ÓBUDAI EGYETEM
ÓBUDA UNIVERSITY

Alba Regia Műszaki Kar
Természettudományi és Szoftvertechnológiai Intézet

SZAKDOLGOZAT

Nagykereskedelmi élelmiszerláncok termékár-összehasonlító webalkalmazása

OE-AMK

2024

Hallgató neve: **Környei Balázs**

Hallgató törzskönyvi száma: **T001199/FI12904/A-M**

ÓBUDAI EGYETEM
ÓBUDA UNIVERSITY

Óbudai Egyetem
Alba Regia Műszaki Kar
Természettudományi és Szoftvertechnológiai Intézet

SZAKDOLGOZAT FELADATLAP

Hallgató neve: Környei Balázs
Szakdolgozat száma: SZD2301230932327284[redacted]
Törzskönyvi száma: T001199/FI12904/A-M **Neptun kódja:** [redacted]

Szak: Mérnökinformatikus
Specializáció: Felhő szolgáltatási technológiák és IT biztonság specializáció

A dolgozat címe: Nagykereskedelmi élelmiszerláncok termékár-összehasonlító webalkalmazása
A dolgozat címe angolul: Products Price Comparison Web Application for Wholesale Food Chains
A feladat részletezése: A hallgató feladata nagykereskedelmi élelmiszerláncok termékár-követő és -összehasonlító webes alkalmazásának készítése. Ennek keretében a hallgató készítsen egy adatbázist az áruházakban kapható termékekről és a termékekkel kapcsolatos további információkról. Az adatbázist backenddel és frontenddel rendelkező weboldalon kell használnia, és felhasználóbarát funkciókkal kell ellátnia. A dolgozat mutassa be a rendszer tervezésének és fejlesztésének lépéseit, az elkészült rendszer értékelését, beleértve annak hatékonyságát, skálázhatóságát és használhatóságát a vásárlási döntések megkönnyítése területén.

Intézményi konzulens neve: Piglerné Dr. Lakner Rozália

Külső konzulens neve: Pigler Péter
Munkahelye: One Beyond Ltd.

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 1.

Kiadva: Székesfehérvár, 2024. 05. 09. PH

.....
Intézetigazgató
helyett

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 dolgozat saját munkám eredménye, a felhasznált szakirodalmat és eszközöket azonosíthatóan közöltem. Az elkészült dolgozatban található eredményeket az Óbudai Egyetem és a feladatot kiíró intézmény saját céljára térítés nélkül felhasználhatja, a titkosításra vonatkozó esetleges megkötések mellett.

Kelt: Székesfehérvár *(hely)*, 2024.05.14. *(dátum)*

.....
hallgató aláírása

Tartalomjegyzék

1. BEVEZETÉS

Az étel az emberiség egyik legalapvetőbb létszükséglete, ahogy maga a víz, vagy az oxigén is. Fontos építőköveket ad át testünk működéséhez, amely meghatározza, hogy nézünk ki, hogy érezzük magunkat, milyen gondolataink lesznek, vagy hogy milyen betegségek alakulnak ki, vagy kerülünk el [1]. Az élelmiszerről tehát beszélhetünk egyfajta utasításként is, hiszen megmondja, hogy DNS-ünk miként működjön: fontos, hogy az adott élethelyzetben milyen táplálékkal látjuk el, hogy megfelelő módon üzemeljen. Tudjuk jól, hogy az emberiség különböző nemzetiségei, országai, külön konyhával, ízhagyománnyal, vagy fűszerekkel rendelkeznek, amelyek úgy épültek fel, hogy az emberi test számára minden szükséges mikró-, és makró szintű elemek fedezve legyenek, ne alakuljon ki hiány bennük. Hagyományos ételek az adott régióban elérhető, megtermelhető, vagy az olcsóbb, kereskedelmi utak kiépülése során biztosítottan megszerezhető alapanyagokból épülnek fel.

Modern korban szintén ugyanez a konstelláció: Abból készülnek napi ételeink, amiket üzletekben, piacon, vagy már ritkább esetben magunk termelünk meg. Ez az átformálódás, amely az ipar fejlődésével felgyorsult, egyrészt kiszolgáltatottá tesz az adott kínálathoz, másrészt nagyobb lehetőséget biztosít abban, hogy milyen árukat, élelmiszereket használhatunk fel a betevőink alapanyagaiként. A nagy választék helyenként változhat nemcsak kínálatban, hanem adott élelmiszernek vételárában is. Kapitalista világban a nyereség növelésének áldozata mindig a fogyasztó, melynek bevétele változik a piac működésétől, illetve változásától. Fontossá válik tehát az, hogy milyen alapanyagot veszünk meg, és mennyiért. Ezekben segíthet a nagyobb élelmiszerláncok weboldalain megtalálható termékpaletta is, amely szerencsés esetben az árakat is tartalmazza, az összetevőikkel együtt. És bár vannak weboldalak, amelyek tartalmaznak bizonyos termékeket összehasonlítással, azok nem teljesek, legfőképp webshopok áruit adják vissza. Szükséges ezáltal egy olyan oldal, amely egyesíti a fogyasztónak mindazon elemeket, amelyek racionális döntéseikhez létfontosságúak a bevásárlás során, mindezt a lehető legkönnyebben megjelenítve.

2. A MEGOLDANDÓ FELADAT ELEMZÉSE

2.1 A probléma megfogalmazása

A 2019 után történt gazdasági átalakulás, majd megnövekedett infláció okozta élelmiszerár nagymértékű és -léptékű növekedése ébresztette fel bennem annak lehetőségét, hogyan lehet ugyanazt a terméket, minőséget kevesebbért beszerezni. Egy adott településen élő emberek kiszolgáltatottak az ottani kínálat árainak ingadozásának, melyet úgy tudnak kiküszöbölni, ha egyrészt maguk termelik meg lehetőségeikhez képest az élelmiszereket, illetve másrészt, - amely kézenfekvőbb – olcsóbb helyekről szerzik be az árukat, minőségromlás nélkül.

2.2 A probléma megoldása

A jelenlegi orvosolandó helyzetet egy olyan eszköz lenne képes megoldani, amely:

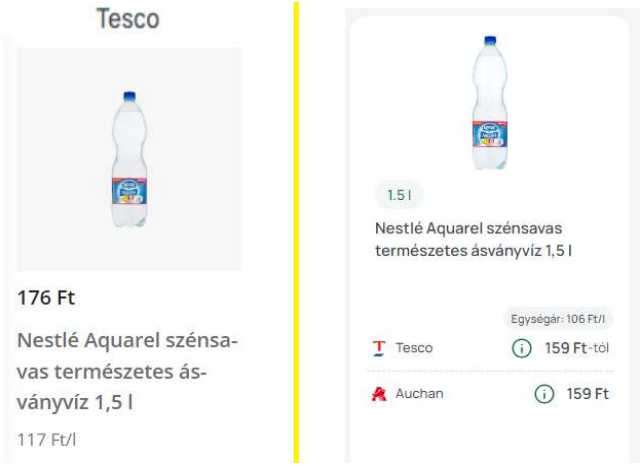
1. Naprakészen tartalmazza az adott áruk értékét,
2. Lehetőséget ad az összehasonlításra anélkül, hogy más weboldalakat kellene felhasználnia egyidejűleg,
3. Elérhetősége könnyű bármilyen eszközről,
4. Felhasználótól nem vár el több tudást a használathoz az eszköz használatának képessége mellett.

Célom volt tehát egy adatbázist létrehozni, amellyel összegyűjtve a releváns árukat, meg tudom jeleníteni őket a megadott pontok alapján. Eleinte az egész országban elérhető élelmiszerláncok weboldalait használtam fel, amely során többszöri igazítás és átírás után jutottam el arra a konklúzióra, hogy nem mindegyik áruház tölti fel a terméklistáját, főleg a hozzá tartozó árait. Az adatok eléréséhez kétféle lehetőséget vettem figyelembe; az API kérést, és a weboldalról való adatok lehúzását. Az elérhető élelmiszerláncok weboldalán található API meghívások nem voltak olyan minőségűek, hogy egyedi termékeket lehessen keresni, csak többszöri szűréssel, kategorizálással lehetett volna használni. Ha nem API kérés, akkor weboldalról lerántott adat maradt volna második opciónak, de többszöri lehívás akadályozva volt az oldalak szerverei által. Mindez annak köszönhető, hogy a

szerver oldaláról egy IP címről intézett többszörös lekérését blokkolja, hogy védje a rendszert a túlterheléses támadástól. De voltak olyan problémák is, amely során a termék elérhetetlen volt adott időszakonként az oldal működéséből kifolyólag.

2.1. ábra Lidl heti termékpaletta változása: Egy nap elérhető, másik nap már nem (https://www.lidl.hu/p/cola/p23)

Az olyan oldalak, mint a Foodora, Wolt, vagy Roksh rendszere, nagyobb méretű, a hiányzó árukat is tartalmazó adatbázissal rendelkeznek. Ezek ellenben nem igazodnak a napi változásokhoz, illetve a termékek árai tartalmazzák a rendszer használatának profitmarginját (2.2. ábra), akár még a szállítási költséget is.



2.2. ábra Árkülönbségek a bal oldalon látható Foodora (https://www.foodora.hu/shop/slel/tesco-hipermarket-szekesfehervar-palota/search?q=v%C3%ADz) és a jobb oldalon látható Árfigyelő (https://arfigyelo.gvh.hu/k/tartos_elelmiszer_ital/viz_gyumolcsle/asvanyviz/t/3700123300281) között

Ezért az előbb felsorolt problémák sorozata miatt döntöttem úgy, hogy a már elérhető Árfigyelő [2] adatai lesznek leginkább kézenfekvőek, megbízhatóság, elérhetőség, és diverzitás szempontjából, mivel teljesíti a korábban megfogalmazott követelményeket. Az Árfigyelő a Gazdasági Versenyhivatal online rendszere, amely segítségével a fogyasztók számos élelmiszer napi árait hasonlítják össze [3], jelenleg a szakdolgozat megírásáig csak nagy élelmiszerláncok áruival. A rendszer segít abban, hogy a termékeket egy bevásárló kocsiba tegyük az üzletet kiválasztva, elkerülve, hogy olyan kerüljön kosarunkba, amely csak az ország másik részén érhető el, másik üzletben.

Ennek a rendszernek az adatait felhasználva szerettem volna olyan információkat, funkciókat is hozzáadni, amelyeket szükségesnek éreztem szerepeltetni az oldalon belül.

Fontosnak találtam az olyan információk, funkciók szerepeltetését, mint:

1. Az élelmiszerek összetevői és energiatáblázatai: a vásárlókat tájékoztatni kell ezekről az információkról, azoknál a termékeknél, amelyek bizonyos kritériumoknak megfelelnek az Európai szabályozás alapján [4].
2. Helyi keresési funkció, ahol szűrten lehet keresni, hogy mely termékek elérhetőek bizonyos távolságon belül körülöttünk.
3. Termékek árváltozásának követhetősége, az elérhető maximum és minimum vételár által.

Ezeket figyelembe véve építettem fel egy olyan weboldalt, amely intuitívan segítheti a felhasználót legjobb döntéshozatalra az adott élelmiszerválasztással kapcsolatban.

2.3 A felhasznált eszközök

A szoftver fejlesztésének nyomon követéséhez szerettem volna verziókövetést alkalmazni, ezért használtam a Github platformot Ez a Git szoftverre épülő verziókövető PaaS (Platform as a Service), amely lehetővé teszi a hibás verziókat visszagörgetését, másrészt, hogy audit-szerű beszámolást az elvégzett munkáról és annak fázisairól. Ezen döntés után térhettem rá a weboldal, és alkalmazás fejlesztésére.

Egy weboldal készítése attól függ, hogy milyen céllal fejlesztik, illetve milyen elemeket kell, hogy tartalmazzon. Egyszerűbb weboldalakhoz, mint például portfoliókhoz vagy

információs portálokhoz leginkább alapvető moduláris motorok, mint a WordPress, vagy online elérhető weboldal fejlesztő platformok közül a Wix, a Blogger, vagy a Weebly használata elterjedt. Ezeket figyelembe véve szűkítenem kellett a kört, amelyben megfogalmaztam, hogy a weboldalamnak mely kritériumoknak kell eleget tennie, amelyek az alábbi pontokban láthatók:

1. Felhasználóbarát kinézet, különböző informatív, intuitív elemmel, de fizetési opció nélkül.
2. A weboldal adatbázisból dolgozik, amely tőle elkülönítve, csak olvasással rendelkezik.
3. Az adatbázis egyszerű, csak tárolásra használt elemként kell, hogy funkcionáljon, hiszen minden adat, amelyet tartalmaz, már nyílt forrású weblapokról származik.
4. Könnyű, helyi back-end oldali működés, amely átlátható és érthető.
5. Nem szükséges új nyelv paradigma használata a webalkalmazás létrehozásához, hanem az eddig számomra ismert korszerű nyelveket használjam fel.

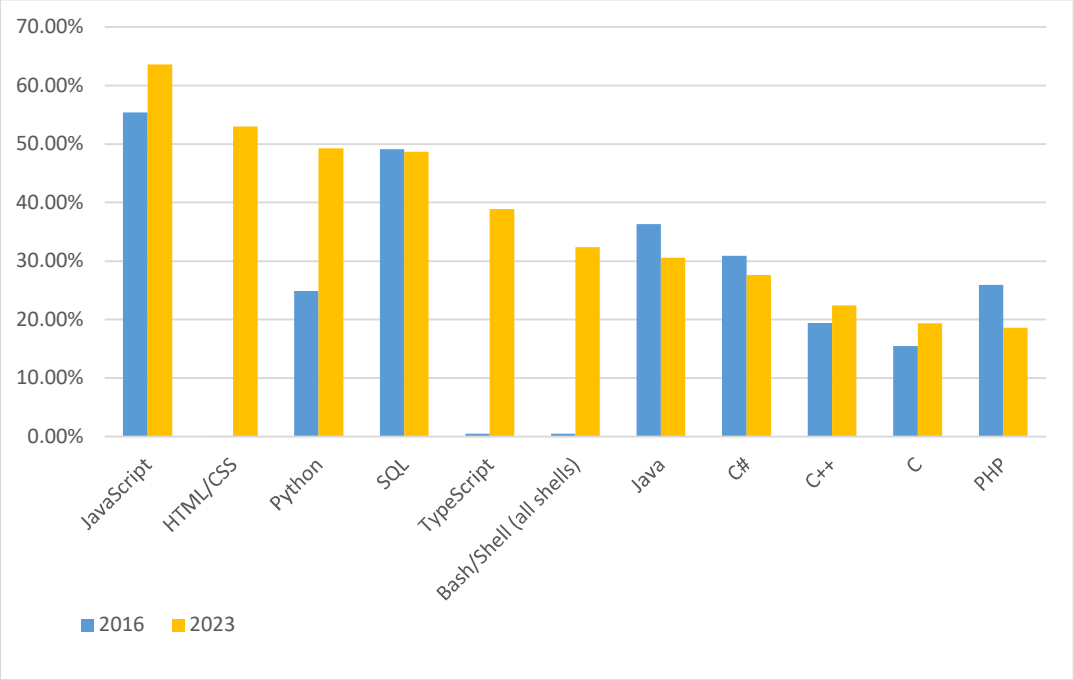
Az eddig megemlített weboldalak nem voltak képesek betölteni annak lehetőségét, hogy a gépem legyen az erőforrásom, illetve, hogy az adatbázis, amelyet felhasználnék sem lenne integrálható. Más platformokat figyelembe véve került elő a leginkább ’Drop shipping’ (olyan üzleti forma, ahol az üzlet elfogad rendeléseket, de nincs készleten raktározva a termék) közösség számára ismert és használt Shopify, GoDaddy, vagy Etsy, amely képes adatbázist használni, de ezek a weboldalra megfogalmazott negyedik kritériumnak nem tettek eleget, mivel egyik platform sem volt elegendő arra, hogy helyileg jelenítsem meg az adatokat. A webalkalmazás tehát komplexebb, szabadabban konfigurálható rendszert igényelt, amely átláthatóság szempontjából két nagy részre bomlik: Front-end és Back-end.

2.3.1 Front-End

Ezen területen belül találkozik a felhasználó a rendszerrel. Nem kell foglalkoznia a háttérben történő kalkulációkkal, átirányításokkal, szimplán azokat az elemeket kell használnia, amelyeket a UI megjelenít. Megjelenését HTML, CSS és JS nyelv segítségével a

böngésző motorja fordítja le, de a megjelenítendő adatok képzését már más nyelv kezeli. Ennek variációi mind modellben, mind nyelvben szerteágazóak lehetnek, hiszen minden attól függ, hogy a back-end milyen formátumot használ (a későbbiekben erre kitérek).

A weboldalt a korábbi években volt lehetőségem szoftverben WYSIWYG (**W**hat **Y**ou **S**ee **I**s **W**hat **Y**ou **G**et – Amit látsz, azt kapod) stílusban létrehozni. Többek között a Freeware Mozilla Kompozer is hasonlóan működött, de akkor még jobban elterjedt volt a weboldal közvetlen megírása különböző plusz kiegészítők segítségével. Ilyen telepíthető webfejlesztő szoftver volt már akkor a NetBeans vagy Komodo, amely a mai napig létezik. Napjainkban ellenben teljesen megváltozott az internethez való hozzáállás, hiszen ez már betölt több olyan funkciót a társadalomban, mint a társkeresés, kikapcsolódás, oktatás, vagy pénzkeresés. A mobilvilág fejlődése, magasabb igények és jobb hardver erőforrások következtében a fejlesztési eszközök, a módszerek, a megvalósítási nyelvek és keretrendszerek mind megváltoztak. Ilyen idomulás volt például a gyors online fizetések lehetősége, a 3D modellek interaktív megjelenése weblapokon keresztül, a duplex előadások megtartása videókonferencián keresztül, a virtuális pénz tőzsdézése, a különféle eszközök rendelése, az élelmiszerek vásárlása vagy komplett weboldal fejlesztése webfelületről. Ezek rendkívüli módon hatottak a piaci részesedésre a nyelvek használatában, amelyet a Stack Overflow 2016-os [5] és 2023-as [6] felmérésének eredménye is igazol. A leggyakrabban használt 12 programnyelv összehasonlítását a 2.3. ábra mutatja be.

