ÓBUDAI EGYETEM
ÓBUDA UNIVERSITY

NEUMANN JÁNOS
INFORMATIKAI KAR

SZAKDOLGOZAT

OE-NIK Hallgató neve: **Buzási Simon**
2022 Hallgató törzskönyvi száma: **T-007015/FI12904/N**

Óbudai Egyetem
Neumann János Informatikai Kar
Szoftvertervezés és -fejlesztés Intézet

SZAKDOLGOZAT FELADATLAP

Hallgató neve: **Buzási Simon**
Törzskönyvi száma: T/007215/FI12904/N
Neptun kódja: [redacted]

A dolgozat címe:

Optimalizált élelmiszer felhasználási alkalmazás készítése
Creating an optimised food consumption application

Intézményi konzulens: Sipos Miklós László
Külső konzulens:

Beadási határidő: 2022. május 15.

A záróvizsga tárgyai: Szoftvertechnológia és grafikus felhasználói interfész tervezése
Szoftvertervezés és -fejlesztés specializáció

A feladat

A feladat egy webalkalmazás készítése optimalizált ebédmenü generálásához a háztartásban fellelhető alapanyagok felhasználásával. A szerver oldali alkalmazást ASP.NET Core segítségével készítse el. Vizsgálja meg az aktuális kliens oldali alkalmazás trendeket és technológiákat, amelyek alapján hozza meg döntését a felhasznált technológiát illetően. Mutassa be, hogy mely algoritmusok segítségével valósítható meg az optimalizált generálás, illetve a köztük lévő különbségeket. Az adott algoritmus melletti döntését támassza alá. Az optimalizálás során magas prioritással vegye figyelembe a különböző alapanyagok lejárati dátumát. A rendszer biztosítsa, hogy a felhasználók rögzíteni tudják az otthonukban fellelhető alapanyagokat, illetve saját receptjeiket. A rendszertől elvárás, hogy a felhasználók nyomon követhessék élelmiszerfelhasználási szokásaikat statisztikák segítségével. Végezzen továbbá elemzést a tesztelési lehetőségekről és készítse is el az egyes teszt típusokat és teszteseteket.

A dolgozatnak tartalmaznia kell:

- a megoldandó feladat pontos leírását és részletes elemzését,
- magas szintű szerkezeti mintát az alkalmazásról,
- az egyes funkciók megvalósításához felhasználható lehetséges technológiákat,
- az elkészített rendszer tervét, blokkdiagramját,
- a megvalósítás menetét, felmerülő problémákat és az azokra adott megoldást,
- a tesztelési tervet, a tesztelés módszertanát és a tesztelés eredményeit,
- az elkészült szoftver felhasználási lehetőségeit.

ÓBUDAI EGYETEM * Neumann János Informatikai Kar * 1. Budapest

.....
Dr. Vámossy Zoltán
intézetigazgató

A szakdolgozat elévülésének határideje: **2024. május 15.**
(OE TVSz 55.§ szerint)

A dolgozatot beadásra alkalmasnak tartom:

.....
külső konzulens

.....
intézményi konzulens

HALLGATÓI NYILATKOZAT

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

Budapest, 2022. 05. 10.

.....

hallgató aláírása

ÓBUDAI EGYETEM
ÓBUDA UNIVERSITY

Neumann János Informatikai Kar
Neumann Faculty of Informatics

KONZULTÁCIÓS NAPLÓ

Hallgató neve:	Neptun kód:	Tagozat:
BUZÁSI SIMON	[redacted]	NAPPALI

Telefon: Levelezési cím (pl: lakcím):

[redacted]

Szakdolgozat címe magyarul:

OPTIMALIZÁLT ÉLELMISZER FELHASZNÁLÁSI ALKALMAZÁS KÉSZÍTÉSE

Szakdolgozat címe angolul:

CREATING AN OPTIMISED FOOD CONSUMPTION APPLICATION

Intézményi konzulens:	Külső konzulens:
SIPOS MIKLÓS LÁSZLÓ	---

Alk.	Dátum	Tartalom	Aláírás
1.	2021.09.10.	Félév teendőinek, menetrendjének és mérföldköveinek ismertetése.	
2.	2021.10.18.	Hasonló rendszerek elemzése.	
3.	2021.11.22.	Irodalomkutatás, annak összegzése valamint konzekvenciák meghatározása.	
4.	2021.12.06.	Tervezés, követelmények specifikációja. Véglegesítés. Leadással kapcsolatos teendők egyeztetése.	

A Konzultációs naplót összesen 4 alkalommal, az egyes konzultációk alkalmával kell láttamoztatni bármelyik konzulenssel.

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

Budapest, 2021.12.06.

Sipos Miklós

intézményi konzulens

ÓBUDAI EGYETEM
ÓBUDA UNIVERSITY

Neumann János Informatikai Kar

KONZULTÁCIÓS NAPLÓ

Hallgató neve: BUZÁSI SIMON

Neptun Kód: [redacted]

Tagozat: NAPPALI TAGOZAT

Telefon: [redacted]

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

Szakdolgozat címe magyarul:

OPTIMALIZÁLT ÉLELMISZER FELHASZNÁLÁSI ALKALMAZÁS KÉSZÍTÉSE

Szakdolgozat címe angolul:

CREATING AN OPTIMISED FOOD CONSUMPTION APPLICATION

Intézményi konzulens: SIPOS MIKLÓS LÁSZLÓ

Külső konzulens: -

Alk.	Dátum	Tartalom	Aláírás
1.	2022.03.04.	**Fejlesztés tervezett menetének revíziója, szakmai egyeztetés, fejlesztési teendők megkezdése.**	
2.	2022.04.01.	**Fejlesztés állapotának és haladásának áttekintése. Technikai problémák és módosítások egyeztetése.**	
3.	2022.04.15.	**Felmerülő problémák egyeztetése és megoldása. Tesztelési fázis megindítása.**	
4.	2022.05.06.	**Dokumentáció véglegesítése. Leadással kapcsolatos teendők egyeztetése.**	

A Konzultációs naplót összesen 4 alkalommal, az egyes konzultációk alkalmával kell láttamoztatni bármelyik konzulenssel.

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

.....

Intézményi konzulens

Budapest, 2022.05.10.

ABSZTRAKT

A dolgozat célja egy olyan rendszer elkészítése, amelynek fő funkciója az optimalizált menügenerálás a háztartásban fellelhető alapanyagok felhasználásával a felhasználó receptjei alapján. A rendszer egy webalkalmazás, amely egy szerver-, illetve kliensoldali alkalmazásból áll. A szerveroldali alkalmazás ASP.NET Core keretrendszer és C# nyelv, a kliensoldali alkalmazás Angular keretrendszer és TypeScript nyelv segítségével készül.

A dolgozatban több különböző, az elkészítendő rendszerhez hasonló funkciókkal rendelkező alkalmazás kerül elemzésre. Folytatólag számos optimalizációs módszer és azok működése kerül megvizsgálásra a 0-1 hátizsák probléma megoldása során. A készítendő rendszer optimalizáló algoritmusa a visszalépéses keresés, amely a szerveroldali alkalmazás részeként kerül implementálásra.

Az esszé részét képezi a tervezés folyamata, és annak dokumentálása. A tervezés magába foglalja a rendszerterv elkészítését, az adatbázis megtervezését, az optimalizált menügeneráló algoritmus folyamatának bemutatását, a rendszertől elvárt funkciók listáját és a fejlesztési menetrendet heti bontásban. A tervezés középpontjában a menügenerálást végző komponens áll, mivel az adja az elkészítendő alkalmazás gerincét.

Bemutatásra kerül a fejlesztés menete, részletesen prezentálva a menügeneráló algoritmus elkészítését. Az algoritmus jóság függvényeinek meghatározása a legfontosabb feladat, ezért annak kidolgozási folyamata kerül a fókuszba. A fejlesztési napló tartalmazza továbbá a felmerült problémákat és azok megvalósított megoldását.

A dolgozat kiterjed az elkészített rendszer tesztelésére is, ami elengedhetetlen ellenőrzési folyamata az elvárásoknak. A kész alkalmazás megfelelőségének vizsgálata unit tesztekkel és manuális tesztekkel valósul meg, amelyeknek részletes leírása táblázatos formában megtalálható a dedikált fejezetben.

A fejlesztési és tesztelési feladatok teljesülését követően rövid összefoglalást tartalmaz a dolgozat, az elért eredmények értékelésének érdekében. Az összefoglaló értékelés magába foglalja a tervezett rendszer és az elkészített alkalmazás közötti eltéréseket. Az értékelést követően a dolgozat kitér a megvalósított alkalmazás továbbfejlesztési lehetőségeinek bemutatására.

