ÓBUDAI EGYETEM
ÓBUDA UNIVERSITY

Alba Regia Műszaki Kar

Természettudományi és Szoftvertechnológiai Intézet

SZAKDOLGOZAT

Szállodai ajánlórendszer fejlesztése Pythonban web scraping alapján

OE-AMK

2024

Hallgató neve: Bencze Emese

Hallgató törzskönyvi száma: T000863/FI12904/A-M

ÓBUDAI EGYETEM
ÓBUDA UNIVERSITY

Óbudai Egyetem
Alba Regia Műszaki Kar
Természettudományi és Szoftvertechnológiai Intézet

SZAKDOLGOZAT FELADATLAP

Hallgató neve: Bencze Emese
Szakdolgozat száma: SZD22011411176407
Törzskönyvi száma: T000863/FI12904/A-M **Neptun kódja:** [redacted]

Szak: mérnökinformatikus
Specializáció: Big Data és üzleti intelligencia specializáció

A dolgozat címe: Szállodai ajánlórendszer fejlesztése Pythonban web scraping alapján
A dolgozat címe angolul: Hotel Recommendation System Development in Python Based on Web Scraping
A feladat részletezése: A hallgató feladata web scraping módszer alkalmazásával gyűjtött szállodai adathalmazon alapuló webes ajánlórendszer fejlesztése Python környezetben. A hallgató mutassa be a lehetséges ajánlórendszer típusokat, részletezve azok működési elveit és alkalmazhatóságukat. A hallgató web scraping módszert alkalmazva gyűjtse és készítse elő a feladathoz kapcsolódó adathalmazt és mutassa be az alkalmazott elemzési módszereket. A hallgató valósítson meg egy felhasználóbarát webes felületet az ajánlások kezeléséhez és megjelenítéséhez, kiemelve a felhasználóbarát kezelhetőséget és interaktivitást. Értékelje a kapott eredményeket és a rendszer hatékonyságát. Mutassa be a továbbfejlesztési lehetőségeket és az ajánlórendszer jövőbeli fejlesztési irányait.

Intézményi konzulens neve: Módné Takács Judit

A kiadott téma elévülési határideje:2026. 07. 10.
Beadási határidő: 2024. 05. 15.

A szakdolgozat: Nem titkos.

ÓBUDAI EGYETEM Alba Regia Műszaki Kar Természettudományi és Szoftvertechnológiai Intézet Székesfehérvár

Kiadva: Székesfehérvár, 2024. 04. 03. P.H. .....
Intézetigazgató

A dolgozatot beadásra alkalmasnak találom: 2024. 05. 15.

.....
belső konzulens

.....
külső konzulens

ÓBUDAI EGYETEM
ÓBUDA UNIVERSITY

Alba Regia Műszaki Kar
Természettudományi és Szoftvertechnológiai Intézet

HALLGATÓI NYILATKOZAT

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

Kelt: SZÉKESFEHÉRVÁR, 2024. 05. 06.

.....

hallgató aláírása

Tartalomjegyzék

1. BEVEZETÉS

A szállodai ajánlórendszerek, mint téma, már bőven a szakdolgozatom elkezdése előtt felkeltette az érdeklődésemet. A téma nemcsak, hogy érdekes számomra, de közel is áll hozzám, mivel a magánéletemben rendszeresen szoktam különféle érdekes helyekre utazni. Egy kellemes, élményekkel teli utazás kulcsfontosságú eleme és egyben első lépése a megfelelő szálláshely kiválasztása. Ennek oka az, hogy egy kiadós túra vagy városfelfedezés után az egész esténket itt fogjuk tölteni, mondhatni erre az időtartamra az adott szálláshely lesz az otthonunk. Ha esetleg valami nincs rendben az idegilenes otthonunkkal, az könnyen el tudja rontani az utazásunkat. Emellett, ha jól megválasztjuk a helyet, a szálláshely által nyújtott extra szolgáltatások tovább fokozhatják az utazás élményét. Szinte minden alkalommal, amikor utazást tervezünk a családommal vagy a barátaimmal, előfordul, hogy akár több napon keresztül is keressük a számunkra megfelelő szállást. Pontosan egy ilyen keresés közben jöttem rá, hogy az egyetemen megszerzett tudásomnak köszönhetően akár meg is oldhatnám a problémám egy ajánlórendszer elkészítésével. Egy ilyen eszköz segítségével személyre szabott ajánlásokat kaphatnék a korábban látogatott szállodáim alapján, így már csak abból a pár, előre kiválogatott szálláshelyből kellene kiválasztanom a legmegfelelőbbet. Ez rendkívül felgyorsítaná az utazás tervezésének folyamatát, és – mivel önállóan szoktam lebonyolítani szinte minden részét – megkönnyítené a dolgom.

Dolgozatom során egy szállodai ajánlórendszert fogok elkészíteni, melyhez az adatokat „webes lekaparással” (angolul, és a továbbiakban: web scraping) fogom begyűjteni. Az ajánlórendszer a korábban látogatott szálláshelyek alapján fogja ajánlani a lehetséges szállásokat, a megadott helyen és időtartamban. Fő célom egy robusztus, könnyen használható, hasznos eszköz elkészítése. A dolgozat első részében először be fogom mutatni az ajánlórendszerek általános tulajdonságait és működési elveit, ezután fel fogom vázolni az ajánlórendszeremmel szembeni elvárásokat és elkészítem a projekt specifikációját, majd bemutatom a konkrét implementációt, a használt eszközökkel és módszerekkel együtt, különböző fejezetekben. Ezt egy kiértékelés, egy felhasználói véleménykutatás és egy lehetséges fejlesztésekkel kapcsolatos fejezet fog követni, végül egy magyar és egy angol nyelvű összefoglalóval zárom a munkám.

2. AJÁNLÓRENDSZEREK BEMUTATÁSA

Az ajánlórendszerek (recommender systems) olyan rendszerek, amelyek személyre szabott javaslatokat, ajánlásokat nyújtanak a felhasználók számára olyan elemekről vagy tartalmakról, amelyek leginkább relevánsak számukra. Az elérhető online tartalmak hatalmas növekedésével a felhasználók folyamatosan választási lehetőségek elé kerülnek. Ez döntési fáradtsághoz (decision fatigue) vezethet. [1]

A döntési fáradtság egy olyan jelenség, ellentétben egy diagnosztizálható orvosi állapottal, amely szerint minél több döntést hoz egy adott személy egy adott nap során, annál inkább kimerülté válik testileg, szellemileg és érzelmileg is. A döntési fáradtságban szenvedő személyeknek nehézségeik vannak a végrehajtó funkciókkal kapcsolatban, amely számos következménnyel járhat, beleértve a károsult ítélőképességet is. Ezért fontos, hogy a webplatformok mindegyik felhasználónak javaslatokat nyújtsanak az elemekről, ezáltal korlátozva a választási halmazt. Annak érdekében, hogy növeljék a felhasználói elégedettséget és elkötelezettséget, és hogy valamilyen szinten csökkentsék a döntési fáradtságot. [2]

Híres weboldalak és alkalmazások, amelyek ajánlórendszereket használnak:

- Youtube: minden percben 500 órányi videót töltenek fel az emberek. Ez azt jelenti, hogy 3.5 évbe telne megnézni pusztán az elmúlt óra videóit [3].
- Spotify: jelenleg a felhasználók több mint 100 millió zene, 5 millió podcast és 350 ezer hangoskönyv közül választhatnak [4].
- Amazon: a felhasználók közel 700 ezer különböző márka és 185 ezer különböző szolgáltató, összesen 350 millió terméke közül választhatnak [5].

2.1 Kollaboratív szűrés

A kollaboratív szűrés egy, az ajánlórendszerek által alkalmazott technika. Arra a gondolatra épül, hogy azok az emberek, akik a múltban egyetértettek, a jövőben is egyet fognak, ezáltal olyan tartalmat fog nekünk ajánlani, amelyet más, hozzánk hasonló felhasználók is kedveltek. Ennél a technikánál tehát figyelmen kívül hagyjuk az adott elemek tulajdonságait. A hasonló embereket csoportokba rendezzük, és olyan elemeket ajánlunk nekik, amelyeket a csoport többi tagja kedvelt. [6]

Létezik az úgynevezett szomszédság alapú kollaboratív szűrés, amelyhez a memória és a heurisztika alapú kollaboratív szűrés tartozik. Ennél a módszernél az eltárolt értékelések közvetlenül a jóslásnál, (azaz hogy mennyire fog tetszeni a felhasználónak az adott elem vagy tartalom) vannak felhasználva. A jóslás lehet felhasználó-alapú vagy elem-alapú. [7]

A modell alapú módszer – a szomszédság alapú kollaboratív szűréssel szemben – az értékeléseket a modellek tanítására használja. Az általános elképzelés az, hogy a felhasználó-elem interakciók alapján modellt készítünk, amely tartalmazni fogja az olyan információkat, mint a felhasználó típusa és az elem kategóriája. Ezután ezt a modellt az elérhető adatokkal tanítjuk, és felhasználjuk a felhasználók értékeléseinek jóslására. [7]

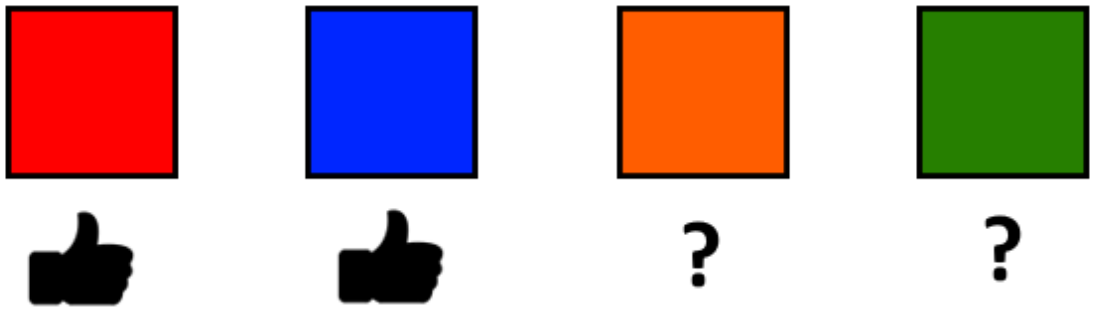
Előnyei: nem igényel területi tudást, és segíthet az új tartalmak megismerésében [8].

Hátrányai: a „hideg indítás" problémája, azaz a még nem értékelt elemekkel nem tud mit kezdeni [8].

Vegyünk egy egyszerű példát: a Youtube összegyűjti az adott felhasználó egyedi megtekintési szokásait, mint például a megtekintési és a keresési előzményeket, a csatornafeliratkozásokat, a lájkokat, a diszlájkokat, a "nem érdekel" és a "ne javasold ezt a csatornát" visszajelzés-választásokat és az elégedettségi felméréseket. Ezeket összehasonlítja más, a hozzá hasonló felhasználók megtekintési szokásaival, és ezen információk alapján tesz javaslatokat, amelyek talán érdekesek lehetnek a számára. [9] Ez a valóságban úgy nézne ki, hogy ha én mondjuk szállodákkal kapcsolatos értékelő videókat nézek és kedvelek, akkor a hasonló érdeklődésű felhasználók kedvelt/megtekintett videói közül ajánlana nekem párat (lehetőleg olyat, amit még nem láttam), és ugyan ez visszafelé is. Természetesen ez ennél egy sokkal komplexebb rendszer, de a kollaboratív szűrés megértéséhez elégséges a példa.

2.2 Tartalom alapú szűrés

Az ilyen rendszerek az elemek tulajdonságaira összpontosítanak, és a közöttük lévő hasonlóság alapján adnak ajánlásokat. Ez csak akkor lehetséges, ha az elemek tulajdonságai tisztán maghatározhatók, és ha létezik egy, az adott felhasználó által keresett tulajdonságokat tartalmazó lista. [10]



2.1. ábra - Példa a tartalom alapú szűrésre (Forrás: Saját szerkesztésű)

Ezen a nagyon egyszerű példán látható, hogy az adott felhasználónk kedvelte az első két elemet, és az utolsó kettőt nem ismeri. A rendszer ebben az esetben az utolsó kettőből az első elemet ajánlaná neki, mivel annak tulajdonsága (ez esetben a színe) hasonlít az első elem tulajdonságára.

Egy másik példa a szállodák ajánlása. Ha én olyan szállodákat látogatok, amelyek vízközeliek, állatbarátok, és alkalmasak egy kisebb család szükségleteinek kielégítésére, akkor valószínűleg a következő utazásom során is hasonló szállodába szeretnék utazni.

Előnyei: nincs „hideg indítási" probléma, mivel az elemek tulajdonságaira alapszik, valamint kiemelkedően személyre szabott ajánlásokat tud tenni [11].

Hátrányai: csak a felhasználó által előnyben részesített tulajdonságok alapján tud ajánlani, ezért az érdeklődési kört nem képes tágítani, valamint területi tudást igényel [11].

2.3 Hasonlóság vizsgálat

A kollaboratív szűrést alkalmazó ajánlórendszerek készítésénél gyakran használnak mátrixfaktorizációt, melynek alapja a felhaszáló-elem mátrix: [7]

	Film 1	Film 2	Film 3	Film 4
Felhasználó 1	5	4	1	5
Felhasználó 2	4	5		
Felhasználó 3	2	1	5	

2.2. ábra - Példa a felhasználó-elem mátrixra (Forrás: Saját szerkesztésű)

Az adott felhasználó-elem mátrix nagyon egyszerű, és kis adathalmazzal rendelkezik, de a lényeget tökéletesen bemutatja: az első és a második felhasználó majdnem ugyan any-

nyira kedvelte az első és a második filmet, tehát mondhatjuk azt, hogy az első két felhasználó hasonló, ezért egy csoportba tartoznak. Mivel az első felhasználónak nagyon tetszett a negyedik film, valószínűleg a második felhasználónak is fog, viszont a harmadik felhasználónak valószínűleg nem.

