ÓBUDAI EGYETEM
ÓBUDA UNIVERSITY

NEUMANN JÁNOS

INFORMATIKAI KAR

SZAKDOLGOZAT

OE-NIK	Hallgató neve:	**Bozsik Zsolt**
2020	Hallgató törzskönyvi száma:	**T-005056/FI12904/N**

Óbudai Egyetem
Neumann János Informatikai Kar
Kiberfizikai Rendszerek Intézet

SZAKDOLGOZAT FELADATLAP

Hallgató neve: **Bozsik Zsolt**
Törzskönyvi száma: T/005056/FI12904/N
Neptun kódja: [redacted]

A dolgozat címe:

Ügyfélszokások elemzése időpontfoglalási adatok alapján
Customer behaviour analytics based on booking data

Intézményi konzulens: Simon-Nagy Gabriella
Külső konzulens:

Beadási határidő: 2020. december 15.

A záróvizsga tárgyai: Számítógép architektúrák
Big Data és üzleti intelligencia specializáció

A feladat

Tervezzen és valósítson meg egy olyan rendszert, amely online időpontfoglalási lehetőséget biztosít egy szolgáltatáshoz (például fodrász, kozmetikus, stb.), és az így nyert adatok alapján elemzi a felhasználói viselkedést. Kutassa fel és vizsgálja meg a feladat megoldásához használható technológiákat. Ismertesse a hasonló rendszerek működését és képességeit. Valósítson meg egy olyan tesztrendszert, amely kezeli az ügyfélprofilokat és az időpontfoglalásokat, valamint képes az ügyfélszokásokat feltérképezni (ki milyen gyakran foglal, mennyire megbízható, tipikusan milyen szolgáltatást vesz igénybe), és ennek alapján predikciókat generál a jövőbeli terhelésre.

A dolgozatnak tartalmaznia kell:

- a feladat pontos ismertetését,
- a hasonló rendszerek működését és képességeit,
- a feladat megoldásához alkalmazható technológiák tömör bemutatását,
- a választott megoldások indoklását,
- a tervezés és megvalósítás részletes leírását,
- a tesztelés tervét és eredményeit,
- az elkészített rendszer eredményeinek részletes bemutatását és értékelését,
- a továbbfejlesztési lehetőségek számba vételét.

Ph.

ÓBUDAI EGYETEM * Neumann János Informatikai Kar * 1. Budapest

.....
Dr. habil. Lovas Róbert
intézetigazgató

A szakdolgozat elévülésének határideje: **2022. december 15.**
(OE TVSz 55.§ szerint)

A dolgozatot beadásra alkalmasnak tartom:

.....
külső konzulens

.....
intézményi konzulens

ÓBUDAI EGYETEM
ÓBUDA UNIVERSITY

Neumann János Informatikai Kar
7. sz. melléklet

HALLGATÓI NYILATKOZAT

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

Budapest, 2021. 05. 10.....

.....
hallgató aláírása

ÓBUDAI EGYETEM
ÓBUDA UNIVERSITY

Neumann János Informatikai Kar

KONZULTÁCIÓS NAPLÓ

Hallgató neve: Neptun Kód: Tagozat:

Bozsik Zsolt ..... [redacted] ..... Nappali

Telefon: Levelezési cím (pl.: lakcím):

[redacted]

Szakdolgozat címe magyarul:

Ügyfélszokások elemzése időpontfoglalási adatok alapján

Szakdolgozat címe angolul:

Customer behaviour analytics based on booking data

Intézményi konzulens: Külső konzulens:

SIMON-NAGY GABRIELLA

Alk.	Dátum	Tartalom	Aláírás
1.	2020. 02. 24	A téma meghatározása	
2.	2020. 03. 25.	A feladatok megbeszélése	
3.	2020. 04. 20.	Az irodalomkutatás összegzése	
4.	2020. 05. 19.	Rendszerspecifikáció véglegesítése	

A hallgató a Szakdolgozat I. tantárgy követelményét teljesítette, beszámolóra bocsátható.

.....

Intézményi konzulens

Budapest, 2020. május 22.

ÓBUDAI EGYETEM
ÓBUDA UNIVERSITY

Neumann János Informatikai Kar

KONZULTÁCIÓS NAPLÓ

Hallgató neve: Bozsik Zsolt

Neptun Kód: [redacted]

Tagozat: Nappali

Telefon: [redacted]

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

Szakdolgozat címe magyarul:

Ügyfélszokások elemzése időpontfoglalási adatok alapján

Szakdolgozat címe angolul:

Customer behaviour analytics based on booking data

Intézményi konzulens:

SIMON-NAGY GABRIELLA

Külső konzulens:

Alk.	Dátum	Tartalom	Aláírás
1.	2021. 02. 23.	Megvalósítási lehetőségek áttekintése	
2.	2021. 03. 17.	Fejlesztési fázis ellenőrzése	
3.	2021. 04. 22.	Tesztelés ellenőrzése	
4.	2021. 05. 10.	Eredmények, összegzés	

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

.....

Intézményi konzulens

Budapest, 2021. május 10.

Tartalomjegyzék

1. Bevezetés

1.1. Feladat ismertetése

Szakdolgozatom témája egy szolgáltatáshoz tartozó online időpontfoglalási rendszer, amely képes adatokat tárolni az ügyfelek foglalási szokásairól és ezeket elemezve segít feltérképezni az ügyfélszokásokat.

Manapság a legtöbb szolgáltatáshoz tudunk időpontot foglalni, hogy ne kelljen annyit várakoznunk, viszont a legtöbb helyre mai napig csak telefonon tudunk foglalni. Ahol pedig lehet online is foglalni, ott nem mindig tudunk olyan részletesen foglalni, mint például telefonon. A tömegesen elterjedt online foglalós rendszerek közül a legtöbb korlátozott és nem személyre szabható, valamint a megbízhatóságuk sem megfelelő néhány esetben. Valamint fejleszthetőségük nem lehetséges vagy részlegesen.

A szakdolgozatom során egy olyan rendszert szeretnék tervezni és megvalósítani, amik a fenti hibákat kiküszöbölik és ezen felül még a vásárlóink szokásait is megismerhetjük. Minél többet tudunk vevőinkről annál könnyebben tudunk nekik olyan szolgáltatást nyújtani, ami garantálja, hogy a jövőben is igénybe veszik azt. Figyelni tudjuk a megbízhatatlan foglalókat, akik nem jelennek meg az időponton, vagy pár perccel előtte mondják azt le.

A magánvállalkozók és kis, illetve középvállalatok, akiknek szól a rendszer többféle módon rögzítenek foglalást. A legtöbb módszer viszont elavult vagy egyszerűen körülményes a használata. Semmilyen módon nem segít a szolgáltatónak, vagy csak annyiban, hogy nem kell egy saját recepciót fogadni az üzlethez.

Ezenkívül a célom egy konkrét szolgáltatáshoz tartozó rendszer megvalósítása, amit a valóéletben könnyen, egyszerűen lehet használni különösebb tudás vagy képzés nélkül. Minden modulnak, amit tartalmaz jól áttekinthető felületet szeretnék, amit a szolgáltató és a vendég is örömmel használ. Olyan funkciókat, amik felgyorsítják és megkönnyítik az életüket és munkájukat.

A szakdolgozatomban bemutatom a jelenleg használt rendszereket ezeknek az előnyeit és hátrányait, majd lehetséges megoldási lehetőségeket. Ezután a kiválasztott megvalósítás megírt szoftver funkcióinak lényegét fejtem ki. A záró részben a tesztelést, és a jövőbeni továbbfejlesztési lehetőségeket ismertetem.

2. Jelenlegi foglalási rendszerek

Barátaim és rokonaim által betekintést nyertem a jelenleg használatban lévő időpontfoglalási rendszerek használatába, mint szolgáltató. Ezeket a rendszereket szeretném összehasonlítani. Mindegyik rendszer elérhető online böngészőből. A megvalósítandó rendszert is hasonló működéssel képzelem el ezért választottam ezeket az összehasonlításhoz és elsőnek pedig a legalapvetőbb „rendszert" a mobilos bejelentkezést választottam.

2.1. Hagyományos telefonos rendszer

A legtöbb szolgáltató mai napig telefonon elérhető csak a foglalásokat pedig egy naptárba tárolja. Bár a legolcsóbb módszer ez viszont embert igényel szóval nem automatikus. Foglalás törlés és módosítás körülményes, adatelemzés nincs vagy szintén manuálisan. Az elérhetőség is kevesebb. Viszont a rendszerünkben helyet kell hagynunk ennek a lehetőségnek is csak elektronikus adattárolással.

2.2. Salonic

Borbély ismerősöm által használt Salonic egy magyar fejlesztésű Facebook-ba integrált rendszer, amivel bárki tud foglalni, aki rendelkezik Facebook fiókkal, vagy egy email címmel és telefonszámmal. Maga az alkalmazás használata ingyenes és egyszerű, viszont itt ki is merülnek az előnyei. Egyik legnagyobb hátránya, hogy nem tudjuk lemondani magát a foglalást, még akkor se, ha több nappal előbb szeretnénk ezt. Adatokat nem tárol. Csak időpont foglalásra jó másra nem.[1]

2.3. Booked4us

Személyiedző barátom által használt Booked4us nevű oldal szintén egy magyar fejlesztésű különálló weblap. Aktív és gyors Support-al rendelkezik. Regisztrációhoz csak egy létező email kell. Többféle csomagot kínál a kezdőtől a haladóig. Mobil oldalról Android-ra és IOs-ra is van applikációjuk. Tárolja, hogy ki mit és mikor vett igénybe. Hátránya, hogy értesítéseket csak akkor küld, ha be vagyunk jelentkezve és nincs benne semmi féle ügyfélszokás figyelő. Esetlegesen felszabadult helyekről se értesíti a felhasználókat.[2]

2.4. ReservIO

Az egyik legnépszerűbb online időpontfoglalás a ReservIO használatával történik. Itt is több csomag közül választhatunk. Előnye, hogy SMS értesítéseket küld már az alapcsomag is, valamint van visszacsatolás a vevő és szolgáltató között. Hátránya, hogy a legtöbb hasznos funkcióért már többet kell fizetni, valamint, ha nem a legdrágább csomagot választjuk korlátozott a foglalások száma. Adatokat tárol viszont nem elemzi azokat, ezáltal nem is készít semmilyen ügyfélelemzést.[3]

2.5. Összegzés

Összefoglalva ezeket a jelenlegi lehetőségeket kimondhatjuk, hogy egyik megvalósítás sem figyeli az ügyfelek viselkedését, valamint kényelmetlenségekkel is járhatnak. A legnagyobb előnyük az aktív és gyors Support. Személyre szabhatóságuk elég korlátozott.

A fenti rendszerek jó tulajdonságait meghagyva, hiányosságaikat pótolva kialakítható egy olyan rendszer, amely teljesen kiszolgálja mind a felhasználói mind a szolgáltatói igényeket.

Összevetve az adatokat a fenti rendszerekből a következő táblázatot készítettem a lényeges információkat megjelenítve rajta. Mindegyik rendszereknek azt az előnyét emelem ki, amelye szerintem a legfontosabb.

Név	Előnyök	Hátrányok	Költségek
Telefon	Elterjedt, megszokott módszer	Manuális kezelés. Nincs adatelemzés Körülményes adatkezelés	„Ingyenes”
Salonic	Facebook integráció. Könnyű kezelni.	Nem lehet lemondani a foglalást. Nincs adatelemzés, és semmilyen viselkedés figyelés.	Ingyenes
Booked4us	Aktív támogatás. Elérhető a legtöbb platformon. Gyors és egyszerű.	Csak akkor küld értesítést, ha bejelentkezve marad a vevő. Nincs adatelemzés.	Alap: 2900 Ft/hó Haladó: 4900 Ft/hó
ReservIO	SMS-ben is értesít. Visszacsatolás vevő és szolgáltató között.	Nem figyeli a vevői viselkedést. A kisebb csomagok korlátozottak	Starter: 2399 Ft/hó Standard: 4500 Ft/hó Pro: 9000 Ft/hó

2.1 táblázat: Jelenlegi rendszerek összehasonlítása

3. Lehetséges megvalósítások áttekintése

3.1. Új rendszer megvalósításának lehetőségei

Ahhoz, hogy egy jól működő rendszert tudjunk létrehozni szükséges, hogy legyen egy saját információs rendszerünk. Figyelembe kell vennünk a jelenleg piacon lévő rendszereket és esetleges szolgáltatásokat, amiket igénybe vehetünk a saját rendszerünk létrehozása során.

3.1.1. Már meglévő rendszer továbbfejlesztése

A jelenleg piacon lévő rendszerek közül tudunk választani és azt tovább fejleszteni a saját igényeink szerint. Ez a megoldás akkor jó, ha a vállalt rendelkezik elég emberrel, akik üzemeltetni tudják azt, hiszen eltérő lesz az eredeti rendszertől, amit nyújtanak. Valamint feltétel, hogy a meglévő rendszer fejleszthető legyen.

3.1.2. Teljesen új rendszer fejlesztése alapjairól

Mivel egy weboldal üzemeltetéséhez szükséges egy szerver legalább, így kis vállalat esetén, ahol nem számít a rendelkezésre állás megoldható egy saját kivitelezésű webszerver, amit a vállalat emberi tartanak karban. Erre még fejleszteni kell egy weboldalt és egy adatbázisszervert és ezeket is karban tartani. Bármi hardveres vagy szoftveres hiba esetén összedőlne a rendszer, ami megzavarná a vállalat működését. Kockázatos megoldás kis vállalatoknak.

3.1.3. Új rendszer fejlesztése meglévő modulok bérlésével

Hogyha felhasználjuk azokat a meglévő szolgáltatásokat, amik segítségével össze tudunk rakni egy weboldalt akkor elérkeztünk a középúthoz. Bérelhetünk egy domain-t vagy szervert az oldalunknak amire aztán felépíthetjük a rendszerünket. Persze minél több modult bérlünk annál több lesz a kiadás havonta, viszont a kockázat csökken. Viszont ezek modulok költsége még mindig kevesebb mint a jelenlegi rendszereké.

3.1.4. Új rendszer fejlesztése modulok vételével

Ugyan az, mint a bérlés csak itt megvesszük a modulokat. Hátránya, hogy ha megvesszük őket nekünk is kell azokat üzemeltetni. Előnye viszont, hogy jobban a saját elképzeléseinkhez tudjuk igazítani a rendszert.

3.2. Az új rendszer bevezetése

Miután megvan a rendszerünk be is kell azt vezetni a vállalat életébe. A legfontosabb, hogy a változás eljusson az eddigi vendégeinkhez és hogy az alkalmazottaink megtanulják kezelni azt. A rendszerünk csak akkor lesz sikeres, ha minél több ember fog online foglalni ezért, ha eddig nem volt ilyen lehetőség reklámozni kell amennyire a vállalat megteheti.

3.2.1. Bevezetés lépési

Amennyiben működik az új rendszer elsőnek az összes ügyfelünket értesíteni kell róla. Ezzel igazából élesben is teszteljük mennyit képes bírni és még időben kijavítani az esetleges hibákat. Be kell tanítanunk az alkalmazottjainkat a rendszer használatára, ez, ha tényleg egyszerű és átlátható rendszert kaptunk akkor könnyedén fog menni. Ki kell emelnünk számukra, hogy nagyon fontos a rendszer használata, mert csak akkor fog jól működni, ha mindenki helyesen használja azt.