ABSTRACT

The aim of this thesis is to develop a system that main function is the optimized menu generation based on the user's recipes using the ingredients available in the household. The system is a web application consisting of a server-side and a client-side application. The server-side application is built using ASP.NET Core framework and C# language, the client-side application is built using Angular framework and TypeScript language.

The paper analyses several different applications with similar functionalities to the system to be built. Further on, several optimization methods and their performance in solving the 0-1 knapsack problem are investigated. The optimization algorithm for the system to be developed is backtrack search, that is implemented as part of the server-side application.

The design process and its documentation are part of the essay. The procedure includes the preparation of a system design, the design of the database, the demonstration of the optimised menu generation algorithm process, the list of the features expected from the system and the development schedule. The focus of the design is on the menu generation component, as it is the backbone of the application to be built.

The presentation of the development process includes a detailed explanation of the menu generator algorithm's creation. The definition of the fitness functions of the algorithm is the most important task, hence its development process is highlighted. The development report will also include the problems encountered and their solutions provided.

The thesis also covers the testing of the completed system, which is an essential verification process of the requirements. Conformance testing of the completed application is carried out by unit tests and manual tests, that are explained in detail in the specific chapter.

As the development and testing tasks are accomplished, a short summary is included in the paper to assess the results achieved. The summary evaluation includes the differences between the designed system and the completed application. Following the evaluation, the paper will present options for further development of the implemented application.

TARTALOMJEGYZÉK

1. BEVEZETÉS

Magyarországon átlagosan több, mint 68 kilogramm az egy főre jutó éves kidobott ételmennyiség, amelynek közel 50%-a elkerülhető lenne. [10] Az általam elkészíteni kívánt alkalmazás fő célja az élelmiszerpazarlás megelőzése optimalizált menügenerálás segítségével. Ezen felül az applikáció célja a kényelmi igények kielégítése, hiszen az elkészítendő menü kitalálása nem mindig kézenfekvő.

Az irodalomkutatás során megvizsgálom az általam készítendő rendszerhez hasonló alkalmazásokat, illetve összehasonlítom azokat. Elemzem a különböző optimalizálási módszereket, mint például a visszalépéses keresés, az oszd meg és uralkodj stratégia stb. A módszerek bemutatását és értékelését követően több szempont alapján összehasonlítom ezeket egymással. Továbbá bemutatásra kerül egy, a menügeneráláshoz hasonló optimalizálási probléma, illetve a probléma megoldásai a különböző optimalizálási algoritmusok segítségével.

A tervezés folyamán bemutatásra kerül az elkészítendő alkalmazás felépítése. A rendszerterv tartalmazza az applikáció főbb komponenseit és a közöttük lévő kapcsolatokat, illetve a menügenerálást végző egység részletesebb kidolgozása is megtalálható. Ezt követi a rendszer mögött levő adatbázis terve, ami magába foglalja a szükséges adattáblák meghatározását és a közöttük lévő kapcsolatok minőségének meghatározását. A menügenerálás folyamata egy magasabb és egy alacsonyabb absztrakciós szinten is bemutatásra kerül, annak érdekében, hogy a működésével szembeni elvárások tisztázva legyenek. Végül a tervezés utolsó részeként meghatározásra kerül az alkalmazástól elvárt funkciók listája, illetve a fejlesztés tervezett menete.

A megvalósítás közben számos olyan probléma merült fel, amelyek a tervezési fázisban nem feltétlenül voltak előre meghatározhatók. A fejlesztés fejezet tartalmazza a felmerült nehézségeket és azok megoldását, illetve a fontosabb komponensek elkészítésének menetét. Többek között itt került dokumentálásra az optimalizált menügenerálás implementációjának leírása és az ahhoz tartozó jóságfüggvények meghatározása.

A fejlesztést követően az elkészült rendszer tesztelésére került sor. Az alkalmazás ellenőrzése unit tesztekkel és manuális teszteléssel történt, amelynek részletes leírását a tesztelés fejezet tartalmazza.

A rendszer elkészítése és annak tesztelése után értékeltem az elért eredményeket, hogy a tervezett rendszerhez képest milyen alkalmazás került megvalósításra. Továbbá elemeztem az elkészült rendszer továbbfejlesztési lehetőségeit mind minőségi, mind kezelhetőségi szempontok szerint.

2. IRODALOMKUTATÁS

2.1. Hasonló rendszerek

A hasonló rendszerek technológiai, illetve technikai megoldásait és a meglévő funkcióit elemeztem. A vizsgált rendszerek a SuperCook [1], a Mealime [2], és a Food Planner [3].

2.1.1. SuperCook

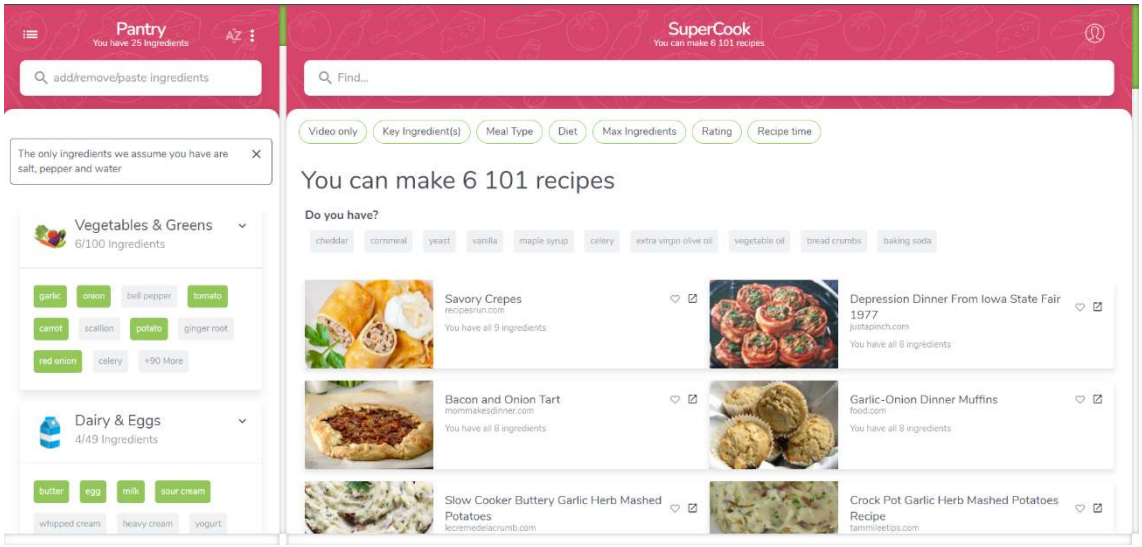
Technológiai és technikai oldal: A vizsgált rendszer alapvetően mobilos felhasználói felülettel rendelkezik, de elérhető böngészőből is. A két felület megegyezik, mind kinézet és funkcionalitás tekintetében. A webes felület a mobilos megjelenítést használja egy iframe segítségével. Az alkalmazás használatához elengedhetetlen az internetkapcsolat, a kommunikáció HTTPS protokollon keresztül történik. A SuperCook magyar nyelven nem elérhető.

Funkciók: Az alkalmazásban csak előre meghatározott alapanyagok közül lehet kiválasztani azokat, amelyeket felhasználva recepteket lehet keresni. Amennyiben a felhasználó nem létező hozzávalót szeretne kiválasztani, lehetősége van kérvényezni a fejlesztők felé annak hozzáadását. Ez a megvalósítás jelentősen rontja a személyre szabhatóságot. Az alapanyagok csoportosítva vannak, de mint entitások, egyszerűek, kizárólag megnevezésük van, mint például krumpli, oregánó, vaj stb.

Recepteket elsősorban a kiválasztott alapanyagok alapján keres a rendszer. Az alkalmazás számos weboldalon megtalálható receptötleteket ajánl a felhasználónak, ezáltal rengeteg recept közül választhat akkor is, ha kevés hozzávalót választott ki: 42 véletlenszerűen kiválasztott alapanyag esetén 7713 receptet ajánl. Néhány recept esetén előfordul, hogy a weboldal már nem létezik vagy nem elérhető. A találatokat szűrni is lehet több paraméter mentén, mint például diéta, értékelés, elkészítési idő stb. Lehetőség van a recepteket kedvencek közé menteni. Az alkalmazáson belül a receptek, mint entitások tartalmazzák az elkészítéshez szükséges hozzávalókat és azok mennyiségét, az elkészítési időt, értékelést, illetve a tápértékeket is, amennyiben azok lekérdezhetőek a receptet tartalmazó weboldalról. Az elkészítési útmutató elérése csak a külső weboldalak megnyitásával lehetséges.

