

Pannon Egyetem  
Műszaki Informatikai Kar  
Rendszer- és Számítástudományi Tanszék  
mérnökinformatikus BSc

## SZAKDOLGOZAT

Vendéglátói szálláshelyek csoportos foglalását  
megvalósító rendszer fejlesztése

Rozsenich Balázs

Témavezető: Frits Márton

2015

## **Feladatkírás**

## Nyilatkozat

Alulírott Rozsenich Balázs hallgató kijelentem, hogy a dolgozatot a Pannon Egyetem Rendszer- és Számítástudományi tanszékén készítettem a mérnökinformatikus végzettség megszerzése érdekében.

Kijelentem, hogy a dolgozatban lévő érdemi rész saját munkám eredménye, az érdemi részen kívül csak a hivatkozott forrásokat (szakirodalom, eszközök, stb.) használtam fel.

Tudomásul veszem, hogy a dolgozatban foglalt eredményeket a Pannon Egyetem, valamint a feladatot kiíró szervezeti egység saját céljaira szabadon felhasználhatja.

Veszprém, 2015. április \_\_

.....

Rozsenich Balázs

Alulírott Frits Márton témavezető kijelentem, hogy a dolgozatot Rozsenich Balázs a Pannon Egyetem Rendszer- és Számítástudományi tanszékén készítette a mérnökinformatikus végzettség megszerzése érdekében.

Kijelentem, hogy a dolgozat védelemre bocsátását engedélyezem.

Veszprém, 2015. április \_\_

.....

Frits Márton

### **Köszönetnyilvánítás**

Köszönettel tartozom Frits Márton témavezetőmnek a szakdolgozati a témám megvalósításában nyújtott segítségéért és támogatásáért.

Köszönöm családomnak és barátaimnak a végtelen biztatást és támogatást.

## TARTALMI ÖSSZEFOGLALÓ

A szállásfoglalás és utazásszervezés világában gyakran előfordul az a probléma, hogy egy csoport számára az utazásszervező nem talál megfelelő kapacitású szálláshelyet és a csoport tagjait több szálláshelyen kell elhelyeznie. Ekkor az utazásszervező feladata a desztinációt övező szálláshelyek felkutatása és kiszűrése a kívánt igények szerint. A folyamat bonyolult és nehézkes, sok kutatást és kalkulációt igényel. Tovább bonyolítja a szervezést, hogy a szálláshelyek a foglalásokat egymástól függetlenül kezelik. A feladatom az, hogy a felvázolt szállásfoglalási folyamatot megkönnyítsem és levegyem a terhet az utazásszervező válláról.

A megoldás egy olyan koncepció, ami a szálláshelyek helyett a szobákra helyezi a hangsúlyt és szálláshelyet csak a szoba egy tulajdonságának tekinti. Ehhez egy olyan szálláskereső portált készítettem el, ami működésében egy webshophoz hasonlít, ahol a termékek a szobák, a raktárkészlet pedig a szálláshelyek összes szobája szobatípusonként a foglaltság függvényében. Egy ilyen felületen a szálláskereső szabadon válogathatja össze a megfelelő szobákat egy virtuális kosárba, amiből aztán a foglalását létrehozza. A szállásadó a foglalás csak számára releváns részét dolgozza fel. A megfelelő szobák kiválasztását automatizáltam, amely folyamathoz optimalizálási feladatokat határoztam meg. Az optimalizálás során a cél az, hogy minél olcsóbb szobákat vagy egymáshoz minél közelebb eső szobákat, esetleg mindkét feltételt kielégítő szobákat ajánljon automatikusan a rendszer.

A munkám eredményeként létrehoztam egy Ruby on Rails webalkalmazást, ami teljes értékű szálláskereső portálként képes szálláshelyek szobáinak keresésére, szűrésére, és magában foglalja a szállásfoglalás teljes folyamatát valamint egy foglalás akár több szállás szobáit is tartalmazhatja. Az elkészült webalkalmazásban implementáltam a szobákat automatikusan ajánló funkciót, amely nemlineáris optimalizálási feladatok megoldásával az ár vagy távolság szempontok szerint javasol a csoport számára szabad szobákat.

**Kulcsszavak:** turizmus, csoportos szállásfoglalás, szoba orientált szálláskereső, nemlineáris optimalizálás, webalkalmazás, Ruby on Rails

## ABSTRACT

In the world of tourism it is a usual occurrence that a group can't be accommodated in one place, but in multiple others, because of lack of capacity. In a situation like this, the tour operator's task is to seek for accommodations around the destination and filter out ones that fit the requirements. This process is elaborate and cumbersome and it requires throughout investigation and calculation. My aim is to provide an easier alternative to solve this problem and disencumber the tour operator.

My solution is a concept, which emphasizes the rooms instead of the accommodations. In my concept an accommodation stands for the quality of a room. To demonstrate this concept I developed a web application, which works like a web shop. In a web shop like this, the products are the rooms, and the stock is all the rooms of the accommodations grouped by room types and are dependent on their availability. On a surface like this the tour operator can freely pool rooms in a virtual basket and then make the reservations through it. The accommodation owners only have to manage the part of a reservation, which belongs to them. I automatized the selection of suitable rooms for which I designated optimization tasks. During optimization the goal is to offer the cheapest or the closest rooms or even both automatically.

The result of my work is a Ruby on Rails based web application which as a fully equipped booking application is capable of searching and filtering through rooms of accommodations and of making reservations of multiple rooms even across multiple accommodations. I implemented an automatized room-recommendation function into the finished application which through non-linear optimization can recommend rooms based on their price and distance to a tourist group.

**Keywords:** tourism, booking for groups, room emphasized booking, nonlinear optimization, web application, Ruby on Rails

## Tartalomjegyzék

Feladatkiírás.....	2
Nyilatkozat.....	3
Köszönetnyilvánítás .....	4
TARTALMI ÖSSZEFOGLALÓ .....	5
ABSTRACT .....	6
Tartalomjegyzék .....	7
1. Bevezetés.....	10
1.1. A probléma és megoldása .....	10
2. Szálláskereső portálok .....	12
2.1. Szállás.hu .....	12
2.2. Booking.com .....	12
2.3. Trivago.hu.....	13
2.4. Konklúzió .....	13
3. Matematikai optimalizálás .....	14
3.1. A matematikai optimalizálás története .....	14
3.2. Matematikai optimalizálási feladat .....	18
3.3. Lineáris optimalizálási feladat.....	19
3.4. Nemlineáris optimalizálási feladat.....	20
3.5. Alkalmazási területek.....	20
4. Ruby on Rails.....	21
5. Specifikáció.....	24
5.1. Szereplők .....	24
5.2. Funkcionális követelmények .....	24
5.2.1. Felhasználói fiókok .....	24
5.2.2. Szobák szűrése.....	24
5.2.3. Szobafoglalás .....	25
5.2.4. Értékelés.....	25
5.2.5. Intelligens keresés .....	25
5.2.6. Törzsadatok.....	25
5.2.7. Tartós címek .....	25
5.3. Célcsoport .....	26
6. Tervezés.....	27

6.1.	A rendszerben megjelenő fő folyamatok.....	27
6.1.1.	Szobafoglalás.....	27
6.1.2.	Foglalás visszaigazolás .....	28
6.1.3.	Intelligens keresés .....	28
6.2.	Nemlineáris optimalizálási modell .....	29
6.2.1.	Olcsó modell .....	32
6.2.2.	Közeli modell .....	32
6.2.3.	Olcsó és közeli modell .....	33
6.3.	Adatbázis tervezet .....	34
6.4.	Technológia .....	35
6.4.1.	Keretrendszer .....	35
6.4.2.	Adatbázis.....	36
6.4.3.	Optimalizálási modellezés .....	36
6.4.4.	Nemlineáris megoldó .....	37
6.4.5.	Megjelenés .....	38
6.4.6.	Autentikáció és autorizáció.....	39
6.4.7.	Geolokáció .....	40
6.4.8.	Úrlapok.....	40
6.4.9.	Képek tárolása és megjelenítése .....	41
7.	Megvalósítás .....	42
7.1.	Adatbázis kapcsolat és modellek elkészítése .....	42
7.2.	Autentikáció és autorizáció .....	45
7.3.	Szobák szűrése .....	49
7.4.	Intelligens keresés .....	53
8.	Felületek és használat .....	62
8.1.	Menüsáv.....	62
8.2.	Szobák.....	64
8.3.	Szálláshelyek .....	67
8.4.	Intelligens keresés .....	69
8.5.	Kosár.....	69
8.6.	Foglalások.....	70
9.	Tesztelés.....	72
9.1.	Tesztelési környezet.....	72
9.2.	Teszt adatok.....	72



9.3. Teszt eredmények.....	73
10. Összefoglalás.....	74
Irodalomjegyzék .....	75
Ábrajegyzék .....	77
Mellékletek .....	78
[1] Adatbázis diagram.....	78
[2] UrlHelper segédosztály .....	79
[3] FilterHelper segédosztály szobák szűrését megvalósító metódusai .....	81
[4] Intelligens keresés felület .....	84
[5] Foglалás véglegesítés felület.....	85
[6] Foglалások (szálláskereső) .....	86
[7] Foglалások (szállásadó) .....	86
[8] Foglалás részletei (szálláskereső) .....	87
[9] Foglалás részletei (szállásadó) .....	88
[10] Tesztelési eredmények .....	89
CD Melléklet .....	93

## 1. Bevezetés

A szakdolgozatom témája egy olyan webalkalmazás elkészítése, amely csoportok (pl.: osztályok, baráti vagy üzleti társaságok) számára teszi egyszerűbbé a több szálláshelyen történő szobafoglalás menetét és kezelését. Az alkalmazás szempontjából fontos az internetes platform, mert így lehet a legolcsóbban a legszélesebb felhasználói körnek elérhetővé tenni. A projekt munkacímének a *VAGATO* szót választottam, amelyet a katalán *vaganto* szóból (jelentése: barangolás) képeztem.

### 1.1. A probléma és megoldása

A csoportos turizmus jelentős szereppel bír a turizmusban, gondoljunk csak a tavasszal és ősszel, százával kirándulni induló diákokra, a közös szórakozásra vágyó baráti társaságokra, vagy az egyéb, üzleti célból utazó társaságokra. Egy csoport számára, különösen főszezonban kivételesen nehéz mind árban, mind távolságban megfelelő szálláshelyet találni, illetve gyakran előfordul, hogy egy szálláshely nem képes megfelelő számú kapacitást kínálni. A kapacitás korlátja lehet az aktuális foglaltság, vagy – jellemzően kisebb településeken – a szálláshelyek alapvető szobakínálatának csekélyisége. Ilyen helyzetekben az utazásszervező feladata az, hogy összegyűjtse a szálláshelyek ajánlatait és az idővel versengve kalkulációk útján kiválassza a megfelelő szálláshelyek megfelelő szobáit.

A szálláshelyek kiválasztása után az utazásszervező szembesül a következő problémával. Minden szálláshely egyedileg kezeli a foglalásokat, az utazásszervezőnek minden szálláshellyel külön-külön kell megegyeznie. Ez rengeteg, egymástól független ügyintézés és papírmunkát jelent és jelentősen megbonyolítja a folyamatot.

Az általam tervezett webalkalmazás a fent vázolt problémákat igyekszik feloldani és használható megoldást kínálni. A koncepció az, hogy a jelenleg szálláshely orientált piacot meg kell fordítani és a középpontba a szobákat kell helyezni. A szobának, csakúgy, mint a légkondicionálás vagy az ellátás, csak egy tulajdonsága az, hogy mely szálláshelyhez tartozik. A szálláshelyek adta

kötöttségek feloldásával már könnyű elképzelni egy olyan portált, ami a szobákat, mint egy webshopban, termékeként sorolja fel. A szobák a szálláshelyektől függetlenül kereshetők, szűrhetők és foglalhatók. A szobák e fajta individuális termékként való kezelése a kulcs ahhoz, hogy az utazásszervező olyan foglalásokat tudjon összeállítani, amiben egyszerre jelenik meg több szálláshely több szobája egy közös felületen. A felvázolt koncepció alapján belátható, hogy a szállásfoglalás megvalósítható úgy, hogy a szálláskereső a szállásfoglalási portál böngészése közben egy virtuális kosárba helyezi a megfelelő szobákat és a böngészés végén a kosarában lévő szobákat egy foglalássá egyesíti, majd a vendégadatok megadása után véglegesíti azt.

A szobák kiválasztásának folyamata bonyolult, azonban jól automatizálható. A keresés szempontjai szobák felszereltsége, a szolgáltatások és a foglaltság. Ezek a feltételek gyorsan és egyszerűen szűrhetők úgy, hogy az utazásszervező egy űrlapon megjelöli a kívánalmakat. A nagyobb nehézséget az ár, a minőség és a távolság feltételei adják. Az utazásszervező olyan szobákat akar, amik olcsók, ugyanakkor nincsenek távol egymástól; vagy a távolság nem számít, de legyenek minél jobb értékelésű szálláshelyeken. Az efféle szempontokhoz már nem elég szimplán sorrendbe állítani a szobákat és kiválasztani az első néhány darabot. Egy optimális megoldás kiválasztása kísérletezés útján kézzel is elvégezhető, azonban kimondottan időigényes feladat. A webalkalmazásnak tehát rendelkeznie kell egy olyan funkcióval, ahol a kényelmi szempontok és a csoport létszáma szerint egy ár, távolság illetve minőség szerint optimális megoldást kap az utazásszervező arról, hogy mely szobákat kell lefoglalnia. A felvázolt funkciót a rendszerben *intelligens keresésnek* neveztem el.

## **2. Szálláskereső portálok**

Ebben a fejezetben a magyar szálláskereső piac legnépszerűbb weboldalait vizsgálom meg a szerint, hogy milyen lehetőségeket kínálnak a szobák, illetve szálláshelyek keresésére, szűrésére. A vizsgálatom kitér arra, hogy ezek a weboldalak mennyire támogatják a csoportos szálláskereső 1.1 fejezetben bemutatott problémáit.

### **2.1. Szállás.hu**

A Szállás.hu egy magyar alapítású és fejlesztésű szálláskereső portál, amely 2007 óta üzemel. A Szállás.hu tekinthető a magyar szálláskereső piac legnépszerűbb szereplőjének. A szállásadók részére egységes megjelenést és könnyű foglalást ígér jutalékért cserébe.

A szálláskereső során részletesen megadhatók a keresés feltételei hely, ár és szolgáltatások terén is. A találati listában szálláshelyek láthatók, egy szálláshelyet kiválasztva válnak láthatóvá az ajánlott szobák. Az utazó személyeket 30 felnőtt és 10 gyermek számosságban maximalizálták a keresés során. Egy foglalás csak egy szálláshely kínálatát tartalmazhatja.

A portál rendelkezik értékelési rendszerrel.

### **2.2. Booking.com**

A Booking.com egy nemzetközi szálláskereső portál, amely 2011-ben lépett be a magyar szálláskereső piacra. A Szállás.hu közvetlen riválisának tekinthető, szolgáltatásaik megegyeznek. A szálláskereső körében a portál hasonló népszerűségű, mint a Szállás.hu, emellett nemzetközi jelenléte miatt jelentős a külföldi szálláskereső látogatottsága.

A Szállás.hu-hoz hasonlóan ezen a portálon is részletesen lehet szűrni a szálláshelyek tulajdonságait. A találatok között szintén a szálláshelyek jelennek meg, amelyeknek részletes leírásában tekinthetők meg a szobák. A foglalásban csak egy szálláshely szobái szerepelhetnek. A keresés során maximálisan 30 felnőtt és 10 gyermek választható ki.

A portál rendelkezik értékelési rendszerrel.

### 2.3. Trivago.hu

A Trivago.hu a Trivago nemzetközi szálláskereső szolgáltatás Magyarországra készült változata, amely 2012-ben indult el. A működése eltér az 2.1 és 2.2 fejezetekben tárgyalt portáloktól, ugyanis a Trivago csak összegyűjti más szálláskereső portálok ajánlatait és azok közül keres.

A keresési feltételekkel nagyvonalúan bánt, nem lehet elég részletesen beállítani a kívánalmakat. Lehet szűrni a teljes foglalás ára és a talált szálláshelyek városközponttól számított távolsága alapján. Az előző fejezetekben megvizsgált portálokhoz hasonlóan ez a rendszer sem képes a szobákat vegyesen ajánlani. A keresési találatok mindig egy-egy szálláshelyre vonatkoznak. Az utazó személyek kiválasztásakor maximum 16 felnőtt és 16 gyermek választható ki.

### 2.4. Konklúzió

A magyar szálláskereső piac meghatározó szereplői jó felületet nyújtanak az egyéni utazók számára. A vizsgált portálok előnyben részesítik a szálláshelyeket és jellemzően egy szálláshelyre koncentrálnak ajánlataikat. A keresési szempontokat mindhárom portál esetében kielégítőnek találtam. Az utazó személyek száma a keresés során mindenütt korlátozott. Egyik portál sem képes több szálláshelyről származó szobákat egy foglalásként kezelni.

A kutatásom során nem találtam olyan szálláskereső portált, amely funkcionalitásában közvetlen vetélytársa vagy alternatívája lehetne az általam felvázolt rendszernek.

### 3. Matematikai optimalizálás

A matematikai optimalizálás a modernkori matematika egyik legfontosabb és leggyorsabban fejlődő ágazata. A matematikai optimalizálás az alternatívák halmazán történő legjobb választás problémájával foglalkozik. A problémát korlátozások és célok írják le. A cél meghatározza azokat a feltételeket, amiknek a legjobb választásnak meg kell felelnie. Formálisabban megfogalmazva, a matematikai optimalizálás egy valós függvény maximum vagy minimum értékének meghatározásával foglalkozik.

Optimalizálás helyett gyakran használatos a programozás megnevezés. A kifejezés nem egyenlő a számítógépes programozással. Az elnevezés Dantzig-tól származik, aki az 1940-es években az amerikai hadseregnél foglalkozott az ott programnak nevezett tréning és logisztika megszervezésének problémáin.

A következő fejezetekben bemutatom az optimalizálás kialakulását, a története során megemlítendő fontos személyeket és eseményeket, illetve kitérek a vonatkozó optimalizálási osztályokra és alkalmazásokra.

#### 3.1. A matematikai optimalizálás története

Az optimalizálási feladatok, ha nem is nevezték nevén őket, már régóta foglalkoztatja a matematikusokat és gondolkodókat. Minden korban minden nagy birodalom és városállam vezetői szembesültek a mezőgazdaság és élelmezés, az élelmiszerelosztás problémáival. A hadjáratok során szintén nagy szükség volt kielégítő, de nem pazarló, vagyis végső soron optimális hadtáprendszer kialakítására. Optimalizálási feladatok megoldásával tehát már az ókorban és a középkorban is foglalkoztak a kor tudósai, matematikusai és mérnökei, jóllehet ők maguk ezzel nem voltak tisztában.

Kr.e. 300 körül Eukleidész, görög matematikus geometriai kutatásai során megoldotta a két pont legrövidebb távolságának és az élek teljes hosszával legnagyobb területet lefedő geometriai alakzat jelentette optimalizálási feladatokat. A kutatásának eredménye, hogy két pont között a legkisebb távolság az egyenes, míg a legnagyobb, korlátozott teljes él hosszal lefedhető terület a négyzet. Kr. e. 100 körül Hérón, szintén görög matematikus és gépész *Catoptrica* című művében

bebizonyította, hogy a tükörben visszatükröződő tárgyak fénye a lehető legrövidebb utat járja be. E felfedezés matematikai alapokra helyezve szintén optimalizálási feladatra vezethető vissza.

A variációszámítás kialakulásáig csak pár optimalizálási feladatot vizsgáltak a tudósok. A 17. és 18. században több említésre méltó esemény is köthető a matematikai optimalizáláshoz.

1615-ben Kepler rájön a boroshordó a kor szempontjai szerint optimális méretére. Ezen kívül megalkotja a titkárnő probléma, a dinamikus programozás egy népszerű alkalmazásának korai formuláját, amikor új feleséget keres magának. 1638-ban Galilei másodfokú egyenletekkel próbálja leírni a függő lánc vagy kötél alakját, de kísérletei kudarcot vallanak. Galilei ott ejtett hibát, hogy azt feltételezte a lengő lánc egy hiperbolát formál. Az 1690-ben Jacob Bernoulli által megfogalmazott láncgörbe probléma megoldása szintén matematikai optimalizálásra vezethető vissza. 1646-ban Fermat megmutatja, hogy egy függvény szélsőértékeinél annak gradiense eltűnik. 1657-ben általánosítja Hérón, a fény útjára tett megállapítását, miként a fény bármely két pont között a lehetséges legrövidebb úton halad. Az 1660-as és 1670-es években Newton és Leibniz megalkotják a matematikai analízist, amely a variációszámítás alapjává válik. Ezt követően felgyorsulnak az optimalizálás területén tett felfedezések és áttörések eseményei. 1687-ben Newton a legkisebb légellenállású testet keresi, ami egy minimalizálási problémához vezet. 1696-ban Johann és Jacob Bernoulli a brachistochron probléma kutatása során megteszik az első lépéseket a variációszámítás megszületéséhez.

1712-ben König megmutatja, hogy a méhek által képzett méhsejt forma alakja optimális. Az eredményt a Francia Tudományos Akadémia isteni jelként aposztrofálja. 1740-től kezdve Euler publikációi nyomán globális figyelmet kapnak a variációszámítás területén végzett kutatások. 1746-ban Maupertuis megfogalmazza a legkisebb hatás elvét, amit arra a feltevésére alapoz, hogy a természetes mozgás szükségszerűen minimalizál valamilyen mennyiséget. 1754-ben a 19 éves Lagrange megteszi első felfedezéseit a variációszámítás területén. 1760-ban megfogalmazza Plateau minimális felületekre vonatkozó problémáját.

1930-ban egymástól függetlenül Jesse Douglass és Radó Tibor is megoldást talál a problémára. 1784-ben Monge elkezd vizsgálni a szállítási problémát, amely egy népszerű optimalizálási feladat.

A 19. században Weierstrass, Steiner, Hamilton és Jacobi a variációszámítás területén végzett mélyebb kutatásai nyomán megjelennek az első optimalizálási algoritmusok.

1806-ban Legendre bemutatja a legkisebb négyzetek módszerét, amelyet Gauss is magáénak tulajdonít. A módszer lényege az eltérések négyzetösszegének minimalizálása. 1826-ban Fourier lineáris programozási problémát fogalmaz meg a mechanikában és valószínűség számításban felmerülő problémák megoldására. 1846-ban Faustmann kidolgoz egy formulát az erdők újraterelítésével realizálható bevétel maximalizálására. 1924-ben Bertil Ohlin megoldja Faustmann formuláját, még ha néhány erdész állítólagosan már az 1860-as években megoldotta azt. 1847-ben Cauchy megalkotja a gradiens módszert, amely egy optimalizálási algoritmus. 1857-ben Gibbs megmutatja, hogy a kémiai egyensúly egy energia minimum.

A közgazdaságtanban, az 1870-es években Walras és Cournot munkája nyomán kialakuló határhaszon-elmélet a közgazdászok figyelmét a fogyasztói szükséglet maximalizálására tereli. Az optimalizálás a közgazdaságtan szerves részévé válik.

A variációszámítás és az optimalizálás területei igazi fejlődést a 20. században mutatnak. A század második felétől, a kutatások az elektronikus számítógépek megjelenésével felgyorsulnak.

