioned alternatives.

As a closure, I have examined a few potential aspects of future upgradeability, which could be useful, organic part of the infrastructure.

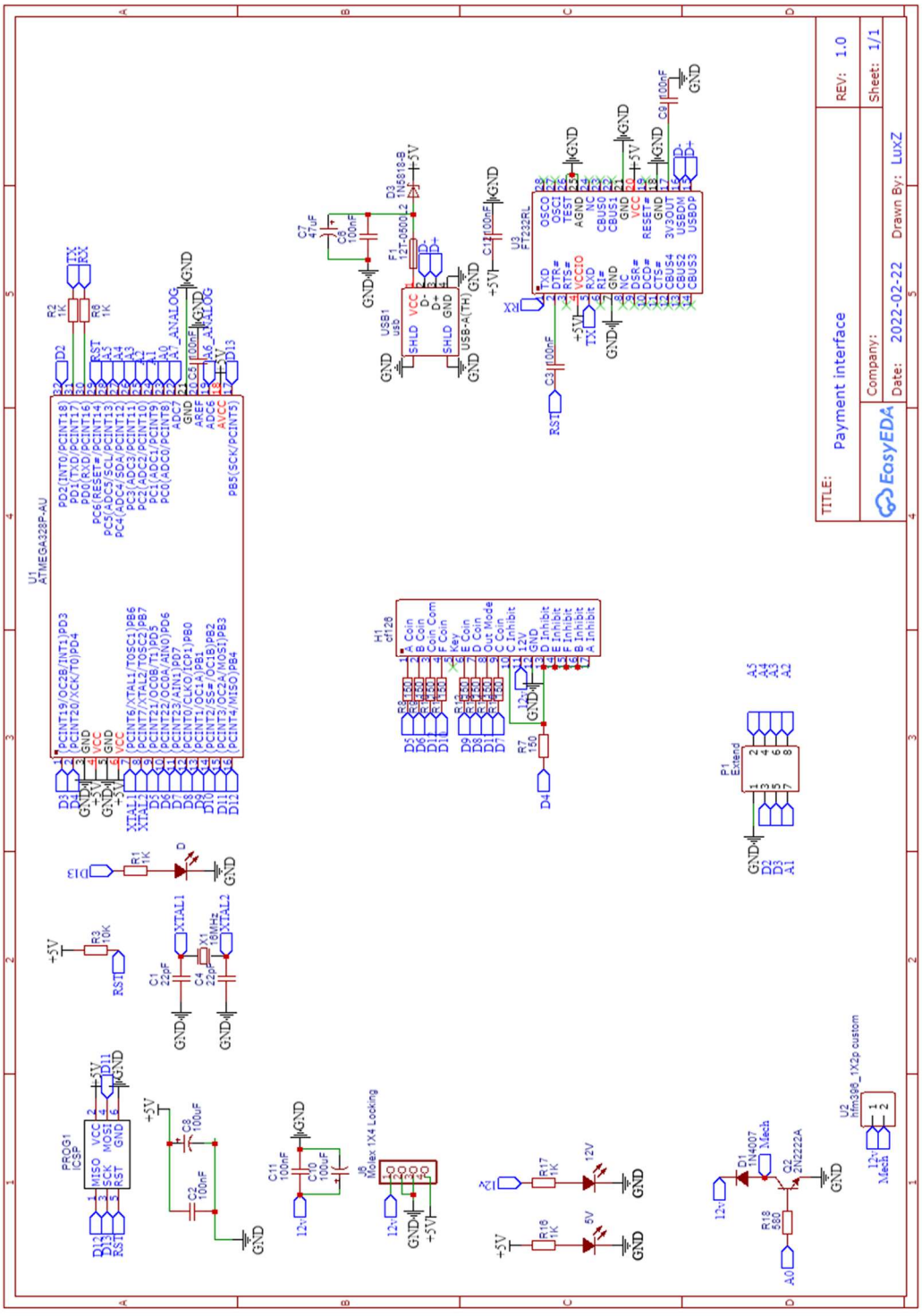
12.FÜGGELÉKEK

12.1. Függelék 1



12.1. ábra: Nyomtatott áramkör terv

12.2. Függelék 2



12.2. ábra: Kapcsolási rajz

13.IRODALOMJEGYZÉK

[1] (https://nrc.hu/news/internetpenetracio-2/), utoljára megtekintve: 2021.11.17.