Az alkalmazás rendelkezik bevásárlólista készítési funkcióval is. A felhasználó az applikációban szereplő alapanyagok közül tudja kiválasztani melyek kerüljenek fel a listára. A bevásárlólistában szereplő hozzávalókat meg lehet jelölni, ekkor átkerülnek a megvásárolt státuszba. A listából egyenként is el lehet távolítani a rendelkezésére álló alapanyagokat, valamint a felhasználónak lehetősége van megjelölni csak azokat, amelyeket már megvásárolt, illetve a teljes listát is törölheti. Az alkalmazás használatához

nem szükséges regisztrálni, de ebben az esetben nem menti el a felhasználó által kiválasztott alapanyagokat, bevásárlólistát és a kedvenceket.



2.1. ábra. A SuperCook webes felülete

2.1.2. Mealime

Technológiai és technikai oldal: A rendszer webes és mobil alkalmazással is rendelkezik. A két felület megegyezik a megjelenítést tekintve, de funkcionalitásukban vannak eltérések. A mobilra készült alkalmazásban számos funkció elérhető internetkapcsolat nélkül is, de a menü kiválasztása vagy generálása nem. A rendszer HTTPS protokollt használ az interneten történő kommunikációra. A felhasználóknak szükséges regisztrálniuk az alapvető funkciók használatához, továbbá lehetőségük van PRO verzióra előfizetni, aminek köszönhetően több funkciót használhatnak. Regisztráció nélkül csak a recepteket lehet böngészni. A rendszer kizárólag angol nyelven érhető el.

Funkciók: Az alkalmazás az első indítása során lépésenként vezeti végig a felhasználót az alapvető beállításokon. Először nyolc étrend közül lehet választani, mint például: klasszikus, vegán, keto stb. Második lépésként meghatározhatóak azok az alapanyagok, amelyekre a felhasználó allergiás, vagy nem kedveli. A kiválasztás során 124 gyakori hozzávaló van felsorolva, ezek között lehet keresni is. Következő lépésként a menüadagokat lehet beállítani, ahol 2, 4 vagy 6 fő választható. Utolsó lépésként opcionálisan heti emlékeztetőt lehet beállítani, de ez csak a mobilos alkalmazásban lehetséges. Az alapvető beállítások bármikor módosíthatók, illetve egyéb beállításokra is van lehetőség, mint például a mértékegységrendszer változtatására (metrikus – brit).

Az alkalmazásban a felhasználó heti menüt képes készíteni, receptek kiválasztásával. A receptekről minden szükséges információ elérhető az applikáción belül, mint például hozzávalók, elkészítési útmutató, szükséges eszközök stb., továbbá egyéni jegyzeteket is hozzá lehet rendelni. A felhasználó mentheti a recepteket a kedvencek közé, megkönnyítve ezzel a keresést.

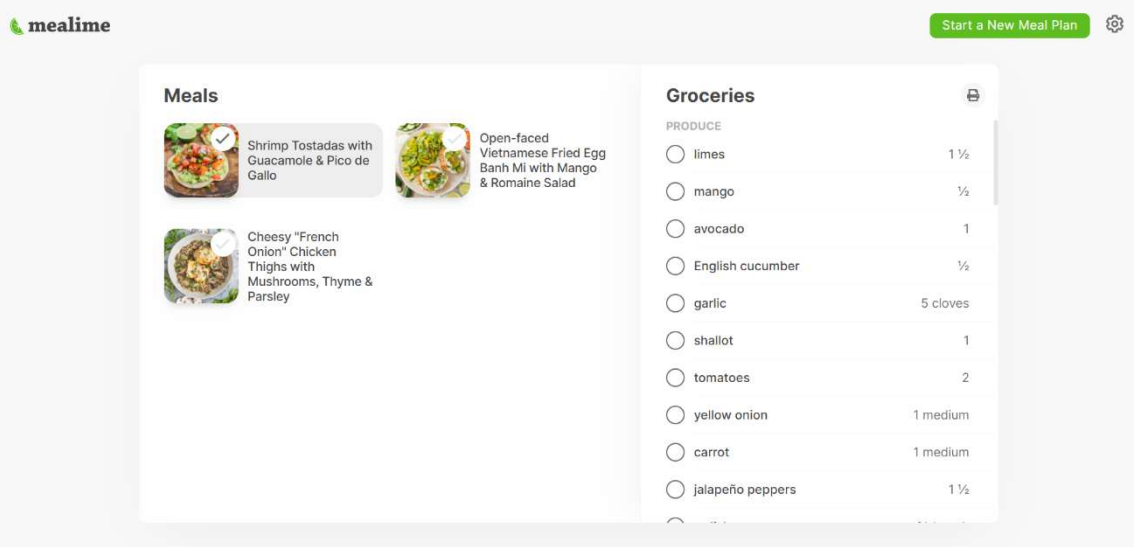
A menü tervezése során számos recept közül lehet választani, amelyek különböző szempontok alapján csoportosítva is vannak, mint például levesek, desszertek, tésztaételek stb. A mobil applikációban lehetőség van menü generálására is. A menügenerálás több menü-alternatívát ajánl a felhasználónak az alapbeállítások alapján. Alapértelmezetten 4 receptet választ ki az alkalmazás élelmiszerpazarlási statisztikák alapján. A felhasználó beállíthatja a receptek számát korlátlanul, szerkesztheti a generált menük összetételét, illetve megadhatja, hogy milyen alapanyagok állnak rendelkezésére. Lehetőség van további finomhangolásra is, mint például milyen mértékben vegye figyelembe a generátor az élelmiszerhulladék minimalizálását, a zöldség-hal-hús arányokat, illetve az ismétlődések sűrűségét. Az applikáció folyamatosan frissíti és tájékoztatja a felhasználót a becsült hulladékkeletkezésről. Miután a felhasználó jóváhagyta a generált menüt vagy kiválasztotta a számára megfelelő recepteket, az alkalmazás egy listában összefoglalja azokat, így azok áttekinthetőbbek és módosíthatók véglegesítés előtt.

Véglegesítés után a rendszer készít egy bevásárlólistát, amiben szerepel a kiválasztott receptekhez szükséges összes hozzávaló, és azok mennyisége. Az elkészített receptek megjelölhetők, de a generált bevásárlólista ennek hatására nem változik.

A bevásárlólistában megadható, hogy mely alapanyagokból mennyi áll a felhasználó rendelkezésére, ezáltal egy személyre szabott lista áll elő. Mind a webes, mind a mobil felületen lehetőség van a bevásárlólista nyomtatására.

A mobil applikációban a felhasználó online is tud vásárolni különböző kereskedőktől, de ez a funkció Magyarországról nem használható, mivel csak egyesült államokbeli boltok érhetőek el. Az online vásárlási lehetőségek egy része integrálva lett az alkalmazásba, de van amelyik egy alacsony funkcionalitású beépített böngészőn keresztül működik. A böngészős vásárlás esetén az alkalmazás a felhasználót lépésről lépésre végig vezeti a szükséges alapanyagok listáján, az online kereskedő oldalán pedig a megfelelő mennyiséget beállítva tud termékeket a kosárba helyezni.

A mobil alkalmazásban szerepel a *Főzés mód* funkció, amely a felhasználót végig vezeti az elkészítési útmutató lépésein. A funkció rendelkezik beépített időzítővel, amennyiben az adott lépés időigényes folyamat. A lépések közötti váltáshoz a telefonba épített távolságérzékelő szenzort is képes használni az applikáció: amennyiben a felhasználó körülbelül 2 másodpercig a szenzor fölött tartja a kezét, a következő lépésre ugrik az útmutatóban.



2.2. ábra. A Mealime webes felülete

2.1.3. Food Planner

Technológiai és technikai oldal: A rendszer rendelkezik webes alkalmazással és mobil applikációval is. A két felület jelentősen eltér egymástól, mind megjelenítés, mind funkciók szempontjából. Az alkalmazások HTTP protokollt használnak az interneten történő kommunikáció lebonyolítására. Regisztráció nélkül nem használható sem a webes, sem a mobil alkalmazás, valamint lehetőség van PRO verzióra előfizetni több funkció eléréséért. A webalkalmazás csak angolul elérhető, míg a mobil applikáció félig angol, félig magyar nyelven van. A nyelvi beállítások módosíthatók, de ez a funkció nem működik. Az alkalmazásban definiált hibaüzenetek nem tartalmaznak értelmezhető adatot arról, hogy mi a hiba. Mivel a mobil applikáció rengeteg hibát tartalmaz és nincs szinkronban a webalkalmazással, ezért a webalkalmazás funkcióit fogom bemutatni.

Funkciók: Az alkalmazásban lehetőség van a háztartásban található alapanyagok eltárolására. A tárolt alapanyagok a következő tulajdonságokkal rendelkeznek: név, mennyiség, mértékegység, lejárati dátum, csoport. Ezek az adatok bármikor szerkeszthetőek, illetve a csoport tulajdonság tetszőlegesen választható, a felhasználó határozza meg milyen csoportok legyenek. A tárolt elemeket az alkalmazás nem mindig tudja megjeleníteni, hibára fut a rendszer az esetek többségében.

