

DUNAÚJVÁROSI EGYETEM



MÉRNÖKINFORMATIKUS BSC

SZAKDOLGOZAT

**ÁLLATKLINIKA TÁMOGATÓ ALKALMAZÁS ASP.NET
ALAPOKON**

Tóth Alex

mérnök informatikus jelölt

A-043-INF-2020

Kivonat

Dolgozatom célja, az állatklinikák számára egy szoftver fejlesztése webes alapokon, mely alkalmas az időpontok rögzítésére adott állatorvoshoz, műtőfoglalásra és adatok nyilvántartására.

Az elején felmértem az igényeket és meghatároztam az első verzió funkcióit és ezeket ismertettem. Ezután bemutattam a tervezett MVC alapú architektúrát és a fejlesztés technikai nehézségeit, megvalósítását különböző rétegekben. Miután az alkalmazás működőképes állapotba került, továbbiakban részleteztem az általam választott manuális tesztelést és ennek eredményességét. A szoftver működőképesnek nyilvánítása után készítettem egy szemléletes útmutatót az alkalmazás használatáról és egy üzemeltetői leírást a használatbavételi folyamatról. Végezetül összefoglaltam a jövőbeni terveket, fejlesztési javaslatokat az alkalmazással kapcsolatosan.

Abstract

The objective of my thesis is to develop a web based software for animal hospitals that is able to make appointments to veterinarian, to book surgery rooms and to registrar datas.

First, I surveyed the needs and specified the functions of the first version of the application and I described it. Than I presented the MVC based architecture and the difficulty techical parts at the different layers. After that I finished programming, I talked about the chosen maunal tests and about the result of it. When I declared it operational I made an illustrative guide for the users and wrote an operating manual. Last, I summerized the plans and developmental suggestions of the application.

Tartalomjegyzék

1. Bevezetés	1
2. Igényfelmérés, követelmények feltárása, specifikáció	2
2.1. Időpont- és műtőfoglaló	2
2.2. Nyilvántartás	3
2.3. A szoftver bevezetésének előnyei	4
3. Fejlesztői dokumentáció	6
3.1. Fejlesztői környezet, eszközök.....	6
3.2. Keretrendszer, felépítés	6
3.2.1. Adathozzáférési réteg	8
3.2.2. Üzleti logika réteg	14
3.2.3. Controller és View réteg.....	17
4. Tesztelés, hibajavítás	23
5. Felhasználói dokumentáció	31
5.1. Funkciói.....	31
5.1.1. Kliensek.....	32
5.1.2. Páciensek	33
5.1.3. Doktorok, Szobák.....	35
5.1.4. Kezelés típusa.....	36
5.1.5. Időpontfoglalás	36
5.1.6. Műtőfoglalás.....	39
5.1.7. Nyilvántartás – Kategóriák és Termékek	40
6. Üzemeltetői dokumentáció	42
6.1. Rendszerkövetelmények.....	42
6.2. Adatbázis telepítése.....	43
6.3. Alkalmazás telepítése	43
7. Tapasztalatok, fejlesztési lehetőségek, irányok	45

8. Összegzés.....	47
Irodalomjegyzék	48
Ábrajegyzék	49
Táblázatjegyzék	50
Mellékletek jegyzéke	51
1. melléklet: A program tervezett osztálydiagramja.....	1

1. Bevezetés

A mai világban nagyon sok ember tart házi kedvencet vagy haszonállatot, emiatt egyre több állatorvosra és klinikára van szükség a megfelelő ellátás érdekében. Az orvosok és asszisztensek számára, hogy ne az adminisztrációs feladatok vegyék el az idejük jelentős részét, kell egy megfelelő szoftver, ahol gyorsabban, egyszerűbben el tudják végezni a dokumentációt. Ez ihlette meg szakdolgozatom témáját, egy állatklinika működését támogató alkalmazás fejlesztését, mely az ÉleterősEB Manager nevet viseli.

Az alkalmazás célja, hogy kiváltson minden olyan nem korszerű adminisztrációs tevékenységet, - mint például Excel-táblázatban való nyilvántartást vagy az időpontok foglalását, feljegyzését naptárban vagy egyéb más módon - ezzel megkönnyítve a mindennapos munkát.

A szoftvert korszerű eszközök, módszerek használatával terveztem és valósítottam meg. Első körben feltártam a szoftverrel szemben állított követelményeket és elkészítettem a specifikációt. A fejlesztői dokumentációban, bemutatom a tervezés, fejlesztés menetét. A dolgozat folytatásaként pedig foglalkoztam a kész termék tesztelésével, esetlegesen felmerülő hibák javításával és a felhasználói dokumentáció elkészítésével. A szakdolgozat végén javaslatot teszek a továbbfejlesztési lehetőségekre, opciókra.

2. Igényfelmérés, követelmények feltárása, specifikáció

Új szoftver terméket fejleszték és megkérdeztem az állatorvost, akihez régóta járunk két kutyámmal, hogy milyen irányban fejlesszem a terméket azért, hogy még jobban megfeleljen az ügyféligényeknek

A kutatásból kiderült, hogy van igény egy egyszerűen kezelhető, átlátható és kevés töltési idővel rendelkező alkalmazásra, melyben nincsenek felesleges funkciók az állatklinika dolgozói számára.

Legfontosabb funkcióként az időpontfoglalás lehetőségét jelölték meg, mert a rendelőben nincs ilyen funkció jelenleg és nagy hasznát vennék. Nem kellene sokáig várakozni a rendelőben, ami időnként előfordul. Szükség lenne az eszközök, betegek, ügyfelek nyilvántartására. Az állatt nyilvántartásánál fontos lenne az eddigi kórelőzményeket látni, egy egységesített kórlap formájában. Utolsónak említették, hogy egy számlázási funkció is megkönnyítené a munkát, mivel jelenleg csak kézzel írt számlával tudnak szolgálni.

A fenti igények tudatában határoztam meg az ÉleterősEB Manager első kiadásának funkcióit, mely részben fedi a támasztott igényeket:

- Időpontfoglalás állatorvoshoz
- Műtő foglalása
- Nyilvántartás

Az alkalmazás ezen verziója kimondottan, csak a klinika dolgozóinak szól. A recepció fogja ellátni az adminisztrátori szerepkört is, de mindenki minden lehetőséghez hozzáfér.

Az 1. ábra mutatja be a program használati eseteit, melyet a következő két alponthan részletezek.

2.1.Időpont- és műtőfoglaló

Az alkalmazásnak két különböző foglaló egységet kell tartalmaznia, mindkét foglalás nézete naptár alapú kell, hogy legyen. Ez a típusú vizuális megjelenítés segíti azt, hogy ne legyen egyszerre két foglalás egyidőben ugyanahhoz az orvoshoz. Az időpontfoglaló résznél van lehetőség adott állatorvoshoz időpontot rögzíteni. Az időpont rögzítése több módon történhet. Az első opció, hogy az ügyfél személyesen megjelenik

vagy telefonál, hogy kedvencének szeretne időpontot vizsgálatra. A recepciós ezt fogadja. Megpróbálja kiválasztani az ügyfelet név alapján, ha ez nem sikerül, akkor rögzíteni szükséges a gazdit és a kedvencét is a „clients”, valamint a „patients” menüpont alatt. A gazda rögzítéséhez elkéri a teljes nevet, e-mail címet, telefonszámát, utána a kedvenc rögzítéséhez pedig a nevét, nemét, fajtát, fajtáját, hogy a sablonokat ki tudja tölteni. Rögzíti őket a rendszerben, ezután kiválasztja a gazdát, majd a hozzá tartozó kedvencet. Ezután a recepciós a naptárban keres szabad időpontot a vizsgálatra. A naptár nézet orvosokra van felosztva így lehetséges a preferált orvoshoz időpontot foglalnia az ügyfélnek. Miután megvan a kiválasztott időpont azt a recepciós rögzíti. Egy időpont egy címsort tárol, amiben a vizsgálat típusának kódja, az ügyfél neve és a kedvenc neve szerepel a gyors átláthatóság érdekében. Foglalásnál kiválaszthatónak kell lennie az előre rögzített vizsgálat típusnak, mely a kezdő időponthoz hozzáadva a vizsgálat idejét, meghatározza a konkrét időpont végét.

A műtőt foglaló rész fogja tárolni azokat az időpontokat, melyet az orvosok foglalnak le a beavatkozásokhoz. Egy műtőfoglalás csak a foglaló orvost és az időintervallumot tartalmazza, valamint a naptárban látszik, hogy melyik műtőbe szól a foglalás. Amennyiben vizsgálat közben megállapítják, hogy műtőre lesz szükség, időpontot egyeztet a műtőre, felviszi a rendszerbe az időpontot és lefoglalja a műtőt. Első körben mindig vizsgálati időpontot kell kérni, miután az orvos felmérte az állat egészségügyi állapotát az alapján ad újabb időpontot a műtőre, és ezt vagy ő vagy a recepciós felviszi a rendszerbe.

2.2.Nyilvántartás

A nyilvántartás résznél külön vettem azon adatok tárolását, melyeket a foglaláshoz szükségesek. Külön menüpontokban lehet létrehozni, törölni, szerkeszteni, valamint módosítani a műtőket, gazdákat, pácienseket, doktorokat és a vizsgálat típusát. Ezen kívül van egy általános nyilvántartás, ahol minden egyéb termék tárolható és kategóriákba sorolható. A terméknel még a következő opciók kitöltésére van lehetőség: termékneve, ára, mennyisége. Ezeket mind kötelező kitölteni. A kategóriák menedzseléséhez a kötelező név mezőn kívül még egy leírást adhatunk. Az ügyfelek adminisztrálásánál a név és a telefonszám mező kitöltése kötelező, az e-mail cím csak opcionális. A pácienseknél kötelező kitölteni a kedvenc nevét, a gazda megadása nem kötelező, mert lehet, hogy csak egy gazdátlan állatot kezelnek. A kötelezően kitöltendő mezők közé tartozik még az állat neme, faja, fajtája. A doktoroknál és a műtőszobáknál csak a név

mező megadása lehetséges, ezért ezek kitöltése kötelező. Az időpont típusának adminisztrálásánál mind a három mező kitöltése előírt, mert az alkalmazás az időpont kódját és az idő tartalmát használja máshol is.

Az adminisztrációkat mind az orvos, mind a recepció el tudja végezni. A termékek nyilvántartására két folyamat valósul meg. Az első esetben megérkeznek a megrendelések és a recepciós felviszi őket a rendszerbe, frissíti állapotukat. A második esetben pedig a doktor módosítja őket, hogy a vizsgálat során mit használt fel.

2.3.A szoftver bevezetésének előnyei

Pontos információkat ad, ha minden alkalmazott megfelelően rögzíti az adatokat

Használhatóság - A szoftver felhasználóbarát, a rendszer az átlátható felhasználói felületnek köszönhetően könnyen kezelhető. Az időpontfoglaló rendszernek a használatból érezhető, ezzel könnyebb menedzselni a vizsgálatokat és nem lesz kellemetlenül sok a várakozási idő. Ezáltal jobb az ügyfélművelés, az időpontot foglalt ügyfeleknek radikálisan csökken a várakozási idejük vagy meg is szűnik.

3. Fejlesztői dokumentáció

A program fejlesztésekor, tervezésekor kiemelt szempontok között szerepelt, hogy a program kompatibilis és könnyen újrafelhasználható maradjon. Ennek érdekében fontosnak tartok betartani legalább néhány alapelvet, mint:

- logikailag tagolt függvények használata,
- duplikáció kerülése,
- beszédes nevek használata.

Ezt figyelembe véve kerültek kiválasztásra a felhasznált eszközök.

3.1. Fejlesztői környezet, eszközök

Az alkalmazás fejlesztéséhez a Microsoft Visual Studio Community 2019-t használtam, amely egy ingyenes változata a termékcsaládnak és támogatja a használt nyelveket (C#, JavaScript). Választásom azért esett rá, mivel ez a környezet a legszélesebb körben elterjed a C# programozási nyelv a fejlesztők körében, és minden használt funkciót tartalmaz, ami a fejlesztéshez szükséges, ha nem is natív, de külön telepíthető „NuGet Package”-ek formájában. Adatbázisnak a fejlesztés idejére pedig a MS SQL Express változatát használom. A verzió előnye, hogy ingyenes és ideális kisebb server, asztali és webalkalmazásokhoz. Természetesen a végleges adatbázis könnyen költöztethető, változtatható az Entity Framework Core-nak köszönhetően, ami .NET Core alkalmazással kompatibilis. Lényege, hogy nem kell saját SQL parancsokat írni, hanem közvetlen az objektumokkal lépünk interakcióba adatmanipuláció céljából.