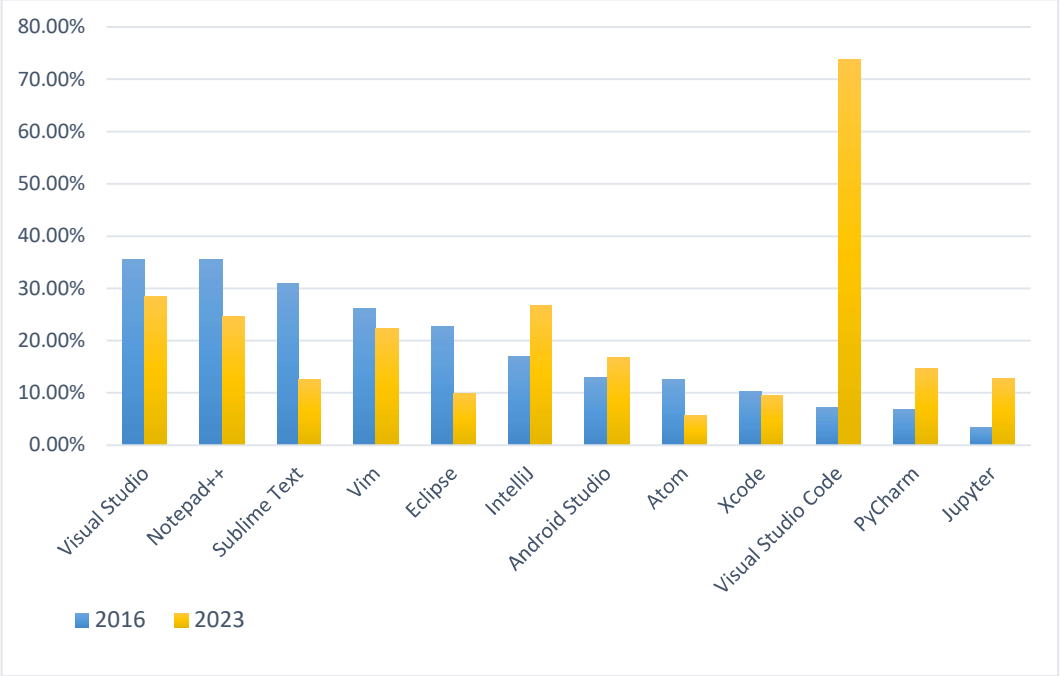


2.3. ábra Top 12 programnyelv összesítése 2016-ban és 2023-ban

Mindezeket figyelembe véve, és mivel tanulmányaim során megismertem a böngésző nyelvek használata, az egyszerűséget megtartva szoftver végleges kinézetét HTML5 nyelvben hoztam létre.

2.3.2 Back End

A szoftver háttérmunkája ezen a területen történik. Ez az, amely változik a böngésző nyelveitől függően, és olyan műveleteket képes háttérben végrehajtani, amelyeket csak feldolgozás után tud megjeleníteni. A nyelvek kiválasztásához sokrétű lehetőség sorakozott, amelyhez a megfelelő IDE is társult. Ez terület is nagy változásokon ment keresztül az évek folyamán beleértve nem csak a nyelvet, hanem magát a fejlesztői környezetet is. Mindezt a Stack Overflow felmérései [5] [6] is megmutatnak és amely részben befolyásolta választásomat is.



2.4. ábra Top 12 fejlesztői környezet összesítése 2016-ban és 2023-ban

A 2.4. ábra bemutatja, hogy napjaink egyik legsokoldalúbb IDE-je, a Microsoft által biztosított Visual Studio Code, a következő pedig az eredeti verziója, a Visual Studio. Ez a fejlesztői környezet sokrétű nyelv támogatásával, kiegészítői mennyiségével és felületével érte el a mostani népszerűségi szintjét. A Visual Studio-t már tanulmányaim és egyéni projektjeim során is használtam, ezért szakdolgozatom megvalósításához is ezt választottam. Verziójából a Közösségi kiadást használtam, és mivel egyetemi fiókkal is képes voltam használni, ingyenesen használtam bizonyos Nagyvállalati extrák nélkül.

Visual Studio

A Visual Studio a weboldal háttérmunkáit határozza meg, amelyet a már ismert ASP (**A**ctive **S**erver **P**ages – Aktív szerver oldalak) webalkalmazás .NET keretrendszerén belül hoztam létre. Az ASP.NET a Microsoft által publikált keretrendszer, C# és weboldal-fejlesztő nyelvek segítségével jeleníti meg az oldalakat. Olyan ASP műveleti elemeket is tartalmaz, amelyek nem igénylik a weboldal-motor alapú nyelvek (JavaScript, PHP) használatát. Minden olyan elem, amely szerver oldalú feldolgozást igényel, felhasználja az

XML formátumot, és plusz programkód segítségével pedig előre feldolgozza az adatokat a böngésző számára. NET 8.0 verziót használtam, mivel jelenleg ez a legfrissebb kiadás, amely támogatottsága hosszútávú: előreláthatóan legalább 2 évig még frissítéseket adnak ki hozzá [7].

Codeium / Github Copilot

A Codeium / Github Copilot egyre több ismert IDE rendszerben jelenik meg AI kiegészítőként, amely másodkezet ad a kódolás, fejlesztés során. Mind a Codeium [8], mind a Github Copilot [9] szoftverfejlesztésre hangolt, előbbi közvetlenül a betáplált tudása alapján ajánl megoldást, míg a Copilot a Github összes publikus törzskönyvtárain tanult, így élesben használt kódokkal ad visszajelzést különböző megoldásokra. Szakdolgozati munkám megvalósítása során ezt a kiegészítőt is alkalmaztam, mivel a mostani világban egyre jobban kezd elterjedni a használata gyors, és kézenfekvőbb opciója miatt. A szoftverfejlesztésem során nagy hangsúlyt kapott a hibák kezelésében, javításában, és lényegesebb gyorsabb választ adott a kérdéseimre, mint a keresőmotorok képesek lettek volna. Emellett fel lehetett használni őket ötletek, alternatívák keresésére, egyszerű programrészek megírására, vagy akár programkód dokumentációjára is. És bár nem tökéletes, többnyire csak magasabb szintű probléma esetében kerestem válaszokért. A Codeium teljesen ingyenes, a Github Copilot pedig diákok és tanárok számára ingyenes használatot biztosít Github Education platformján keresztül [10].

Adatbázis: SQLite

A nagy mennyiségégű adathoz és azok tároláshoz egy olyan adatbázist kerestem, amely gyors műveleteket és kompatibilitást biztosít ASP.NET-hez anélkül, hogy plusz extra szoftvert kellene használni, vagy külön szervert kellene biztosítani hozzá. Jártas voltam már az SQL-ben így keresésem folyamán találtam meg a szintén nyílt forráskódú relációs adatmodellű SQLite adatbázist, amelynek használata az integráláson kívül nem változott.

DBeaver

Mivel a megvalósítandó weboldalam tartalmaz külön adatbázist, szükségem volt egy olyan alkalmazásra, amellyel azt gyorsan elérhetem és emellett különféle szerkesztési és

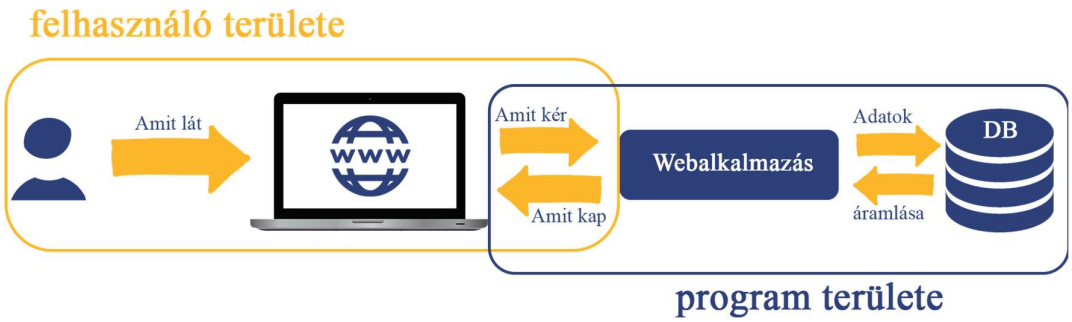
lekérdezési lehetőséget biztosít. Eleinte Pycharmban, vagy kezdetlegesen VS Code-ban talált kiegészítőkkel próbálkoztam elérni, de később külső konzulensem javaslatára a DBeaver-t [11] választottam, amely az igényeimet teljesen kielégítette. A DBeaver nyílt forráskódú, szabad felhasználással rendelkező szoftver, amely többféle adatbázis típust tud kezelni egyszerre, s amelynél az adatbázis kapcsolatának létrehozása varázsló segítségével egyszerűen megoldható.

PyCharm (Python)

A PyCharm [12] a Jetbrains cég által biztosított Python IDE, amely egyszerűsíti a nyelv használatát és gyorsítja a fejlesztés ütemét többféle automatikusan beépített kiegészítőkkel, gyors szintaktikai, néhány esetben szemantikai hibák orvoslásával, vagy épp kiegészítőkkel. Ez a verzió leginkább Python programkód írására szakosodott, de képes kezelni más formátumú nyelveket is. Kétféle verziójából a Pro verziót használtam, mivel a vállalat ugyancsak biztosít diákoknak és tanároknak ingyenes szoftver-hozzáférést komplett szoftvercsomaghoz, többek között ehhez a szoftverhez is [13].

3. RENDSZERTERV

A megvalósítandó webalkalmazásom felépítését a 3.1. ábra mutatja be.



3.1. ábra A webalkalmazás egyszerűsített felépítése

Ahogy az ábrán látható, két nagyobb részre tudjuk osztani az alkalmazás részeit: a felhasználó területére és a program, vagyis a webalkalmazás területére. A webalkalmazás további három elemre osztható, amelyek meghatározzák mivel kell számolni egy alkalmazás fejlesztésénél: kinézet, webalkalmazás és az adatbázis. Ezekhez az elemekhez szükséges a megfelelő eszközök kiválasztása és használata. Az alkalmazás csak akkor jó, ha az adatgyűjteménye is hasznos, és helyes a felépítése. Az első feladatom volt tehát az adatbázis létrehozása.

3.1 Adatbázis terv

Az adatkészlet elemeit a 2.2 alfejezetben meghatározott forrásoldal, az Árfigyelő API kéréseinek felhasználásával hoztam létre. Az adatokat egy jól elérhető relációs modellű SQLite adatbázisban tároltam a következő formában:

- Termékek és hozzá tartozó információik,
- Üzletek és hozzájuk kapcsolódó élelmiszerláncaik,
- Élelmiszerláncokként található árak,
- Termékek elérhetősége üzletenként,
- Árváltozások követése.

A lekért értékes információkat az alábbi adatcsomag tábláiba helyeztem be a kellő helyre:

Termékek

Termék táblát kell feltölteni az API termékre eső kérésére, amely csak a szükséges elemeket tartalmazza:

- **Termék Azonosító**: amely szöveg formátumú egyedi kulcsként azonosítja a táblát,
- **Termék Név**: amely redundáns elemek nélkül helyezi el szövegként,
- **Termék Kép**: amely az API-ben szereplő URL címét menti el szövegként,
- **Termék Egység**: amely a termék mennyiségi vételegységét tárolja el lebegőpontos elemként,
- **Termék Egység Típus**: amely a termék mennyiségi vétel-mértékegységét tárolja el szövegként,
- **Termék Egységnyi Mennyisége:** amely a termék egy vételnél egy csomagban található darabszámát határozza meg egész szám formátumban.
- **Termék Kategória:** amely a termék már előre bekategorizált helyét adja meg, többszintű elrendezéssel szövegben.

Üzletek

Az Üzlet tábla kell, hogy tartalmazza az országban található nagy élelmiszerláncok (Spar, Lidl, Aldi, Penny, Auchan, Tesco) üzleteit és paramétereit az API üzletekre eső lekérésével, amelyek a következők:

- **Üzlet Azonosító:** egyedi elsődleges kulcs az üzletekhez szövegként,
- **Üzletlánc Azonosító:** a megadott üzlet élelmiszerlánchoz kötése ezen belül történik meg, szövegformátumban,
- **Irányítószám:** Az üzlet helyzetének megyei irányítószáma egész számként,
- **Település:** Az üzlet helyzetének szöveges elhelyezése,
- **Cím:** Az üzlet településen belül található helyzetének szöveges tárolása,

- **Latitúdó:** Térképen pontos elhelyezése, szélességi irány lebegőpontos számaként,
- **Disztancia:** Térképen pontos elhelyezése, hosszúsági irány lebegőpontos számaként,
- **Nyitvatartás:** Az üzlet heti nyitvatartási rendjének szöveges tárolása.

Élelmiszerláncok

Az Élelmiszerlánc táblában kell eltárolni az API élelmiszerláncokra eső lekérését a következő információkkal:

- **Élelmiszerlánc azonosító:** Tábla egyedi kulcsa azonosítja az élelmiszerláncot szöveges formátumban,
- **Élelmiszerlánc neve:** Az adott élelmiszerlánc neve, szöveges formátumban.

Árak üzletenként

Ide kerülnek a Termék tábla kiválasztása során lehívott termékek vásárlási ár típusai, amelyek a következők szerint mutatkoznak meg:

- **Termék Azonosító:** Termék beazonosítására szolgál,
- **Élelmiszerlánc Azonosító:** Az adott termék élelmiszerlánca,
- **Ár Típus:** Termék adott élelmiszerláncához kötött ár típus,
- **Termék Vásárlási Ár:** Az adott termékhez tartozó vásárlási ára,
- **Termék Egységár:** Az adott termékhez tartozó vásárlási egységára,
- **Azonos Egység Mennyiség:** Az adott termék egyforma egységben megvásárolható igaz-hamis értéke.

Termékek üzletenkénti elérhetősége

Az előzőhöz hasonlóan ide kerülnek a Termék tábla során lehívott termékek boltjai:

- **Termék Azonosító:** Termék beazonosítására szolgál,
- **Bolt azonosító:** Az adott bolt azonosítója.

Termékár története

E területen belül kötelező megjeleníteni a következőket:

- **Termék Azonosító:** Termék beazonosítására szolgál (Idegen kulcs),
- **Dátum**: A bejegyzés időpontja szöveges formátumban,
- **Maximum érték-e,** amely logikai (igaz [1]; hamis [0]) értéket vesz fel,
- **Adott ár**, amely adott nap maximum vagy minimum értékét tárolja el lebegőpontos formátumban.

Termék információk

A Termék információ táblát az Open Food Facts [14] adatbázisából kell feltölteni elsődlegesen. Ez egy nyílt forráskódú és adatbázisú, nonprofit szervezet, amelynek oldala az egész világból tartalmaz ételeket, illetve hozzájuk lényeges kapcsolódó adatokat, például összetevőit, energia-, tápanyag-táblázatukat, minőségét, és akár karbonlábnyomukat is. A termék információk tábla a következő oszlopokat kell, hogy tartalmazza:

- **Termék Azonosító:** Termék beazonosítására szolgál (Idegen kulcs),
- **Termék Összetevői,**
- **Tápérték táblázat** elemei 100 grammra leosztva, mint:
    - **Kilo Joule,**
    - **Kilo kalória,**
    - **Zsír**, amelyből
        - **Transzzsír,**
        - **Telített zsírsavak,**
        - **Telítettlen zsírsavak**, amelyből:
            - **Egyszeresen telítetlen**,
            - **Többszörösen telítetlen**,
    - **Szénhidrát**, amelyből
        - **Cukor**, amelyből
            - **Hozzáadott cukor**,
    - **Rostok,**

- **Só**, amelyből
  - **Hozzáadott só**,
- **Fehérje,**
- **Laktóztartalom**

Kapcsolatok

Adatbázis táblái közötti kapcsolatok meg kell, hogy jelenjenek az adatbázisban, amelyet ellenőrizni kell a hozzáírt algoritmus használata során. Ezen helyen szerepeltetni szükséges az adatbázis létezésének ellenőrzését, majd létrehozását hiány esetén, amellett, hogy a használt táblákat is elérhető állapotban tartsa. Az adatbázis kapcsolatai létrejöttének egyrészt az algoritmus kell, hogy feleljen, másrészt az implementálása során a Webalkalmazás is le kell, hogy kezelje.

A kapcsolatok felépítése a következő:

Termékek (Elsődleges kulcs):

- **Termék információ**: Egy az egyhez (idegen kulcs, 1-1)
- **Termékár története**: Egy a többhöz (idegen kulcs, 1-Több)
- **Termékek üzletenkénti elérhetősége**: Egy a többhöz (idegen kulcs, 1-Több)
- **Termékár élelmiszerláncokként**: Egy a többhöz (idegen kulcs, 1-Több)
- **Árak üzletenként**: Egy a többhöz (idegen kulcs, 1-Több)

Üzletek (Elsődleges kulcs):

- **Termékek üzletenkénti elérhetősége**: Egy a többhöz (idegen kulcs, 1-Több)
- **Termékár élelmiszerláncokként**: Egy a többhöz (idegen kulcs, 1-Több)

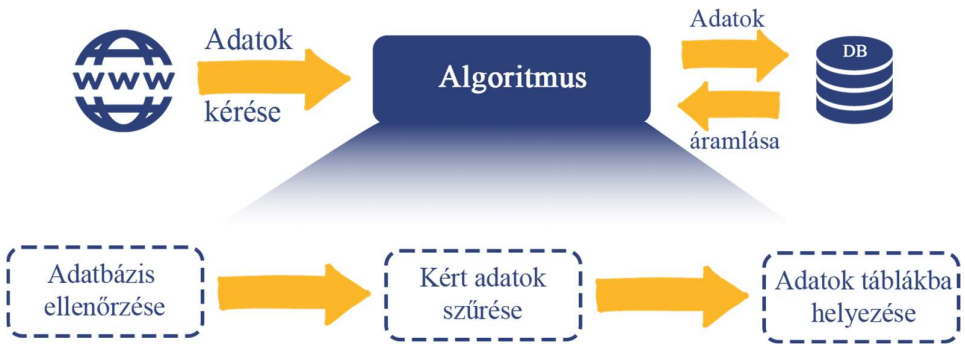
Élelmiszerláncok (Elsődleges kulcs):

- **Termékek üzletenkénti elérhetősége**: Egy a többhöz (idegen kulcs, 1-Több)
- **Üzletek**: Egy a többhöz (idegen kulcs, 1-Több)

A táblák feltöltéséért külön-külön algoritmusok felelnek, amelyeket a következő alfejezetben mutatok be.