3.2.2. Az új rendszer bevezetésének buktatói

Lehetnek viszont buktatói az új rendszerünknek, ami miatt nem fogja visszahozni a pénzt, amit a cég rááldozott. A legnagyobb az, ha az emberek nem használják az online felületet, avagy nem jut el hozzájuk. A második, ha az alkalmazottak nem tudják használni és olyan hibákat vétenek vele, amik plusz kiadást generálnak. Valamint, hogyha nem használjuk ki a rendszer nyújtotta lehetőségeket teljes körben. Ha ezek nem következnek be akkor az új rendszer bevezetése sikeresen zárult.

3.3. Az új rendszer célja

A fő cél az, hogy egy olyan egységes rendszert nyújtsunk, amivel a dolgozók és vendégek is maximálisan elégedettek. Ezzel együtt az, hogy megismerjük az ügyfeleink szokásait.

Jelenleg az egyik legnagyobb hiányossága a rendszereknek az, hogy csak tárolják az adatokat, viszont nem használják azokat. Rengeteg kiaknázatlan lehetőség van, amivel növelhetjük a vásárlók elégedettségét. Figyelhetjük, ha egy hely megüresedik, vagy ha gyorsabban végez egy vendég. Értesíthetjük a vendégeket esetleges csúszásról is akár, mind annak köszönhetően, ha figyeljük a változásokat az adatbázisunkban. Megkönnyíti a munkáját az alkalmazottnak és időt spórolhat a vendégeknek. Ha egy alkalmazott túlterhelt a nem kifejezetten hozzá foglalt vendégeket szét tudjuk osztani a szabad alkalmazottak között egy pillanat alatt anélkül, hogy felborítanánk a rendszert. Valamint, ha bővülnénk egyszerűen megtehetjük plusz fejlesztés nélkül

4. Jelenleg elérhető szoftverek az új rendszer fejlesztéséhez

4.1. Hosting szolgáltatások

A hosting-on kívül szükség lesz még egy domain névre, amin keresztül elérik majd az oldalunkat az emberek. Ez egy-két ezer forintos összeg évente.

4.1.1. Szerver bérlés

Egy kényelmes megoldás, ha kibérelünk egy szervert. Ennek havi költségei vannak gyakran hűséggel együtt. Az üzemeltetést és bármiféle problémát a szolgáltatóra hárítunk ezzel. A rendelkezésreállás közelíteni fog a 100%-hoz és bármikor bővíthetjük azt a szolgáltató limitjeihez mérten persze.

4.1.2. Felhő alapú

Egy szintén havidíjjal rendelkező szolgáltatás igénybevétele, ha egy felhőt bérelünk ki. Előnye a szerverhez képest, hogy bárhonnan elérhető az adat, amit tárol. Hátránya, hogy jóval kisebb a tárhely, ami a rendelkezésünkre áll, valamint a le és feltöltési sebesség is korlátozottabb.

4.1.3. Saját szerver

Ahhoz, hogy saját szerverünk legyen kell egy szervergép. Amit a mai korban akár magunk is összerakhatunk, nyilván minél nagyobb a rendszerünk annál erősebb gép fog kelleni. Ha ezzel megvagyunk nekünk kell üzemeltetni is a szervert. Szóval, ha elmegy az internet vagy az áram a házunkban a szervereink is leállnak, emiatt kell egyfajta védelem is nekik, hogy ne veszítsünk adatokat és egy ilyen esetleges leállás után vissza tudjuk állítani a normális állapotukba.

4.1.4. Összegzés

A fenti megvalósítások összevetése:

Név	Előnyök	Hátrányok	Költségek
Szerver bérlés	Nem kell foglalkoznunk az üzemeltetéssel és a hibákat sem nekünk kell elhárítani. Magas rendelkezésreállás.	Korlátozott a változtatásokra.	20.000 Ft-tól akár 40.000-ig is
Felhő alapú	Bárhonnan elérhető.	Kevés tárhely.	4.000 Ft-tól 10.000-ig
Szerver építés	Teljesen személyre szabható.	Üzemeltetés nálunk van. Hibaelhárítás szintén.	Géptől függően változó. Valamint internet szolgáltatótól is.

	Bármikor bővíthetjük, vagy cserélhetjük.	Biztonsági kockázat	

4.1 táblázat: Hosting szolgáltatások összehasonlítása

4.2. Content Managment System

Röviden CMS. Ilyen rendszerek segítségével tehetjük sokkal egyszerűbbé a munkánkat, valamint segítenek, hogy a későbbi módosításokat a felhasználó is meg tudja csinálni.

4.2.1. Wordpress.org

Az egyik legjobban elterjedt nyílt forrású tartalomkezelő rendszer. Egyszerű drag and drop technológiával építhetünk fel benne weboldalakat. Rengeteg minta közül válogathatunk. A Wordpress.com-al ellentétben, ha ezt használjuk nekünk kell szolgáltatni a hosting-ot. [4]

4.2.2. Drupal

Egy másik elterjedt nyílt forráskódú CMS, viszont a flexibilitása, ami elválasztja a többitől; a modularitás a gerince. Ahhoz, hogy ezzel a CMS-el dolgozzunk szükségünk van PHP tudásra. PostgreSQL szervert használ, ami szintén nyílt forráskódú. [5]

4.2.3. Joomla

Teljesen ingyenes szintén viszont kevésbé elterjedt CMS. Könnyen belassulhat, ha túl sok kiegészítőt használunk. Adatbázisnak a MySQL szervert használ. Flexibilis viszont, ha SEO (Search Engine Optimization) vagy testre szabás szempontjából nézzük alulmarad a többivel szemben. [6]

4.2.4. Umbraco

.NET alapú CMS, kevésbé elterjedt viszont az egyik legjobban testre szabható. A használatához C# tudás szükséges. Alapvetően nyílt forráskódú viszont üzleti felhasználásra nem ingyenes. [7]

4.2.5. Összegzés

Név	Előnyök	Hátrányok	Költségek
Wordpress	Rendkívül könnyű használat. Legjobb SEO	Kevés kontroll	ingyenes
Drupal	Nagyobb kontroll.	PHP tudás szükséges. Megszűnőben lévő támogatás.	ingyenes
Joomla	Kezdő barát	Rossz SEO. Limitált testre szabás. Könnyen belassul.	ingyenes

Umbraco	Teljes kontroll minden felett.	C# tudás szükséges	5000$/év

4.2 táblázat: CMS összehasonlítása

4.3. Adatbázisok

Az adatok, amiket tárolni szeretnénk beleillenek egy relációs adatbázis környezetbe, így a NO SQL adatbázisokat kihagyom az összehasonlításból.

4.3.1. Microsoft SQL

Microsoft által fejlesztett MS SQL egy zárt forráskódú adatbázis-kezelő rendszer. Aktív támogatással rendelkezik, valamint beépített felhasználói dokumentációval, amivel a használata nagyon egyszerű. Van belőle fejlesztői kiadás ingyen, amin bemutathatjuk és tesztelhetjük a rendszerünket. A rendes szerverek viszont már fizetősek, de egy kis vagy közepes vállalatnak eléggé megfizethetetlenek.[8]

4.3.2. Oracle SQL

Az Oracle által fejlesztett adatbázis-kezelő rendszer, ami az egyik legjobban elterjedt SQL nyelvet használja. Itt is elmondható az aktív támogatás és jó dokumentáció. Nagyon stabil és viszonylag jól optimalizált. Viszont az árai a legmagasabbak, akár négyszerese a Microsoft termékeinek, ezért ez a „lehetőség” csak a nagy vállalatok számára igazi lehetőség.[9]

4.3.3. Postgre SQL

PostgreSQL Global Development Group által fejlesztett környezet, ami egy nyílt forráskódú rendszer, ami azt jelenti, hogy ingyenesen használható. Bár a nyílt forráskódnak vannak hátrányai is. A szoftvert a közösség „fejleszti” és készít hozzá dokumentációkat. Nagy előnye viszont, hogy nyitottsága miatt rengeteg segítséget találunk róla az interneten keresztül. [10]

4.3.4. Összegzés

A fenti adatbázis-kezelő rendszereket összesítve:

Név	Előnyök	Hátrányok	Költségek
MS SQL	Áttekinthető felhasználói dokumentáció. Aktív Support. Rengeteg modul bővítésnek.	Csak windows alapú környezetben futtatható. Drága a kis és közép vállalatoknak.	Enterprise $14,256 Standard $3,717 Standard-server+CAL $931
Oracle SQL	Rendkívül stabil. Nagy rendelkezésre állás.	Nagyon drága még nagy vállalati szinten is.	Enterprise Edition $47,500

	Elterjedt nyelvezet. Optimalizálás.		Standard Edition $17,500 Standard Edition One $5,800
PostgreSQL	Nyílt forráskódú. Támogatja a kliens-szerver hálózati architektúrát. A legtöbb operációsrendszeren futtatható	Még nem teljesen kiforrott rendszer.	Ingyenes

4.3 táblázat: Jelenlegi adatbáziskezelő rendszerek összehasonlítása

4.4. Keretrendszerek és nyelvek

Az összehasonlítandó nyelvek azok, amikkel az egyetemi tanulmányaim során megismerkedtem. Fontos szempont volt még, hogy támogassák a weblap készítést. Mindkét keretrendszer támogatja az MVC tervezési mintát. Mindkét keretrendszer nyelvei támogatják az agilis felépítést és az automatikus kódgenerálást.

4.4.1. .NET

Microsoft által fejlesztett jelenleg is fejlődő keretrendszer. Objektum orientált és több megvalósítási módszert is kínál webes felületekre. A keretrendszeren belül a C# nyelvvel ismerkedtem meg az egyetem alatt és ezt a nyelvet tanultam a legtovább is. Unit testek nagyszerűek a rendszerünk tesztelésére. A legtöbb adatbázis-kezelő rendszerrel ki tud alakítani kapcsolatot, amit LINQ nyelv használatával nagyon könnyen tudunk kezelni.[11]

4.4.2. Java

Jelenleg az Oracle által birtokolt keretrendszer. Szintén objektum orientált nyelv. A Netbeans, illetve Eclipse szoftverekkel ismerkedtem meg tanulmányaim alatt viszont mindkettővel csak egy félévet foglalkoztam. Kicsit bonyolultabb a felépítése, mint a .NET-es weboldalaknak viszont nagyobb a rálátásunk a teljes folyamatra ezáltal.[12]

4.4.3. Összegzés

A fenti keretrendszerek áttekintése:

Név	Előnyök	Hátrányok	Költségek
.NET	Agilis kezelhetőség. Gyors. Folyamatosan fejlődik.	Bővítmények kellenek, hogy effektíven tudjunk weblapot készíteni	Ingyenes

Java	Alapértelmezetten kezel json formátumot. Az egész a webprogramozás kiszolgálására épül.	Lassabb és jobban ki van téve biztonsági behatolásnak	Ingyenes

4.5 táblázat: Jelenlegi fejlesztői szoftverek összehasonlítása

4.5. Predikciós módszerek

Az eltárolt foglalási adatok alapján a program képes lesz megmondani azt, hogy várhatóan mennyi lesz az adott napra érkező foglalások száma, kihez és milyen szolgáltatást fognak várhatóan igénybe venni. Ezáltal képesek leszünk esetleg a túlterhelt alkalmazottról felszabadítani néhány foglalást természetesen, ha a vevő beleegyezik. A legfontosabb adatunk a foglalások között eltelt napok száma.

4.5.1. Átlag

Ha összeadjuk az értékeket és elosztjuk az elemszámmal akkor megkapjuk az átlagot. Igazából ez a legegyszerűbb módszer, viszont nem vesz figyelembe semmit. Ha az átlagtól való eltérés kicsi akkor használható.

4.5.2. Medián

Ha sorba rendezzük az értékeket visszaadja a középső elemet vagy a két középső elem átlagát. Mivel ez a módszer érték és nem idő szerint rendezi sorrendbe az elemeket így nem adna jó megoldást a problémára.

4.5.3. Módusz

Visszaadja a legtöbbször előforduló elemet. Mivel az ember nem pontosan ugyan annyi időközönként szokott foglalni így előfordulnak egy két napos eltérések, ezért ez a módszer sem a legjobb a problémánkra.

4.5.4. Lineáris regresszió

Azt feltételezi, hogy az értékeket el tudjuk helyezni egy koordináta rendszerben olyan módon, hogy létezik egy lineáris függvény, ami leírja az elemek elhelyezkedését. Megoldható lenne, ha az időközök összegét tekintenénk az értékeknek, viszont így az egyszerű átlag módszernél nem sokkal jobb ez a módszer.

4.5.5. Autoregresszív modell

Egy olyan modell, ami azt feltételezi, hogy a véletlen esemény csak az előzőleg felvett értékeitől függ, ezen kívül pedig az úgy nevezett fehérzajtól, ami a hibát fogja adni. Ezt úgy teszi meg, hogy egy adott méreten belül megnézi, hogy mennyire függenek az adott értékek

egymástól. Ez a módszer nagyon jó, ha egy bizonyos mintát követnek a foglalások, viszont minél véletlenszerűbb annál pontatlanabb értéket ad.

4.5.6. Mozgó átlag folyamat

Hasonlóan az átlaghoz egy adott érték halmaz átlagát adja meg, viszont nem az összes érték alapján. Elsőnek kiválaszt egy méretet, ami megadja, hogy hány elem átlagát veszi bele a számításba, így „mozog" az érték halmazon ahogy egyre több értékünk van. Ennek is van egy alfajtája a súlyozott mozgó átlag folyamat, ami már egy szorzót rendel minden értékhez a méreten belül így az elhanyagolható értékek kisebb szorzót kapnak és kevésbé változtatják majd meg a predikált értéket.

4.5.7. Autoregresszív mozgó átlag modell

Ez a modell az előbbi kettő kombinációja. Két bemenő paraméterrel rendelkezik, ami megadja a két folyamat méretét. Hogyha az egyik paraméter nulla, akkor értelemszerűen csak az egyik folyamat képletét használja. Egyetlen hibája, hogy ha az átlagtól való eltérés nulla akkor nem működik viszont ezt könnyen ki lehet küszöbölni. Ez megoldja a sima autoregresszív modell problémáját a véletlenszerű foglalásoknál, valamint finomítja azt.

4.5.8. Autokorreláció

Egy fontos mutató, hogy ha az autoregresszív mozgó átlag modellt szeretnénk használni, mivel ez fogja megadni a bemenő paraméterét a mozgó átlag résznek. Ez egy olyan eljárás, ami megmutatja milyen kapcsolatban állnak az egymást követő értékek, viszont a közvetlen ráhatáson kívül még azt is számba veszi, hogy esetleg több értéken át milyen hatása van az adott értéknek. Például, hogy az ötödik érték milyen hatással van a hetedikre a hatodik értéken keresztül.

4.5.9. Parciális autokorreláció

A parciális autokorrelációs eljárás során csak azt vizsgáljuk, hogy mekkora a közvetlen hatása az adott értékeknek egymásra. Például mekkora hatással van az ötödik elem a hatodikra, a hatodik a hetedikre és így tovább. Ez az eljárás fogja megadni az autoregresszív folyamat paraméterét.

4.5.10. Összegzés

A predikciókat az autoregresszív mozgó átlag modell segítségével fogom megvalósítani, mivel ez fedi le a legjobban a problémát. Viszont csak akkor fogom használni, ha az átlagtól való átlagos eltérés kevesebb mint kettő nap. Erre azért van szükség, mert kis eltérés esetén az átlagtól rosszabb becslést ad a modell, nulla eltérésnél pedig nem is lehet használni.

5. Követelmény specifikáció

5.1. Áttekintés

A megvalósítandó rendszer célja, hogy az időpontfoglalás folyamatait teljes körben kiszolgálja, a vendégek és az alkalmazottak részéről egyaránt. Az oldalt bárki el tudja érni az interneten bármilyen eszközt is használ. Fontos, hogy rendelkezésre álljon a nap nagyrészén, hiszen az alkalmazottak itt látják mikor kell dolgozniuk. Az adatok módosítása azonnal meg kell jelenjen és értesíteni kell a hozzátartozókat amennyiben szükséges.