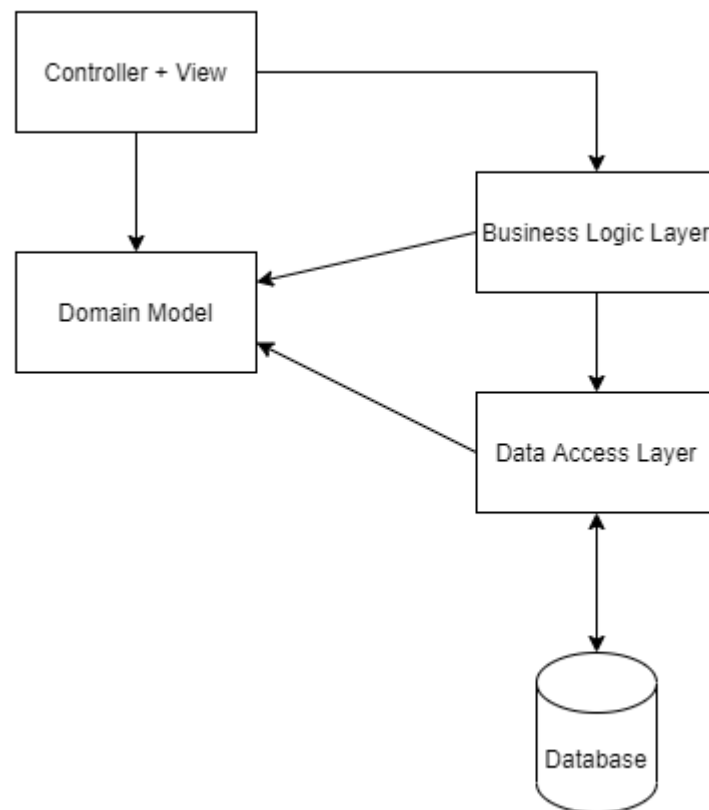
3.2. Keretrendszer, felépítés

A program az ASP.NET Core fejlesztők körében széleskörűen alkalmazott Model-View-Controller (továbbiakban: MVC) programtervezési minta alapján készült. Előnye abban rejlik, hogy mindenért külön réteg felel, ezért sokkal könnyebben tesztelhetők és javíthatók a hibák, ha mindennek csak egy feladata van. [1]

A program az ASP.NET Core 3.1 keretrendszerre épül. A Microsoft fejlesztette és támogatja, nyíltforráskódú, könnyű, gyors, multiplatform és többfajta hostolási lehetőséggel bír például. ISS, Docker, Apache, Nginx és így tovább.

A szoftver felépítését az 2. ábra szemlélteti. Legalul helyezkedik el az adatbázis, itt tárolódik minden információ, ezzel a réteggel van kétirányú kapcsolatban az

adathozzáférési réteg. Fő feladata, hogy kinyerjen, létrehozzon vagy módosítson adatokat a DB-ben és azt az Üzleti logikai réteg elérje. Ebben a rétegben található a program „lelke”, itt határozzuk meg a programunk funkcióit, amik szükségesek, ezért nagyon fontos, hogy az itt található logikákat alaposan teszteljük. Erre alkalmasak az úgynevezett funkcionális tesztek. Legfelül található a Controller + View réteg, amely az Üzleti logikával van kapcsolatban. A Controller felel a felhasználói interakciókért, a View felel a kívánt tartalom prezentálásáért a felhasználónak. Nagyon kevés logikát tartalmaz, az is főként a megjelenítésért felelős. Mindhárom utóbb említett réteg kapcsolatban áll a domain model réteggel. Ez írja le osztályok formájában az összes modellnek, hogy milyen viselkedése, adatai vannak az adott kontextusban.



2. ábra: A szoftver architektúra modellje

Az alkalmazás tervezésekor elkészült osztálydiagram az 1. számú mellékletben tekinthető meg, ábrázolja az osztályokat, rétegeket és a köztük lévő kapcsolatokat. Ebből kiindulva fejlesztettem le a rendszert. A rétegeket és az azokhoz tartozó osztályokat az alfejezetekben fejtem ki.

3.2.1. Adathozzáférési réteg

Az alkalmazás ezen rétege felel az úgynevezett CRUD (Create-Read-Update-Delete) műveletéért, azaz adatok létrehozásáért, olvasásáért, módosításáért és törléséért az adatbázisban. Az üzleti logikai rétegtől kapja az információt, hogy éppen mi a teendője melyik adattal.

Ebben a rétegben kap szerepet az Entity Framework Core. Egy ORM (Object-Relational Mapping) keretrendszer. Segítségével nincs szükség külön SQL parancsok írására, hanem ő maga elvégzi ezt helyettünk. A 3. ábrán látható adatbázist is ő hozza létre, felhasználva a domain modelleket, melyek a kontextusnak megfelelően leírják, hogy az állatklinikának a szoftver jelenlegi állapotában milyen igényeket kell kielégítenie. Az 1. táblázat foglalja össze a modelleket és tulajdonságait.

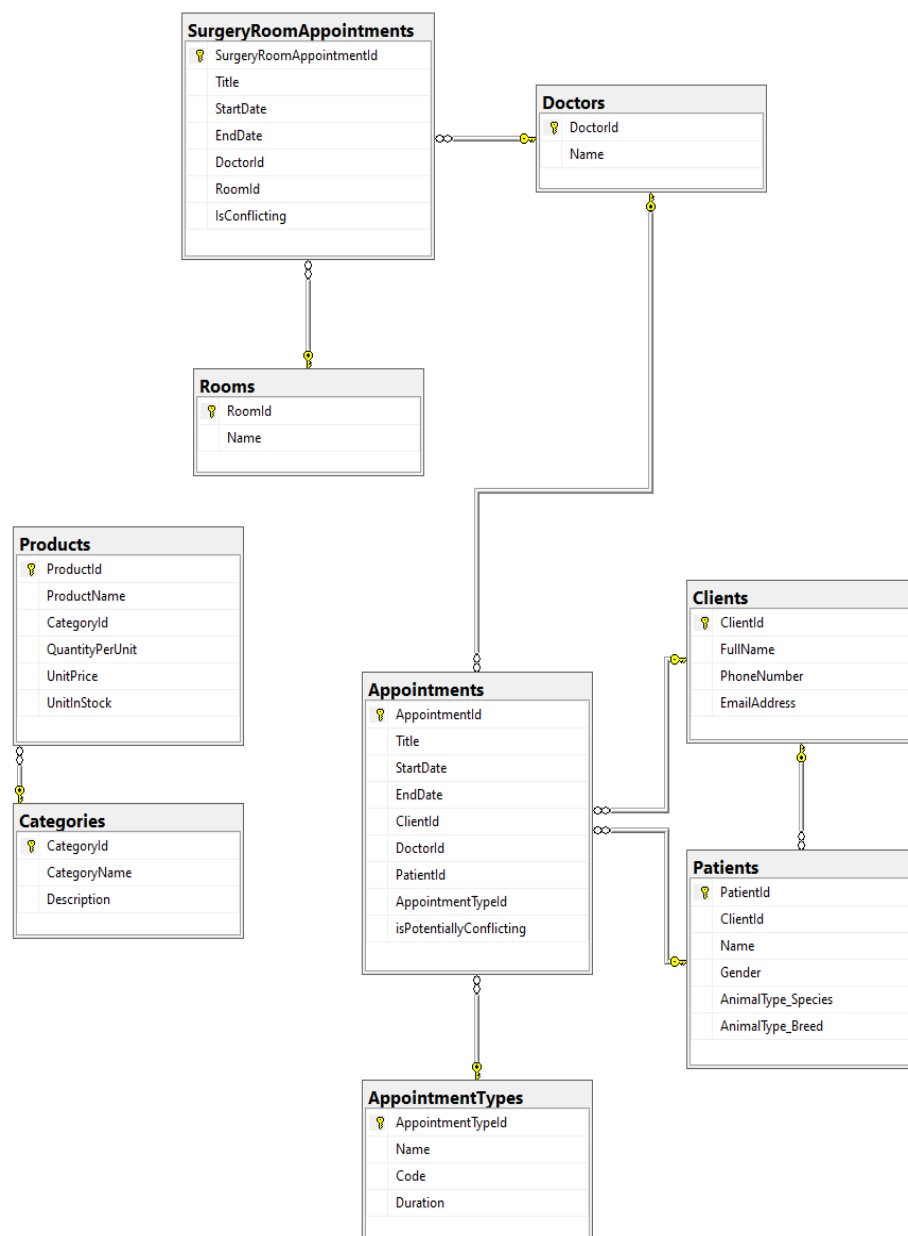
1. táblázat: A program modelljei és azok tulajdonságai

Modell név	Tulajdonságok
Appointment - időpont egyeztetés	<ul style="list-style-type: none">• AppointmentId - egyedi azonosító• ClientId• DoctorId• AppointmentTypeId – Egyedi azonosító alapján hivatkozás a időpont típusára• Title - Címe, rövid leírása a foglalásnak• StartDate – Az időpont kezdő dátuma• EndDate – Az időpont vég dátuma• IsPotentiallyConflicting – Igaz/Hamis értéket ábrázol, hogy ütközik e az időpont egy másikkal.
AppointmentType – időpont egyeztetés típusa	<ul style="list-style-type: none">• AppointmentTypeId - egyedi azonosító• Name – Az időpont fajtájának megnevezése

	<ul style="list-style-type: none"> • Code – Az időpont fajtájának kódja • Duration – Az időpont fajtájának a hossza percben.
Client – Gazda, ügyfél	<ul style="list-style-type: none"> • ClientId - egyedi azonosító • FullName – A gazda teljes neve • PhoneNumber – Telefonszáma • EmailAddress – Email címe
Doctor - Orvos	<ul style="list-style-type: none"> • DoctorId - egyedi azonosító • Name – A doktor neve
Patient – Páciens (kedvencek)	<ul style="list-style-type: none"> • PatientId - egyedi azonosító • ClientId – Hivatkozás egyedi azonosító lapján az állat gazdájára • Name – A kedvenc neve • Gender – A kedvenc neme • AnimalTypeSpecies – A kedvenc faja • AnimalTypeBreed – A kedvenc fajtája
Room - szoba	<ul style="list-style-type: none"> • RoomId – egyedi azonosító • Name – a szoba megnevezése
SurgeryRoomAppointment– Műtő foglалás	<ul style="list-style-type: none"> • SurgeryRoomAppointmentId – egyedi azonosító • Title – Címe, rövid leírása a foglalásnak • StartDate – A foglalás kezdeti dátuma • EndDate – A foglalás végének dátuma

	<ul style="list-style-type: none"> • DoctorId – A doktor egyedi azonosítója, idegen kulcsként való hivatkozáshoz • RoomId – A műtőszoba egyedi azonosítója, idegen kulcsként való hivatkozáshoz • IsConflicting – Igaz/Hamis értéket ábrázol, hogy ütközik e az időpont egy másikkal
Category – Kategória	<ul style="list-style-type: none"> • CategoryId • CategoryName • Description
Product - Termék	<ul style="list-style-type: none"> • ProductId – a termék egyedi azonosítója • ProductName – a termék megnevezése • CategoryId – a kategória egyedi azonosítója, idegen kulcsként való hivatkozáshoz • QuantityPerUnit – egy egységhez tartozó mennyiséget jelzi • UnitPrice – egy egység árát jelzi • UnitInStock – megmutatja hány egység van raktáron

Az EF Fluent API-jával – Application Programming Interface - írjuk le a szabályokat úgy mint, hogy mi legyen a tábla elsődleges kulcsa, milyen kapcsolatban állnak a táblák egymással, meghatározhatjuk, hogy pontosan mik legyenek SQL-ben az oszlopok típusai. Továbbá itt határozzuk meg, hogy egy tulajdonság, mint mondjuk egy Cím mező maximum hány darab karaktert tartalmazhat. Ezeket a szabályokat az EleterosEBContext osztály tartalmazza.



3. ábra Az adatbázis diagramja

Ebben a rétegben alkalmaztam a Repository[2] és Unit of Work[3] programozási mintákat. A repository programozási minta az adatok absztrakcióját biztosítja, hogy az alkalmazás képes legyen egy egyszerű absztrakcióval dolgozni, interfészeket használva a kollekciókkal való munkához. Az elemek hozzáadása, eltávolítása, frissítése és kijelölése a kollekcióból egyszerű módszerek sorozatával történik, anélkül, hogy olyan adatbázis-problémákkal kellene foglalkozni, mint a kapcsolatok, parancsok, kurzorok vagy olvasók. A minta használatával könnyen elérjük a „loose coupling” -ot.