1902-ben Farkas kidolgozza a Farkas-lemmát. A lemma jelentőségét csak 1950-ben fedezi fel két amerikai matematikus, Kuhn és Tucker. A felfedezés után a lemma a lineáris optimalizálás alaptételévé válik. 1905-ben Jensen kialakítja a konvexitás fogalmát és bemutatja az első konvex függvényeket. Minkowski 1911-ben mutatja be konvex halmazokon végzett vizsgálatainak első eredményeit. 1917-ben megjelenik az első optimalizálással foglalkozó kiadvány, amelynek címe *Theory of Maxima and Minima*, szerzője Harris Hancock. 1925-ben Morse elméletének publikálásával általánosítja a variációszámítás területét. A Morse



elmélet a modern matematikai fizika egyik legfontosabb tétele. 1928-ban Ramsey a variációszámítást használja az optimális gazdasági növekedési vizsgálataihoz. Munkássága az 1950-es években kerül újra elő, az optimális növekedési elmélet fejlesztése során. 1932-ben Menger általánosan megfogalmazza az utazó ügynök problémáját. 1939-ben Kantorovich publikálja lineáris programozási modelljét és megoldó algoritmusát a problémára. Később, 1975-ben Kantorovich és Koopmans a munkájukért Közgazdasági Nobel-emlékdíjat kapnak.

A II. világháború után az optimalizálás az operációkutatással párhuzamosan fejlődik. Az operációkutatás legnagyobb alakja Neumann, aki 1944-ben Morgensternnel együtt szekvenciális döntési problémákat oldanak meg dinamikus programozás alkalmazásával. 1947-ben az Amerikai Légierőnél dolgozó Dantzig meglakotja a lineáris programozási feladatokat megoldó Szimplex módszert, ugyanebben az évben alakítja ki Neumann a lineáris programozási problémák dualitás-elméletét. 1949-ben megtartják az első nemzetközi optimalizálásról szóló matematikai konferenciát Chicagóban *International Symposium on Mathematical Programming* címmel. 1951-ben Kuhn és Tucker, John (1948) és Karush (1939) után újra felfedezik nemlineáris problémák optimalitási-korlátait. 1954-ben Ford és Fulkerson hálózati problémák körében végzett kutatási nyomán kialakul a kombinatorikus optimalizálás.

Az 1950-es évek második felétől az űrverseny ad újabb lökést az optimalizálás területén, a szabályozáselmélet megjelenésével pedig főleg az optimális szabályozás elméletének területén. 1956-ban Pontryagin és kutatócsoportja bemutatja a Maximum-elvet. A következő évben Bellman publikálja az Optimum-elvről szóló munkáját. 1960-ban Zoutendijk módszereket mutat be, amikkel a Szimplex módszer általánosítható és alkalmazható nemlineáris problémákon. Ugyanekkor Rosen, Wolfe és Powell is hasonló eredményekről számol be. 1963-ban Wilson elsőként találja fel a szekvenciális kvadratikusan programozást. A módszert később Han (1975) és Powell (1977) is sajátjaként mutatja be. 1984-ben Karmarkar lineáris programozási problémákhoz kifejlesztett polinomiális idejű algoritmusát fellendülést hoz a belső pont módszerek használatában. Az 1960-70-es években kialakuló komplexitás elmélet érezhető hatást gyakorol az optimalizálás területén végzett kutatásokra. Az 1980-as évektől

elérhetővé váló egyre olcsóbb és hatékonyabb számítógépek a globális optimalizálás és a nagyméretű problémák megoldására tereli a hangsúlyt. Az 1990-es években a belső pont módszereket kiterjesztik a szemidefinit optimalizálás területére.

#### 3.2. Matematikai optimalizálási feladat

A matematikában és a számítástudományban optimalizálási problémának nevezünk egy feladatot, ha a cél egy probléma lehetséges megoldásai közül a legjobbat kiválasztani. Egy optimalizálási feladat a következő formában írható fel:

Adott egy  $f: A \rightarrow \mathbf{R}$  függvény, ami az  $A$  halmazból a valós számokba képez

Keressük  $x_0$ -t úgy, hogy  $\forall x \in A \mid f(x_0) \leq f(x)$  minimalizálás vagy  $\forall x \in A \mid f(x_0) \geq f(x)$  maximalizálás esetén.

A fenti formulával sok, valós életbeli probléma általánosan modellezhető. A fizika, illetve a gépi látás területén a fenti formulát az energiaminimalizálás modellezésére használják, ahol  $f$  a modellezett rendszer energiája.

Az  $A$  halmaz jellemzően az Euklidészi  $\mathbf{R}^n$  tér részhalmaza, ahol  $A$  elemeinek egy sor egyenlőségi és egyenlőtlenségi feltételnek kell megfelelniük. Az  $f$  függvény értelmezési tartománya, más szavakkal a keresési terület vagy választási halmaz.  $A$  elemeit lehetséges megoldásoknak nevezzük. Az  $f$  függvénynek több elnevezése létezik. Általában célfüggvénynek nevezzük, minimalizálás esetén használatos a költségfüggvény, maximalizálás esetén a hasznossági függvény, egyes alkalmazási területeken az energiafüggvény elnevezés. Egy lehetséges megoldást, ami céltól függően minimalizálja vagy maximalizálja a célfüggvényt, optimális megoldásnak nevezünk.

A matematikában általánosan elfogadott, hogy minden optimalizálási problémát minimalizálásként kell felírni. Általánosságban, ha a célfüggvény és a megoldási halmaz nem konvex, a minimalizálási problémának több lokális minimuma is létezhet. Egy lokális minimum az az  $x^*$  pont, aminek létezik olyan  $\delta > 0$  környezete ahol nem található nálánál kisebb érték. Formálisan:

$$\|x - x^*\| \leq \delta; f(x^*) \leq f(x)$$

A lokális maximum a fenti formulához hasonlóan definiálható.

A nagy számban létező nem-konvex problémákat megoldó algoritmusok nem tudnak különbséget tenni a lokális és globális minimum között, és a probléma megoldásaként a helyi minimumot szolgáltatják. Az alkalmazott matematika és a numerikus analízis globális optimalizálás ágazata foglalkozik olyan determinisztikus algoritmusok kifejlesztésével, amelyek véges időn belül képesek a valós minimális megoldáshoz konvergálni.

### 3.3. Lineáris optimalizálási feladat

A lineáris optimalizálás a matematikai optimalizálás egy speciális esete. A lineáris optimalizálás módszerével megoldhatók azon optimalizálási feladatok, ahol a célfüggvény lineáris függvény és a korlátozások lineáris egyenlőségek vagy egyenlőtlenségek. A lineáris optimalizálási feladat megoldási halmaza egy konvex politóp, amit véges sok fél-tér határol, amelyek mindegyikét lineáris egyenlőtlenségek határoznak meg. A probléma célfüggvénye a poliéderen értelmezett valós értékű affín transzformáció. A lineáris optimalizálási algoritmus a poliéderen keresi azt a pontot, ahol a célfüggvény értéke optimális.

A lineáris optimalizálási feladat általános mátrix alakja a következőképpen írható fel:

$$\begin{aligned}x &\geq 0 \\A \cdot x &\geq b \\z = c^T \cdot x &\rightarrow \min;\end{aligned}$$

vagy másként:

$$\min\{c^T x : Ax \geq b, \quad x \geq 0\},$$

ahol  $A \in \mathbf{R}^{m \times n}$ ,  $b \in \mathbf{R}^m$ ,  $c, x \in \mathbf{R}^n$ . A formulában  $x$  a változók vektora,  $c$  a célfüggvény együtthatóinak vektora,  $b$  kapacitásvektor,  $A$  pedig a technikai együtthatómátrix. Az  $Ax \geq b$  és  $x \geq 0$  korlátozások határozzák meg azt a konvex politópot, ami felett a célfüggvény optimalizálandó.

Az optimális megoldás létezése nem szükséges feltétel. Tekintsük a következő két korlátozást  $x$ -re:  $x \geq 2$  és  $x \leq 1$ . Ez esetben a korlátozások által képzett tereknek nincs metszete, az optimalizálási feladat megoldhatatlan. Egy másik példában tegyük fel, hogy a korlátozások által képzett politóp korlátlan a célfüggvény gradiensének irányában (amely esetben a célfüggvény gradiense megegyezik a célfüggvény együttható vektorával). Ekkor belátható, hogy az optimális megoldás véges időn belül nem elérhető. Azonban a lineáris optimalizálás alaptétele kimondja, hogy ha egy lineáris problémának van optimális megoldása, akkor, az megtalálható a konvex poliéder sarokpontjainak vizsgálatával.

A lineáris optimalizálási feladatok megoldására alapvetően három módszer kínálkozik. Az egyszerű, legfeljebb kétváltozós problémák megoldhatóak grafikus úton, míg a bonyolultabb feladatokhoz valamilyen Szimplex vagy belsőpontos módszert alkalmazó algoritmus használható.

#### **3.4. Nemlineáris optimalizálási feladat**

Nemlineáris optimalizálásról beszélünk akkor, ha a megoldandó probléma egyenlőségekkel vagy egyenlőtlenségekkel korlátozott, valós értékű változókat alkalmaz, a cél maximalizálás vagy minimalizálás és a korlátozások vagy a célfüggvény bármelyike nemlineáris függvény. A nemlineáris optimalizálás a lineáris optimalizálás általánosítása.

#### **3.5. Alkalmazási területek**

A matematikai optimalizálás, azon belül a lineáris és nemlineáris optimalizálás széleskörűen alkalmazott világszerte, szinte minden tudományterületen és az iparban. Optimalizálási feladatok gyakran előfordulnak az operációkutatás, logisztika, közgazdaságtan, kémia és fizika területén. Ilyen jellemző problémák például a portfólió kiválasztás, ellátás tervezés, ütemezés, fuvarozási és egyéb logisztikai problémák vagy a hálózattervezés.

### 4. Ruby on Rails

A Ruby on Rails – röviden Rails – egy közel 10 éves múltra visszatekintő, nyílt forráskódú, Ruby nyelven íródott, webalkalmazások készítésére alkalmas keretrendszer. Világszerte népszerű a fejlesztők körében és több neves weboldal is ezt a rendszert használja, úgy, mint a Twitter, a GitHub vagy a Shopify.

A Ruby nyelv, amely a Rails alapját képezi, egy nyílt forráskódú, az egyszerűséget és a produktivitást szem előtt tartó, erősen objektum orientált programozási nyelv. Megalkotója Yukihiro “Matz” Matsumoto japán programozó, aki 1995 decemberében tette közzé a nyelv első változatát.

David Heinemeier Hansson a 37signals számára készített projektmenedzsment rendszer fejlesztése közben alkotta meg a Ruby on Rails keretrendszert. A keretrendszer első, nyílt forráskódú kiadása 2004-ben történt, Hansson azonban 2005 februárjáig nem adott változtatási jogot senkinek. Az 1.0 verzió 2005 decemberében került kiadásra. A jelenlegi legfrissebb verzió a 4.2.1, amely 2015 márciusában jelent meg. Érdekes, hogy a keretrendszer bárki számára elérhető *repository*-ja a GitHub-on<sup>1</sup> szabadon böngészhető, amely szintén a Rails keretrendszert használja. A *repository* 2015 tavaszán több, mint 50.000 *commit*-ot, 259 kiadást és közel 2700 közreműködő fejlesztőt számlál. A keretrendszer már a kezdetektől hatalmas népszerűsége tette szert, igazi fénykorát 2005 és 2008 között élte. Bár a Python Django már 2006 óta ellenfele és a Java alapú keretrendszerek mindig is népszerűbbek voltak nála az üzleti megrendelők körében, a piaci részesedése a mai napig meghatározó. A 2010-es évektől a Node.js, a Laravel és egyéb divatos PHP és Javascript alapú keretrendszerek hasonló népszerűségi növekedést mutatnak, mint a Rails a 2000-es évek közepén. Manapság a Ruby on Rails népszerűsége szinte megegyezik a Java alapú keretrendszerekkel, de elmarad a divatos PHP és Javascript alapú megoldásoktól.

A Rails egy szoftver könyvtár, amely a Ruby nyelv kiegészítése, és mint olyan, a RubyGem csomagkezelővel telepíthető, több más kiegészítővel együtt. A Rails a Ruby nyelvet kombinálja webes technológiákkal, úgy, mint a HTML, a CSS

---

<sup>1</sup> <https://github.com/rails/rails>

és a Javascript. Működését tekintve szerveroldali megoldás. A Rails alkalmazások az MVC (Model-View-Controller) tervezési mintát követik. A keretrendszer működése és használata négy alapelvet követ:

##### 1. A Rails makacs

A '90-es években a webalkalmazásokat főleg Perl nyelven írták, amely úgy hirdette magát, mint egy nyelv, amiben egy probléma számtalan módon megoldható. Ezzel szemben a Rails keretrendszerben minden gyakori, a webfejlesztés során előforduló problémának létezik egy jó megoldása, az úgynevezett *Rails way*. Ha a fejlesztő követi a Rails ajánlott konvencióit, akkor kevesebb döntéssel szembesül, tudván, hogy a legtöbb problémának már létezik használható megoldása.

##### 2. A Rails omakase

Az *omakase* egy japán kifejezés, jelentése „*Csináld ahogy akarod!*”. A kifejezés a sushi éttermekben használatos, amikor a vendég egy *omakase* fogást kér a séftől. Ekkor a séf összeállít egy saját ízlése és stílusa szerint kellemes válogatást. A Rails omakase jellege azt jelenti, hogy a vezető fejlesztők – köztük az alapító, Heinemeier Hansson – tapasztalataik és egyéni döntéseik alakítják a keretrendszert. Természetesen a fejlesztők számításba veszik más, a projektben résztvevő fejlesztők véleményét is.

##### 3. Konvenció a konfiguráció felett (Convention over configuration)

A Rails keretrendszer egy gazdag, konvenciókra épülő mechanizmust mutatott be a 2000-es évek közepén. Jelentősége, hogy egy alkalmazás a működéséhez nem igényel rengeteg konfigurációs fájlt. A Rails rendszer feltételezi, hogy a fejlesztő követi a szabályokat. A szabályok leginkább a modellek, vezérlők, adatbázis táblák és metódusok nevezéktanát határozzák meg. Sokszor fontos, hogy az angol helyesírás szerint helyesen határozzuk meg a neveket, mert a rendszer több helyen támaszkodik az egyes számú elnevezésből képzett többes számú kifejezésekre. A konvenciók szigorú megkövetelése a produktivitást helyezi előtérbe.

### 4. Ne ismételd magadat (Don't Repeat Yourself, DRY)

Az angol kifejezésből képzett *DRY* mozaikszóként emlegetett szoftverfejlesztési alapelve Andy Hunt és Dave Thomas fogalmazta meg és a Rails fejlesztők körében erősen szorgalmazott szemléletmód. Az elv lényege, hogy a fejlesztő kerülje el a kód ismétlést, mert minden duplikátum bonyolítja az elkészülő rendszert. Az elvhez szorosan kötődnek a hatékony szoftvertervezési minták és a kód újrahasznosítás technikája.

A produktivitást és gyors fejlesztést támogatja a RubyGems csomagkezelő, amellyel rengeteg, egyéni és általános problémát megoldó kiegészítő modul telepíthető és illeszthető a Rails alkalmazásokhoz. A kiegészítők összefoglaló neve *gem*.

## 5. Specifikáció

A feladat teljesítéséhez egy webalkalmazás tervezése és implementálása volt a cél. A webes technológia választásának oka, hogy a már megszokott és ismert szálláskereső portálokhoz hasonuljon. Ezen kívül az internetes platformra való fejlesztéssel lehet a legolcsóbban és leggyorsabban a legszélesebb felhasználói kört elérni. A manapság rendelkezésre álló úgynevezett *responsive*, magyarul alkalmazkodó web dizájnok alkalmassá tesznek egy weboldalt arra, hogy egyszerre legyen áttekinthető és kezelhető minden képernyőméreten.

### 5.1. Szereplők

A tervezett rendszerben négy felhasználói szerepkör különül el, amelyek a következők:

1. **Látogató:** bejelentkezés nélkül böngészi a portál publikus tartalmát.
2. **Szálláskereső:** bejelentkezés után szobát keres és foglal
3. **Szállásadó:** bejelentkezés után szobákat hirdet, foglalásokat kezel
4. **Adminisztrátor:** bejelentkezés után a rendszer törzsadatait és beállításait kezeli

### 5.2. Funkcionális követelmények

A fejezet a webalkalmazással szemben támasztott funkcionális követelményeket és elvárásokat taglalja.

#### 5.2.1. Felhasználói fiókok

A rendszernek tudnia kell kezelnie az 5.1-ben meghatározott felhasználói szerepköröket. Minden bejelentkezéshez kötött szerepkörnek tudnia kell regisztrálni, bejelentkezni és kijelentkezni a rendszerből. Adminisztrátort csak adminisztrátor regisztrálhat. A rendszernek tudnia kell szerepkörönként eltérő tartalmat megjeleníteni.

#### 5.2.2. Szobák szűrése

A látogatónak és a szálláskeresőnek lehetőséget kell biztosítani a szobák szűrésére. A szűrési feltételek között szerepelnie kell a szálláshely szolgáltatásainak, a szoba



felszereltségének, a szoba elérhetőségét jelző kezdő- és végdátumnak, a szoba típusát jelző ágyak számának valamint a városnak.

### **5.2.3.Szobafoglalás**

A szálláskereső csak a kiválasztott időszak szerint a rendszerben elérhetőként nyilvántartott szobákat foglalhatja le. A foglalás véglegesítése előtt a szálláskeresőnek minden vendég adatát meg kell adnia.

A szobafoglalásról minden szállásadónak egyénileg kell visszajelzést készítenie. A szobafoglalást el lehet fogadni és vissza lehet utasítani. Egy foglalás akkor tekinthető teljesíthetőnek, ha minden szállásadó pozitív visszajelzést küldött. A foglalás nem teljesíthető, ha legalább egy szállásadó negatív visszajelzést küldött.

A szobafoglalások a szálláskereső és a szállásadó részéről is legyenek bármikor visszakereshetők és megtekinthetők.

### **5.2.4.Értékelés**

A teljesült szobafoglalások esetén, az utazás befejező dátumát követően a szálláskereső értékelheti a meglátogatott szálláshelyeket szövegesen és egy 1-től 10-ig terjedő skálán, ahol 10 a legjobb értékelés..

### **5.2.5.Intelligens keresés**

Az intelligens keresés funkció ár és távolság, vagy ezek kombinációja szerint legyen képes automatikus szoba ajánlást készíteni. A választható szempontok mellett figyelembe kell vennie a szálláshelyek értékeléseit és törekednie kell a jobb értékelésűek ajánlására.

### **5.2.6.Törzsadatok**

Az adminisztrátornak a rendszerben megjelenő törzsadatokat tudnia kell szerkeszteni és bővíteni.

### **5.2.7.Tartós címek**

A rendszerben megjelenő oldalak címeit és a keresések eredményoldalaira mutató címeket úgy kell kialakítani, hogy azok bármikor újra meglátogathatóak és linkelhetőek legyenek.

### 5.3. Célcsoport

A webalkalmazás felhasználói célcsoportjaként a szállásadó szerepkör részéről a jellemzően vidéki, alacsony kapacitású panziókat és apartmanokat azonosítottam. Számukra a rendszer ugyanúgy a foglalások egyszerű kezelhetőségét nyújtja, mint a szálláskeresők számára. A szálláskereső szerepkör szempontjából a célcsoport tagjaiként az iskolai kirándulásokat szervező osztályfőnökök, a baráti társaságok, illetve az üzleti célból szállást kereső szervezőket tekintem.

## 6. Tervezés

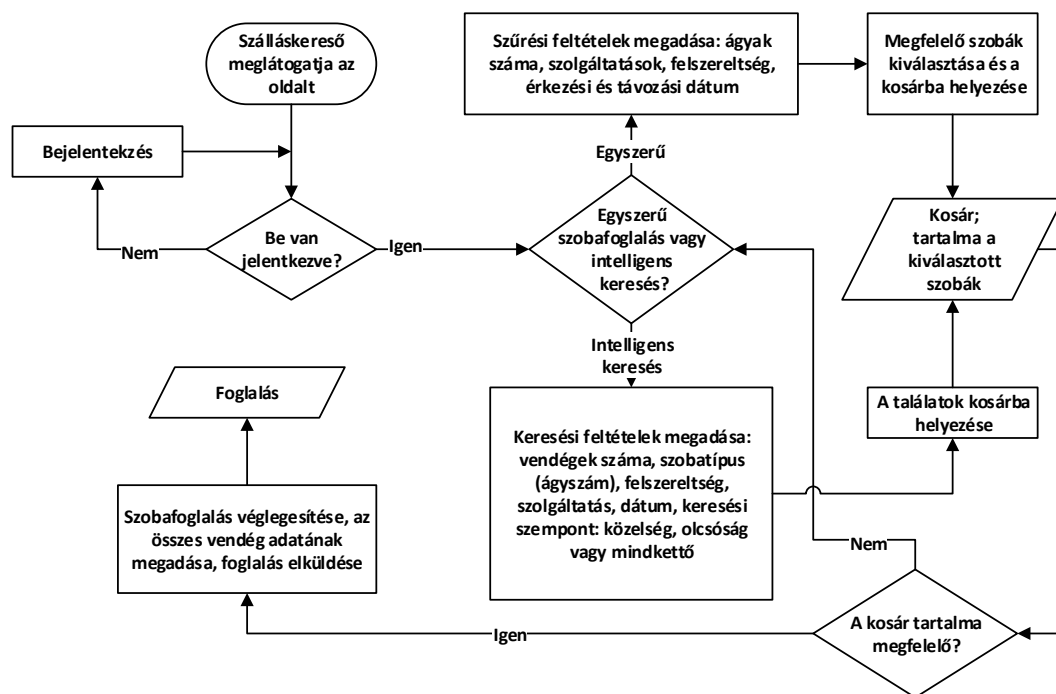
A fejezet a feladat megvalósításához szükséges tervezés eredményét mutatja be. A fejezet kitér az alkalmazásban megjelenő folyamatok tárgyalására, bemutatja az intelligens keresés működéséhez szükséges optimalizálási modelleket. A fejezet második felében a tervezett adatbázis entitásai és a megvalósítás során felhasznált technológiákról lesz szó.

### 6.1. A rendszerben megjelenő fő folyamatok

Ez a fejezet a rendszerben megjelenő fő interakciós és háttérfolyamatokat mutatja be.