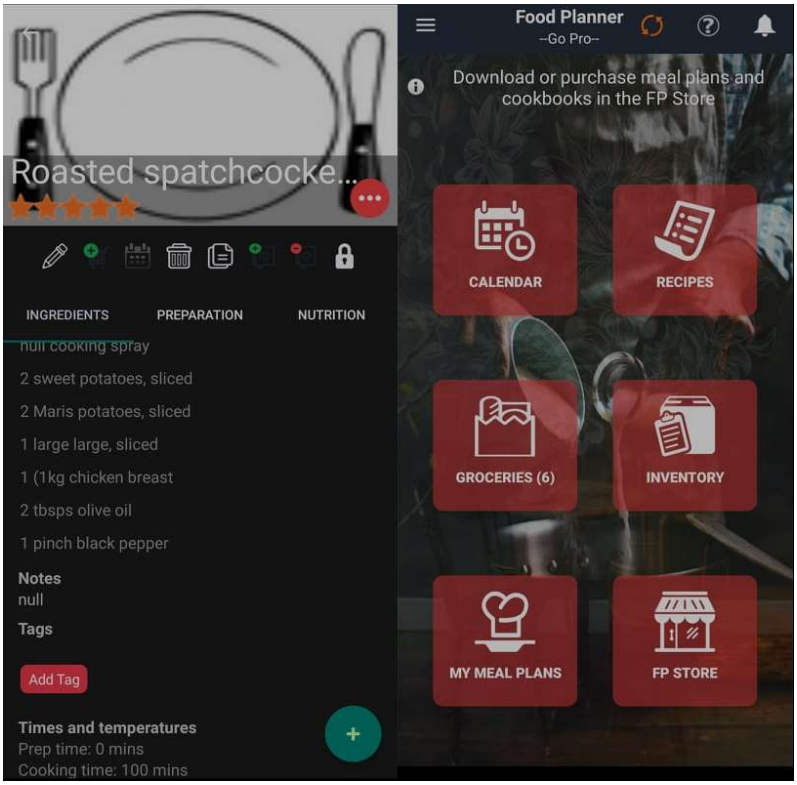
Lehetőség van receptkönyvek létrehozására, melynek segítségével a felhasználó rendszerezni tudja a különféle receptjeit. Receptek hozzáadására kétféle opció van, az egyik a saját recept létrehozása, a másik receptek importálása. Létrehozásnál a következő tulajdonságokat definiálhatja a felhasználó: recept neve, kép a receptről, adagok száma, elkészítési idő, hozzávalók, útmutató leírása, egyéni jegyzetek, illetve egyedi jelölők. Importálás csak meghatározott weboldalakról lehetséges, amelyek listája elérhető az alkalmazásban. A recept importálás során a hozzávalók beolvasása nem működik tökéletesen, a beolvasott értékek nem a megfelelő helyre kerülnek, ezért a felhasználónak szükséges

azokat felülvizsgálnia. A receptek között lehet keresni, név, illetve jelölők szerint, továbbá lehetőség van bármely receptet kinyomtatni.

A webalkalmazás rendelkezik egy naptár nézettel, ahol a menütervezésre van lehetőség. A naptár esetében választható napi, heti, illetve havi nézet. A recepteket drag-and-drop módszerrel lehet hozzárendelni a napokhoz. Megadható, hogy a receptet az adott nap melyik étkezéséhez (reggeli, ebéd vagy vacsora) rendelje hozzá az alkalmazás. Bármely étkezéshez tetszőleges recept választható, illetve definiálható egyedi étkezés is, mint például uzsonna, tízórai stb. A napi és heti nézetben a felhasználó mentheti az összeállított menütervet, illetve a mentett terveket betöltheti tetszőlegesen kiválasztott napra vagy hétre. A betöltés során megjelölhetők azok a receptek, amelyeket szeretne majd lecserélni a felhasználó. Az elkészült menütervet bármely nézetben lehet nyomtatni.

Bevásárlólista összeállítására is képes a rendszer, de felhasználói interakció szükséges hozzá. A listához hozzáadhatjuk bármely recept hozzávalóit, de lehetőség van egyesével is hozzáadni tetszőleges alapanyagokat. A webes felületről a lista kinyomtatható, míg a mobil applikációban interaktív a lista, tehát ki lehet húzni a megvásárolt elemeket. A listában az alapanyagok mennyisége szerepel, de a mértékegységük nem, illetve a webes és mobil felületeken megjelenő listák különböznek, nincsenek szinkronban.

A Food Planner-nek hátránya, hogy számos hibával rendelkezik, mind a webalkalmazás, mind a mobil applikáció, ez jelentősen rontja a felhasználói élményt és az alkalmazások használhatóságát.



2.3. ábra. A Food Planner mobilos felülete

2.1.4. Hasonló rendszerek összefoglalása

Mindegyik megvizsgált rendszer rendelkezik olyan tulajdonságokkal, amelyeket fel tudok használni a saját alkalmazásomban. A főbb szempontokat egy 0-10-ig terjedő skálán pontoztam, aminek alapja, hogy az adott szempontból mennyire áll közel a saját rendszeremhez. Ezeket a 2.1. táblázat foglalja össze, amely alapján az általam készítendő alkalmazás megjelenése a Mealime-ra, míg a személyre szabhatóság tekintetében a Food Planner-re hasonlítana leginkább.

Szempont / Rendszer	Felhasználói felület	Optimalizált menügenerálás	Személyre szabhatóság	Összesen
SuperCook	7	0	3	10
Mealime	**8**	**7**	7	**22**
Food Planner	2	0	**9**	11

2.1. táblázat. Hasonló rendszerek összehasonlítása

A hasonló rendszerek mindegyike alkalmas valamilyen formában menü tervezésére, de menügenerálásra, a Mealime kivételével, egyik sem képes. A Mealime menügenerálója statisztikák alapján ajánl recepteket, de alapértelmezetten nem veszi figyelembe a háztartásban fellelhető alapanyagokat és azok lejárati dátumát.

2.2. Optimalizációs módszerek

Optimalizáció során feltételezzük, hogy az adott problémának számos megoldása van. A cél az optimum meghatározása az adott feladatra, de ez függ a feladat típusától, azaz nincsen rá általános képlet. A megfelelő megoldás kiválasztásához jóság/fitnesz vagy költség függvényt definiálnak általában. A függvények célja, hogy számszerűsítse, hogy az egyes megoldások mennyire optimálisak. A jóság/fitnesz függvény visszatérési értéke minél nagyobb, annál közelebb áll a megoldás az optimumhoz. A költség függvény esetében a minél alacsonyabb visszatérési érték jellemzi az optimumhoz közeli állapotot. Optimális megoldásból akár több is előfordulhat, amennyiben ezek jósága vagy költsége megegyezik és nem találtunk jobbat. [4]

2.2.1. Nyers erő módszer (*brute force*)

Egyszerűen alkalmazható stratégia, de cserébe az erőforrásigénye hatalmasra duzzadhat feladattól függően. Alapvető elvárás a használatához, hogy az adott feladatnak véges számú megoldása legyen. A módszer lényege, hogy az összes lehetséges megoldást megvizsgálva megkeresi a probléma tényleges megoldását, amennyiben van ilyen. Többnyire egy ciklusból áll, amelyben a lépések a következők:

1. A soron következő megoldás előállítása.
2. A lehetséges megoldás elemzése.

3. Ha valódi megoldás, akkor
    a. kilép a ciklusból, amennyiben csak egy megoldást kerestünk,
    b. összehasonlítjuk az előző megoldás jóságával, amennyiben optimalizálni szeretnénk.
4. Ha van még lehetséges megoldás, akkor tovább vizsgáljuk a megoldások halmazát.

Az algoritmus egyszerű és gyorsan megvalósítható, továbbá minden esetben alkalmazható, ha a részmegoldások száma véges. Az abszolút megoldás megtalálása garantált, amennyiben létezik, csak idő és teljesítmény kérdése. Az algoritmus hátránya, hogy a futásidő meglehetősen nagy lehet. [4]

2.2.2. Oszd meg és uralkodj paradigma (*divide-and-conquer*)

A módszer két feladattípust különböztet meg, az egyszerűen megoldhatókat, illetve az összetett problémákat. Ha a megoldandó probléma egyszerű, az algoritmus megoldja a feladatot és előáll az eredménnyel, ezzel szemben az összetett feladatok megoldását több lépésben végzi el. A lépések a következők:

1. A teljes feladat feldarabolása kisebb problémákra.
2. A töredék problémák megoldása, általában rekurzív módszerrel.
3. A részeredmények összegzése és a feladat abszolút megoldásának prezentálása.

Az algoritmus optimalizációs feladatok megoldására is alkalmazható, az alapelv nem különbözik.