A rögzített adatokról automatikus kimutatást kell csinálni, ami heti vagy havi bontásban megjelenik, valamint az elemzéseknek pontosnak és áttekinthetőnek kell lenniük. Minden információ, amit tárolunk naprakésznek kell lennie. Vevői oldalról a foglalásnak kényelmesnek és egyszerűen történjen. Lehetőség legyen módosítani és törölni megadott időn belül.

5.2. Felhasználói követelmények

5.2.1. Vendég

- Képes időpontot foglalni
- Képes a foglalását megtekinteni
- Képes a foglalását lemondani
- Képes a foglaláshoz megjegyzést fűzni
- Kap értesítést a foglalásról

Igényeik:

- Értesítés leárazásról
- Egyszerű és kényelmes foglalás

5.2.2. Szolgáltató

- Képes megadni a beosztását
- Képes megadni az általa kínált szolgáltatásokat
- Képes módosítani a szolgáltatásain
- Képes törölni a szolgáltatásait
- Képes megnézni a vendégei adatai

- Képes visszaigazolni foglalást
- Képes lemondani foglalást
- Képes foglalást rögzíteni
- Értesítést kap, ha valaki hozzá foglalt
- Értesítést kap, ha lemondják a foglalást

Igényeik:

- Könnyű használat
- Átlátható beosztási rendszer
- Ügyfélprofilok megismerése

5.3. Adminisztrátor

- Képes a szolgáltatásokat kezelni
- Képes a foglalásokat kezelni
- Képes az alkalmazottakat kezelni
- Képes a weboldalt kezelni
- Értesül, ha bármi hiba van a rendszerrel

Igényei:

- Rendszer agilis kezelhetősége
- Tovább fejleszthetőség lehetősége

5.4. Funkcionális követelmények

5.4.1. Autentikáció

Az oldalra bárki beléphet. és időpontot tud foglalni vagy lemondani azt, hogy ha nem aznap van. Az admin felületre csak jelszóval lehet belépni, amit az admin meg tud változtatni.

5.4.2. Szolgáltatások kezelése

Az admin meg tud adni új szolgáltatásokat, ehhez szükséges egy szolgáltatás név, ami alapján majd a vevők tudnak választani egy ár és egy időtartam, ami a szolgáltatás idejét adja meg. Valamint tudja ezeket az adatokat módosítani, illetve törölni is.

5.4.3. Foglalások kezelése

Ha létrejött egy foglalás akkor tudnunk kell kezelni azt. Az alkalmazott is le tudja mondani és a vevői is, a lemondásról pedig mindkét felt értesíteni kell. Valamint főként ezek alapján tudunk vevő csoportokat kialakítani így az elemző szoftverünknek is hozzá kell férnie ezekhez az adatokhoz. Mindig naprakésznek kell lennie és minden módosítást azonnal meg kell jelenítsen. Azt is meg kell oldani, hogy az alkalmazottak tudjanak telefonszám alapján foglalást rögzíteni a rendszerben.

6. Megvalósítási módszer kiválasztása

Figyelembe véve, hogy a rendszert főleg kis és középvállalkozások, valamint magánvállalkozók veszik majd igénybe így a legoptimálisabb megoldás egy saját rendszer fejlesztése.

A megoldás kiválasztása során mindenhol az árat vettem figyelembe és ha lehetett ingyenes szoftvert használtam. A tanulmányaim során a C# nyelvvel ismerkedtem meg, így a programot is ezen a nyelven írtam meg. Ez egy fejlődő objektum orientált nyelv, és figyelembe véve azt, hogy az emberek nagytöbbsége Microsoft operációs rendszerű számítógépet használ még optimális is ebben a rendszerben írni a szoftvert mivel, az operációs rendszer támogatja azt. A fejlesztőkörnyezete a Visual Studio is rengeteg lehetőséget nyújt más IDE1-khez képest, ezzel felgyorsítva a fejlesztési folyamatot.

A szoftver ASP.NET környezetben készült, ami egy Microsoft által fejlesztett keretrendszer, ami implementálja a Model-View-Controller (MVC) tervezési mintát.

A technológia mellett szóló érvek a következők:

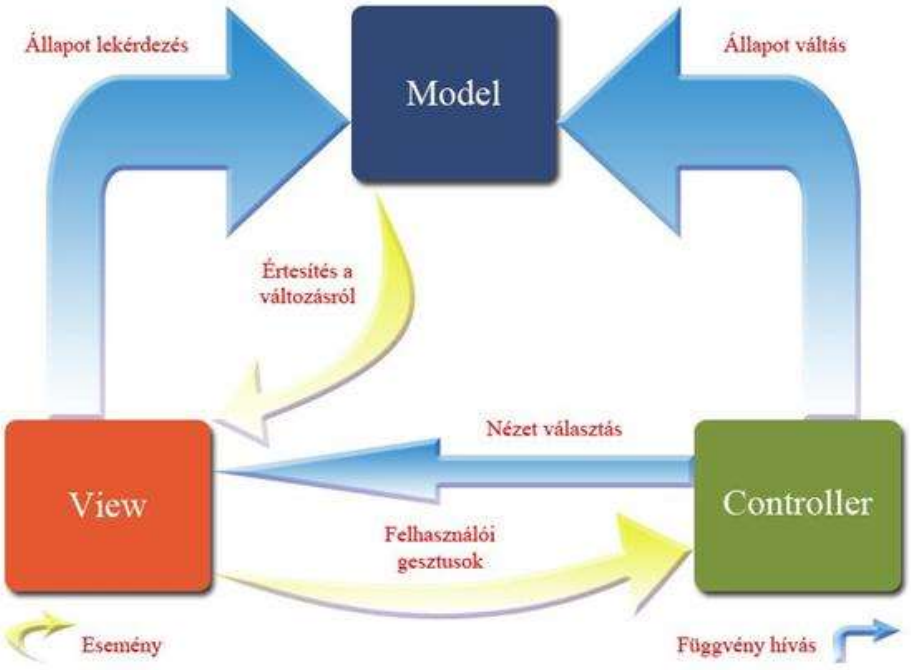
- nyílt forráskódú
- folyamatosan fejlődő
- könnyen bővíthető
- átlátható
- számos beépített funkcióval rendelkezik.

Az MVC részei:

- Model: az adatok tárolásáért felel
- View: a megjelenítésért felel
- Controller: az adatok kezelésére és a kommunikációért felel

Az architektúra teljes felépítése az 8. ábrán látható

1 Integrated Development Environment: Fejlesztőkörnyezet

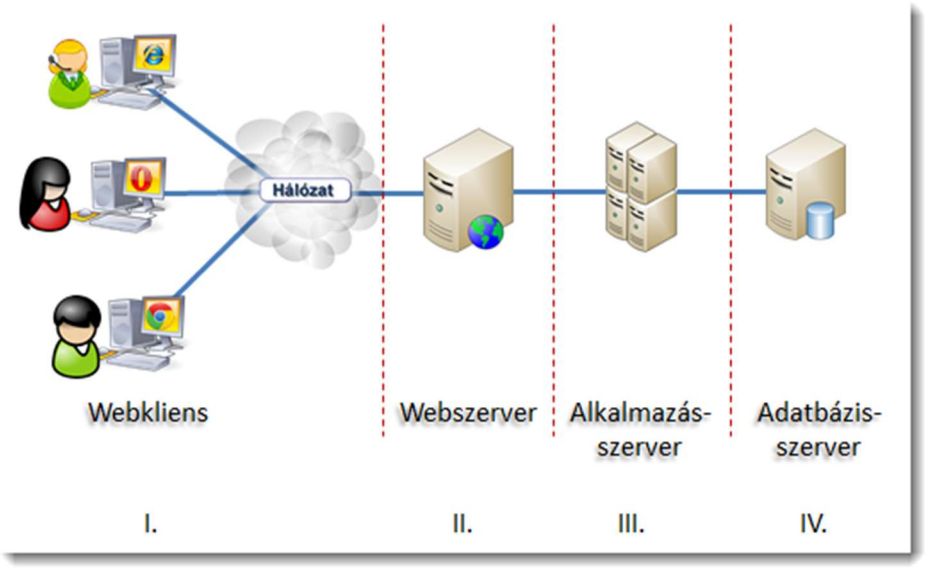


6.1 ábra: MVC működése

Az adatok tárolására pedig a PostgreSQL rendszerét választottam, mivel teljesen ingyenes, még üzleti felhasználásra is, valamint könnyen össze lehet kötni a .NET-el.

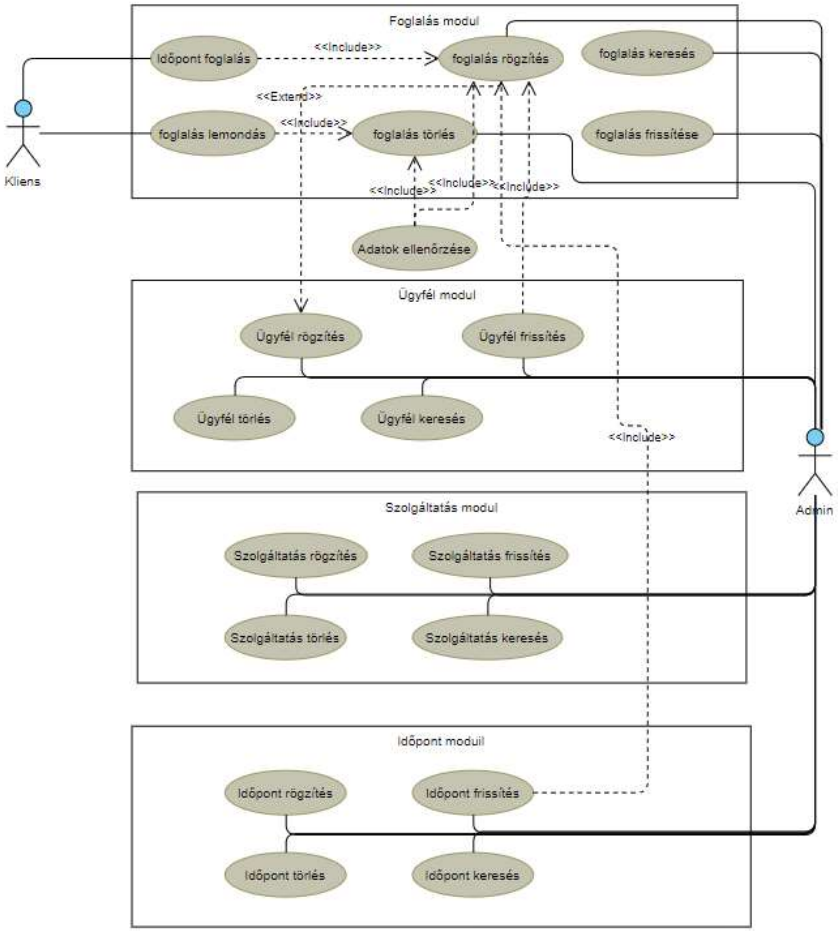
7. Részletes rendszerterv

Hogy az admin felületet bárhonnan el tudja érni a szolgáltató így az is, mint a foglalás a weben lesz. Ehhez szükségünk van egy olyan eszközre, ami összeköttetést ad a felhasználói gép és a program között. A program négyrétegű architektúrát használ, ahol a kliens a böngésző segítségével tud feladatokat végrehajtani. Az alkalmazás az alkalmazásszerveren fut, ennek a kezelőfelülete ugyanúgy a böngészőből érhető el, a webszerver biztosítja a köztük lévő kapcsolatot, valamint az adatbázis szerver az, ahol a program által használt adatokat tároljuk. Az architektúrát az a 9. ábra szemlélteti [9]:



7.1 ábra: Rétegek bemutatása

A szoftverben négy nagy modul van, más-más folyamatokkal. Azt, hogy melyik modulokhoz mely szerepkörök férnek hozzá, és milyen folyamatokat lehet bennük megtenni, azt 10. ábrán szereplő használati-eset diagram mutatja. Az ábrát az **online.visual-paradigm.com** oldal segítségével készítettem.

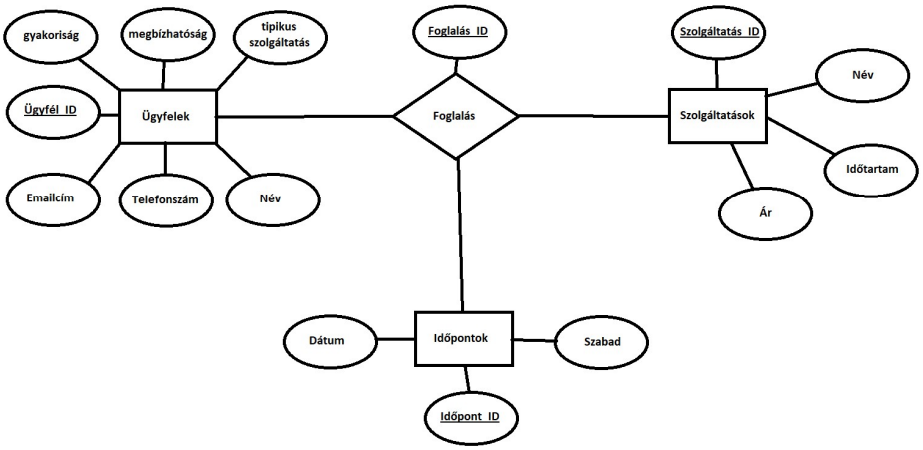


7.2 ábra: Use Case modell

7.1. Adatbázis tervezés

A specifikáció alapján meghatározhatók voltak az egyedhalmazok, az adatbázisban, amik a foglalás, időpont, ügyfél és szolgáltatás nevet kapták. Kapcsolataikat és a hozzájuk tartozó tulajdonságokat egy EK diagramban2 ábrázoltam, ennek segítségével átláthatóbb volt az adatbázis szerkezete. [10]

2 Egyed-kapcsolat diagram (Entity-Relationship diagram)



7.3 ábra: E/K diagramm

A Szolgáltatás tábla tartalmazza az üzletben elérhető szolgáltatások adatait. Az Időpont tábla a szabad és foglalt időpontokat, ami alapján tud foglalni a vevő. Az Ügyfelek tábla pedig az eddig nálunk valaha foglalt vevőket tartalmazza, amihez a személyes adataikon kívül még néhány vevőszokási tulajdonság tartozik. És végül a foglalás tábla, ahol az eddigi három tábla adatait felhasználva létrejön egy foglalás.