#### 6.1.1. Szobafoglalás

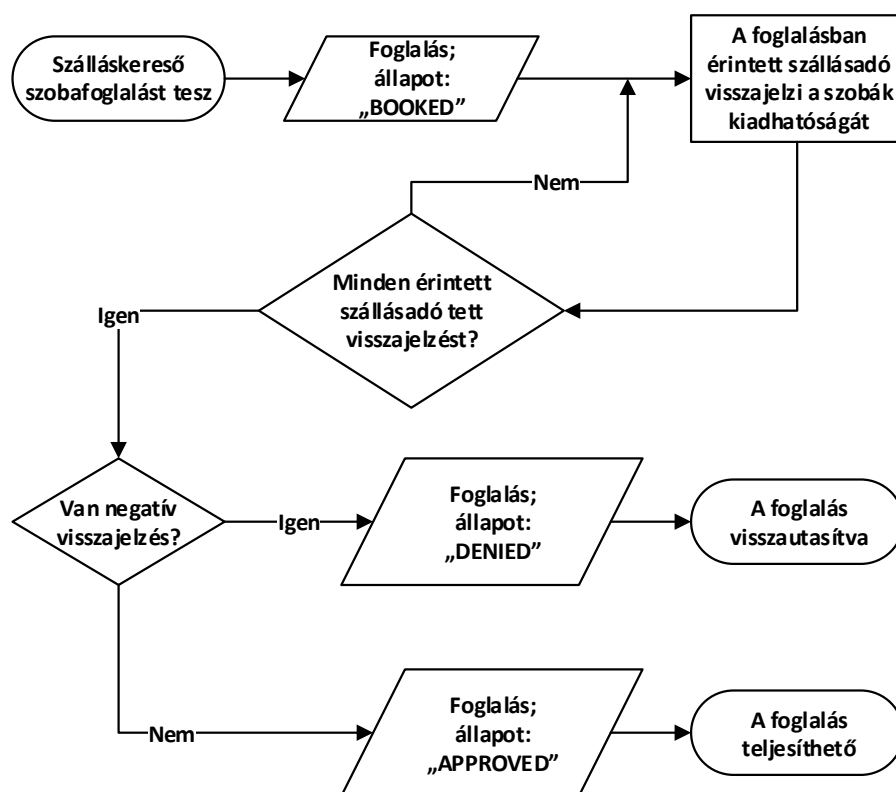
A szobafoglalás folyamatában a bejelentkezett szálláskereső valamelyik keresési mechanizmust választva feltölti a virtuális kosarát a foglalni kívánt szobákkal. A kosár feltöltése után a szálláskereső véglegesíti a foglalását, megadja a foglalásban részt vevő vendégek adatait és a foglalást elküldi. Az alábbi ábra a folyamat lépéseit mutatja be részletesen.



6.1 ábra Szobafoglalás folyamata

### 6.1.2. Foglалás visszaigazolás

A rendszerbe érkező szobafoglalásokat a szállásadóknak egyénileg vissza kell igazolniuk. A foglalás állapota csak akkor változhat meg, ha minden, a foglalásban részt vevő szállásadó megtette visszajelzését. Az alábbi ábra bemutatja a visszaigazolás folyamatát a foglalás szempontjából.

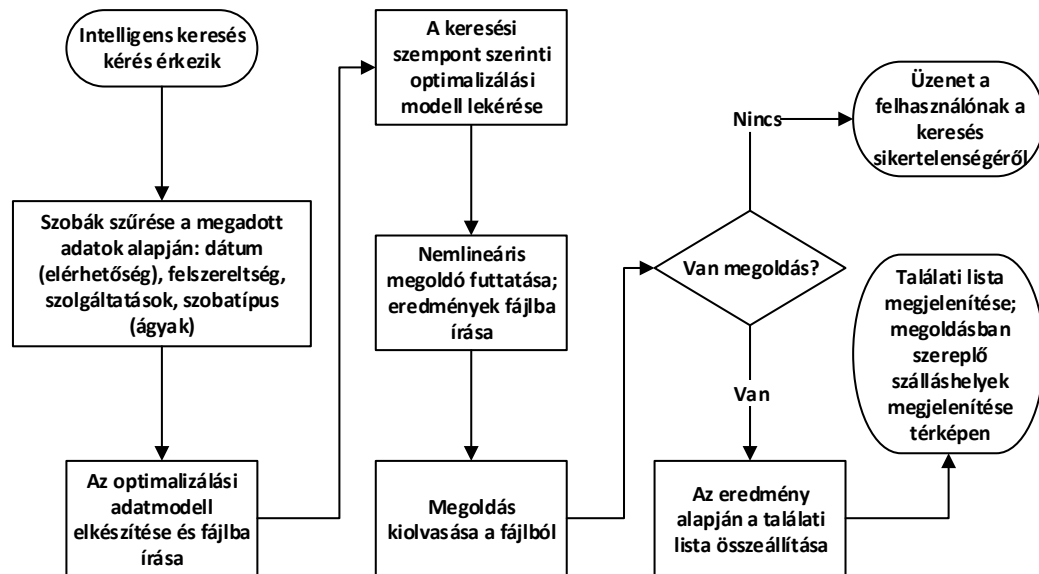


6.2 ábra Foglалás visszaigazolás folyamata

Miután minden szállásadó visszaigazolta a rá vonatkozó szobákat, a rendszer új állapotba lépteti a foglalást. A foglalás teljesíthető állapotúvá válik, ha minden szállásadó pozitív visszajelzést adott. A foglalás nem teljesíthető állapotú lesz, ha legalább egy szállásadó negatív választ adott.

### 6.1.3. Intelligens keresés

Az intelligens keresés háttérfolyamatát a rendszer a szálláskereső által megadott keresési feltételek alapján végzi el. A folyamat lépéseit részletezi az alábbi ábra.



6.3 ábra Intelligens keresés háttérfolyamata

A rendszer nem tud közvetlenül kommunikálni a nemlineáris feladatmegoldó eszközzel ezért előbb a szűrési feltételek szerint kiválogatott szobák alapján elkészíti az optimalizáláshoz szükséges adatmodellt és azt, az adatbázisból kiolvasott optimalizálási modellel együtt fájlba írja. Ezután parancssorból végzi a nemlineáris megoldó futását és az eredmények kiolvasását.

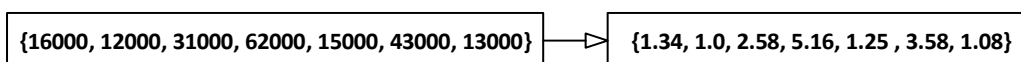
## 6.2. Nemlineáris optimalizálási modell

A nemlineáris optimalizálás során a cél az, hogy az ár, a távolság, illetve a minőség szempontjából optimális megoldást kell találni. A minőség, vagyis a szobák a szálláshelytől örökölt értékelése minden modellben megjelenik, hiszen cél az is, hogy a szálláskereső számára nem csak racionálisan, de emocionálisan is elfogadható megoldást kínáljon a rendszer. Az ár és a távolság választható külön-külön és együttesen is. Tehát három különféle modellt kellett kialakítanom.

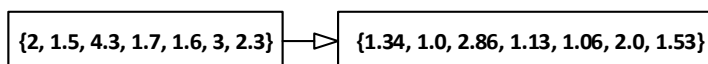
A modellek kialakítása során figyelembe kellett vennem, hogy a különböző szempontokhoz különböző nagyságrendű és szórású értékek tartoznak. Az ár jellemzően tízezres nagyságrendű érték. A távolság jellemzően a pár tizedes és pár tízes nagyságrend között mozog. Az értékelés egy rögzített, 1-től 10-ig terjedő skálán számított átlagos érték. A nemlineáris modellben a célfüggvény a kifejezés minimalizálására törekszik. Ezáltal belátható, hogy a nagyobb nagyságrendű

értékektől fog függni a megoldás. Ez a megoldás szempontjából nem megfelelő, hiszen minden szempontnak egyenlően kell teljesülnie.

A különböző nagyságrendű értékeket két módszerrel tettem összehasonlíthatóvá. Az első módszerem az, hogy az ár és távolság értékeket nem közvetlenül használom fel az adatmodellben. Az adatmodellbe jegyzés előtt az értékeket átalakítom. Az átalakítás során az értékeket a sokaság legkisebb értékéhez viszonyítom, és azt határozom meg, hogy egy érték a legkisebb érték hányszorosa. Az átalakításra mutat példát az alábbi két ábra.



6.4 ábra Árak átalakítása (Ft)



6.5 ábra Távolságok átalakítása (km)

Az árak és távolságok új értékeinek nagyságrendje és szórása is hasonló, tehát az értékek összehasonlíthatók. A módszer előnye továbbá, hogy megtartja a kiugró szélsőértékeket.

A fenti konverziós módszert az értékelésekre nem alkalmaztam, mert azok rögzített skálán, alapjában véve is egy 10-hez viszonyuló értékeket tartalmazó sokaság. Mivel azonban a célfüggvény kiértékelése során a cél az, hogy a találatokra általánosan legyen igaz az ár, távolság és értékelések támasztotta feltételek, ezért az ár, távolság és értékelés sokaságok tulajdonságát vizsgálom relatív szórás útján. Egy olyan speciális relatív szórási képletet alkalmazok, ahol a szórást nem a középértékhez közelítem, hanem az ár- és távolság esetében a mindenkor legkisebb 1 értékhez, míg az értékelések esetében, a legnagyobb 10 értékhez. A relatív szórás eredménye egy százalékérték, vagyis a célfüggvény három százalékérték összegét minimalizálja. Az alábbi képlet az alkalmazott relatív szórási képletet mutatja be, ahol  $s_i$  a vizsgált sokaság egy értéke,  $x_i$  a bináris súly,  $s_{min}$  a vizsgált sokaság lehetséges legkisebb értéke.

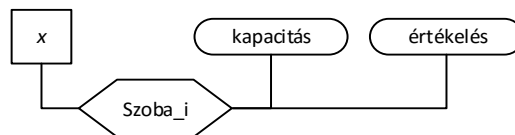
$$V_s = \frac{\sqrt{\frac{\sum_{i=1}^n x_i (s_i - s_{min})^2}{\sum_{i=1}^n x_i}}}{s_{min}}$$

### 6.1 képlet Speciális relatív szórás képlet

A fenti módszerekkel el tudtam érni, hogy több, különböző nagyságrendű sokaságot összehasonlítsak és a célfüggvény kiértékelésekor az algoritmus azokat egyenlőként kezelje.

A 6.1 ábrán bemutatott képletből belátható, hogy az optimalizálás célfüggvénye nem lineáris, vagyis az optimalizálási feladat egy nemlineáris probléma megoldása.

A három kialakított modell mindegyike bináris változókat használ, amik azt mutatják, hogy mely szobákat kell a megoldáshalmazba beválasztani. A modellekhez alapvetően két adatra van szükség. Az első a szobák halmaza, amely minden eleméhez tartozik két bázisparaméter, amely minden esetben megadandó. Ez a kapacitás és az értékelés. A második szükséges adat a vendégek száma. Az optimalizálási szempontok szerint a szobák további paraméterekkel bővülnek. Az alábbi ábra a szobák halmazának egy elemét és a hozzá kapcsolódó változót, valamint a két bázisparamétert mutatja be.



**6.6 ábra A modellben megjelenő szoba objektum a hozzá kapcsolódó változóval és bázisparaméterekkel**

Mindhárom modellben egy alapvető korlátozást vezettem be, egyértelmű módon azt, hogy a kiválasztott szobák kapacitása legyen egyenlő a vendégek számával.

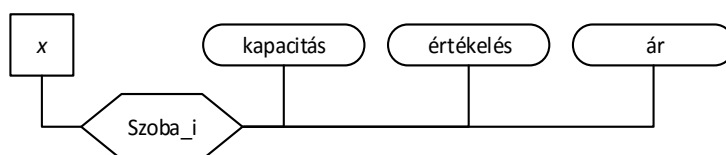
$$\sum_{i=1}^n x_i * c_i = v$$

### 6.2 képlet Korlátozás a vendégek száma alapján

A fenti képletben  $f_i$  bináris változó,  $c_i$  az  $i$ -edik szoba kapacitása,  $v$  pedig a vendégek száma.

### 6.2.1. Olcsó modell

Az olcsó modell azokat a szobákat adja eredményül, amelyek a legolcsóbbak és a lehető legmagasabb értékeléssel bírnak.



6.7 ábra Az olcsó modellhez szükséges paraméterek

Ahogy azt a fenti ábra is mutatja, ehhez a modellhez a szoba halmaz paraméterlistáját ki kell egészíteni a szobák árával.

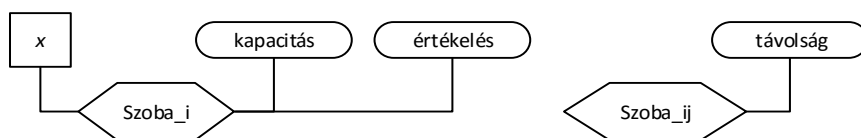
$$\min \frac{\sqrt{\frac{\sum_{i=1}^n x_i (p_i - p_{\min})^2}{\sum_{i=1}^n x_i}}}{p_{\min}} + \frac{\sqrt{\frac{\sum_{i=1}^n x_i (r_i - r_{\max})^2}{\sum_{i=1}^n x_i}}}{r_{\max}}$$

6.3 képlet Az olcsó modell célfüggvénye

A modell célfüggvényét a fenti ábra mutatja be, ahol  $p_i$  az  $i$ -edik szoba ára,  $p_{\min}$  a legalacsonyabb szobaár a sokaságban,  $r_i$  az  $i$ -edik szoba átlagos értékelése  $r_{\max}$  pedig a lehetséges legmagasabb értékelés.

### 6.2.2. Közeli modell

A közeli modell azokat a szobákat választja ki, amelyek egymáshoz képest a legközelebb helyezkednek el és a lehető legmagasabb értékeléssel bírnak.



6.8 ábra A közeli modellhez szükséges paraméterek

A távolságok tárolásához egy, a szobák halmazán képzett Descartes szorzatból kialakított mátrixra van szükség. Ezen a mátrixon van értelmezve a távolság paraméter, ahogy az a fenti ábrán is látható. A közös szálláshelyen lévő



szobák távolsága 0, azonban, hogy a relatív szórás számolható legyen, minden távolság értéket 1-gyel növeltem.

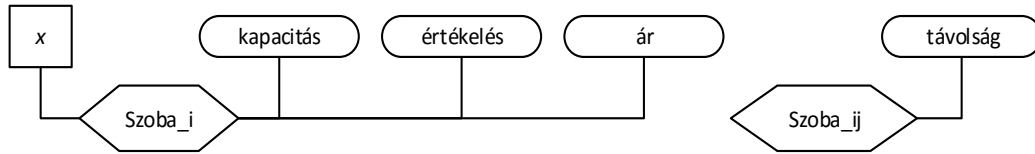
$$\min \frac{\sqrt{\frac{\sum_{i=1}^n \sum_{j=1}^n x_i x_j (d_{ij} - d_{min})^2}{\sum_{i=1}^n x_i}}}{d_{min}} + \frac{\sqrt{\frac{\sum_{i=1}^n x_i (r_i - r_{max})^2}{\sum_{i=1}^n x_i}}}{r_{max}}$$

#### 6.4 képlet A közeli modell célfüggvénye

A modell célfüggvényét a fenti ábra mutatja be, ahol  $d_{ij}$  az i-edik és j-edik szoba távolsága,  $d_{min}$  a legalacsonyabb távolság a sokaságban,  $r_i$  az i-edik szoba átlagos értékelése  $r_{max}$  pedig a lehetséges legmagasabb értékelés.

#### 6.2.3. Olcsó és közeli modell

Az olcsó és közeli modell egyesíti a 6.2.1 és 6.2.2 fejezetekben taglalt modelleket, vagyis az egymáshoz legközelebb eső legolcsóbb és lehető legmagasabb értékeléssel bíró szobákat adja eredményül.



6.9 ábra Az olcsó és közeli modellhez szükséges paraméterek

Az összevont modellnek szüksége van minden, az előző két fejezetben tárgyalt kiegészítő paraméterre, ahogy az a fenti ábrán is látható.

$$\min \frac{\sqrt{\frac{\sum_{i=1}^n \sum_{j=1}^n x_i x_j (d_{ij} - d_{min})^2}{\sum_{i=1}^n x_i}}}{d_{min}} + \frac{\sqrt{\frac{\sum_{i=1}^n x_i (p_i - p_{min})^2}{\sum_{i=1}^n x_i}}}{p_{min}} + \frac{\sqrt{\frac{\sum_{i=1}^n x_i (r_i - r_{max})^2}{\sum_{i=1}^n x_i}}}{r_{max}}$$

#### 6.5 Az olcsó és közeli modell célfüggvénye

Az egyesített célfüggvényt mutatja a fenti ábra, ahol  $d_{ij}$  az i-edik és j-edik szoba távolsága,  $d_{min}$  a legalacsonyabb távolság a sokaságban,  $p_i$  az i-edik szoba ára,  $p_{min}$  a legalacsonyabb szobaár a sokaságban,  $r_i$  az i-edik szoba átlagos értékelése  $r_{max}$  pedig a lehetséges legmagasabb értékelés.

### 6.3. Adatbázis tervezet

A rendszer működéséhez 18 adatbázistáblát terveztem meg. A táblákat, azok attribútumait és a kapcsolatokat az [1] mellékletben található ábra mutatja be.

Az adatbázisban négy tábla, a *User*, az *Admin*, a *Guest* és az *Owner* a felhasználók adatainak tárolásához és azonosításukhoz szükséges. Az *Admin*, *Guest* és *Owner* táblák és a *User* tábla között polimorfikus kapcsolat áll fenn. Ez azt jelenti, hogy a *User* tábla kapcsolódik a három felhasználói tábla egyikéhez, a *role\_id* attribútumban külső kulcsként tárolva annak elsődleges kulcsát, és a *role\_type* mezőben tárolva a hivatkozott tábla nevét. Tehát egy *User* entitás rendelkezik egy szerepkörrel, amit az *Admin*, *Guest* vagy *Owner* táblákkal képzett kapcsolat azonosít.

Hasonlóan a felhasználói táblákhoz, az *Address* táblához is polimorfikus kapcsolatot terveztem. Címe a szálláskereső (*Guest*) felhasználónak és egy szálláshelynek (*Accommodation*) van. Az *Address* tábla az *addressable\_id* attribútumban tárolja külső kulcsként annak az entitásnak az elsődleges kulcsát, amihez a cím tartozik, és az *addressable\_type* mező azonosítja a hivatkozott tábla típusát.

A *Booking* tábla tárolja a szobafoglalásokat. Minden szálláskereső (*Guest*) felhasználó rendelkezik legalább egy *Booking* példánnyal, ami a virtuális kosaraként van dedikálva. A *Booking* tábla *guest\_id* külső kulcsa azonosítja a foglalás tulajdonosát. A *Booking* táblához két kapcsolótáblát terveztem.

A *BookingRoom* kapcsolótábla a foglaláshoz tartozó szobákat tárolja. Mivel a szobák a *Room* táblában nem egyedi példányok, hanem szobatípusok, amelyek a *num\_of\_this* mezőben jelölik számosságukat, ezért minden, a foglalásba beválasztott szobához hozzárendelek egy, a foglalás során egyedi azonosítót, amit az *index* mező tárol. Ez az egyedi azonosító teszi lehetővé, hogy a foglalásban szereplő vendégeket a szobához lehessen csatolni.

A *BookingGuest* kapcsolótábla a foglalásban szereplő vendégeket, és azok elhelyezését tárolja. A *room\_index* mező a *BookingRoom* táblában bejegyzett *index* mezőre hivatkozik, és azt jelenti, hogy a vendég melyik szobában kerül

elhelyezésre. Ezen kívül, a *bed* mező jelenti azt, hogy a vendég a szobában melyik ágyon kap helyet. A *bed* mező főként a weboldalon megjelenő űrlapok mezőinek azonosításakor használatos. A foglalás számából, a szobaindexből és az ágy sorszámból képzett azonosító egyedi a teljes rendszerben.

A szálláshelyek szolgáltatásait az *Accommodation* táblához az *AccommodationService* kapcsolótáblán keresztül hozzárendelt *Service* entitásokkal lehet tárolni. A *Service* tábla neve azért tartalmaz két *i* betűt, mert a Ruby on Rails keretrendszer egy védett kulcsszava a *service*. A *category* szintén védett szó, ezért kell a szálláshely kategóriákat tároló táblát *Category*-nek nevezni. A szobák (*Room*) felszereltségét az *Equipment* tábla tárolja és az *EquipmentRoom* táblán keresztül kapcsolódnak a szobákhoz.

Az értékeléseket a *Comment* tábla tárolja. A *Comment* táblában hivatkozás van a foglalásra a *booking\_id* külső kulccsal, mert egy vendég több foglalás útján újra és újra tehet értékelést. Az értékelés egy szálláshelyről szól, ezért szükséges az *accommodation\_id* külső kulcs. A *guest\_id* külső kulcs azonosítja az értékelő vendéget.

A rendszer beállításait és paramétereit a *Property* tábla tárolja. A *Property* táblában a sorok kulcs-érték pároknak tekinthetők. A *group* mező az összetartozó bejegyzéseket megjelölésére használható.

### 6.4. Technológia

Ebben a fejezetben a fejlesztés során felhasznált technológiák kerülnek bemutatásra.

#### 6.4.1. Keretrendszer

Az 5. fejezetben meghatározott cél, hogy a feladatot egy webalkalmazás képében oldjam meg. Ehhez több programozási nyelv még több keretrendszere közül válogathattam. A tapasztalataim alapján és gyors, hatékony fejlesztés ígérete miatt a Ruby on Rails keretrendszert választottam, amit a 4. fejezetben részletesen bemutatam.

A Ruby on Rails keretrendszerben való fejlesztés előfeltétele, hogy telepítve legyen a Ruby nyelvcsomag a számítógépre. A Ruby nyelvcsomag telepítése után

telepíteni a kell a *rails* gem-et, aminek segítségével lehetővé válik Ruby on Rails projektek készítése és futtatása.

### 6.4.2. Adatbázis

A webalkalmazás adatbázis megvalósításának a PostgreSQL-t választottam. A PostgreSQL egy több mint 15 éves múltra visszatekintő, nyílt forráskódú, megbízhatónak és stabilnak tartott adatbázis motor. Magában foglalja a legtöbb, az SQL:2008 szabványban meghatározott adattípust, és szinte minden népszerű programozási nyelvhez létezik kommunikációs interfésze, emellett kiválóan illeszkedik a Ruby on Rails környezetbe.

A PostgreSQL adatbázis kétszer is elnyerte a *Linux New Media Award For Best Database* díjat, a *Linux Journal* újságírói pedig ötször is neki ítelték az *Editors' Choice Awards for Best Database* díjat.

A PostgreSQL adatbázis használatához a rendelkezni kell egy lokális vagy távoli kiszolgálóval, amihez a Ruby on Rails alkalmazásban a *pg* gem telepítésével lehet kapcsolódni.

### 6.4.3. Optimalizálási modellezés

Az optimalizálást az AMPL-lel végeztem. Az AMPL egy modellező eszköz, amivel az optimalizálás teljes életciklusát le lehet fedni. Az AMPL része egy részletes és jól dokumentált modellező nyelv. A modellező nyelv használatával az optimalizálási feladat minden eleme leírható kezdve az adattól, a korlátozásokon át a célfüggvényekig. Emellett a nyelv gazdag programozási lehetőségeket kínál ciklusok és elágazások használatával. Rendelkezik számos beépített matematikai függvénnyel és operátorral. Képes az adatot a modelltől elválasztani és ezáltal paraméterezhető modelleket előállítani. Az AMPL-hez modulárisan illeszthetők különféle, az igények szerint választott lineáris és nemlineáris megoldók. Az AMPL mindhárom (Windows, UNIX, Linux) népszerű operációs rendszert támogatja.

Az AMPL nem egy ingyenes eszköz, de kínál lehetőséget a kipróbálásra. A legegyszerűbben hozzáférhető verzió az AMPL Demo Version, amely nem funkcionalitásban, hanem teljesítményben van korlátozva. Az AMPL Demo

Version egy lineáris modellezési feladatnál 500 változót és 500 korlátozást, míg nemlineáris feladat esetén 300 változót és 300 korlátozást képes feldolgozni. A Demo Version-ön kívül létezik egy 30 napos teljes próbaverzió diákoknak. A 30 nap nem hosszabbítható meg és számítógépenként korlátozott. A szakdolgozatomban az AMPL Demo Version-t használtam.

Az AMPL telepítő fájljai letölthetők az AMPL weboldaláról<sup>2</sup>.