3.2 Algoritmusok

Az elkészítendő algoritmusok fő funkcióit és az adatok áramlását a 3.2. ábra foglalja össze.



3.2. ábra Algoritmus elem részei

Az adatbázis adatainak betöltését Python nyelvvel végzem. Ehhez az adatbázisba behelyezés előtt le kell ellenőrizni, hogy az adatbázis kész-e a befogadásra. Ellenőrzés után a weboldal API kéréseit Pandas DataFrame [15] formába átalakítja. A DataFrame a Pandas csomag által elérhető tárolási forma, amely táblaszerű elhelyezést tesz lehetővé különféle típusú adatokhoz. Az adatokat később más formátumba átalakítva átformálja párhuzamos adatok elkerülése érdekében, csökkentve a redundáns elemek szereplését. Fontos szempont a működése során a reszponzív, gyors működés.

A kész adatokkal végül SQLite csomag felhasználásával műveleteket végez az adatbázison, attól függően, hogy elhelyezést, frissítést, vagy törlést indokol az adott helyzet. Adatok elhelyezése után az adatbázist felhasználó webalkalmazás fejlesztése következik.

A továbbiakban bemutatom az adatlekérések eredményét és a tárolandó adatokat, valamint azok formáját.

Termékek


```
{
  "products": [
    {
      "id": "2101770000004",
      "name": "Trappista tömbsajt Ft/kg",
      "imageUrl": "https://cdnarfigyeloprodweu.azureedge.net/images/product-image-2101770000004.jpg",
      "unit": "kg",
      "unitTitle": "db",
      "bulk": true,
      "packaging": "1",
      "categoryPath": "tejtermek_sajt_tojas/tombsajt/trapista_tombsajt",
      "minUnitPrice": 1949,
      "pricesOfChainStores": [
```


3.3. ábra Termékek lekérésének egy szelete

Az adatokból következőket kell tárolni:

„id” „imageUrl” „unitTitle”

„name” „unit” „categoryPath”

Az adatok egyszeri tárolásúak, ezért azonosítóval való beillesztés lehetséges. Frissítés nem szükséges, csak adatbevitel.

Üzletek


```
{
  "uuid": "lidl-316",
  "postalCode": "1027",
  "city": "Budapest",
  "address": "Csalogány u. 43",
  "chainStoreUuid": "f95031d8-84a5-4edf-b4fa-18b2cbae2ea1",
  "location": {
    "latitude": 47.50813,
    "longitude": 19.03156
  },
  "openingTime": "Nyitvatartás: 06:30-21:30#Nyitvatartás - Vasárnap: 07:00-19:00"
},
```


3.4. ábra Üzletek lekérésének egy szelete

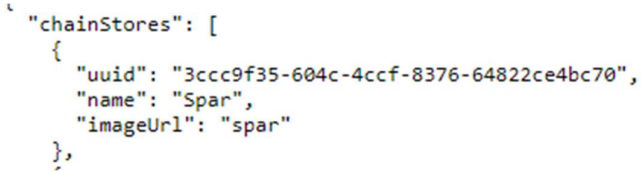
Az adatokból következőket kell tárolni:

„uuid” „postalcode” „address” „latitude”

„chainstoreUuid” „city” „openingTime” „longitude”

Az adatok hasonlóan a termékeknél, egyszeri tárolásúak, ezért azonosítóval való beillesztés lehetséges. Frissítés nem szükséges, csak maga az adatbevitel.

Élelmiszerláncok



3.5. ábra Az élelmiszerláncok egy szelete

Az adatok hasonlóan az üzletekhez, egyszeri tárolásúak, ezért azonosítóval való beillesztés lehetséges. Frissítés nem szükséges, csak maga az adatbevitel. Használt elemek a „uuid” és a „name”.

Árak üzletenként

3.6. ábra A 3.3. ábra „pricesOfChainStores” egysége

Ez a tábla több utasítást igényel: az áruk árai, árának típusai naponta változnak, ezáltal a darabszámuk is, amelyhez igazodnia kell. Többek között úgy kell belehelyezni a táblába az adatokat, hogy törölni kell a már elavult ártípusokat, ha a napi frissítés az adott terméknél nem mutat ennek létezésére, illetve frissítenie kell az árakat, ártípustól függetlenül. Egyedi kulcsa nincs, maximum a Sor azonosítója lehet az Entity Framework számára.

Az adatok, amelyeket szükséges tárolni:

„id”	„type”	„unitAmount”
„name”	„amount	„sameAmountEverywhere”

Termékek üzletenkénti elérhetősége


```
"id": "5998207700425",
"name": "Dunatej UHT tej 2,8%",
"imageUrl": "https://cdnarfigyeloprodweu.azureedge.r
"unit": "l",
"unitTitle": "db",
"bulk": false,
"packaging": "1",
"categoryId": "3",
"categoryPath": "tejtermek_sajt_tojas/tej/uht_28",
"chainStores": [
  {
    "uuid": "f95031d8-84a5-4edf-b4fa-18b2cbae2ea1",
    "name": "Lidl",
    "imageUrl": "lidl",
    "prices": [
      {
        "type": "NORMAL",
        "amount": 229,
        "unitAmount": 229
      }
    ],
    "pricesMax": [],
    "priceSameEverywhere": true,
    "availableInShops": [
      "lidl-102",
      "lidl-103",
      "lidl-104",
      "lidl-106",
      "lidl-107",
      "lidl-110",
```


3.7. ábra Példa a 'Dunatej UHT tej 2.8%'-os termék API kérésének eredményéből

A termék elérhető üzletei szintén változhatnak, ellenben mivel nagysága nagy (akár 1000+ is lehet az üzletszám), ezért a tárolására egyedi kulcs hiányában az „id”, „storeUuid” lesz alkalmas, illetve sor Azonosító, ahogy azt az Árak üzletenként megadásánál említettem.

Termékár története

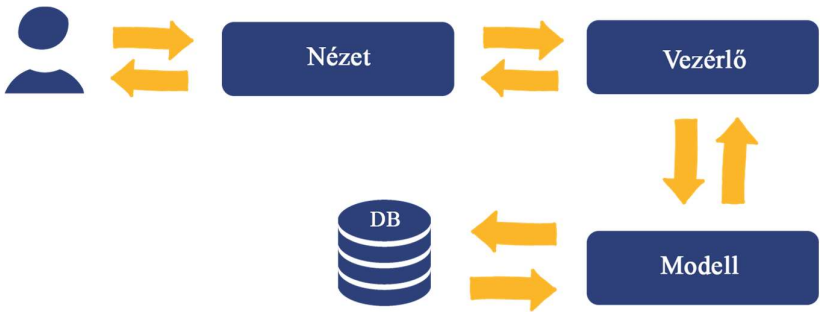
A Termékár története tábla tartalma új adatként a napi frissítések maximum-minimum árát tárolja, amelyet dátummal fog kiegészíteni későbbi grafikon megjelenítésére.

Termék információk

A Termék információ tábla a megadott forrásnak azon elemeit kell tárolnia, amelyeket megemlítettem az Adatbázis tervben. Egyszeri bevitelről van szó, és mivel a kulcsa a Termék azonosítója lesz, így frissítés nem szükséges az adatokon többszöri futtatás során.

3.3 Webalkalmazás

A tervezett webalkalmazásom egyszerű felépítését mutatja a 3.8. ábra.



3.8. ábra Webalkalmazás egyszerűsített felépítése

A webalkalmazás létrehozását Visual Studio segítségével terveztem, melyben az oldalak ASP.NET formában, MVC elven jönnek létre. Az alábbi legfrissebb csomagokat kell felhasználni a fejlesztés során:

- *'Microsoft.EntityFrameworkCore'* 8.0+ verzióval,
- *'Microsoft.EntityFrameworkCore.Sqlite'* 8.0+ verzióval,
- *'Microsoft.EntityFrameworkCore.SqlServer'* 8.0+ verzióval,
- *'Microsoft.EntityFrameworkCore.Tools'* 8.0+ verzióval,
- *'Microsoft.VisualStudio.Web.CodeGeneration.Design'* 8.0+ verzióval.

A nézetben szerepelniük kell a felhasználó számára interakcióra képes elemeknek. Ezen területen belül kell, hogy megjelenítse az adatbázis elemeit, amely rendelkezik olyan funkciókkal, mint:

- Termékek szűrése, keresése, például tulajdonság vagy megadott távolság alapján,
- Lapozási funkció az adott terméklistára.

A nézetet létrehozó oldalak mindegyikét a saját kontrollere kell, hogy irányítsa, feldolgozza, melynek adatait az adatbázissal összekötött modellek kell, hogy adják. Mindezen adatokat a kontroller csak olvasásra használhatja fel, így a weboldal nem ad lehetőséget szerkesztésre, csak adatok manipulálására.

Az alábbi oldalak szükségesek a webalkalmazás létrejöttéhez:

- **Főoldal**, amely bemutatja az oldalt,

- **Termékek**, ahol a teljes termékpaletta megtalálható, keresési-, kosárhoz hozzáadó funkcióval, melynek eredményét lapozási lehetőséggel adja vissza.
- **Keresés**, amely a főoldalról, vagy a saját oldaláról irányított keresés funkciókkal rendelkezik, melynek eredményét lapozási lehetőséggel adja vissza.
- **Kosár**, ahol a kiválasztott termékek jelennek meg a kosárhoz tartozó információkkal.

Az oldal helyes működése érdekében szükséges használni JavaScript elemeket is, amelyek reszponzívan kell, hogy működjenek. A nézet végleges kinézetét HTML5 és CSS használata fogja adni, melyeket a felhasználó számára érthető elemekkel kell kiegészíteni, hasonlóan a nézetben megtalálható ASP irányítóegységekkel és más komponensekkel, amelyeket az ideiglenes nézet tartalmaz.

4. AZ ADATBÁZIS

A webalkalmazáshoz tehát először egy adatbázist kell létrehozni, mivel az alkalmazás abból meríti az adatokat működése során. A rendszerterv szerint meghatározott oldalról szerzett adatok alapján ezt fogja feldolgozni a külön-külön API kérésekre megírt algoritmus, majd gyűjti össze, rakja megfelelő helyre, illetve a létező elemeket pedig frissíti vagy törli a determinált funkció által. Minden művelet Python kódban, vagy Pythont és csomagjait felhasználva valósult meg. A kódok eredményét egy külön fájlban tárolva, az adatbázis kezelését pedig egy osztályon keresztül valósítottam meg, melyben objektumként kezeli az adatbázist és tölti fel, vagy egészíti ki.

4.1 Előzetes függvények

A lekéréseket megelőzték a felépített metódusok, amelyek két részre, adatbázisra és weboldal lekérésekre oszlottak.

Árfigyelő metóduslap

API lekérésre külön-külön hoztam létre metódusokat, hogy az elemeket szűrve adja vissza a használathoz. Minden API meghívás egy JSON formátumú szöveget ad vissza, szótárként. Ezeket a szövegeket szükség szerint szűrtem, és azt küldtem vissza Pandas DataFrame formátumba, vagy szűrt listába, attól függően, hogy milyen méretű adatról beszélünk. Ennek a csomagnak a használata egy ideiglenes konverziót tett lehetővé.

```
def get_products(category, limit=5000):
    try:
        response = requests.get(
            "https://arfigyelo.gvh.hu/api/products-by-category/" + str(category),
            params={'limit': limit})
        data_list = json.loads(response.text)
        return pd.DataFrame(data_list)
    except requests.exceptions.RequestException:
        print(f"Request failed")
```

4.1. ábra Pandas példa: A termékek megszerzése kategóriánkként

Eleinte ezek a függvények rendelkeztek osztály szinttel, de utólag átgondolva kivettem az osztályból, mivel nem volt szükség objektumok létrehozására, linkek eléréséhez és visszaküldéséhez. A továbbiakban a szövegek az adatbázisban lesznek tovább formázva.

Adatbázis Osztálylap

A program e területe már objektumként kezeli az adatbázist, a bekért elérés helyével. Ezután az adatbázist a metódusokban található módszerek alapján fogja feltölteni. Először megvizsgálja, hogy létezik-e egyáltalán az adott elérésen a fájl. Ha nem szerepel, akkor létrehoz egyet megadott néven, majd feltölti a szükséges táblákkal, melyeket szintén egyesével leellenőriz, hogy már szerepelnek-e benne, vagy sem. Előfordulhat például, hogy az adott táblákból csak egy hiányzik, így ki tudja javítani, és használhatóvá válik a program számára. Minden táblát úgy hoz létre, ahogy a Visual Studio NuGet csomagja létrehozza a migrálás, majd adatbázis frissítés után. Ezeket a táblákat ezután fixálja a kapcsolatán keresztül. A táblák feltöltését később a program fogja meghívni.

```
def update_tables(self):
    """
    A function to update tables in the database with predefined schema.
    """
    self.conn = sql.connect(self.filename)
    self.cur = self.conn.cursor()
    if not table_exists(self.cur, table_name: 'Products'):...

    if not table_exists(self.cur, table_name: 'ProductInformations'):...

    if not table_exists(self.cur, table_name: 'ChainStores'):...

    if not table_exists(self.cur, table_name: 'Stores'):...

    if not table_exists(self.cur, table_name: 'PriceAtStores'):...

    if not table_exists(self.cur, table_name: 'AvailableStores'):...

    if not table_exists(self.cur, table_name: 'PriceHistories'):...

    self.conn.commit()
```

4.2. ábra Táblák ellenőrzése

4.2 Táblák meghatározása, feltöltése

A táblákat a forrásoldalon talált adatokkal töltöttem fel az alábbi kód segítségével:

```
def __init__():
    start = time.time()
    with Database(filename) as db:
        db.create_tables()
        db.insert_table_chainstores(get_chainstores())
        for category in get_categories():
            df = get_products(category)
            prod_id = []
            for i in df['products']:
                prod_id.append(i['id'])
                output = db.insert_table_products(i)
                output_str = f"Output Products: (Database, Console): {output}"
                print(output_str)
            with concurrent.futures.ThreadPoolExecutor() as executor:
                # Submit the tasks and store the future objects
                futures = [executor.submit(get_available_stores_by_product, product) for product in prod_id]
                # Retrieve the results from the future objects
                available_stores = [future.result() for future in futures]
            prod_id = [prod_id, available_stores]
            for i in range(0, len(prod_id[0])):
                output = db.insert_table_available_stores(prod_id[0][i], prod_id[1][i])
                for shop in output[1]:
                    print(f"Output Available Stores: (Database, Console): {output[0]}: {shop}")
        for store in get_stores():
            output = db.insert_table_stores(store)
            print(f"Output Stores: (Database, Console): {output}")
    stop = time.time()
    print(f"Runtime: {stop - start} seconds")
```

4.3. ábra A tábla teljes feltöltése

A program először előhívja a fájlnévben megadott adatbázis elérését és leellenőrzi a szükséges táblákat. Ha nem találja bármelyiket is, létrehozza azt. Következő lépésként összegyűjti a kategória API által megadott kategória azonosítókat, majd azon keresztül lekéri az összes terméket. Ezeket az árukat behelyezi egy listába, amelyen végig futva behelyezi a saját táblájába amellett, hogy *PriceAtStores*-t és *PriceHistories*-t is feltölti ezáltal. Miután behelyezte, parallel lekéri a termékek oldalait, hogy megkapja, melyik üzletben érhető el, majd elhelyezi azokat az *AvailableStores* táblába. Legvégén pedig beépíti az összes üzletet a táblába. A következő részek megmutatják, miként működtek a beépítések táblánkként.

Product

Az API kérésekből a megadott paraméterek a begyűjtés után, könnyen elérhető formátumban vannak, amit 3.3. ábra is bemutat.

Ezekre az adatokra olyan logikát kellett választani, amely eleget tudott tenni annak, hogy redundancia nélkül tárolhatóak legyenek. Az API-okon keresztül elérhetőek voltak az üzletek, a termékek, a kategóriák, amelyeket a mennyiségük és megjelenésük folytán kell eltárolni.

A ’Products’ tábla tartalmazza az üzletekben található termékeket, amelynek termékazonosítója lett az elsődleges kulcs, amellett tartalmazza a termékek nevét, jelenlegi legkisebb beszerezhető egység-vételárát, weboldalon megjelenítendő kép URL címét, űrmérték típusát, termék eladási formáját, űrmértékét, kategóriáját. A képek minőségét elegendőnek találtam, ezért helyspórolás szempontjából csak az URL-t tároltam el.

A termék nevét ellenben szerkesztenem kellett több okból kifolyólag:

- Feleslegesen tartalmazott plusz információkat, mint az adatbázis felépítése során névben megjelenő azonosítószám, amely nem egyezik az eredeti kulccsal.
- Többször előforduló mértékegységek, amelyek egyébként is adatbázisban szerepelnek.
- Szöveghibák, nyelvtani hibás- vagy helytelen termékelnevezések, gyártói eltérések.
- Tisztább, felhasználó számára szebb, letisztultabb megjelenés.