Megvalósításához létrehoztam egy általános interfészt IRepository néven, mellyel meghatározom milyen feladatokat kell tudnia mindegyik reponak.

```
public interface IRepository<TEntity>
{
    void Add(TEntity entity);
    void Update(TEntity entity);
    void Delete(TEntity entity);
    Task<TEntity> GetByIdAsync(int id);
    Task<IReadOnlyList<TEntity>> ListAsync(params Expression<Func<TEntity,
object>>[] includes);
}
```

Majd létrehoztam egy BaseRepository osztályt, amelybe dependency injection¹ segítségével használom a EF DbContext-et és implementálom az alap függvényeket, amiket meghatároztam az interfészben. A függvények közül a ListAsync a legérdekesebb, mellyel egy táblát lehet lekérni. Ha egy Expressiont kap paraméterként (például ProductRepository.ListAsync(p => p.Category)), akkor a termékekhez betölti a kategóriát, különben nem tenné meg a lazy loading miatt.

```
public abstract class BaseGenericRepository<T>
    : IRepository<T> where T : class
{
    private readonly EleterosEBContext _context;

    protected BaseGenericRepository(EleterosEBContext context)
    {
        _context = context;
    }

    public virtual void Add(T entity)
    {
        _context.Add(entity);
    }

    public virtual void Update(T entity)
    {
        var entry = _context.Entry(entity);
        if (entry.State == EntityState.Detached)
        {
            _context.Set<T>().Attach(entity);
            _context.Entry(entity).State = EntityState.Modified;
        }
        else
        {
            _context.Entry(entity).CurrentValues.SetValues(entity);
        }
    }
}
```

¹ A Dependency Injection egy olyan programozási technika, amivel objektumok függőségeit lehet kielégíteni.[4]

```

    }

    public virtual void Delete(T entity)
    {
        _context.Remove(entity);
    }

    public virtual async Task<T> GetByIdAsync(int id)
    {
        return await _context.FindAsync<T>(id);
    }

    public virtual async Task<IReadOnlyList<T>> ListAsync(params
Expression<Func<T, object>>[] includes)
    {
        IQueryable<T> result = _context.Set<T>();
        foreach (var include in includes)
        {
            result = result.Include(include);
        }
        return await result.ToListAsync();
    }
}

```

Ezt az alap interfészt implementálja minden reponak a saját interfésze és a táblákhoz tartozó repository-k valósítsák meg a hozzájuk tartozó interfészt, valamint a BaseRepository osztályt. Lentebb látható példaként a kedvencek táblázatot kezelők osztálya, a többi is a minta alapján készült:

```

public class PatientRepository: BaseGenericRepository<Patient>, IPatientRepository
{
    public PatientRepository(EleterosEBContext context: base(context, logger)
    {}
}

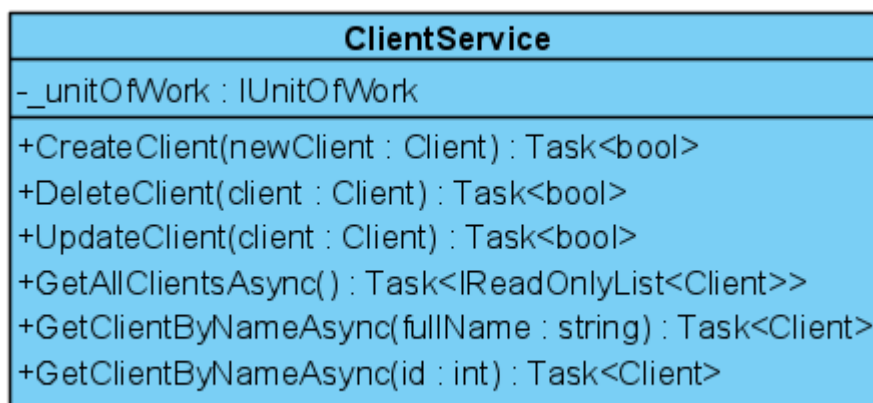
```

Az így elkészült repository-kat használjuk fel a Unit Of Work minta osztályában. A minta lényege, hogy több SQL műveletet egyetlen egy tranzakcióban kezelünk. A repository-kat publikusan és a DbContextet privátként hozzáadjuk a dependency injection segítségével a konstruktorban és implementáljuk a függvényt, ami ment és elküldi a tranzakciót az adatbázis felé. Így a következő üzleti logikai rétegben elegendő ennek az egy osztálynak a publikus interfészét használni, minden más rejtve marad.

3.2.2. Üzleti logika réteg

A réteg feladata az előre specifikált üzleti folyamatok futtatása. Ebben a logikai rétegben helyezkednek el azok a megoldandó problémák melyekre a program készült. Az itt lévő osztályok döntenek arról, hogy milyen adatra van szükség, amit megkapnak az adathozzáférési rétegtől, esetleg továbbítanak feléjük. Szükség szerint az információk manipulálása is itt történik. Az alkalmazásban a logika 9 osztályra van szétbontva mindegyik egy-egy szolgáltatást lát el.

A „ClientService” osztály (lásd 4. ábra) látja el az ügyfél adataival kapcsolatos feladatokat aszinkron módon. Van egy privát attribútuma unitOfWork, melynek típusa IUnitOfWork interfész, így olyan osztálynak kell lennie, ami ezt megvalósítja. Jelen esetünkben ez a UnitOfWork osztály lesz az adathozzáférési rétegben. Ezt a dependency injection segítségével kapja meg a service. Így a privát _unitOfWork-ön keresztül hozzáférhetünk az adatbázisban szereplő adatokhoz anélkül, hogy tudnunk kellene ez miképpen valósul meg.



4. ábra A ClientService osztály diagramja

Összesen hat publikus metódusa van, melyek CRUD alapfunkciókat látnak el.

CreateClient felelős az új ügyfél létrehozásáért. Paraméterként vár egy Client típust, ez modellezi az ügyfelet és a műveletek elvégzése után visszatér egy igaz/hamis (bool) értékkel, hogy sikerült e végrehajtani a műveletet. A DeleteClient (ügyfél törlése) és az UpdateClient (ügyfél frissítése) metódus is ugyan úgy működik, vár egy Client típust paraméterként és visszaad egy igaz/hamis értéket a művelet sikerességéről.

A GetAllClientsAsync egy olvasási függvény, paraméterként nem vár semmit és visszaadja egy olyan listában a klienseket, melyek csak olvashatóak. Paraméterként vár

egy szöveg típust, ami az ügyfél teljes neve és ez alapján visszaadja az első találatot. A `GetClientByNameAsync` pedig paraméterként megkap egy integer típusú azonosítót, ez alapján kikeresi az azonosítóhoz tartozó klienst és visszaadja azt.

A többi osztály is ugyanezekkel a logikával rendelkezik, ahogy a 6. ábra is mutatja. Mindegyiknek megvan a saját maga `Create-Delete-Update` függvénye, ahol vár a szolgáltatásnak megfelelő típust paraméterként és visszaad egy igaz/hamis értéket a feladat végrehajtásának sikerességéről. Szintén rendelkeznek egy olyan metódussal, mely csak egy olvasható listát ad vissza a konkrét típussal. Két olyan függvénnyel, amikor szöveget vár paraméterként név/cím vagy egy egész típusú azonosítót és ez alapján visszaadja a legelső találatot a típusból.

Az `AppointmentService` osztály (5. ábra) a fentiekén kívül, még tartalmaz privát függvényeket, melyek a program más részei számára nem láthatóak. Ezek rendre a `GetAllAppointmentWithSpecificStartingDay` és `DateIsConflicting`. Az első paraméterként kap egy `DateTime` típust és végeredményképpen visszaad egy listát az összes olyan időponttal, mely a paraméterként megadott dátum napjával kezdődik. Az utóbbi pedig paramétereként vár egy `DateTime` és egy `Appointment` típust. Kiszámolja, hogy a megadott dátum már ütközik-e az időpont dátumával és ezt igaz/hamis formájában visszaadja. Ezt a két megoldást a `CreateAppointment` függvénynél használom, hogy beállítsam az „`IsPotentiallyConflicting`” tulajdonságát az `Appointment`nek, annak megfelelően, hogy van vagy nincs ütközés.

AppointmentService
- _unitOfWork : IUnitOfWork
+CreateAppointment(newAppointmentter : Appointment) : Task<bool>
+DeleteAppointment(appointment : Appointment) : Task<bool>
+UpdateAppointment(appointment : Appointment) : Task<bool>
+GetAllAppointmentsAsync() : Task<IReadOnlyList<Appointment>>
+GetAppointmentByIdAsync(id : int) : Task<Appointment>
-DateIsConflicting(targetDateTime : DateTime, appointment : Appointment) : bool
-GetAllAppointmentWithSpecificStartingDay(startDateTime : DateTime) : Task<IEnumerable<Appointment>>

5. ábra: Az `AppointmentService` osztály

RoomService
- _unitOfWork : IUnitOfWork
+CreateRoom(newRoom : Room) : Task<bool> +DeleteRoom(room : Room) : Task<bool> +UpdateRoom(room : Room) : Task<bool> +GetAllRoomsAsync() : Task<IReadOnlyList<Room>> +GetRoomByNameAsync(name : string) : Task<Room> +GetRoomByIdAsync(id : int) : Task<Room>

PatientService
- _unitOfWork : IUnitOfWork
+CreatePatient(newPatient : Patient) : Task<bool> +DeletePatient(patient : Patient) : Task<bool> +UpdatePatient(patient : Patient) : Task<bool> +GetAllPatientsAsync() : Task<IReadOnlyList<Patient>> +GetPatientByNameAsync(name : string) : Task<Patient> +GetPatientsByIdAsync(id : int) : Task<Patient>

ProductService
- _unitOfWork : IUnitOfWork
+CreateProduct(newproduct : Product) : Task<bool> +DeleteProduct(product : Product) : Task<bool> +UpdateProduct(product : Product) : Task<bool> +GetAllProductsAsync() : Task<IReadOnlyList<Product>> +GetProductByIdAsync(id : int) : Task<Product> +GetProductByNameAsync(name : string) : Task<Product>

CategoryService
- _unitOfWork : IUnitOfWork
+CreateCategory(newCategory : Category) : Task<bool> +DeleteCategory(category : Category) : Task<bool> +UpdateCategory(category : Category) : Task<bool> +GetAllCategoriesAsync() : Task<IReadOnlyList<Category>> +GetCategoryByIdAsync(id : int) : Task<Category> +GetCategoryByNameAsync(categoryName : string) : Task<Category>

AppointmentTypeService
- _unitOfWork : IUnitOfWork
+CreateAppointmentType(newAppointmentType : AppointmentType) : Task<bool> +DeleteAppointmentType(appointmentType : AppointmentType) : Task<bool> +UpdateAppointmentType(appointmentType : AppointmentType) : Task<bool> +GetAllAppointmentTypesAsync() : Task<IReadOnlyList<AppointmentType>> +GetAppointmentTypeByIdAsync(id : int) : Task<AppointmentType> +GetAppointmentTypeByNameAsync(name : string) : Task<AppointmentType>

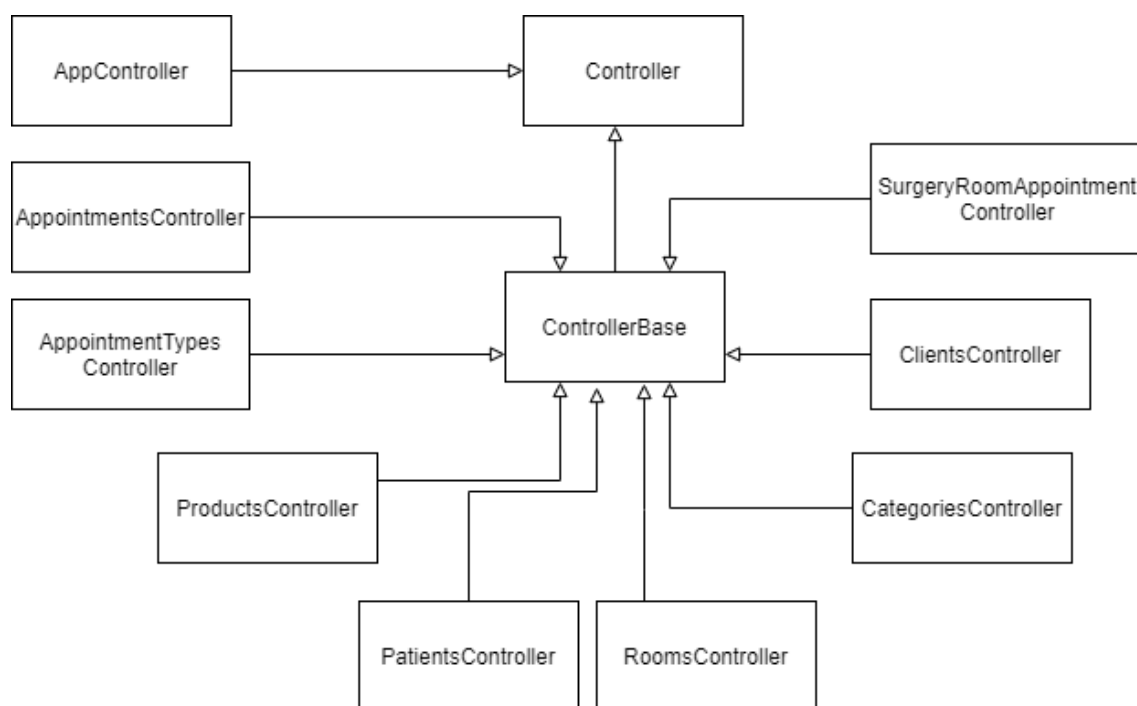
SurgeryAppointmentService
- _unitOfWork : IUnitOfWork
+CreateSurgeryRoomAppointment(newSurgeryRoomAppointment : SurgeryRoomAppointment) : Task<bool> +DeleteSurgeryRoomAppointment(surgeryRoomAppointment : SurgeryRoomAppointment) : Task<bool> +UpdateSurgeryRoomAppointment(surgeryRoomAppointment : SurgeryRoomAppointment) : Task<bool> +GetAllSurgeryRoomAppointmentAsync() : Task<IReadOnlyList<SurgeryRoomAppointment>> +GetSurgeryRoomAppointmentByTitleAsync(title : string) : Task<SurgeryRoomAppointment> +GetSurgeryRoomAppointmentByIdAsync(id : int) : Task<SurgeryRoomAppointment>