A fent említett táblák mezőinek magyarázata:

7.2. Táblák

7.2.1. Időpontok tábla

Név	Leírás	Típus
ido_id	az időpont egyedi azonosítója	serial
ido_datum	az időpont dátuma	DateTime
ido_szabad	az időpont foglaltságát jelzi	number

7.4 táblázat: Időpontok tábla

7.2.2. Szolgáltatások tábla

Név	Leírás	Típus
szolg_id	a szolgáltatás egyedi azonosítója	serial
szolg_nev	a szolgáltatás neve	text

szolg_ido	a szolgáltatás időtartama	number
szolg_ar	a szolgáltatás ára	number

7.5 táblázat: Szolgáltatások tábla

7.2.3. Foglalások tábla

Név	Leírás	Típus
foglalas_id	a foglalás egyedi azonosítója	serial
foglalas_nev	a foglaló neve	text
foglalas_email	a foglaló e-mail címe	text
foglalas_telefon	a foglaló telefonszáma	text
szolg_nev	a foglalt szolgáltatás neve	text
ido_datum	a foglalás időpontja	DateTime
foglalas_megjelent	a foglaláson a vevő megjelenését jelzi	number

7.6 táblázat: Foglalások tábla

7.2.4. Ügyfelek tábla

Név	Leírás	Típus
ugyfel_id	az ügyfél egyedi azonosítója	serial
ugyfel_nev	az ügyfél neve	text
ugyfel_email	az ügyfél e-mail címe	text
ugyfel_telefon	az ügyfél telefonszáma	text
ugyfel_gyakorisag	az ügyfél foglalási gyakorisága	text
ugyfel_megbizhatosag	az ügyfél megbízhatósága	text
ugyfel_tipikus_szolg	az ügyfél legtöbbször választott szolgáltatása	text

7.7 táblázat: Ügyfelek tábla

7.2.5. Dolgozó tábla

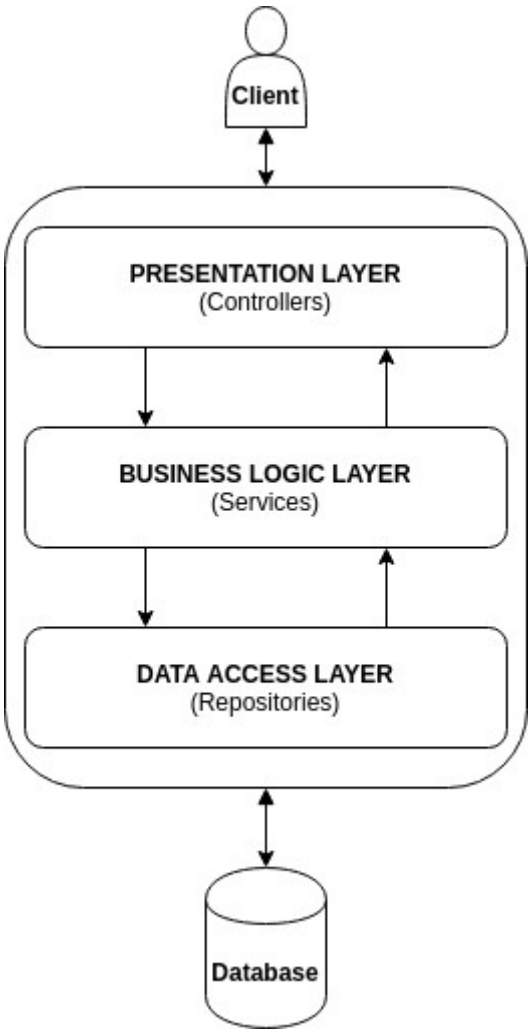
Név	Leírás	Típus
dolgozo_id	A dolgozó azonosítója	serial
dolgozo_nev	A dolgozó felhasználó neve	text
dolgozo_jelszo	A dolgozó jelszava	text

dolgozo_szolgaltatasok	Tömb ami tartalmazza a dolgozó által nyújtott szolgáltatások azonosítóját	number[]

7.8 táblázat: Dolgozók tábla

7.3. Program tervezés

A programomban három fő réteg lesz. Amiknek a tartalmát lejjebb ki is fejtem. Mivel az adatbázisunk nem a programon belül lesz, hanem kívül ezért a Repository réteg fogja elérni az adatbázis szervert.



7.9 ábra: Rétegek működése

7.3.1. Repository réteg

Ennek a rétegnek a feladata lesz az adatbázis elérése és az SQL műveletek végrehajtása. Mivel szövegként kell majd továbbítanunk a műveleteket ezért szükségünk lesz egy olyan osztályra, ami ezeket a lekérdezéseket generálja. Ezen belül kettő olyan folyamatra, amik

létesítik a kapcsolatot az adatbázis szerverrel és eljuttatják a lekérdezést. Azért kell kettő mivel, ha frissítünk, beillesztünk vagy törlünk az adatbázisból az nem jár visszatérési értékkel, míg, ha kiolvasunk az igen.

Ebben a rétegben fogom létrehozni azokat az osztályokat, amik az adatbázisban szereplő egyedeket képviselik. Hogyha lekérdeztük az adatokat az adatbázisból akkor ilyen osztály típusú listákat fogunk létrehozni. Mivel az adatbázisból jövő adatok nem lesznek típusosak azért minden ilyen osztálynak kell egy olyan konstruktor is, amivel könnyedén létrehozhatjuk a nyers adatokból.

Magát a Repository réteget egy interface-en keresztül lehet majd elérni. Lesz egy ős Repository osztály és minden egyed osztályhoz tartozni fog egy külön al Repository, ami az ősnek lesz a leszármazottja. Erre azért van szükség mert minden táblából más-más módon szeretnénk majd adatokat lekérdezni.

7.3.2. Logic réteg

Ez a réteg fogja összekötni a webes felületet a Repository réteggel, ami majd eljuttatja a kéréseinket az adatbázishoz. Ezt szintén egy interface-en keresztül lehet majd elérni.

Itt lesznek pluszba azok a műveletek, amik nem kapcsolódnak az adatbázishoz. Az egyik ilyen az e-mail küldés, amihez szükségünk lesz egy google fiókra. Ahhoz, hogy a programunk tudjon majd e-mail-eket küldeni, majd be kell kapcsolnunk a biztonsági beállításoknál a nem google-hoz tartozó kevésbé biztonságos alkalmazások hozzáférését az email címünkhöz. Ez azt fogja eredményezni, hogy bármilyen program, aminek megvan az e-mail címünk és jelszavunk be tud majd lépni a fiókunkba.

Ezen kívül itt lesznek azok a folyamatok is, amik az ügyfél szokásait elemzik. Emiatt kell majd egy olyan művelet, amivel le tudjuk kérdezni adott vevő összes eddigi foglalását. Három tulajdonság szerint elemzünk majd:

- Megbízhatóság
- Foglalási gyakoriság
- Tipikusan igénybevett szolgáltatás

A megbízhatóságot az alapján fogja számolni, hogy a vevő a foglalásai hány százalékán jelent meg.

Arány	Érték
50% alatt	Megbízhatatlan
51% és 75% között	Kevésbé megbízható
76% és 95% között	Általában megbízható
96% fölött	Megbízható

7.10 táblázat: Megbízhatósági táblázat

A gyakoriságot az alapján fogjuk számolni, hogy a vevő az első foglalása és a mai nap között milyen időközönként foglalt. Ha megvan az összes foglalás ezt könnyen lekérdezhetjük majd egy LINQ művelettel, csak sorba kell rendezni dátum szerint és az első elemet kivenni. Ha ez megvan kiszámoljuk hány nap telt el azóta és mennyiszer foglalt. Eltelt napok/ összes foglalás

Arány	Érték
kisebb vagy egyenlő mint 7	Hetente
kisebb vagy egyenlő mint 31	Havonta
nagyobb mint 31	Véletlenszerű

7.11 táblázat: Gyakorisági táblázat

A tipikusan igénybevett szolgáltatás már egyszerűbb, csoportosítjuk a foglalásokat a foglalt szolgáltatás neve szerint és kiválasztjuk azt, amit a legtöbbször vett igénybe a vevő.

Hogy a beosztást ne egyesével kelljen megadni létre fogok hozni egy olyan metódust, ami egy dátum, valamint egy kezdő és vég óra segítségével létrehozza aznapra a beosztást.

Végül pedig itt lesznek azok a folyamatok, amik az időpontfoglalást szolgálják ki. Mivel csak olyan időpontra lehet foglalni, ami szabad és ami legalább egy nappal később lesz ezért kell majd egy ilyen szűrés is. Viszont figyelembe kell venni azt is, hogy minden szoláltatásnak más-más az időtartama így bele kell építenünk egy olyan részt, ami ezeket az időközöket szedi ki az összes szabad időpont közül. Ha pedig a foglalás sikeres akkor meg kell hívni a foglalás rögzítése után egy olyan folyamatot, ami a beosztásban foglaltra állítja azokat az időpontokat, amiket a szolgáltatás ideje lefed.

7.3.3. Web réteg

Mivel az MVC mintát választottam ezért a Web rétegnek ez a három fő része lesz. Minden a Repository-ban szereplő egyed osztályhoz létre kell hozni egy hasonlót a Model részben. Valamint itt lesz egy olyan osztály, ami a Repository-ban lévő osztályokká alakítja a Model-ben lévő osztályokat, hogy könnyedén tudjuk átadni a Logic rétegnek az adatokat. A Model részben lesznek még a ViewModel-ek, amik csak arra a célra fognak szolgálni, hogy az adott oldalon könnyen tudjunk adatokat továbbítani a Controller-ek felé.

A második nagyobb rész a Controller-ek, amik irányítják majd az oldalakat. Minden főbb típus kap egy Controller-t.

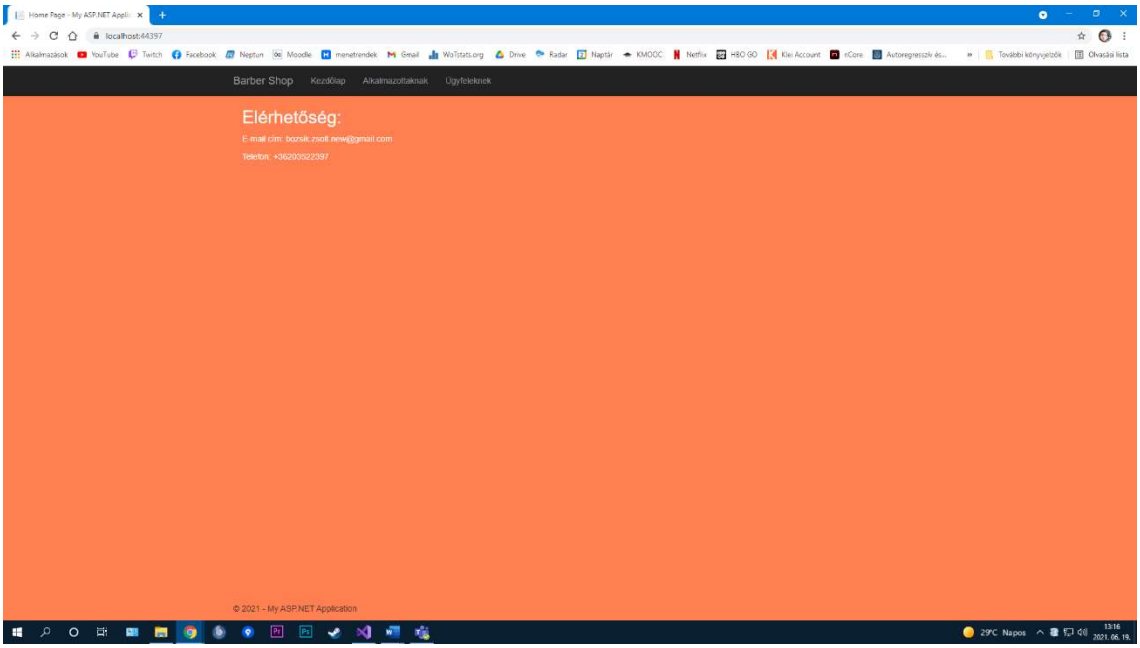
Végül a harmadik főbb rész a View, avagy nézetek. Az admin által elérhető adatbázis műveletekhez tartozó nézetek általában hárman lesznek egy, ami a betöltést szolgálja, egy a szerkesztés/hozzáadást és egy a megjelenítést. A beosztásnál a megadás és szerkesztés külön lesz, hogy napra is lehessen beosztást megadni. Mivel a statisztikát nem szerkeszthetjük, mert azzal átvernénk saját magunkat, ezért ott csak megjelenítésre

szolgálnak a nézetek, valamint szűrésekre. Magához a foglaláshoz pedig tartozni fog egy olyan nézet, ahol a szolgáltatást választják ki, egy ahol az időpontot és végül ahol megadják az adataikat. A kliensek számára elérhető nézetek még a lemondás.

8. Implementáció

8.1. Adatbázis megvalósítása

Első lépésként magát az adatbázist készítettem el az E/K diagram alapján, a pgAdmin 4 segítségével. Az alkalmazás a PostgreSQL szerverek, valamint az adatbázisok menedzselésére használható. Nagyon egyszerűen létrehozhatunk új táblákat, valamint ezeket közben tudjuk módosítani is. Az adatbázisom nevének a FoglalasData nevet adtam majd létrehoztam benne a tábláimat.



8.1 ábra: Kezdőlap

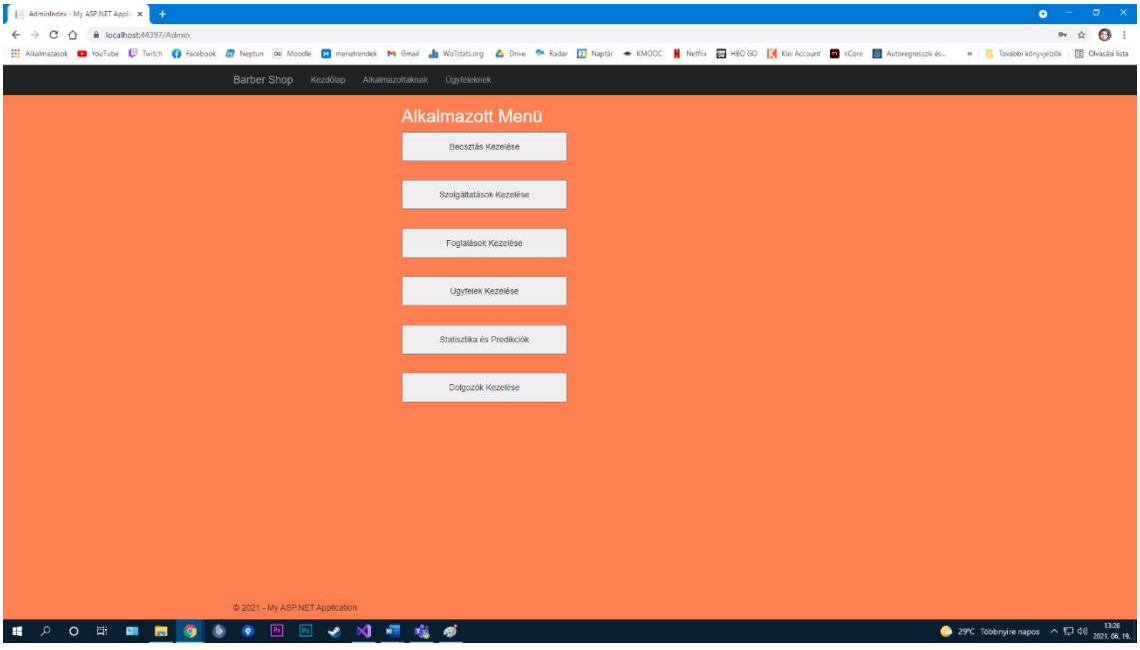
Az adatbázist, elkészülte után, a programmal kellett összekapcsolnom. Létrehoztam egy új projektet - ASP.NET MVC-t. Ezután egy Repository és egy Logic réteget. Mivel a PostgreSQL szerverrel nem lehet ADO.NET kapcsolatot létesíteni ezért egy connection-string-re lesz szükségünk. Mivel a lekérdezéseket szöveges formában juttatjuk el az adatbázishoz ezért a Repository rétegben létrehoztam egy SQLMaker osztályt, ami segít ezeknek a parancsoknak a létrehozásában. Mivel vannak olyan folyamatok, amiknek van visszatérése és olyanok, amiknek nincs ezért elsőnek létrehoztam egy metódust, ami a visszatérési érték nélküli folyamatokat kezeli, valamint egy függvényt, ami visszatérési értékkel rendelkező folyamatokat kezeli. Az MVC rétegben a Model részében létrehoztam azokat a modelleket, amik az adatbázisban szerepelnek. Valamint a Repository rétegben is ezekhez a megfelelő modelleket. Minden CRUD művelet a Repository rétegben van míg az adatelemző és egyéb metódusok a Logic rétegben találhatóak.

```
35 references
public class SQLMaker
{
    6 references
    public static void CUD_Action(string commandtext)
    {
        NpgsqlConnection conn = new NpgsqlConnection("Server=localhost;Port=5432;Database=FoglalasData;User Id=postgres;Password=Dota2");
        conn.Open();
        NpgsqlCommand comm = new NpgsqlCommand();
        comm.Connection = conn;
        comm.CommandType = CommandType.Text;
        comm.CommandText = commandtext;
        comm.ExecuteNonQuery();
        comm.Dispose();
        conn.Close();
    }

    13 references
    public static DataTable Read_Action(string commandtext)
    {
        DataTable dt = new DataTable();

        NpgsqlConnection conn = new NpgsqlConnection("Server=localhost;Port=5432;Database=FoglalasData;User Id=postgres;Password=Dota2");
        conn.Open();
        NpgsqlCommand comm = new NpgsqlCommand();
        comm.Connection = conn;
        comm.CommandType = CommandType.Text;
        comm.CommandText = commandtext;
        NpgsqlDataAdapter nda = new NpgsqlDataAdapter(comm);
        try
        {
            nda.Fill(dt);
        }
        catch (Exception)
        {

        }

        comm.Dispose();
        conn.Close();

        return dt;
```