A módszer futásideje függ a bemenő adatok mennyiségétől, továbbá a rekurzív megvalósítás miatt jelentősen megnőhet. A lépések száma általában kisebb, mint a nyers erő módszerének alkalmazása esetén. Hátránya, hogy nem minden feladattípusnál alkalmazható, illetve többször is megvizsgálja ugyanazokat a töredék problémákat. [4]

2.2.3. Feljegyzéses módszer

Alapvetően az Oszd meg és uralkodj elvű algoritmusok hátrányát igyekszik kiküszöbölni. A módszer lényege, hogy a részeredményeket egy tetszőleges adatszerkezetben (tömb, lista, halmaz stb.) tárolja, ezzel segítve az algoritmust, hogy a már meghatározott részeredményeket ne számítsa ki újból. Mielőtt az algoritmus újból kiszámolná az adott részeredményt, megvizsgálja a feljegyzett eredmények gyűjteményt, ha szerepel a keresett érték akkor azzal dolgozik tovább, egyéb esetben kiszámítja a részmegoldás értékét majd eltárolja azt.

A feljegyzéses stratégia működéséből fakadóan csökkenti a lépésszámot, de cserébe az algoritmus tárhelyigénye nő. Érdemes akkor használni ezt a módszert, ha a részeredmények gyakran számítódnak újra. Általában az Oszd meg és uralkodj elvű algoritmusok gyorsítására használják. [4]

2.2.4. Visszalépéses keresés alapelve és fajtái (*backtrack*)

A visszalépéses keresés egy rendszerezett módja annak, hogy az összes lehetséges konfigurációt megvizsgáljuk egy probléma keresési terében. [6]

A módszer effektív működésének feltétele, hogy az elvégzendő feladat felbontható olyan részfeladatokra, amelyek megoldása egymástól közvetlenül nem függ, illetve a részmegoldásokból megállapítható, hogy a teljes megoldás nem elérhető a vizsgált ágon. Az előbb felsorolt feltételeknek, nem kötelező megfelelnie a megoldandó problémának, de így biztosítható az algoritmus hatékony működése. Kedvező feltételek mellett a keresés által vizsgált tér mérete meglehetősen zsugorodhat, ez pedig jelentős performancianövekedést idézhet elő. [4]

A stratégia alaptétele, hogy az algoritmus egyesével vizsgálja meg a részfeladatokat. A visszalépéses keresés az első részfeladatra ad egy elfogadható megoldást, majd egy szinttel feljebb lép és elkezdi a következő részproblémát vizsgálni. Amint talált egy, az adott szinten elfogadható részeredményt, ellenőrzi, hogy a korábbi részmegoldásokkal ez összhangban van-e, ha igen halad a következő szintre és így tovább. Amennyiben az algoritmus egy olyan szinten vizsgálódik, ahol nem talált olyan megoldást, ami megfelelne a korábbi választásoknak, akkor visszalép egy szintet és keres egy másik elfogadható eredményt. Ha véges részfeladatra bontható az elvégzendő feladat, akkor az algoritmus véges számú iteráció során adhat megoldást az összes részproblémára. Amennyiben az összes részmegoldás kielégíti a hozzá tartozó részfeladatot, akkor az algoritmus megtalálta a keresett megoldást, ha viszont bekövetkezik az, hogy az első részfeladat minden lehetséges részmegoldása elemzésre került és a függvény már onnan is visszalépne, akkor kijelenthető, hogy az eredeti problémának nincs megoldása. [4]

A keresés általában rekurzív algoritmus segítségével történik. A rekurzió olyan folyamat, amely a problémát önmagában határozza meg egy egyszerűbb változatban. A végrehajtás során az algoritmusban deklarált utasítások több alkalommal is kiértékelésre kerülnek. Bármely rekurzív algoritmus az alábbi lépésekből áll:

1. Alapeset: ez lehet a kilépési feltétel meghatározása, vagy annak megállapítása, hogy a feladatnak nem létezik megoldása.
2. Haladás az alapeset felé: igyekszik a problémateret addig csökkenteni, amíg el nem éri az algoritmus az alapesetet, ahol könnyedén meghatározható a megoldás.
3. Rekurzív függvényhívás: biztosítja, hogy az egyszerűsített problématérben tovább tudjon vizsgálódni az algoritmus mígnem eléri az alapesetet.

A rekurzió a verem asszisztálásának köszönhetően működik. Minden egyes hívási szint információi bekerülnek a verembe, amint a következő rekurzív függvényhívás bekövetkezik. Ha az algoritmus elérte a rekurzió legalsó szintjét, a veremből visszatöltődnek az adatok a memóriába, LIFO (*last in, first out*) elven alapulva, és azokkal tovább tud dolgozni a számítógép. [4] [5]

A visszalépéses keresés rekurzív mivoltából fakadóan előre nem meghatározható, hogy jó részmegoldás lett-e kiválasztva, de az előzőleg választott részeredmények alapján következtetni lehet az aktuális részmegoldás helyességére. [4]

Az általános visszalépéses keresés algoritmusának pszeudókódja a 2.4. ábrán található, ami alapján az algoritmus bemenete és kimenete az alábbiak szerint alakul:

Bemenet:

- N: A megoldandó részproblémák számossága.
- M_k: A k-adik szinten a kiválasztható részeredmények számossága, ahol $1 \leq k \leq N$.
- $R_{k,i}$: A k-adik szinten az i-edik részeredmény, ahol $1 \leq k \leq N$ és $1 \leq i \leq M_k$.

Kimenet:

- létezik: logikai érték, amely tartalmazza, hogy a feladatnak létezik-e teljes megoldása.
- E: a részeredményeket tároló vektor, ahol a tároló n-edik eleme az n-edik szinten rögzített részmegoldást tartalmazza.

Az általános algoritmus kétféle keresési feltételt határoz meg, amelyek biztosítják, hogy különböző típusú feladat megoldására is alkalmazható legyen az elv. A két feltétel a következő:

- $F_t(k, j)$: olyan függvény, ami eldönti, hogy *j* megoldás a *k*-adik szinten elfogadható-e.
- $F_k(k, j, E)$: olyan függvény, ami meghatározza, hogy a *j* részeredmény kielégítő megoldás-e a *k*-adik szinten figyelembevéve a korábbi szinteken választott megoldásokat, amelyeket az *E* vektor tartalmaz.

A két keresési feltételt megvalósító függvény között tehát a különbség az, hogy az F_t függvény azt hívatott eldönteni, hogy az adott részproblémára megoldás-e a vizsgált részmegoldás, míg az F_k függvény az előzmények ismeretében dönti el, hogy a részeredmény ellentmondásba ütközik-e a korábban választott részmegoldásokkal. Az előzőekből adódóan az F_t függvény elhagyható abban az esetben, ha minden szinten csak azok a részeredmények kerülnek felsorolásra, amelyek biztosan kielégítenék az F_t függvényt, azaz bármely eshetőség során választhatók. [4]

Bemenet: $szint$ - **egész**, E - **T tömb**, van - **logikai**
Kimenet: E - **T tömb** ($az\ eredményt\ tartalmazza$), van - **logikai** ($van\text{-}e\ eredmény$)

```
1: eljárás VisszalépésesKeresés(szint : egész, címszerint E : T tömb, címszerint van : logikai)
2:     i ← 0
3:     ciklus amíg ¬van ∧ i < M_szint
4:         i ← i + 1
5:         ha F_t(szint, R_szint,i) akkor
6:             ha F_k(szint, R_szint,i, E) akkor
7:                 E_szint ← R_szint,i
8:                 ha szint = N akkor
9:                     van ← igaz
10:                különben
11:                    VisszalépésesKeresés(szint + 1, E, van)
12:                elágazás vége
13:            elágazás vége
14:        elágazás vége
15:    ciklus vége
16: eljárás vége
```

Felhasznált változók és függvények

- $szint$ - Az aktuális szint, ahol keresünk.
- N - A megoldandó részfeladatok száma.
- M_{szint} - A lehetséges részmegoldások száma a megadott szinten.
- $R_{szint,i}$ - Az i. részmegoldás a megadott szinten.
- E - Egy N elemű vektor, ami az eredményeket tárolja.
- van - Folyamatosan azt mutatja, hogy találtunk-e már teljes megoldást.
- F_t, F_k - A keresési feltételeket meghatározó függvények.