### 6.4.4. Nemlineáris megoldó

Az AMPL modellező eszközhöz számos lineáris és nemlineáris megoldó modul is választható. Ezek egy részéért fizetni kell, de vannak nyílt forráskódú, ingyenes modulok is. A nemlineáris megoldók közül három ingyen letölthető csomagot kínálnak: az Ipopt-ot, a Bonmin-t és a Couenne-t. Mindhárom termék a COIN-OR projekt része, de különböző tulajdonságokkal bírnak.

Az Ipopt csak folyamatos nemlineáris problémákat tud megoldani belső pontos módszerrel. Mivel az általam felírt probléma diszkrét bináris változókat használ, ezért nem alkalmazható. Megpróbáltam azonban futtatni, és a megoldó képes optimális megoldást találni, de figyelmen kívül hagyja változók bináris korlátozását.

A Bonmin folytonos és diszkrét változójú konvex problémák globális optimumát szolgáltatja és heurisztikus úton képes nemkonvex problémák megoldására is.

A Couenne megoldó a konvexitástól függetlenül képes megoldani folytonos vagy diszkrét változókkal rendelkező nemlineáris problémákat.

Az általam felírt problémához tehát használhattam a Bonmin és Couenne megoldókat, de a fent bemutatott tulajdonságok miatt először a Couenne-t választottam. A kísérletek azonban rácsáfoltak az elképzelésekre. A Couenne megoldó, az Ipopt-hoz hasonlóan figyelmen kívül hagyta a változók bináris jellegét és helytelen megoldásokat adott. Ezzel szemben a Bonmin használata jónak bizonyult, bármilyen adathalmazon jó megoldást kínált.

---

<sup>2</sup> <http://ampl.com/>

A Bonmin megoldó négy különböző optimalizálási algoritmust tartalmaz, amelyek a következők:

- B-BB: Nemlineáris programozás (NLP) alapú korlátozás és szétválasztás (branch and bound) algoritmus
- B-OA: Külső közelítéssel algoritmus kimondottan vegyes-egész (mixed-integer) nemlineáris problémák megoldására fejlesztve
- B-QG: Quesada és Grossmann korlátozás és vágás (branch and cut) algoritmus
- B-Hyb: Egy hibrid, külső közelítés alapú korlátozás és vágás algoritmus

Ezek közül kísérleti úton választottam ki a B-OA algoritmust, mert szignifikáns különbséget mutatott futási időben a többi algoritmushoz képest.

A nemlineáris megoldó telepítő fájljai letölthetők az AMPL weboldaláról.

### 6.4.5. Megjelenés

A webalkalmazás célcsoportját tekintve fontos, hogy a megjelenő felületek átláthatóak, a szemnek kellemesek és divatosak legyenek. Manapság nem szükséges, hogy egy weboldalhoz a fejlesztője egyedi megjelenésű gombokat, szövegmezőket és egyéb alkotóelemeket tervezzen.

A gyors fejlesztés és a trendek egyszerű követése hívta életre az általános webes megjelenítési csomagokat (angolul: *UI kit*). Ezek olyan ingyenes vagy megvásárolható csomagok, amelyek egy egységes kinézetet biztosító megjelenést ígérnek. A csomagok jellemzően CSS és Javascript fájlokat tartalmaznak. A CSS fájlokban meghatározott osztályokat az egyszerű HTML elemeken kell alkalmazni. A HTML elemek átmaszkolásán túl a legtöbb csomaghoz tartozhatnak animációk, ikonok, összetett építőelemek (pl.: legördülő menü, lebegő ablak, menüsáv) és komplett, az elrendezést segítő rácsszerkezetek (*grid system*). A legtöbb megjelenítési csomag a rácsszerkezetek használatával képes a megjelenést automatikusan bármilyen méretű képernyőhöz igazítani.

A webalkalmazás elkészítéséhez a Bootstrap nevű megjelenítési csomagot választottam. A Bootstrap a Twitter által kifejlesztett és nyílt forráskódúvá tett

megjelenítési csomag. Mivel a Twitter is Ruby on Rails keretrendszert használ, ezért a Bootstrap remekül illeszkedik az én környezetembe is. A Bootstrap a HTML szabvány minden kezelőelemét egyedivé teszi, ezen kívül rendelkezik egy 12 oszloppal operáló rácsszerkezettel, valamint többféle, az összetartozó elemeket egybezáró konténer-elemmel.

A kellemes megjelenés mellett fontos volt, hogy az egyes űrlapelemek és gombok funkcióit ikonok jelezzék a könnyebb érthetőség miatt. A Bootstrap csomag rendelkezik egy korlátozott ikonkészlettel, azonban az általam választott FontAwesome csomag több, mint 500 ikonja gazdagabb megjelenést biztosít.

A fenti csomagokat Ruby gem-ek telepítésével lehet Ruby on Rails alkalmazásban használni. A Bootstrap csomaghoz a *bootstrap-sass* gem-et, a FontAwesome csomaghoz a *font-awesome-sass* gem-et kell telepíteni.

### 6.4.6. Autentikáció és autorizáció

Az 5.2.1-ben meghatározott követelmények valamilyen autentikációs és autorizációs modul kialakításával elégíthetők ki. A Ruby on Rails alkalmazásokhoz több kész megoldás is kínálkozik. Ezek közül én a Devise nevű implementációt választottam.

A Devise egy Warden alapú, a Rails alkalmazások számára készült autentikációs és autorizációs megoldás. A Warden a Rack környezetet használó Ruby alkalmazásoknak nyújt autentikációs szolgáltatást. A Devise teljes egészében támogatja a Rails alkalmazások MVC (Model-View-Controller) architektúráját. Képes egyszerre több felhasználót beléptetve tartani és kezelni. Ezen kívül moduláris felépítésű, így konfigurálható az alkalmazásban való használata. A fő szolgáltatásai, hogy a felhasználókat adatbázisból azonosítja, a jelszavakat BCrypt algoritmussal titkosítva tárolja, kezeli az elfelejtett jelszavakat, képes email értesítéseket küldeni, használ email és jelszó validálást, valamint, hogy használatával időben korlátozhatók a munkamenetek.

A Devise használatához a *devise* gem-et kell telepíteni.

### 6.4.7. Geolokáció

A szálláskeresők számára előnyös, ha a szobák és szálláshelyek böngészése közben a szálláshelyek címeik szerint megjelenítésre kerülnek térképen is. A térképes megjelenítés segítségével a felhasználó könnyebben elhelyezi a szálláshelyet a környezetében.

A szálláshelyek térképen való megjelenítéséhez az első lépés a regisztráció során felvitt teljes cím leképzése koordinátákká. Ehhez egyszerű és kézenfekvő megoldást kínál a Geocoder nevű megoldás. A Geocoder a geokódolásra megjelölt modelleket az adatbázisba mentés előtt megvizsgálja, és a megadott mezők alapján meghatározza a címhez tartozó koordinátákat, amiket a modell *latitude* és *longitude* mezőibe elment. Alapértelmezetten a Google térkép szolgáltatását használja. A Geocoder abban is segít, hogy két pont között meghatározza a légvonalbeli távolságot. Ennek nagy hasznát vettem, amikor elkészítettem a szobák távolságának mátrixát az optimalizáláshoz.

A koordináták megjelenítéséhez a Google Maps-et szerettem volna használni népszerűsége miatt. A Google Maps Ruby on Rails alkalmazásokba való egyszerű integrálását ígéri a Gmaps4Rails nevű megoldás. Használatával nem kell az alkalmazáshoz API kulcsot regisztrálni, a legtöbb konfigurációt elrejt, és kényelmes interfészt biztosít a megjelenő térkép személyre szabására. A térképen megjelenítendő pontokat egy JSON tömbbe foglalva várja.

A fent bemutatott szolgáltatások használatához a *geocoder* és a *gmaps4rails* gem-eket kell telepíteni a Ruby on Rails alkalmazásban.

### 6.4.8. Űrlapok

A Ruby on Rails keretrendszer alapértelmezett űrlapsegédjei megkönnyítik a modellekhez kapcsolódó űrlapok elkészítését, azonban még egyszerűbbé tehető a Simple Form nevű megoldás használatával.

A Simple Form egy olyan űrlapsegéd, amely használatával az űrlapok elkészítésekor nem kell foglalkozni a beviteli mezők típusával, mert azt a Simple Form maga választja ki a beviteli mezőhöz tartozó attribútum alapján. Ezen kívül a



telepítéskor megadott opció eredményeként a beviteli mezőkbe integrálja a megfelelő Bootstrap osztályokat.

A webalkalmazás rendszeresen visszatérő eleme a dátumválasztó beviteli mező. Legtöbbször a szobák elérhetőségének vizsgálatakor és a foglalás időtartamának beállításához használatos. Egy jól használható, intuitív megoldás a Bootstrap 3 Datepicker nevű kiegészítő. Használatával a beviteli mező alatt vagy fölött megjelenik egy ablak, amiben akár másodpercre pontos időpontok is kiválaszthatók. Ahogy az a nevéből is sejthető, megjelenése jól illeszkedik a Bootstrap stílusához. A Bootstrap 3 Datepicker a MomentJS Javascript könyvtárat használja a dátumkezeléshez.

Az intelligens keresés felületén a felhasználónak ki kell választania, hogy milyen szempont szerint (ár, távolság) ajánljon szobákat a rendszer. Ehhez az egyszerű jelölőnégyzet helyett valamilyen látványosabb elemet választottam. A Bootstrap Switch nevű megoldás a jelölőnégyzetekből nagy, színes, felirattal ellátott kétállású kapcsolókat készít.

A Simple Form használatához a *simple\_form*, a Bootstrap 3 Datepickerhez a *bootstrap3-datetimepicker-rails* és *momentjs-rails*, a Bootstrap Switch kiegészítőhöz pedig a *bootstrap-switch-rails* gem-eket kell telepíteni.

### 6.4.9. Képek tárolása és megjelenítése

Hasznos, ha a szobákról és a szálláshelyekről képek is megjelennek az egyes oldalakon. A képek növelik a szálláskereső bizalmát és szűrőként is funkcionálnak. A Ruby on Rails alkalmazásokban megjelenő modellekhez a Paperclip nevű megoldással lehet hatékonyan képeket és egyéb fájlokat csatolni.

A Paperclip kiegészítő a fájlok modellekhez való csatolásán kívül elvégzi azok típus szerinti validációját, a képeket képes átméretezni és előnézeti képeket készíteni. Képek tárolásához szükség van az ImageMagick nevű képfeldolgozó könyvtárra.

A Paperclip kiegészítő használatához a *paperclip* gem-et kell telepíteni.

## 7. Megvalósítás

A következő fejezetek a webalkalmazás funkcionális működésének szempontjából fontos részeinek implementációját mutatják be.

### 7.1. Adatbázis kapcsolat és modellek elkészítése

A fejlesztés során a fejlesztő gépen egy lokális adatbázist üzemeltettem, ehhez kapcsolódott a készülő webalkalmazás. Ruby on Rails környezetben a kevés konfigurációs feladatok egyike az adatbázis kapcsolat beállítása. A Rails alkalmazások megkülönböztetnek fejlesztési (*development*), tesztelési (*test*) és éles (*production*) környezeteket. Ezek, mint névterek léteznek egy Rails alkalmazásban és hozzájuk konfigurációs beállítások és környezeti változók rendelhetők.

Az adatbázis konfigurációját az alkalmazás gyökérkönyvtárában lévő *config* mappában található *database.yml* fájlban kell elvégezni. Egy tipikus konfiguráció a következőképpen néz ki:

```
default: &default
  adapter: postgresql
  encoding: utf8
  password: m3dw3
  username: *****
  host: localhost
development:
  <<: *default
  database: vagato
test:
  <<: *default
  database: vagato_test
production:
  adapter: postgresql
  encoding: utf8
  database: vagato
  username: deploy
  password: *****
  host: localhost
```

A fenti konfigurációból is látszik, hogy az egyező tulajdonságokat nem kell minden környezetben újra definiálni. A YAML leíró nyelv megengedi korábban definiált objektumok használatát későbbi bejegyzések tartalmaként.

A fenti konfigurációban egyedül az *encoding*, az adatbázis karakterkódolását beállító bejegyzés nem kötelező, minden más elengedhetetlen a sikeres kapcsolódáshoz és adatmanipulációhoz. Az *adapter* az adatbázis típusa szerinti interfészt azonosítja, a *host* az adatbázis elérési címe. A *username* és *password* mezők a védett adatbázisok bejelentkezési adatait tárolják.

Az adatbázis kapcsolat beállítása után, ha még nem tettük meg, létre kell hozni az adatbázis példányt. Ezt megtehetjük közvetlenül a kiszolgálón, a választott adatbázis saját mechanikájával, azonban a Rails környezet nyújt egy egyszerű megoldást. A Rake parancssori eszköz *db* névtére rendelkezik több, adatbázis műveleteket végrehajtó paranccsal. Egy adatbázis konfiguráció érvényesítéséhez a következő parancsot kell a parancssorban végrehajtani:

**rake db:create**

A Ruby on Rails alkalmazásokban a modellek és az adatbázis megfelelő táblái között nagyon szoros kapcsolat áll fenn. A modelleket, más nyelvi megvalósításoktól eltérően, nem kell teljesen deklarálni. Ez azt jelenti, hogy nem szükséges felsorolni az attribútumokat, mert azokat a Rails motor az adatbázis tábláiból azonosítja, és az oszlopneveket használja. Egy Ruby on Rails alkalmazásban nem kell az adatelérési műveletek implementációjával foglalkozni, mert Az Active Record nevű szolgáltatás kész megoldást nyújt bármilyen adatmanipulációs művelet elvégzésére. Az egyszerű beillesztések, törlések és frissítések mellett képes bonyolult kapcsolatok és egybeágyazott, összekapcsolt lekérések végrehajtására is. Minden, az *ActiveRecord::Base* osztályból származó modell osztályon végrehajthatók adatbázis műveletek.

Az adatbázis szerkezeti változásait úgynevezett migrációs fájlokban kell bejegyezni. Minden migrációs fájl egy olyan Ruby osztály, amely az *ActiveRecord::Migration* osztályból származik. A fájl nevének tükröznie kell a tartalmában leírt változást. A szobákat tároló *rooms* tábla – kapcsolódó modell: *Room* – létrehozása a következő migrációs kód futtatásával érhető el:

```
class CreateRooms < ActiveRecord::Migration
  def change
    create_table :rooms do |t|
```

```

t.string :name
t.integer :accommodation_id
t.integer :num_of_this
t.integer :capacity
t.text :description
t.timestamps
end
end
end

```

A migrációs fájlokat el lehet készíteni kézzel, menteni az alkalmazás gyökérkönyvtárában a *db*, azon belül a *migrate* mappába kell. A kézi szerkesztést könnyíti meg a Rails környezet másik népszerű szolgáltatása, a generátorok. Sokféle generátor létezik, ezek közül egy, amelyik modelleket hoz létre. Ahhoz, hogy a fent bemutatott *Room* modellt, és annak *rooms* tábláját létrehozhassuk a következő parancsot kell futtatni a parancssorban:

```

rails generate model Room name:string description:text
capacity:integer num_of_this_integer
accommodation_id:integer

```

A fenti parancs sikeres futtatásakor a generátor létrehozza a korábban bemutatott migrációs fájlt. Ezen kívül a modellt bejegyzi az *app/models* mappában *Room.rb* névvel és a következő tartalommal:

```

class Room < ActiveRecord::Base
end

```

A változtatások lejegyzése után frissíteni kell az adatbázist, hogy azok életbe lépjenek. Ehhez újra a Rake eszközt kell használni és annak *db:migrate* parancsát, ahogy az alább látható:

```

rake db:migrate

```

A modellek létrehozása után a következő feladat a kapcsolatok kialakítása. A fenti *Room* modell létrehozásánál már sejthető volt, hogy az *accommodation\_id* mező azonosít egy kapcsolatot az *Accommodation* (szálláshely) modellel. Az Active Record szolgáltatás a táblák *id* mezője alapján azonosítja a kapcsolatokat és fontos, hogy a hivatkozó mező a *{tábla\_név}\_id* formátumot kövesse. Az Active Record hatféle kapcsolatot különböztet meg:

1. *belongs\_to*: egy-egy kapcsolat. A modell a hivatkozott modellhez tartozik.
2. *has\_one*: egy-egy kapcsolat. Egy *belongs\_to* kapcsolat másik oldala. A modellnek nincs idegen kulcsa a hivatkozó irányába.
3. *has\_many*: egy-sok kapcsolat. Egy *belongs\_to* kapcsolat másik oldala. A modellnek nincs idegen kulcsa a hivatkozó irányába.
4. *has\_many :through*: sok-sok kapcsolat. A kapcsolat kapcsolótáblán keresztül valósul meg. A hivatkozott modellekben nincs idegen kulcs, amely a kapcsolótáblára vagy a másik félre mutatna.
5. *has\_one :through*: egy-sok kapcsolat. Egy másik modellt használ kapcsolótáblaként.
6. *has\_and\_belongs\_to\_many*: sok-sok kapcsolat. Direkt kapcsolótáblás kapcsolat, ahol a kapcsolótábla nem jelenik meg modellként.

Az [1] mellékletben látható diagram alapján a *Room* modell kapcsolatait az alábbi módon kellett bejegyezni:

```
class Room < ActiveRecord::Base
  belongs_to :accommodation
  has_one :price
  has_many :bookings_rooms
  has_many :bookings, through: :bookings_rooms
  has_many :bookings_guests
  has_many :guests, through: :bookings_guests
  has_and_belongs_to_many :equipments
end
```

### 7.2. Autentikáció és autorizáció

A webalkalmazás jellege és az 5.1 fejezetben bemutatott szerepkörök megkövetelik valamilyen autentikációs és autorizációs modul használatát. Ahogy azt a 6.4.6 fejezetben bemutattam, a választás a Devise nevű megoldásra esett.

A Devise használatához a Rails alkalmazás függőségei közé fel kell venni a *devise* és *bcrypt* gem-eket. Ezt az alkalmazás gyökérkönyvtárában található Gemfile szerkesztésével lehet megtenni. A következő sorokat kell a fájlhoz fűzni:

```
gem 'devise'
gem 'bcrypt'
```

Miután a függőségek bejegyzésre kerültek a Bundle parancssori eszköz segítségével lehet telepíteni őket a környezeten, ha még nem lettek volna telepítve. A művelethez a következő parancsot kell futtatni:

```
bundle install
```

A függőségek telepítése után a Rails alkalmazást fel kell készíteni a Devise használatára. Ehhez a korábban már említett generátor eszközt kell használni. A Devise telepítéskor létrehozza a saját generátor parancsait, amivel inicializálhatjuk az integrációt. A következő parancsot kell futtatni a parancssorban:

```
rails generate devise:install
```

A generálás eredménye egy `devise.rb` nevű konfigurációs fájl, ami a `config/initializers` mappában található. Itt lehet a Devise szolgáltatás működését beállítani. A Devise az autentikációhoz adatbázisbeli modelleket használ. Kézenfekvő tehát, hogy egy modell generátor kiegészítésével készíthető el az az entitás akit hitelesíteni kell a rendszerben. A modell nevének a *User*-t választottam. A Devise-specifikus *User* modellt a következő paranccsal lehet létrehozni:

```
rails generate devise User
```

A generálás eredménye egy olyan *User* modell, amely több, az autentikációhoz elengedhetetlen (*email*, *encrypted\_password*), valamint egyéb, a Devise szolgáltatásaihoz szükséges mezőt (*last\_sign\_in\_ip*, *sign\_in\_count*, *reset\_password\_token*, stb.) tartalmaz. A felhasználók tehát az email címük és egy jelszó használatával léphetnek be az oldalra. A jelszavak hossza a korábban említett `devise.rb` konfigurációs fájlban állítható be, én legalább 6 karakterben határoztam meg.

A három felhasználói szerepkörnek nem kívántam három különböző Devise modellt létrehozni, hanem, ahogy azt a 6.3 fejezetben bemutattam, polimorfikus kapcsolattal reprezentálom őket. A bejelentkezést igénylő felhasználói szereplő minden példányához tartozik egy *User* entitás, amely az hitelesítéshez használatos.

A polimorfikus kapcsolathoz két új attribútumot kell a *User* modellhez rendelni, ezek a *role\_id* és *role\_type*. A két új mező létrehozását a *users* táblát létrehozó, korábban a modell generálás során létrehozott migrációs fájlba jegyeztem be. Az utolsó lépés a modell generálás érvényesítése az adatbázison.

Az modell generálás eredményeként a Devise többek között létrehozta a bejelentkező és regisztrációs oldalt és a kijelentkezési útvonalat. A generált felületek személyre szabásához futtatni kell az alábbi parancsot, amelynek eredményeként az alkalmazás *app/views* mappájában létrejön egy *devise* nevű mappa, amely az összes generált oldalt tartalmazza. Ezek az oldalak szabadon módosíthatók.

### **rails generate devise:views**

Mivel a *User* modell csak egy segédentitás az autentikáció implementációjához, ezért új, a felhasználói szerepkör szerint specifikus regisztrációs oldalakat kellett készítenem. Önállóan regisztrálni csak a szálláskereső és a szállásadó tud, adminisztrátort csak adminisztrátor tud regisztrálni. A regisztrációs űrlapok elemei részben a regisztrálandó felhasználó attribútumaiból, részben a *User* modell attribútumaiból tevődnek össze. Az űrlapok inicializálását és mentését is az egyes modellekhez tartozó vezérlőkkel (*controller*) kellett megoldani a Devise generálta vezérlők helyett. Az alábbiakban az *Owner* modellhez tartozó vezérlő regisztrációhoz köthető metódusai kerülnek bemutatásra.

```
# GET /owners/new
def new
  @owner = Owner.new
  @owner.build_user
end

# POST /owners
def create
  @owner = Owner.create(owner_params)
  if @owner.save
    redirect_to '/'
  else
    render :template => "owners/new"
  end
end
```

```
def owner_params
  params.require(:owner).permit(:name,
    user_attributes: [:email, :password,
      :password_confirmation])
end
```

Amikor egy szállásadó a regisztrációhoz vezető linkre kattint, akkor a webalkalmazás az `/owners/new` URL-re irányít egy HTTP GET kérést. Ezt a kérést az `OwnersController` vezérlőosztály `new` metódusa kezeli. A metódus feladata, hogy inicializálja azokat az objektumokat, amikhez a megjelenő űrlap kapcsolódik. A metódus az első sorban létrehoz egy új `Owner` entitást. A második sorban a `build_user` paranccsal úgy hoz létre egy új `User` entitást, hogy abban a külső kulcs a korábban létrehozott `@owner` objektumra mutat. A regisztrációs űrlap elküldésekor az `/owners` URL-re egy POST kérés indul. Ezt a kérést a `create` metódus dolgozza fel. A Rails környezetben minden kéréshez tartozó adatot a `params` objektum tárol, amely globálisan elérhető, a kontextustól függő objektum. A nem kívánt működést és gyanús behatolást megelőzendő a Rails környezet bevezetett egy paraméterkorlátozó eljárást. Ezt mutatja be az `owner_params` metódus. A metódus lényege, hogy a `params` objektumból, csak a `require` paranccsal meghatározott objektumhoz és csak a `permit` paranccsal meghatározott paramétereket engedi felhasználni. A `create` metódus az `owner_params` metódus segítségével kinyert paraméterek alapján létrehozza és elmenti az `Owner` entitást, miközben a beágyazott paramétereknek köszönhetően a kapcsolódó `User` entitás is létrejön az adatbázisban.