6. ábra: A Room-, Patient-, Product-, Category-, AppointmentType- és SurgeryAppointmentService osztályok

3.2.3. Controller és View réteg

A Controller réteg felelős az alkalmazás vezérléséért. Ezt a feladatát úgy oldja meg, hogy fogadja a nézetről érkező eseményeket. Ezeket feldolgozza és meghívja az üzleti logikai réteg megfelelő metódusait. Az így kapott eredményt továbbadja a nézet számára. A controllerek a Controllers mappában a nézetek pedig a Views mappában találhatóak az EleterosEB.Web projekten belül.

Ahogy a 7. ábrán is jól látszik, az alkalmazás minden elérhető funkciójáért külön controller felelős, melyek az alap gyári ControllerBase, valamint Controller osztályokból származnak. Azok a controllerek származnak a ControllerBase őssztályból, melyek API endpointként szolgálnak. A Controller osztályból csak az ApplicationController származik, mert ő Viewkat is létrehoz.



7. ábra: A controller osztályai

Az ApplicationController osztály csak simán a navigációért és a nézet legenerálásáért felelős, amikor betöltjük a programot innen hívódik meg, generálódik le például az Index nézet is, valamint lehet tovább navigálni a többi nézetre.

Az összes többi controller API endpointként szolgál és JSON – JavaScript Object Notation - formátumban várják és küldik az adatokat. Mindegyikben megtalálható két GET-es lekérdezés, egyikkel listázzuk az összes lekért adott adattípust, míg a másik vár

paraméterként egy id-t, melynek megadásával csak azzal az egy objektummal végezhetünk műveleteket. Tartalmaz még POST-tal működő végpontot is, amely létrehoz egy teljesen új objektumot, valamint PUT-ot és DELETE-t is. Az utóbbi paraméterként vár egy id-t, még az előbbi pluszban a modell adatait is, hogy tudja melyik objektummal mit kell végrehajtani. Az első frissíti, míg az utóbbi törli az objektumot. Mindegyik végpont ugyanezen az elven működik, a controller dependency injection segítségével megkapja az üzleti logikát, ezáltal privát adattagon keresztül el tudja érni a szolgáltatásokat. Ezen kívül még ugyanezzel a módszerrel kap egy linkgenerátort és mappert, amely egyik objektumot mappeli a másikra. Ez utóbbi egy külön letölthető NuGet csomag, neve AutoMapper. Ezzel a domain modelleket mappeljuk az ebben a rétegben található Data Transfer Object-ekkel (továbbiakban Dto). Ezt azért használjuk, mert nem feltétlen van szükségünk minden adattagjára a domain modellnek, és nem akarunk felesleges adatot küldeni vagy kapni.

Az említett ApplicationController által legenerált View-k felelősek az alkalmazás adatainak megjelenítéséért és a felhasználóval való kapcsolattartásért. A nézetek a Views mappában találhatóak meg. Van egy darab közös nézet is a Shared mappában, ez adja meg a keretét a nézeteknek. A megjelenítésre sima html, css, és javascriptet használtam kendoUI keretrendszerrel, kivéve a közös _Layout.cshtml-nél, ahol Razor markup nyelvet is használtam. Utóbbira azért volt csak szükség, hogy a menüben a linkeket maga generálja a nézetekhez, például:

```
@Html.ActionLink("Patients", "Patients", "App", new { area = "" }, null) -ből lesz  
<a href="/App/Patients">Patients</a>
```

A @Renderbody()-ra azért van szükség, hogy a többi nézetet a keretnek ebbe a részébe töltsse be. Itt vannak meghivatkozva a szükséges JavaScript könyvtárak is, melyeket a megjelenítés használ. A közös felületen kapott helyet a bal felső sarokban megtalálható ügyfél-beteg beválasztó is, melynek kódja a /wwwroot/js/header.js fájlban található meg.

Az adatok nyilvántartásáért felel a legtöbb nézet:

- AppointmentTypes.cshtml
- Categories.cshtml
- Products.cshtml
- Clients.cshtml
- Doctors.cshtml

- Patients.cshtml
- Rooms.cshtml

Ezekben kap fontos szerepet a KendoUI keretrendszer. Mindegyiknek az alapját a következő javascript kód adja:

```
$(function () {
    const serviceUrl = "/api/categories";
    var dataSource = new kendo.data.DataSource({
        type: "json",
        transport: {
            read: {
                url: serviceUrl,
                contentType: "application/json"
            },
            update: {
                url: function (category) { return serviceUrl + "/" +
category.categoryId },
                type: "PUT",
                contentType: "application/json"
            },
            destroy: {
                url: function (category) { return serviceUrl + "/" +
category.categoryId},
                type: "DELETE",
                contentType: "application/json"
            },
            create: {
                url: serviceUrl,
                type: "POST",
                contentType: "application/json"
            },
            parameterMap: function (options, operation) {
                return JSON.stringify(options);
            }
        },
        pageSize: 15,
        schema: {
            model: {
                id: "categoryId",
                fields: {
                    categoryId: { type: "number", editable: false },
                    categoryName: { editable: true },
                    description: { editable: true },
                }
            }
        }
    });

    $("#grid").kendoGrid({
        dataSource: dataSource,
```



```

        editable: "inline",
        toolbar: ["create"],
        columns: [
            {
                field: "categoryName",
                title: "Name",
                filterable: false
            },
            {
                field: "description",
                title: "Description",
                filterable: false
            },
            {
                command: ["edit", "destroy"], title: "&nbsp;", width: "210px"
            }
        ],
        pageable: true
    });
});

```

Egy konstans változóban megadtam az API végpontnak az elérési címét, és létrehoztam egy `dataSource` nevű változót, amely az adatok CRUD műveleteiért felel. Itt adtam meg azt is, hogy milyen típusú adatokat várunk, jelen esetben JSON formátumban. A `transport` résznél – külön véve az olvasási, frissítési, törlési és létrehozási műveleteket – jelölve lettek a művelethez tartozó url-ek, a http verb típusa és a tartalom típusa. Majd a `pageSize`-nál lehetett megadni, hogy egy oldalra hány darab adatot jelenítsen meg. A `schema`-nál adjuk meg, hogy a modellünk az adatforrástól milyen néven kapja meg az azonosítót, valamint jelöljük a `fields`-nél, hogy milyen további mezőket akarunk szerepeltetni. A mezőknél, ha ugyanaz a neve mint a kapottnak, akkor nem szükséges jelölni semmit, de ha más nevet használunk, akkor a `from` kulcsszó után adható meg: `<mezőnév>: { from: " <kapott mezőnév>" }`.

A következő részben jQuery-vel kiválasztom azzal az id-vel rendelkező html taget, melyben szeretném, hogy megjelenjen a kendo keretrendszer grid szerkezetű megjelenítési formája. A beállításokban megadtam az előzőleg előállított adatforrást, a szerkesztés módját, milyen oszlopok szerepeljenek a rácsszerkezetben és hogy lapozható legyen.

Egyedül a betegek megjelenítésénél volt szükség plusz logikára, aminek a segítségével egy szám helyett a gazda neve jelenik meg. Ennek megoldásához a betegeknel létrehoztam plusz egy kezdetben üres változót, ami a gazdákat tartalmazza, és a gazdák API végpontján keresztül feltöltöttem adatokkal.

```
var clients = [];
$.getJSON("http://localhost:8888/api/clients", function (data) {
    clients = data;
    dataSource.fetch();
});
```

Ahhoz, hogy az ügyfél neve jelenjen meg az ID helyett, a kendoGrid szerkezet megadásánál az oszlopok résznél jelölni kell template-tel:

```
columns: [
    { field: "name", title: "Name" },
    {
        field: "clientId", width: "150px",
        editor: clientDropDownEditor,
        title: "Client",
        template: "#=getClientName(clientId)#"
    },...
]
```

A template-nél egy metódust hívunk meg, ami végrehajtja az id-szöveg cserét a következő kóddal:

```
function getClientName(clientId) {
    for (var idx = 0, length = clients.length; idx < length; idx++) {
        if (clients[idx].clientId === clientId) {
            return clients[idx].fullName;
        }
    }
}
```

A getClientName paraméterként megkapja a kliens azonosítóját. Végigmegy egy cikluson az előzőekben betöltött klienseken egyesével összehasonlítva az azonosító mezőket, ha egyezés van, akkor visszatér a kliens nevével. A külön betöltött kliensek összetett adatszerkezetek és tartalmazzák az id-kat valamint az ügyfelek teljes nevét is.

A másik két felhasználói felület naptár alapú időpontfoglalást tesz lehetővé, ennek megvalósítására a kendoUI Scheduler-jét használtam:

- Index.cshtml
- SurgeryRoomAppointment.cshtml

A Scheduler-nél is az előzőekben bemutatott adatforrást kell megadni. Itt megadtam, hogy a naptárban mettől-meddig mutassa az időket (jelen esetben 7 és 18 óra között), illetve milyen nézetek legyenek választhatóak. Be lett állítva napi, munkaheti és napirendi nézett, valamint az is, hogy egy óra hány egyenlő részre legyen elosztva jelen esetben ez 15 perces időintervallum. Meg kellett adni, hogy mi szerint csoportosítja a naptárt, mik legyenek az erőforrások az időponthoz (műtőszoba és/vagy orvos). Az „editable” mezőnél, egy template megadására is szükség volt, hogy egyedire lehessen

szabni, és csak a szükséges mezők legyenek megjelenítve szerkesztéskor vagy új foglalás esetén. Mivel sajnos a keretrendszer template része nem tudja az erőforrásokat hozzákötni a templatekhez, szükség volt plusz két függvényre.

```
function getDoctorResourceData(e) {
    getData(e, 1);
}
function getData(e, index) {
    var scheduler = $("#scheduler").data('kendoScheduler');
    var resourcesData = scheduler.resources[index].dataSource.view();
    e.success(resourcesData.toJSON());
}
```

A `getData` függvény visszaadja a naptárnak az `index`-edik helyen lévő erőforrását json formátumban (az erőforrások 0-tól számozódnak). Így a fenti példában szemléltetett `getDoctorResourceData` visszaadja nekünk az orvosokat. A következőkben megadott mintában nagy segítségemre volt, hogy a megjelenítés megfelelően működjön.

```
<script id="editor" type="text/x-kendo-template">
    # if(surgeryRoomAppointmentId) { #
    <h3>Edit Surgery Room Booking</h3>
    # } else { #
    <h3>Add Surgery Room Booking</h3>
    # } #
    <p>
        <label>Title: <input name="title" /></label>
    </p>
    <p>
        <label>Start: <input data-role="datetimepicker" name="start" /></label>
    </p>
    <p>
        <label>End: <input data-role="datetimepicker" name="end" /></label>
    </p>
    <p>
        <label for="doctorId">
            Doctor:
            <input id="doctorId" data-bind="value:doctorId" data-role="dropdownlist"
name="doctorId"
                data-value-field="doctorId" data-text-field="name"
                data-source="{
                    transport: {
                        read: getDoctorResourceData}}" />
        </label>
    </p>
</script>
```

4. Tesztelés, hibajavítás

A program tesztelését manuális módon hajtottam végre. Az első tesztelések az első API végpontok elkészültével történtek, a Postman nevezetű programmal. A Postman egy hasznos program API-k fejlesztésénél. Funkciókban bővelkedik és könnyen ellenőrizhetőek a végpontok a különböző http igékkel. A 2-10 táblázatok jól összefoglalják az egyes végponton elvégzett teszteket, melyek szükségesek a felhasználói felület számára. Ezek mind sikeresen zárultak. A Postman segítségével elküldtem a megfelelő http igével és a body-val a végpontokra a tesztkérdéseket, majd ezeket az adatbázisban visszaellenőriztem. A Postmanben is látható volt már a művelet sikeressége, mivel a végpontok úgy lettek kialakítva, hogy sikeresség esetén 200 vagy 201 egyes válasszal és az adatokkal térjen vissza. Ha nem megfelelő formátumban nem megfelelő adatot küldtünk a végpontnak, akkor kliens oldali hiba esetén 400-as, szerver oldali hiba esetén pedig 500-as hibakóddal ad választ. Amikor törölni vagy frissíteni szeretnénk akkor az id alapján keressük ki az adatot, viszont, ha ezt nem találja a végpont 404 Not Founddal válaszol.