8.2 ábra: Adatbáziskapcsolat

8.2. Admin

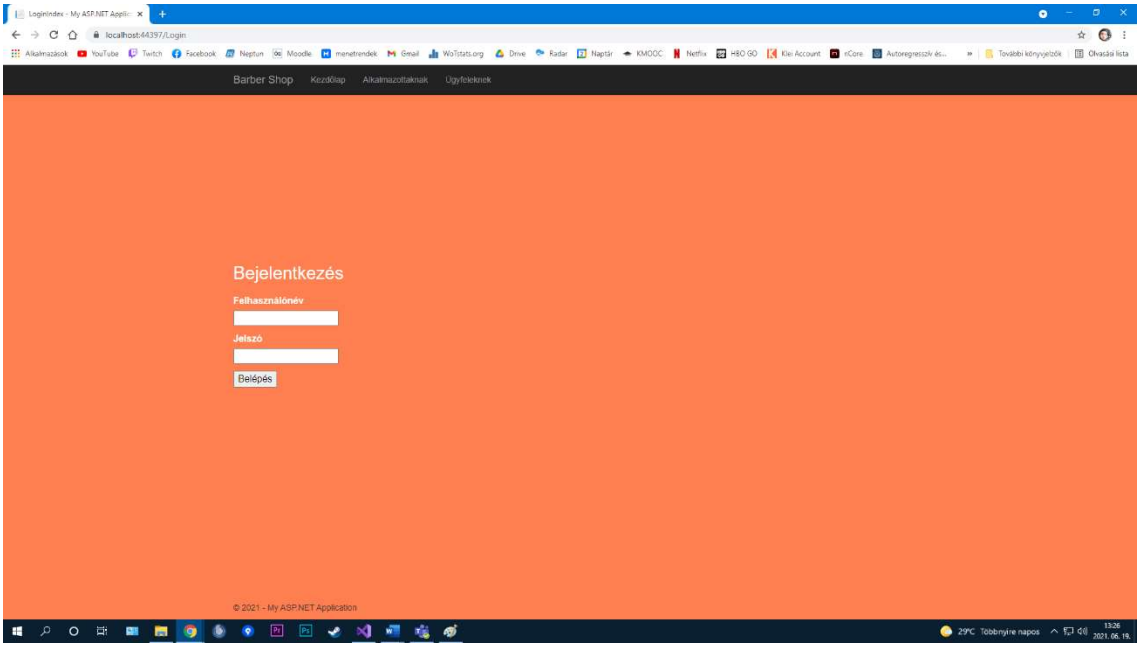
Az Admin az összes adattáblát tudja szerkeszteni így, ha belép egy menü köszönti, amiben kiválaszthatja, hogy szerkeszteni akar vagy a statisztikákat megtekinteni. Minden almodulhoz tartozik egy Controller és egy ViewModel, ami a könnyebb szerkeszthetőségért felel. A View-k a következőképpen néznek ki: elsőnek van egy lehetőség visszatérni az admin felületre majd ezután a szerkesztési/megadási felület végül a lista, az utóbbi kettő PartialView-ként van az oldalon.



8.3 ábra: Alkalmazotti menü

8.2.1. Bejelentkezés

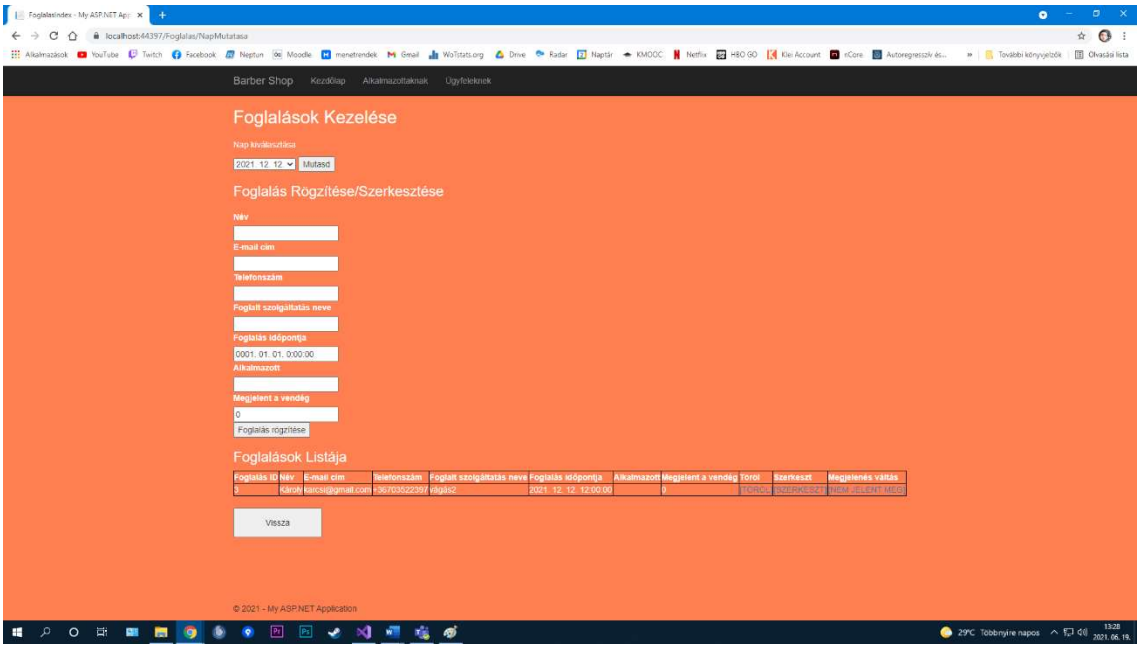
Ahhoz, hogy az admin felülethez hozzáférjünk rendelkeznünk kell egy rögzített fiókkal amit más alkalmazott tud felvenni. Ha ez megvan akkor egy felhasználónév jelszó párossal be is tudunk lépni a felületre.



8.4 ábra: Alkalmazotti kezdőlap

8.2.2. Foglalások

Az első ilyen a foglalások, amit a FoglalasController irányít. A FoglalasViewModel segítségével, ha a listában az „[SZERKESZT]” cellába kattint adott foglalásnál akkor behozza a foglalás adatait és a szövegdobozok segítségével könnyedén át tudja írni ezzel megváltoztatva az adatt foglalás értékeit. Az utolsó oszlop azért felel, hogyha a vevő nem jelenik meg az időpontban ekkor a Megjelent értékét megváltoztatja egy kattintással az Admin. A „[TÖRÖL]” cella pedig a törlésért felel, ha rákattintanak akkor az abban a sorban lévő foglalás törlődik a rendszerből. Ha egy foglalást mi szeretnénk rögzíteni akkor könnyen megtehetjük csak írjuk be az adatokat az üres cellákba majd küldjük el.

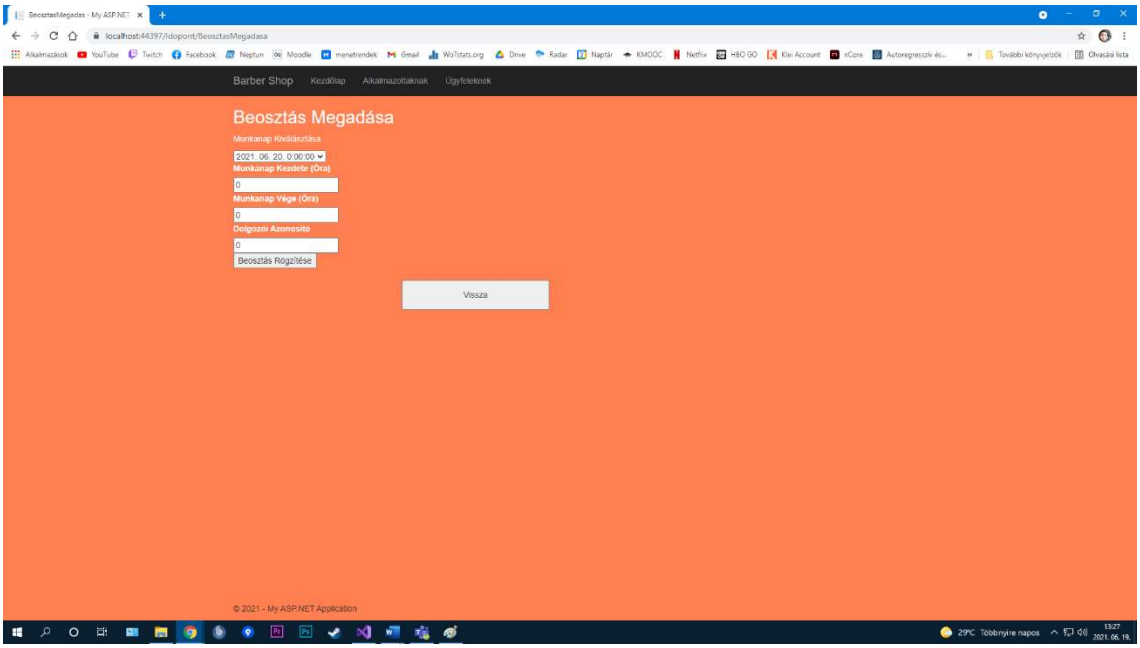


8.5 ábra: Foglalások kezelő felülete

Hogyha egy ügyfél nem jelenik meg és ezt felviszik a rendszerbe akkor a Logic réteg újra számolja a vevő megbízhatóságát és megváltoztatja azt az Ügyfél táblában.

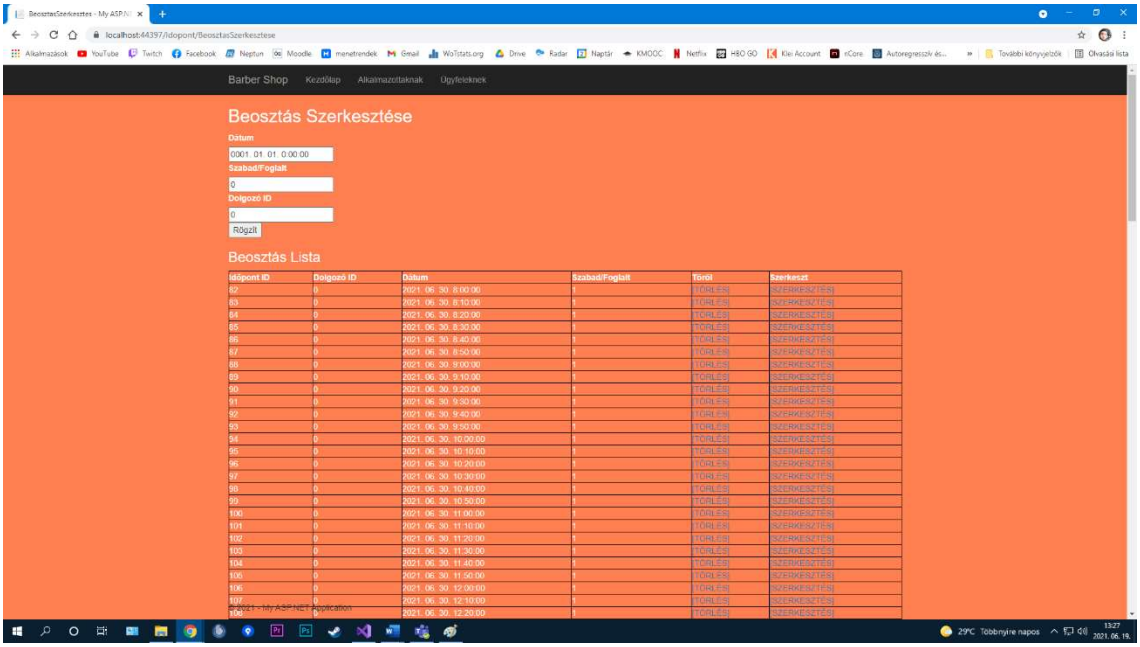
8.2.3. Beosztás

Mivel a vevők csak akkor tudnak foglalni, ha van megadott szabad időpont így az Időpont menü alatt két lehetőség van, amiből az egyik az új beosztás megadása. A legördülő cellában csak azok a napok szerepelnek, ahol még nincs megadva beosztás. Hogy ezt elérjem létrehoztam egy olyan lekérdezést, ami a következő hatvan nap dátumaiból kiszűri azokat, amik nem szerepelnek az időpontok között. A beosztás a kezdőóra és a végóra közti idő lesz tíz perces osztásokban. Ezt az IdopontController kezeli és az IdopontViewModel segít az adatok továbbításában. A vég és kezdő óra számát, valamint a kiválasztott napot szintén a ViewModel-ben tárolom így mikor megadják azt a Controller továbbítani tudja a Logic-nak, ami a tízperces osztásokért felel és azt a listát a Repository-nak átadva rögzíti az adatbázisban. Alapértelmezetten a megadott beosztás minden időpontja szabad.



8.6 ábra: Beosztás megadása

Hogyha egy naphoz van megadott beosztás és szerkeszteni akarják akkor azt a Beosztás Szerkesztésé menüpont alatt tudják megtenni. Ugyan úgy ahogy a Foglalásnál itt is lehet törölni.