Az autorizációt két szinten lehet megvalósítani a Devise nyújtotta segéd metódusokkal. Az első szint az, hogy a vezérlő osztályok komplett metódusait megvédjük az illetéktelen kérésektől. Ehhez az alábbi kódot kell a vezérlő osztályhoz írni:

```
before_action :authenticate_user!, only: [:edit,
  :update, :destroy]
```

A kód lényege, hogy mielőtt az `only` objektumban bejegyzett metódusok lefutnának a felhasználót hitelesíteni kell. Ha már van bejelentkezett felhasználó,



akkor a futás folytatódik, különben a bejelentkező oldalra navigál a rendszer. A bejelentkezés követően a felhasználó a korábban elérni kívánt oldalra jut.

A második szintű autorizáció az, hogy a megjelenő oldalak tartalma és a rendszer viselkedése a felhasználói körök szerint módosul. Ehhez a szintén Devise nyújtotta `current_user` segédmetódus használható. A metódus segítségével elkérhető az aktuálisan bejelentkezett `User` entitás. Erre példa a következő kód:

```
def check_user
  unless current_user == @accommodation.owner.user ||
    current_user.role.is_a?(Admin)
    redirect_to root_url, alert: "Nincs jogosultsága!"
  end
end
```

### 7.3. Szobák szűrése

A szálláskereső és a látogató felhasználók megkövetelik, hogy szobalista hosszas böngészése helyett, kiszűrjék az igényeiknek megfelelő szobákat. Erre a szobák listája felett megjelenő szűrési panel nyújt lehetőséget.

A szűrési panel egy űrlap, amely a következő látható beviteli mezőket tartalmazza: város, érkezés dátuma, távozás dátuma, szoba kapacitása, szoba felszereltsége és szálloda szolgáltatásai. A szűrési panelt a 8.6 ábra mutatja be. A szűrés mechanizmusa a következő. A felhasználó kitölti szűrési feltételeket és elküldi az űrlapot. Az alkalmazás ezt érzékeli és a szűrési feltételeket a szobák listájának címéhez fűzve képez egy URL-t, majd átirányítja a felhasználót erre a címre. Az alkalmazás ekkor új kérést érzékel a szobák listájának megjelenítésére, azonban most vannak szűrési feltételek is a kérésben, ezért a megjelenítendő szobákra azokat alkalmazza. E mechanizmus előnye, hogy a szűrések során képzett URL-ek bármikor újra megtekinthetők és a felhasználók között megoszthatók. A továbbiakban a lépések részletes bemutatása következik.

A szűrési panelben megjelenő űrlap mezőinek könnyebb kezelése céljából létrehoztam a *Filter* modellt. A modellhez kizárólag az Active Record működése miatt tartozik adatbázistábla, de adat nem kerül mentésre a táblába. Az Active Record származtatás a felszereltség (*Equipment*) és szolgáltatás (*Serviece*) mezők

miatt szükségesek, amelyek egy-sok kapcsolatban állnak a *Filter* modellel. A modellhez tartozó osztály látható alább.

```
class Filter < ActiveRecord::Base
  has_many :equipments
  has_many :serviices

  def load_params(params)
    self.city = params[:city]
    self.start_date = params[:start_date]
    self.end_date = params[:end_date]
    self.capacity = params[:capacity] if
    params.has_key? :capacity

    if params.has_key? :equipment_ids
      params[:equipment_ids].split(',').each do |e|
        self.equipments.push(Equipment.find(e))
      end
    end

    if params.has_key? :serviice_ids
      params[:serviice_ids].split(',').each do |s|
        self.serviices.push(Serviice.find(s))
      end
    end

    self.guests = params[:guests] if params.has_key?
    :guests
    self.one_bed = params[:one_bed] == '1' ||
    !params.has_key?(:one_bed)
    self.two_bed = params[:two_bed] == '1' ||
    !params.has_key?(:two_bed)
    self.three_bed = params[:three_bed] == '1' ||
    !params.has_key?(:three_bed)
    self.four_or_more_bed = params[:four_or_more_bed]
    == '1' || !params.has_key?(:four_or_more_bed)
    self.cheap = params[:cheap] == '1' ||
    (params[:cheap] == '1' && params[:close] != '1')
    || !params.has_key?(:cheap)
    self.close = params[:close] == '1'
  end
end
```

A *Filter* modell abban is segítséget nyújt, hogy a weboldal megjelenítése előtt, az attribútumait az URL paraméterlistája alapján kitöltve az űrlap az aktuális

szűrés állapotát tartalmazza. A paraméterek betöltését a modell saját *load\_params* metódusa végzi el.

A szűrési panel űrlapját a *FilterController* vezérlő osztály *filter* metódusa dolgozza fel. A metódus feladata, hogy az *UrlHelper* segédosztály segítségével összeállítsa a szűrést azonosító URL-t, és átirányítsa rá a felhasználót. Az URL összeállításának sikertelensége esetén nem történik átirányítás és a hibáról üzenetben kap tájékoztatást a felhasználó. Az *UrlHelper* segédosztály *build\_parameterised\_url* metódusa két lépésben végzi el az URL képzését. Először a paraméterként kapott *params* objektumból eltávolítja azokat a bejegyzéseket, amelyek az űrlap elküldésekor bekerültek, de nem tartozik hozzájuk érték. Ezt a műveletet a *remove\_empty\_params* metódus végzi el. Fontos, hogy a képzett URL-ben ne legyenek üres paraméterek, mert a szűrés erősen támaszkodik a paraméterek jelenlétére. A második lépésben a *write\_params* metódus hozzáfűzi az űrlap rejtett, *base\_url* mezőjének értékéhez a megmaradt paramétereket. A *base\_url* mező azt a címet tartalmazza, ahova az átirányítás mutatni fog. Az *UrlHelper* segédosztály és metódusai a [2] mellékletben tekinthetők meg.

Szobaszűrés esetén a képzett URL szobalistára fog mutatni és tartalmazza a szűrési feltételeket, mint paramétereket. Egy ilyen URL-t mutat be az alábbi példa.

**`/rooms?filter=fine&city=Budapest&start_date=2015.04.19&end_date=2015.04.20&capacity=2&equipment_ids=1,3&service_ids=3,5`**

Ez erre a címre érkező GET, tehát oldalletöltés kezdeményező kérést a *RoomsController index* metódusa kezeli. Az *index* metódus feladata, hogy összeállítsa a szobák halmazát, amiket a felhasználó számára meg kell jeleníteni. Azt, hogy a szobákon kell-e a szűrést alkalmazni, a *filter* paraméter jelenlétének ellenőrzésével dönti el. Magát a szűrést a *FilterHelper* segédosztály *filter\_rooms* metódusa végzi el. A *filter\_rooms* metódus megkapja a teljes paraméterlistát és az egyes paraméterek, mint feltételek szerint lekéri az adatbázisból a feltételt kielégítő szobákat. Minden paraméterhez tartozik ekkor egy szobákat tartalmazó tömb. A következő lépésben a tömböket összefűzi a program, kiszűri az ismétlődő szobákat és feljegyzi az ismétlődések számát. A szűrési feltételek és az ismétlődések számából meghatározható, hogy mely szobák elégítik ki az összes szűrési feltételt,

és kerülnek végül megjelenítésre. Az a szoba, amely ismétlődéseinek száma megegyezik a szűrési feltételek számával, a szűrési eredmény része lesz. A *FilterHelper* segédosztály szűrést elvégző metódusai a [3] mellékletben tekinthetők meg.

Az egyes paraméterek szerinti szűréseket adatbázis lekérések segítségével végzi el a program. A város feltétel ellenőrzéséhez a program összevonja a *rooms*, *accommodations* és *address* táblákat egy kétszintű JOIN utasításban. A felszereltség és szolgáltatás feltételek ellenőrzéséhez az Active Record egy hasznos tulajdonságát használtam ki. A *where* parancs feltételei közé lehetőség van tömb megadására, és ekkor az utasítás az egy-sok vagy sok-sok kapcsolat összes, a tömbben megadott azonosítóval egyező találatát szolgáltatja. A parancs eredményének és a feltételek számának összehasonlításával ellenőrizhető, hogy egy adott szoba rendelkezik-e minden felszereltséggel, vagy egy szálláshely rendelkezik-e minden szolgáltatással. A dátum szerinti szűrést a *BookingsHelper* *is\_bookable* metódusa végzi. A metódus ellenőrzi, hogy az adott szoba szerepel-e olyan foglalásban, amelynek ideje a kiválasztott időszakkal egybeesik és nem valamelyik felhasználó dedikált kosara. Ha az egybeesések száma eléri a szobák mennyiségének számát, akkor az adott időszakra nincs szabad szoba a kiválasztott típusból. Az *is\_bookable*, és az általa használt *overlaps* metódusokat mutatja be az alábbi kódrészlet.

```
def self.is_bookable(room, start_date, end_date)
  if start_date.nil? && end_date.nil?
    return true
  end

  bookings = Booking.joins(:rooms).where.not('state'
=> 'CART').where('rooms.id' => room.id)

  overlapping = 0
  bookings.each do |b|
    if overlaps(start_date, end_date, b)
      overlapping += 1
    end
  end

  return true if overlapping < room.num_of_this
  return false
end
```

```

end
def self.overlaps(start_date, end_date, base)
  if start_date == base.end_date
    return false
  elsif end_date == base.start_date
    return false
  else
    return (start_date - base.end_date) *
      (base.start_date - end_date) >= 0
  end
end
end

```

Az *overlaps* metódus két időszak metszetét vizsgálja. Megengedett átfedés az, ha a két időpont kezdő és vég, illetve vég és kezdő dátuma megegyezik, hiszen egy szálláshely szobáját délelőtt kell elhagyni, és délután már új vendéget fogadhat.

#### 7.4. Intelligens keresés

Az intelligens keresés funkcióhoz a 6.2 fejezetben bemutatott optimalizálási modelleket kellett az AMPL modellező eszköz számára feldolgozható formátummá alakítani. Az optimalizálási modelleket az AMPL csomaghoz tartozó modellező nyelvvel lehet deklarálni és hozzájuk adatot csatolni. Az AMPL képes arra, hogy az optimalizálási modelleket és az adatot szétválassza, így azok megadhatók két külön bemenetként.

Az AMPL számára értelmezhető optimalizálási modellek elkészíthetők egy egyszerű szövegszerkesztő segítségével. Az elkészült fájlok kiterjesztése ajánlás szerint *.mod*. A 6.2.3 fejezetben bemutatott, az ár és távolság szerint egyaránt optimalizáló modell az alábbi formában került leírásra:

```

param min_distance = 1;
param min_price = 1;

param guests >= 0;

set ROOMS;

param capacity {ROOMS} >= 0;
param stars {ROOMS} >= 0;
param price {ROOMS} >= 0;
param distance {ROOMS,ROOMS} >= 0;

```

```

var Occupation {ROOMS} binary;
var occupied_rooms = sum {i in ROOMS} Occupation[i];

subject to Accommodation:
sum {i in ROOMS} Occupation[i]*capacity[i] = guests

minimize OPTIMUM:
sqrt(( sum{i in ROOMS} Occupation[i] * (price[i] -
min_price)^2) / occupied_rooms + 1) / min_price +
if occupied_rooms > 1 then (
sqrt(( sum{(i,j) in {ROOMS,ROOMS}} (Occupation[i] *
Occupation[j]) * ((distance[i,j] - min_distance)^2)) /
occupied_rooms + 1) / min_dist
) +
sqrt(( sum{i in ROOMS} Occupation[i] * ((stars[i] -
10)^2)) / occupied_rooms + 1) / 10;

```

A modell első felében a paraméterek és változók vannak rögzítve, a második felében a korlátozások és a célfüggvény. A *min\_distance* és *min\_price* paraméterek nem szükséges, hogy változtathatók legyenek, mert a 6.2 fejezetben bemutatott konvertálás után a legkisebb érték mindig 1, emiatt rögzített az értékük. A *guests* paraméter a vendégek számát tárolja. Ezt követi a szobák halmazának deklarálása *ROOMS* néven. A szobák halmazához tartozik a *capacity*, mint kapacitás, a *price* mint ár és a *stars*, mint átlagos értékelés paraméter. A szobák közti távolságot a *distance* paraméter tárolja, amely a szobák halmazából képzett két dimenziós mátrixhoz tartozik. A modellben szereplő bináris változó *Occupation* névre hallgat és a *ROOMS* halmaz felett van értelmezve. Az *occupied\_rooms* egy segédváltozó, a célfüggvény átláthatóbbá tétele, és a redundancia elkerülése miatt került bevezetésre, értéke a megoldásba beválasztott szobák száma.

A modellnek egy korlátozása van, amely a kiválasztott szobák kapacitását teszi egyenlővé a vendégek számával. Ezt a korlátozást fogalmazza meg az *Accommodation* nevű kifejezés. A modell utolsó eleme az *OPTIMUM* elnevezésű célfüggvény. A 6.2.3 fejezetben bemutatott célfüggvényhez képest az implementáció alkalmaz egy kiegészítést. A nemlineáris megoldó az optimalizálás során választhatja azt, hogy az összes változó 0. Ekkor a gyökvonás operátor hibát jelez, és a folyamat leáll. Ezt elkerülendő, a gyökjel alatti kifejezéshez hozzáadok

egyét, ami az eredményt nem befolyásolja, de segítségével a megoldó minden esetben helyesen fut.

Az optimalizálási modell megszerkesztése után össze kell állítani azt az adathalmazt, amelyen az optimalizálást el kell végezni. Amint azt korábban említettem, az adatokat a modelltől függetlenül, egy másik fájlban is meg lehet adni. Egy ilyen adatfájl kiterjesztése ajánlás szerint *.dat*. Egy, a modellhez illeszthető adatfájl tartalma a következőképpen néz ki:

```
param guests := 9;

set ROOMS:= R1 R2 R3 R4 R5 R6;

param: capacity stars price :=
  R10  2      8      1.34
  R11  3      7.5    1.0
  R20  2      8.2    2.58
  R30  2      8.2    5.16
  R40  4      9      1.25
  R50  2      8.1    3.58;

param distance:
      R10 R12 R20 R30 R40 R50 :=
  R10  1.0  1.0  5.0  4.0  2.0  9.0
  R11  1.0  1.0  5.0  4.0  2.0  9.0
  R20  5.0  5.0  1.0  6.0  4.0  7.0
  R30  4.0  4.0  6.0  1.0  1.0  7.0
  R40  2.0  2.0  4.0  1.0  1.0  6.0
  R50  9.0  9.0  7.0  7.0  6.0  1.0;
```

Az adatfájlban a modellben meghatározott paraméterekhez rendelünk értékeket, tehát ugyanazokat az elnevezéseket kell alkalmazni. Fontos továbbá, hogy a mátrixok esetében az első index a sornak, a második index az oszlopnak felel meg, illetve egydimenziós halmaz esetén a sor a halmaz eleme és az oszlopok a hozzá tartozó paraméterek.

Ahhoz, hogy az AMPL-lel a fent definiált modell szerint a meghatározott adathalmazt optimalizálni lehessen, szükség további konfigurációra és parancsokra, amelyeket a modell fájl végére kell felvinni. Ezek a beállítások a megoldás módjára, az adatok helyére, az optimalizálás elvégzésére és az eredmény kijelzésére vonatkoznak.

```
option solver bonmin;
data vopt.dat;
solve;
option display_1col 1000000;
display Occupation;
```

A konfiguráció első sora beállítja, hogy mely megoldóval történjen az optimalizáció. Jelen esetben ez a Bonmin. A második sor az adat fájlt csatolja a feladathoz, amely a példában *vopt.dat* névre hallgat. A negyedik sor *solve* parancsának hatására az AMPL levezényli az optimalizálást, annak folyamatát és eredményét a standard kimeneten jelzi. Az utolsó két sor a végső eredmény kijelzését állítja be. Először az kerül meghatározásra, hogy az eredményhalmazt egy oszlopban jelezze maximum egymillió sorig. Végül a *display* parancs paramétereként az AMPL utasítást kap arra, hogy az *Occupation* nevű változóhalmazt írja a képernyőre.

Az optimalizálást a parancssorból lehet elindítani, és az eredményt is ott lehet megtekinteni. A művelet elvégzéséhez a következő parancsot kell kiadni:

```
ampl vopt.mod
```

A művelet teljes kimenetét hely hiányában megcsonkítottam. A végső eredményt a teljes kimenet végén, az alább látható módon jelzi ki a megoldó. Értelemszerűen az egyesek az elfoglalt szobákat, míg a nullák az optimális eredményből kihagyott szobákat jelzik.

```
Occupation [*] :=
R10 1
R11 1
R20 0
R30 0
R40 1
R50 0
;
```

Mivel az AMPL egy önálló, parancssori program és nincs interfésze egy Ruby program számára, ezért olyan megoldást kellett találnom, amivel a webalkalmazásból közvetlenül tudok parancssori műveleteket végrehajtani és az



eredményeket kiolvasni. A Ruby nyelvben, és így egy Rails alkalmazásban is több lehetőség adódik parancssori műveletek futtatására. Ezek közül kettőt alkalmaztam:

1. `system("[command]")`: a parancsot egy új környezetben hajtja végre, nem szakítja meg a hívó folyamatot. Visszatérési értéke *true*, ha a futás sikeres, *false* különben.
2. ``[command]``: a parancsot szintén egy új környezetben hajtja végre. Visszatérési értéke a művelet képernyőkimenete.

A Ruby programnak az optimalizáló futtatása előtt össze kell gyűjtenie az adatokat és fájlba írnia. Az optimalizálás közvetlen előkészületeit az *OptDataHelper* nevű segédosztály metódusai végzik. Az eddigi példákat követve, az olcsó és közeli optimalizálás elvégzéséhez a következő statikus metódust kell meghívni:

```
def self.find_cheap_and_close_solution(rooms,
  distances, guests)
  problem = generate_problem_name

  File.open("smartfilter/tasks/#{problem}.dat", 'w')
  do |data|
    write_rooms_set(rooms, data)
    write_base_params(rooms, guests, data)
    write_capacity_and_stars_and_price_params(rooms,
      data)
    write_distance_params(rooms, distances, data)
  end

  run_solver_on(problem, PROPERTIES['smartfilter -
    models']['cheap_and_close'], rooms)
end
```

A kódból látható, hogy az alkalmazás gyökérkönyvtárában létrehoztam egy *smartfilter* nevű mappát, amin belül a *tasks* mappa tárolja az optimalizálási feladatok fájljait. A *tasks* mappa mellett található még itt egy *models* nevű mappa, ami biztonsági mentést tárol az optimalizálási modellekről.

A *generate\_problem\_name* metódus egy egyedi azonosítót képez, amivel a feladat azonosítható. Minden, a feladathoz tartozó fájl neve ez lesz, különbség csak

a kiterjesztésükben lesz. Az azonosító tartalmazza a végrehajtás dátumát és két 1 és 100 közötti véletlen számot. Az azonosító képzését mutatja be az alábbi kódrészlet.

```
def self.generate_problem_name
  return "#{DateTime.now}_VSF#{rand(100)}#{rand(100)}"
end
```

A *find\_cheap\_and\_close\_solution* metódus fő feladata, hogy a korábban ismertetett formátumban fájlba írja az optimalizáláshoz szükséges adatokat. A különböző adatok paraméterekhez rendelését a *write\_rooms\_set*, a *write\_capacity\_and\_stars\_and\_price\_params*, a *write\_base\_params* és a *write\_distance\_params* metódusok végzik. Az adatokat a metódusparaméterként kapott *rooms* tömb tartalmazza. A tömbben kulcs-érték párok vannak, ahol a kulcsok egy szoba példányt azonosítanak. A kulcsok a szobatípus adatbázisbeli *id* azonosítója és a *num\_of\_this* attribútum szerinti inkrementált számból áll. Ezek a kulcsok láthatók a korábban bemutatott optimalizálási kimeneten. A *rooms* halmaz elemei nem minden esetben tartalmazzák az adatbázisban tárolt összes szobát. Ha a szálláskereső kiválaszt egyéb, az elérhetőség, a szobatípus, a felszereltség vagy a szolgáltatásokra vonatkozó feltételeket, akkor a 7.3 fejezetben bemutatott szűrési mechanizmus segítségével a program előszűri a szobákat.

Az adatfájl elkészítése után a metódus elindítja az optimalizálást a *run\_solver\_on* metódushívással, amely alább kerül bemutatásra. A metódus első feladata, hogy elkészítse a futtatási konfigurációt. Ehhez a *write\_solver\_script* metódust használja, amely a korábban bemutatott futtatási beállítások mellett az adatbázisból elkért optimalizálási modellt is a fájlhoz fűzi. Az optimalizálási modellt a metódus második paramétereként átadott, egy konfigurációs fájlból kiolvasott érték azonosítja. Ez az érték egy *Property* entitás *key* mezőjét tartalmazza és segítségével olvasható ki az adatbázisból a modell leírás. A modelleket a könnyebb módosíthatóság miatt tárolom adatbázisban.

A futtatási konfiguráció elkészülte után a program *command* néven összeállítja azt a kifejezést, amelyet a parancssorban futtatva elindítható az optimalizálás. Az utasítás nem csak az optimalizálást indítja el, hanem a művelet

minden kimenetét egy *.solve* kiterjesztésű fájlba irányítja. Ebből a *.solve* kiterjesztésű fájlból kell kiolvasni az eredményt.

```
def self.run_solver_on(problem, model, rooms)
  write_solver_script(problem, model)
  command = "ampl smartfilter/tasks/#{problem}.solve >
smartfilter/tasks/#{problem}.solution"
  has_run = system(command)

  if has_run
    lines = `cat smartfilter/tasks/#{problem}.solution
| wc -l`
    if lines.to_i <= rooms.size+3
      puts 'NOT FEASIBLE'
      return Array.new
    else
      system("cat
smartfilter/tasks/#{problem}.solution | tail -
n#{rooms.size+4} | head -n#{rooms.size+2} >
smartfilter/tasks/#{problem}.raw_solution")
      OptSolutionHelper.read_solution_for(problem,
rooms)
    end
  end
end
```

A *command* parancsot a *system* metódussal futtatva, a *has\_run* változó felveszi a futás sikerességének eredményét. Ezt ellenőrzi a következő elágazás. Mivel a nemlineáris megoldó akkor is képez kimenetet, ha nincs optimális megoldás, ezért ellenőrizni kell a fájlt annak tartalma szerint. Az első ellenőrzés, hogy a fájl hossza megegyezik-e az optimalizálásba bevont szobák száma plusz 3 sorral, ha ugyanis a megoldó nem képes a feladatot megoldani, akkor azt egy ennél rövidebb üzenetben közli a felhasználóval. Az eredményfájl sorainak számát megadó parancsot a visszaaposztróf operátorok közé írva futáskor a *lines* változó megkapja a parancs eredményét. Ha a sorok száma megfelelő, akkor az optimalizálás eredményét tartalmazó sorokat a fájl végéről levágja a program és egy új, *.raw\_solution* kiterjesztésű fájlba menti. A művelet során nem csak a szobák foglaltságát jelző *Occupation* változók értéke kerül levágásra, hanem az előtte lévő két sor is, ahol ugyanis további információ olvasható ki az eredményről. Miután az

eredmény elsődleges feldolgozása megtörtént, a folyamatot az *OptSolutionHelper* segédosztály veszi át a *read\_solution\_for* metódushívás útján.

Az *OptSolutionHelper* segédosztály feladata, hogy az optimalizálás eredményéből összeállítsa a szobák halmazát. A műveletet a *read\_solution\_for* metódus végzi, ezt mutatja be az alább látható kódrészlet. A metódus második paramétere azon szobák halmaza, amelyekből az optimalizálási adathalmazt korábban a program összeállította.