A hasonlóság vizsgálat másik nagyon gyakori módszere a Pearson korrelációs együttható használata: [7]

$$\text{Pearson (x, y)} = \frac{\sum(x,y)}{\sigma_x \times \sigma_y}$$

2.3. ábra - A Pearson korrelációs együttható képlete. [7]

2.4 Hibrid ajánlórendszerek

Névből adódóan ezek olyan rendszerek, amelyek a korábban említett technikákat alkalmazva megpróbálják egymás hibáit kiküszöbölni, a módszerek vegyítésével. Például a kollaboratív ajánlórendszerek egyik gyengepontja az új elemek, mivel nem tud értékelés nélküli elemeket ajánlani. Ez viszont a tartalom szerinti ajánlórendszereknek nem akadály, hisz az az elemek tulajdonságai alapján ajánl. [7]

A legjobb példa erre a módszerre a Netflix, mivel egyrészt összehasonlítja a hasonló felhasználók nézési és keresési szokásait (kollaboratív szűrés), valamint emellett olyan filmeket ajánl, melyeknek a tulajdonságai megegyeznek a korábban nézett filmekével (tartalom alapú szűrés). [12]

3. SPECIFIKÁCIÓ

3.1 Funkcionális specifikáció

Adatgyűjtés

Ahhoz, hogy az ajánlórendszerünk működőképes legyen, adatokkal kell ellátnunk. Az adatokat web scraping módszer használatával kell begyűjteni egy adott weboldalról, ezáltal biztosítva az adatok frissességét. Mind a felhasználó által korábban látogatott szállodák, mind a lehetséges szállodák adatait össze kell gyűjteni. A begyűjtött adatokon adattisztítást és adattranszformációt kell elvégezni, azaz egységes, megfelelő formátumúra kell alakítani az adatokat, és el kell távolítani a hiányos, zajos vagy kiugró értékekkel rendelkező rekordokat.

Ajánlás

A rendszer a felhasználó által korábban látogatott szállodák tulajdonságait összegyűjtve meg kell határozza, hogy a felhasználó által megszabott területen és időtartamon elérhető szállások közül melyek lehetnek számára a legideálisabbak.

Weboldal

Egy könnyen átlátható és használható weboldalon keresztül kell működjön a rendszer. Ezen a weboldalon a felhasználó megadhatja az általa korábban látogatott szállodákat, valamint azt, hogy hova, és mennyi időre szeretne utazni, és azt is, hogy hány felnőttel és gyerekkel. Ezután egy gombnyomásra megtörténik a háttérben az adatbeszerzés és az ajánlás elvégzése, és ezt követően a felhasználónak meg fog jelenni három, a számára legideálisabb szállás.

3.2 Nem funkcionális specifikáció

Sebesség

A web scraping természete miatt érdemes úgy elkészíteni a projektet, hogy a lehető legkevesebb időt vegye igénybe az adatbeszerzés. Az adatok mennyiségétől függően célszerű, ha maximum egy perc alatt megkapja a felhasználó az eredményt, a gombnyomástól számítva.

Használhatóság

A projekt használata intuitív kell legyen, és törekedni kell arra, hogy minél felhasználó barátabb legyen. Ehhez hozzájárul a weboldal igényes kinézete, és a könnyen kezelhető felhasználói felület megvalósítása.

Hibatűrés

A web scraping módszer természete miatt törekedni kell arra, hogy az esetleges hibákat a projekt jól tudja kezelni. Ez azt jelenti, hogy ha valamilyen nem kritikus hibába ütközne (például egy szállás nem minden keresett adata elérhető), akkor ne omoljon össze, hanem végezze el a szükséges korrigálást, és folytassa tovább a munkát.

Pontosság és megbízhatóság

A projekt által tett szálloda-ajánlások a felhasználó számára ténylegesen hasznosak és érdekesek kell legyenek, valamint az összegyűjtött adatok valósághűségére is vigyázni kell.

Karbantarthatóság

A kódnak jól strukturáltnak és átláthatónak kell lennie, a könnyebb karbantarthatóság és fejlesztés érdekében.

4. ADATGYŰJTÉS

4.1 Használt technológiák

BeautifulSoup és Requests

A Beautiful Soup könyvtár segítségével adatokat tudunk kinyerni HTML és XML fájlokból, mivel lehetővé teszi számunkra a dokumentumok navigálását [13]. Ez a projekt gerince, ugyanis ez maga a web scraping könyvtár, amellyel a számunkra fontos adatokat ki tudjuk nyerni az adott weboldalakról.

A Requests könyvtár segítségével pedig könnyen tudunk HTTP/1.1 kéréseket küldeni [14]. Ez a két könyvtár kéz a kézben jár, azért is vannak együtt megemlítve.

Pandas

A Pandas egy gyors, nagy erejű, rugalmas, könnyen használható, nyílt forráskódú adatelemző és adatmanipulációs eszköz, amely a Python nyelvre lett fejlesztve [15]. Ez a könyvtár kimondottan a DataFrame adatszerkezete miatt kulcsfontosságú, mivel lehetővé teszi a kétdimenziós (mátrixos) adattárolást: az oszlopok tartalmazzák a különböző tulajdonságokat, a sorok pedig az egyedeket (rekordokat). Emellett azonban számos egyéb funkciója is hasznunkra válhat (például az adatexportálás Excel-be).

Selenium

A Selenium egy ernyőprojekt, amely rengeteg eszköz és könyvtár segítségével lehetővé teszi a böngészők automatizálását. Kiterjesztések segítségével emulálni tudja a felhasználók interakcióját a böngészőkkel. Gyökere a WebDriver interfész, melynek segítségével hordozható kódot lehet készíteni, a különböző böngészőkre [16]. Erre főleg a dinamikus weblapelemek automatizálása miatt van szükség, mivel azokat a BeautifulSoup könyvtár nem tudja jól kezelni.

SelectorsHub

A SelectorsHub egy XPath és CSS kiválasztó böngésző bővítmény, melynek segítségével pár kattintás alatt el lehet érni egy adott weboldal elem XPath-jét [17]. Rendkívül fontos a Selenium és a BeautifulSoup könyvtárak használatához, mivel a weblapelemekre általában az XPath-jükkel szoktunk hivatkozni.

4.2 Web scraping

A web scraping az a folyamat, amikor botok (automatizált szoftverek) segítségével adatot nyerünk ki egy weboldalról. Ez a technika az adott weboldal HTML struktúrájára épül, és abból nyeri ki a számára szükséges adatokat. Számos területen alkalmazzák legálisan, például a keresőmotorok a weboldal megismerésére használják, hogy rangsorolni tudják őket. Egy másik legális példa az ár-összehasonlító weboldalak. Ezek a botok képesek a különböző HTML struktúrák navigálására, az adatgyűjtésre és az adattranszformációra, valamint a begyűjtött adatok tárolására. [18]

4.2.1 Legális problémák

Mivel mindegyik web scraper botnak ugyan az a célja (adatgyűjtés), ezért nehéz megkülönböztetni a legális és az illegális robotokat. Egy legális robot azonosítja magát a HTTP fejlécében, míg az illegális robotok álcázzák magukat. A másik fontos különbség, hogy a legális robotok követik az adott oldal robots.txt szabályzatát. A robots.txt egy szöveges dokumentum, amely tartalmazza mely al-oldalakat (subpages) érhetik el a robotok. [18]

A Google robots.txt fájlja

```
User-agent: *
Disallow: /search
Allow: /search/about
Allow: /search/static
Allow: /search/howsearchworks
```

4.1. ábra - A Google robots.txt fájljának egy részlete. (Forrás: https://www.google.com/robots.txt)

A fenti kép egy részlet a Google saját robots.txt fájljából. Az első sor azt jelenti, hogy minden web scraperre érvényes (de meg lehetne szabni, hogy például csak a Google keresőmotor robotjára legyen érvényes, akkor * helyett Googlebot lenne). Az „Allow:” sorok engedélyezik, a „Disallow:” sorok pedig tiltják az adott URL-en található al-oldalak elérését. [19]

Tehát a Google mindegyik web scrapernek tiltja a search, azaz a keresési találatok által nyújtott adatok begyűjtését. Ez számomra is fontos, ugyanis volt olyan kód a szakdolgozatomban, amely a Google keresőmotorját használta volna, az egyszerűbb adatgyűjtés érdekében. Ezt a Google robots.txt fájljának megismerése után eltávolítottam, hogy a kódom minden legális elvárásnak megfeleljen.

4.2.2 Technikai problémák

A web scraping módszernek számos technikai problémája is van.

Anti-scraping

Nem minden weboldal tulajdonosa szeretné lehetővé tenni a web scrapinget, ezért különböző, úgynevezett anti-scraping módszereket alkalmazhatnak a web scraperek ellen. Az egyik ilyen módszer a robots.txt fájl, de ez csak a „legális betolakodókat" tartja távol, ugyanis ez a fájl becsületre épül, és az illegális botok figyelmen kívül fogják hagyni. További anti-scraping módszerek lehetnek például: a HTTP fejlécek figyelése, bejelentkezéshez kötött adatelérés, IP cím figyelés, CAPTCHA-k használata, felhasználói viselkedéselemzés, stb. [20]

Gyakori weboldal struktúra változás

A különböző weboldalak folyamatos fejlesztés alatt állnak, hogy gyorsabbak, hatékonyabbak és szebbek legyenek, valamint hogy ellenállóbbak legyenek a web scraping ellen. Az ilyen fejlesztések gyakran járhatnak struktúrán belüli változással is, amely lehet kisebb (például egy gomb áthelyezése), vagy drasztikusabb, egész oldalas változtatás is. Már akár egy kisebb változtatás is könnyen üzemképtelenné teheti a web scrapereinket, mivel mindig egy specifikus struktúrára íródnak. [21]

Komplex weboldalak

Általánosságban az oldalak 50%-ánál könnyű, 30%-ánál közepes, 20%-ánál pedig nehéz web scrapinget alkalmazni. A weboldalak manapság egyre bonyolultabbak, mivel egyre gyakrabban használnak dinamikus összetevőket (például AJAX). [21]

4.2.3 Miért a web scraping?

A másik gyakori módszer a web scraping mellett az API-k (application programming interface, alkalmazásprogramozási felület) használata. A web scraping több technikai és legális problémába is ütközhet, viszont sokkal nagyobb területet fed le, azaz bármilyen weboldalról képes bármilyen adatot begyűjteni. De a legnagyobb előnye, hogy az API-kkal szemben mindig valós idejű adatokat tud beszerezni, mivel az API-k által nyújtott adatok csak periodikusan vannak frissítve. [22]

4.3 A kód

Legelőször mikor nekiláttam a szakdolgozatomnak, akkor a booking.com weboldallal próbálkoztam, azonban pár nap fejtörés után nyílvánvalóvá vált, hogy nem ezt fogom hosszútávon választani. Az egyik fő probléma technikai, ugyanis rengeteg beépített anti-scraping védelemmel rendelkezik. A másik fő probléma pedig legális, ugyanis a booking.com kimondottan tiltja a web scrapinget, a weboldal szabályzatában: a booking.com vagy licencadóinak írásos engedélye nélkül tilos a platformon található dolgokat figyelni, másolni, feltérképezni, letölteni, reprodukálni vagy egyéb módon kereskedelmi célokra felhasználni [23]. Emiatt keresnem kellett egy másik weboldalt.

Minden kód a szallaskeres.hu weboldalhoz íródott. Azért erre esett a választás, mivel nem rendelkezik anti-scraping védelemmel (még csak robots.txt fájllal sem), a weboldal struktúrája rendkívül egyszerű és átlátható, valamint (és ez teljesen személyes vélemény) nem tűnik olyan weboldalnak, amit gyakran változtatnának.

Az adatbegyűjtéshez tartozó kódot két különböző fájlra bontottam: scraper (amely tartalmazza az adatbegyűjtéshez tartozó fő függvényt, a main_scraper-t) és scraper_functions (amelyben az általános, web scrapinggel kapcsolatos függvények találhatók). Emellett van egy variables fájl, ami tartalmazza a gyakran használt, projektszintű változókat.

A szállodák kigyűjtése

```
page = requests.get(url, headers=variables.HEADERS)
soup = BeautifulSoup(page.content, "html.parser")
set = soup.find_all("a", attrs={"class": "head_h3_link"})
hotel_links = [l.get("href") for l in set]
hotel_params = [p.get("data-urlparams") for p in set]
all_hotel_links = []
for i in range(len(hotel_links)):
    all_hotel_links.append(hotel_links[i] + hotel_params[i])
```

4.2. ábra – scraper fájl kódrészlet, szállodák kigyűjtése. (Forrás: Saját szerkesztésű)

Első lépésként kigyűjtjük a szallaskeres.hu-ról a felhasználó paramétereinek megfelelő szállodákat. A weboldalt a request könyvtár segítségével érjük el. A weboldal URL címét

(és a hozzátartozó paramétereket) az url változó tartalmazza, amelyet a függvény meghívásakor adunk meg. Az azonosíthatóság érdekében meg kell adnunk a megfelelő HTML fejlécet, amelyet a variables fájl tartalmaz:

```
HEADERS = {
    "User-Agent": "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/62.0.3202.94 Safari/537.36",
    "Accept-Encoding": "gzip, deflate, br",
    "Accept": "text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,image/apng,*/*;q=0.8",
    "Connection": "keep-alive",
    "Accept-Language": "en-US,en;q=0.9,lt;q=0.8,et;q=0.7,de;q=0.6",
}
```

4.3. ábra – variables fájl kódrészlet, HTML fejléc. (Forrás: Saját szerkesztésű)

Ezután elkészítjük a soup változónkat, a BeautifulSoup könyvtár segítségével. Ez a weboldal tartalmát fogja beszerezni és eltárolni. Miután megvan a tartalom, kiválasztjuk belőle az összes <a> HTML elemet, amely a „head_h3_link” osztályhoz tartozik. Ezekből kinyerjük a „href” és a „data-urlparams” értékeket, hogy megkapjuk a szállodák linkjeit, és az url paramétereket.

```
<a class="head_h3_link" id="name_4832" href="https:
//szallaskeres.hu/hotel-platan%2A%2A%2Aszekesfeherv
ar?&t=1711361694" data-urlparams="?&t=1711361694"
target="_blank" title="Hotel Platán*** adatlap »">
```

4.4. ábra – szallaskeres.hu kódrészlet, egy szálloda adatai. (Forrás: szallaskeres.hu)

Végül az url címeket és a hozzájuk tartozó paramétereket egyesítjük, és a kapott linkeket eltároljuk egy listában.