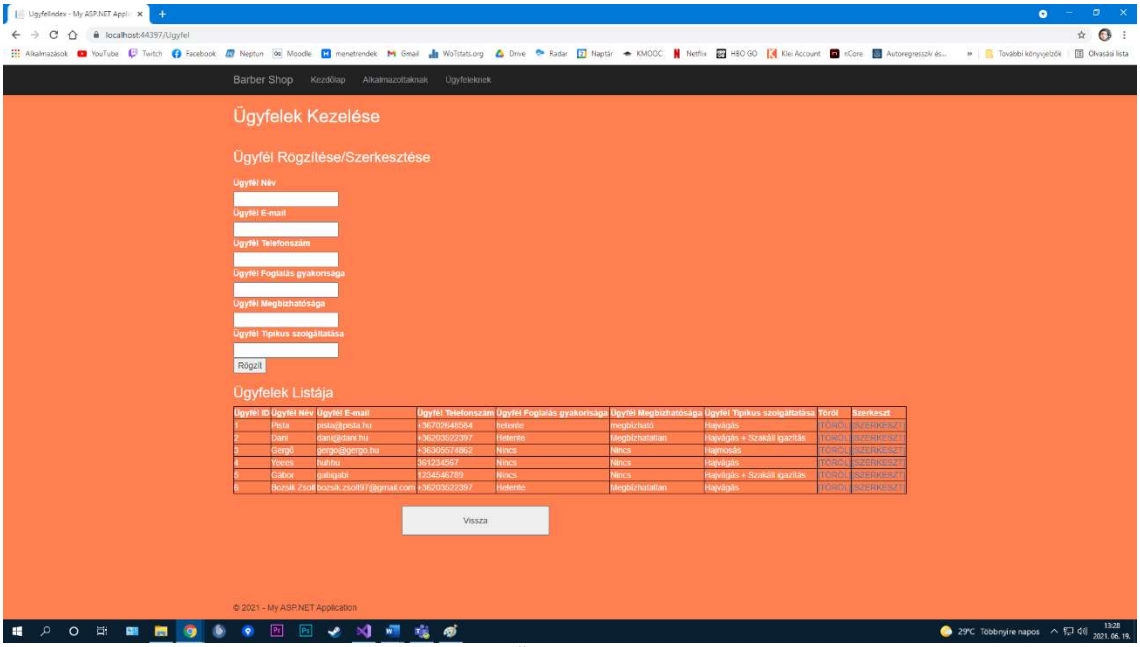
8.7 ábra: Beosztás szerkesztő felület

8.2.4. Szolgáltatások

A szolgáltatásokat a SzolgaltatasController kezeli és a SzolgaltatasViewModel segít az adatok tárolásában. Ahogy feljebb itt is lehetséges új szolgáltatás megadására, régi módosítására és törlésre is.

8.2.5. Ügyfelek

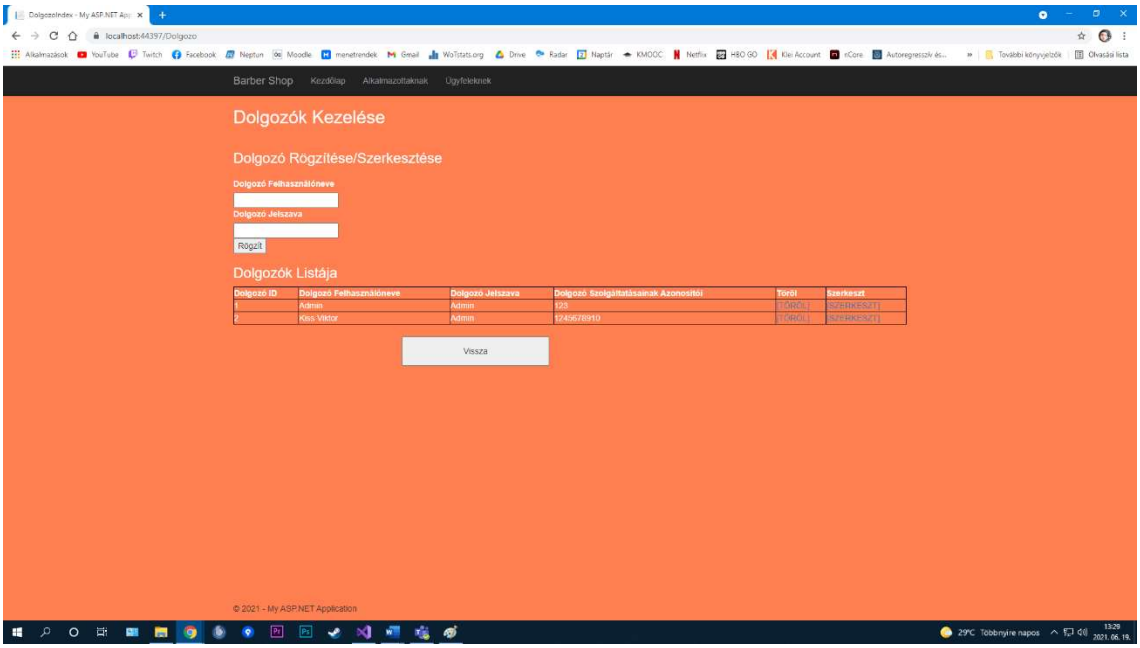
Az ügyfelek személyes adatait a foglaláskor adják meg, viszont a szokásaikat már a rendszer teszi hozzá és ha foglalnak vagy nem jelennek meg akkor változtatja azt.



8.8 ábra: Ügyfeleket kezelő felület

8.2.6. Dolgozók

Hasonlóan a többihez a dolgozókat is tudjuk szerkeszteni, valamint felvenni és törölni is. Az azonosítót automatikusan generálja a rendszer, esetleg a jövőben, ha nagyobb szerepe lesz lehetőséget kell nyújtani a megváltoztatására, de csak az Admin csoport tagjainak.



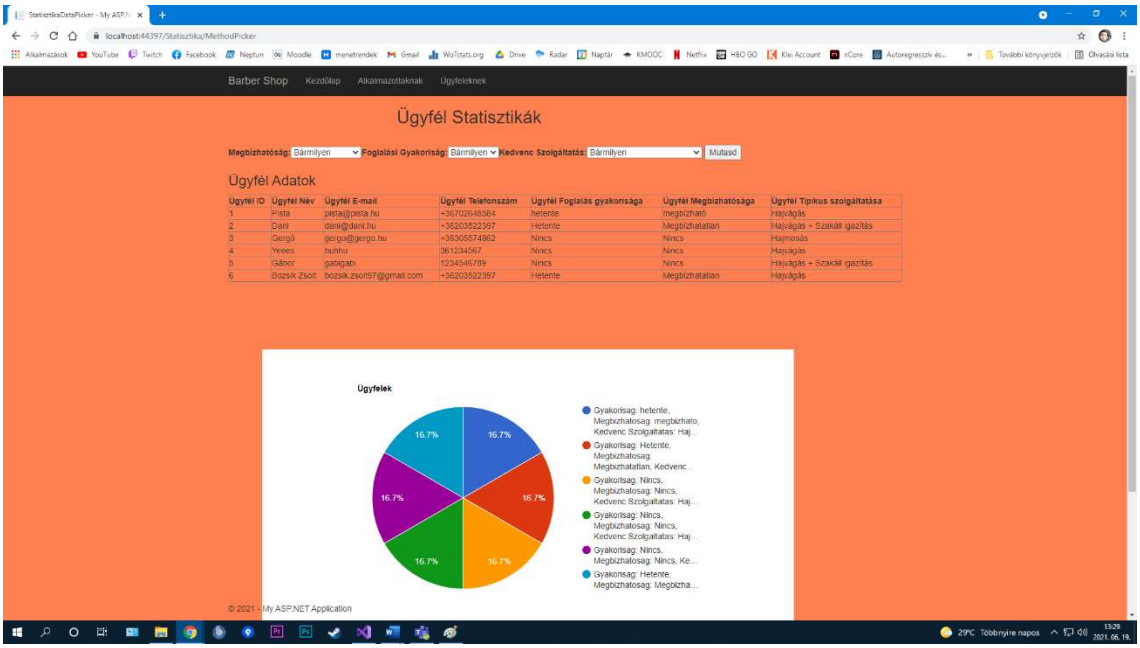
8.9 ábra: Dolgozók kezelő felülete

8.2.7. Statisztikák

Ha az ügyfél statisztikákat kívánjuk megtekinteni akkor ezt megtehetjük három szempont szerint:

- Megbízhatóság
- Gyakoriság
- Tipikus szolgáltatás

Lehetőség van egy vagy több szűrő szerint végezni a keresést, láthatjuk részletesen a listába a vevőket, valamint a lista alatt egy kördiagrammon is hogy még látványosabb legyen. A diagramm egyedeit a szűrők kombinációi adják.



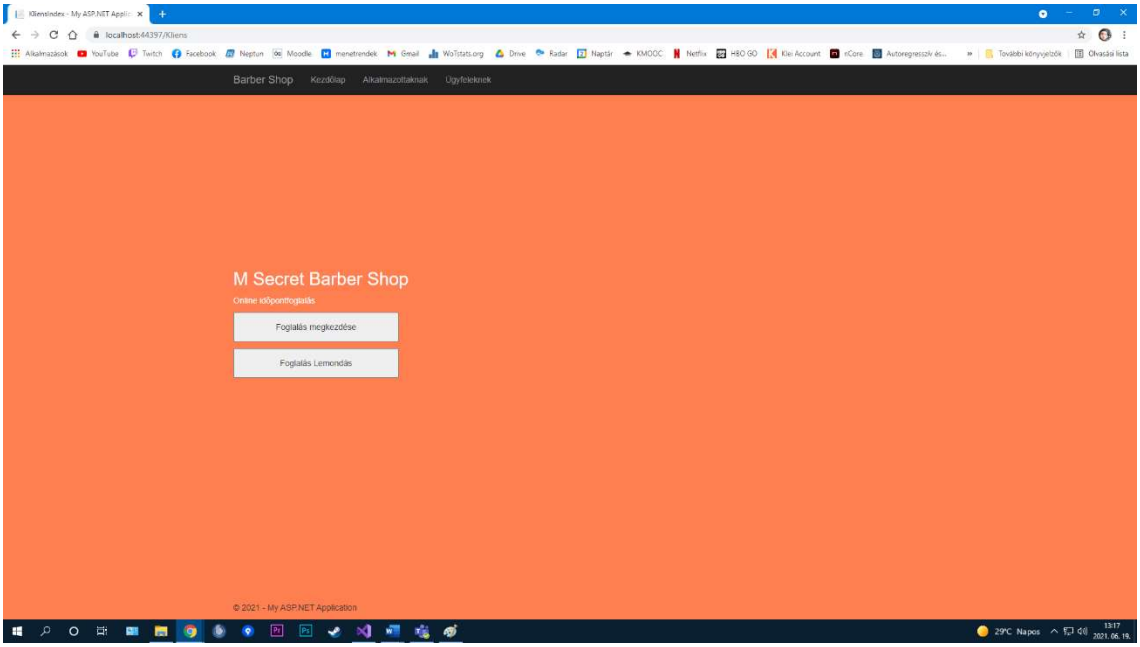
8.10 ábra: Statisztika felület

8.3. Kliens

Mivel a kliens folyamatok elég összetettek ezért ennek egy külön Controller-e és ViewModel-e van.

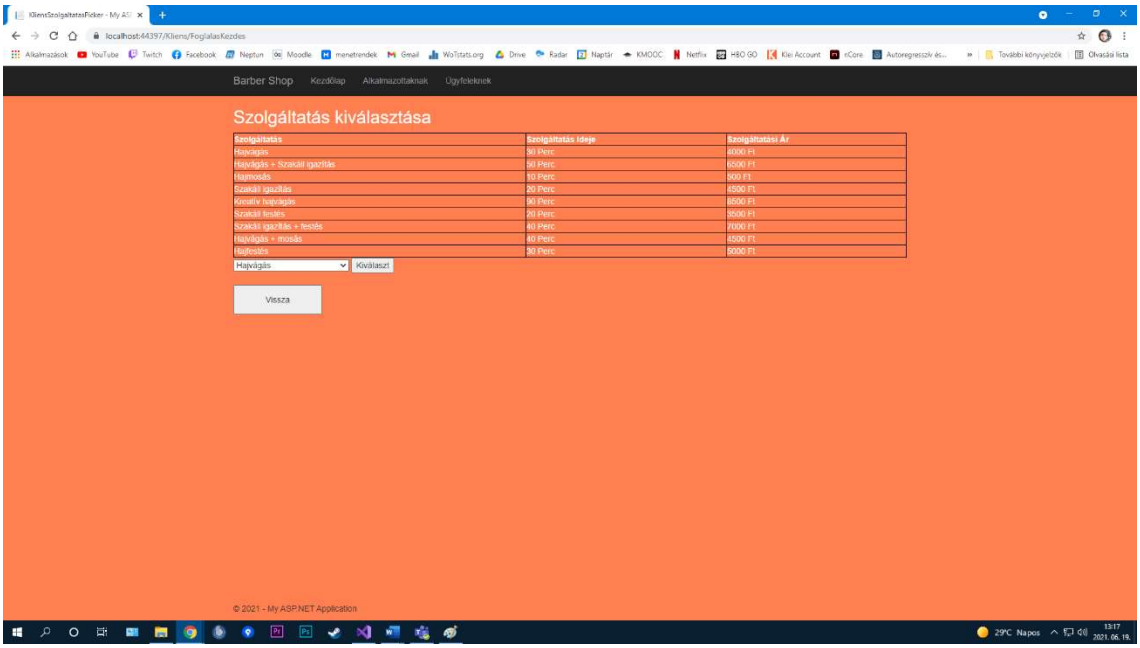
8.3.1. Foglalás

Ha belépünk az ügyfeleknek szánt oldalra akkor eldönthetjük, hogy foglalni, vagy lemondani szeretnénk egy alkalmat.



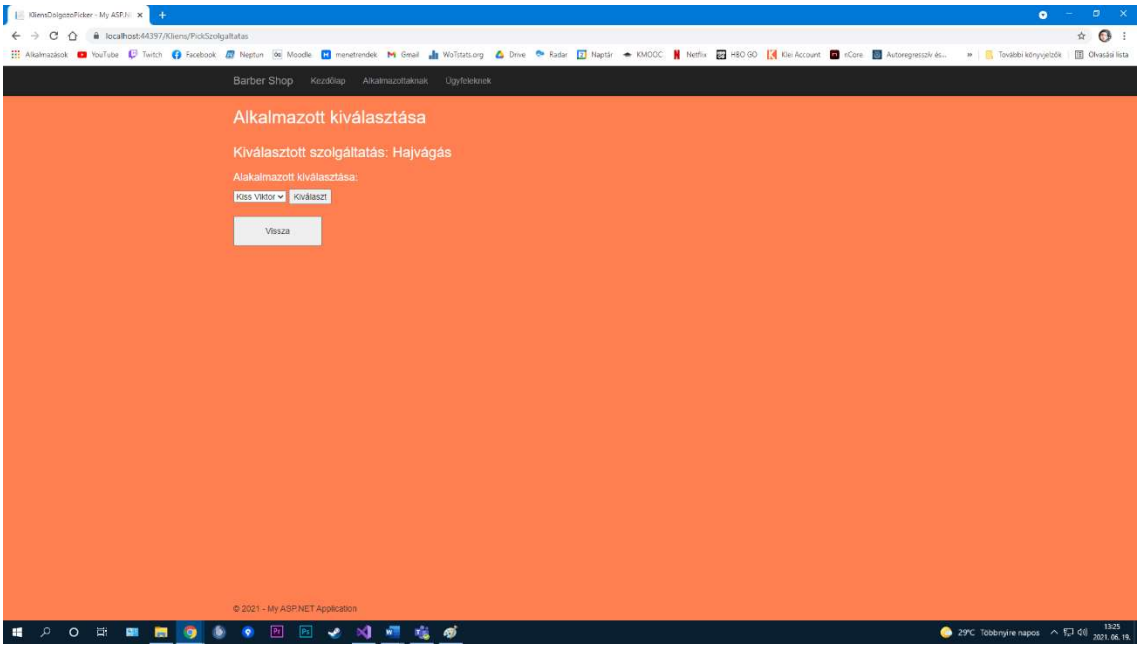
8.11 ábra: Ügyfél kezdőlap

Hogyha a foglalást választjuk akkor elsőnek a szolgáltatást kell kiválasztanunk. Ehhez elsőnek a ViewModel-ben tárolt szolgáltatások listáját használtam fel, valamint, ha választ a vevő akkor azt is letárolja a ViewModel.