Ezek az elnevezések akkor kerültek közvetlenül szerkesztésre, amikor adatbázis létrehozása történt. Ez esetben legalább 2000+ elemről beszélünk, amelyeket (tekintve méretüket) egyesével szerkeszteni fáradságos, ezért hatékonyabb egyszerre az összeset egy vagy több szabállyal meghatározni. Szükség volt tehát egy olyan eszközre, amely képes a szövegeket átformálni egy megadott szabály alapján.

Ilyen eszköz az úgynevezett Regex. Angolul Regular Expressions (reguláris kifejezések) a bővebb jelentése, amely szövegre szabható szabályokat hoz létre egy megadott minta alapján. Kereshetünk karaktertípus, ismétlődés vagy szövegben megadott pozícióval, csak hogy pár példát említsek. Ebből is többféle formátum létezik, melyet a programnyelv dönt el, hogy fogad el. Sokféle felhasználási módja létezik, csoportosító funkciója is ezt erősíti, amelynél egy sémában több szabályt is alkalmazhatunk. Tartalmaz mutatókat is,

amellyel magát a szerkesztendő szöveget azonosítjuk a regex számára. Ilyen lehet a többsoros szöveg, szöveg kódolása és további beállítás. Érdemes viszont az írása során egy logikát kigondolni, amelyet egy regex szerkesztő oldalon, a Regex101 [16] felületén egyszerű letesztelni. Élőben leképezi a különböző mintákat a megadott forrásszövegre, és megmutatja, hogy milyen sorrendben mit talál, illetve a végeredményt is megmutatja, ha helyettesíteni szeretnénk az elkapott szövegre. Interaktivitása mellett kiválasztható több regex forma különböző programnyelvekből, melynek elemeit egy referencia ablakban megmutatja, hogy mire hogyan képes mintát létrehozni.

A 4.4. ábra mutatja be a készített regex sémákat. A szövegem alapján több, különböző mintát kellett használnom, ahol egy mintát adtam meg, amit szeretnék felismerni, és kijelölni, amit képes vagyok felhasználni, vagy cserélni, az esetemben pedig törölni. Sémát ellenben a weboldalt értelmezi, illetve kiírja az elkapott szövegrészeket és csoportokat. Összesen 6 mintát használtam minden egyes szöveghez.

```
def clean_product_name(product_name):
    """
    Removes numeric and special characters from the product name
    and returns the cleaned product name.
    """
    product_name = re.sub( pattern: r'\b(\d*[,.]?\d*-?\d*[,xX./]?\d+\s?'
                           r'(?:darab|db|DB|G|g|GR|kg|KG|l|L|ml|ML)(?:-os)?)\b',
                           repl: "",
                           product_name)
    product_name = re.sub( pattern: r'(\d+\s?x\d*|\w+\d+$|Ft/kg|Ft/db)', repl: "", product_name)
    product_name = re.sub( pattern: r'(\(\s*\w*\W*\)|\(.*|kg|db)', repl: "", product_name)
    product_name = re.sub( pattern: r'(,?\s*TZ,?)', repl: "", product_name)
    product_name = re.sub( pattern: r'( {2,})', repl: " ", product_name)
    product_name = re.sub( pattern: r'([ ,]+$)', repl: "", product_name)
    return product_name
```

4.4. ábra Regex sémák

Ezen minták csökkentett, leginkább hibákkal előforduló, élő szövegmintákkal lettek formálva, vagyis további változás esetén több séma fejlesztése, vagy átírásra szükség lehet.

Egy példaszövegen keresztül a 4.5. ábra mutatja be a regex hatását az elkészített különböző sémákkal.

’ UHT Tej 1,5%, (1257912), 1l, Ft/kg’

Regex	*Eredmény*	*Értelmezés*
1. séma	’UHT Tej, (1257912), , Ft/kg’	Megszünteti az extra mértékegységeket és azok értékeit.
2. séma	’UHT Tej 1.5%, (1257912), , ’	Kiszedi a felesleges darab és mennyiség értékeket.
3. séma	’UHT Tej 1,5%, , ,’	Eltávolítja a nem oda illő zárójeles elemeket.
4. séma	’UHT Tej 1,5%, , ,’	Egy bizonyos ismétlődő elemeket kitöröl.
5. séma	’UHT Tej 1,5%, , ,’	Cseréli az egynél többször szereplő szóközöket egyre.
6. séma	’UHT Tej 1,5%’	A terméknév végén található szóközöket és vesszőket teljesen kiszedi.

4.5. ábra A regex hatása a szövegre több sémán keresztül

Ezen változtatások kiküszöbölték manuális szövegszerkesztések nagy részét, így már csak a hiányos elnevezéssel rendelkező elemeket kellett egyedül kézzel szerkeszteni, kiegészíteni.

Maguk a termékek több üzletben kaphatóak más-más árakon, melyeket a következő rész taglal.

Price At Stores

A ’Price At Stores’ táblából lett közvetlenül felhasználva, mely termék mely élelmiszerláncokban érhető el. A táblában a termékek különböző élelmiszerláncokban elérhetőségét tárolja amellett, hogy tartalmazza a vételárát, a mennyiségi árát, azonos-e a vételár az

adott láncon belül, illetve, hogy a jelenleg kapható áru normál, akciós, vagy hűségprogram által szerezhető be.

Mérete körülbelül 5-10%-ban változhat a ’Products’ táblához képest, tehát a műveletek nagysága is majdnem ugyanakkora.

Mivel minden nap változhat a napi kedvezmény, tárolni úgy kellett, hogy az aktuális bemenetet ellenőrizze a behelyezés előtt, hogy szerepel már benne, vagy sem. Bár elsődleges kulcs van benne a rekord ID-ért, ’UPSERT’ művelet nélkül vizsgáltam meg a következőképp:

```
self.cur.execute( _sql: """
DELETE FROM PriceAtStores
WHERE ProductId = ? AND StoreChainUuid = ? AND PriceType NOT IN (
    SELECT tpas.PriceType
    FROM PriceAtStores tpas
    WHERE tpas.ProductId = ? AND tpas.StoreChainUuid = ?
)
""",  _parameters: (product['id'], store['id'], product['id'], store['id']))
```

4.6. ábra Kapott értékek megvizsgálása az adatbázisban

A képen látható, hogy egy SQL lekérdezéssel megvizsgáltam, hogy tartalmazza-e a jelenlegi kéréseket, és csak azokat törölje, amelyeknél az adott üzletlánc adott terméknél különbség van. Ha ezzel végzett, a jelenlegi értékeket vagy frissíti az adatbázisban, vagy beépíti, ha nem talált frissíthető elemet.

Stores

A ’Stores’ tábla az élelmiszerláncok Magyarország teljes területén elérhető áruházainak listáját tárolja, amely tartalmazza a saját és üzletláncának ID-ját, pontos helyzetét a térképen belül distanciában, és latitúdóban (későbbiekben fontos elemként szolgálnak), emellett a települést, az üzlet elérését és nyitvatartását, ha van. Mivel ennek van egyedi kulcsa, behelyezése és ellenőrzése nem kívánt nagy erőfeszítést, az egy szimpla elemlekéréssel, majd ’UPSERT’ művelettel történt.

```
def insert_table_stores(self, store):
    self.cur.execute(f"SELECT * FROM Stores WHERE storeUuid = '{store[0]}'")
    row = self.cur.fetchone()
    if row:
        if any(row[i] ≠ store[i] for i in range(1, len(row))):
            self.cur.execute( _sql: "UPDATE Stores SET "
                              "postalCode = ?, "
                              "city = ?, "
                              "address = ?, "
                              "latitude = ?, "
                              "longitude = ?, "
                              "openingHours = ? WHERE storeUuid = ?",
                              _parameters: (store[2], store[3], store[4], store[5],
                               store[6], store[7], store[0]))
    else:
        self.cur.execute( _sql: "INSERT INTO Stores VALUES(?,?,?,?,?,?,?,?)", store)
    self.conn.commit()
    return store
```

4.7. ábra Stores tábla feltöltése

Available Stores

Az ’Available Stores’ táblában találjuk meg egyesével, hogy mely termékek érhetők el más-más áruházban. A termékek naponta változnak, és ezáltal az elérhetőségük is. Ennek felépítése külön-külön termékek oldalán volt elérhető szótárként, egy változó alá belezsúfolva, tehát az elhelyezését át kellett gondolni. Több mint 1200 üzletről beszélünk, amelynek tárolása semmiképp sem lehet egy mátrix, üzletre és termékre leosztva, hiszen az teljesen átláthatatlanná tenné. Egyik egyszerű megoldás erre egy termékazonosítót egy szöveghez csatolni, melybe az üzleteket vesszővel elválasztva tárolja. Ez a megoldás problémákat okozott volna később, mivel felesleges feldolgozási műveletekkel lehet csak hozzáférni az adatokhoz, ami közel sem optimális egy ilyen feladathoz. Más megközelítéssel dolgoztam: létrehoztam egy termék- és üzletazonosítójú oszlopot, majd hozzátettem egy elérhetőségi tulajdonságként még egy oszlopot. Ezzel optimálissá tettem a tárolását kevesebb művelettel, és lehetőség szerint csak frissítéseket kellett használnom.

A feltöltés teljes mértékben működött, egy dolgot leszámítva: az eddigi 2 perces futási idő 3-4 órává nőtte ki magát. Az eddigi logika alapján a feltöltésnél megkapta a termék oldalát, megfelelő adatformátumba kezelte az adatokat, végignézte, hogy léteznek-e, s ha igen, frissítette az elérhetőségét, egyébként pedig behelyezte.

Ez kisebb, egyedi azonosítóval rendelkező tábláknál nem volt probléma: azonban ennél a táblánál a kapcsolat több az egyhez típusú! Fel kellett mérnem tehát, hogy mely műveletek azok, amely nagy időt vesznek el feleslegesen, illetve a többszálú szerkesztések (párhuzamos műveletek) sem feltétlen működnek adatbázis szerkesztésénél.

Alkalmazva a műveletek szeparálását, először összegyűjtöttem párhuzamos feldolgozással az adott termékek oldalait. Ezután egy ciklusba szervezve meghívtam egy metódust egy termékkel és az üzleteivel. Gondolva arra, hogy változhat egy termék előfordulása, egy 'UPDATE' műveletet hoztam létre ezekre az esetekre.

```
available_shop = [shop for store in available_stores['chainStores'] for shop in
                  store['availableInShops']]

self.cur.execute( _sql: f"""
            UPDATE AvailableStores SET available = 0
            WHERE productId = ?
            AND storeUuid NOT IN ({', '.join(['?'] * (len(available_shop)))})
        """, [product_id] + available_shop)

for shop in available_shop:
    self.cur.execute( _sql: "SELECT 1 FROM AvailableStores "
                      "WHERE productId = ? AND storeUuid = ?",
                      _parameters: (product_id, shop))
    row = self.cur.fetchone()
    if row:
        if row[0] == 0:
            update_shops.append((product_id, shop))
    else:
        insert_shops.append((product_id, shop))
if update_shops:
    self.cur.executemany( _sql: "UPDATE AvailableStores SET available = 1 "
                          "WHERE productId = ? AND storeUuid = ? AND available = 0",
                          update_shops)
if insert_shops:
    self.cur.executemany( _sql: "INSERT INTO AvailableStores "
                          "(productId, storeUuid, available) VALUES (?, ?, 1)",
                          insert_shops)
```

4.8. ábra A feltöltés első módszere

A paraméterben létrehoz egy 'szöveg' sort, melybe behelyettesíti az összes üzletet elemként, és a futtatás során minden olyan termék-üzlet párost frissít, amire ez igaz. Ezután a kapott üzleteket két listára szedi szét: amelyeket frissítésre és behelyezésre szán azáltal, hogy mely párosokat találta meg, vagy sem. Legvégül lefuttatja azokat az elkészített listájuk szerint.

A módszer gyorsasága szembetűnő volt: körülbelül 25%-kal csökkentette a futási időt. Azonban volt még egy rész, amely szintén új megközelítést igényelt: az injektálás. Használva a ’Time’ csomagot kiderült, hogy az SQL lekérdezése sokkal gyorsabb volt az elérhetőségre, mintha Pythonon keresztül történne az ellenőrzés a termék-üzlet párosra. Úgy éreztem, ha túlságosan két nyelvre hagyatkozom, akkor kettő közötti váltás sok időbe telhet. Áttértem tehát egy biztosabb megoldásra, amelyben magát az SQL lekérdezést használtam fel. Létrehoztam egy ideiglenes táblát, ahol beinjektáltam jelenlegi termék üzleteit, és két SQL parancsot futtattam le, egyet arra, hogy átírja az elérhetőségét, ha a táblában benne van, de a termék jelenlegi üzletei között pedig nem, illetve az ellentétét, ahol, ha benne van, de nem elérhető állapotban van, váltsa vissza. Ezzel egyrészt lecsökkent a felesleges ’UPDATE’ műveletek száma, illetve célzott megoldást adott arra, hogy csak azt frissítse, amire tényleg szükség volt. Az így megvalósított megoldás kódját a 4.9. ábra mutatja.

```
self.cur.execute( _sql: """
    UPDATE AvailableStores
    SET available = CASE
        WHEN storeUuid IN (SELECT storeUuid FROM tempAvailableShops) THEN 1
        ELSE 0
    END
    WHERE productId = ?
    AND (
        (storeUuid IN (SELECT storeUuid FROM tempAvailableShops) AND available = 0)
        OR
        (storeUuid NOT IN (SELECT storeUuid FROM tempAvailableShops) AND available = 1)
    )
""",  _parameters: (product_id,))
```

4.9. ábra A feltöltés második verziója

Átlagosan 10-20-szoros volt a futási idő különbsége, ezért kiváltva ezt SQL lekérdezéssel, a futási idő az első próbálkozás 3,2 órájáról, majd fejlesztett változatának 45 percéről lecsökkent 3 percre, amely több mint 64-szeres gyorsulást jelentett.

Mivel fontos volt számomra az, hogy ne csak a termékek, hanem összetevőik, és energiatáblázatuk is elérhető legyen egy másik adatbázishoz is kellett nyúlnom.

Product Information

A ’Product Information’ táblázaton belül tároljuk a termékek egyedi összetevőit, illetve az tartalmazza oszloponként az energiatáblázat részeit. Az Open Food Facts [14] oldal adatbázisát nem csak a tulajdonosok, hanem a felhasználók is töltik, amely a szakdolgozat írásáig nem kevesebb mint 8,5 GB nyers adatot foglal, ami nem tűnhet soknak, ellenben több mint 3 millió sort tartalmaz, és oszlopainak száma eléri a 200-at. Az adatbázis felhasználása során igazodtam a nyílt adatbázis-, illetve az adatbázis tartalmához kapcsolódó licenszéhez [17] emellett az erre felhasznált CC BY-SA 3.0 licenszhez [18] is.

Két opció közül választhattam a felhasználásánál: API lehívással lekérni a számomra fontos adatokat, vagy pedig letölteni az adatbázist (jsonl, rdf, mongodbdump, vagy csv formátumban), majd abból lekérdezni. Nem beszélhetünk kis méretekről, ezért csökkentettem akkora méretre, amekkorára szükségem volt. Tekintve, hogy csak az összetevőkre és az energiatáblázatra volt szükségem, lecsökkentettem a méretét 37 oszlopra és 1.1 GB-ra. Ellenben ebből az adatbázisból csak arra volt szükségem, ami a többi termékben is benne van.

Az elemek feltöltéséhez algoritmust szerettem volna készíteni, de ennek fejlesztése során több problémába ütköztem. Az adatbázis, bár nagy méretű volt, nem volt egyértelmű szerkezete, hiányos elemeket tartalmazott, és nem egynyelvű adattárolási formátummal rendelkezett, aminek az lett a következménye, hogy az algoritmust teljes mértékben csak becslés és találgatás segítségével készíthettem volna el. Probléma volt még az, hogy a csomagolt termékek, amelyek szerepeltek a már meglévő ’Áruk’ adatbázisban, nem tartalmazták a termékek hivatalos EAN kódját, amely ebben az esetben teljesen az áru elnevezésre és a hozzá tartozó képre lehetett egyedül hagyatkozni. A termékek más nyelven megtalálható elnevezései után úgy döntöttem, hogy a termékeket manuális módszerrel töltöm fel, amely bár lassú folyamatú, precízebb adatokkal szolgál.