2.4. ábra. A visszalépéses keresés általános alakjának algoritmusa

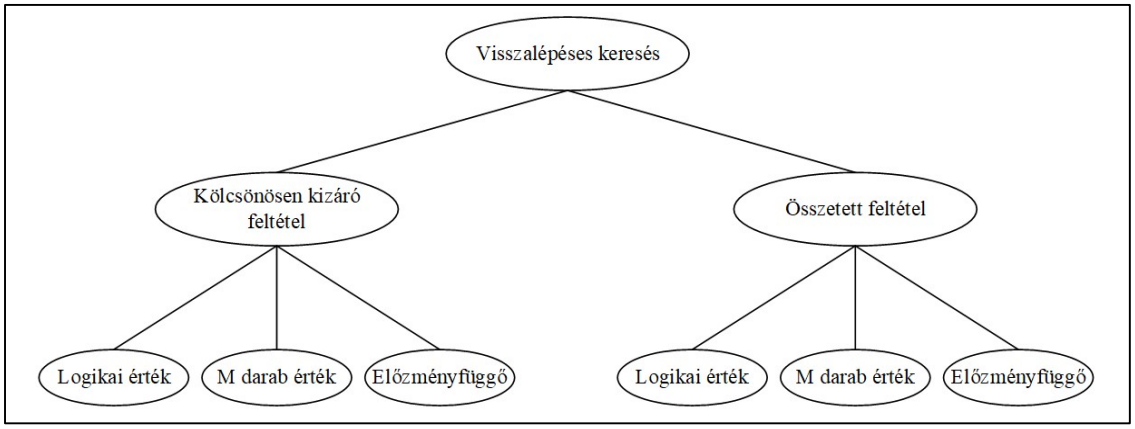
A visszalépéses keresés eddig taglalt formája általánosnak mondható ezért megvalósítása nehézkes lehet speciális esetekben. Az F_k függvény alapján megkülönböztethetünk két feladattípust, ahogy a 2.5. ábrán is látható:

- Kölcsönösen kizáró feltétel: az aktuálisan vizsgált részmegoldást elegendő a korábbi szinteken lévő részeredményekkel külön-külön összehasonlítani.
- Összetett kizárási feltétel: nem elég a korábban választott részeredményekkel összehasonlítani az aktuálisan vizsgált megoldást, hanem nagyobb kiterjedésben kell elemezni a különbségeket, azaz komplexebb vizsgálatra van szükség.

A keresés tovább csoportosítható aszerint, hogy a részfeladatoknál mennyi, illetve milyen típusú részeredmények közül lehet választani. A csoportok a 2.5. ábra szerint a következők:

- Logikai érték: minden részfeladatnál egyértelműen eldönthető, hogy az adott elem szerepel az eredmények között vagy nem.
- M darab érték: minden részfeladatnál egy vagy több részeredmény választható és feltételezzük, hogy a részeredmények gyűjteményei és azok számossága előre ismert.

- Előzményfüggő: minden részfeladat megoldása során a részeredmények halmaza függ a korábbi részfeladatok eredményeitől. [4]



2.5. ábra. Visszalépéses keresés csoportosítása

Az algoritmus alkalmazható optimalizációs problémák megoldására is. Ebben az esetben a keresési feltételeket úgy kell meghatározni, hogy azok minden lehetséges megoldást megvizsgáljanak és ezeknek az optimumhoz való közelségét egy jóság függvény segítségével meghatározza. Az eredmények közül a legnagyobb jóságú lesz az optimum. [4]

2.2.5. Dinamikus programozás (*dynamic programming*)

A dinamikus programozás egy olyan optimalizálási módszer, amely egy komplex problémát egyszerűbb problémák sorozatává alakít át. Az optimalizálást ebben az esetben is többlépcsős eljárás jellemzi. Az előzőekben bemutatott rekurzív algoritmusok a teljes probléma felől indultak a kisebb részproblémák megoldása felé, ezzel szemben a dinamikus programozás az egyszerűbb problémamegoldásokkal kezd és ezekből állítja elő a teljes megoldást. A stratégia általános keretet biztosít számos problématípus elemzéséhez. Gyakran kreativitásra van szükség mielőtt felismernénk, hogy az adott probléma megoldható dinamikus programozással, továbbá kifinomult meglátásokat is igényel, hogy a feladatot úgy alakítsuk át, hogy az hatékonyan megoldható legyen. [7]

A módszer alkalmazhatóságának érdekében a megoldandó feladat két alapkövetelménynek kell, hogy megfeleljen:

- A feladat legyen optimális részstruktúrájú: a probléma optimális megoldása a részfeladatokra adott optimális megoldások összessége legyen.
- Legyenek a részfeladatok átfedőek: a részfeladatok között legyenek olyanok, amelyek megoldása több alkalommal is felbukkan.

Rendszerint az alábbi lépésekből áll egy probléma megoldása dinamikus programozással:

1. Az optimális megoldás szerkezetének jellemzése.
2. A megoldás értékének rekurzív módon való definiálása.

3. Az optimális megoldás értékének kiszámítása alulról felfelé történő módon.
4. Amennyiben szükséges, magának az optimális megoldásnak a megadása.

Általában az algoritmus csak a megoldás értékét adja meg. Amennyiben szükség van a tényleges megoldásra is, azt az algoritmus kiegészítésével lehet előállítani. Az optimális megoldás a részmegoldás értékeit tartalmazó adatszerkezetből visszafejtéssel nyerhető ki. [4]

A nyers erő és a rekurzív módszerekhez képest nagyobb tárhelyigénye lehet, mivel előre nem tudható, hogy mely részeredmények kiszámítása lenne elhagyható. Az algoritmus viszonylag egyszerű, és a teljesítménye is kedvezőbb, mint a korábbi lehetőségek, mert nem használ rekurziót. A feljegyzéses módszerhez képest nagyobb tárhelyigénye lehet, amiért nem képes előre meghatározni, hogy mely számítások lennének elhagyhatók. [4]

2.2.6. Mohó algoritmusok

Általában egyszerűek, mindig a jelenleg legoptimálisabbnak tűnő megoldás irányába haladnak. Gyakran nem találják meg az optimumot, csak egy közelítő értéket adnak meg végeredményként. Megfelelően választott szabályrendszerrel lehet optimálisabb mederbe terelni, de ekkor sem biztos az optimum megtalálása minden esetben, mivel a szabályrendszer meghatározása meglehetősen bonyolult is lehet. Ha az optimum megtalálása nem szigorú elvárása a megoldandó feladatnak és elég egy becslés az optimális megoldásra, akkor alkalmazható ez a stratégia.

A feladatokkal szemben az alábbi követelményeket támasztják a mohó algoritmusok:

- Optimális részproblémák: az optimális eredmény előállítható a részfeladatok optimális megoldásából.
- Mohó választás: a globális optimum elérhető lokális optimumok megtalálásán keresztül.

A módszer hasonlít a dinamikus programozás elvére, azzal a kivétellel, hogy ez a módszer fentről lefelé halad a feladat megoldása során. Számít a vizsgálandó részfeladatok sorrendje, tehát kiinduláskor a megfelelő sorba rendezéssel segíthetjük az algoritmus hatékonyabb működését, ezáltal más-más szempontok szerint ad különböző optimumokat.

A mohó algoritmusok erőforrásigénye kedvezőbb, mint a korábban felsorolt módszereké, mind futásidő, mind tárhelyigény szempontjából. Általában jó becslést ad az optimális eredményre. Hátránya, hogy csak a feladatok nagyon szűk körében alkalmazható. [4]

2.3. A 0-1 hátizsák probléma (*0-1 knapsack problem*)

A probléma alapja, hogy adott N darab tárgy és egy zsák. A zsáknak van egy felső súlykorlátja (W), ami megadja, hogy a benne lévő tárgyak összsúlya mekkora értéket vehet fel. Minden tárgynak van súlya (w) és értéke (p), az egyszerűség kedvéért ezek pozitív egész számok. A tárgyak atominak tekinthetők, azaz vagy bekerülnek a zsákba, vagy nem. A feladat, hogy találjunk egy olyan érvényes pakolást, ahol a kiválasztott tárgyak összértéke a legnagyobb, ez lesz az optimális megoldása a problémának. Érvényes pakolásnak tekinthető bármely részhalmaza a tárgyaknak, amelyek összsúlya nem haladja meg a zsák súlykapacitását. [8]

2.3.1. A probléma megoldása különböző optimalizációs módszerekkel

A végeredmény minden esetben az optimális pakolás értéke. Kiegészítő függvény segítségével megadható a tényleges optimális pakolás. A probléma sajátosságából eredően mindegyik esetben érvényes pakolásnak minősül, ha nem rakunk egy tárgyat sem a zsákba. A 2.3. fejezetben meghatározott jelöléseket (N, W, w, p) alkalmazom a módszerek leírása során.

- Nyers erő módszere

  Az N darab tárgy esetében 2^N lehetséges megoldás van, mivel egy k elemű halmaz részhalmazainak a száma 2^k. A módszer megvizsgálja az összes lehetséges megoldást. A nem érvényes pakolásokat nem értékeli ki, azaz nem határozza meg a pakolás összértékét, mivel az a megszorítások miatt nem lenne megoldása a feladatnak. Az érvényes pakolások esetén kiszámítja a kiválasztott tárgyak összértékét, majd összehasonlítja azt a korábban optimálisnak választott értékkel. Ha az új összérték nagyobb, akkor onnantól az számít az optimális megoldásnak.