Böngésző indítása

```
available_hotel_links = []
op = webdriver.ChromeOptions()
op.add_argument('--headless=new')
driver = webdriver.Chrome(options=op)
driver.maximize_window()
```

4.5. ábra – scraper fájl kódrészlet, böngésző indítása. (Forrás: Saját szerkesztésű)

Sajnos nem minden szálloda megfelelő, amit előhoz a weboldal. Előfordulhat, hogy a tartózkodásunk időtartama alatt már foglalt a szállás, illetve az is, hogy az adott szállás-hely jelenleg nem üzemel. Ezért létrehozunk egy újabb listát, amely az elérhető szállodák linkjeit fogja tartalmazni. A tartózkodási napok foglaltságának ellenőrzéséhez a Selenium könyvtárat fogjuk használni, melynek segítségével „nyitunk” egy headless, másnéven háttérben futó Chrome böngészőt.

Szállodák ellenőrzése

```
for i in range(len(all_hotel_links)):
    given_link = all_hotel_links[i]
    in_service = scraper_functions.in_service(given_link)
    if (in_service):
        driver.get(given_link)
        available = scraper_functions.check_availability(
            checkin, checkout, driver)
        if (available):
            available_hotel_links.append(given_link)
driver.close()
```

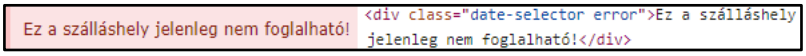
4.6. ábra – scraper fájl kódrészlet, szállodák ellenőrzése. (Forrás: Saját szerkesztésű)

Az összes szálloda leellenőrzéséhez egy ciklus segítségével végig kell mennünk mindegyik linken. Az első teszt az, hogy éppen üzemel-e az adott szálloda. Ezt a scraper_functions fájl in_service függvénye fogja leellenőrizni.

```
def in_service(hotel):
    page = requests.get(hotel, headers=variables.HEADERS)
    soup = BeautifulSoup(page.content, "html.parser")
    set = soup.find_all(
        "div", attrs={"class": "date-selector error"})
    return len(set) == 0
```

4.7. ábra – scraper_functions fájl kódrészlet, üzemelés ellenőrzése. (Forrás: Saját szerkesztésű)

Az in_service függvény megnézi, hogy a paraméterként kapott linkhez tartozó szálloda tartalmazza-e azt a HTML elemet, amely a „date-selector error” osztályba tartozik. Ha létezik ez az elem, akkor a változó hossza nagyobb, mint nulla, így a visszatérő érték hamis lesz, azaz a szálloda nem üzemel jelenleg. Egyéb esetben pedig igaz lesz, azaz a szálloda üzemel. Ez a weboldalon így nézne ki:



4.8. ábra – szallaskeres.hu kódrészlet, egy nem foglalható szálloda. (Forrás: szallaskeres.hu)

Tehát ha ez a hibaüzenet létezik, akkor a szálloda nem üzemel.

A második tesztben azt vizsgáljuk, hogy az adott időintervallumon, a becsekkolás és a kicsekkolás közötti napokon foglalt-e bármelyik nap. Ezt a scraper_functions fájl check_availability függvénye fogja leellenőrizni.

```
def check_availability(checkin, checkout, driver):
    available = True
    checkin_date = datetime.datetime.strptime(checkin, '%Y-%m-%d')
    checkout_date = datetime.datetime.strptime(checkout, '%Y-%m-%d')
    stay_days = (checkout_date-checkin_date).days
    for i in range(stay_days):
        given_date = convert_date_to_unix_string(
            checkin_date + datetime.timedelta(days=i))
        given_day = driver.find_element(
            By.XPATH, f"//a[@data-time='{given_date}']")
        if str(given_day.get_attribute("class")).strip() == "day-item is-booked":
            available = False
    return available
```

4.9. ábra – scraper_functions fájl kódrészlet, foglaltság ellenőrzése. (Forrás: Saját szerkesztésű)

A függvény elején azt feltételezzük, hogy az adott szállás elérhető, tehát az available változó értéke True. A kicsekkolás dátumából kivonva a becsekkolás dátumát megkapjuk a tartózkodási napok számát, melynek segítségével indítunk egy ciklust. Itt egy érdekességet figyelhetünk meg a szallaskeres.hu weboldallal kapcsolatban. Egyrészt a dátumkiválasztó HTML elem dinamikus, így a BeautifulSoup könyvtár nem lenne alkalmas az adatbegyűjtésre, és ezért használjuk a Selenium könyvtárat. Másrészt minden napnak saját „data-time” tulajdonsága van, amely az adott dátum UNIX-időben, miliszekundum nagyságrendben. Az UNIX-idő az 1970. január 1. óta eltelt másodpercek összessége [24].

```
<a href="#" class="day-item" data-time="17114940000
00">27</a>
```

4.10. ábra – szallaskeres.hu kódrészlet, egy dátum tulajdonságai. (Forrás: szallaskeres.hu)

Az ábrán látható dátum 2024. március 27., ami UNIX-időben 1711494000, a „data-time” tulajdonság értéke pedig 1711494000000. A kettő között három nulla a különbség, ami a korábban említett miliszekundum nagyságrend miatt van. Ez azt jelenti, hogy ezzel a tulajdonsággal bármelyik dátumot megtalálhatjuk. Erre a módszerre épül a check_availability függvény is, amely az adott dátumokat UNIX-időre konvertálja. Ehhez a scraper_functions convert_date_to_unix_string függvényét fogja használni.

```
def convert_date_to_unix_string(date):
    date_with_time = datetime.datetime(
        date.year, date.month, date.day)
    date_in_unix = datetime.datetime.timestamp(date_with_time)
    unix_string = str(date_in_unix).split('.')[0] + "000"
    return unix_string
```

4.11. ábra – scraper_functions fájl kódrészlet, UNIX konvertáló. (Forrás: Saját szerkesztésű)

A függvény a kapott dátumot dátumidő formátumúra alakítja, mivel az UNIX-időnél másodperc pontossággal kell megadni egy dátumot, utána pedig UNIX-időre. A kapott UNIX-időt a szám puszta mérete miatt float típusban kell eltárolni, amely .00-ra végződik, ha string típusúra alakítjuk. Az utolsó előtti sor ezt eltávolítja, és három nullát fűz a végére, hogy megfelelő nagyságrendben legyen a számunk. Ezáltal megkapjuk az adott dátum „data-time" tulajdonságát. A check_availability függvény megkeresi azt a HTML elemet, amelynek a „data-time" tulajdonsága megegyezik a kapott számmal, és megnézi annak az osztályát. Ha az osztály „day-item is-booked", akkor azon a napon foglalt a szálláshely, és az available változót hamisra állítjuk. Mivel egy ciklus segítségével végigmegyünk az összes dátumon (becsekkolástól a kicsekkolásig), ezért ha bármelyik foglalt lenne, akkor az elérhetőség hamis lesz.

Adatbeszerzés

Az adatbeszerzést a scraper_functions data_scraper függvénye végzi. Első lépésként létrehozunk egy listát minden olyan tulajdonságnak, amelyet ki szeretnénk gyűjteni.

```
all_names = []
all_address = []
all_ratings = []
all_nrofratings = []
all_price = []
all_otherattributes = []
all_szep = []
all_tipus = []
all_capacity = []
all_responsetime = []
```

4.12. ábra – data_scraper függvény kódrészlet, tulajdonság listák. (Forrás: Saját szerkesztésű)

Az ábrán látható, hogy a szállodák nevét, címét, értékelését, értékelések számát, árát, jellemzőit, SZÉP kártya elfogadását, típusát, férőhelyét és válaszidejét nézzük.

```
for i in range(len(hotel_links)):
    page = requests.get(
        hotel_links[i], headers=variables.HEADERS)
    soup = BeautifulSoup(page.content, "html.parser")

    set = soup.find_all("h1", attrs={"itemprop": "name"})
    names = [l.get_text() for l in set]
    all_names.append(names[0])

    set = soup.find_all("span", attrs={"itemprop": "postalCode"})
    code = [l.get_text() for l in set]
    set = soup.find_all("span", attrs={"itemprop": "addressLocality"})
    address = [l.get_text() for l in set]
    all_address.append(code[0] + " " + address[0])
```

4.13. ábra – data_scraper függvény kódrészlet, alapvető tulajdonságok kigyűjtése. (Forrás: Saját szerkesztésű)

Következő lépésként indítunk egy ciklust, amely végigmegy az elérhető és foglalható szállodák linkjein. Az adott szállodához tartozó linkkel készítünk egy BeautifulSoup változót, amely segítségével elérhetjük a szállodához tartozó oldal adatait. Az alapvető tulajdonságokat, mint a név, irányítószám és cím, könnyű kigyűjteni: a megfelelő „itemprop” tulajdonságokkal kikeressük őket a BeautifulSoup változónkból, és a hozzá tartozó szöveget (tehát az adott szálloda nevét, irányítószámát és címét) hozzáadjuk a megfelelő listához. Az egyszerűség kedvéért az irányítószámnak és a címnek nincs külön listája, ezek összefűzve, szóközzel elválasztva kerülnek ugyan abba a listába.

```
set = soup.find_all("span", attrs={"itemprop": "ratingValue"})
ratings = [l.get_text() for l in set]
if len(ratings) > 0:
    all_ratings.append(str(ratings[0]))
else:
    all_ratings.append("0")

set = soup.find_all("a", attrs={"href": "#velemenyek"})
nrofratings = [l.get_text() for l in set]
if len(nrofratings) > 0:
    all_nrofratings.append(str(nrofratings[0]).split(' ')[0])
else:
    all_nrofratings.append("0")
```

4.14. ábra – data_scraper függvény kódrészlet, az értékelések és az értékelések számának kigyűjtése. (Forrás: Saját szerkesztésű)

Az értékelés olyan elemben található, amelynek "itemprop" tulajdonsága "ratingValue", az értékelések száma pedig olyan <a> elemben található, amelynek "href" tulajdonsága "#velemenyek". Az értékelés és az értékelések száma már érdekesebb, mivel előfordulhat, hogy egy szállodát még nem értékelt senki. Ezért a kigyűjtött, BeautifulSoup változóhoz tartozó szövegeknél meg kell győződnünk arról, hogy egyáltalán van szöveg, vagyis értékelés. Ha nincs, azaz a szövegek hossza 0, akkor az értékelésekhez és az értékelések számához tartozó listákhoz 0-t fűzünk eredményül. Egyéb esetben pedig az értékelésnél szimplán magát az értékelést, az értékelések számánál pedig az „"x értékelés alapján" karakterláncból kinyerjük az x-et, és azt fűzzük hozzá a megfelelő listánkhoz.

```
set = soup.find_all("span", attrs={"id": "min_price"})
price = [l.get_text() for l in set]
if len(price) > 0:
    actual_price = str(price[0]).split(' ')
    if len(actual_price) > 1:
        actual_price = str(price[0]).split(' ')
        all_price.append(actual_price[1] + actual_price[2])
    else:
        all_price.append("?")
else:
    all_price.append("?")
```

4.15. ábra – data_scraper függvény kódrészlet, az árak kigyűjtése. (Forrás: Saját szerkesztésű)

Az ár elemekben található, melyek "min_price" azonosítójúak. Az ár abból a szempontból érdekes, hogy hiányozhat, vagy akár lehet alkuképes is. A hiányzást ugyan azzal a módszerrel teszteljük, mint az értékeléseket: ha a visszakapot szöveg (ár) hossza 0, akkor kérdőjelet fűzünk az ár listájához. Ha nem nulla, akkor szóköz szerint különválasztjuk a karakterláncunkat. Ha a különválasztott karakterláncok száma több, mint 1, akkor tudjuk, hogy a karakterlánc nem az „Alkuképes” szó volt. Ez azért működik, mivel az árak úgy vannak eltárolva, hogy „akár x Ft/éj”, ami tele van szóközökkel. Emellett a számoknál az ezres nagyságrend után van egy szóköz, tehát a 13000 úgy van ábrázolva, hogy 13 000 (például „akár 13 000 Ft/éj”). Így nekünk ennek a karakterláncnak a két, szóközzel elválasztott szám részét kell összefűzve hozzáadnunk az ár listájához. Egyéb esetben pedig szintén kérdőjelet fűzünk a listához.

```
set = soup.find_all("div", attrs={"class": "jelleg"})
attrlist = ""
other_attributes = [l.get_text() for l in set]
for i in range(len(other_attributes)):
    if ('(' in other_attributes[i]):
        other_attributes[i] = other_attributes[i].split('(')[0].strip()
    attrlist = attrlist + other_attributes[i] + ", "
attrlist = attrlist.strip().removesuffix(',')
if len(attrlist) == 0:
    all_otherattributes.append("?")
else:
    all_otherattributes.append(attrlist.strip())
```