Price History

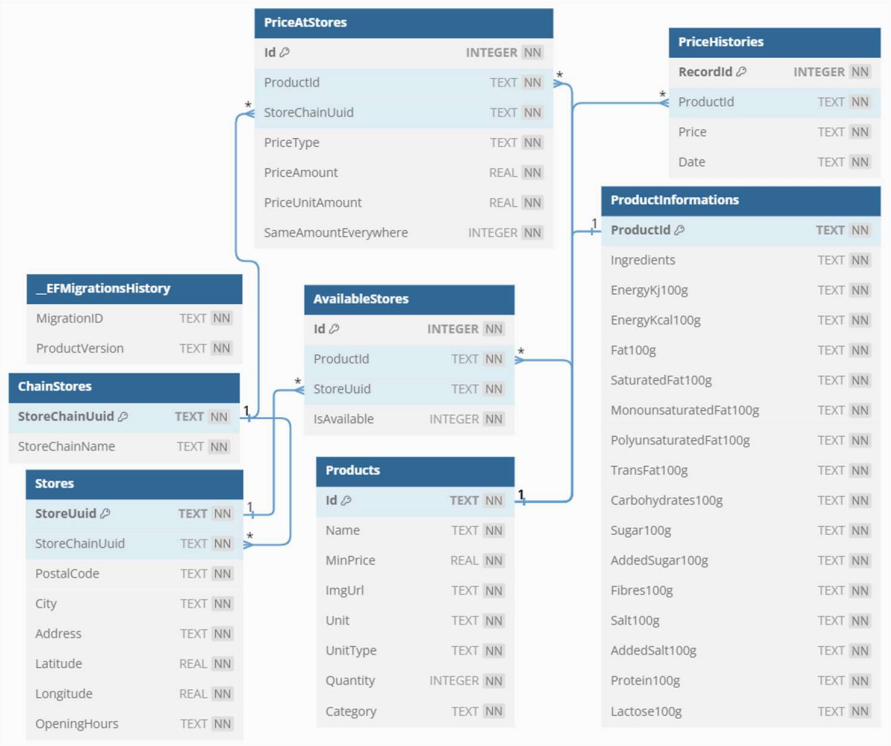
A termékeknél fontosnak éreztem az ár ingadozásának történetét megtartani, ezért egy ’Price History’ táblázatot készítettem ezen információk megtartására, ahol a táblázat frissítése során minden alkalommal az adott termék adott élelmiszerlánc árát beteszi futtatás dátumával, illetve típusával akkor, ha a maximum és minimum vételár és a dátum más.

```
def insert_table_pricehistories(self, unique_prices):
    max_value = max([price[3] for price in unique_prices])
    min_value = min([price[3] for price in unique_prices])
    prod_id = unique_prices[0][0]
    today = date.today()
    self.cur.execute( _sql: """
        SELECT 1 FROM PriceHistories
        WHERE ProductId = ? AND Date = ? AND
        (IsMax = 1 AND Price = ? OR IsMax = 0 AND Price = ?)
        """, _parameters: (prod_id, today, max_value, min_value))
    if not self.cur.fetchone():
        self.cur.execute( _sql: """
        INSERT INTO PriceHistories (ProductId, IsMax, Price, Date)
        VALUES (?, ?, ?, ?)
        """, _parameters: (prod_id, 1, max_value, today))
        self.cur.execute( _sql: """
        INSERT INTO PriceHistories (ProductId, IsMax, Price, Date)
        VALUES (?, ?, ?, ?)
        """, _parameters: (prod_id, 0, min_value, today))
    self.conn.commit()
```

4.10. ábra Price History feltöltése

Ezt a műveletet, nem a főmetódus hajtja végre, hanem a ’Products’ tábla feltöltésekor ezen részek is lefutnak, hasonlóan a ’Price At Stores’ esetéhez.

4.3 Táblák kapcsolata



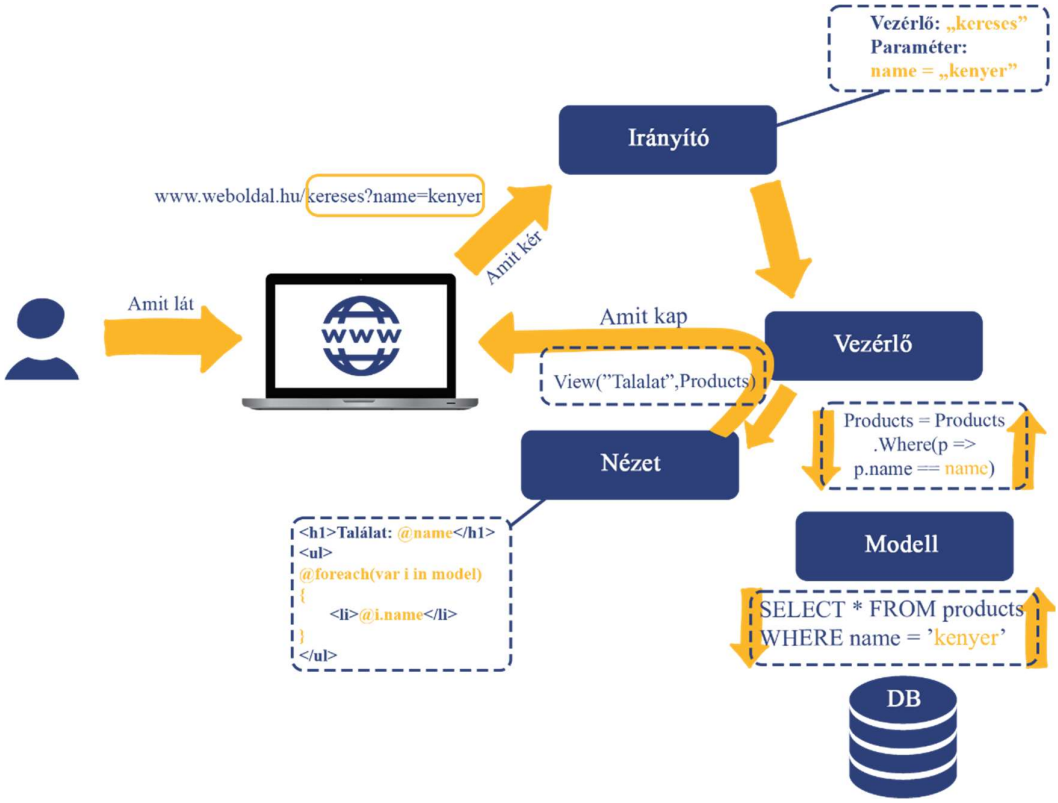
4.11. ábra A táblák kapcsolata

A 4.11. ábra bemutatja, hogy a táblák ’egy a többhöz’ típusú kapcsolatban épültek fel, ha változásokat követünk, és ’egy az egyhez’ típusút, ha információt tárolunk. Ezeket az összeköttetéseket eleinte Pythonon belül valósítottam meg, majd később rábíztam a Visual Studio Entitás Keretrendszerére.

5. WEBALKALMAZÁS

5.1 ASP.NET működése

Web alkalmazás működését, benne az adatok megjelenítését, az ASP.NET MVC rendszerben készítettem el. Az MVC megnevezés egy angol rövidítés; az aktív szerver oldalak, 'Modell-Nézet-Vezérlő' elvű NET keretrendszert támogató megvalósítását fedi le. Ebben a megvalósítási stílusban a felhasználó három komponenssel áll közvetett, vagy közvetlen formában: Nézet, amellyel kapcsolatban van, Vezérlő, amely irányítja a felhasználó által látott adatokat és a Modell, amely lefedi az adatbázist. Az 5.1. ábra bemutatja, hogy miként is épül fel.

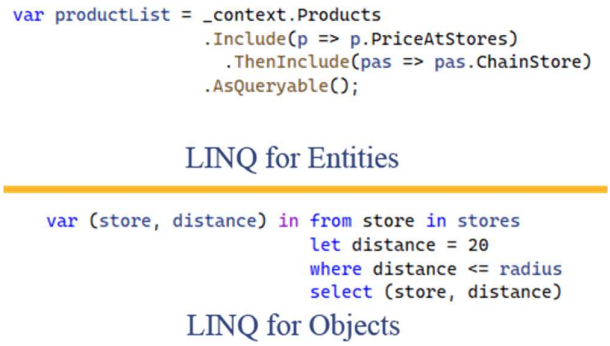


5.1. ábra MVC - a 'Modell-Nézet-Vezérlő' felépítése

A felhasználó az alkalmazás Nézet komponensével van közvetlen kapcsolatban addig, amíg utasításokat nem ad a vezérlőnek a Nézetben található vezérlő elemekből. Ez a megvalósítás ellenben nem egyedüli kivitel. Az MVC kialakítás nem csak az ASP egyedi tulajdonsága, hanem - amellett, hogy egyik legismertebb elvű kivitelezés– egy szoftver tervezési változat. Mint ahogy a 2.3.1 fejezetben kitértem, többféle megvalósítási módszer létezik az MVC mellett: például a Django esetében MVT (Modell-Nézet-Sablon) elvű a megvalósítás Python nyelvezettel, ahol a felhasználó a Django rendszerén keresztül a Nézet komponenssel kapja meg a kérését, majd a Modellből kapott adatokkal a Sablonon keresztül jeleníti meg. Ezen kívül még vannak más megvalósítási formák, de azok nem szerepelnek a megvalósításban.

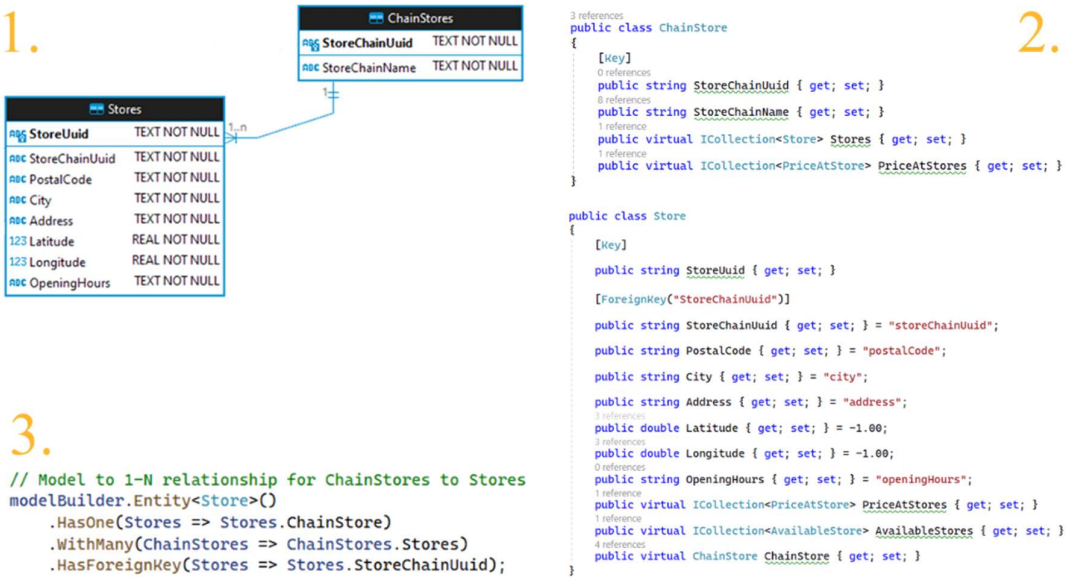
Az ASP Nézetét a saját Kontrollere fogja szolgáltatni, a szükséges háttérben történő kalkulációkkal együtt, hogy a Nézet számára már a megjelenítési módot kelljen egyedül megszabnia. A Kontroller amellett, hogy feldolgozza az adatokat, kapcsolatban van az adatokat szolgáltató modellekkel is. Ezek a Modellek osztályokként működnek, amelyben a változók az adatok helytartói, amelyek meghatározzák az adatbázis oszlopait, szerkezetét, hogy kapcsolatba álljanak az adatbázissal és annak elemeivel.

Az Entitás keretrendszer felel ezek működésének helyességéért, hiszen az említett egy ORM (Object-Relational Mapper, Objektum-Reláció Feltérképező) eszköz, melynek segítségével a táblák kapcsolatai közt lévő adattranzakciók létrejönnek, az adatbázis kontextusában beleépítve. Ennek tudatában úgynevezett integrált leképező nyelv megoldásával kapja meg az adatokat, amelyben az elemekre meghívott metódusokon SQL szerű műveletek történnek. Ez történhet LINQ-t objektumra használva, illetve Entitásra is, melyet az 5.2. ábra is bemutat.



5.2. ábra LINQ használatai

A modellek egymáshoz való viszonyát a Modellben megadott publikus változók adják meg. Ezen megoldások a kapcsolatok mikéntjét a változó egy-, vagy kollektív csatolása teszi élővé (5.3. ábra 2. szegmense). Ezen kapcsolatok (5.3. ábra 1. szegmense) ellenben akkor lesznek ténylegesen felhasználva, ha az adatok lekérése során fel vannak használva, vagyis összeköttetést hoznak létre. Ha kijelöl egy táblát az adatbázisból, csak azokat az elemeket tudja hozzákapcsolni LINQ segítségével, amelyek az adatbázis kontextusban szerepelnek (5.3. ábra 3. szegmense), és a Modellek publikus változókkal rendelkeznek a meghatározásuk helyénél. Ha nem használ összekötést, akkor is megjelenik elemként az objektum, de a tartalma mindig ’null’ érték lesz. LINQ használatánál nem kell foglalkozni az összeköttetés után a kapcsolatok helyes azonosító-azonosító összekötésével, mivel a EF már ezt dinamikusan felismeri és rendelkezésre állítja.



5.3. ábra Egy rész a modell adatbázis tábláinak és kapcsolatainak beépítéséből

ASP esetében ezek az oldalak tartalmaznak olyan elemeket, amelyek egyedileg ASP jelöléseket tartalmaznak a többi HTML jelölések mellett. Természetesen a komponensek XML formátumban jelennek meg a Nézetben, de tartalmazhatnak a szoftveres háttér miatt C# nyelv alapú kódokat is, melyeket szintén használtam. Az ASP jelölések a szerver számára, pontosabban a kontroller számára fontosak, mivel ő kapja meg a felhasználótól az adatokat majd dolgozza fel őket. Hogy miket kap meg, függ a Nézetben megadott utasításoktól és átirányításoktól, de ezen keresztül adhatunk meg ugyancsak változókat is.

```
<li class="page-item">
    <a class="page-link" asp-controller="AllProducts" asp-action="Search"
       asp-all-route-data="@routeValues">&lt;&lt;&lt;</a>
</li>
```

5.4. ábra HTML és ASP jelölések könnyed módon kiegészítik egymást

Az alkalmazásban található nézetekért a saját vezérlőjük felel, amely többek között azt jelenti, hogy egy vagy több vezérlő is szerepelhet az alkalmazásban. Ezen elemek csak a saját nézeteiket képesek autonómiával kezelni, ellenben a vezérlők átirányíthatják más vezérlők nézeteire a felhasználót, így az egyedüli megkötést a nézetekben található modell adja meg. Ez a modell erősen típusos, egyszerre egy modell szerepelhet a nézetben, amelyhez az adatokat a vezérlője adja át. Modell viszont lehet bármelyik modell, amely

lehet adatbázisban használt tábla, vagy táblában található oszlop elemei, de lehet úgynevezett Nézet Modell is, amely szintén egyszeres modellként szerepel.

A nézetmodell nem szerepel az adatbázis kontextusában, mivel a feladata egyedül az többes adatok megjelenítése, amelyben az adatbázisra ráhúzott modelleket használja fel egy közös nézetben (5.5. ábra). Természetesen az egységnyi modellek tartalmazhatnak EF kapcsolatokat, de a külön meghatározott modellek nem. A Nézet Modell használata során tehát meg kell adni, hogy mely adatokat kell tartalmaznia, mivel alapvetően nincs összetétele. Fontos tehát, hogy a Nézet Modell, ugyanúgy viselkedik, mint a Nézet, adatot kell szolgáltatni hozzá, hogy felhasználható legyen.

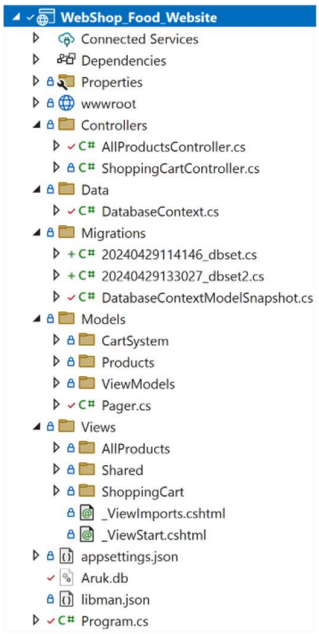
```
namespace WebShop_Food_Website.Models.ViewModels
{
    4 references
    public class ShoppingCartViewModel
    {
        2 references
        public ICollection<Cart> CartItems { get; set; }
        2 references
        public decimal CartTotal { get; set; }
        2 references
        public ICollection<Product> Products { get; set; }
    }
}
```

```
public IActionResult Index()
{
    var cart = ShoppingCart.GetCart(_context, HttpContext);

    var viewModel = new ShoppingCartViewModel
    {
        CartItems = cart.CartItems,
        CartTotal = cart.Total,
        Products = _context.Products.ToList()
    };
    return View(viewModel);
}
```

5.5. ábra Nézetmodell meghatározása, és feltöltése

5.2 Az alkalmazás felosztása

Az alkalmazást használva, ASP.NET MVC projektet elindítva a következő alkalmazás elosztás látható:



5.6. ábra Felépítés elrendezése

A webalkalmazásnak több része is van, amellyel foglalkoztam, ezek a következők:

- **wwwroot**: Webalkalmazás beállításai, mint a CSS, vagy a Javascript, illetve alapértelmezetten tartalmazza a Bootstrap és JQuery könyvtárakat, illetve az összes oldal egyedi kinézetének aspektusát, vagy JS algoritmusát abban az esetben, ha globális szerkesztést vettünk figyelembe. Ezek a ’layout’ fájlban már standard hivatkozásként szerepelnek.
- **Controllers**: A vezérlők helye is itt található meg, amelyek az utasításokat tartalmazzák a nézeteik számára.
- **Data**: Ez a mappa nem létezik alapértelmezetten, ellenben érdemesnek találtam elkülöníteni a többi fájltól, mivel ennek tartalma a DBContext, a modellek közötti kapcsolatfelépítést határozza meg.
- **Migrations**: E mappa és tartalma az adatbázis integrálása során fog létrejönni, amelyről később szó lesz.
- **Models**: Ebben elkülönítettem a modelleket és a nézetmodelleket, melyek használati helyüktől függően vannak szétszedve. Fontos, hogy a főmappában több

mappa létrehozása engedett és fontos, hogy a modellek felhasználása során az elérési helye is ugyancsak tartalmazza az új mappában lévő pozícióját.

- **Views**: A nézetek kontrollerhez tartozó nevük alapján vannak izolálva, amellett, hogy az alapul vett megosztott kinézetek, mint a 'layout', 'Error', a 'Viewimports, és a ViewStart is szerepel cshtml-ként.
- **appsettings**: Itt van beépítve az adatbázis alapértelmezett elérése, amelyről később említést teszek.
- **Aruk**: A webalkalmazás adatbázisa.
- **Libman**: A BS (BootStrap) frissítése során jött létre az elem.
- **Program**: Itt található meg a program futásához szükséges összes beállítás.