A metódus megnyitja a korábban létrehozott *.raw\_solution* kiterjesztésű fájlt és sorról sorra beolvassa a tartalmát. A fájl első sorában az optimalizálás eredményessége olvasható ki. Ha a program eljut idáig, akkor az optimalizálás sikeres volt. Két lehetőséget kell megvizsgálni. Az első, hogy az optimalizálás során lett-e kizárva szoba. Ha ugyanis minden szoba az optimális megoldás része, akkor nincs további teendő, a metódus paramétereként kapott szobák halmazát minden további nélkül továbbítani lehet. Ellenkező esetben a kizárt szobákat el kell távolítani a szobák halmazából.

```
def self.read_solution_for(problem, rooms)
  File.open("smartfilter/tasks/#{problem}.raw_solution", 'r') do |f|
    l_idx = 1
    f.each_line do |line|
      if l_idx == 1
        return compact_solution(rooms) if
          feasibility(line) == 'Full'
        return Array.new unless feasibility(line) ==
          'Optimal'
      elsif l_idx > 2
        unless is_part_of_solution?(line)
          room_key = extract_room_key_from(line)
          rooms.delete(room_key)
        end
      end
      l_idx += 1
    end
  end

  return compact_solution(rooms)
end
```

A megoldást tartalmazó fájl harmadik sorától kezdődik az *Occupation* változók felsorolása. A felsorolásban a szobákhoz rendelt azonosító és egy 0 vagy 1 érték szerepel. Azt, hogy az adott sorban szereplő szoba a megoldás rész-e, a *part\_of\_solution?* metódus ellenőrzi. A sorokat regexp (Regular Expression) kifejezésekkel, mintaillesztés útján ellenőrzi a program. Ha egy szoba nem része a megoldásnak, akkor az *extract\_room\_key\_from* metódus szintén regexp kifejezés használatával kiemeli az azonosítót a sorból, amelynek segítségével a *rooms* halmazból eltávolítható a szoba. A két, regexp kifejezéseket használó metódust mutatja be az alábbi kódrészlet.

```
def self.is_part_of_solution?(line)
  activity =
    /\w\d+\d+\s+(?<activity>\d)/.match(line)[:activity]
  return activity.to_i == 1
end

def self.extract_room_key_from(line)
  key = /( ?<key>\w\d+\d+)\s+\d/.match(line)[:key]
  return key
end
```

Visszatérésként a *rooms* halmaz átalakításra kerül a *compact\_solution* nevű metódus által. Az átalakítás során a program eltávolítja a kulcsokat és a szobákat egy listába rendezi.

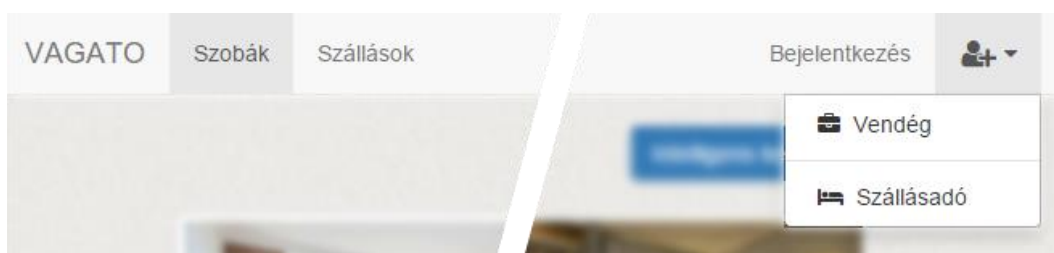
## 8. Felületek és használat

Ebben a fejezetben az elkészült webalkalmazás felületeit és a funkciók használatát mutatom be. Az ábrákról a jobb láthatóság miatt kivágtam az üres részeket, amit ferde, fehér sávval jeleztem, továbbá a nem releváns részeket elhomályosítottam.

### 8.1. Menüsáv

A webalkalmazás fő eleme a minden oldal tetején megjelenő navigációs menüsáv. A menüsávban megjelenik a portál neve, valamint a felhasználó számára fontos menüpontok és akciók. Az 5.1 fejezetben bemutatott felhasználói szerepkörök számára más-más elemek jelennek meg a menüsávon. Ezen kívül a szállásadó és az adminisztrátor felhasználó bejelentkezésekor a menüsáv színe is megváltozik, jelezve, hogy az érzékeny adatokat kezelő fiók aktív. Azt, hogy a felhasználó mely oldalon tartózkodik, a megfelelő menüelem sötétebb háttérszíne jelzi.

A nem bejelentkezett, látogató felhasználó számára az alább látható menüsáv jelenik meg.

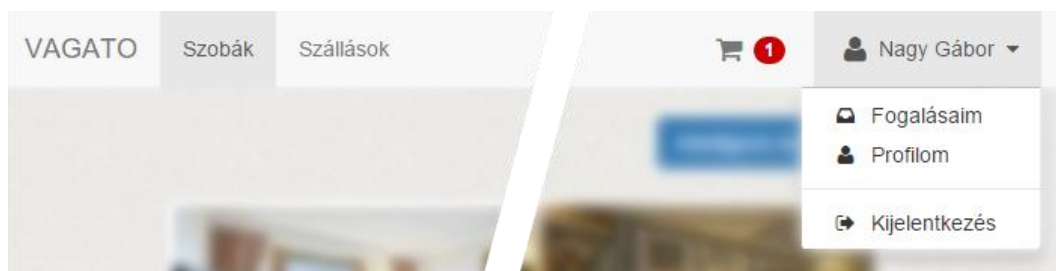


8.1 ábra Látogató számára látható menüsáv

A menüsáv elemeit balról jobbra irányuló bejárással ismertetem. Az első elem a webalkalmazás fantázianeve, amelyre kattintva a főoldalra lehet jutni. Ezt követik a szobák listáját és a szálláshelyek listáját megjelenítő menüpontok. A menüsáv jobb oldalán a bejelentkező oldalra mutató link található. Az utolsó elem egy legördülő lista, ahonnan a szálláskereső, vagyis vendég, illetve a szállásadó regisztrációs oldalára lehet jutni.

A bejelentkezett szálláskereső számára a menüsáv az előbb ismertetett megjelenést követi, ahogy az az alábbi ábrán is látható. A menüsáv bal oldala megegyezik a látogatóéval, változás a jobb oldalon vehető észre. Itt az első elem a

vendég kosarára mutató link, ahova a kiválasztott szobákat gyűjti. A kosár ikon mellett piros háttérrel annak tartalmát jelző címke jelenik meg.



**8.2 ábra Szálláskereső számára látható menüsáv**

A bejelentkezett állapotot jelzi, hogy megjelenik a felhasználó neve, ami egyben egy legördülő lista is. A legördülő listában a szálláskereső foglalásait listázó oldalra, valamint a felhasználói fiók beállításait tartalmazó oldalra lehet navigálni. Szintén a legördülő listában található a kijelentkezés akció.

A szállásadó számára mind megjelenésében, mind funkcióiban más menüsáv kerül megjelenítésre. Fontosnak tartottam, hogy a szállásadó számára a portál megjelenése különbözzön, ugyanakkor ne különüljön el lényegesen a szálláskereső vagy látogató megjelenésétől, ezért választottam a menüsáv háttérszínének megváltoztatását. Az alábbi ábra mutatja a be a szálláskereső menüsávját.



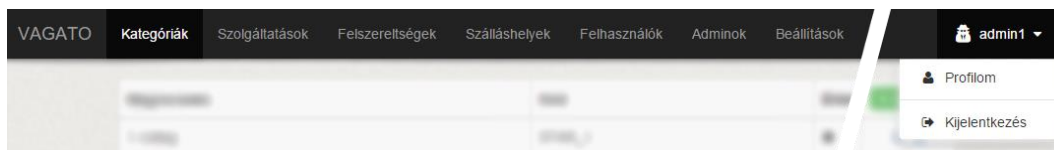
**8.3 ábra Szállásadó számára látható menüsáv**

A szállásadó menüsávjának első fontos eleme a saját szállásokra mutató *Szállások* menüpont. Ezt követi a foglalások kezelési oldalára navigáló bejegyzés. A menüsáv jobb oldalán a szálláskeresőnél látott módon a szállásadó neve jelenik meg legördülő menüként, ahol a profilszerkesztő oldalra mutató link és a kijelentkezés található.



## 8. Felületek és használat

Az adminisztrátori szerepkörbe tartozó felhasználók, a szállásadó esetében említett megfontolásból szintén más megjelenésű menüsávot láthatnak.

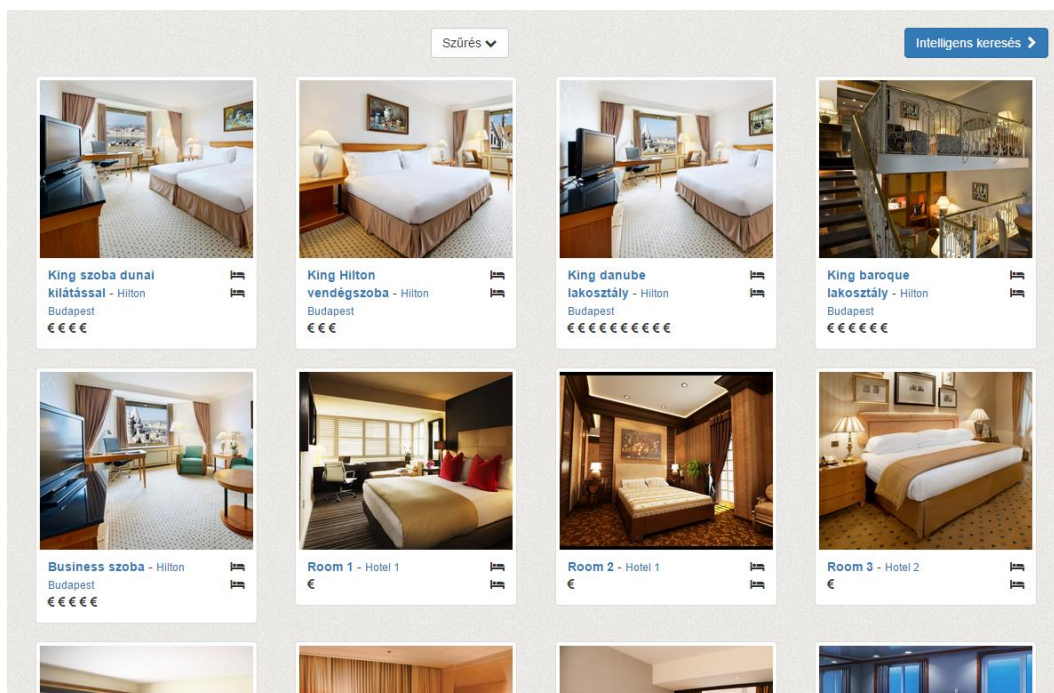


8.4 ábra Adminisztrátor számára látható menüsáv

Az adminisztrátor hét fő menüpontot lát a menüsávon. Ezek mindegyike a rendszerben található törzs- és felhasználói adatok szerkesztésére alkalmas oldalakra mutatnak. A menüsáv jobb oldalán a szállásadóhoz hasonló módon jelenik meg az adminisztrátor neve és a legördülő menü, benne a profilra mutató link és a kijelentkezés.

### 8.2. Szobák

A látogató és szálláskereső számára a portál a főoldala a szobák listája. A listában a rendszerben szereplő minden szoba megjelenik, de lehetőség van a megjelenő szobák szűrésére. A szobák kártyákként kerültek megjelenítésre, ahol több kapcsolódó információ is feltűnik, ahogy az alábbi ábrán is látható.



8.5 ábra Szobák listája



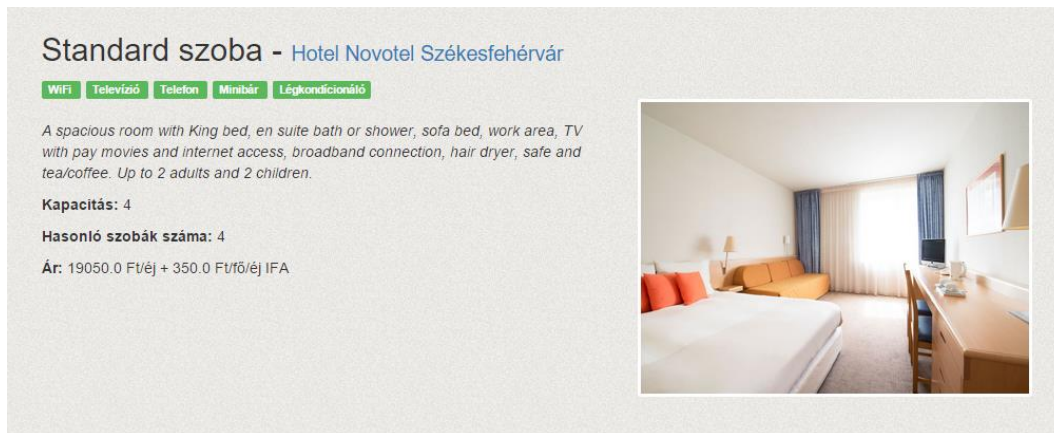
A felületen minden szobához tartozó kártyán megjelenik a feltöltött előnézeti kép, a szobatípus neve, a szálláshely neve, ahol a szoba található, az árkategória és az ágyak száma. A lista tetején két gomb található. Középen a *Szűrés* feliratú gomb megnyomására megjelenik a szűrési feltételeket tartalmazó panel, amelyet az alábbi ábra mutat be. Az *Intelligens keresés* feliratú gomb az intelligens keresés funkció oldalára navigál.

**8.6 ábra Szobák szűrési feltételeit tartalmazó panel**

A szűrési feltételeket tartalmazó panel három részből áll. A felső rész a helyiség, az érkezési és távozási dátum, valamint a kívánt típusú szoba ágyainak száma adható meg. Bal alsó rész a szoba felszereltségének kiválasztására használatos, míg a jobb alsó részben a szálláshely nyújtotta szolgáltatások szűrhetők. A szűrés műveletét az alul, középen található zöld *Szűrés* feliratú gomb megnyomásával lehet elvégezni. A fent, középen továbbra is megjelenő fehér *Szűrés* feliratú gomb hatására a panel eltüntethető.

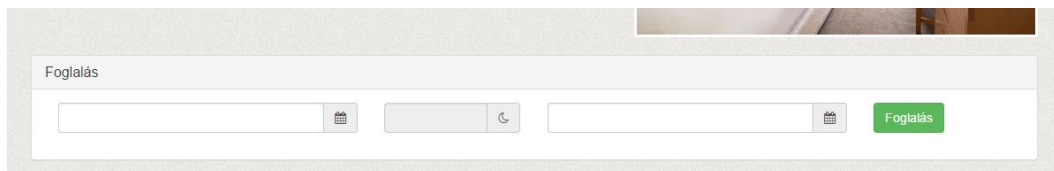
A szobákat bemutató kártyákon a szoba nevére, vagy a megjelenő képre kattintva a felhasználó a szobát részletesen bemutató oldalra jut. Egy ilyen oldalt mutat be a 8.7 ábra. A megjelenő felületen a szoba neve mellett linkként olvasható a szálláshely neve, ahol a szoba található. A linkre kattintva a szálláshelyet részletező oldalra jut a felhasználó. A szoba neve alatt zöld háttérű címkék sorolják fel a szoba felszereltségét. Ezt követi a szoba egy rövid bemutatkozó leírása, majd az egyéb adatok. A *Kapacitás* a szobában elszállásolható vendégek, vagyis az ágyak számát jelenti. A többszemélyes ágy több ágynak minősül. A *Hasonló szobák*

*száma* a szálláshely további hasonló típusú szobáinak számát jelenti. Az utolsó megjelenő adat az ár. Az oldal jobb oldalán a szoba bemutató képe jelenik meg.

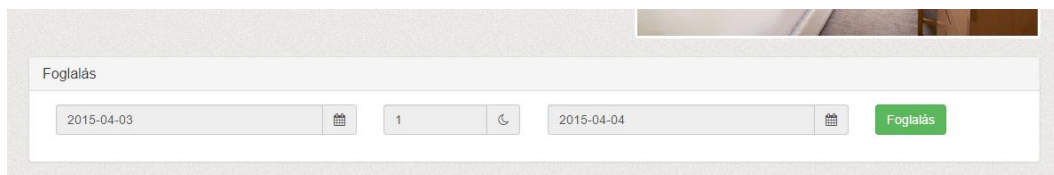


8.7 ábra Egy szoba részletes bemutató oldala

A bejelentkezett szálláskereső számára a szobák adatai és a kép alatt található egy szobafoglalási panel, amit a 8.8 ábra mutat be. Itt két dátumválasztó segítségével a szálláskereső kiválasztja az érkezési és távozási dátumot, majd *Foglalás* gombra kattintva a kosarába helyezi a szobafoglalást. Amennyiben a foglalás nem lehetséges, mert a szoba típusból a kiválasztott időszakban már mind foglalt, akkor a rendszer figyelmezteti a szálláskeresőt.



8.8 ábra A szálláskereső számára megjelenő szobafoglalási panel



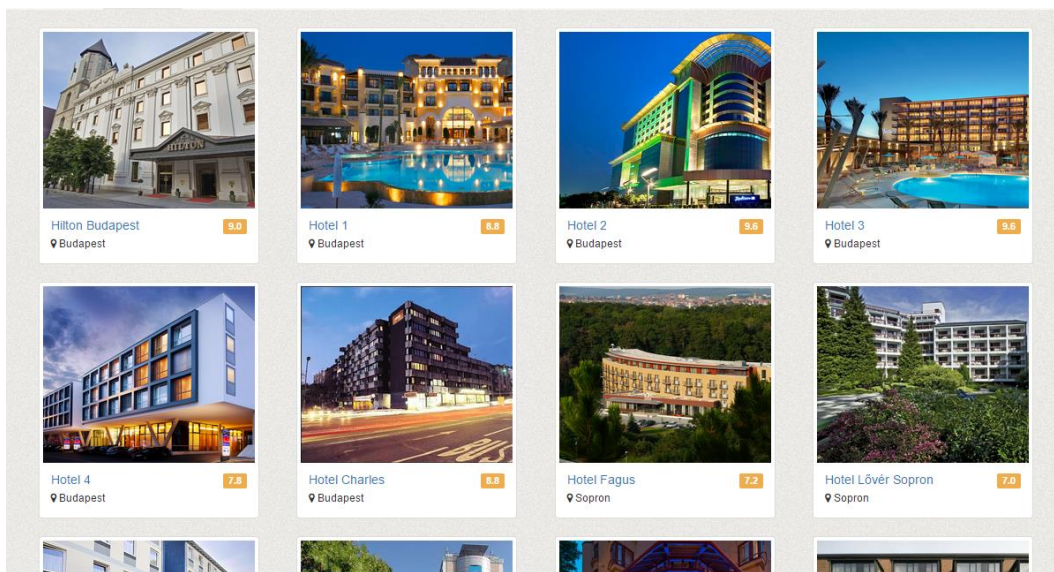
8.9 ábra Rögzített szobafoglalási panel

Amennyiben a szálláskeresőnek már van a kosarában szobafoglalás, akkor a dátumok már nem módosíthatók. A dátumválasztók ekkor azt a dátumot mutatják, amely a korábbi foglaláskor kiválasztásra került és letiltott állapotban vannak. A

*Foglalás* gombbal a szoba jelzett időszakban foglalható. Egy ilyen, kitöltött panelt mutat be a 8.9 ábra.

### 8.3. Szálláshelyek

A látogatók és a szálláskeresők a szobákon kívül a rendszerben található szálláshelyeket is böngészhetik. A szálláshelyek a szobákhoz hasonlóan kártyás megjelenítést kaptak és hasonló listába vannak szervezve. A szálláshelyek listáját mutatja be az alábbi ábra.



8.10 ábra Szálláshelyek listája

Az egyes kártyákon látható a szálláshelyről készült bemutató kép és a szálláshely neve. A kép és a név egyben egy link, amely a szálláshelyet részletesen bemutató oldalra mutat. A szálláshely neve alatt a város olvasható, a jobbra igazítva pedig egy sárga háttérű címkén jelenik meg az átlagos értékelés.

A szálláshelyet részletesen bemutató oldal, az egységesség jegyében szintén követi a szobát részletező oldal megjelenését. Tartalma három fő szekcióra osztható. Az első harmadban a szálláshely neve olvasható, mellette a kategóriájának megfelelő számú csillag követ. A név alatt sárga címke jelzi a szálláshely átlagos értékelését, míg a zöld címkék a szolgáltatásokat jelentik. Ezután a szálláshelyet röviden bemutató szöveg következik. A bemutató szöveg alatt a pontos cím és autós

navigációhoz a GPS koordináták jelennek meg. A könnyebb érthetőség kedvéért a szálláshely címe térképen is megjelenítésre került.

### Pannónia Hotel Sopron ★★★★★

7
Halálár
Masszázs
Parkoló

Lorem ipsum dolor sit amet, consectetur adipiscing elit. In nec tincidunt justo. Suspendisse potenti. Nunc vehicula cursus sagittis. Etiam mollis justo leo, vitae consequat diam sollicitudin sed. Etiam tincidunt, nibh vestibulum faucibus condimentum, odio massa dictum turpis, a cursus enim nibh quis ante. Nulla facilisi. Ut viverra est vitae neque laoreet, finibus aliquam nisi ultricies. Etiam tempor, nunc eget viverra aliquam, felis nibh finibus sapien, vitae aliquet ante diam eget purus. Nam eleifend elementum massa in luctus. Nullam varius sapien ipsum. Sed ut nunc libero.

**Cím:** 9400 Sopron, Várkerület 75., HU (47.6837231, 16.592449)

**Szobák:**

**Léghűtött Superior kétágyas -**  
Pannónia Hotel Sopron  
€ €

**Standard egyágyas -**  
Pannónia Hotel Sopron  
€

**Standard kétágyas -**  
Pannónia Hotel Sopron  
€

**Standard négyágyas -**  
Pannónia Hotel Sopron  
€ € € €

**Vélemények:**

Nagy Gábor véleménye ekkor: 2015.04.17

Szép épület, kedves kiszolgálás. Nehéz a parkolás és viszonylag drága.

★★★★★☆☆☆☆

Kis Imre véleménye ekkor: 2015.04.17

Mindennel meg voltunk elégedve, mindenkinek csak ajánlani tudom!

★★★★★☆☆☆☆

Kertész Vilmos véleménye ekkor: 2015.04.17

Akadozott a fűtés és az ételek sem ízlettek.

★★★★★☆☆☆☆

11. ábra Egy szálláshely részletes bemutató oldal

A második szekcióban a szálláshely szobatípusai tekinthetők meg. A felsorolásban a szobák kártyás megjelenése megegyezik a szobák listáján látottakéval. Az oldal utolsó harmadában a szálláshelyről megfogalmazott vélemények kerültek felsorolásra. Egy vélemény egy kártyán jelenik meg, amelyen szerepel a véleményt tevő vendég neve, a véleményezés ideje, a szöveges értékelés és a megítélt pontszám csillagok formájában.



### 8.4. Intelligens keresés

Az intelligens keresés funkció a szobák listája felett található Intelligens keresés feliratú gomb megnyomásával érhető el. Ekkor a felhasználó egy új oldalra kerül, ahol a szűrési panelhez hasonló űrlap fogadja. Az űrlapon ki kell tölteni a keresés szempontjait, úgy, mint város, érkezés dátuma, távozás dátuma, összes utazó vendég száma, felszereltség, szolgáltatás, szobatípus ágyak szerint és a keresés szempontja. Az intelligens keresés funkció felületét találatokkal együtt a 0 melléklet mutatja be.