4.16. ábra – data_scraper függvény kódrészlet, a szállás jellegének kigyűjtése. (Forrás: Saját szerkesztésű)

Az adott szálláshoz tartozó jellegek elemekben találhatók, és "jelleg" osztállyal rendelkeznek. Mivel több ilyen van, ezért több szöveg fog visszatérni. Minden ilyen szövegen ciklus segítségével végigmegyünk, és ha bármelyikben lenne zárójeles rész (például Vízparti (max. 1 km)), akkor a zárójel előtti részt vesszük csak figyelembe, és azt fűzzük hozzá az attrlist változóhoz. Egyéb esetben nem történik változás, és az eredeti értéket fűzzük hozzá. A ciklus után eltávolítjuk az attrlist végén található extra vesszőt. Előfordulhat, hogy egy szállásnak nem adnak meg jelleget. Ilyenkor az attrlist hossza nulla, és kérdőjelet fűzünk hozzá a jelleg listájához. Egyéb esetben pedig magát a változót, amely tartalmazni fogja az összes jelleget, vesszővel elválasztva.

```
set = soup.find_all("div", attrs={"class": "main_prop_tuls"})
all_main_attributes = attribute_unboxing(set)
all_szep.append(all_main_attributes[0])
all_tipus.append(all_main_attributes[1])
all_capacity.append(all_main_attributes[2])
all_responsetime.append(all_main_attributes[3])
```

4.17. ábra – data_scraper függvény kódrészlet, egyéb tulajdonságok kigyűjtése. (Forrás: Saját szerkesztésű)

Az egyéb tulajdonságok (SZÉP kártya elfogadás, típus, férőhely és válaszidő) begyűjtése hosszadalmasnak bizonyult, ezért az átláthatóság kedvéért egy attribute_unboxing nevű függvénybe tettem ezen tulajdonságok begyűjtését.

```
raw = str(set[0])
text = raw.split('</div>')
if (len(text) != 13 and len(text) != 10):
    return ["?", "?", "?", "?"]
```

4.18. ábra – attribute_unboxing függvény kódrészlet, egyéb tulajdonságok kibontása. (Forrás: Saját szerkesztésű)

Az attribute_unboxing függvény azzal kezdődik, hogy a begyűjtött "main_prop_tuls" osztályú elemek záró ( ) részei szerint elválasztjuk a tartalmakat. Ha a kapott elemszám 13, akkor minden egyéb jellemző elérhető, ha pedig 10, akkor csak a válaszidő hiányzik. Egyéb esetben nem éri meg foglalkozni az egyéb jellemzőkkel, ilyenkor egy négy darab kérdőjelből álló listát adunk vissza, amely a hiányzó adatokat reprezentálja.

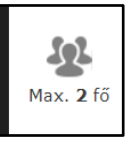
```
try:
    szep_raw = text[2]
    szep_raw = szep_raw.replace("<b>", "")
    szep_raw = szep_raw.replace("</b>", "")
    szep_raw = szep_raw.replace("&amp;", "&")
    szep_raw = szep_raw.replace(" ", " ")
    szep_raw = szep_raw.replace("SZÉP Kártya:", "")
except:
    szep_raw = "?"
```

4.19. ábra – attribute_unboxing függvény kódrészlet, SZÉP kártya kibontása. (Forrás: Saját szerkesztésű)

A típust leszámítva, minden egyéb tulajdonság egy try-except blokkban van, mivel ez a rész nagyon érzékeny a hibákra. Ezért ha nem sikerülne az adatbegyűjtés, akkor az adott tulajdonság helyett egy kérdőjel szerepelne.

A SZÉP kártyához tartozó adatokból eltávolítjuk a fölösleges karakterláncokat, majd eltároljuk a tisztított értéket egy szep_raw nevű változóban.

```
try:
    nr_raw = text[7].split("<b>")[1].split('</b>')[0]
except:
    nr_raw = "?"
```



4.20. ábra – attribute_unboxing függvény kódrészlet, férőhely kibontása. (Forrás: https://szallaskeres.hu/ és saját készítésű)

A férőhely száma "Max. **x** fő" formátumban van megjelenítve a weboldalon, ahogy az ábrán is látható. Ez <b> HTML elemeket jelent, amelyek között található a számunkra érdekes x értéke, ezért ezt fogjuk kinyerni az ábrán látható kód segítségével, és egy nr_raw nevű változóban fogjuk eltárolni.

```
answer_raw = "?"
if (len(text) == 13):
    try:
        answer_raw = text[10].split("<b>")[1].split(
            '</b>')[0] + text[10].split("<b>")[1].split('</b>')[1]
    except:
        answer_raw = "?"
        print(f"Answer time error: {text[10]}")
return [szep_raw.strip(), text[4].strip(), nr_raw.strip(), answer_raw.strip()]
```

4.21. ábra – attribute_unboxing függvény kódrészlet, válaszidő kibontása. (Forrás: Saját szerkesztésű)

A válaszidő alapértéke kérdőjel, és csak abban az esetben fog más értéket kapni, ha az elválasztott tartalmak elemszáma 13. Ez a weboldalon hasonló formátumban van eltárolva, mint a férőhelyek száma: "Visszaigazolás: **x** y", ahol x egy szám, y pedig a nagyságrend (például perc, óra). Annyival egészül ki most a feladatunk, hogy nem csak maga a szám kell, hanem a szám nagyságrendje is, ezért azt is kinyerjük az ábrán látható kód segítségével. A kapott eredményt egy answer_raw nevű változóban tároljuk el.

A függvény végén pedig a kinyert értékeket egy lista formájában visszaadjuk, és a visszatérő elemeket a saját listájukhoz fogjuk hozzáadni.

```
hotels = pd.DataFrame({'Név': all_names, 'Cím': all_address, 'Link': hotel_links,
                       'Price': all_price, 'Rating': all_ratings, 'Number of Ratings': all_nrofratings,
                       'SZÉP': all_szep, 'Típus': all_tipus, 'Férőhely': all_capacity,
                       'Válaszidő': all_responsetime, 'Jelleg': all_otherattributes})
hotels = hotels.drop(hotels[hotels["Price"] == "?"].index)
hotels = hotels.drop(hotels[hotels["Jelleg"] == "?"].index)
hotels = hotels.drop(hotels[hotels["Típus"] == "?"].index)
return hotels
```

4.22. ábra – data_scraper függvény kódrészlet, a DataFrame létrehozása. (Forrás: Saját szerkesztésű)

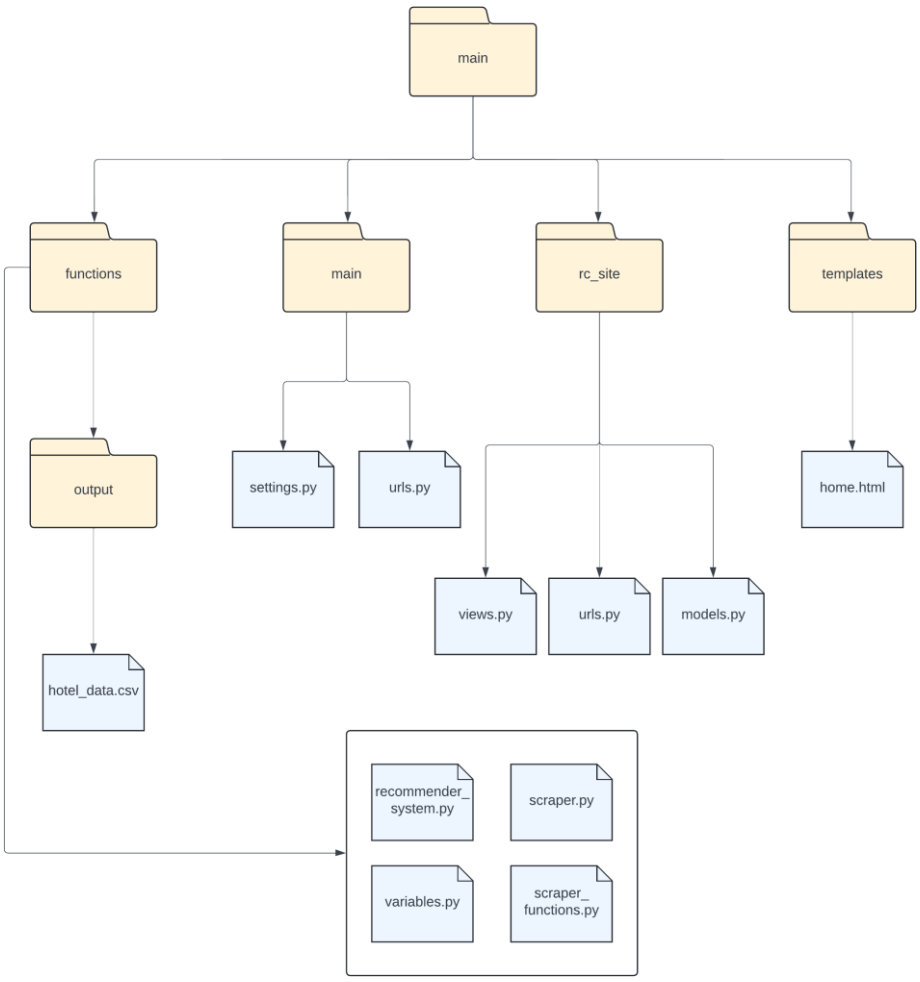
Az összes eddigi tulajdonság, amelyeket összegyűjtöttünk a különböző listáinkba, most kerülnek felhasználásra: létrehozzuk velük a hotels nevű DataFrame változót. A kész változóból eltávolítjuk azokat a sorokat, amelyeknek az ár, jelleg vagy típus tulajdonsága hiányzik (azaz értéke kérdőjel).

```
hotels = scraper_functions.data_scraper(available_hotel_links)
hotels.to_csv((variables.data_path +
               'functions\\output\\hotel_data.csv'), index=False)
```

```
data_path = str(Path(__file__).resolve().parent.parent) + "\\"
```

4.23. ábra – scraper és variables fájl kódrészlet, szállodai adatok exportálása. (Forrás: Saját szerkesztésű)

A data_scraper függvény visszatér a DataFrame típusban tárolt szállodai adatokkal, amelyeket exportálunk, és később beolvasunk. Erre azért van szükség, mivel a begyűjtött adatok nagy része rossz formátumban van (mindegyik string típusú). Beolvasáskor ezek megfelelő formátumúra lesznek konvertálva a pandas könyvtárban található read_csv függvény által, így nekünk ezzel nem kell foglalkoznunk. Az elérési út gyökere vagyis a data_path változó a variables fájlban található, és az alábbi képen látható main mappára mutat. Ezen belül az output mappába lesznek elmentve az adatok.



4.24. ábra – A projekt mappaszerkezete. (Forrás: Saját készítésű, LucidChart segítségével.)

5. AJÁNLÓRENDSZER

Az előző fejezetben bemutatásra került a functions mappa majdnem összes Python fájlja (scraper, scraper_functions, variables), ezek a web scrapinghez tartoztak. Ebben a fejezetben bemutatásra kerül a mappa utolsó Python fájlja, a recommender_system, mely az ajánlórendszerrel kapcsolatos számításokat tartalmazza. A tartalma egy recommender nevű függvénybe van ágyazva, hogy könnyebb legyen a weboldalon végrehajtani az ajánlást.

Véleményem szerint a szállodáknál a tartalom szerinti szűrésnek sokkal több értelme van, mint a kollaboratív szűrésnek (valamint egyszerűbb is megvalósítani). Ezért az ajánlórendszerem a felhasználó korábban látogatott szállodáinak tulajdonságaira alapul, tehát tartalom szerinti szűrést alkalmaz.

```
def recommender(been_to_hotels):
    all_hotels = pd.read_csv(variables.data_path +
                             'functions\\output\\hotel_data.csv')
    already_visited_hotels = scraper_functions.data_scraper(been_to_hotels)
    already_visited_hotels['Price'] = pd.to_numeric(
        already_visited_hotels['Price'])
    already_visited_hotels['Rating'] = pd.to_numeric(
        already_visited_hotels['Rating'])
    already_visited_hotels['Number of Ratings'] = pd.to_numeric(
        already_visited_hotels['Number of Ratings'])
```

5.1. ábra – recommender_system fájl kódrészlet, előkészítés. (Forrás: Saját szerkesztésű)

Első lépésként beolvassuk az előző fejezetben összegyűjtött szállodák adatait, a melyet az output mappában található hotel_data.csv fájl tartalmaz. A beolvasásnak köszönhetően megfelelő formátumúra lettek konvertálva az adatok. Ezt követően el kell végeznünk az adatbegyűjtést a felhasználó korábban már látogatott szállodáin is, az előző fejezetben bemutatott data_scraper függvény segítségével, amely a scraper_functions fájlban található. Ezen szállodák listája a függvény meghívásakor paraméterként át lesz adva. A begyűjtött adatok ár, értékelés és értékelések száma oszlopait szám formátumúra kell konvertálnunk, a pandas to_numeric függvényének segítségével.