  Ez a stratégia biztosan megtalálja az optimumot, de minél nagyobb N értéke, annál tovább fog tartani és annál több erőforrást igényel. Az algoritmus futásideje $O(2^N)$, amennyiben feltételezzük, hogy az összes részhalmaz ismert. [4]

- Oszd meg és uralkodj

  A probléma feldarabolása ennél a feladatnál azt jelenti, hogy egyszerre csak egy tárggyal foglalkozunk, azaz betesszük-e a zsákba vagy sem, és a többi *N-1* darab tárgyat is vizsgáljuk. A problémát rekurzió segítségével oldja meg a paradigmán alapuló algoritmus. A rekurzív megoldás képletét a 2.6. ábra mutatja, ahol $F_{t,h}$ a rekurzív függvény, t - a részmegoldás keresésekor a tárgyak száma, h - a részmegoldás keresésekor fennmaradt szabad hely a zsákban, w_t - a t-edik tárgy súlya, p_t - a t-edik tárgy értéke.

$$F_{t,h} = \begin{cases} 0, & \text{ha } h = 0 \\ 0, & \text{ha } t = 0 \\ F_{t-1,h}, & \text{ha } h > 0 \wedge t > 0 \wedge h < w_t \\ max\{F_{t-1,h},\ F_{t-1,h-w_t} + p_t\}, & \text{ha } h > 0 \wedge t > 0 \wedge h \geq w_t \end{cases}$$

2.6. ábra. 0-1 hátizsák probléma rekurzív megoldásának képlete

Az egyértelmű esetek meghatározása:

- Ha nincsenek tárgyak, akkor az optimum 0.
- Ha a zsák súlykapacitása 0, akkor az optimális megoldás is 0.

Nem egyértelmű esetek:

- Ha nem lehet az utolsó, még nem vizsgált tárgyat berakni a zsákba, akkor meg kell vizsgálni, hogy a fennmaradó tárgyak hogyan pakolhatók be a zsákba optimálisan.
- Ha az utolsó, még nem vizsgált tárgyat is be lehet rakni a zsákba, akkor azt kell ellenőrizni, hogy az adott tárggyal együtt milyen pakolási érték érhető el. Ebben az esetben vizsgálni kell, hogy az adott tárggyal együtt vagy anélkül érhető el az optimum.

Az algoritmus futásideje O(2^N), de ennél a legtöbb esetben jobb, mivel az eleve rossz megoldásokat nem vizsgálja. [4]

- Feljegyzéses módszer

A megoldás felépítése hasonló, mint az *Oszd meg és uralkodj* elvnél, az eltérés a részeredmények feljegyzése és keresése. Mielőtt az algoritmus elkezd kiszámolni egy részeredményt, megvizsgálja, hogy az aktuális részmegoldás szerepel-e már a tárol eredmények között. Ha igen, akkor azzal számol tovább, amennyiben nem, akkor kiszámítja azt majd feljegyzi.

Az algoritmus futásideje O($N * W$). Az *Oszd meg és uralkodj* elvű algoritmushoz képest sokkal jobb a futásidő, mivel nem számítja ki újra az azonos részeredményeket. [4] [9]

- Visszalépéses keresés

A megoldás menete a korábban ismertetett elveket alkalmazza, az általános alak minimális átalakításával. A megoldandó részfeladatok ebben az esetben azok, hogy az aktuálisan vizsgált tárgy bekerül-e a zsákba, vagy sem.

Az F_t függvény azt vizsgálja, hogy a vizsgált tárgy súlya önmagában meghaladja-e a zsák súlykapacitását. Ha feltételezzük, hogy minden vizsgált tárgy súlya kevesebb, mint a zsák kapacitása, akkor ez a függvény elhagyható. Az F_k függvény azt vizsgálja, hogy az aktuálisan kiválasztott tárgy és a korábban kiválasztásra került tárgyak összsúlya meghaladja-e a zsák kapacitását. Ha igen, akkor a

megoldást helytelennek tekinti, ezzel szemben, ha nem, akkor a pakolást helyesnek ítéli. Amennyiben az F_k függvény érvényesnek ítéli az aktuálisan vizsgált pakolást, akkor két irányba haladhat tovább a keresés. Ha elért a legfelső szintre, akkor talált egy érvényes pakolást, ha viszont nem érte el a legfelső szintet, akkor meghívja az algoritmus önmagát és egyel magasabb szinten vizsgálódik. Feltéve, hogy a keresés talált egy érvényes pakolást, akkor annak az összértékét összehasonlítja a korábban optimumnak választott megoldás értékével. Amennyiben az új pakolás értéke meghaladja a korábbiét, akkor onnantól az új eredmény számít optimálisnak, ezzel szemben, hogyha az új pakolás értéke kisebb a korábbinál, akkor az algoritmus tovább vizsgálódik. Az algoritmus legalább addig fut, amíg nem vizsgálta meg az összes érvényes pakolást, mivel itt a probléma nem egy lehetséges megoldást vár, hanem az összes megoldás közül a legjobbat.

Az algoritmus futásideje legrosszabb esetben O(2^N). [4]

- Dinamikus programozás

Elsősorban meg kell vizsgálni, hogy a probléma megfelel-e a módszer által támasztott alapkövetelményeknek.

- Optimális részstruktúra: a feladat megfelel ennek a követelménynek, ez jól látható a rekurzív megoldási képletből, mivel egyre kevesebb tárgy esetében kell az optimumot keresni.
- Részfeladatok átfedőek: az *Oszd meg és uralkodj* stratégia alkalmazása a 0-1 hátizsák problémára ezen okból lehet lassabb, mivel adott részproblémát többször is megold.

A megoldás az alábbi lépések szerint történik:

1. Az optimumot tartalmazza egy N elemű logikai tömb, amelynek i-edik eleme megadja, hogy az i-edik tárgy szerepel-e a zsákban.
2. A feladat rekurzív megoldása a 2.6. ábrán látható.
3. Egy kétdimenziós táblázat kitöltésének segítségével állítható elő a megoldás.
4. Az optimális pakolás megadható egy segédfüggvénnyel, amely a mátrixot visszafejtve feltölti az N elemű logikai tömböt a megfelelő értékekkel.

Az algoritmus tárhelyigénye jelentősen megnőhet sok bemenő adat esetén, futásideje O($N * W$). Ha W túl nagy N-hez viszonyítva, az rosszabb futásidőt eredményezhet, mint a *nyers erő* módszer futásideje. Ennek elkerülése érdekében az algoritmust tovább lehet finomítani, úgy, hogy futásideje legrosszabb esetben megegyezzen a *nyers erő* módszer futásidejével. A tovább finomított algoritmus futásideje O(minimum($N * W$, 2^N)). [4] [8] [9]

- Mohó algoritmusok

A legegyértelműbb mohó stratégia a probléma megoldására az, ha a tárgyak az értékük csökkenő sorrendjében kerülnek kiválasztásra. Ennek a hátránya

az, hogy ha a magas értékű tárgyaknak nagy súlya lenne az értékükhöz képest, akkor nem találná meg az algoritmus az optimumot.

A másik egyértelmű mohó stratégia, ha a tárgyak a súlyuk szerinti növekvő sorrendben kerülnek kiválasztásra. Ez a stratégia abban az esetben megbukik, ha a könnyű tárgyak a súlyukhoz képest alacsony értékkel rendelkeznek.

Az előző két mohó algoritmus buktatóinak elkerülése érdekében egy kifinomultabb mohó stratégia az egységnyi súlyra vetített legnagyobb értékkel rendelkező tárgyak kiválasztását preferálja. Vagyis az egységnyi súlyra jutó érték szerint csökkenő sorrendbe rendezzük a tárgyakat, és sorban haladva választjuk ki azokat. Értelemszerűen egy elem akkor kerül a hátizsákba, ha a súlya nem növeli az összsúlyt W fölé. Még ez a stratégia sem ad minden esetben optimális megoldást.

Az algoritmus futásideje O(*N*), de a 0-1 hátizsák problémára nem létezik olyan mohó stratégia, amely minden esetben képes meghatározni az optimumot. [8]

2.4. Összegzés

A hasonló rendszerek elemzése során arra jutottam, hogy az általam készítendő rendszerhez a Mealime mobiltelefonos alkalmazásából tudnám a legtöbb ihletet meríteni. A három elemzett rendszer közül a Mealime rendelkezik olyan felhasználói felülettel, ami a jelen kor igényeinek a leginkább megfelel. A felület letisztult, nem fárasztó a szemnek, továbbá a megfelelő színválasztással kiemeli a fontosabb komponenseket a felhasználók számára. Ezeket a megismert stratégiákat tudom hasznosítani a saját alkalmazás készítése során. További pozitívuma a Mealime mobilalkalmazásnak, hogy rendelkezik menügenerálással. Habár ezen funkciója nem egyezik meg az általam készítendő menügenerálással, de számos aspektusát tekintve tudok ötleteket meríteni, mint például a személyre szabhatóság, statisztikák figyelembevétele, a menügenerálás eredményének indoklása stb.