2. táblázat: Az időpontfoglaló végpont tesztelése

Tesztelt végpont	Kérés típusa	Várt eredmény	Body	Sikeresség
/api/apointments/	GET	A végpontnak vissza kell adnia az összes időpontot.	Üres	✓
/api/apointments/{id}	PUT	Az {id}-t megadva egy adott időpontot frissítenie kell.	<pre>{ "appointmentId": 3, "title": "Tesztputttt", "startDate": "2020-11-05T09:00:00", "endDate": "2020-11-07T09:30:00", "clientId": 1, "doctorId": 1, "appointmentTypeId": 1, "patientId": 1, "isPotentiallyConflicting": false }</pre>	✓
/api/apointments/{id}	Delete	Az {id}-t megadva egy adott	Üres	✓

		időpontot törölnie kell.		
/api/appointments/	POST	Létre kell hoznia egy új időpontot.	{ "title": "Teszt", "startDate": "2020-11-05T09:00:00", "endDate": "2020-11-07T09:30:00", "clientId": 1, "doctorId": 1, "appointmentTypeId": 1, "patientId": 1 }	✓

3. táblázat Az időpont fajtájának végpontjának tesztelése

Tesztelt végpont	Kérés típusa	Várt eredmény	Body	Sikeresség
/api/appointmenttypes/	GET	A végpontnak vissza kell adnia az összes időpont típust.	Üres	✓
/api/appointmenttypes/{id}	PUT	Az {id}-t megadva egy adott időpont típust frissítenie kell.	{ "appointmentTypeId": 1, "name": "Alap vizsgálat", "code": "V-001", "duration": 20 }	✓
/api/appointmenttypes/{id}	Delete	Az {id}-t megadva egy adott időpont típust törölnie kell.	Üres	✓
/api/appointmenttypes/	POST	Létre kell hoznia egy új időpont típust.	{ "name": "Teszt vizsgálat", "code": "V-002", "duration": 30 }	✓

4. táblázat A műtő foglalás végpont tesztelése

Tesztelt végpont	Kérés típusa	Várt eredmény	Body	Sikeresség
/api/surgeryroomappointments/	GET	A végpontnak vissza kell adnia az összes műtőfoglalást.	Üres	✓

/api/surgeryroomappointments/{id}	PUT	Az {id}-t megadva egy adott műtőfoglalást frissítenie kell.	{ "surgeryRoomAppointmentId": 1, "title": " Teszt ", "startDate": "2020-11-04T10:00:00", "endDate": "2020-11-04T12:00:00", "doctorId": 1, "roomId": 1, "isConflicting": false }	✓
/api/surgeryroomappointments/{id}	Delete	Az {id}-t megadva egy adott műtőfoglalást törölnie kell.	Üres	✓
/api/surgeryroomappointments/	POST	Létre kell hoznia egy új időpontot.	{ "title": "Teszt", "startDate": "2020-11-04T10:00:00", "endDate": "2020-11-04T12:00:00", "doctorId": 1, "roomId": 1, "isConflicting": false }	✓

5. táblázat A szobák végpont ellenőrzése

Tesztelt végpont	Kérés típusa	Várt eredmény	Body	Sikeresség
/api/rooms/	GET	A végpontnak vissza kell adnia az összes műtő szobát.	Üres	✓
/api/rooms/{id}	PUT	Az {id}-t megadva egy adott műtő szobát frissítenie kell.	{ "name": "Műtő-01" }	✓
/api/rooms/{id}	Delete	Az {id}-t megadva egy adott műtőt törölnie kell.	Üres	✓
/api/rooms/	POST	Létre kell hoznia egy új műtőt.	{ "name": "Műtő-00x" }	✓

6. táblázat A doktorok végpont ellenőrzése

Tesztelt végpont	Kérés típusa	Várt eredmény	Body	Sikeresség
/api/doctors/	GET	A végpontnak vissza kell adnia az orvosokat.	Üres	✓
/api/doctors/{id}	PUT	Az {id}-t megadva egy adott időpontot frissítenie kell.	{ "name": "Doctor Who" }	✓
/api/doctors/{id}	PUT	Egy nem létező {id}-t megadva 404 Not Found választ kell adnia		✓
/api/doctors/{id}	Delete	Az {id}-t megadva egy adott időpontot törölnie kell.	Üres	✓
/api/doctors/	POST	Létre kell hoznia egy új időpontot.	{ "name": "Doctor Test" }	✓

7. táblázat A páciensek végpont ellenőrzése

Tesztelt végpont	Kérés típusa	Várt eredmény	Body	Sikeresség
/api/patients/	GET	A végpontnak vissza kell adnia az összes időpontot.	Üres	✓
/api/patients/{id}	PUT	Az {id}-t megadva egy adott időpontot frissítenie kell.	{ "patientId": 1, "name": "Pötyi", "clientId": 1, "gender": 0, "animalTypeSpecies": "Dog", "animalTypeBreed": "Dalmata" }	✓
/api/patients/{id}	Delete	Az {id}-t megadva egy adott időpontot törölnie kell.	Üres	✓
/api/patients/{id}	Delete	Egy nem létező {id}-t	Üres	✓

		megadva a törlési kísérletnek 404 Not Founddal kell visszatérnie.		
/api/patients/	POST	Létre kell hoznia egy új időpontot.	<pre>{ "name": "Bodri", "clientId": 1, "gender": 0, "animalTypeSpecies": "Dog", "animalTypeBreed": "Dalmata" }</pre>	✓

8. táblázat Az ügyfelek végpont ellenőrzése

Tesztelt végpont	Kérés típusa	Várt eredmény	Body	Sikeresség
/api/clients/	GET	A végpontnak vissza kell adnia az összes ügyfelet.	Üres	✓
/api/clients/{id}	PUT	Az {id}-t megadva egy adott ügyfelet frissítenie kell.	<pre>{ "clientId": 3, "fullName": "Teszt Elek", "phoneNumber": "-", "emailAddress": "teszt@gmail.com" }</pre>	✓
/api/clients/{id}	Delete	Nemlétező {id}-t megadva 404 hibaüzenetet kell kapni a végponttól.	Üres	✓
/api/clients/{id}	Delete	Az {id}-t megadva egy adott ügyfelet törölnie kell.	Üres	✓
/api/clients/	POST	Létre kell hoznia egy új ügyfelet.	<pre>{ "fullName": "Teszt", "phoneNumber": "-", "emailAddress": "tst@gmail.com" }</pre>	✓

9. táblázat A kategóriák végpont ellenőrzése

Tesztelt végpont	Kérés típusa	Várt eredmény	Body	Sikeresség
/api/categories/	GET	A végpontnak vissza kell adnia az összes kategóriát.	Üres	✓
/api/categories/{id}	PUT	Az {id}-t megadva egy adott kategóriát frissítenie kell.	<pre>{ "categoryName": "Test Put", "description": "Test Put Desc.. leírása1" }</pre>	✓
/api/categories/{id}	Delete	Az {id}-t megadva egy adott kategóriát törölnie kell.	Üres	✓
/api/categories/	POST	Létre kell hoznia egy új kategóriát.	<pre>{ "categoryName": "Test Product", "description": "Test desc.." }</pre>	✓

10. táblázat A termékek végpont ellenőrzése

Tesztelt végpont	Kérés típusa	Várt eredmény	Body	Sikeresség
/api/products/	GET	A végpontnak vissza kell adnia az összes terméket.	Üres	✓
/api/products/{id}	PUT	Az {id}-t megadva egy adott terméket frissítenie kell.	<pre>{ "productName": "Tst Prod", "categoryId": 1, "quantityPerUnit": "1", "unitPrice": 20.0000, "unitInStock": 1 }</pre>	✓

/api/products/{id}	Delete	Az {id}-t megadva egy adott terméket törölnie kell.	Üres	✓
/api/products/	POST	Létre kell hoznia egy új terméket.	<pre>{ "productName": "Test product", "categoryId": 1, "quantityPerUnit": "1", "unitPrice": 20.0000, "unitInStock": 1 }</pre>	✓

Miután a végpontok megfelelően működtek és a felhasználói felület is lefejlesztésre került, a program használhatósága is tesztelve lett manuálisan, hogy a funkciók a meghatározottaknak megfelelően működnek-e. Kezdve a navigációs menüvel, melyet a 11. táblázat foglal magába.

11. táblázat: A felhasználói felület navigációs menüjének tesztelése

Teszteset	Elvárt eredmény	Eredmény
Az alkalmazás indítása	Időpontfoglaló megjelenése kezdőképernyőként.	Sikeres
Az ügyfelek menüpontra kattintás	Ügyfelek kilistázódnak.	Sikeres
Az betegek menüpontra kattintás	Ügyfelek kilistázódnak.	Sikeres
Az doktorok menüpontra kattintás	Az orvosok kilistázódnak.	Sikeres
Az szobák menüpontra kattintás	Műtők kilistázódnak.	Sikeres
Az időpont típus menüpontra kattintás	Az időpont típusok kilistázódnak.	Sikeres
A termék kategóriák menüpontra kattintás	Termék kategóriák kilistázódnak.	Sikeres
A termékek kategóriák	Termékek kilistázódnak.	Sikeres

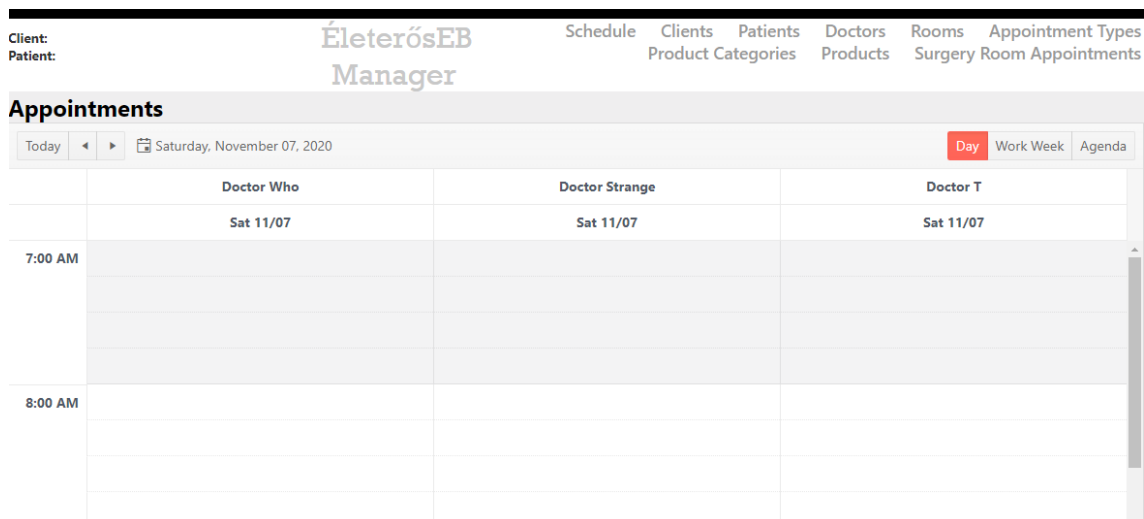
menüpontra kattintás		
A műtőfoglalás menüpontra kattintás	A foglалások megjelennek naptárban.	Sikeres
Az időpontfoglalás menüpontra kattintás	Az időpontok megjelennek a naptárban.	Sikeres

Miután az adatok listázási tesztje sikeres volt, a hozzáadás-frissítés-törlés tesztelésén volt a sor a nyilvántartó menüpontokban. Mindegyik menüpontban külön-külön végig néztem, hogy sikeresen végrehajtható-e a CRUD műveletek. A tesztelés során felvittem mindegyik fajtából egy-egyét, majd ellenőriztem, hogy az adatbázisban és a Postmannál is szerepelnek-e az adatok. Ezután az újonnan felvitt adatot a felhasználói felületen módosítottam, majd rámentettem és az előzőkhez hasonlóan az API végpont lekérdezésével, valamint az adatbázis ellenőrzésével visszakaptam, hogy a módosítási teszt sikeres. Ezt követően töröltem a felhasználói felületen az adatot, és a fenti ellenőrzések szintén visszaadták, hogy már nem szerepel ott. A nyilvántartó modul tesztjét sikeresnek nyilvánítottam, mivel a tesztek alapján a felhasználói felület, a szerveroldali végpontok és az adatbázis integráltan jól működnek együtt, ezért hibajavításra nem volt szükség.