5.3 Adatbázis integrálása

Most, hogy meghatároztam, miként működik a rendszerkörnyezet, a már létező adatbázist be kellett integrálni. Az adatbázis megtervezésének, folyamatos javításának majd fejlesztésének a végleges eredménye a kész integrálható adatbázis behelyezése lett. Az integrálás folyamata a következő lépésekből áll:

1. Az adatbázis helyének programba elhelyezése,
2. Modellek adatbázisra való leképezése,
3. Adatbázisban található kapcsolatok felépítése Entitás Keretrendszeren belül,
4. Adatbázis migráció létrehozása, hogy kezelni tudja az adatbázist.

Az integrálás első lépéseként felhasználtam a NuGet csomagok közül az SQLite és SQL Server-t, amellyel a program képes lesz értelmezni és használni az SQLite alapú adatbázist. Adatbázis helyét 'appsettings.json' beállítási konfigurációban alapértelmezett kapcsolatként kell megadni, majd később hivatkozni rá a 'Program.cs' alkalmazás beállításokban. Az ábrán az adatbázist relatív elhelyezéssel a projekten belül hoztam létre, majd későbbiekben is így használtam.

```
builder.Services.AddDbContext<WebShop_Food_Website.Data.DatabaseContext>(options =>
options.UseSqlite(builder.Configuration.GetConnectionString("DefaultConnection")));
```

```
"AllowedHosts": "*",
"ConnectionStrings": {
  "DefaultConnection": "Data Source=Aruk.db"
},
"Logging": {
  "LogLevel": {
    "Default": "Information",
    "Microsoft.AspNetCore": "Warning"
  }
}
```

5.7. ábra Adatbázis webalkalmazásba integrálása

Miután megvan az adatbázis, az EntityFrameWorkCore csomag segítségével létrehoztam a következő módon:

1. Az adatbázis tábláit lemásolva Modelleket húztam rá, hogy helyesen tudja kezelni,
2. Később összekötöttem az adatbázis Kontextében a modelleket a megfelelő módon,
3. Legutolsó sorban pedig megnyitottam a NuGet csomag kezelő konzolját, majd beírtam folytatólagosan a két sort:
    I. ’Add-Migration dbset’: ahol a ’dbset’ egy számunkra meghatározható szöveget jelent. Ennek folyamán az adatbázis a modellek és azok kapcsolataiból létrehoz egy C# nyelvű SQL szerű kódot, ahol leellenőrizhető, miként fog adatbázist létrehozni, vagy kezelni.
    II. ’Update-Database’: Ezzel a migráció kódját lefuttatjuk az adott helyen található adatbázisra. Ezzel probléma volt eleinte, hogy nem tudott lefutni a Python SQLite csomag által létrehozott adatbázissal, így törölve, majd egy EF keretrendszer által létrehozott áttelepülés használható, és Python algoritmus által feltölthető adatbázist hozott létre.

```
migrationBuilder.CreateTable(
    name: "ChainStores",
    columns: table => new
    {
        StoreChainUuid = table.Column<string>(type: "TEXT", nullable: false),
        StoreChainName = table.Column<string>(type: "TEXT", nullable: false)
    },
    constraints: table =>
    {
        table.PrimaryKey("PK_ChainStores", x => x.StoreChainUuid);
    });
```

5.8. ábra Példa migráció eredményének egy szelete

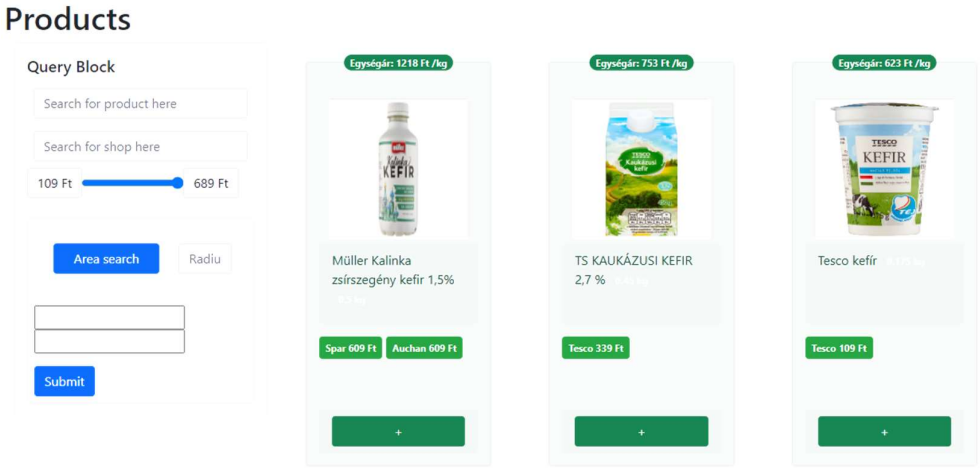
Adatbázis kezelhetősége után elkezdtem létrehozni a Nézeteket, melyekhez később a kontrollereket hozzáigazítottam.

5.4 Oldalak és elemeik

Ahhoz, hogy a nézet megjelenítse az adatokat, a következő 4 akció eredményre volt szükség: Index, Search, és Details és Shopping Cart. Ezen metódusok külön működnek, de működésük összefügg, ezért a továbbiakban a beépített elemekre térek ki: elemek kiírása, lapozásuk, keresés bennük, illetve a kosár funkció megjelenése.

5.4.1 Elemek kiírása

Az Index, ahol az elemek megjelennek kezdetleges kinézetét megalkotva az 5.9. ábra mutatja be miként jelentek meg először az elemek.



5.9. ábra Kezdetleges kinézet egy részlete

Az Index oldal jelenik meg legelőször, amely tartalmazza az összes elemet, amely adatbázis ’Products’ táblájában szerepel. Ahhoz, hogy megjelenítse elsődlegesen az oldalt, meghatároztam a program alap funkcióinál (’Program.cs’), hogy melyik vezérlőt és nézetet jelenítse meg alapértelmezettként:

```
app.MapControllerRoute(
    name: "default",
    pattern: "{controller=AllProducts}/{action=Index}/{id?}");
```

5.10. ábra Program.cs alapoldal beállítása

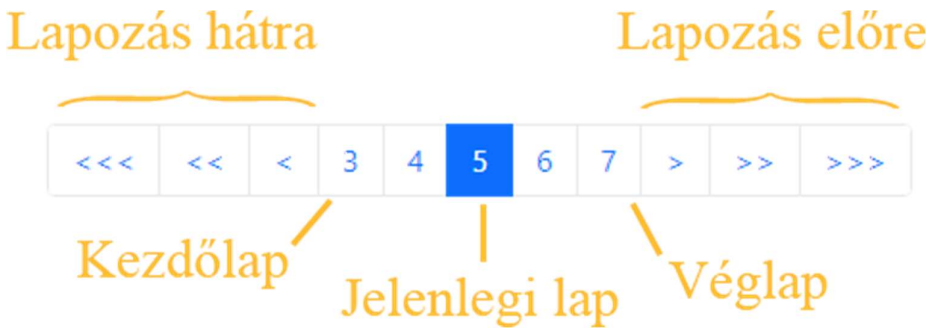
Mivel ez a tábla összesen több mint 2000 elemet tartalmaz, szükségesnek éreztem a lapozási funkciót beépíteni.

5.4.2 Lapozás

5.11. ábra Lapozás minta

A felhasználó számára ez egy könnyebbségi elem, hiszen nem kell végig görgetni az elemeken, hanem lapozás segítségével végigmegy a termékek között.

Lapozás beintegrálására léteznek szintén csomagok, de önálló megoldásra hagyatkoztam.



5.12. ábra Lapozás felépítése

Ahogy az 5.12. ábra mutatja, a lapozásnak van egy terjedelme, amelyben megmutatkozik, mennyi lapszám jelenik meg, mi a kezdő, és véglapja, illetve az irányító nyílgombok, amelyek a jelenlegi laphoz képest jelennek meg. Az egyszeres egy lapot lapoz, a kétszeres

a vég-, vagy kezdőlaphoz ugrik, illetve háromszoros a kezdeti-, vagy legvégső laphoz lapoz.

Ahhoz, hogy ez működjön, szükség van egy külön Modellre, amelyben el kell tárolni a lap méretét, vagyis, hogy hány elem szerepel egy lapon, hány elemünk van, amelyből meg szeretnék jeleníteni elemeket, illetve szintén fontos a jelenlegi lap-, a kezdőlap- és a véglap száma. Ezeket publikus változóként eltároltam, melyet a saját konstruktora használ a következő módon:

```
public Pager(int totalItems, int currentPage, int pageSize = 24)
{
    TotalItems = totalItems;
    PageSize = pageSize;
    TotalPages = (int)Math.Ceiling((decimal)totalItems / (decimal)pageSize);
    CurrentPage = currentPage;
    StartPage = Math.Max(1, CurrentPage - 2);
    EndPage = Math.Min(TotalPages, StartPage + 4);
    if (EndPage - StartPage < 4) StartPage = Math.Max(1, EndPage - 4);
}
```

5.13. ábra Lapozó konstruktora

Az 5.13. ábra konstruktora már egyszerűsítve van, ellenben elágazások segítségével is ugyanaz az eredmény születik. A konstruktor alapértelmezett értékekkel dolgozik akkor, ha nincs megadott érték, de dinamikusan változtatható a kódban, ha arra van szükség. Ezen kódon belül meghatároztam az összes lapszámot, elosztva az összes elemszámot a lapszámmal (jelenleg 24-gyel). Ezt ismerve meg tudtam határozni a véglap értékét, megnézve, hogy melyik kisebb, kiütve annak lehetőségét, hogy túlmenjen az összes lapszámon, illetve a kezdőlapnál pedig azt, hogy a jelenlegi lapnál kettővel kevesebb vagy az 1 a nagyobb érték. A nézetben található lapozás pedig ehhez mérten fogja a nyilakat és oldalakat megjeleníteni.

Ez viszont csak akkor generálódik le helyesen, ha a nézet tartalmaz lapozó objektumot, amely azonos lesz nézetben ViewBag segítségével (amely erősen típusos) megkapott lapozó objektummal. A nézet lapszáma minden hívásnál a vezérlőhöz kerül, ahol a következő módon fogja visszaadni az elemeket:

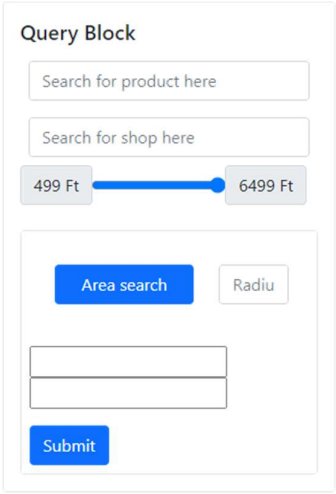
```
public async Task<IActionResult> Index(int page = 1)
{
    page = page < 1 ? 1 : page;
    var pageSize = 24;
    var skip = (page - 1) * pageSize;
    var productsCount = _context.Products.Count();
    ViewBag.Pager = new Pager(productsCount, page, pageSize);

    var productList = await _context.Products
        .Include(p => p.PriceAtStores)
            .ThenInclude(pas => pas.ChainStore)
        .AsNoTracking()
        .Skip(skip)
        .Take(pageSize)
        .ToListAsync();

    return View(productList);
}
```

5.14. ábra Index vezérlő működése

A lap értéke itt is aszinkron módon lett ellenőrizve: megadtam a lap elemeinek maximális számát, kiszámoltam, hogy mekkora elemszámot kell átugrani, illetve a jelenlegi elemmennyiség méretét (24) meghatároztam és visszaadtam a Nézet számára. Ahogy azt az 5.14. ábra is mutatja, ezen műveletek mind LINQ segítségével történtek, amely során egy objektumon többszörös műveletet végeztem csupán egy sorban.

5.4.3 Keresés

A keresés opció - amellett, hogy a keresés nézetben is szerepel - az Index oldalon elsődlegesen található meg.



5.15. ábra Keresés területe

Kezdetleges kinézetén megjelenik a terméknév-, az élelmiszerlánc-, a maximális ár-, illetve a lokáció alapú keresés. Mindegyik kérvény egy Form elemen keresztül jelenik meg, melyben az elemek értékei lesznek beküldve, már nem az index számára, hanem a keresés metódusának.

```
public async Task<IActionResult> Search(int page = 1, string name = "", string shop = "", double price = 0,
                                        double latitude = 0, double longitude = 0, double radius = 0)
{
    var productList = _context.Products
                      .Include(p => p.PriceAtStores)
                        .ThenInclude(pas => pas.ChainStore)
                      .AsQueryable();

    if (!name.IsNullOrEmpty()) productList = await
            Task.FromResult(productList.Where(p => p.Name.ToLower().Contains(name.ToLower())));

    if (!shop.IsNullOrEmpty()) productList = await
            Task.FromResult(productList.Where(p => p.PriceAtStores
                            .Any(pas => pas.ChainStore.StoreChainName.ToLower().Contains(shop.ToLower()))));

    if (!price.Equals(0) && price < 10000) productList = await
            Task.FromResult(productList.Where(p => p.PriceAtStores.Any(pas => pas.PriceAmount <= price)));

    if (!latitude.Equals(0) && !longitude.Equals(0) && !radius.Equals(0))
        productList = await FilterPosition(productList, latitude, longitude, radius);

    var pageSize = 24;
    if (page < 1) page = 1;
    var pager = new Pager(productList.Count(), page, pageSize);
    var skip = (page - 1) * pageSize;
    ViewBags(productList, pager, name, shop, price, latitude, longitude, radius);

    var pageProductList = await productList.AsNoTracking().Skip(skip).Take(pageSize).ToListAsync();
    return View(pageProductList);
}
```

5.16. ábra Keresés akciómetódusa

Az 5.16. ábra az 5.15. ábra beviteli mezőiből kapott összes értéket az akcióeredmény bemeneteként megkapja, majd ahhoz képest, hogy melyik adat nem üres, hoz létre szűréseket.

Név

Egyszerű név keresés, melyben a bemeneti ’name’ szöveget lekicsinyíti és összehasonlítja a ’Products’ táblában szereplő elemek kicsinyített szövegeivel, LINQ módszerrel.

Üzlet

Egyszerű üzlet keresés, melyben a bemeneti ’shop’ szöveget lekicsinyíti és összehasonlítja a ’Products’ táblájához kötött ’PriceAtStores’ táblában található ’ChainStores’ táblából megszerzett élelmiszerlánc nevek kicsinyített szövegeivel, LINQ módszerrel.

Vételár

Egyszerű vételár keresés, melyben a bemeneti ’price’ lebegőpontos számot hasonlítja össze a ’Products’ táblájához kötött ’PriceByStores’ táblában található elemekkel, amelyek kisebbek vagy egyenlőek a megadott vételárnál, LINQ módszerrel.

Helyzet

```
public async Task<IQueryable<Product>> FilterPosition(IQueryable<Product> query, double centerLatitude, double centerLongitude, double radius)
{
    var maxLat = centerLatitude + radius / 111;
    var minLat = centerLatitude - radius / 111;
    var maxLon = centerLongitude + radius / (111 * Math.Cos(ToRadians(centerLatitude)));
    var minLon = centerLongitude - radius / (111 * Math.Cos(ToRadians(centerLatitude)));
    var stores = await _context.Stores
        .Include(s => s.ChainStore)
        .Where(s => s.Latitude <= maxLat && s.Latitude >= minLat
                && s.Longitude <= maxLon && s.Longitude >= minLon)
        .ToListAsync();

    var availableStoresProducts = await _context.Products.Include(p => p.AvailableStores)
        .Select(p => p.AvailableStores).ToListAsync();

    var filteredStores = new List<string>();
    var distances = new List<(string, double)>();
    foreach (var (store, distance) in from store in stores
                                      let distance = CalculateDistance(centerLatitude, centerLongitude, store.Latitude, store.Longitude)
                                      where distance <= radius
                                      select (store, distance))
    {
        filteredStores.Add(store.StoreUuid);
        distances.Add((store.StoreUuid, distance));
    }
    ViewBag.Distance = distances;
    query = query.Where(p => p.AvailableStores.Any(asp => filteredStores.Contains(asp.StoreUuid)));
    return query;
}
```

5.17. ábra Helyzeti szűrés

A helyzeti szűrés már kicsit bonyolultabb. Az ’Stores’ táblában található üzletek tartalmazzák a pontos helyzetüket a Földön hosszúsággal és szélességgel. Minden szélességi kör körülbelül 111 km távolságnak felel meg. A szűrés három bemeneti értéket kap meg, melyből kettő a felhasználó jelenlegi pozíciója szélességben és hosszúságban, illetve az érték kilométerben, hogy milyen távol szeretne keresni a helyzetétől.