[2] (https://hu.eyewated.com/internetes-kavezok-hogyan-talalhatunk-meg-egy-es-tippeket-a-hasznalatukhoz/), utoljára megtekintve: 2021.11.17.

[3] (https://caniuse.com/usage-table), utoljára megtekintve: 2021.11.17.

[4] (https://www.sitekiosk.com/wp-content/uploads/sitekiosk-en.pdf), utoljára megtekintve: 2021.11.17.

[5] (https://web.archive.org/web/20200920070011/https://www.provisio.com/web/us/features/payment-devices), utoljára megtekintve: 2021.11.17.

[6] (https://web.archive.org/web/20210302135330mp_/http://www.provisio.com/en-US/Downloads/VersionHistory.aspx), utoljára megtekintve: 2021.11.17.

[7] (https://www.sitekiosk.com/web/CustomerSupportCenter/ArticleDetails.aspx?ArticleID=25192), utoljára megtekintve: 2021.11.17.

[8] (https://www.sitekiosk.com/web/Shop/Shop.aspx?GroupIdx=1), utoljára megtekintve: 2021.11.25.

[9] (https://porteus-kiosk.org/), utoljára megtekintve: 2021.11.25.

[10] (https://porteus-kiosk.org/automatic-updates.html), utoljára megtekintve: 2021.11.25.

[11] (https://safeexambrowser.org/about_overview_en.html), utoljára megtekintve: 2021.11.25.

[12] (https://cefsharp.github.io/), utoljára megtekintve: 2021.11.27.

[13] (https://www.arduino.cc/en/uploads/Main/Arduino_Nano-Rev3.2-SCH.pdf), utoljára megtekintve: 2021.11.04.

[14] (https://www.analog.com/en/analog-dialogue/articles/uart-a-hardware-communication-protocol.html), utoljára megtekintve: 2021.12.04.

[15] (https://www.manualslib.com/manual/99203/Mars-Cashflow-126.html), utoljára megtekintve: 2021.12.04.

[16] (https://blogs.microsoft.com/latinx/2020/01/15/new-year-new-browser-the-new-microsoft-edge-is-out-of-preview-and-now-available-for-download), utoljára megtekintve: 2022.05.03.

[17] (https://docs.microsoft.com/en-us/windows/win32/winmsg/about-hooks), utoljára megtekintve: 2022.05.04.

[18] (https://stackoverflow.com/questions/3213606/how-to-suppress-task-switch-keys-winkey-alt-tab-alt-esc-ctrl-esc-using-low), utoljára megtekintve: 2022.05.04

[19] (https://github.com/randyrants/sharpkeys), utoljára megtekintve: 2022.05.04.

[20] (https://ww1.microchip.com/downloads/en/DeviceDoc/Atmel-7810-Automotive-Microcontrollers-ATmega328P_Datasheet.pdf), utoljára megtekintve: 2022.05.04.

[21] (https://docs.microsoft.com/en-us/microsoft-edge/webview2/concepts/distribution), utoljára megtekintve 2022.05.04.

[22] (https://github.com/binaryupdates/xLoader), utoljára megtekintve: 2022.05.04.

[23] (https://github.com/bitfieldlabs/afterglow/tree/master/afterglow_config), utoljára megtekintve: 2022.05.04.

14.ÁBRAJEGYZÉK

15.TÁBLAJEGYZÉK