Utolsónak megnéztem, hogy a szoftver az ajánlott böngészőn kívül is megfelelően jeleníti meg a felhasználói felületet és a funkciók helyesen működnek. A tesztelt böngészők: Brave, Opera, Edge. Vivaldi, Firefox és Internet Explorer. Különbséget megjelenítés és funkciók terén nem tapasztaltam, viszont teljesítményben igen. Az Explorer a többihez képest érzékelhetően mindig lassabban töltötte be a menüpontokat. Ez nagyban köszönhető a KendoUI keretrendszernek, mivel támogatja az összes elterjedt böngészőt. Mobiltelefon böngészőt használva – Safari - is kipróbáltam az ÉleterősEB-et. Azt észleltem, hogy a megjelenítés nem volt esztétikus, volt elcsúszások, nincs mobilfelületre optimalizálva a program, de a funkciók használhatóak voltak.

5. Felhasználói dokumentáció

A programot a kliens gépre nem szükséges telepíteni. A programot a Google Chrome böngészőben lehet elérni a helyi hálózaton, az üzemeltető által megadott címen. A program elindításakor a 8. ábrán látható képernyő fogad. Az egyes funkciók működéséhez, mint időpontfoglalás vagy műtőfoglalás előzetes adminisztrációk, adatfeltöltés szükséges.

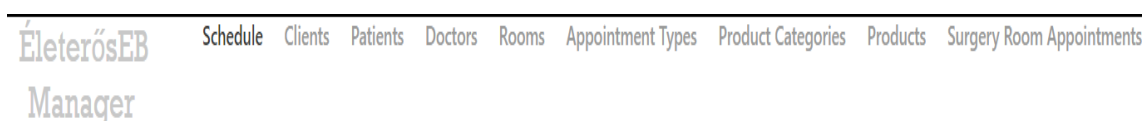


8. ábra: Az alkalmazás kezdő képernyője

5.1. Funkciói

Az alkalmazás menüsorából érhető el minden funkció, amit a 9. ábra felső sorában látható:

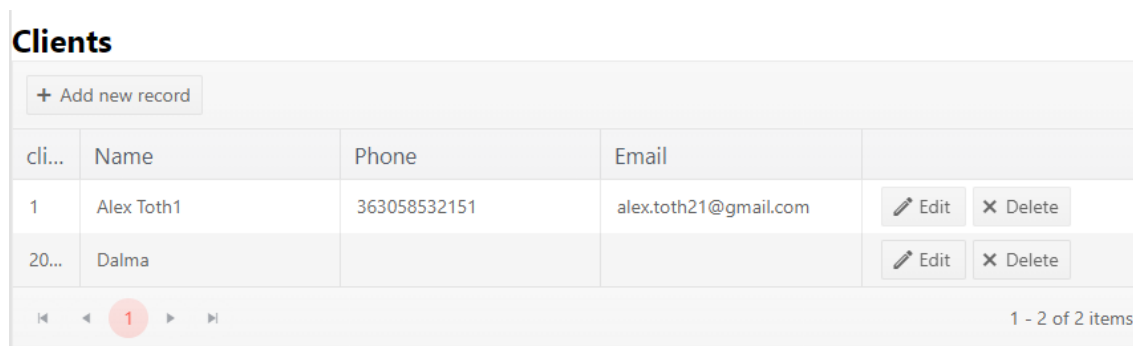
- Schedule – időpontfoglalás
- Surgery Room Appointment – műtőfoglalása
- Clients, Patients, Doctors, Rooms, Appointment Types, Product Categories, Products – Ezekben a menüpontokban történik a nyilvántartása minden adatnak



9. ábra: Az ÉleterősEB applikáció menüje.

5.1.1. Kliensek

A „Clients” menüpont szolgál a gazdák nyilvántartására. A felhasználó a menüpontban a 10. ábrán látható felületet látja, ami megmutatja az összes gazdát adataival együtt. Egy oldal 15 gazdáról tartalmaz információt, megmutatja a tulajdonos nevét, telefonszámát, e-mail címét. A felhasználó az oldalak között tud lapozni.

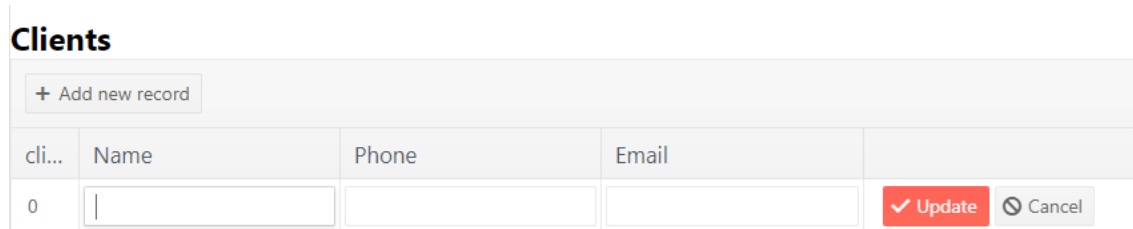


cli...	Name	Phone	Email	
1	Alex Toth1	363058532151	alex.toth21@gmail.com	Edit Delete
20...	Dalma			Edit Delete

1 - 2 of 2 items

10. ábra: A kliens nyilvántartó menüpont

Új ügyfelet az „Add new record” gomb megnyomásával hozhatunk létre. Ilyenkor a 11. ábrán látható módon keletkezik egy üres sor ezt értelemszerűen a felhasználónak kell kitölteni és az Update gomb megnyomásával létrejön az új bejegyzés.



cli...	Name	Phone	Email	
0				Update Cancel

11. ábra: Új ügyfél felvétele

Egy meglévő ügyfél szerkesztéséhez a 12. ábrán látható „Edit” gombra kell kattintani. Ilyenkor az adott ügyfélhez tartozó mezők szerkeszthetők. Miután végeztünk kattintsunk az „Update” gombra, hogy a változtatásokat elmentsük.

Clients

+ Add new record

cli...	Name	Phone	Email	
1	Alex Toth1	363058532151	alex.toth21@gmail.com	✓ Update ⛔ Cancel
20...	Dalma			✎ Edit ✕ Delete

⏪ ⏩ 1 ⏪ ⏩ 1 - 2 of 2 items

12. ábra: Meglévő ügyfél szerkesztése

5.1.2. Páciensek

A „Patients” menüpont alatt tartjuk nyilván kedvencek, betegek adatait. A menüpontban az adatok a 13. ábrának megfelelően listázódnak, szintén lapozhatóan, oldalanként 15 darab. Az ábrán jól látszik, hogy egy kedvencet lenyitva, látható a gazda adata is a könnyebb beazonosíthatóság érdekében.

Patients

+ Add new record

	Name ▼	Client ▼	Gender ▼	animalTypeSpecies ▼	animalTypeBreed ▼	
▶	Naga	Dalma	female	Dog	other	✎ Edit ✕ Delete
▼	Mokka	Alex Toth1	female	Dog	other	✎ Edit ✕ Delete

Client Information

Name	Phone Number	Email Address
Alex Toth1	363058532151	alex.toth21@gmail.com

▶	Potyi	Alex Toth1	female	Dog	Dalmata	✎ Edit ✕ Delete
---	-------	------------	--------	-----	---------	-----------------

13. ábra: Kedvencek menüpont

Az „Add new record” gomb megnyomásával új páciens vehető fel, ahogy a 14. ábra mutatja. Itt megadhatjuk a beteg nevét, nemét, fajtát, fajtáját és tulajdonosát. A gazda kiválasztását legördülő menü segíti, kulcsszavakkal csökkenthetőek a találatok. Az „Update” gomb megnyomásával a bejegyzés elmentésre kerül.

Patients

+ Add new record

Name	Client	Gender	animalTypeSpecies	animalTypeBreed	
					✓ Update ✕ Cancel
▶ Naga	<input type="text"/>	female	Dog	other	Edit ✕ Delete
▶ Mokka	Alex Toth1	female	Dog	other	Edit ✕ Delete
▶ Potyi	Dalma	female	Dog	Dalmata	Edit ✕ Delete

Gender

▼

female

male

14. ábra: Új páciens felvétele

Páciensek törlése a rendszerből a „Delete” gomb lenyomásával történik, ilyenkor felugrik egy ablak és megkérdezi, hogy biztos szeretnénk-e törölni az adatot – ahogy a 15. ábra is mutatja. A törlés végrehajtásával csak a páciens törlődik a rendszerből, a gazda nem.

A(z) localhost:8888 közlendője

Are you sure you want to delete this record?

OK

Mégse

15. ábra: Törlés megerősítés felugró ablaka

A páciens menüpont ábráin az oszlopok neve mellett egy tölcsér ikon helyezkedik el, ami segít a találatokat leszűkíteni. Oszloponként két darab szűrési feltétel adható meg és/vagy kapcsolattal. Az „és” kapcsolattal mindkét feltételnek teljesülni kell, a „vagy” kapcsolatnál elég, ha csak az egyik teljesül – ezt szemlélteti a 16. ábra.

Show items with value that:

Is equal to ▼

And ▼

Is equal to ▼

Filter Clear

16. ábra: Páciensek szűrése

A „Filter” gomb megnyomásával alkalmazzuk az oszlopra a szűrőt, a „Clear” gombbal pedig törölhetjük.

5.1.3. Doktorok, Szobák

A „doctors” és a „rooms” menüpont (lásd. 17. ábra) is nagyban hasonlít a többihez. Az „Add new record” gomb lenyomásával új orvos vagy szoba vihető fel. Módosítani az „Edit”, frissíteni/menteni az „Update”, törölni a „Delete” gombbal lehet, a törlési szándék megerősítése után.

Doctors

+ Add new record	
Name	
doc	Edit Delete
doc233	Edit Delete
<div> ⏪ ⏴ 1 ⏵ ⏩ </div> <div>1 - 2 of 2 items</div>	

Rooms

+ Add new record	
Name	
doc	Edit Delete
doc233	Edit Delete
<div> ⏪ ⏴ 1 ⏵ ⏩ </div> <div>1 - 2 of 2 items</div>	

17. ábra: A doktorok és szobák menüpont

5.1.4. Kezelés típusa

Az „AppointmentTypes” menüpontban adminisztrálhatók a kezelés típusa. Kezdő képernyőként kilistáz 15 darabot oldalanként. Ugyanazok a funkciók érhetőek el itt is, mint az előző menükben, például „clients”. A különbség csak a megadott adatokban van. Ennél a menüpontnál a kezelés típusának neve, kódja és hossza percben adható meg, amint a 18. ábrán látható.

Name	Code	Duration	
<input type="text"/>	<input type="text"/>	<input type="text" value="0.00"/>	<input checked="" type="button" value="Update"/> <input type="button" value="Cancel"/>
műtét	S-001	15	<input type="button" value="Edit"/> <input type="button" value="Delete"/>
műtét2	s-002	100	<input type="button" value="Edit"/> <input type="button" value="Delete"/>

1 - 3 of 3 items

18. ábra: Az időpontok típusai menüpont

5.1.5. Időpontfoglalás

Ez a funkció a program kezdő felülete, ami a 8. ábrán látható. A naptár orvosokra lebontva jelenik meg. A jobb felső sarokban nézetet válthatunk napi, heti lebontásban vagy összefoglaló táblázatban is megtekinthetjük az időpontfoglalásokat. A naptárban az órák reggel 7 óra és este 18 óra között jelennek meg, így csak a nyitvatartási időre lehet időpontot foglalni. Egy óra négy egyenlő részre osztozik, ezzel segítve a kezdő időpont kiválasztását. A „Today” gombbal az aktuális mai napra ugorhatunk, és a nyilakkal lapozhatunk előre-hátra a dátumok között.

Az időpontfoglalás megkezdéséhez, legelsőnek ki kell választunk az ügyfelet és a beteget. Ez a bal felső sarokban található részre kattintva tehető meg. Ezután felugrik egy ablak (19. ábra), mely elsőnek bekéri az ügyfelet. Ha elkezdjük beírni a nevét, akkor a felajánlottak közül ki tudjuk választani a megfelelőt. Ezután bővül az ablak egy újabb opcióval, ahol a gazdához tartozó kedvencet választhatjuk ki egy legördülő menüből.