Ehhez először létrehoztam egy nagyobb szűrést nagyobb rátájú pontatlansággal a következőképpen:

1. Meghatároztam, kiszámolva a kapott pozíció maximális hosszúsági és szélességi értékét, melyben egy ideiglenes kört írok le a térképen.
2. A ’Stores’ táblában lévő pozíció elemein ellenőriztem, hogy mely elemek esnek bele ebbe a kritériumba.
3. Létrehoztam egy listát, melyben meghívtam az árukhoz tartozó összes üzletet.
4. Futtattam egy elemi ciklust, melyben egyesével szűrtem a listában lévő üzleteket pontosabb számolással, hogy ténylegesen benne vannak-e a kért kört leíró területen belül, pozitív eredmény esetében pedig hozzáadta a szűrt üzletek listájához.

```
public static double CalculateDistance(double lat1, double lon1, double lat2, double lon2)
{
    var R = 6371; // Radius of the Earth in kilometers
    var dLat = ToRadians(lat2 - lat1);
    var dLon = ToRadians(lon2 - lon1);
    var a = (Math.Sin(dLat / 2) * Math.Sin(dLat / 2)) +
            Math.Cos(ToRadians(lat1)) * Math.Cos(ToRadians(lat2)) *
            Math.Sin(dLon / 2) * Math.Sin(dLon / 2);
    var c = 2 * Math.Atan2(Math.Sqrt(a), Math.Sqrt(1 - a));
    var distance = R * c; // Distance in kilometers
    return distance;
}

public static double ToRadians(double angle)
{
    return angle * (Math.PI / 180);
}
```

5.18. ábra Üzlet és a felhasználó pozíciója távolságának kiszámítása Haversine képlet segítségével

Az 5.18. ábra egy matematikai számolást mutat be, amelyet az úgynevezett Haversine összefüggéssel határoztam meg, amely során két pont távolságát meg lehet határozni a gömb (jelen esetben a Föld) felszínén. Az alábbi módon történik a számolás (0,5% hibarátával a Föld formájából adódóan), amelyet Stack OverFlow [19] segítségével oldottam meg:

$$a = sin^2\left(\frac{\Delta\varphi}{2}\right) + \cos(\varphi 1) \cdot \cos(\varphi 2) \cdot sin^2\left(\frac{\Delta\lambda}{2}\right)$$

$$c = 2 \cdot atan2(\sqrt{a}, \sqrt{(1-a)})$$

$$d = R \cdot c$$

Melyben $\varphi 1$ és $\varphi 2$ két pont szélességi fokának radiánban mért értékei, melynek különbsége $\Delta\varphi$. Két pont hosszúsági különbségét radiánban a $\Delta\lambda$ adja meg, melynek eredménye ('a') Földön létrejött háromszög oldalainak négyzetének összege, a 'c' pedig a Földi háromszög középponti szögének radiánban mért értéke. Ebből kiszámolva kapjuk meg a Föld sugarával megszorozva (körülbelül 6371 km) a távolságot a megadott két pontunk között.

5. A szűrt üzletek listát felhasználva szűri le a 'Products' táblából az elérhető üzletek listáját, és adja vissza az eredményt.

Célja az volt, hogy a felhasználó ne essen abba a problémába, hogy olyan terméket adjon a kosarába, amely nem elérhető az üzletei számára, ezáltal csökkentve az időt, és egyszerűsítve a keresést.

5.4.4 Kosár

A kosár létrehozásához a Professional ASP. NET MVC 4 tanulmányt [20] használtam fel, amely bár más verziójú C# NET keretrendszeren íródott, lehetőségem volt alkalmazni belőle e területet.

A kosár esetében új elemeket kellett behelyeznem az alkalmazásban: Sütik, új modell, új tábla, Ajax JavaScript és metódusok.

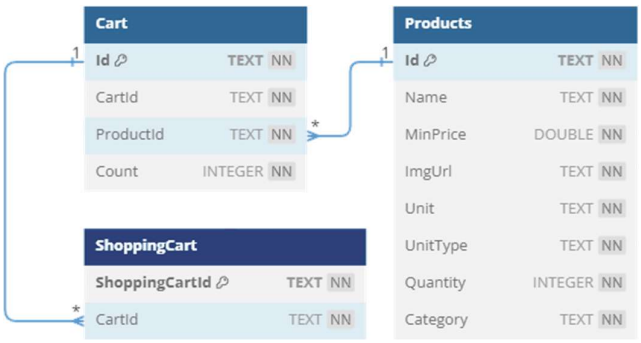
Websütik használata rendkívül elterjedt az oldalak használata során, mivel egyszerűvé teszi a lapok számára, hogy a felhasználó gépén adatokat tároljanak ideiglenesen, amelyeket fel tudnak használni későbbiekben több böngészőlap között. Ennek funkcióját a kosár létrehozására használtam fel. Használat előtt érvénybe kell léptetni az elérhetőségének lehetőségét azáltal, hogy a programba elhelyeztem a websütik engedélyezését 1000 másodpercre, ahol az adatbázis integrációja is megtörtént (Program.cs).

```
builder.Services.AddSession(options =>
{
    options.IdleTimeout = TimeSpan.FromSeconds(1000);
    options.Cookie.HttpOnly = true;
    options.Cookie.IsEssential = true;
});
```

5.19. ábra Websütik engedélyezése

Kosár (’Cart’ tábla) létrehozásához létrehoztam egy új modellt, amelyben eltároltam a Sor Azonosítóját, A kosár azonosítóját, a termék azonosítóját (a ’Products’ táblának a termék azonosítójához kötve), illetve a darabszámát. Az új táblát elhelyeztem az adatbázis kontextusában is egy-több kötésként, mely során a felhasznált adatbázis az új modell táblájával bővült, migráció segítségével.

E Modell használatát az 5.20. ábra bemutatja.



5.20. ábra Új modell beépítése és felépítése

A ’Cart’ tábla tartalmazza mindazon elemeket, amelyeket felsoroltam. Új táblaként, amely az adatbázistól függetlenül, de szerepel, az a Bevásárlókocsi táblája (’ShoppingCart’), melynek feladata a felhasználó ideiglenes munkamenet azonosítóját hozzácsatolása a kosárhoz. Így csak a kosár tartalma lesz olvasható, a felhasználó munkamenete pedig csak a rendszer számára látható. Minden új felhasználó érkezése csak akkor lesz felhasználva, ha felhasználja a kosár funkcióját. Abban az esetben van felhasználva, ha a felhasználó berakja a terméket a kosarába: a program megkapja a kérést (Ajax küldéssel), ellenőrzi, hogy a felhasználó már rendelkezik saját azonosítóval, ha nem, ad hozzá egy Globális Egyedi Azonosítót (Guid-t) amely később hozzácsatolja a jelenlegi munkamenethez.

```
private static string GetCartId(HttpContext context)
{
    if (context.Session is null)
        throw new InvalidOperationException(
            "Session has not been configured for this application or request.");
    if (context.Session.GetString(CartSessionKey) is null)
        if (!string.IsNullOrWhiteSpace(context.User.Identity.Name))
            context.Session.SetString(CartSessionKey, context.User.Identity.Name);
        else
        {
            Guid tempCartId = Guid.NewGuid();
            context.Session.SetString(CartSessionKey, tempCartId.ToString());
        }

    return context.Session.GetString(CartSessionKey);
}
```

5.21. ábra Session (Munkamenet) beépítése

Miután kész volt az azonosítás, hozzáadtam a kosarakat kezelő függvényeket, mint a hozzáadást vagy a törlést, illetve bizonyos tulajdonságokat; a darabszám és lista lekérése.

```
public void AddToCart(Product product)
{
    var cartItem = context.Carts.SingleOrDefault(
                        c => c.CartId == ShoppingCartId &&
                        c.ProductId == product.Id);

    if (cartItem is null) context.Carts.Add(new Cart
        {
            CartId = ShoppingCartId,
            ProductId = product.Id,
            Product = product,
            Count = 1,
        });
    else cartItem.Count++;
    context.SaveChanges();
}
```

5.22. ábra Hozzáadás

Ebben a megadott részletben kikereste a terméket, majd hiány esetén hozzáadta a kosárhoz, vagy növelte a darabszám tulajdonságát. A törlés hasonló elven alapult, melyben a megkapott azonosító (fontos, hogy ebben az esetben már a sor azonosítóját keressük meg), melynek felfedezése után az algoritmus teszteli állapoti jogosultságát és töröl, darabszámot csökkent, a szituációnak megfelelően.

```
public int RemoveFromCart(int id)
{
    var cartItem = context.Carts.FirstOrDefault(
        cart => cart.CartId == ShoppingCartId &&
        cart.RecordId == id);
    if (cartItem != null)
    {
        if (cartItem.Count > 1) cartItem.Count--;
        else { context.Carts.Remove(cartItem);
               cartItem.Count = 0;}
        context.SaveChanges();
        return cartItem.Count;
    }
    return 0;
}
```

5.23. ábra Törlés

Ezen elemek mind a Bevásárlókocsi függvényei voltak, a kontrollerhez más elemek tartoztak.

Vezérlő és Ajax

Az Index teljes működését az 5.24. ábra mutatja.

```
public async Task<IActionResult> Index()
{
    var cart = ShoppingCart.GetCart(_context, HttpContext);
    var cartProductIds = cart.CartItems.Select(item => item.ProductId).ToList();
    var itemsChainStores = await _context.ChainStores
        .Where(ccs => ccs.PriceAtStores
        .Any(pas => cartProductIds.Contains(pas.ProductId)))
        .Select(ccs => new
        {
            ccs.StoreChainName,
            PriceAtStores = ccs.PriceAtStores
            .Where(pas => cartProductIds.Contains(pas.ProductId)).ToList()
        })
        .ToDictionaryAsync(
            kvp => kvp.StoreChainName,
            kvp => kvp.PriceAtStores.AsEnumerable());

    var chainStoresPricesTotal = itemsChainStores.ToDictionary(
        kvp => kvp.Key,
        kvp => kvp.Value.Sum(pas =>
        pas.PriceAmount * cart.CartItems
        .Find(item => item.ProductId == pas.ProductId).Count));

    var viewModel = new ShoppingCartViewModel
    {
        CartItems = cart.CartItems,
        ItemsChainStores = itemsChainStores,
        ChainStorePricesTotal = chainStoresPricesTotal,
        Products = await _context.Products
        .Where(p => cartProductIds.Contains(p.Id)).ToListAsync(),
        CartCount = cart.Count
    };
    return View(viewModel);
}
```

5.24. ábra Index teljes működése

A termékek kiválasztását és megjelenítését másképp képzeltem el, mint szokásos, szerettem volna élelmiszerláncra leosztva megjeleníteni őket. Ahhoz, hogy így megjelenítsem a kosárban található teljes termékpalettát, többféle szűrést és ellenőrzést kellett végrehajtanom:

1. Először megszerzem a jelenlegi kosarat,
2. Kiszedem a teljes termékazonosítót, ami a kosárban található,
3. Szótárban teszem az üzleteket, illetve a hozzájuk kulcsként tartozó termékeket, így lehívásnál automatikusan megkapom, hogy melyik üzlet, mely termékeket tartalmazza,
4. Egy másik szótárban pedig a teljes vételár összeget tárolom el üzletenként, amely az adott termék és darabszámának szummáját jelenti.
5. Végül visszaküldöm nézetmodellben az elemeket, mint a kosár elemeit, az élelmiszerlánconként elemeket, illetve ebből származó teljes összeget, a termékeket, és az összes elemszámot.

Hozzáadás

```
public IActionResult AddToCart(string id)
{
    var addedProduct = _context.Products
        .Single(product => product.Id == id);

    var cart = ShoppingCart.GetCart(_context, HttpContext);

    cart.AddToCart(addedProduct);
    var results = new ShoppingCartAddViewModel
    {
        cardItemsCount = cart.Count
    };
    return Json(results);
}
```

5.25. ábra Kosárba tett tétel alkalmazása

Kosárba tétel meghívása Ajax segítségével történik (5.25. ábra), így weboldal frissítés nélkül tevődik be a kosárba a tétel, növelve a felhasználói élményt. Az üzenetben csak az elemszám jelenik meg.

Törlés

```
public async Task<IActionResult> RemoveFromCartAsync(int id)
{
    var cart = ShoppingCart.GetCart(_context, HttpContext);
    var message = "";
    var itemCount = 0;
    try
    {
        var cartItem = await _context.Carts
         .Include(item => item.Product)
         .SingleOrDefaultAsync(item => item.RecordId == id);
        if (cartItem is not null)
        {
            message = $"{cartItem.Product.Name} has been removed from your shopping cart.";
            itemCount = cart.RemoveFromCart(id);
            await _context.SaveChangesAsync(); // Save changes asynchronously.
        }
    }
    catch (Exception) { message = "Error: Unable to remove the item from your shopping cart."; }
```

5.26. ábra Törlés első részlete

A törlés kettő részből áll, egyrész töröl a kosár tartalmából (5.26. ábra), másrészt pedig a tartalom frissítéséből (5.27. ábra).

```
var cartProductIds = cart.CartItems.Select(item => item.ProductId).ToList();
var itemsChainStores = await _context.ChainStores
    .Where(ccs => ccs.PriceAtStores.Any(pas => cartProductIds.Contains(pas.ProductId)))
    .ToListAsync();

var chainStoresTotal = itemsChainStores.ToDictionary(
    ccs => ccs.StoreChainName,
    ccs => ccs.PriceAtStores.Where(pas => cartProductIds.Contains(pas.ProductId))
    .Sum(pas => pas.PriceAmount * cart.CartItems
    .Find(item => item.ProductId == pas.ProductId)?.Count ?? 0));

var results = new ShoppingCartRemoveViewModel
{
    Message = message,
    ChainStorePricesTotal = chainStoresTotal,
    CartCount = cart.Count,
    ItemCount = itemCount,
    DeleteId = id
};

return Json(results);
```

5.27. ábra A törlés második részlete, a visszajelzés

Az ábrán jól látszik, hogy a totális összeg számolásához elengedhetetlen, hogy ismételt 3 lépéses LINQ lekérdezést kell intézni, hogy lekérhessem a pontos összeget.

Ennek adatait JS használatán keresztül felhasználom a következő módon:

1. Elküldöm a kérést a szervernek, majd várom a visszajelzését

```
<script type="text/javascript">
    $(document).ready(function () {
        // Cache frequently used selectors
        const $cartTotal = $('#cart-total');
        const $updateMessage = $('#update-message');
        const $cartStatus = $('#cart-status');

        $(document).on('click', '.RemoveLink', function () {
            const recordToDelete = $(this).data('id');
            removeFromCart(recordToDelete);
        });

        function removeFromCart(recordId) {
            if (!recordId) return;

            $.post("/ShoppingCart/RemoveFromCart", { "id": recordId })
                .done(updateCartUI)
                .fail(handleRemoveError);
        }

        function handleRemoveError(error) {
            console.error("Error occurred during RemoveFromCart request:", error);
        }
```

5.28. ábra Szerver POST Javascripten keresztül

2. A választ megkapva frissítem funkciókra lebontva a nézetben megjelent elemeket, amelyek a kellő azonosítóval rendelkeznek:

```
function updateCartUI(data) {
    console.log("Data received from server:", data);
    updateItemCount(data);
    $cartTotal.text(data.cartTotal);
    updateMessage(data.message);
    $cartStatus.text(`Cart (${data.cartCount})`);
    updateChainStoreTotals(data.ChainStorePricesTotals);
}

function updateItemCount(data) {
    const $rowToDelete = $('#row-' + data.deleteId);
    if (data.itemCount == 0 && $rowToDelete.length) {
        $rowToDelete.fadeOut();
    } else {
        $('#item-count-' + data.deleteId).text(data.itemCount);
    }
}

function updateMessage(message) {
    if ($updateMessage.length) {
        $updateMessage.text(message);
        $updateMessage.stop()
        $updateMessage.delay(200).fadeIn().delay(1000).fadeOut();
    } else {
        console.error("Element with id 'update-message' not found.");
    }
}

function updateChainStoreTotals(chainStoreTotals) {
    for (const [storeName, total] of Object.entries(chainStoreTotals)) {
        const storeTotalId = '#total-' + storeName.toLowerCase();
        const $storeTotal = $(storeTotalId);

        if ($storeTotal.length) {
            $storeTotal.text(total.toFixed(2));
        } else {
            console.error("Element with id '" + storeTotalId + "' not found.");
        }
    }
}
```

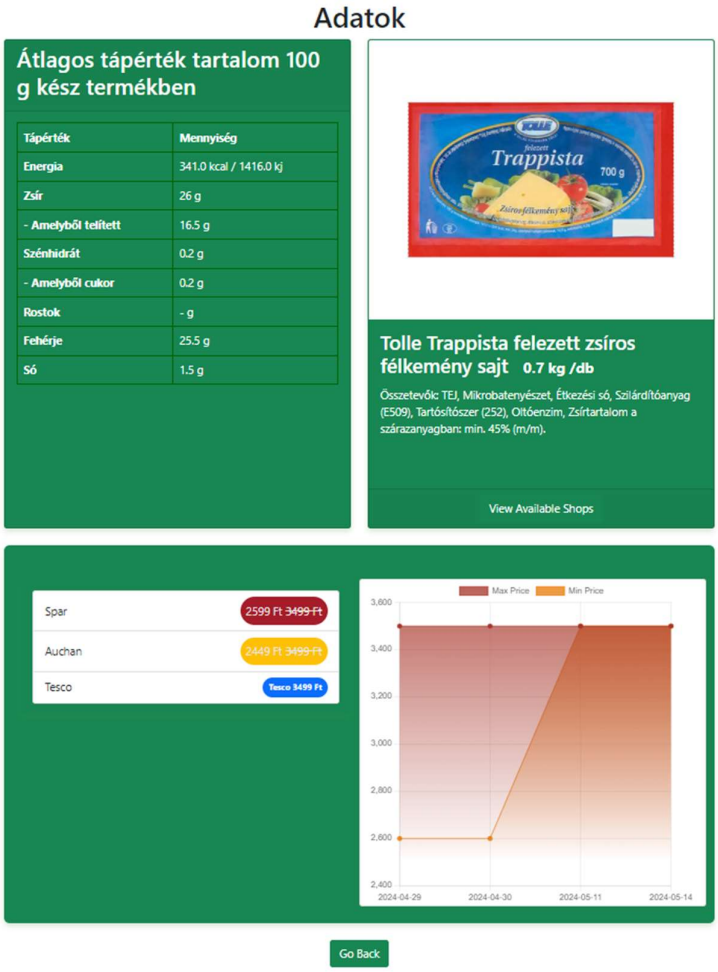
5.29. ábra A kapott adatok nézetben való integrálási frissítése

Végül a következő kinézetet használtam: úgy hoztam létre, hogy azok az üzletek jelenjenek meg csak a termékekhez, amelyekben ténylegesen szerepelnek a termékek, így tényleg csak a lényeges információ marad meg.

5.30. ábra Ideiglenes kosár kinézet

5.4.5 Részletek

Termékek megtekintése az felsorolt elemre való kattintás után irányítja át a ’Részletek’ nézetre. Ennek során a felhasználó találkozik olyan információkkal, mint az energiatáblázat, összetevők, az ár változása, illetve a termékkel kapcsolatos alapvető információk.