Értékelés

```
hotel_rating_scores = []
for i in range(len(all_hotels)):
    given_rating = all_hotels['Rating'][i]
    given_nr = all_hotels['Number of Ratings'][i]
    rating_score = round(
        (given_rating**2 * given_nr / len(all_hotels))/100, 2)
    hotel_rating_scores.append(rating_score)
all_hotels['Rating score'] = hotel_rating_scores
```

5.2. ábra – recommender_system fájl kódrészlet, az értékelés-pontszám kiszámítása. (Forrás: Saját szerkesztésű)

A szállodák a tulajdonságaik szerint lesznek értékelve, egy végső pontszám segítségével, amely több különböző pontszám alapján lesz kiszámolva. Az egyik ilyen pontszám az értékelés és az értékelések száma alapján kiszámított hotel_rating_score lesz. Vesszük minden egyes szálloda értékelését és értékeléseinek számát, majd kiszámoljuk velük a pontszámot. A pontszám az értékelés négyzetének, és az értékelések számának a szorzata, amelyet elosztunk az összes szálloda (amelyet az előző fejezetben begyűjtöttünk) darabszámával. A végeredmény két tizedesjegyre lesz kerekítve, és hozzáfűzzük a hotel_rating_scores listához. A ciklus végén ezt a listát hozzáfűzzük a DataFrame-hez.

Ár

```
hotel_price_score = all_hotels['Price'].values.flatten().tolist()
for i in range(len(hotel_price_score)):
    hotel_price_score[i] = hotel_price_score[i]/10000
```

5.3. ábra – recommender_system fájl kódrészlet, az ár-pontszám kiszámítása. (Forrás: Saját szerkesztésű)

A következő ilyen tulajdonság az ár-pontszám lesz, amely valójában maga az ár tízezred része. A kiszámításhoz szükségünk lesz az árakra, amelyeket a DataFrame-ből kinyerünk, és a hote_price_score nevű listába tárolunk. Végül egy ciklus segítségével minden árat felülírunk a tízezredével.

Tulajdonságok

```
attribute_list = []
for i in range(len(already_visited_hotels)):
    raw = already_visited_hotels['Jelleg'][i]
    given_attributes = raw.split(',')
    for j in range(len(given_attributes)):
        if given_attributes[j].strip() not in attribute_list:
            attribute_list.append(given_attributes[j].strip())
```

5.4. ábra – recommender_system fájl kódrészlet, a tulajdonságok kigyűjtése. (Forrás: Saját szerkesztésű)

Ahhoz, hogy meg tudjuk határozni a felhasználó számára fontos tulajdonságokat, először meg kell néznünk, hogy egyáltalán milyen tulajdonságok fordultak elő. Egy ciklus segítségével végigmegyünk a felhasználó által korábban látogatott szállodák listáján, és minden iterációban az adott szálloda "jelleg" értékét vesszők szerint feldaraboljuk (mivel adatbegyűjtéskor vesszővel elválasztott formátumúra hoztuk), és az így kapott elemeken végigmegyünk. Ha bármelyik új (azaz még nincs benne a listában), akkor hozzáadjuk az attribute_list listához, egyéb esetben pedig átugorjuk. A külső ciklus végén az attribute_list tartalmazni fogja a különböző tulajdonságokat, amelyek előfordultak a felhasználó korábban látogatott szállodáinál, ismétlődés nélkül.

```
attribute_scores = {}
for i in range(len(attribute_list)):
    attribute_scores[f'{attribute_list[i]}'] = 0
for i in range(len(already_visited_hotels)):
    raw = already_visited_hotels['Jelleg'][i]
    given_attributes = raw.split(',')
    for j in range(len(given_attributes)):
        try:
            attribute_scores[f'{given_attributes[j].strip()}'] += 1
        except:
            pass
sorted_attribute_scores = dict(sorted(
    attribute_scores.items(), key=lambda x: x[1], reverse=True))
```

5.5. ábra – recommender_system fájl kódrészlet, a tulajdonságok előfordulásai. (Forrás: Saját szerkesztésű)

Miután meghatároztuk az előforduló tulajdonságokat, most meg kell számolnunk, hogy melyik hányszor fordult elő, hogy egy fontossági sorrendet tudjunk köztük felállítani. Ennek a legegyszerűbb módja egy dictionary használata, amely egy kulcsérték párokat tartalmazó adattípus. A kulcs itt a tulajdonság neve, az érték pedig az előfordulásának száma lesz. Egy ciklus segítségével minden tulajdonsághoz nullát rendelünk értékül, hogy

találat esetén növelni tudjuk majd eggyel. Egy másik ciklus segítségével végigmegyünk a felhasználó korábban látogatott szállodáin, és ismételten feldaraboljuk a "jelleg" tulajdonságot, vesszők szerint. Végigmegyünk a kapott értékeken (és egyben kulcsokon), és a hozzájuk tartozó dictionary értéket növeljük eggyel. A ciklus végén az attribute_scores dictionary tartalmazni fogja az összes tulajdonságot, az előfordulások számával együtt. Végül ezt a dictionary-t az értékei szerint csökkenő sorrendbe rendezzük.

```
important_attributes = []
for i in range(len(sorted_attribute_scores)):
    if (i == 3):
        break
    else:
        important_attributes.append(
            list(sorted_attribute_scores.keys())[i])
```

5.6. ábra – recommender_system fájl kódrészlet, a legfontosabb tulajdonságok. (Forrás: Saját szerkesztésű)

Következő lépés a fontos tulajdonságok kigyűjtése, ami a dictionary első három kulcsa lesz (vagy ha esetleg kevesebb lenne, akkor az első x kulcs). Egy ciklus segítségével végigmegyünk a dictionary elemein, és minden iterációban leellenőrizzük, hogy az index értéke elérte-e a hármat. Ha igen, akkor kilépünk a ciklusból, egyéb esetben pedig az adott kulcsérték párból a kulcsot, tehát a tulajdonságot hozzáfűzzük az important_attributes listához. Így a ciklus végén a felhasználó számára fontos tulajdonságok az important_attributes listában lesznek, fontossági sorrendben.

```
hotel_attribute_scores = []
score = 0
for i in range(len(all_hotels)):
    score = 0
    given_attributes = all_hotels['Jelleg'][i]
    for j in range(len(important_attributes)):
        if important_attributes[j] in given_attributes:
            score += (len(important_attributes) - j)
    hotel_attribute_scores.append(score)
all_hotels['Attribute score'] = hotel_attribute_scores
```

5.7. ábra – recommender_system fájl kódrészlet, a tulajdonság-pontszám kiszámítása. (Forrás: Saját szerkesztésű)

Az előkészítések után kiszámolhatjuk minden szálloda tulajdonság-pontszámát, a fontos tulajdonságok alapján. Egy külső ciklus segítségével végigmegyünk az összes szálloda jellegén, egy belső ciklus segítségével pedig a fontos tulajdonságok listáján. Ha az adott

fontos tulajdonság szerepel az adott szálloda tulajdonságai között, akkor x pontot kap. Mivel a fontos tulajdonságok listája sorrendben van (a legfontosabb van legelöl), ezért ha az első fontos tulajdonság szerepel az adott szálloda tulajdonságai között, az több pontot fog érni, mintha a harmadik fontos tulajdonság szerepelne. Ezért lesz a kapott pontszám a fontos tulajdonságok számának és a belső ciklus adott iteráció-indexének a különbsége. *Például ha három fontos tulajdonság van, és feltesszük, hogy mindegyik szerepel az adott szálloda tulajdonságai között, akkor az első 3 – 0 = 3, a második 3 – 1 = 2, a harmadik pedig 3 – 2 = 1 pontot fog érni, ami összesen 3 + 2 + 1 = 6 pont. Ha csak a második és a harmadik szerepel, az csak 3 - 1 + 3 - 2 = 2 + 1 = 3 pont.*

A belső ciklus végén a kapott pontot hozzáfűzzük a hotel_attribute_scores listához. Ezt az értéket a belső ciklus futása előtt mindig nullázzuk, hogy ne akkumulálódjanak a pontszámok a szállodák között. A külső ciklus végén pedig hozzáfűzzük a DataFrame-hez a pontokat.

Végső pontszám

```
final_score = []
for i in range(len(all_hotels)):
    final_score.append(
        round((hotel_rating_scores[i] * hotel_attribute_scores[i]) - hotel_price_score[i], 2))
all_hotels['Final score'] = final_score
recommended_hotels = all_hotels[['Név', 'Price', 'Rating', 'Number of Ratings',
                                 'Jelleg', 'Final score', 'Link', 'Cím', 'SZÉP','Típus','Férőhely',
                                 'Válaszidő']].sort_values(by=['Final score'], ascending=False)

return recommended_hotels
```

5.8. ábra – recommender_system fájl kódrészlet, a végső pontszám kiszámítása. (Forrás: Saját szerkesztésű)

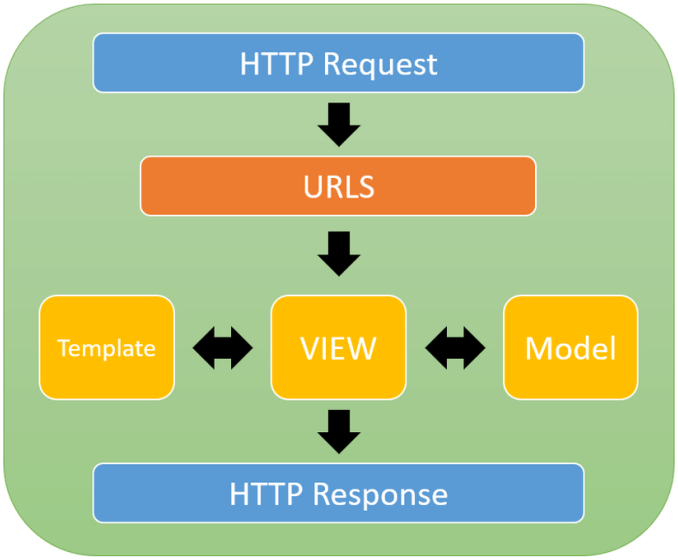
Az utolsó lépés a végső pontszám kiszámítása. Egy ciklus segítségével végigmegyünk az összes szállodán, és az adott szálloda végső pontszámát hozzáfűzzük a final_score listához. A végső pontszám az értékelés-pontszám (hotel_rating_scores) és a tulajdonság-pontszám (hotel_attribute_scores) szorzata, amelyből kivonjuk az ár-pontszámot (hotel_price_score). A kapott érték két tizedesjegyre lesz kerekítve. A final_score listát pedig hozzáfűzzük a DataFrame-hez. Az összes szállodához tartozó DataFrame elemeiből elkészítjük a recommended_hotels DataFrame-et, melynek rekordjai a végső pontszám alapján lesznek csökkenő sorrendbe rendezve. A függvény visszatérő értéke a recommended_hotels DataFrame.

6. WEBOLDAL

Az adatbeszerzés és az ajánlórendszer elkészítése után az utolsó lépés a weboldal elkészítése, amely össze fogja kötni a felhasználót az ajánlórendszerrel.

6.1 Backend

Mivel az egész eddigi projekt Python nyelven készült, ezért a weboldal backend-je, tehát a szerver-oldali része Django keretrendszert használ. A Django egy magas szintű, ingyenes, nyílt forráskódú, Python alapú webkeretrendszer, amely gyors fejlesztést és pragmatikus tervezést tesz lehetővé [25].



6.1. ábra – Egy Django alapú weboldal struktúrája. (Forrás: https://www.researchgate.net/figure/Structure-of-Django_fig5_339343963)

URL-ek (webcímek)

A weboldalhoz tartozó URL-ek (uniform resource locator, magyarul egységes erőforrás-helymeghatározó (avagy webcím), továbbiakban URL) az urls fájlban találhatók. A fájl célja az, hogy leképezze az URL útvonalakat a megfelelő Python függvényekhez.

```
urlpatterns = [
    path('', views.home, name='home'),
    path('recommend/', views.recommend, name='recommend'),
    path('home/', views.home, name='home'),
]
```

6.2. ábra – urls fájl kódrészlet, a weboldal URL-jei. (Forrás: Saját szerkesztésű)

View-k (nézetek)

Minden view két dologért felelős: vagy visszaad egy HttpResponse objektumot, amely tartalmazza a kért oldal tartalmát, vagy kivételt dob (pl. 404). Általában egy view beszerzi a paramétereinek megfelelő adatokat, betölti az adott sablont, majd megjeleníti a sablont a kapott adatokkal. [26]

```
def recommend(request):
    if request.method == 'POST':
        hotels = request.POST.get('hotels', '')
        city = request.POST.get('city', '')
        checkin = request.POST.get('checkin', '')
        checkout = request.POST.get('checkout', '')
        adults = request.POST.get('adults', '')
        children = request.POST.get('children', '')
        search_query = f"""https://szallaskeres.hu/{city}-szállás?erk={checkin}
                           &tav={checkout}&szemelyek={adults}&gyerekek={children}"""

        scraper.main_scraper(search_query, checkin, checkout)
        links = hotels.split(',')
        recommended_hotels = recommender.recommender(links)
    upload_data_to_model(recommended_hotels)
    hotels_all = Recommended_Hotels.objects.all()
    if (len(hotels_all) == 0):
        return render(request, 'home.html',
                      {'hotels': 'Nem található a megadott paramétereknek megfelelő szállás.'})
    else:
        hotels = [hotels_all[0], hotels_all[1], hotels_all[2]]
        return render(request, 'home.html', {'hotels': hotels})
```

6.3. ábra – views fájl kódrészlet, az ajánlás folyamata. (Forrás: Saját szerkesztésű)

Az első függvény a recommend, amely a weboldalon található „Ajánlj!” gomb megnyomásával fog lefutni. A kitöltött form adatait megkapja, és a megfelelő változókba eltárolja őket. A változók segítségével összeállítjuk a keresési linket, és átadjuk a korábban bemutatott main_scraper függvénynek, a becsekkolás és a kicsekkolás dátumával együtt. Ezután a felhasználó által megadott linkeket átadjuk a korábban bemutatott recommender függvénynek, ami elvégzi az ajánlást, és visszaadja az ajánlott szállodákat. A szállodák adatait feltöltjük a megfelelő modellbe, és ha esetleg nem lenne egy szálloda se (mondjuk ha nemlétező várost adna meg a felhasználó), akkor ezt jelezzük a felhasználónak. Egyéb esetben pedig átadjuk az első három (tehát top 3 legrelevánsabb) szálloda adatait, és megjelenítjük őket a weboldalon, néhány adattal együtt. Ha a lehetőségek közül bármelyik megtetszik a felhasználónak, akkor a szálloda nevén található hiperhivatkozás segítségével könnyen megtekintheti az adott szállodát a szallaskeres.hu weboldalon.

```python
def upload_data_to_model(recommended_hotels):
    Recommended_Hotels.objects.all().delete()
    for i in range(recommended_hotels.shape[0]):
        row = recommended_hotels.iloc[i].values.flatten().tolist()
        szepk = row[8]
        if (szepk == "?" or szepk == "nem"):
            szepk = "Nincs"
        hotel = Recommended_Hotels(
            name=row[0],
            price=row[1],
            rating=row[2],
            nrofratings=row[3],
            attributes=row[4],
            link=row[6],
            address=row[7],
            szep=szepk,
            type=row[9],
            capacity=row[10],
            responsetime=row[11]
        )
        hotel.save()
```

6.4. ábra – views fájl kódrészlet, a modellfeltöltés folyamata. (Forrás: Saját szerkesztésű)

A második függvény az upload_data_to_model, amely feltölti adatokkal a modellünket. Az adatok feltöltése előtt a modellünket ki kell üríteni, hogy a régi adatok ne legyenek benne. Egy ciklus segítségével végigmegyünk az ajánlott szállodákon, és listává alakítjuk az adott szállodák sorait. Ha a SZÉP kártya hiányozna (tehát értéke kérdőjel vagy „nem” lenne), akkor a szépség kedvéért „Nincs”-re állítjuk. Ezek után pedig feltöltjük a modellünket a szállodák adataival, és elmentjük a változtatásokat.