Choose Patient [X]

Client

Tóth Dalma

Start typing to find a client

Patient

Mokka ▼

Mokka

Naga

19. ábra: Kliens-Páciens kiválasztó felugró ablak

A „Select” gomb megnyomásával kötelező kiválasztani a beteget és a gazdáját, utána enged a rendszer időpontot választani. Egy piros keret figyelmeztet minket arra, ha a kliens-beteg kiválasztása nem történt meg – lásd 20. ábra.

Client:

Patient:

20. ábra: Figyelmeztetés, amennyiben elfelejtünk beválasztani klienst és beteget

Miután kijelöljük az ügyfelet és beteget, a naptárban a doktorok alatt az üres helyeken kiválasztunk egy időpontot, ide duplán kattintunk az egérrel, ekkor felugrik az időpont hozzáadása ablak, ami a 21. ábrán látható.

Appointment

Add Appointment

Appointment Type:

Alap vizsgálat

Duration:
20 minutes

Patient

Save

Cancel

21. ábra: Új időpont hozzáadása

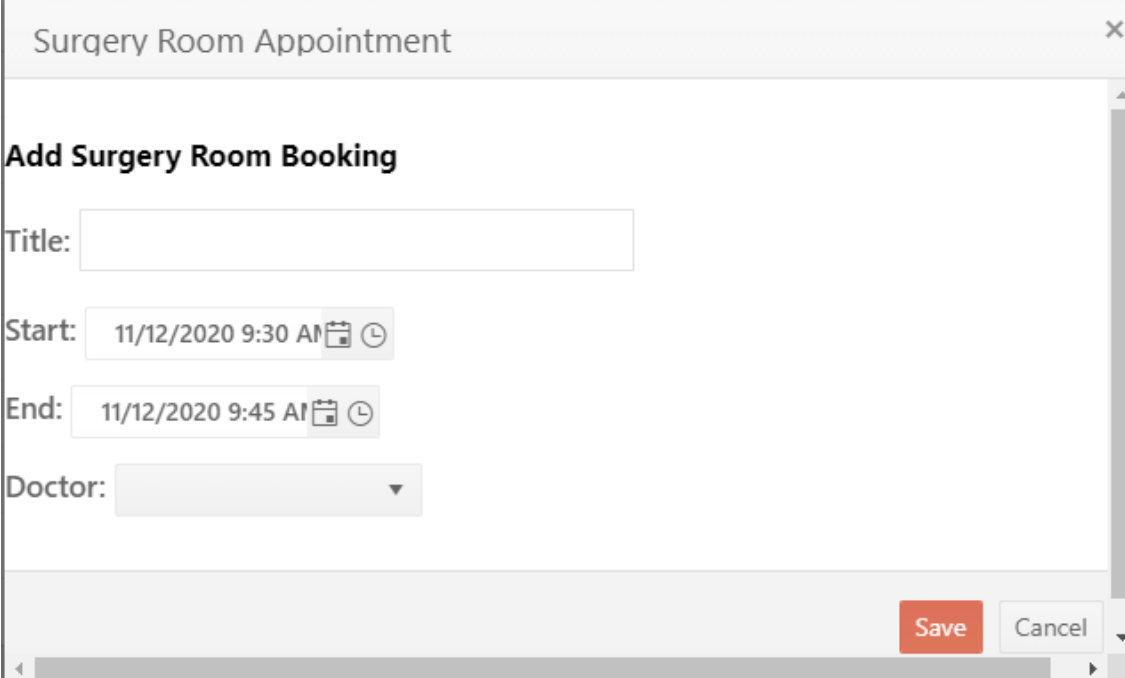
Kiválasztjuk a vizsgálat típusát, majd a rendszer automatikusan megjeleníti mennyi ideig fog tartani a kezelés, ami a Save gombbal menthető. Mentés után az időpont a 22. ábrán látható módon megjelenik az adott doktor alatt a többi foglalással együtt.

Appointments					
Today		11/4/2020		Day	
	Doctor Who	Doctor Strange		Doctor T	
	Wed 11/04	Wed 11/04		Wed 11/04	
7:00 AM					
8:00 AM		M-001 Pótyi Tóth Alex	M-001 Pótyi Tóth Alex	M-001 Mokka Tóth Dalma	M-001 Pótyi Tóth Alex
9:00 AM				M-002 Mokka Tóth Dalma	
10:00 AM					

22. ábra: Időpontok megjelenés a naptárban

5.1.6. Műtőfoglalás

A funkció a „Surgery Room Appointment” menüpontban érhető el. A naptárban láthatjuk a műtőfoglalásokat, ugyan úgy, mint az időpontfoglalásokat. A különbség abban rejlik, hogy műtőkre van felosztva a naptár orvosok helyett. Ha műtőt akarunk foglalni belekattintunk a naptárba a megfelelő időpontnál és felugrik egy ablak – lásd 23. ábra.



23. ábra: Új műtőfoglalás

Megadjuk a műtét címét, kezdési és befejezési időpontját, valamint egy legördülő menüből kiválasztjuk a műtétet végző orvost. A Save gombbal mentjük a műtő foglalását.

Szerkesztésnél a naptárban az adott foglalásra kétszer kattintunk, ekkor felugrik egy ablak a korábban kitöltött adatokkal – lásd 24. ábra. Miután ezeket módosítottuk szintén a Save gombbal menthetjük el.

24. ábra: Műtőfoglalás szerkesztése törlése

A foglalás törléséhez elegendő a foglalásra vinni az egeret, a jobb felső sarkában megjelenik egy „x”, erre kattintva eltávolíthatjuk a foglalást vagy megnyitjuk szerkesztésre és a bal alsó sarokban található „Delete” gombbal tudjuk törölni. Mindkét fajta törlésnél kapunk egy megerősítésre vonatkozó felugró ablakot – 25. ábra. A „Cancel” gombbal vagy az ablak becsukásával még visszakozhatunk és nem történik változás vagy a „Delete” gombra kattintva véglegesítjük a törlést.

25. ábra: Műtőfoglalás törlésének megerősítése/visszakozás

5.1.7. Nyilvántartás – Kategóriák és Termékek

A nyilvántartáshoz először szükséges a megfelelő kategóriákat rögzíteni. Ez a „Product Categories” menüpontban tehető meg. A menüpontban kattintva megtekinthető az összes kategória. Új rögzítése, szerkesztése vagy törlése a megfelelő akciógomb kiválasztásával kezdeményezhető, amit a 26. ábra szemléltet. Kategóriákhoz két adatra van szükség, a kategória nevére és leírására.

Product Categories		
+ Add new record		
Name	Description	
Eszközök	Eszközök leírása1	Edit Delete
Other things	Más fajta eszközök leírása	Edit Delete
1		1 - 2 of 2 items

26. ábra: A kategóriák menüpont

Amikor a kategóriák felvitelre kerültek az alkalmazásban, azután lehet rögzíteni a termékeket, mivel szükséges azokat kategóriákba sorolni. A termékek adminisztrálása a „Products” menüpontban érhető el. Az előző menüpontok elvén működik az új termék létrehozása, szerkesztése és törlése is. A 27. ábrán szerkesztés közben látható a menüpont, ahogy éppen a kategória beválasztása történik. Egy lenyíló ablakban választható ki és szűkíthető kulcsszavakkal a kategória.

Products					
+ Add new record					
Product Name	Category	Quantity Per Unit	Unit Price	Unit In Stock	
Termék1	Eszközök	1	20	1	Edit Delete
Termék2	Other things	5	33.00	1.00	Update Cancel
Termék3	<input type="text" value=""/>	1	444	4	Edit Delete
termék4	Eszközök	2	2	3	Edit Delete
	Other things				

27. ábra: A termékek menüpont szerkesztés közben

6. Üzemeltetői dokumentáció

Az alkalmazás első verziója kisebb állatklinikák számára készült, ahol pár orvos és egy recepciós dolgozik, ezért nincsenek felhasználói szintek. Csak egy központi felhasználó kezeli a programot. A szoftver lehetőséget biztosít az adminisztrációra. Nyilvántartható és kategóriákba sorolható benne minden, valamint lehetőség van időpont és műtő foglalására is.

A szoftvert .NET Core típusú webalkalmazásokat támogató webserverre szükséges telepíteni. A végfelhasználó pedig a saját eszközén böngészőn keresztül éri el és tudja használni.

6.1.Rendszerkövetelmények

Kliens oldal

A rendszer futtatásához, alkalmazásához a kliens oldalon minimum Google Chrome 86.0.4240.183 böngésző szükséges. A kliens a szerverhez a 8888-as porton kapcsolódik http hivatkozással. A minimum konfigurációt a 12. táblázat szemlélteti.

12. táblázat: Kliens oldali konfiguráció

Minimum számítógép konfiguráció:	2 GHz Processzor
	1024 MB RAM
Operációs rendszer környezet:	Windows 10
	Google Chrome 86.0.4240.183 vagy újabb

Szerver oldal

13. táblázat: Szerver konfiguráció

Minimum számítógép konfiguráció:	3 GHz Processzor
	8GB RAM
Operációs rendszer környezet:	Windows 10 vagy Windows Server 2016

Hálózat

A hálózat beállításainál nincs szükség egyedi speciális beállítást elvégezni. A szerver a kliensekkel TCP/IP protokollt használva kommunikál. A beállítások elvégzése után ellenőrizni kell a forgalmat és a szerver elérhetőségét a hálózaton elérhetőségét az operációs rendszer eszközeivel. A szükséges konfigurációt a 14. táblázat tartalmazza.

14. táblázat Hálózat konfiguráció

Kiszolgáló:	IIS
Hálózat:	Bármely TCP/IP alapú hálózat
Adatbázis:	Microsoft SQL Server Express

6.2. Adatbázis telepítése

A program használatához az alábbi adatbázisok közül egyre szükségünk van:

- SQL Server,
- SQLite,
- Azure Cosmos DB,
- PostgreSQL,
- MySQL.

Ajánlott a MS SQL Express használata, a fejlesztés és tesztelés alatt is ezt a típusú adatbázist használtam. Letölthető a <https://www.microsoft.com/en-us/sql-server/sql-server-downloads> webhelyről telepíteni szükséges. Miután telepítettük az adatbázist a menedzseléséhez a Microsoft SQL Server Management Studio programot használjuk, ami itt érhető: <https://docs.microsoft.com/en-us/sql/ssms/download-sql-server-management-studio-ssms?view=sql-server-ver15>.

A menedzselő szoftverben felcsatlakozva az adatbázisra lefuttatjuk a mellékletként kapott fájlt: eleterosEB_db_script.sql. Ez létrehozza az adatbázist és a tábláit.

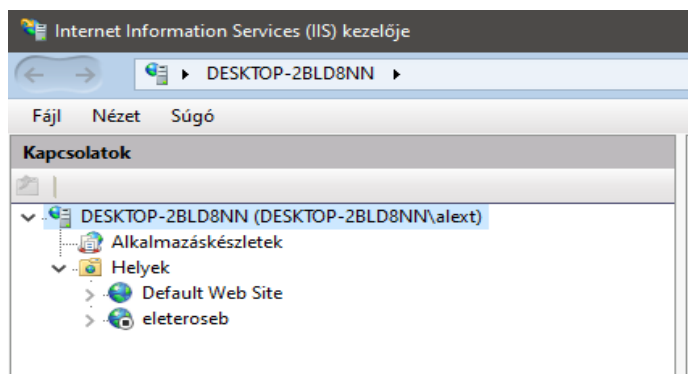
6.3. Alkalmazás telepítése

Miután feltelepítettük az adatbázist, meg kell adnunk a programunk számára is, hogy hol éri el. Ezt az appsettings.json fájlban tehetjük meg. Minta:

```
{
  "ConnectionStrings": {
    "EleterosEBConnex": "Data Source = (localdb)\\MSSQLLocalDB; Initial Catalog = EleterosEBAppData; User Id = username; Password = testpass123"
  }
}
```

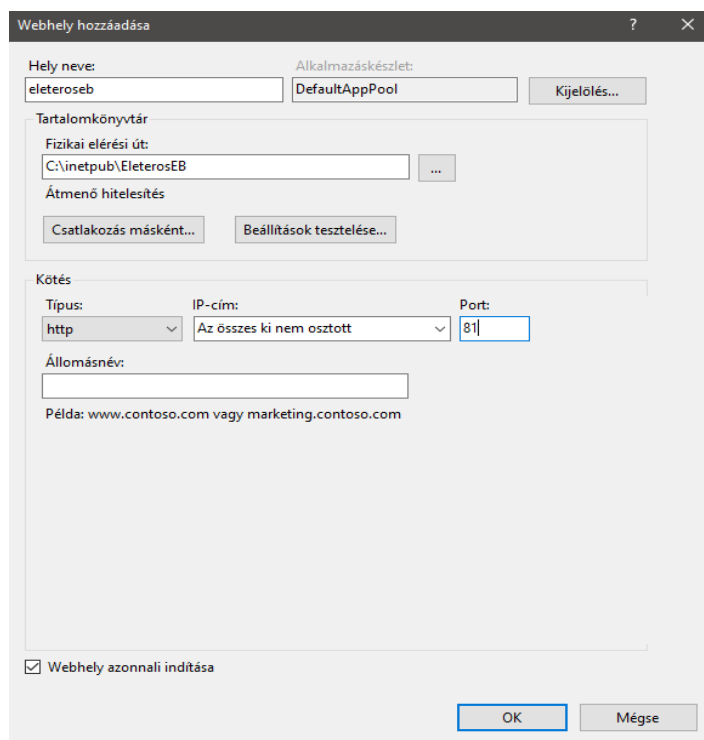
A Data Source-nál megadjuk az adatbázis elérési címét, az Initial Catalogot, a felhasználónevet és jelszót. Amennyiben a scriptet használtuk az adatbázis és táblái létrehozására, marad az EleterosEBAppData név.