5.31. ábra Nézet a termék részleteiről

Az adatok megjelenítését Nézetmodellel valósítottam meg, amely a következő elemeket tartalmazta: ’Termékek’ tábla, csatolva az ’Elérhető Üzletek’, ’Üzletek’, ’Élelmiszerláncok’, és ’Termék Információk’ tábla, illetve szintén a termék maximum és minimum árai és ezek dátumai.

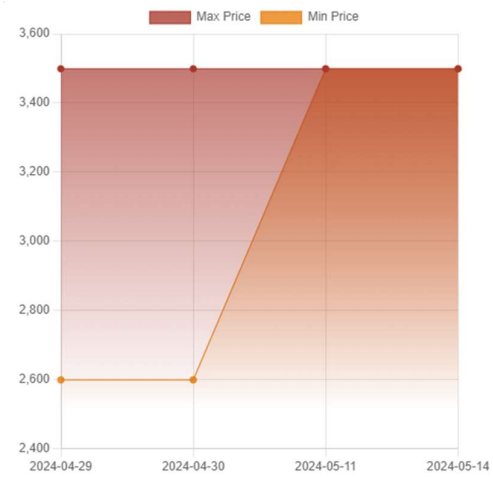
```
public async Task<IActionResult> Details(string id)
{
    if (id is null) return NotFound();

    var products = await _context.Products.AsNoTracking()
        .Include(p => p.AvailableStores)
            .ThenInclude(asp => asp.Store)
                .ThenInclude(s => s.ChainStore)
        .FirstOrDefaultAsync(p => p.Id == id);

    if (products is null) return NotFound();
    else products = await CheckInfosAsync(products, id);


    var price = await _context.PriceHistories.Where(ph => ph.ProductId == id).ToListAsync();
    var DetailsViewModel = new DetailsViewModel
    {
        Product = products,
        MaxPrices = string.Join(",", price.Where(ph => ph.IsMax == 1).Select(item => item.Price)),
        MinPrices = string.Join(",", price.Where(ph => ph.IsMax == 0).Select(item => item.Price)),
        Dates = string.Join(",", price.Where(ph => ph.IsMax == 0)
        .Select(item => "'" + item.Date.ToString("yyyy-MM-dd") + "'")),
    };
    return View(DetailsViewModel);
}
```

5.32. ábra Részletek kiírása

A feltöltése a következőképp történt: A weblapról indított kérés a Vezérlő 'Részletek' akcióját meghívva megkereste a megadott azonosítójú elemet a táblából, hozzárendelte az előbb felsorolt táblákat, hozzácsatolta a szükséges adatokat, illetve ahogy az 5.31. ábra is mutatja, ha nem talált, pótolja hiányát hibára futás nélkül. A megkapott terméket egy Nézetmodellben felhasználva, belehelyeztem a kellő helyére, a többi elemet pedig szövegként átalakítva küldtem el a nézetnek, a kellő adatok LINQ-s kiválasztásával együtt. A termékkel érkező adatokat használtam fel JavaScripten belül, megjelenítve az árak változását.

Ár történet

Az árváltozás nyomon követhetőségének érdekében egy JS könyvtárat a Chart.JS [21] nyílt forrású professzionális vizualizációs csomagot használtam fel, ahol különféle diagram típusokat jeleníthetek meg, amelyet felhasználtam az esetemben egy egyszerű, két vonaldiagramot tartalmazó ábra megjelenítésére (5.33 ábra) az árak minimum és maximum értékeinek nyomon követéséhez.



5.33. ábra Az árak minimum és maximum értékeinek megjelenítése

Az 5.34. ábra mutatja a diagram feltöltésének kódját, amelynek elemei többek között a modelltől kapott elemekből épülnek fel (5.35. ábra), ahol átalakulnak szöveges nyers formátummá, amelyet a JavaScript is feldolgoz, illetve a diagramok elkészítéséhez használt átmeneti kitöltés, amelynek létrehozását Stack Overflowt [22] felhasználva hoztam létre.

```
var myChart = new Chart(ctx, {
    type: 'line',
    data: {
        labels: labels,
        datasets: [
            {
                label: 'Max Price',
                data: maxPrices,
                fill: true,
                backgroundColor: gradientMax,
                borderColor: 'rgba(255, 99, 132, 1)',
                borderWidth: 1,
                pointBackgroundColor: 'rgba(255, 99, 132, 0.8)',
                tension: 0.1
            },
            {
                label: 'Min Price',
                data: minPrices,
                fill: true,
                backgroundColor: gradientMin,
                borderColor: 'rgba(54, 162, 235, 1)',
                borderWidth: 1,
                pointBackgroundColor: 'rgba(54, 162, 235, 0.8)',
                tension: 0.1
            }
        ]
    },
    options: {
        scales: {
            y: {
                beginAtZero: false,
                suggestedMax: @Html.Raw(@Model.MaxPrices.Split(',').Max().ToString()) + 100,
                suggestedMin: @Html.Raw(@Model.MinPrices.Split(',').Min().ToString()) - 100
            }
        }
    }
});
```

5.34. ábra A diagram feltöltéséhez használt kód

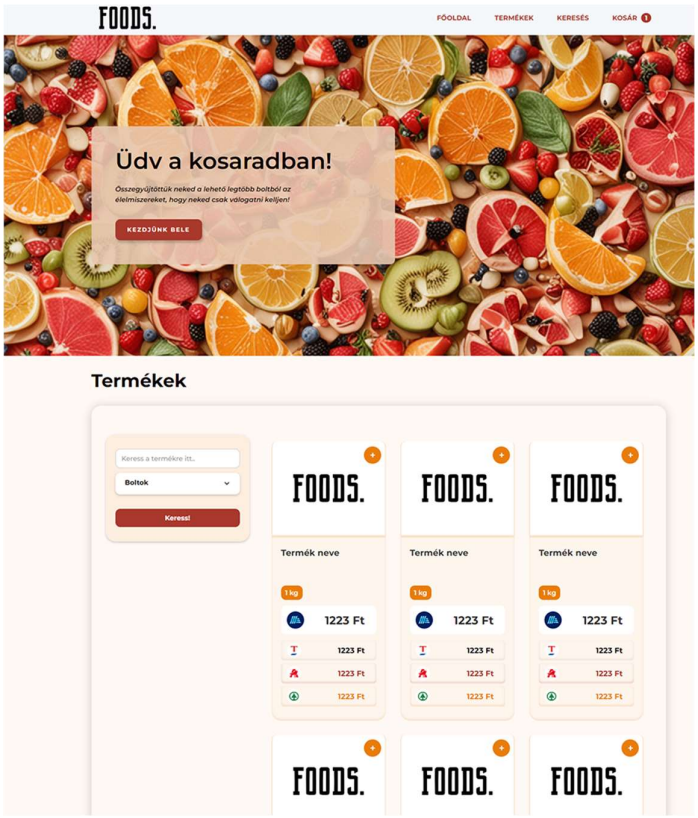
```
var ctx = document.getElementById('priceChart').getContext('2d');
var maxPrices = [@Model.MaxPrices.ToString()]
var minPrices = [@Model.MinPrices.ToString()]
var labels = [@Html.Raw(@Model.Dates)];
var gradientMax = ctx.createLinearGradient(0, 0, 0, 400);
gradientMax.addColorStop(0, 'rgba(169, 56, 45, 0.8)');
gradientMax.addColorStop(1, 'rgba(169, 56, 45, 0)');
var gradientMin = ctx.createLinearGradient(0, 0, 0, 400);
gradientMin.addColorStop(0, 'rgba(233, 125, 11, 0.8)');
gradientMin.addColorStop(1, 'rgba(233, 125, 11, 0)');
```

5.35. ábra A diagram adatainak meghatározása

6. FEJLESZTÉSI LEHETŐSÉGEK

A weboldalt tekintve, az oldal használható, és a felhasználó igényeit teljes mértékben ki tudja elégíteni. Fejlesztési lehetőségekben viszont vannak olyan részek, amelyek megkívánják a további elemek megjelenését, úgy, mint a kinézet, több elem szerepeltetése, az adatbázis elemeinek további szűrése, vagy a kosárban lévő elemek kiegészítése.

Ezen elemek közül már foglalkoztam egy-egy oldal új kinézetével, melyeket Webflow [23] a ’Fogd és vidd’ elvű weboldalkészítő oldalán készítettem el. Ezekben igyekeztem barátságos, és hívogató elemekkel szolgálni, mint az elemszintű animációk, reszponzív kialakítás és meleg színpalettával melyet 70-20-10 elven használtam fel.



6.1. ábra Főoldal tervezett kinézete Webflowt felhasználva

Természetesen ennek feladata a használati élmény növelése, funkcionalitásban viszont nem okoz újabb változást. Abban az esetben lenne még ideálisabb a használata, ha a termékek kiválasztásánál a Kosár oldalon nem csak az új kinézet, hanem az ottani ajánló rendszer is megvalósulna. Ez úgy egészítené ki a működését, hogy azokból az üzletekből, amelyekben nem található meg az adott áru (például kenyér), akkor alternatívaként az adott kategóriából ajánlana olyan termékeket, amelyek más élelmiszerláncokban - melyekben a kiválasztott termék nem volt elérhető, de más áruk viszont igen – alternatívaként berakható a kosárba, a kosár elhagyása nélkül.

Az elemek szűrése pedig leginkább az alapvető, nem előre csomagolt termékekre vonatkozik, mint a zöldségek vagy gyümölcsök. Az adatbázisban bár egy kategóriában szerepelnek, csak egy-egy üzletben jelzi a létét, ellenben mindahány áru szintén megtalálható a többi élelmiszerláncban is. Úgy gondolom tehát, hogy ideális lenne ezeket egyként használni, nem pedig egyesével behelyezni őket.

Utolsó sorban pedig kiegészíteném az adatbázisban a megtalálható árukat azokkal, amelyek külön-külön megtalálhatóak a saját élelmiszerlánc oldalukon, elérhetőség nélkül, ezáltal nagyobb termékpalettát tudna felhasználni a felhasználó, aki számára természetesen nyilvánvalóvá kell tenni, hogy ezen áruk csak tájékoztatóak, nem feltétlen tartalmazzák naprakészen az árakat, illetve üzleteket.

7. ÖSSZEFOGLALÓ

A szakdolgozatom témájának lefejlesztése számomra nagy kihívást jelentett, de végül rendkívüli örömmel tölt el, hogy az eredmény végül egyezett a célommal. Pozitív visszacsatolás volt, hogy az eszközök, melyeket használtam a fejlesztés során, nem hagytak cserben, a hozzáadott eddigi ismeret és az interneten talált hatalmas fejlesztői szaktudás segítségemre volt a felépítés során lehetetlennek tűnő feladatok végrehajtásában.

A nehézséget a gondolataim, megoldási meneteim leprogramozása jelentette, de a különféle IDE-k melyeket ingyenesen használhattam, megkönnyítették ennek kibontakozását. Örülök, hogy diákként ennyi új ajtó nyílt meg előttem, melyeket felhasználhattam a programozás során, úgy, mint a Pycharm-, VS Közösség IDE-k, vagy a különféle programozást segítő másodlagos kiegészítők, a Codeium, vagy a Github Copilot. A C# és Python nyelv használata számomra jó döntésnek bizonyult, mint alapnyelv, melyet könnyedén tudtam használni és javítani, ha probléma merült fel a használatukban.

Úgy érzem van bőven lehetőség az elkészített webware tovább fejlesztésére, ami az új elemek, funkciók integrálása után már akár élő, mindenki számára elérhető weboldalként funkcionálhatna, és egyszerűsíttethetné a mindennapokat. Szerencsésnek érzem magam, hogy ez választottam témának, mert nagy lehetőség rejlik benne, melynek akadályozó elemei csupán az idő, a pénz és az energiabefektetés, mellyel úgy gondolom sok más jelenlegi weboldal is így van ezzel

Összességében számomra a szakdolgozat témájának fejlesztése elégedetten tölt el, és sikerült eleget tennem a téma megjelölésének is, és bár a fejlesztése tele volt magas és mély ’repüléssel’, de ezek formálták át igazán az élményét, mely végül kellemes emlékként marad meg bennem.

8. SUMMARY

Developing my dissertation topic was a great challenge for me, which made me extremely happy to have a result that matched my goal. It was positive feedback, that my development tools, which I used for programming, did not let me down, previous knowledge and vast development expertise found on the internet helped me to fulfil my duty.

My difficulty was in my thoughts and knowledge of solutions, but other IDEs that I could use for free made the development process easier. I was glad to be as a student and to have new doors opening in front of me, that I could use them in programming, such as Pycharm, VS Community, or the secondary extensions, Codeium and Github Copilot. The C# and Python was turned out to be a good choice, as a base language, that I could easily use and debug, if some problem occurred during their use.

I feel that there is a lot of scope for further development of the webware, and the integration of the new elements and features would even make it as a live, production-ready site, available for everyone and would simplify everyday life I feel fortunate to have chosen this as my thesis topic because of the potential it holds, the only obstacles being the time, money and energy investment, which I believe many other currently available websites feel the same way about.

All in all, the development of the thesis topic makes me feel contented and managed to meet with the marking of the topic, although its journey was filled with ups and downs, they were the ones that really shaped my experience which ultimately remain as a pleasant memory.

Irodalomjegyzék

[1] C. Shanahan, Deep nutrition: Why your genes need traditional food, Flatiron Books, 2017.

[2] „Árfigyelő," Gazdasági Nemzeti Hivatal, 01 07 2023. [Online]. Available: https://arfigyelo.gvh.hu. [Hozzáférés dátuma: 25 04 2024].

[3] R. C. Balázs, „A Gazdasági Versenyhivatal inflációellenes munkájának bemutatása," *POLGÁRI SZEMLE: GAZDASÁGI ÉS TÁRSADALMI FOLYÓIRAT,* %1. kötet19, %1. szám4-6, pp. 82-118, 2023.

[4] *AZ EURÓPAI PARLAMENT ÉS A TANÁCS 178/2002/EK RENDELETE (2002. január 28.) az élelmiszerjog általános elveiről és követelményeiről, az Európai Élelmiszerbiztonsági Hatóság létrehozásáról és az élelmiszerbiztonságra vonatkozó eljárások megállapításáról,* 2002.

[5] „Stack Overflow," Stack Overflow, 01 2016. [Online]. Available: https://survey.stackoverflow.co/2016. [Hozzáférés dátuma: 25 04 2024].

[6] „Stack Overflow," Stack Overflow, 05 2023. [Online]. Available: https://survey.stackoverflow.co/2023. [Hozzáférés dátuma: 25 04 2024].

[7] Microsoft, „.NET and .NET Core Official Support Policy," Microsoft, 9 04 2023. [Online]. Available: https://dotnet.microsoft.com/en-us/platform/support/policy/dotnet-core. [Hozzáférés dátuma: 25 04 2024].

[8] „Codeium • Free AI Code Completion & Chat," Exafunction, 2024. [Online]. Available: https://codeium.com/. [Hozzáférés dátuma: 02 05 2024].

[9] „Github Copilot Your AI Pair programmer," 2024, [Online]. Available: https://github.com/features/copilot. [Hozzáférés dátuma: 02 05 2024].

[10] „Github Education," Microsoft, 2024. [Online]. Available: https://education.github.com/. [Hozzáférés dátuma: 25 04 2024].

[11] „DBeaver Community | Free Universal Database Tool,” 2024, [Online]. Available: https://dbeaver.io/. [Hozzáférés dátuma: 02 05 2024].

[12] „Pycharm: the Python IDE for data science and web development,” JetBrains, 2024. [Online]. Available: https://www.jetbrains.com/pycharm/. [Hozzáférés dátuma: 02 05 2024].

[13] „JetBrains Community Programs,” JetBrains, 2024. [Online]. Available: https://www.jetbrains.com/community/education/#students. [Hozzáférés dátuma: 25 04 2024].

[14] „Open Food Facts,” [Online]. Available: https://world.openfoodfacts.org/. [Hozzáférés dátuma: 30 04 2024].

[15] NumFOCUS, „Pandas Documentation, Version 2.2.2,” NumFOCUS, 10 04 2024. [Online]. Available: https://pandas.pydata.org/docs/index.html. [Hozzáférés dátuma: 30 04 2024].

[16] F. Lib, „Regex 101,” 2024. [Online]. Available: https://regex101.com/. [Hozzáférés dátuma: 25 04 2024].

[17] „Database Contents License (DbCL) v1.0,” Open Knowledge Foundation, [Online]. Available: https://opendatacommons.org/licenses/dbcl/1-0/. [Hozzáférés dátuma: 01 05 2024].

[18] „CC BY-SA 3.0 DEED,” Creative Commons, [Online]. Available: https://creativecommons.org/licenses/by-sa/3.0/deed.hu. [Hozzáférés dátuma: 01 05 2024].

[19] „Stack Overflow,” 2024. [Online]. Available: https://stackoverflow.com/questions/27928/calculate-distance-between-two-latitude-longitude-points-haversine-formula#27943. [Hozzáférés dátuma: 10 05 2024].

[20] J. Galloway, P. Haack, B. Wilson és K. S. Allen, Professional ASP. NET MVC 4, John Wiley & Sons, 2012.

[21] „Chart.JS | Open Source HTML5 Charts for your websites,” 2024. [Online]. Available: https://www.chartjs.org/. [Hozzáférés dátuma: 11 05 2024].

[22] „Stack Overflow | How to create a gradient fill line chart in latest Chart JS version (3.3.2)?,” 2021. [Online]. Available: https://stackoverflow.com/questions/67812003/how-to-create-a-gradient-fill-line-chart-in-latest-chart-js-version-3-3-2. [Hozzáférés dátuma: 11 05 2024].

[23] „Webflow: Create a custom website | Visual website builder,” WebFlow, Inc., 2024. [Online]. [Hozzáférés dátuma: 02 05 2024].

9. ÁBRAJEGYZÉK