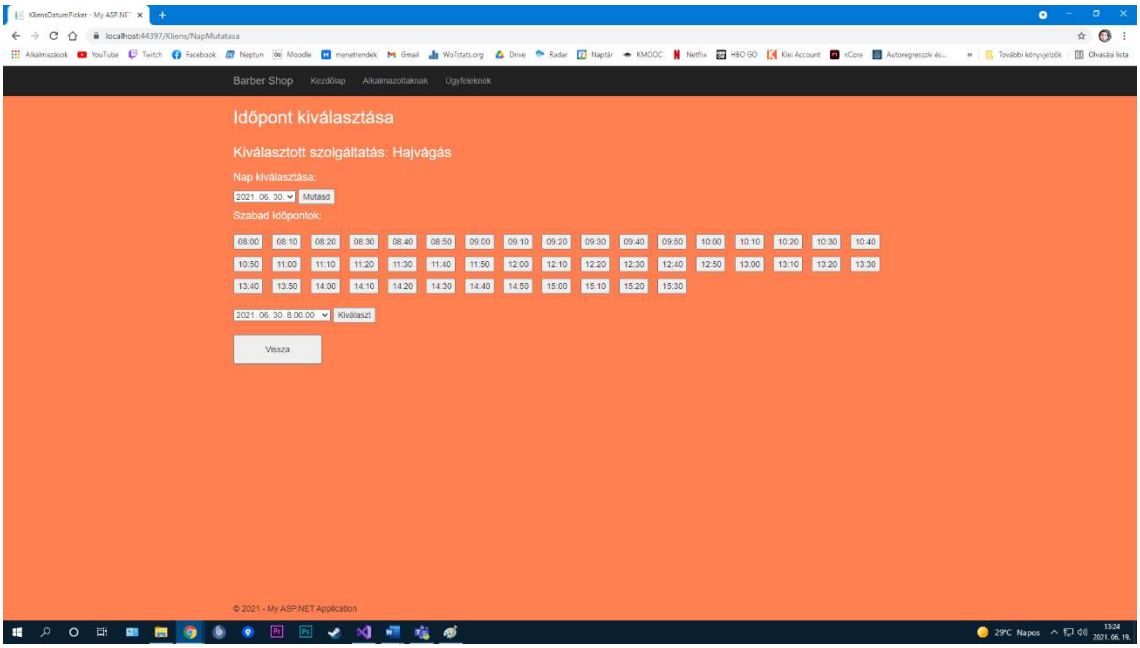
8.12 ábra: Szolgáltatás kiválasztása

Ha megvan a szolgáltatás akkor a szolgáltatást nyújtó alkalmazottat kell kiválasztani.



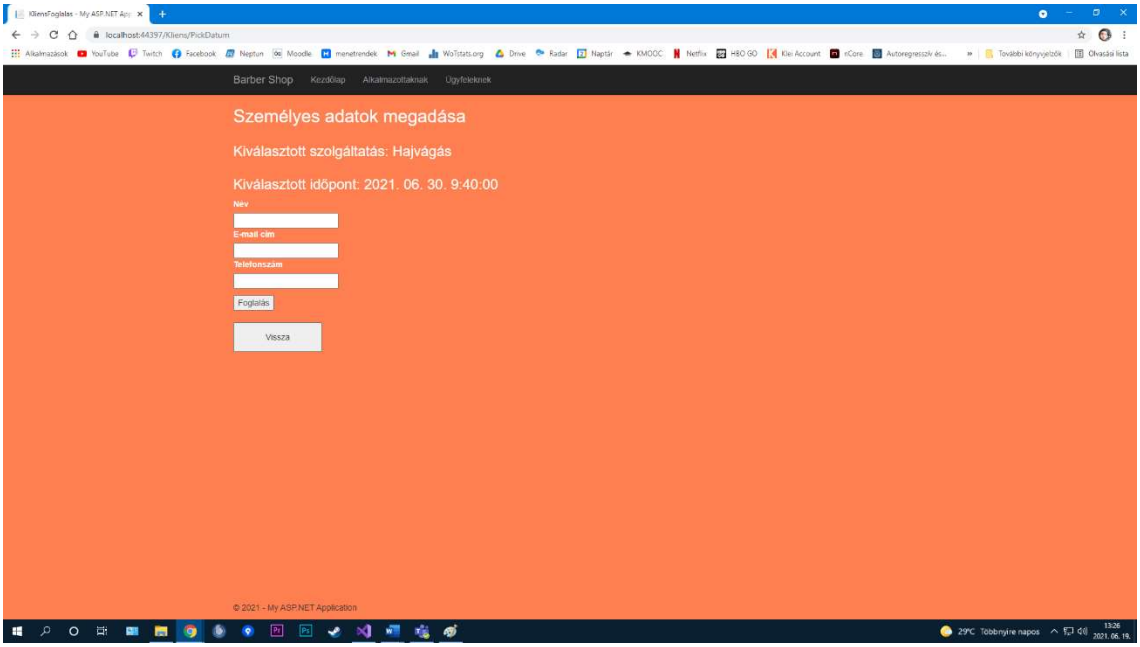
8.13 ábra: Alkalmazott kiválasztása

Ha az alkalmazott választás megtörtént a vevő választhat időpontot. Ez a három lépcső azért kell mivel a szolgáltatás időtartamától függ az, hogy mikor van szabad hely. Ezt úgy oldottam meg hogy mikor tovább lép a szolgáltatás lapról akkor a KliensController átadja a szolgáltatás nevét a Logic rétegnek, ami a Repository-ból megkapja annak időtartamát és mivel tíz perces osztásban van a beosztás ezért könnyen megkeresi a lehetséges helyeket. Ezután visszaadja a KliensController-nek, ami a ViewModel-ben eltárolja azokat, így látja a vevő az időpontokat. Ha pedig választ azt eltárolja a ViewModel.



8.14 ábra: Időpont kiválasztása

Hogyha minden megvan akkor jöhet az utolsó oldal, ahol a vevő a személyes adatait megadja és végez a foglalással. Amennyiben sikeres a foglalás visszadobja a főoldalra és kiírja azt. Ezeket szintén a ViewModel tárolja



8.15 ábra: Személyes adatok megadása

Viszont a folyamat itt nem áll meg. A Logic rétégben le ellenőrzi, hogy a vevő megtalálható-e az adatbázisunkban, ha nem akkor felveszi azt. Ha szerepel akkor lekérdezi az eddigi foglalásait és az új foglalást figyelembe véve módosítja az ügyfélszokással kapcsolatos tulajdonságokat.

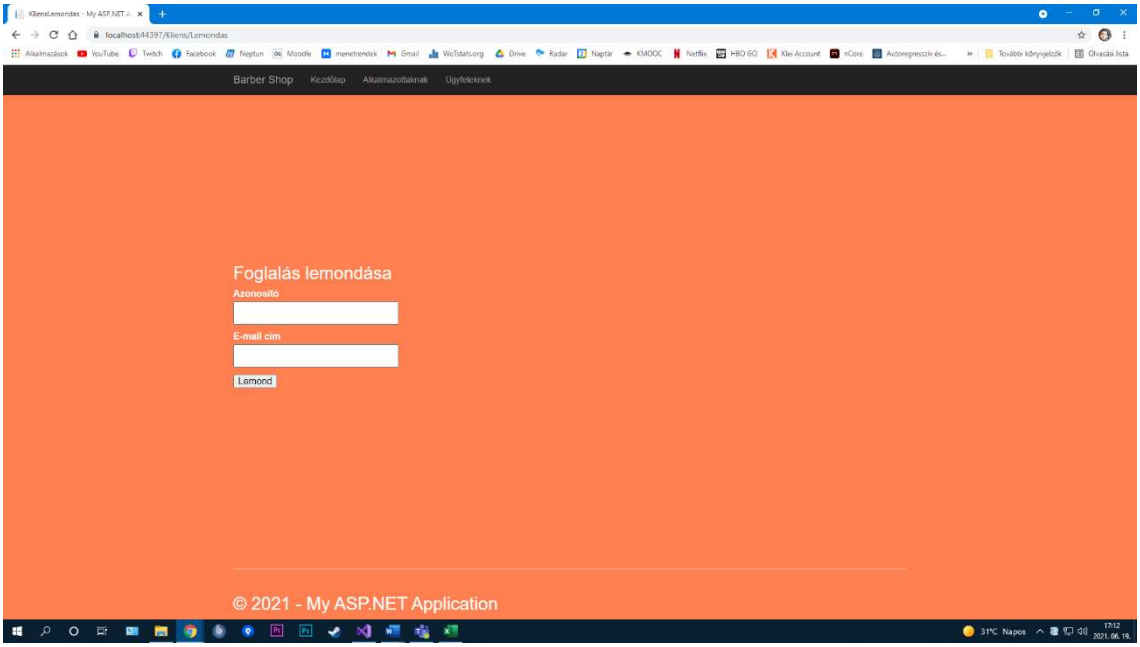
Végezetül pedig az ügyfél e-mail címére küld egy levelet, hogy a foglalása sikeres, valamint egy lemondáshoz szükséges számmal. Ahhoz, hogy ez megvalósuljon a System.Net.Mail könyvtárat vettem igénybe. Ezen kívül kell még egy email cím, amiről küldjük az üzeneteket.

```
private void SendEmail(string email, string nev, int id)
{
    SmtpClient Client = new SmtpClient()
    {
        Host = "smtp.gmail.com",
        Port = 587,
        EnableSsl = true,
        DeliveryMethod = SmtpDeliveryMethod.Network,
        UseDefaultCredentials = false,
        Credentials = new NetworkCredential()
        {
            UserName = "bozsik.zsolt.new@gmail.com",
            Password = "Szakdoga2"
        }
    };

    MailAddress FromEmail = new MailAddress("bozsik.zsolt.new@gmail.com", "Üzlet");
    MailAddress ToEmail = new MailAddress(email, nev);
    MailMessage message = new MailMessage()
    {
        From = FromEmail,
        Subject = "Online Foglalása Rögzítésre került",
        Body = "Kedves " + nev + ", \n Foglalása sikeres volt. \n Lemondáshoz szükséges kód: " + id
    };
    message.To.Add(ToEmail);

    try
    {
        Client.Send(message);
    }
```

8.16 ábra: Az email küldés folyamata

8.3.2. Lemondás

A lemondó felületen meg tudjuk adni az emailben kapott kódunkat és az email címünket és ha foglalásunk több mint egy nap múlva lesz akkor lemondhatjuk azt. A rendszer ellenőrzi, hogy létezik-e ilyen foglalás és bele fér-e az időbe. Ezután felszabadítja a helyeket amire a foglalás szólt és végül kitörli azt a foglalások táblából.



8.17 ábra: Lemondó felület

Ha a lemondás sikeres visszadob a főoldalra és kiírja, hogy sikeres lemondás, ha nem akkor pedig, hogy sikertelen lemondás.

8.4. Predikciók megvalósítása

8.4.1. Várható foglalások

Elsőnek lekérdezzük a foglalási adatokat, ezeket majd tovább szűrjük attól függően mennyire specifikus predikciót szeretnénk. A minimum foglalás az öt darab lesz a rendszerben. Ha valaki már ötször foglalt, akkor a rendszer tud számára várható foglalást számítani. Erre azért van szükség mivel négy autokorrelációs értéket fog számítani, amihez öt bemeneti adat kell. Hogyha megvannak az adataink akkor egy átlagszámítás után könnyen megmondhatjuk az átlagtól való eltérés négyzetösszegét, ami, ha átlagosan kevesebb mint kettő, akkor az átlag módszert használjuk. Erre azért van szükség mert ha az eltérés négyzetösszege nulla akkor egy nullával való osztás lép fel a képletbe, ami így kifagyna.

Ha megvan az átlageltérés akkor jön egy öt lépcsős számítás, ami segít megadni milyen erős kapcsolatba állnak egymással az értékek. Ezt úgy teszi meg, hogy elsőnek önmagából kivonja az átlagot majd ezt négyzetre emeli, hogy a negatív előjelek eltűnjenek és a kiugrások még jobban kitűnjenek. Ezt még négy lépcsőn keresztül megtesszük azzal az eltéréssel, hogy ahogy haladunk előre annál távolabb lévő értékek összefüggését vizsgáljuk olyan módon, hogy a négyzetre emelés helyett az előző érték átlagtól való eltérésével szorozzuk, így már lehetnek negatív számok, de ugye más értéktől függhet negatívan is. Az ötödik lépcső után már csak el kell osztani egyenként a lépcsők összegét az átlagtól való eltérés négyzetösszegével és meg is kapjuk az autokorrelációs folyamat eredményét. Összesen öt eredményt kapunk az öt lépcső miatt, ahol az első mindig egy lesz, mivel egy érték önmagától száz százalékban függ. Az így kapott eredményeket AFC(x)-nek, a parciálisat pedig PACF(x)-nek fogom nevezni, ahol az x a sorszámot jelöli.[13]

Most, hogy a sima autokorrelációs eredmények megvannak jöhetnek a parciális eredmények is. A PACF(1)-nek az értéke egyenlő lesz az ACF(2) eredményével, a többi viszont már másképpen jön ki. Ehhez viszont szükségünk lesz egy mátrixra, ami a következőképpen fog kinézni:

ACF(1)	ACF(2)	ACF(3)	ACF(4)
ACF(2)	ACF(1)	ACF(2)	ACF(3)
ACF(3)	ACF(2)	ACF(1)	ACF(2)
ACF(4)	ACF(3)	ACF(2)	ACF(1)

8.18 táblázat: ACF mátrix

Miután feltöltöttük a mátrixot az értékekkel, ezután egy összetett mátrixművelet jön. Elsőnek a balsarokból indulva akkora mátrixot jelölünk ki ahanyadik PACF-et akarjuk kiszámolni, ha a másodikat akkor egy 2x2-es mátrixot. Ha megvan a mátrixunk akkor azt elsőnek megszorozzuk önmagával, majd kiszámítjuk az inverzét, ezt követően

megszorozzuk az eredetileg kijelölt mátrix-al majd végül készítünk az ACF értékekből is egy mátrixot, ami egy olyan oszlop mátrix lesz, amely sorba tartalmazza az ACF eredményeket az elsőt kihagyva. Így kapunk egy oszlopmátrixot, aminek a legalsó értéke lesz az adott PACF. Így összesen négy PACF értéket kapunk.

Megkaptuk az adatokat már csak egy eloszlás elemzés kell, hogy meg tudjuk adni melyik modellt használjuk és milyen paraméterekkel.

ACF eloszlás	PACF eloszlás	Modell
Geometrikus	Szignifikáns	Autoregresszív
Szignifikáns	Geometrikus	Mozgó átlag
Szignifikáns	Szignifikáns	Autoregresszív mozgó átlag

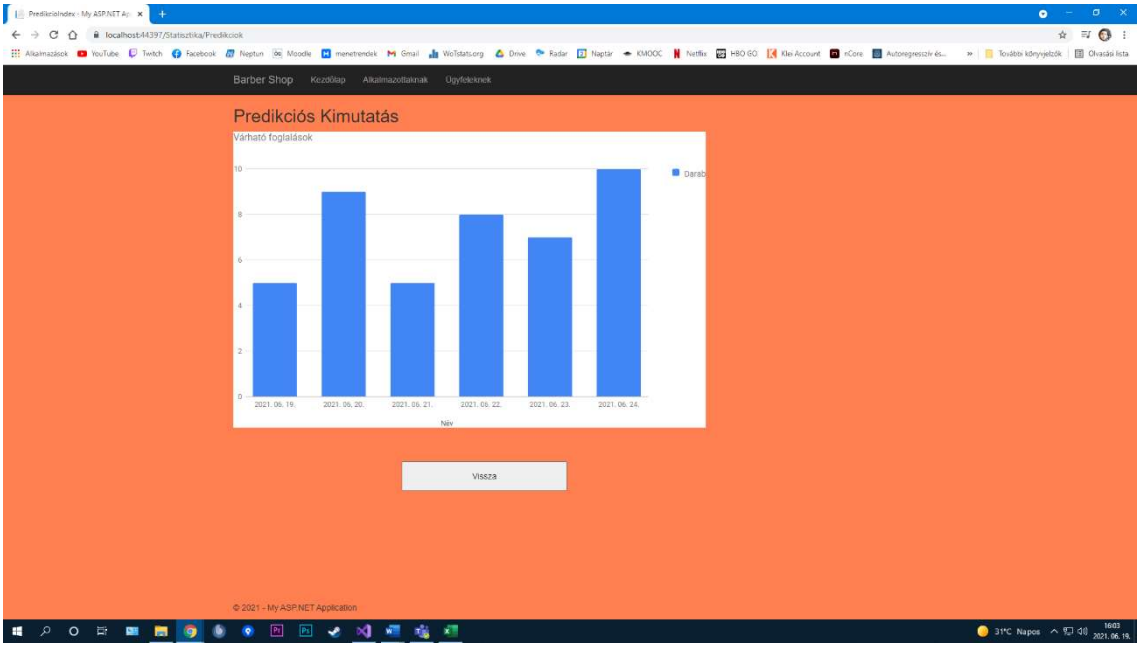
8.19 táblázat: Modell tulajdonságok

Hogyha egy érték szignifikánsan kiugrik belekerül a modellbe, amit a táblázat alapján választ ki a rendszer.