A szoftvert ISS-re telepítjük. Az ISS kezelőjét elindítva a 28. ábrán látható képernyő fogad. Itt a „Helyek”-re jobb egérgombbal kattintva előugrik egy menü, ahol a Webhely hozzáadására kell kattintani.



28. ábra ISS kezelője

Ezután a felugró ablakban, amit a 29. ábra szemléltet, beállítjuk a tartalom elérési útját, a mellékelt fájlt – EleterosEbApp.zip – kitömörítjük arra a helyre, ahol szeretnénk tárolni a programot és behivatkozzuk. Megadjuk a nevét „EleterosEB” és a kötés típusát – a példában localhost elérést szemléltetve.



29. ábra Webhely hozzáadása - IIS

Miután rákattintottunk az OK gombra a webalkalmazás elindul és, máris használatba vehetjük a böngészőben a beállított címen és porton.

7. Tapasztalatok, fejlesztési lehetőségek, irányok

Az elkészült szoftvernek az előzetesen meghatározott követelményeknek kell megfelelnie, tehát olyan funkciókat kell ellátnia, mint az időpont- és műtőfoglalás, valamint az adatok nyilvántartása. Ugyan nem volt leszögezve, hogy webes alapúnak kell lennie, de a mai trendeket és a jövőbeni könnyű fejlesztési lehetőségeket figyelembe véve úgy döntöttem, hogy webes alapú lesz, és a szolgáltatásokat REST API végpontokon keresztül biztosítja, JSON struktúrában. Így a felhasználói felület a jövőben könnyen cserélhető, akár más nyelven íródottira is vagy natív mobil, tablet, esetleg desktop alkalmazást is lehet fejleszteni hozzá. Ebből következik, hogy az alkalmazást nem célszerű monolitikusnak tervezni, ahhoz túl terjedelmes, hanem MVC architektúrát követve több rétegűre kell kialakítani. Ez meghatározta, hogy a programozási nyelvnek ezt támogatnia kell, és az egyik legnépszerűbb C# ASP.NET Core keretrendszere ezt támogatja is. Ezért választottam ezt, a jövőben sem választanék más nyelvet és keretrendszert hasonló projektre, mert könnyű és a core verzió már nincs ISS hostinghoz feltétlen kötve. A fejlesztés végeztével és a program tesztelése közben arra a következtetésre jutottam, hogy sokkal egyszerűbb a TDD-t – Test Driven Development – követve fejleszteni. Ebben az esetben először az üzleti logikát úgy fejlesztettem volna, hogy legelsőnek az egység tesztjét írom meg és utána foglalkozom az alkalmazás logikájával, így mindig garantálva van, hogy az adott rész megfelelően működjön. Az én általam elsőnek választott manuális tesztelés nagy hátránya az volt, hogy a programnak már kész kellett lennie így amennyiben hibát észleltem, sokkal nehezebb volt megtalálni annak okát.

Elsőként a kórlap funkcióval bővíteném a szoftvert, ahol rögzíteni lehetne mikor milyen kezelésben részesült a páciens, milyen gyógyszereket kapott. Ezáltal az előzményeket megtekintve az orvos mindig teljes képet kapna az állat betegtörténetéről. Nem lenne külön kórlapok menü, a lapokat az páciensek menüpontra belül lehetne megtekinteni. Itt a házikedvencet lenyitva lenne elérhető a hozzá tartozó kórlap, a tulajdonos adatainak mintájára. Bevezetném a szűrés és rendezés lehetőségét oszloponként a könnyebb kereshetőség érdekében. A beteglapokat külön oldalra lehetne megnyitni, ahol már teljes információt kapunk szerkeszthetően. Új beteglap létrehozásra az adott kedvencnél lenne lehetőség egy akciógombbal. Ezzel egyből relációba lépne a gazda és páciens a kórlappal, csak a vizsgálattal kapcsolatos adatok kitöltése válna szükségessé.

Az alkalmazás a továbbfejlesztésének következő lépcsőfokaként felhasználói- és jogosultságkezeléssel bővíteném. Ezzel szükségessé válna mindenkitől az alkalmazásba való bejelentkezés. Külön lenne rendszer adminisztrátori jogkör, mely hozzáférést biztosítana mindenhez. Külön bontanám az orvos és a recepciós jogkört, hogy mindenki csak a feladatához szükséges funkciókhoz férjen hozzá, ezzel is csökkentve a biztonsági kockázatot. Ennek az új funkciónak köszönhetően létrehoznék egy új felhasználói típust az ügyfeleknek. Ezzel és a webes mivoltának köszönhetően egy jó pár hasznos funkció továbbfejlesztésére nyílik lehetőség, úgymint például az ügyfél is tudjon időpontot foglalni webes felületen keresztül. Ez szintén időt spórolna meg a klinika dolgozóinak.

Ehhez SignalR Hub-okat hoznék létre a szerver oldalon. Ez egyből megjeleníti a felhasználónak, ha a foglalás közben más bejelentkezett egy időpontra.

Magát az időpontfoglaló rendszert még tovább fejleszteném egy automatikus emlékeztető e-mail küldésével a vizsgálatot megelőző napon. Ezt viszonylag egyszerű lenne bevezetni, mivel már az alap program tárolja az ügyfelek e-mail címeit.

Ezen kívül szeretném bevezetni az alkalmazás többnyelvű támogatását is, hogy magyar nyelven is elérhető legyen, de a jövőbeni lehetőségeknek csak az ügyféligények szabhatnak határt.

8. Összegzés

Szakdolgozatomban a kitűzött cél az volt, hogy egy átlátható, megfelelően működő webalkalmazást tervezek, fejlesszek, mely lehetőséget biztosít adminisztrálásra, időpontfoglalásra és műtőfoglalás vezetésére az állatklinikák számára. Azonban továbbfejlesztésre szorul ahhoz, hogy élesben minden igényét kiszolgálja egy állatklinikának. Ezt a továbbfejlesztési lehetőségek fejezet alatt kifejtettem.

A tervezés és fejlesztés során mindent megtettem, hogy a tanultak szerint a legjobb módszert válaszam ki és alkalmazzam. A fejlesztés során sikerült minden funkciót megvalósítani, azok közül, amiket előre elterveztem. A program írása során új tapasztalatokra tettem szert mind az Entity Framework Core, mind az ASP.NET Core keretrendszerekben. A dolgozatban folyamatosan igyekeztem a technikai részt is érthetően, átláthatóan bemutatni és fejlesztés során a tiszta kód elvét alkalmazni, annak érdekében, hogy más fejlesztő is megértse és könnyen hozzá tudjon nyúlni az alkalmazáshoz.

A fejlesztés során a felhasználói felület programozása volt kihívásokkal teli, mert ezen a téren nem rendelkezem tapasztalattal rendelkezem, ezért is választottam a kendoUI keretrendszert, melyet egy ismerősöm ajánlott. Sajnos, ezek elég korlátozottak tudnak lenni, ha olyan funkciót szeretnénk velük megvalósítani, melyet alapból nem tartalmaznak. Ezek implementációja nehézkes. Így voltam, amikor a naptár nézetnek az erőforrásait akartam egy egyedi szerkesztőablakban elérhetővé tenni. Szerencsére már más is ütközött hasonló problémába, így hivatalos fórumon találni rá megoldást. Ennek ellenére a jövőben, nem használnék kész megoldásokat. Ezért célszerűbbnek, fejleszthetőbbnek tartom a saját megoldást.

A tesztelési folyamat során minden lépésben leteszteltem manuálisan, hogy megfelelően működik-e az alkalmazás, az eredményeket táblázatosan szemléltettem és megállapítottam, hogy megfelelően működnek a tervezett funkciók.

Ezután elkészítettem egy kellően illusztrált használati útmutatót, hogy a program jövőbeni használói könnyedén elsajátíthassák használatát.

Mindent összevetve jó tapasztalat volt a tervezéstől a megvalósításig végigvinni a projektet. Elégedett vagyok az alkalmazás első verziójával, mindenféleképpen továbbfejleszttem a jövőben az említett tervekkel.

Irodalomjegyzék

- [1]<https://docs.microsoft.com/hu-hu/aspnet/core/mvc/overview?view=aspnetcore-3.1>
Utoljára megtekintve: 2020. 11. 04.
- [2]<https://deviq.com/repository-pattern/> Utoljára megtekintve: 2020. 11. 04.
- [3]<https://www.programmingwithwolfgang.com/repository-and-unit-of-work-pattern/>
Utoljára megtekintve: 2020. 11. 04.
- [4]https://hu.wikipedia.org/wiki/A_f%C3%BCgg%C5%91s%C3%A9g_befecskendez%C3%A9se Utoljára megtekintve: 2020. 11. 12.
- [5] Julia Lerman, Programming Entity Framework 2nd Edition. O'Reilly Media Inc. 2010
- [6] Julia Lerman & Rowan Miller, Programming Entity Framework Code First . O'Reilly Media Inc. 2012
- [7] Andrew Troelsen & Philip Japikse. Pro C# 7 Eighth Edition, Apress 2017
- [8] Adam Freeman, Pro ASP.NET Core , Apress 2020
- [9] Robert C. Martin, Clean Code, O'Reilly Media Inc., 2008

Ábrajegyzék

1. ábra Az alkalmazás USE CASE diagramja	5
2. ábra: A szoftver architektúra modellje	7
3. ábra Az adatbázis diagramja.....	11
4. ábra A ClientService osztály diagramja	14
5. ábra: Az AppointmentService osztály	15
6. ábra: A Room-, Patient-, Product-, Category-, AppointmentType- és SurgeryAppointmentService osztályok	16
7. ábra: A controller osztályai	17
8. ábra: Az alkalmazás kezdő képernyője	31
9. ábra: Az ÉleterősEB applikáció menüje.....	31
10. ábra: A kliens nyilvántartó menüpont	32
11. ábra: Új ügyfél felvétele	32
12. ábra: Meglévő ügyfél szerkesztése	33
13. ábra: Kedvencek menüpont	33
14. ábra: Új páciens felvétele	34
15. ábra: Törlés megerősítés felugró ablaka.....	34
16. ábra: Páciensek szűrése	35
17. ábra: A doktorok és szobák menüpont	35
18. ábra: Az időpontok típusai menüpont	36
19. ábra: Kliens-Páciens kiválasztó felugró ablak.....	37
20. ábra: Figyelmeztetés, amennyiben elfelejtünk beválasztani klienst és beteget	37
21. ábra: Új időpont hozzáadása.....	38
22. ábra: Időpontok megjelenés a naptárban	38
23. ábra: Új műtőfoglalás	39
24. ábra: Műtőfoglalás szerkesztése törlése	40
25. ábra: Műtőfoglalás törlésének megerősítése/visszakozás	40
26. ábra: A kategóriák menüpont	41
27. ábra: A termékek menüpont szerkesztés közben.....	41
28. ábra ISS kezelője	44
29. ábra Webhely hozzáadása - IIS	44

Táblázatjegyzék

1. táblázat: A program modelljei és azok tulajdonságai.....	8
2. táblázat: Az időpontfoglaló végpont tesztelése	23
3. táblázat Az időpont fajtájának végpontjának tesztelése	24
4. táblázat A műtő foglalás végpont tesztelése.....	24
5. táblázat A szobák végpont ellenőrzése.....	25
6. táblázat A doktorok végpont ellenőrzése	26
7. táblázat A páciensek végpont ellenőrzése	26
8. táblázat Az ügyfelek végpont ellenőrzése	27
9. táblázat A kategóriák végpont ellenőrzése	28
10. táblázat A termékek végpont ellenőrzése	28
11. táblázat: A felhasználói felület navigációs menüjének tesztelése	29
12. táblázat: Kliens oldali konfiguráció	42
13. táblázat: Szerver konfiguráció	42
14. táblázat Hálózat konfiguráció.....	43

Mellékletek jegyzéke

1. A program tervezett osztálydiagramja

1. melléklet:

A program tervezett osztálydiagramja