Modellek (adatbázis)

```python
class Recommended_Hotels(models.Model):
    name = models.TextField()
    price = models.IntegerField()
    rating = models.FloatField()
    nrofratings = models.IntegerField()
    attributes = models.TextField()
    link = models.TextField()
    address = models.TextField(null=True, blank=True)
    szep = models.TextField(null=True, blank=True)
    type = models.TextField(null=True, blank=True)
    capacity = models.TextField(null=True, blank=True)
    responsetime = models.TextField(null=True, blank=True)
```

6.5. ábra – models fájl kódrészlet, a modellfeltöltés folyamata. (Forrás: Saját szerkesztésű)

A modell egy objektum-relációs leképző, amelyben adatbázis-struktúrát írhatunk le Python kódban [27]. A modellen belül meg lehet szabni, hogy milyen típusú adatokat várunk benne – pont, mint egy relációs-adatbázis táblázatában. Az ábrán látható modellben főleg szövegesek a különböző tulajdonságok, de előfordul egész és tört szám típusú tulajdonság is. Ezen felül még olyanokat is be lehet állítani, hogy az adott tulajdonsághoz tartozó mező lehet-e üres.

6.2 Frontend

A weboldal frontend-je, azaz a kliensoldali része, HTML (Hyper Text Markup Language) technológiát felhasználva készült, melynek kinézete Bootstrap használatával lett megformázva. Itt kerül bemutatásra a home sablon (template). A Bootstrap a leghíresebb CSS keretrendszer, melynek segítségével reszponzív weboldalakat lehet létrehozni [28]. A CSS (Cascading Style Sheets) pedig maga a stíluskészítés eszköze [29]. Az egyéb, kódot igénylő funkciók a JavaScript nyelv segítségével lettek elkészítve. A JavaScript egy szkript- vagy programozási nyelv, amely lehetővé teszi a komplex szolgáltatások megvalósítását a weboldalakon [30].

```
<head>
    <title>Szállodai ajánlórendszer</title>
    <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0-alpha1/dist/css/bootstrap.min.css" rel="stylesheet">
    <link href="https://cdn.jsdelivr.net/npm/bootstrap-datepicker@1.9.0/dist/css/bootstrap-datepicker.min.css"
        rel="stylesheet">
    <script src="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0-alpha1/dist/js/bootstrap.bundle.min.js"></script>
    <script src="https://code.jquery.com/jquery-3.6.0.min.js"></script>
    <script src="https://cdn.jsdelivr.net/npm/bootstrap-datepicker@1.9.0/dist/js/bootstrap-datepicker.min.js"></script>
</head>
```

6.6. ábra – home fájl kódrészlet, Bootstrap meghívása. (Forrás: Saját szerkesztésű)

A Bootstrap használatához az ábrán látható kódot kell beilleszteni a weboldal fejrészébe. Minden egyéb kódrészlet a weboldal testrészében helyezkedik el.

Adatbevitel

```
<div class="mb-3 row">
    <label for="city" class="col-sm-2 col-form-label">Város:</label>
    <div class="col-sm-6">
        <input type="text" class="form-control" id="city" name="city">
        <div id="city-error">
        </div>
    </div>
</div>
```

6.7. ábra – home fájl kódrészlet, egy beviteli mező. (Forrás: Saját szerkesztésű)

A beviteli mezőket (input) címkékkel (label) látjuk el, az értelmezhetőség érdekében. Emellett egy külön hiba részt is kapnak, saját azonosítóval, ha esetleg valamilyen hibaüzenetet kéne megjeleníteni az adott mezők mellett például, hogy nem maradhat üresen.

```
<div class="mb-3 row">
    <div class="col-sm-1">
        <button id="recommend" type="submit"
            class="btn btn-primary bg-success border border-dark">Ajánlj!</button>
    </div>
    <div class="col-sm-1">
        <a id="back" class="btn btn-primary border border-dark" href="{% url 'home' %}">Vissza</a>
    </div>
    <div class="col-sm-4">
        <label id="scraping_started" class="col-form-label">
        </label>
    </div>
</div>
```

6.8. ábra – home fájl kódrészlet, gombok. (Forrás: Saját szerkesztésű)

A beviteli mezők alatt található két gomb közül az egyik valójában csak egy link, amely az alap, üres weboldalra mutat. A másik pedig egy valódi gomb, melynek lenyomásával (ha nincs hiba a bevitt adatokban) kiürülnek a beviteli mezők, és átadásra kerülnek a bennük található adatok. Ezen kívül a gombok mellett van egy címke, amely figyelmezteti a felhasználót, hogy az ajánlás folyamata elkezdődött.

Eredmények

```
{% if hotels and hotels|length <= 3 %} <div class="container">
    <h1 class="mb-4">Ajánlások</h1>
    <div class="table-responsive">
        <table class="table table-bordered table-striped">
            <thead class="table-dark border border-2 border-dark">
                <tr>
                    <th class="text-center align-middle border-start-0 border-end border-2 border-white">Név</th>
                    ...
                    <th class="text-center align-middle">Válaszidő</th>
                </tr>
            </thead>
            <tbody class="border border-2 border-dark">
                {% for hotel in hotels %}
                <tr>
                    <td class="align-middle"><a href="{{ hotel.link }}" target="_blank">{{ hotel.name }}</a></td>
                    ...
                    <td class="text-center align-middle">{{ hotel.responsetime }}</td>
                </tr>
                {% endfor %}
            </tbody>
        </table>
    </div>
```

6.9. ábra – home fájl kódrészlet, az ajánlások megjelenítése. (Forrás: Saját szerkesztésű)

Ha a home fájl meghíváskor megkapja a szállodákat (és nem hibaüzenetet kap, hogy nem talált egyet se), akkor táblázatos formában, fejlécekkel együtt, megjeleníti az adatokat. A

fejlécek mindig ugyan azok, viszont a konkrét tartalom a szállodák számától függ, és egy ciklus segítségével jelenítjük meg őket. Ha pedig hibát kapna, akkor csak annak a szövegét jelenítené meg, táblázat nélkül.

Szkriptek

A weboldal megfelelő működését és a felhasználói élmény növelését különböző funkciók segítségével lehet elérni.

```
$(document).ready(function () {
    var checkout_day = new Date()
    var checkin_day = new Date()
    checkin_day.setDate(checkin_day.getDate() + 1)
    checkout_day.setDate(checkin_day.getDate() + 3)
    $('#checkin').datepicker({
        format: 'yyyy-mm-dd',
        autoclose: true
    });
    $('#checkin').val(checkin_day.toISOString().split('T')[0]);
    $('#checkout').datepicker({
        format: 'yyyy-mm-dd',
        autoclose: true
    });
    $('#checkout').val(checkout_day.toISOString().split('T')[0]);
});
```

6.10. ábra – home fájl kódrészlet, a dátumok alapbeállítása. (Forrás: Saját szerkesztésű)

Alapból, amikor betöltődik a weboldal, a becsekkolás dátuma a holnapi napra, a kicsekkolás dátuma pedig a mai naptól számított hárommal későbbi napra van állítva. Ehhez létrehozzuk a megfelelő dátum típusú változóinkat, és beállítjuk az értéküket a mai naphoz képest. Ezután a megfelelő dátumkiválasztó objektumoknak beállítjuk őket, mint kezdőértékek.

```
$('#search-form').submit(function (event) {
    event.preventDefault();
    var today = new Date()
    var list = ["checkin", "checkout", "hotels", "city", "adults", "children"]
    var isEmpty = false;
    for (const element of list) {
        var labelElement = document.getElementById(`${element}-error`);
        labelElement.textContent = "";
        var inputElement = document.getElementById(`${element}`);
        inputElement.classList.remove('is-invalid')
        if (!$(`#${element}`).val()) {
            isEmpty = true;
            setRed(element)
            setErrorText(`${element}-error`, 'Nem maradhat üresen a mező.')
        }
    }
```

6.11. ábra – home fájl kódrészlet, a mezők ürességének ellenőrzése. (Forrás: Saját szerkesztésű)

A következő ellenőrzés az adatbevitel befejezésekor, azaz az „Ajánlj!” gomb megnyomásakor történik. Ilyenkor minden egyes beviteli mező tartalmát ellenőrizzük, és ha bármelyik üres, akkor a hozzá tartozó hibacímke segítségével ezt jelezzük, a setRed és a setErrorText függvények használatával. Emellett az isEmpty változó értékét igazra állítjuk (alapból ez hamis).

```
if (!isEmpty) {
    if (new Date($('#checkin').val()) < today) {
        setErrorText("checkin-error", "A check-in napja nem lehet a mai napnál korábbi dátum!")
        setRed("checkin")
    }
    else {
        if (new Date($('#checkout').val()) <= new Date($('#checkin').val())) {
            setErrorText("checkout-error", "A check-out legalább 1 nappal a check-in után kell legyen!")
            setRed("checkout")
        }
        else {
            var labelElement = document.getElementById("scraping_started");
            labelElement.textContent = "Az ajánlás folyamata elkezdődött, kis türelmet...";
            var recommendButton = document.getElementById("recommend");
            recommendButton.disabled = true;
            this.submit();
        }
    }
}
```

6.12. ábra – home fájl kódrészlet, egyéb ellenőrzések. (Forrás: Saját szerkesztésű)

Ha nincs üres mező (azaz az isEmpty változó értéke hamis), akkor elvégezhetjük a többi ellenőrzésünket. Ha a megadott becsekkolási dátum a mai nap előtt van, akkor hibát fog dobni a weboldal, és a becsekkoláshoz tartozó hibacímkén keresztül figyelmeztetni fogja erről a felhasználót. Továbbá, ha a kicsekkolási dátum a becsekkolási dátum előtt van,

vagy ugyan azon a napon, akkor ismételten hibát fog dobni a weboldal, és a becsekkoláshoz tartozó hibacímkén keresztül figyelmezteti a felhasználót. Ha pedig minden teszt sikeres volt, akkor elfogadjuk az adatokat, és megkezdjük az ajánlást.

```
function setRed(elementId) {
    var inputField = document.getElementById(elementId);
    inputField.classList.add('is-invalid');
}

function setErrorText(elementId, newText) {
    var labelElement = document.getElementById(elementId);
    labelElement.textContent = "Hiba: " + newText;
    labelElement.classList.add('text-danger')
}
```

6.13. ábra – home fájl kódrészlet, hibaüzenetek megjelenítése és formázása. (Forrás: Saját szerkesztésű)

A korábban említett setRed függvény az adott objektum osztálylistájához hozzáadja az „is-invalid” osztályt, amely pirosra fogja állítani az objektum körvonalát, és megjelenít egy piros felkiáltójelet. A setErrorText függvény pedig beállítja a megadott hibaüzenetet piros színnel, a megadott címkének.