A keresés eredménye az űrlap alatt kerül megjelenítésre. A felhasználó láthatja, hogy az eredmény hány szobát foglal magába, mennyi ideig tartott a művelet, mi az ajánlott szobák átlagos ára és egymáshoz viszonyított átlagos távolsága. Ezután a kiválasztott szobák kerülnek felsorolásra hasonlóan, mint a szobák listáján. A felsorolás alján, egy térképen piros jelzések mutatják az eredményben szereplő szálláshelyek elhelyezkedését. Egy jelzésre kattintva a szálláshely nevét tartalmazó szövegbuborék jelenik meg.

A találati lista felett, jobb oldalon található a *Kosárba* feliratú gomb, amellyel a szálláskereső a kosarába helyezheti a felsorolt szobákat.

### 8.5. Kosár

A szálláskereső felhasználó a virtuális kosarába gyűjti a foglalni kívánt szobákat. A kosár tartalmát bemutató felület megjelenése látható az alábbi ábrán.

2015-04-20		3		2015-04-23	
	Erdőre néző Deluxe kétágyas Hotel Fagus	2	41148.0 Ft/ éj		Töröl
	Superior kétágyas Hotel Fagus	1	48768.0 Ft/ éj		Töröl
<p>A foglalás teljes ára: <b>393192.0 Ft</b></p> <p>A feltüntetett árak tartalmazzák a 27%-os ÁFA-t. Az éjszakánként fizetendő idegenforgalmi adó mértéke szállodánként eltérhet.</p>					

Az oldal tetején a kiválasztott időszak látható, amely tartalmazza az érkezési és távozási dátumot, valamint az eltöltendő éjszakák számát. Ezt követi egy

táblázat, ami a választott szobákat szobatípusonként csoportosítva mutatja. Az első oszlop a szoba előnézeti képét, a második oszlop a szoba nevét, a szálláshely nevét és az ágyak számát, a harmadik oszlop a kiválasztott hasonló szobák számát, a negyedik oszlop a szoba árát az utolsó oszlop pedig a *Törlés* akciót tartalmazza. A *Törlés* akcióval egy szoba példány távolítható el a kosárból. A táblázatot a foglalás teljes árát összesítő sor követi. Ez alatt található két gomb. A bal oldali, *Kiürít* gomb a kosár teljes tartalmát törli, míg a középen lévő *Foglal* gomb a kosár tartalmául szolgáló szobák foglalását kezdeményezi.

A *Foglalás* gomb megnyomásakor a felhasználó egy új oldalra kerül, ahol véglegesítheti foglalását. Ez az oldal a vendégadatok megadására szolgál, felsorol minden, korábban a kosárba helyezett szobát, és a szobákhoz rendelve egyenként kell megadni a vendégek nevét és születési idejét. A foglalás véglegesítés felületet az [5] melléklet mutatja be. Az első megjelenő szoba mezőit a rendszer automatikusan kitölti a szálláskereső saját adataival.

### 8.6. Foglalások

A foglalások kezelése és megtekintése a szálláskereső és a szállásadó számára is fontos funkció, ezért a felületeket az egyszerűség és átláthatóság jegyében valósítottam meg.

A foglalásokat a szálláskereső számára összesítő oldalt a [6] mellékelt mutatja be. A foglalások állapotuk szerint vannak három táblázatba osztva, mely táblázatok egymás alatt helyezkednek el. A táblázatok állapotuk szerint más-más színűek, így könnyen megkülönböztethetők. A foglalás állapota lehet függő, visszaigazolt vagy teljesített. A táblázatban megjelenő adatok sorrendben a következők: foglalási szám, foglalt szobák, érkezési dátum, éjszakák száma, távozási dátum. A foglalási szám egy link, ami az adott foglalás részleteit bemutató oldalra vezet.

A szállásadó a szálláskeresőtől valamivel eltérő módon tekintheti át a foglalásokat. A három állapotot reprezentáló táblázat három külön felületen kapott helyet, hiszen a szállásadónak lényegesen több foglalást kell áttekinteni, mint a szálláskeresőnek. A felületeket a [7] melléklet mutatja be. A táblázatokban

megjelenő adatok megegyeznek a szálláskeresőnél látottakkal. Mivel a táblázatok külön felületeken jelennek meg, ezért nem kaptak különböző színezést. Más színűek viszont a visszaigazolt állapotú foglalásokat felsoroló táblázat bejegyzéseinek háttere. Az elfogadott állapotot zöld, míg az elutasított foglalást piros háttérrel jeleztem.

Egy foglalás részleteit bemutató oldal is másként jelenik a szálláskereső és szállásadó számára, de alapjaiban hasonlóak. A szálláskereső a foglalás időszakát, a foglalt szobákat, a szobákhoz bejegyzett vendégadatokat és a foglalás állapotát tekintheti meg. A szálláskereső az oldal alján megjelenő panelben tud a foglalás állapota szerint műveleteket végrehajtani. Visszautasított foglalás esetén a foglalás visszahelyezhető a kosárba és újra foglalható. Teljesített foglalás esetén a szálláskereső megírhatja az utazás során tapasztalt élményeit a megjelenő értékelő mezőkben. A taglalt felületeket a [8] melléklet mutatja be.

A szállásadó részére a foglalás részletei oldalon láthatók a foglaló szálláskeresőhöz tartozó személyes adatok is, úgy, mint név, születési idő, lakcím, email cím és telefonszám. A szobák és a foglalás állapota hasonlóan jelenik meg, mint a szálláskereső számára, ahogy az a [9] mellékletben látható. A szállásadó szintén tud akciókat végrehajtani a foglalás állapota szerint. Foglalt állapotban az elfogadás és visszautasítás akciókat végrehajtó gombok jelennek. Elfogadott foglalás esetén, ha a napi dátum a foglalás dátuma utáni, akkor aktiválódik a teljesítést igazoló gomb.

## 9. Tesztelés

Ez a fejezet a webalkalmazás intelligens keresés funkciójának tesztelését mutatja be. A szoba szűrés funkció része az intelligens keresésnek, ezért annak tesztelésére külön nem tértem ki.

### 9.1. Tesztelési környezet

A webalkalmazás elkészítése közben célom volt, hogy a működő verziók nem csak a fejlesztői gépen, de bárhol elérhetők legyenek. Ehhez egy VPS (Virtual Private Server) szolgáltatást használtam, vagyis egy virtuális szervergépre telepítettem olyan környezetet, amely Ruby on Rails alkalmazások futtatását és internetes közzétételét valósítja meg. A megoldás lehetővé tette, hogy a fejlesztői környezet elindítása nélkül, valós körülmények között tesztelhessem az elkészült funkciók működését bármilyen internetes kapcsolattal rendelkező eszközről.

A webalkalmazás működését és megjelenését Windows és Linux operációs rendszereken Google Chrome és Mozilla Firefox webböngészők használatával teszteltem, emellett kipróbáltam Android rendszert futtató mobiltelefonon és tableten is egyaránt.

### 9.2. Teszt adatok

Az intelligens keresés funkció teszteléséhez a webalkalmazás adatbázisát releváns tartalommal kellett feltöltenem. Regisztráltam több szállásadót, akik több fiktív vagy valós szálláshellyel rendelkeznek. A szálláshelyek mindegyikéhez töltöttem fel szobákat, úgy, hogy azok árai reális értéket képviseljenek, de érzékelhető szórást mutassanak. A szálláshelyek szolgáltatásait és a szobák felszereltségét szintén igyekeztem minél változatosabban, de az átfedéseket szem előtt tartva meghatározni. A tesztelés megkönnyítése miatt a szobák számát lényegesen lecsökkentettem, ezzel kívántam érzékeltetni a szálláshelyek csúcsidei foglaltságát és egy szabad szobákban szegény időszakot. Egy szobatípusból jellemzően 2-8 példányt jegyeztem be egy szálláshelyen. A szálláshelyeket úgy választottam ki, hogy azok Budapest területén, de elszórtan helyezkedjenek el. Így került a regisztrálásra belvárosi luxusszálloda és hostel, valamint külvárosi tranzit szálloda és apartmanház is.



### 9.3. Teszt eredmények

Az intelligens keresés funkció tesztelése során bizonyítást nyert, hogy a megtervezett optimalizálási modellek helyesen működnek. A három szempont szerinti keresés a várt eredményeket szolgáltatta. Megjegyzendő azonban, hogy az eredmények nem tökéletesek, egyes esetekben kissé eltérnek az optimalitástól. Az eltérések azonban a hétköznapi felhasználás során elhanyagolhatók és elsősorban a távolság támasztotta feltételek szerint helytelenek. A nem várt működés okozója az lehet, hogy a nemlineáris megoldó algoritmusok még viszonylag korai fejlesztések és további pontosításukra van szükség. A találatok közti különbségek összehasonlításához jó felület a találati lista alatt megjelenített térkép, amely az eredményben szereplő szálláshelyeket mutatja. A teszteredményeket a 0 melléklet mutatja be részletesen.

Technikai szempontból az operációs rendszerek és a webböngészők között nem tudtam különbséget tenni. A webalkalmazás minden platformon az elvárások szerint helyesen működött. Az alkalmazkodó web dizájn használatával mobiltelefonon és tableten is élvezhető működést értem el.

## 10. Összefoglalás

Kutatásom arra derített fényt, hogy a magyarországi népszerű szálláskereső portálok csak korlátozott mértékben támogatják a csoportos utazók szállásfoglalási igényeit. A terület ilyen elhanyagoltsága vezérelt a szakdolgozati téma megfogalmazása és a feladat megoldása során. Az általam kitalált koncepció a szokásostól eltérően a szobákat helyezi a szállásfoglalás középpontjába, ezáltal a szobák egy szálláshelyektől független vegyes piacát képezve.

A munkám során elkészítettem egy Ruby on Rails alapú webalkalmazást, amely csoportok számára könnyíti meg a szállásfoglalást azáltal, hogy egy szoba orientált webshop-szerű működést biztosít. Az elkészült web alkalmazás képes szobák szűrésére, keresésére és olyan foglalások feldolgozására, amelyekben több szálláshely szobái szerepelnek egyszerre. Implementáltam az intelligens keresés funkciót, amelyhez kialakítottam az optimalizálási modelleket és sikeresen a rendszerhez illesztettem az optimalizálást végrehajtó szoftvert. Mindezek mellett létrehoztam azokat a felületeteket, ahol a szállásadók a szálláshelyeiket és szobáikat szerkeszthetik, illetve a foglalásokat kezelhetik. Az adminisztrátorok számára szerkeszthetővé tettem a rendszerben szereplő törzsadatokat és beállításokat.

Az elkészült szoftver a tervezés során kialakított feltételeket teljesíti és bír minden elképzelt funkcióval. Sikeresen oldja meg a csoportos szállásfoglalás egyszerűbbé tételét, amelyben nem csak a létrehozott automatikus szobaajánló funkció, de a letisztult, ugyanakkor modern és szép felület is fontos szerepet játszik.

## **Irodalomjegyzék**

- [1] BERTÓK, B., KOVÁCS, Z. (2011). Gyártórendszerek modellezése: *Lineáris programozás*. Typotex, Veszprém
- [2] [http://en.wikipedia.org/wiki/Mathematical\\_optimization](http://en.wikipedia.org/wiki/Mathematical_optimization) (letöltés dátuma 2015. április 17.) *Optimization problems*
- [3] <http://www.mitrikitti.fi/opthist.html> MITRI K. (letöltés dátuma 2015. április 17.) *History of Optimization*
- [4] [http://en.wikipedia.org/wiki/Linear\\_programming](http://en.wikipedia.org/wiki/Linear_programming) (letöltés dátuma 2015. április 17.) *Linear programming*
- [5] [http://en.wikipedia.org/wiki/Nonlinear\\_programming](http://en.wikipedia.org/wiki/Nonlinear_programming) (letöltés dátuma 2015. április 17.) *Nonlinear programming*
- [6] <http://www.postgresql.org/about/> (letöltés dátuma 2015. április 17.) *About*
- [7] <http://ampl.com/products/ampl/> (letöltés dátuma 2015. április 17.) *AMPL*
- [8] <http://ampl.com/products/solvers/open-source/> (letöltés dátuma 2015. április 17.) *Nonlinear solvers*
- [9] <http://www.uzletresz.hu/vallalkozas/20120206-budapesti-iroda-nyitasat-tervezi-a-bookingcom.html> [ORIGO] (letöltés dátuma 2015. április 17.) *Budapesti irodát nyit a Booking.com*
- [10] <http://hu.wikipedia.org/wiki/Szallas.hu> (letöltés dátuma 2015. április 17.) *Szallas.hu*
- [11] <https://gorails.com/setup/ubuntu/14.04> (letöltés dátuma 2015. április 17.) *Setup Ruby On Rails on Ubuntu 14.04 Trusty Tahr*
- [12] [http://guides.rubyonrails.org/association\\_basics.html](http://guides.rubyonrails.org/association_basics.html) (letöltés dátuma 2015. április 17.) *Active Record Associations*
- [13] [http://en.wikipedia.org/wiki/Ruby\\_on\\_Rails#History](http://en.wikipedia.org/wiki/Ruby_on_Rails#History) (letöltés dátuma 2015. április 17.) *Ruby on Rails*

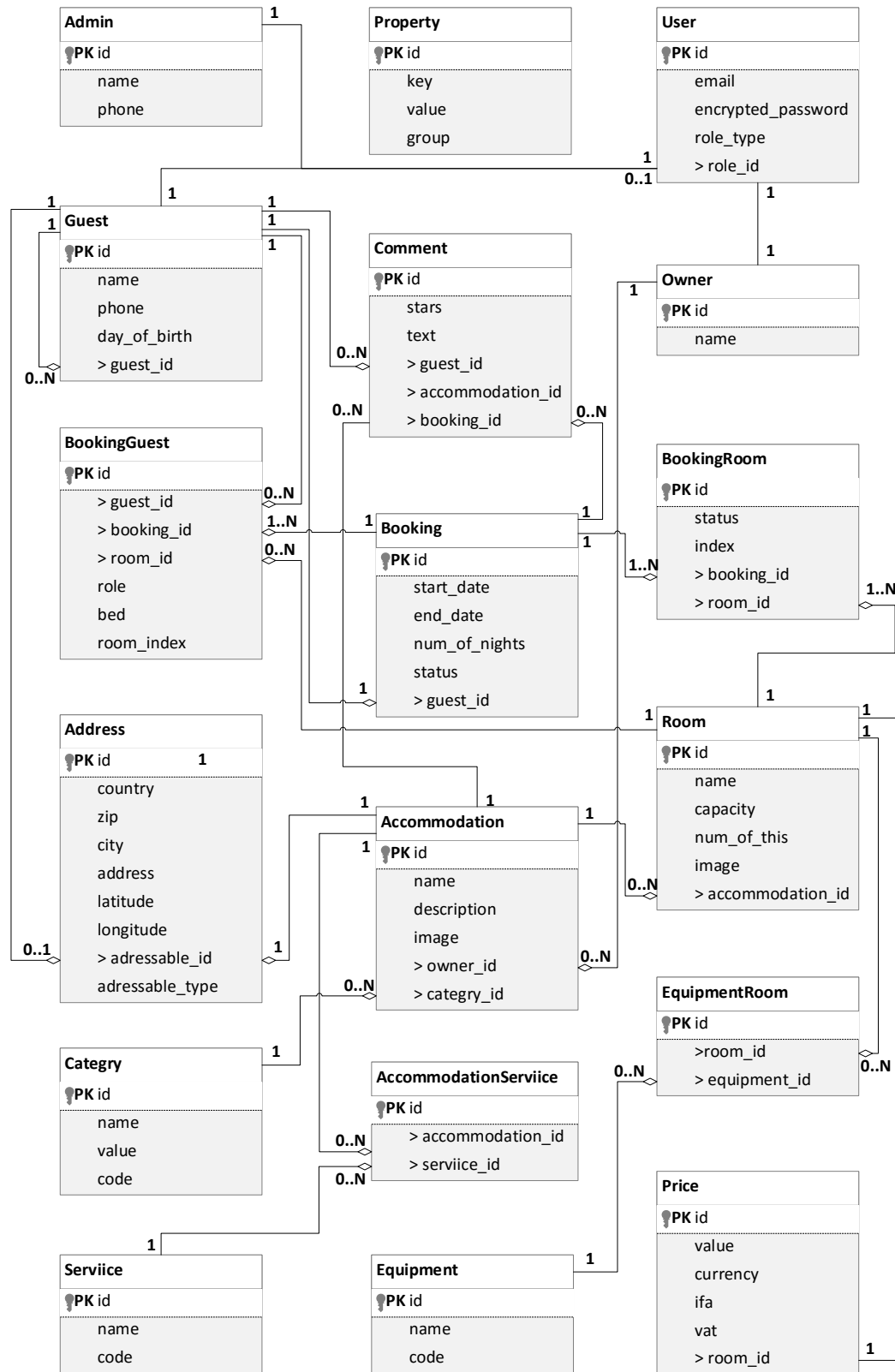
- [14] <http://railsapps.github.io/what-is-ruby-rails.html> (letöltés dátuma 2015. április 17.) *What is Ruby on Rails?*
- [15] <http://www.google.com/trends/explore#q=ruby%20on%20rails%2C%20laravel%2C%20nodejs%2C%20jsf%2C%20angularjs&cmpt=q&tz=>  
(letöltés dátuma 2015. április 17.) Google Trends: *Összehasonlítás: ruby on rails, laravel, nodejs, jsf, angularjs*
- [16] <https://www.ruby-lang.org/en/> (letöltés dátuma 2015. április 18.) *Ruby is ...*
- [17] [http://en.wikipedia.org/wiki/Yukihiro\\_Matsumoto](http://en.wikipedia.org/wiki/Yukihiro_Matsumoto) (letöltés dátuma 2015. április 18.) *Yukihiro Matsumoto*

## Ábrajegyzék

6.1 ábra Szobafoglalás folyamata.....	27
6.2 ábra Foglалás visszaigazolás folyamata .....	28
6.3 ábra Intelligens keresés háttérfolyamata.....	29
6.4 ábra Árak átalakítása (Ft) .....	30
6.5 ábra Távolságok átalakítása (km).....	30
6.6 ábra A modellben megjelenő szoba objektum a hozzá kapcsolódó változóval és bázisparaméterekkel .....	31
6.7 ábra Az olcsó modellhez szükséges paraméterek .....	32
6.8 ábra A közeli modellhez szükséges paraméterek .....	32
6.9 ábra Az olcsó és közeli modellhez szükséges paraméterek.....	33
8.1 ábra Látogató számára látható menüsáv .....	62
8.2 ábra Szálláskereső számára látható menüsáv .....	63
8.3 ábra Szállásadó számára látható menüsáv .....	63
8.4 ábra Adminisztrátor számára látható menüsáv.....	64
8.5 ábra Szobák listája .....	64
8.6 ábra Szobák szűrési feltételeit tartalmazó panel .....	65
8.7 ábra Egy szoba részletes bemutató oldala .....	66
8.8 ábra A szálláskereső számára megjelenő szobafoglalási panel.....	66
8.9 ábra Rögzített szobafoglalási panel.....	66
8.10 ábra Szálláshelyek listája.....	67
20. ábra Egy szálláshely részletes bemutató oldal .....	68
6.1 képlet Speciális relatív szórás képlet .....	31
6.2 képlet Korlátozás a vendégek száma alapján .....	31
6.3 képlet Az olcsó modell célfüggvénye .....	32
6.4 képlet A közeli modell célfüggvénye .....	33
6.5 Az olcsó és közeli modell célfüggvénye .....	33

## Mellékletek

### [1] Adatbázis diagram



## [2] UrlHelper segédosztály

```
module UrlHelper
  def self.build_parameterised_url(params)
    puts params

    base = params[:base_url]
    params.delete(:base_url)

    if base.nil?
      return nil
    else
      params = remove_empty_params(params)
      write_params(base, params)
    end
  end

  def self.remove_empty_params(params)
    params[:equipment_ids].delete_at(params[:equipment_ids].length-1) unless params[:equipment_ids].nil?
    params[:service_ids].delete_at(params[:service_ids].length-1) unless params[:service_ids].nil?

    params.delete(:city) if params[:city].empty?
    params.delete(:start_date) if params[:start_date].empty?
    params.delete(:end_date) if params[:end_date].empty?
    params.delete(:equipment_ids) if params[:equipment_ids].empty?
    params.delete(:service_ids) if params[:service_ids].empty?

    if params[:filter] == 'fine'
      params.delete(:capacity) if params[:capacity].empty?
    elsif params[:filter] == 'smart'
      params[:close] = params[:close] if params.has_key?('close')
      params[:cheap] = params[:cheap] if params.has_key?('cheap')

      params[:one_bed] = params[:one_bed] if params.has_key?('one_bed')
      params[:two_bed] = params[:two_bed] if params.has_key?('two_bed')
```

```

    params[:three_bed] = params[:three_bed] if
    params.has_key?('three_bed')
    params[:four_or_more_bed] =
    params[:four_or_more_bed] if
    params.has_key?('four_or_more_bed')

    params.delete(:guests) if params[:guests].empty?
end

return params
end

def self.write_params(base, params)
  url = base + '?'

  params.keys.each_with_index do |key, idx|
    url += "#{key}="
    if params[key].is_a? Array
      params[key].each_with_index do |v,i|
        url += "#{v}"

        unless i+1 == params[key].length
          url += ','
        end
      end
    else
      url += "#{params[key]}"
    end

    url += '&' if idx + 1 < params.keys.length
  end

  return url
end
end

```



### [3] FilterHelper segédosztály szobák szűrését megvalósító

#### metódusai

```
def self.filter_rooms(params)
  rooms_by_date = get_rooms_by_date(params[:start_date],
  params[:end_date])
  rooms_by_city = get_rooms_by_city(params[:city])
  rooms_by_equipment =
  get_rooms_by_equipment(params[:equipment_ids])
  rooms_by_service =
  get_rooms_by_service(params[:service_ids])
  rooms_by_capacity = get_rooms_by_capacity(params)
  filter_viewpoints = get_filter_viewpoints(params)
  combine_filters(rooms_by_date, rooms_by_city,
  rooms_by_equipment, rooms_by_service,
  rooms_by_capacity, filter_viewpoints)
end

def self.get_rooms_by_date(start_date, end_date)
  rooms_by_date = Array.new
  unless start_date.nil? && end_date.nil?
    Room.all.each do |room|
      if BookingsHelper.is_bookable(room,
      Date.strptime(start_date, '%Y.%m.%d'),
      Date.strptime(end_date, '%Y.%m.%d'))
        rooms_by_date.push(room)
      end
    end
  end
  return rooms_by_date
end

def self.get_rooms_by_city(city)
  rooms_by_city = Array.new
  unless city.nil?
    rooms_by_city = Room.joins(:accommodation =>
    [:address]).where('lower(addresses.city) = ?',
    city.downcase)
  end
  return rooms_by_city
end

def self.get_rooms_by_equipment(equipment_ids)
  rooms_by_equipment = Array.new
  unless equipment_ids.nil?
    equipment_ids =equipment_ids.split(',')
  end
end
```

```

    Room.all.each do |room|
      re = room.equipments.where(id: equipment_ids)
      if re.length == equipment_ids.length
        rooms_by_equipment.push(room)
      end
    end
  end
  return rooms_by_equipment
end

def self.get_rooms_by_serviice(serviice_ids)
  rooms_by_serviice = Array.new
  unless serviice_ids.nil?
    serviice_ids = serviice_ids.split(',')
    Accommodation.all.each do |accommodation|
      as = accommodation.serviices.where(id:
        serviice_ids)
      if as.length == serviice_ids.length
        rooms_by_serviice.concat(accommodation.rooms)
      end
    end
  end
  return rooms_by_serviice
end

def self.get_rooms_by_capacity(params)
  rooms_by_capacity = Array.new
  if params.has_key? :capacity
    rooms_by_capacity = Room.where(:capacity =>
      params[:capacity])
  else
    if params[:one_bed] == '1'
      rooms_by_capacity.concat(Room.where(:capacity =>
        1))
    end
    if params[:two_bed] == '1'
      rooms_by_capacity.concat(Room.where(:capacity =>
        2))
    end
    if params[:three_bed] == '1'
      rooms_by_capacity.concat(Room.where(:capacity =>
        3))
    end
    if params[:four_or_more_bed] == '1'
      rooms_by_capacity.concat(Room.where('capacity >=
        ?', 4))
    end
  end
end