Mozgó átlag modell esetén az átlaghoz hozzáadjuk azokat az előző értékek ACF(x)-szeresét ahanyadik ACF értékeket kaptuk vissza az elemzésből.

Autoregresszív modell esetén kiszámítjuk a nulladik becsült értéket, amit úgy kapunk, hogy az átlagot megszorozzuk az egy mínusz PACF(1)-el. Ehhez a nulladik értékhez adjuk hozzá az elemzés során kikerült PACF(x)-el súlyozott értékét az előző értékeknek.

Autoregresszív mozgó átlag modell esetén, ugyan úgy kiszámoljuk a nulladik értéket, mint előbb és hozzáadjuk a PACF által súlyozott előző értékeket viszont ezen felül még azokat az ACF értékkel súlyozott hibákat, amiket úgy kapunk, hogy megnézzük a predikcióhoz képest mikor jött a vendég. Így megkapjuk a predikált értéket.[14]



8.20 ábra: Predikciós diagramm

8.4.2. Várható alkalmazott terhelés

Ha minden olyan foglalást, ami egy adott alkalmazott egy adott szolgáltatására tekintünk egy foglalásnak akkor meg tudjuk jósolni, hogy a jövőben kinek mennyi vevője lesz és milyen szolgáltatást vesznek nála igénybe. Ha úgy tekinti az admin, hogy túlterhelt lesz az adott alkalmazott akkor lehetősége van email-ben egy átjelentkezést felajánlani azoknak a vevőknek, akik foglaltak.

8.4.3. Predikált ideig nem foglalók

Hogyha az adott ügyfél nem foglal a várható időpontig lehetőségünk van emailben ajánlatot küldeni nekik, mivel a foglalásuk esedékes. Ilyenkor a rendszer megnézi a várható foglalás időpontját és ha nem foglalt addig, akkor az ügyfél bele kerül ebbe a halmazba.

9. Tesztelés

A tesztelést két részre osztottam a felhasználó szerint. Az admin tesztelés alatt azok a tesztek eredményei vannak, amik az admin-hoz köthetőek mint pl: a beosztás megadás vagy a statisztika megnézése míg a klienshez olyanok, mint az időpontfoglalás.

Hardver adatok:

Hardver	Típus
Processzor	Intel® Core™ i5-8400 CPU @ 2.80GHz
Memória	16,0 GB

9.1 táblázat: Hardver adatok

Szoftver adatok:

Szoftver	Típus
Operációs rendszer	Microsoft Windows 10 pro
Keretrendszer	Microsoft .NET Framework 4.8
Fejlesztő környezet	Microsoft Visual Studio 2019
Programnyelv	C#
Alkalmazás típusa	ASP.NET web application
Adatbázis	PostgreSQL 13 Database server

9.2 táblázat: Szoftver adatok

9.1. Admin tesztelés

Teszt	Érték
Szolgáltatás hozzáadása	sikeres
Szolgáltatás szerkesztése	sikeres
Szolgáltatás törlése	sikeres
Ügyfél hozzáadása	sikeres
Ügyfél szerkesztése	sikeres
Ügyfél törlése	sikeres
Időpont hozzáadása	sikeres
Időpont szerkesztése	sikeres
Időpont törlése	sikeres
Foglalás hozzáadása	sikeres
Foglalás szerkesztése	sikeres
Foglalás törlése	sikeres
Statisztika megjelenítése szűrés szerint	sikeres
Beosztás megadása egy napra	sikeres

9.3 táblázat: Admin teszt eredmények

9.2. Kliens tesztelés

Teszt	Érték
Foglalásnál az összes szolgáltatás megjelenik	sikeres
Csak azok az időpontok jelennek meg ahol belefér az adott szolgáltatás	sikeres
Sikeresen lehet foglalni	sikeres
Email megérkezik	sikeres
Kód segítségével lemondás sikeres	sikeres
Lemondást követően felszabadulnak a helyek	sikeres

9.4 táblázat: Kliens teszt eredmények

10. Elért eredmények

A szakdolgozatom készítése során, hogy jobban bele lássak a statisztika alapú predikciós módszerek világába, valamint megismerjem mélyebben a weboldal készítés folyamatát és hogy milyen kihívásokat rejtenek ezek. Most már megértem, hogy a legtöbb hasonló ingyenes rendszer miért is olyan korlátolt, hiszen, ha az alapjairól kezdünk egy szoftvert fejleszteni az rengeteg idő lesz és lehet nem is térül meg a végén.

A legnagyobb kihívás és majd a legnagyobb eredmény szerintem a predikciós módszer maga volt az. A nehézség abban rejlett, hogy nagyon kevés olyan tanulmány, videó, leírás található meg az interneten ezekhez a módszerekhez és ami van az inkább az egyszerűbbeket mutatja be például a lineáris regressziót. Viszont miután megértettem az autoregresszív modell működését el tudtam érni, hogy egy nap pontossággal előre jelezze a foglalásokat. Nyilván, ha valaki teljesen véletlenszerűen foglal nála nagyobb lesz az eltérés.

A másik nagy kihívás a diagrammal volt, mivel alapértelmezetten nagyon kezdetleges az, amit a .NET képes weben megjeleníteni. Szóval külső elemre volt szükségem és itt szintén a leírás hiány okozta a nehézséget, de végül a google chart segítségével sikerült megoldani a problémát. Így egy informatív dinamikus ábra is segíti az embereket megérteni a statisztikákat.

A fejlesztés során figyeltem fel arra, hogy mennyire is fontos a megjelenés egy weboldal számára, hiszen az emberek a weben a legbizonytalanabbak és ha egy oldal rosszul vagy nem biztonságosnak néz ki akkor egyszerűen nem használják, így próbáltam törekedni, hogy bár minimalista stílusban, de egy biztonságot sugárzó weboldalt alakítsak ki. Ezenfelül egy alkalmazottak által is szívesen használt szoftver legyen, ami kielégíti az elvárásaikat.

Összességében sikerült egy olyan rendszert alkotni, amivel könnyen lehet foglalni online időpontot, le lehet mondani azt, az alkalmazottak könnyen megadhatják a beosztásukat és ezen kívül statisztikákat valamint predikciókat kapnak amik segítségével jobb szolgáltatást tudnak nyújtani az ügyfeleknek.

11. Továbbfejlesztési lehetőség

Mint az összes informatikai rendszert ezt is szinte a végtelenségig lehetne bővíteni, továbbfejleszteni. Mivel a szokások változnak, valamint a szolgáltatások kínálata, a vevők kereslete szintén és mindig van egy új trend, ami alapján a megjelenítést változtatni lehet ezért a jövőben célszerű is ezeket a fejlesztéseket meglépni. Ilyen például az automatikus számlagenerálás, ami sokat segíteni főleg a kis üzleteknek, viszont ahhoz, hogy hivatalos legyen elég nagy fejlesztés szükségeltetik. Néhány lehetséges fejlesztés, ami tovább könnyítené a vállalkozás munkáját és a vevők elégedettségét növelné:

- Dinamikus google térkép a főoldalon
- Szünetek létrehozása a beosztásban
- Automatikus bérszámítás
- Termék vásárlási lehetőség
- Egyszerre több szolgáltatásra jelentkezés
- Kupon rendszer
- Online fizetés
- Összekötés online közösségi médiával

12. Összegzés

A szakdolgozatom alapjául a Magyar Telekom Nyrt.-nél megszerzett gyakornoki tapasztalatom szolgált, ezen kívül az egyetem tanult webfejlesztés. Egy online időpontfoglalási rendszer kialakítását tűztem ki célul, ami a lehető legkevesebb fizetős modullal rendelkezik. Mivel a legtöbb olyan modul, mint például a Customer Managament System csak a legkezdetleges funkciókig ingyenes ezért úgy döntötte, hogy alapjairól fogom felépíteni az alkalmazást. Így esett a választás az ASP.NET web alkalmazásra. Mivel az egyetlen adatbázis kezelő rendszer, ami minden tekintetben ingyenes a PostgreSQL ezért egy ilyen adatbázis szerver fut az alkalmazásom alatt. Ez szült egy olyan problémát, amit a gyakornoki éven alatt már tapasztaltam, mégpedig, hogy nem lehet közvetlen kapcsolatot könnyen létrehozni az alkalmazás és az adatbázis között viszont ezt sikerült megoldani azzal. hogy a lekérdezéseket az alkalmazás állítja elő. A rendszernek két fő felhasználói csoportja van az ügyfelek és az alkalmazottak. Ezek teljesen más funkciókat igényelnek így a rendszer is lényegében két fő részből áll. A tervezési folyamat elején az adatbázis megtervezésével kezdtem, ami ugyan bővült a legelső tervezet óta, de nem változtak meg meglévő tagok. Ezután a rendszer terv jött, ahol három részre bontottam a szoftvert, elsőnek az adatfeldolgozó részt majd az üzleti logika részt végül pedig a megjelenítésért felelő részt terveztem meg. A megjelenítésnél az MVC architektúrára esett a választás, mivel jól érthető és már volt tapasztalatom benne az egyetem által. A terv elkészítése után jött a megvalósítás. A predikciós módszer kiválasztása nehéz volt mivel több módszert is összehasonlítottam teszt adatokkal és addig kerestem amíg nem találtam egy olyat, ami ténylegesen képes előre megjósolni az eredményt így esett a választás az autoregresszív mozgó átlag modellre.

A program természetesen nem érte el végső formáját a fejlesztés végére, hiszen rengeteg továbbfejlesztési lehetőség van benne, de az eredeti célját elérte. Képes egy ügyfél online foglalni és lemondani azt, amivel már többet tud, mint a jelenleg ingyenes társai, valamint ezen kívül statisztikákat is képes mutatni és predikciót a jövőbeli terhelésre. Az egyik legszükségesebb és talán legnagyobb fejlesztés a jövőben a beépített számlagenerálás lesz és azzal együtt már a legtöbb folyamatot le fogja tudni fedni a rendszer, amit egy kis vagy közép vállalat igényel.

Eddigi tanulmányaim és munkám során a C# nyelvet sajátítottam el, viszont a szakdolgozat által lehetőségem volt belekóstolni az igazi ASP.NET MVC alapú webfejlesztésbe, valamint abba, hogy milyen is mikor olyan dolgot akarunk megvalósítani, ami a közelmúltban született meg és ennek ellenére, hogyan lehet forrásokat találni hozzá. Ezen kívül a statisztika mélyebb részeibe is betekintést nyerhettem, ami csak tovább gyarapította tudásomat.

13. Summary

My thesis was based on my intership experience at Magyar Telekom Plc., In addition with my university studies. I set out to create an online appointment booking system with as few paid modules as possible. Since most modules such as the Customer Managament System are free only for the most basic features, I decided that I would build the application from the ground up. This is how the ASP.NET web application was chosen. Since PostgreSQL is the only database management system that is free in all respects, such a database server runs under my application. This gave rise to a problem that I had already experienced during my internship year, namely that it was not possible to easily establish a direct connection between the application and the database, but it managed to solve this. that the queries are generated by the application. The system has two main user groups, customers and employees. These require completely different functions so the system essentially consists of two main parts. At the beginning of the design process, I started by designing the database, which, although expanded since the very first draft but , did not change existing members. Then came the system plan, where I divided the software into three parts, first the data processing part and then the business logic part, and finally the part responsible for visualization. The MVC architecture was chosen for the visualization as it is well understood and I already had experience with it by the university. After the plan was completed, implementation came. Selecting the prediction method was difficult as I compared several methods with test data and searched until I could find one that could actually predict the result so the choice fell on the autoregressive moving average model.

Of course, the program did not reach its final form by the end of the development, as it has plenty of room for improvement, but it has achieved its original purpose. The ability for a customer to book and cancel online is what they already know more than their currently free counterparts, as well as the ability to show statistics and predict future. One of the most necessary and perhaps the biggest improvements in the future will be the built-in invoice generation and with it, the system will already be able to cover most of the processes that a small or medium company needs.

In my studies and work so far, I have mastered the C # language, but the dissertation gave me the opportunity to get a taste of real ASP.NET MVC-based web development, as well as what we want to accomplish and what can be done with recently find resources. In addition, I was able to gain insight into the deeper parts of the statistics, which only further enriched my knowledge.

14. Irodalomjegyzék

[1] Salonic rendszer bemutatása
https://www.salonic.hu/hogyan-mukodik
Utoljára megtekintve: 2020.05.22.

[2] Book4us rendszer bemutatása
https://booked4.us/hu/Prices
Utoljára megtekintve: 2020.05.22.

[3] Reservio rendszer bemutatása
https://reservio.freshdesk.com/hu/support/home
Utoljára megtekintve: 05.22.

[4] Wordpress.org dokumentációk
https://codex.wordpress.org/Developer_Documentation
Utoljára megtekintve: 2020.12.10.

[5] Joomla dokumentációk
https://docs.joomla.org/Main_Page
Utoljára megtekintve: 2020.12.10.

[6] Drupal dokumentációk
https://www.drupal.org/docs/user_guide/en/understanding-drupal.html
Utoljára megtekintve: 2020.12.10.

[7] Umbraco dokumentációk
https://our.umbraco.com/documentation
Utoljára megtekintve: 2020.12.10.

[8] MS SQL szerver bemutatása
https://docs.microsoft.com/hu-hu/sql/2014-toc/?view=sql-server-2014 Utoljára megtekintve: 2020.05.22.

[9] Oracle SQL szerver bemutatása
https://docs.oracle.com/cd/E11882_01/server.112/e41084.pdf Utoljára megtekintve: 2020.05.22.

[10] Postgre SQL szerver
https://www.postgresql.org/files/documentation/pdf/12/postgresql-12-A4.pdf/
Utoljára megtekintve: 2020.05.22.

[11] .NET C# dokumentáció

https://docs.microsoft.com/hu-hu/dotnet/csharp/?WT.mc_id=dotnet-35129-website
Utoljára megtekintve: 2021.05.05.

[12] Java Netbeans dokumentáció
https://bits.netbeans.org/12.0/javadoc/
Utoljára megtekintve: 2021.05.05.

[13] Autokorrelációs folyamat leírás
https://www.real-statistics.com/time-series-analysis/stochastic-processes/autocorrelation-function/
Utoljára megtekintve: 2021.05.05.

[13] Mozgóátlag folyamat leírás
https://www.real-statistics.com/time-series-analysis/moving-average-processes/calculating-ma-coefficients-solver/
Utoljára megtekintve: 2021.05.05.