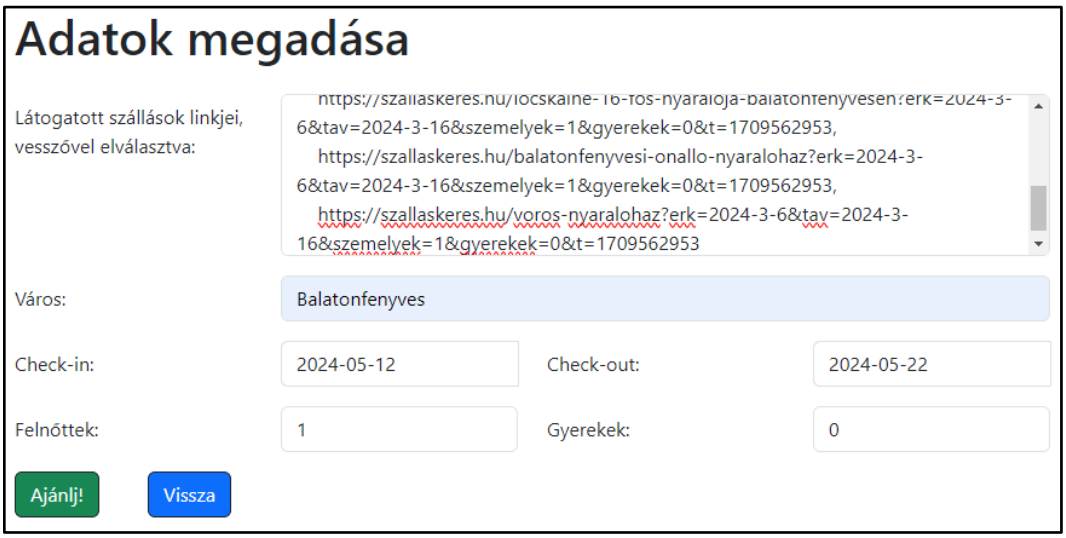
```
function handleLoad() {
    var labelElement = document.getElementById("scraping_started");
    labelElement.textContent = "";
    var recommendButton = document.getElementById("recommend");
    recommendButton.disabled = false;
    var textarea = document.getElementById("hotels");
    textarea.style.resize = "none";
}
window.onload = handleLoad;
```

6.14. ábra – home fájl kódrészlet, alaphelyzetre állítás. (Forrás: Saját szerkesztésű)

A handleLoad függvény a weboldal betöltésekor eltávolítja a setRed és a setErrorText függvények által elvégzett módosításokat.

6.3 Használat

A weboldal működése bemutatásra került programozói szemmel, most pedig bemutatom felhasználói oldalról is.



6.15. ábra – A weboldal feltöltése adatokkal. (Forrás: Saját szerkesztésű)

A felhasználónak először meg kell adnia azon szállodák linkjeit, amelyeket már korábban látogatott, ugyanis ezen szállodák tulajdonságai alapján fog történni az ajánlás. Ezen kívül meg kell adnia a várost, ahová szeretne utazni, a becsekkolás és a kicsekkolás dátumát (tehát, hogy mettől meddig szeretne maradni), valamint a felnőttek és a gyerekek számát.

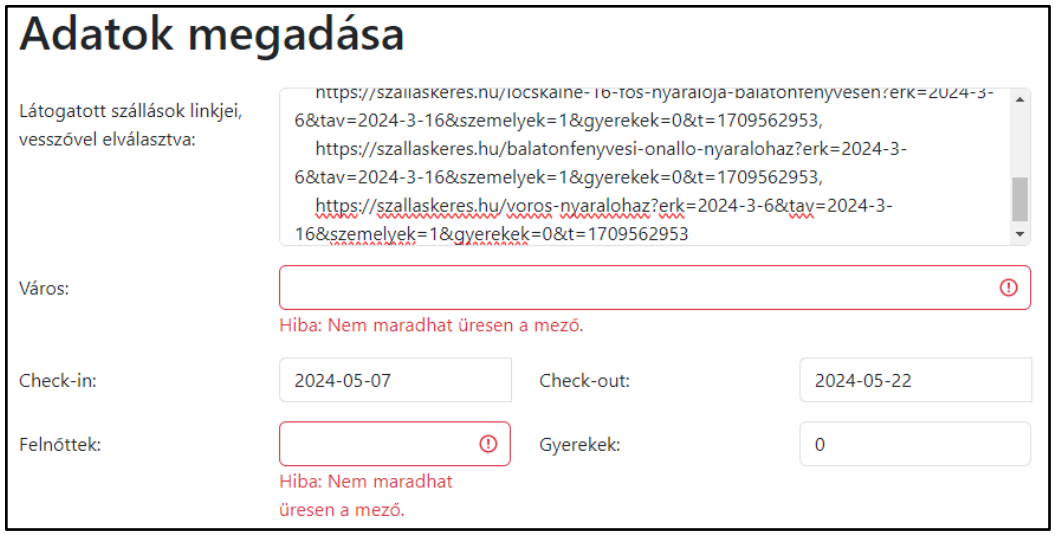
Ajánlások

Név	Cím	Ár/éj	Értékelés		Jellemzők	SZÉP	Típus	Férőhely	Válaszidő
Apartmanház Fenyves Balatonfenyves	8646 Balatonfenyves, Rákóczi u.17	14000	9.2	58	Vízközeli, Bababarát	OTP, K&H, MKB	Apartman	23	4 óra
Somogyi Apartman Balatonfenyves	8646 Balatonfenyves, Fenyvesi utca 106	20000	10.0	14	Gyógyfürdő közeli, Vízközeli, Családias, Falusi, Bababarát	Nincs	Magánszállás	12	5 perc
Dani Nyaraló Balatonfenyves	8866 Balatonfenyves, Közép út 14	21000	9.9	3	Vízközeli, Családias, Bababarát	Nincs	Magánszállás	5	3 óra

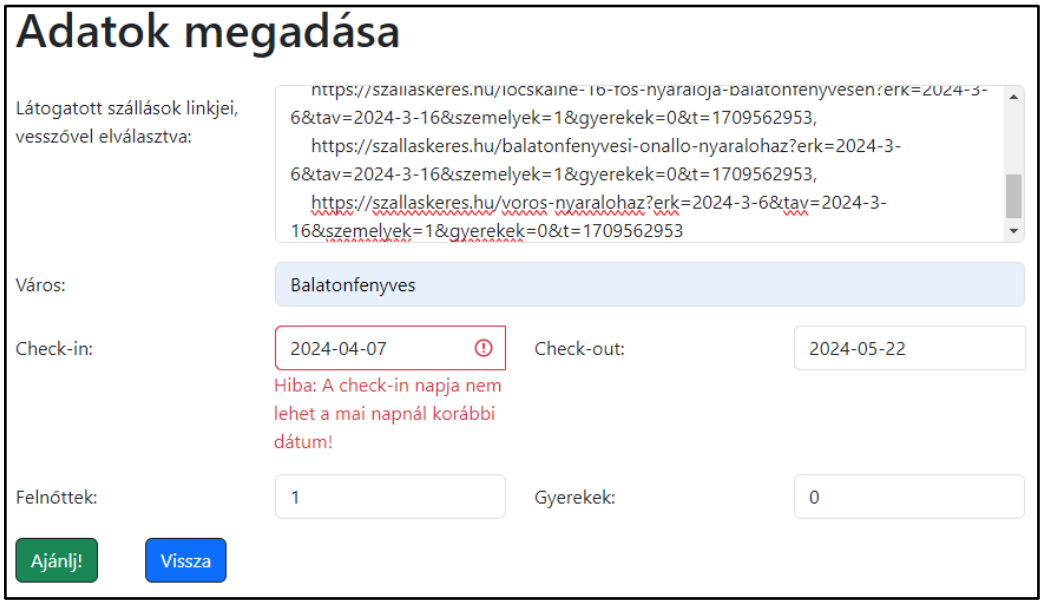
6.16. ábra – A weboldal ajánlásai. (Forrás: Saját szerkesztésű)

Az adatok megadása után a felhasználónak csupán rá kell nyomnia az „Ajánlj!” gombra, és máris előjönnek az ajánlott szállodák. A felhasználó a vízközeli, családias és bababarát szállodákat preferálja a korábban látogatott szállodák alapján, ezért egy ilyen szálloda került legelőre. Látható, hogy az elsővel ellentétben a harmadik mindhárom tulajdonsággal rendelkezik, viszont drágább, és az értékelése (főleg az értékelések száma miatt) nem olyan jó, mint az elsőnek.

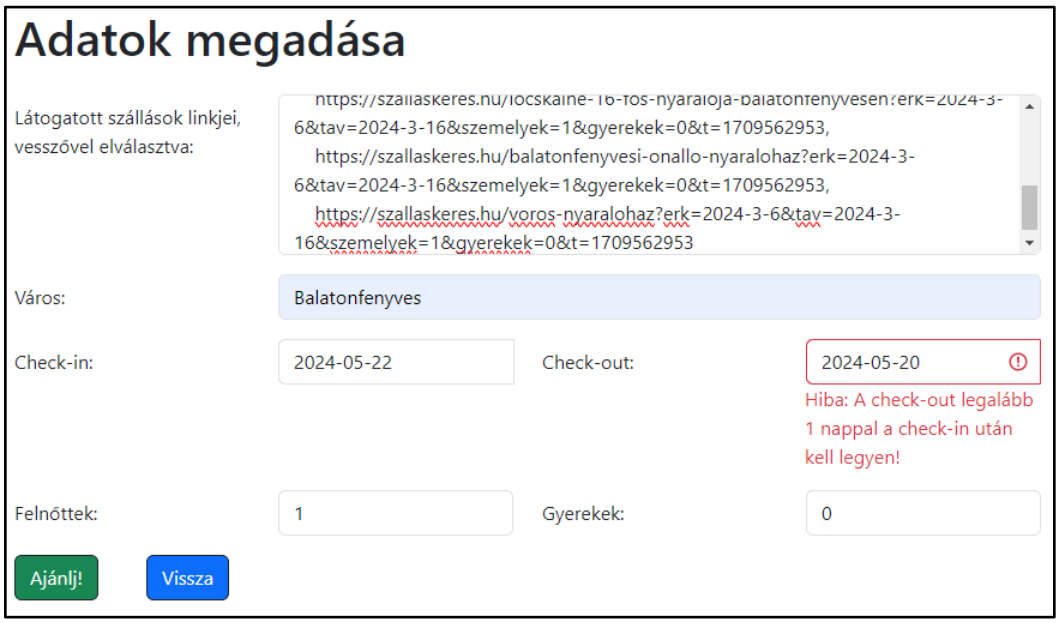
Hibaüzenetek



6.17. ábra – Hibaüzenet: hiányzó adatok. (Forrás: Saját szerkesztésű)



6.18. ábra – Hibaüzenet: nem megfelelő becsekkolási dátum. (Forrás: Saját szerkesztésű)



6.19. ábra – Hibaüzenet: nem megfelelő kicsekkolási dátum. (Forrás: Saját szerkesztésű)

7. IMPLEMENTÁCIÓ KIÉRTÉKELÉSE

7.1 Funkcionális specifikációk teljesülése

A funkcionális specifikációkat teljes mértékben sikerült teljesíteni. Az adatok begyűjtése a szallaskeres.hu weboldalról történik webscraping módszerrel, mind a lehetséges szállodák, mind a felhasználó által korábban látogatott szállodák vizsgálatakor. A begyűjtött adatok alapján a tartalom szerinti ajánlórendszerünk elvégzi a szállodák összehasonlítását, majd visszaadja a sorrendbe rendezett szállodákat a weboldalnak. A weboldalon meg lehet adni a korábban látogatott szállodák linkjeit, az utazás időtartamát, az utazás helyszínét, valamint a résztvevő felnőttek és gyerekek létszámát. Az adatok megadása után a rendszer elvégzi az ajánlást, és megjeleníti a három legígéretesebb szállodát, a felhasználó számára.

7.2 Nem funkcionális specifikációk teljesülése

Az adatbevitel után az ajánlás folyamata nem haladja meg a kitűzött egy percet, általánosságban 30-40 másodpercnyi időt vesz igénybe. Természetesen ez eltérhet a megadott időtartam, időszak és város függvényében, mivel ezek mind befolyásolják a szállodák számát. A projekt minden összetevője hibatűrő, a hibateszteknek és a biztonságos programozási technikáknak köszönhetően. A weboldalról kinyert adatok, valamint a megjelenített szállodák adatai a webscrapingnek köszönhetően pontosak, és megfelelnek a valóságnak. Maga a használhatóság az egyetlen specifikáció, ami bár teljesült, de nem a legfelhasználóbarátabb.

8. FELHASZNÁLÓI VISSZAJELZÉSEK

A projekt kiértékeléséhez és a lehetséges fejlesztések megismeréséhez felkértem néhány embert, hogy próbálják ki a weboldalt, és adjanak róla részletes visszajelzéseket. Általánosságban pozitív visszajelzéseket kapott a projekt.

Az embereknek kimondottan tetszett, hogy a weboldal könnyen használható és határozottan intuitív, valamint az is, hogy a megadott szállodák alapján valóban személyre szabott ajánlásokat kaphatnak. Kaptam pozitív megjegyzést a weboldal működésére is, a legtöbb ember szerint gyorsan működött, és pontos, valósághű ajánlásokat tett. Néhány embernek tetszett a minimalisztikus design, mivel ezáltal könnyű navigálni a weboldalt.

Volt azonban negatív visszajelzés is, avagy építő jellegű kritikák. A legtöbb ember furcsállta, hogy a korábban látogatott szállodákat link formájában kell megadni, és hiányoltak egy keresőfelületet, amelyben a szállodák nevei között lehetne válogatni. Néhány ember szerint túl szimpla a weboldal kinézete, és szerintük fejlesztésre szorulna. Elsősorban a képeket hiányolták róla, meg esetleg egy térképet. Azt is mondták, hogy célszerű lenne egy visszajelzés-rendszert beépíteni magába a weboldalba, hogy azok a felhasználók is tudjanak véleményt mondani, akik nem ismernek engem.

Felmerült egy érdekesebb probléma is, amikor a weboldalt meg szerettem volna mutatni egy külföldi barátomnak: az oldal teljesen magyar, valamint az ajánlott/látogatott szállások is szinte teljes mértékben magyarok, ezáltal ő ki se tudta próbálni.

Egy szoftverfejlesztő barátom a kódot is megnézte, és neki is voltak pozitív és negatív véleményei. Elsősorban meglepődött, hogy egy ilyen széleskörű és nagy projektet ilyen kevés technológiával, ilyen jól meg tudtam valósítani. A kód szerinte jól átlátható, jól dokumentált (a példák során kivettem a kommenteket a helytakarékosság érdekében) és értelmes. Az általa felhozott kritika az volt, hogy a weboldal ajánlórendszere használhatna több kritériumot az ajánláshoz, mint például a SZÉP kártya elérhetősége. Emellett felhívta a figyelmem arra is, hogy a „30-40” másodperces ajánlási idő nem rossz, de egy valós környezetben ezt a töredékére kéne csökkenteni ahhoz, hogy használható legyen a weboldalam.