```

```

    end
    return rooms_by_capacity
end

def self.get_filter_viewpoints(params)
  filter_viewpoints = 0
  filter_viewpoints += 1 if params.has_key?(:start_date)
  && params.has_key?(:end_date)
  filter_viewpoints += 1 if params.has_key? :city
  filter_viewpoints += 1 if params.has_key?
    :equipment_ids
  filter_viewpoints += 1 if params.has_key? :service_ids
  filter_viewpoints += 1 if params.has_key?(:capacity) ||
  params.has_key?(:one_bed) || params.has_key?(:two_bed)
  || params.has_key?(:three_bed) ||
  params.has_key?(:four_or_more_bed)
  return filter_viewpoints
end

def self.combine_filters(rooms_by_date, rooms_by_city,
rooms_by_equipment, rooms_by_service, rooms_by_capacity,
filter_viewpoints)
  rooms = Room.all
  intersected_rooms_ids = Hash.new
  filtered_rooms = Array.new
  rooms.concat(rooms_by_city) unless rooms_by_city.empty?
  rooms.concat(rooms_by_capacity) unless
  rooms_by_capacity.empty?
  rooms.concat(rooms_by_service) unless
  rooms_by_service.empty?
  rooms.concat(rooms_by_equipment) unless
  rooms_by_equipment.empty?
  rooms.concat(rooms_by_date) unless rooms_by_date.empty?
  rooms.each do |room|
    if intersected_rooms_ids.has_key?(room.id)
      intersected_rooms_ids[room.id] += 1
    else
      intersected_rooms_ids[room.id] = 1
    end
  end
  intersected_rooms_ids.keys.each do |key|
    if intersected_rooms_ids[key] == filter_viewpoints+1
      filtered_rooms.push(Room.find(key))
    end
  end
  return filtered_rooms
end

```

## [4] Intelligens keresés felület

### Intelligens keresés

Sopron

2015.04.20

2015.04.23

14

Felszereltség

☒ WiFi ☐ Ethernet ☐ Televízió ☐ Telefon ☐ Minibár ☐ Széf ☐ Légkondicionáló

Keresési szempontok

A szobák legyenek minél olcsóbbak:

IGEN

A szobák legyenek minél közelebb egymáshoz:

NEM

Szolgáltatások

☐ Jacuzzi ☐ Fedett uszoda ☐ Italbár ☐ Masszázs ☒ Parkoló


Szobatípus

☒ Egyágyas ☒ Kétágyas ☐ Háromágyas ☐ Négy vagy több ágyas


Keresés

Találatok 8 szoba (0.05 másodperc) - Átlagos ár: 29670.38 Ft - Átlagos távolság: 0.97 km


Kosárba




Standard kétágyas -  
Hotel Lövér Sopron  
€ € €




Standard kétágyas -  
Hotel Lövér Sopron  
€ € €




Standard kétágyas -  
Hotel Lövér Sopron  
€ € €




Standard kétágyas -  
Hotel Lövér Sopron  
€ € €




Superior kétágyas -  
Hotel Lövér Sopron  
€ € €



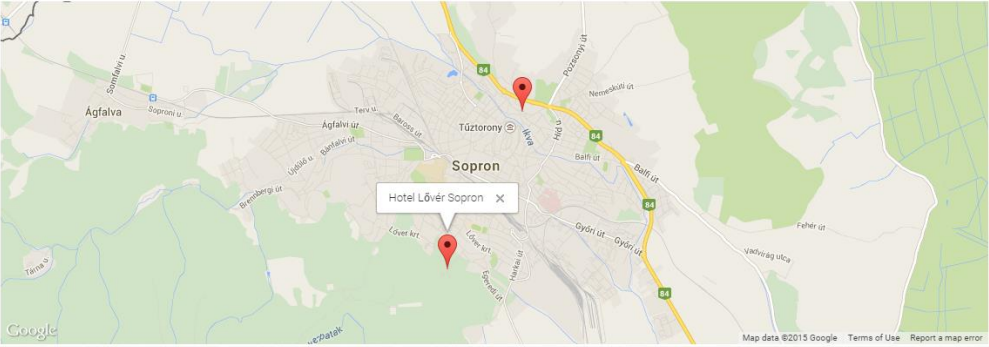
Superior kétágyas -  
Hotel Lövér Sopron  
€ € €



Fürdőszobás Standard  
egyágyas - Hotel Sopron  
€ €



Fürdőszobás Standard  
egyágyas - Hotel Sopron  
€ €



Map data ©2015 Google Terms of Use Report a map error

84


## [5] Foglalás véglegesítés felület

### Foglalás véglegesítés

2015-04-20

3

2015-04-23




Erdőre néző Deluxe kétágyas - Hotel Fagus  
€ € € €

Név

Született

Név

Született




Erdőre néző Deluxe kétágyas - Hotel Fagus  
€ € € €

Név

Született

Név

Született



Superior kétágyas - Hotel Fagus  
€ € € €

Név

Született

Név

Született

A foglalás ára: 393192.0 Ft

A feltüntetett árak tartalmazzák a 27%-os ÁFA-t. Az éjszakánként fizetendő idegenforgalmi adó mértéke szállodánként eltérhet.

Mégse

Foglalás véglegesítése

## [6] Foglалások (szálláskereső)

Foglалások				
Függő foglalások				
Foglалási szám	Szobák	Érkezés	Távozás	Éjszakák
24	Superior szoba (Mercure Budapest Buda) Standard studio (Hotel Charles)	2015-06-25	2015-06-27	2
Visszaigazolt foglalások				
Foglалási szám	Szobák	Érkezés	Távozás	Éjszakák
22	Erdőre néző Deluxe kétágyas (Hotel Fagus) Erdőre néző Deluxe kétágyas (Hotel Fagus) Superior kétágyas (Hotel Fagus)	2015-04-20	2015-04-23	3
Teljesült foglalások				
Foglалási szám	Szobák	Érkezés	Távozás	Éjszakák
21	Standard kétágyas (Hotel Lővér Sopron)	2015-03-30	2015-03-31	1

## [7] Foglалások (szállásadó)

Függő foglalások		Visszaigazolt foglalások		Teljesült foglalások	
Foglалási szám	Szobák	Érkezés	Távozás	Éjszakák	
25	Fürdőszobás Standard háromágyas (Hotel Sopron )	2015-05-28	2015-05-30	2	
Függő foglalások		Visszaigazolt foglalások		Teljesült foglalások	
Foglалási szám	Szobák	Érkezés	Távozás	Éjszakák	
25	Fürdőszobás Standard háromágyas (Hotel Sopron )	2015-05-28	2015-05-30	2	
22	Erdőre néző Deluxe kétágyas (Hotel Fagus) Erdőre néző Deluxe kétágyas (Hotel Fagus) Superior kétágyas (Hotel Fagus)	2015-04-20	2015-04-23	3	
Függő foglalások		Visszaigazolt foglalások		Teljesült foglalások	
Foglалási szám	Szobák	Érkezés	Távozás	Éjszakák	
21	Standard kétágyas (Hotel Lővér Sopron)	2015-03-30	2015-03-31	1	



## [8] Foglalás részletei (szálláskereső)

### 25. számú foglalás


Időszak

2015-05-28

2015-05-30

2

Elhelyezés



**Fürdőszobás Standard háromágyas** - Hotel Sopron  
€ € € €

Név: Nagy Gábor

Született: 1925.02.20

Név: Nagy Gáborné

Született: 1925.02.20

Név: Nagy Péter

Született: 1955.02.20

A foglalás ára: **96012.0 Ft**

A feltüntetett árak tartalmazzák a 27%-os ÁFA-t. Az éjszakánként fizetendő idegenforgalmi adó mértéke szállodánként eltérhet.

Státusz - válaszra vár

A foglalást vissza kell igazolnia a szálláshely(ek)nek.

Státusz - teljesítésre vár

A foglalást elfogadásra került a szálláshely(ek) által.

Státusz - visszautasítva

A foglalás vissza lett utasítva a szálláshely(ek) által.

A foglalás visszahelyezése a kosárba

Státusz - teljesítve

A foglalás teljesült.

Értékelje az üdülést!

Hotel Lóvér Sopron

★★★★★★★★★★

Mentés

87

## [9] Foglalás részletei (szállásadó)

### 25. számú foglalás

Foglaló adatai

Név:

Nagy Gábor

Született:

1925.02.20

Telefonszám:

0123435

Email cím:

v@v.hu

Lakcím:

0123 AL Sopron Deák Ferenc tér 1.


Időszak

2015-05-28

2

2015-05-30

Elhelyezés



**Földszobák Standard**  
háromgyás - Hotel  
Sopron  
€ € € €

Név:

Nagy Gábor

Született:

1925.02.20

Név:

Nagy Gáborné

Született:

1925.02.20

Név:

Nagy Péter

Született:

1955.02.20

A foglalás vonatkozó részének ára: **96012.0 Ft**

A feltüntetett árak tartalmazzák a 27%-os ÁFA-t.

Státusz - válaszra vár

Elutasít

Elfogad

Státusz - teljesítésre vár

Tejesítve

Státusz - teljesítésre vár

Tejesítve

Státusz - visszautasítva

A foglalás vissza lett utasítva.

Státusz - teljesítve

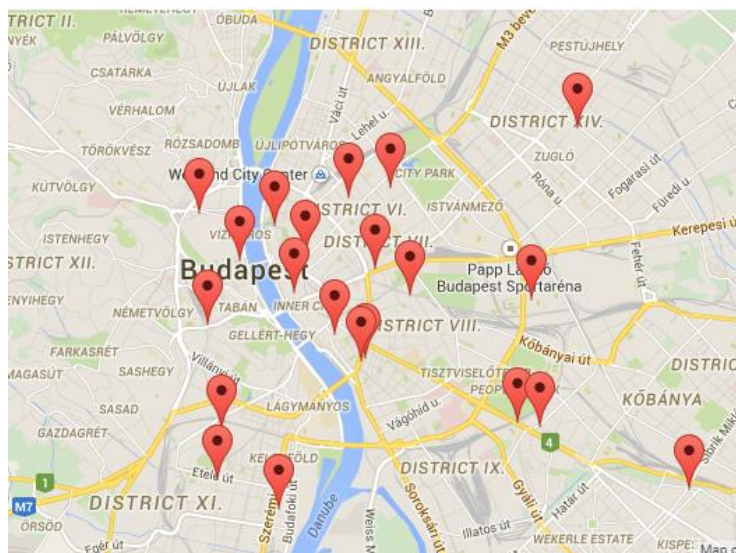
A foglalás teljesült.

88



## [10] Tesztelési eredmények

Az adatbázisba felvitt összes szálláshely elhelyezkedését mutatja be az alábbi ábra:



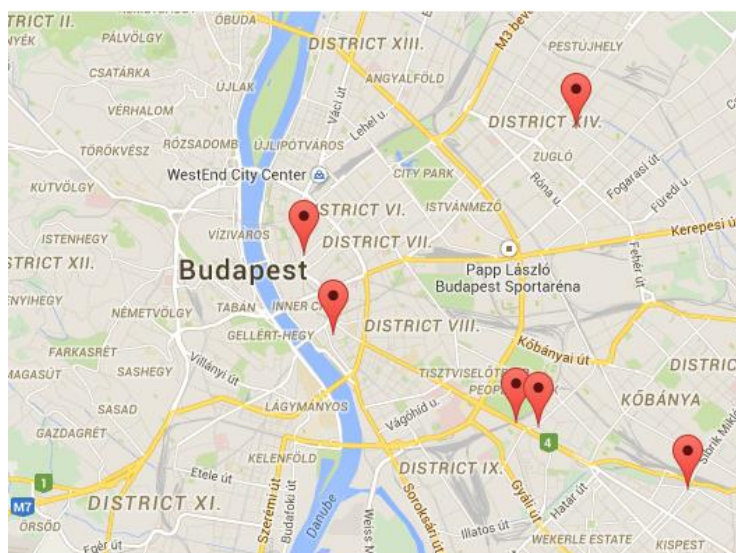
### 1. Teszt

A tesztelésben szereplő feltételek: **58 vendég, Budapest város**

A tesztelés szempontja: **olcsó**

Átlagos ár: **13.909,04 Ft**

Átlagos távolság: **4,39 km**



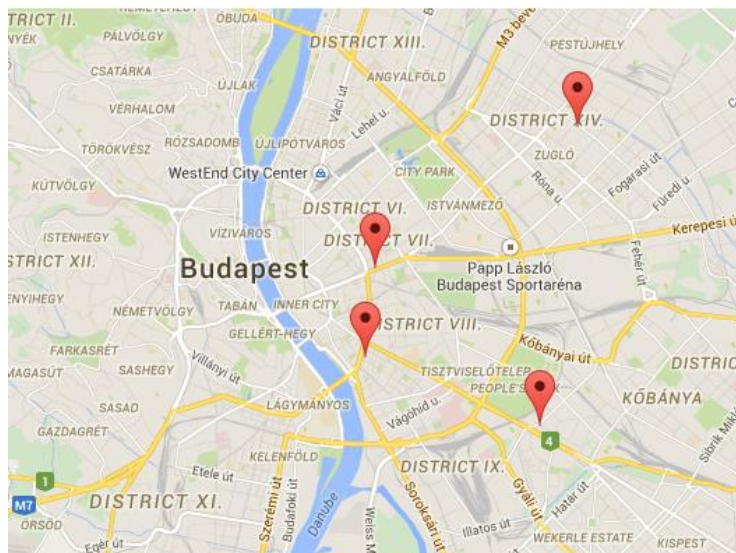
## 2. Teszt

A tesztelésben szereplő feltételek: **58 vendég, Budapest város, WiFi, Televízió, Parkoló**

A tesztelés szempontja: **olcsó**

Átlagos ár: **17.421,23 Ft**

Átlagos távolság: **3,28 km**



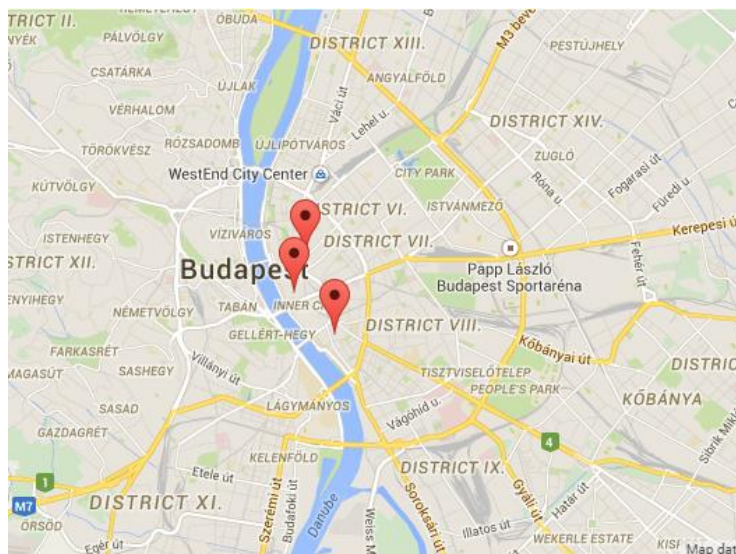
## 3. Teszt

A tesztelésben szereplő feltételek: **58 vendég, Budapest város**

A tesztelés szempontja: **közel**

Átlagos távolság: **0,78 km**

Átlagos ár: **26.312,28 Ft**





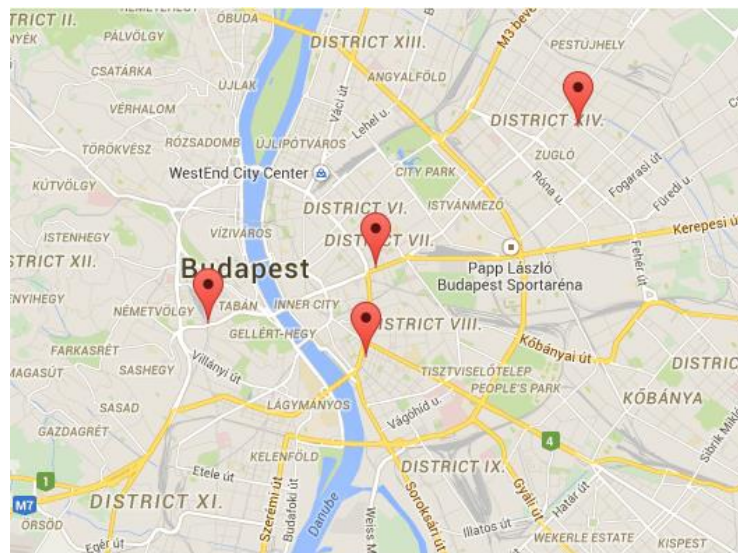
#### 4. Teszt

A tesztelésben szereplő feltételek: **58 fős csoport, Budapest város, WiFi, Televízió, Parkoló**

A tesztelés szempontja: **közeli**

Átlagos távolság: **1,76 km**

Átlagos ár: **25.784,91 Ft**



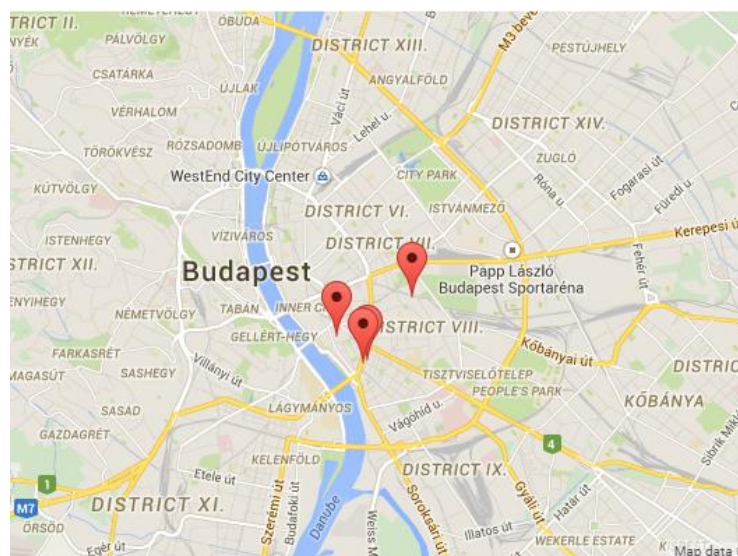
#### 5. Teszt

A tesztelésben szereplő feltételek: **58 vendég, Budapest város**

A tesztelés szempontja: **olcsó és közeli**

Átlagos ár: **18.998,05 Ft**

Átlagos távolság: **0,45 km**



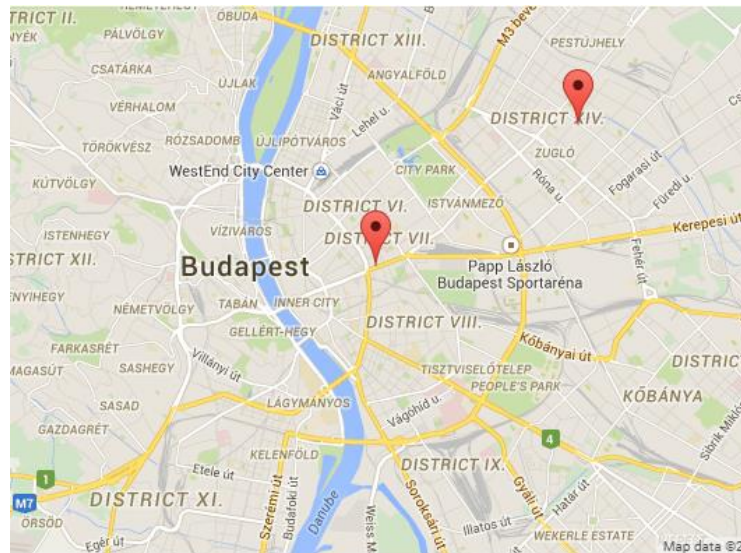
## 6. Teszt

A tesztelésben szereplő feltételek: **58 vendég, Budapest város, WiFi, Televízió, Parkoló**

A tesztelés szempontja: **olcsó és közeli**

Átlagos ár: **23.648,46 Ft**

Átlagos távolság: **2,14 km**



## **CD Melléklet**

A CD melléklet tartalma:

- Dolgozat
  - Szakdolgozat\_RozsenichBalázs\_Y9UXKT.docx
  - Szakdolgozat\_RozsenichBalázs\_Y9UXKT.pdf
- Források
  - Active Record Associations — Ruby on Rails Guides.pdf
  - AMPL \_ AMPL.pdf
  - bertok\_kovacs\_gyartorendszer.pdf
  - Budapesti irodát nyit a Booking.pdf
  - Google Trends - Internetes kereső – érdeklődés\_ ruby on rails, laravel, nodejs, jsf, angularjs – Világszerte.pdf
  - History of Optimization.pdf
  - Linear programming - Wikipedia, the free encyclopedia.pdf
  - Mathematical optimization - Wikipedia, the free encyclopedia.pdf
  - Nonlinear programming - Wikipedia, the free encyclopedia.pdf
  - Open Source Solvers \_ AMPL.pdf
  - PostgreSQL\_ About.pdf
  - Ruby on Rails - Wikipedia, the free encyclopedia.pdf
  - Ruby Programming Language.pdf
  - Szallas.pdf
  - What is Ruby on Rails\_ · RailsApps.pdf
  - Yukihiro Matsumoto - Wikipedia, the free encyclopedia.pdf
- Ábrák
  - Képernyőképek
    - accommodation.png
    - accommodations.png
    - booking\_guest\_approved.png
    - booking\_guest\_closed.png

- booking\_guest\_denied.png
- booking\_guest\_waiting.png
- booking\_owner\_approved.png
- booking\_owner\_approved\_can\_close.png
- booking\_owner\_closed.png
- booking\_owner\_denied.png
- booking\_owner\_waiting.png
- bookings\_guest.png
- bookings\_owner\_answered.png
- bookings\_owner\_closed.png
- bookings\_owner\_waiting.png
- cart.png
- cart\_book.png
- full\_page\_cart.png
- full\_page\_rooms.png
- full\_page\_smartfilter.png
- menu\_bar\_no\_login.png
- menubar\_admin\_login.png
- menubar\_guest\_login.png
- menubar\_owner\_login.png
- room.png
- room\_with\_booking.png
- room\_with\_booking\_empty\_cart.png
- rooms\_no\_filter.png
- rooms\_with\_filter.png
- smartfiltering.png
- smartfilter\_with\_results.png
- test\_all\_accommodation.png
- test\_cheap\_and\_close1.png
- test\_cheap\_and\_close2.png
- test\_cheap1.png

- test Cheap2.png
- test Close1.png
- test Close2.png
- Diagramok
  - distance\_category.vsdX
  - foglalasvisszaigazolas.vsdX
  - models.vsdX
  - price\_category.vsdX
  - room\_nlp\_object.vsdX
  - room\_nlp\_object\_extra1.vsdX
  - room\_nlp\_object\_extra2.vsdX
  - room\_nlp\_object\_extra3.vsdX
  - smartfiltering.vsdX
  - szobafoglalas.vsdX
- Forráskód
  - vagato-master.zip
  - vagato.dump