A visszajelzések nagyon hasznosak voltak számomra. Sok mindent kijavítottam a véleményeknek köszönhetően, mivel olyan hibákra hívták fel a figyelmem, amelyekbe bele se gondoltam volna korábban, mivel túlságosan is programozói szemmel néztem a projektem.

9. LEHETSÉGES FEJLESZTÉSEK

Felhasználóbarátabb adatbevitel

Az adatbevitelnél a felhasználó a korábban látogatott szállodáit link formájában kell megadja. Ez sajnos nem a legkényelmesebb megoldás, ezért a következő verzió egyik legelső fejlesztése egy keresőmező lenne, amelyben a felhasználó a szállodák nevei közül kiválaszthatná, hogy melyikeket látogatta már korábban.

Széleskörűbb támogatottság

A weboldal jelenleg a szallaskeres.hu által támogatott szállásokat vizsgálja, ami azt jelenti, hogy a felhasználó általánosságban Magyarországra, vagy esetleg a szomszédos országokra van lekorlátozva. A következő verzióban megfontolandó egy másik, nemzetközibb weboldal használata. Ez azonban a webscraping függvények teljes kicserélését jelentené, ami önmagában hatalmas változtatás, emellett az adatbeszerzés a nagyobb weboldalakról sokkal nehezebb, a beépített anti-scraping védelmek, és a jogi kérdések miatt.

Nyelvi támogatás

Jelenleg a weboldal teljes mértékben magyar, mivel a szállások nagyrésze szintén magyar. A következő verzióban a nemzetközi weboldal mellett be lehetne vezetni az egyéb nyelvek támogatását is. A nyelveket a felhasználók a weboldalon állíthatnák be.

Több kritérium az ajánlórendszerben

Az ajánlórendszer jelenleg az árat, az értékeléseket, az értékelések számát, valamint a szálloda jellegét veszi figyelembe. Ezt ki lehetne egészíteni az olyan tulajdonságok vizsgálatával, mint például a SZÉP kártya lehetősége, vagy a szállás típusa.

Webdesign

Jelenleg a weboldal kinézete határozottan minimalisztikus. A következő verzióban ki lehetne egészíteni képekkel, több menüvel, esetleg még egy térképpel is. A Bootstrap sokkal több lehetőséget kínál, mint amire ennél a projektnél használva lett.

Visszajelzések

A weboldalnak a következő verzióban lehetne készíteni egy visszajelző rendszert, amely használatával a felhasználók értékelhetnék a kapott ajánlások hatékonyságát.

10.ÖSSZEFOGLALÁS

A projekt egy webes ajánlórendszert valósít meg, webscraping módszer segítségével, Python nyelven. Az ajánlórendszerek olyan rendszerek, amelyek személyre szabott javaslatokat, ajánlásokat nyújtanak a felhasználók számára olyan elemekről vagy tartalmakról, amelyek leginkább relevánsak számukra. Manapság minden híresebb szolgáltató használ ajánlórendszert, például: Youtube, Spotify, Amazon. Az ajánlórendszereknek két fajtája van: a kollaboratív szűrés, amely a hasonló felhasználókra épül, és a tartalom szerinti szűrés, amely az elemek tulajdonságaira épül. Ez a projekt tartalom szerinti szűrést valósít meg, mivel a szállodák tulajdonságaira összpontosít.

A projekt három fő funkcionális részből áll: adatbegyűjtés (webscraping), ajánlórendszer és weboldal. Emellett számos nem funkcionális követelmény is van a projekttel szemben, mint a sebesség, használhatóság, karbantarthatóság, pontosság és hibatűrés.

Az adatok begyűjtése a szallaskeres.hu weboldalról történik. Ez egy kimondottan szimpla, jól struktúrált és könnyen használható weboldal, viszont csak magyar szállásokat tartalmaz. A nagyobb weboldalakról (például booking.com) való adatbegyűjtéssel az a probléma, hogy számos anti-scraping védelemmel rendelkeznek, és esetleges jogi problémák is felmerülhetnek, ha "scrapelni" szeretnénk őket. Az adatbeszerzés legfőképpen a BeautifulSoup könyvtárral, valamint a dinamikus elemek miatt néhány helyen a Selenium könyvtárral. A begyűjtött adatokat a pandas könyvtár DataFrame típusával tároljuk el, amely lehetővé teszi a táblázatos formájú adattárolást. Az eltárolt adatokat megtisztítjuk, és egységes formára hozzuk. A kész DataFrame-t átadjuk az ajánlórendszernek, amely elvégzi az ajánlást.

Az ajánlórendszer a szállodák árára, értékelésére, értékeléseinek számára, valamint a tulajdonságaira épül. Minden szállodának kiszámolunk egy ár-pontszámot, egy értékelés-pontszámot, valamint egy tulajdonság-pontszámot, amelyekkel a végén kiszámítjuk a végső pontszámot. A végső pontszám alapján lesznek sorrendbe rendezve a szállodák (legrelevánsabb legelöl), amelyekből az első hármat jelenítjük meg a felhasználónak.

A weboldal szerveroldali része Django keretrendszer segítségével készült, amely egy model-view-template struktúrát követ. A modellbe töltjük fel az adatokat, a view-k maguk

az elvégzendő függvények, a sablonok pedig maguk a megjelenítendő weboldalak. A kliensoldali rész HTML, Bootstrap és Javascript használatával készült, és számos hibaellenőrző szkriptet tartalmaz.

Az elkészült projekt megfelel a funkcionális és a nem funkcionális specifikációknak, viszont több lehetséges fejlesztés is felfedezhető benne. Az egyik ilyen a felhasználóbarátabb adatbevitel, hogy minél jobb élményben részesüljenek a felhasználók. Egy másik ilyen a nemzetköziség bevezetése, mivel jelenleg főleg magyar szállásokra, és egy magyar weboldalra vannak lekorlátozva a felhasználók.

Összességében a projekt véleményem szerint sikeresnek mondható. Rengeteg érdekes kihívással találkoztam fejlesztés közben, és egy olyan eszközt sikerült alkotnom, amely számomra (és remélhetőleg mások számára is) hasznos.

11.SUMMARY

This project is a web-based recommender system, using webscraping for data collection, written in the Python programming language. Recommender systems provide customized recommendations for users about items that might be interesting for them. Nowadays every prelevant platform uses recommender systems, for example: Youtube, Spotify and Amazon. There are two types of recommender systems: collaborative filtering, which is based on similar users, and content based filtering, which focuses on the attributes of the given elements. This project implements content based filtering, because it focuses on hotel attributes.

The project has three main functional parts: data collection (using webscraping), the recommender system and the website itself. There are several non-functional specifications as well, like efficiency, user experience, maintainability, precision and error tolerance.

Data is gathered from the website "szallaskeres.hu". This is a very simple, well structured and easy to use website, but we can only book Hungarian hotels on it. Collecting data from bigger websites (like booking.com) could prove problematic, because these sites have several anti-scraping protections, and we could even face legal issues if we try to scrape them without permission. The main library used for webscraping is BeautifulSoup, and for some dynamic elements, Selenium is used. The collected data is stored in DataFrames, a courtesy of the pandas library. The gathered data is then cleaned and standardized, and is given to the recommender system.

The recommender system focuses on price, rating, number of ratings, and type. We calculate a price-score, rating-score and type-score for each hotel, then we use these scores to calculate the final score. The final score is used to order the hotels (first one is the most relevant), of which only the first three are shown to the user.

The website's server side technology is Django, which uses a model-view-template structure. Models are used for data storage, views are the functions themselves, and the templates are the sites we want to show. The client side uses HTML, Bootstrap and Javascript, with several error checking scripts.

The finished project meets all the functional and non-functional expectations, but there are several points on which it could improve. One of these points is being more user friendly, and thus improving user experience. Another is being more international, because currently users are limited to Hungarian hotels, and a Hungarian website.

To sum it up, the project is a success, in my eyes. I have been faced with several interesting problems throughout development, and I created a tool that is not only useful to myself, but hopefully for others too.

12.IRODALOMJEGYZÉK

[1] „Towards Data Science,” [Online]. Available: https://towardsdatascience.com/recommender-systems-a-complete-guide-to-machine-learning-models-96d3f94ea748. [Hozzáférés dátuma: 20. február 2024.].

[2] „Cleveland Clinic,” [Online]. Available: https://health.clevelandclinic.org/decision-fatigue. [Hozzáférés dátuma: 8. február 2024.].

[3] „Oberlo,” [Online]. Available: https://www.oberlo.com/blog/youtube-statistics. [Hozzáférés dátuma: 13. február 2024.].

[4] „Spotify,” [Online]. Available: https://newsroom.spotify.com/company-info/. [Hozzáférés dátuma: 17. február 2024].

[5] „Retail Touch Points,” [Online]. Available: https://www.retailtouchpoints.com/resources/how-many-products-does-amazon-carry. [Hozzáférés dátuma: 13. február 2024].

[6] „Turing,” [Online]. Available: https://www.turing.com/kb/collaborative-filtering-in-recommender-system. [Hozzáférés dátuma: 20. február 2024.].

[7] „Springer,” [Online]. Available: https://www.cse.iitk.ac.in/users/nsrivast/HCC/Recommender_systems_handbook.pdf. [Hozzáférés dátuma: 20. február 2024.].

[8] „Google,” [Online]. Available: https://developers.google.com/machine-learning/recommendation/collaborative/summary. [Hozzáférés dátuma: 20. február 2024.].

[9] Youtube. [Online]. Available: https://www.youtube.com/howyoutubeworks/product-

features/recommendations/#signals-used-to-recommend-content. [Hozzáférés dátuma: 20. február 2024.].

[10] „Turing," [Online]. Available: https://www.turing.com/kb/content-based-filtering-in-recommender-systems. [Hozzáférés dátuma: 20. február 2024.].

[11] „Google," [Online]. Available: https://developers.google.com/machine-learning/recommendation/content-based/summary. [Hozzáférés dátuma: 20. február 2024.].

[12] ACM. [Online]. Available: https://dl.acm.org/doi/10.1145/2843948. [Hozzáférés dátuma: 1. március 2024.].

[13] „BeautifulSoup," [Online]. Available: https://www.crummy.com/software/BeautifulSoup/bs4/doc/. [Hozzáférés dátuma: 4. február 2024.].

[14] „Requests," [Online]. Available: https://requests.readthedocs.io/en/latest/. [Hozzáférés dátuma: 17. február 2024.].

[15] „Pandas," [Online]. Available: https://pandas.pydata.org/. [Hozzáférés dátuma: 17. február 2024].

[16] „Selenium," [Online]. Available: https://www.selenium.dev/documentation/. [Hozzáférés dátuma: 17. február 2024.].

[17] „SelectorsHub," [Online]. Available: https://selectorshub.com/selectorshub/. [Hozzáférés dátuma: 17. február 2024].

[18] „imperva," [Online]. Available: https://www.imperva.com/learn/application-security/web-scraping-attack/. [Hozzáférés dátuma: 10. február 2024.].

[19] „moz," [Online]. Available: https://moz.com/learn/seo/robotstxt. [Hozzáférés dátuma: 10. február 2024.].

[20] „ZenRows," [Online]. Available: https://www.zenrows.com/blog/anti-scraping. [Hozzáférés dátuma: 13. február 2024.].

[21] „Octoparse,” [Online]. Available: https://www.octoparse.com/blog/web-scraping-limitations. [Hozzáférés dátuma: 10. február 2024].

[22] „ZenRows,” [Online]. Available: https://www.zenrows.com/blog/web-scraping-vs-api. [Hozzáférés dátuma: 13. február 2024.].

[23] Booking.com. [Online]. Available: https://www.booking.com/content/terms.hu.html. [Hozzáférés dátuma: 17. február 2024.].

[24] „Epoch,” [Online]. Available: https://www.epochconverter.com/.

[25] „Django,” [Online]. Available: https://www.djangoproject.com/. [Hozzáférés dátuma: 7. február 2024.].

[26] „Django,” [Online]. Available: https://docs.djangoproject.com/en/5.0/intro/overview/#write-your-views. [Hozzáférés dátuma: 7. február 2024.].

[27] „Django,” [Online]. Available: https://docs.djangoproject.com/en/5.0/intro/overview/#design-your-model. [Hozzáférés dátuma: 7. február 2024.].

[28] „W3Schools,” [Online]. Available: https://www.w3schools.com/whatis/whatis_bootstrap.asp. [Hozzáférés dátuma: 25. február 2024.].

[29] „W3Schools,” [Online]. Available: https://www.w3schools.com/css/css_intro.asp. [Hozzáférés dátuma: 25. február 2024.].

[30] „Mozilla,” [Online]. Available: https://developer.mozilla.org/en-US/docs/Learn/JavaScript/First_steps/What_is_JavaScript. [Hozzáférés dátuma: 25. február 2024.].

[31] „Geeks For Geeks,” [Online]. Available: https://www.geeksforgeeks.org/collaborative-filtering-ml/. [Hozzáférés dátuma: 20. február 2024.].

[32] „Turing,” [Online]. Available: https://www.turing.com/kb/collaborative-filtering-in-recommender-system. [Hozzáférés dátuma: 20. február 2024].

[33] Youtube. [Online]. Available: https://www.youtube.com/howyoutubeworks/product-features/recommendations/#signals-used-to-recommend-content. [Hozzáférés dátuma: 20. február 2024.].

13.ÁBRAJEGYZÉK