A funkcionalitások gazdagságát, illetve a teljes alkalmazás személyre szabhatóságát tekintve a Food Planner kiemelkedik a többi hasonló rendszer közül, ahogy az a 2.1. táblázatban is látható. Az általam készítendő alkalmazással szemben elvárás a magas fokú személyre szabhatóság, ezért inspirációnak megfelelőek a Food Planner szerteágazó funkciói. A Food Planner alkalmazásban a felhasználóknak lehetőségük van saját receptek felvételére, ami a másik két hasonló rendszer esetében nem, vagy csak részben lehetséges. Lehetőség van tetszőleges alapanyagok felvételére is, ami szintén egy fontos elvárás az általam készítendő alkalmazással szemben. A recepteket és az alapanyagokat a felhasználók szinte korlátlanul módosíthatják, ami növeli a személyre szabhatóságot.

Az általam készítendő rendszer fő célja az optimalizált menügenerálás. Ez az optimalizálási feladat nagy mértékben hasonlít a 0-1 hátizsák problémára, hiszen a feladat *N* darab recept közül kiválasztani *M* darabot, úgy, hogy a kiválasztott receptek összesített jósága maximális legyen. A korábban bemutatott optimalizálási algoritmusok különböző futásidővel tudták megoldani a 0-1 hátizsák problémát, ezek összesítését a 2.2. táblázat tartalmazza.

	Nyers erő	Oszd meg és uralkodj	Feljegyzéses módszer	Visszalépéses keresés	Dinamikus programozás	Mohó algoritmusok
Futásidő	$O(2^N)$	$O(2^N)$	$O(N * W)$	$O(2^N)$	$O(N * W)$	$O(N)$

2.2. táblázat. Optimalizációs módszerek futásideje 0-1 hátizsák probléma esetén

A nyers erő módszere az egyetlen, amelynek futásideje minden esetben $O(2^N)$, a többi módszernél legrosszabb esetben egyezik meg a futásidő a 2.2. táblázatban megadott értékekkel. Tehát a nyers erő módszerének alkalmazása ebben az esetben nem indokolt. A mohó algoritmusok stratégia futásidő szempontjából nagyon kedvező opciónak tűnik, de nagyon kevés feladat esetében képesek meghatározni az optimumot. A 0-1 hátizsák problémára nincs olyan mohó algoritmus, amely minden esetben megtalálná az optimális megoldást, ezért valószínűleg az optimalizált menügenerálásra se lenne megfelelő. Az Oszd meg és uralkodj stratégia futásidő szempontjából általában kedvezőbb, mint a nyers erő módszer, de ha a feladat átfedő részproblémákat tartalmaz, akkor többször is kiszámolja ugyanazokat a részeredményeket ez pedig ront az alkalmazás teljesítményén. A feljegyzéses módszer, illetve a dinamikus programozáson alapuló algoritmus futásidő szempontjából kedvezőnek bizonyul, de mindkét stratégia esetén jelentősen megnőhet az algoritmus tárhelyigénye, amennyiben *N* értéke nagyon magas. Ha az általam készítendő rendszert egy időben több felhasználó is használni kívánja, a tárhely kihasználtság még tovább nőhet, ezzel rontva az összteljesítményt.

Ezen ismeretek tükrében az optimalizált menügenerálást a **visszalépéses keresés** segítségével fogom megvalósítani. Természetesen ennek a stratégiának is megvannak a maga hátrányai, mint például a számos rekurzív függvényhívás. Ezzel szemben az előnyei közé sorolható, hogy a keresési feltételek jól elkülöníthetők az alapvető algoritmustól, tárhelyigénye viszonylag alacsony. Továbbá a 0-1 hátizsák probléma esetén ez a módszer általában hatékonyabb, mint a dinamikus programozás. Amennyiben a visszalépéses keresés az elvárt teljesítménytől jelentősen elmarad, akkor tovább lehet növelni a hatékonyságot egy felső becslés bevezetésével. [8]

3. KÖVETELMÉNY SPECIFIKÁCIÓ

Az elkészítendő rendszer egy webalkalmazás, amelynek fő funkciója az optimalizált menügenerálás a háztartásban fellelhető alapanyagok felhasználásával. A rendszer jellegéből fakadóan szükséges egy szerver-, illetve kliensoldali alkalmazás elkészítése. Továbbá szükséges egy adatbázis megtervezése és elkészítése, ami a felhasználók adatait, receptjeit, alapanyagait stb. tárolja.

Az adatbázistól elvárás, hogy az adatokat konzisztensen tárolja több felhasználó esetén is. Ebből fakadóan kellő körültekintéssel kell megtervezni az adattáblák felépítését és az adattáblák közötti kapcsolatokat. Továbbá a szerveroldali alkalmazás számára biztosítani kell az adatbázis és a benne található adatok elérését.

A szerveroldali alkalmazásnak szükséges megvalósítania az alapvető CRUD műveleteket minden, a felhasználók számára hozzáférhető objektum esetében, mint például az alapanyagok, receptek stb. A felhasználókezelést is a backend oldali rendszernek kell ellátnia, illetve az optimalizált menügenerálásnak is itt kell történnie. Továbbá a szerveroldali alkalmazásnak elérhetőnek kell lennie a kliensoldali rendszer számára.

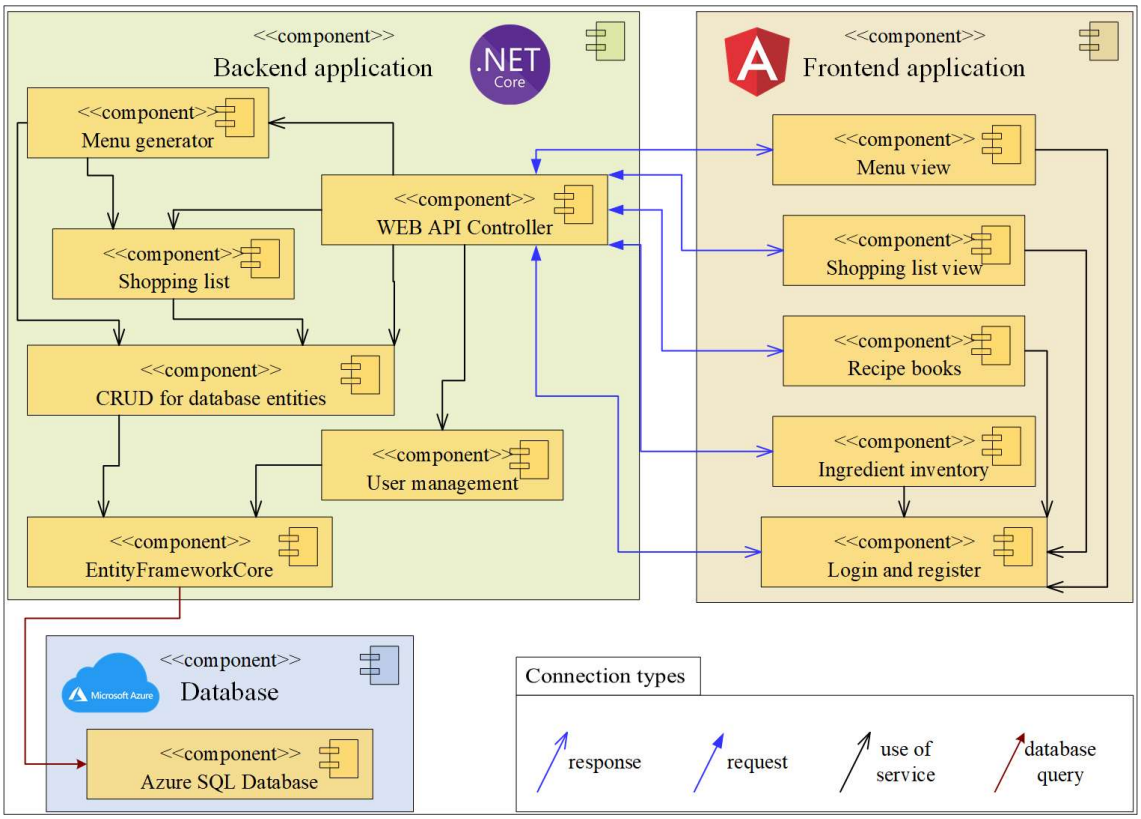
A kliensoldali alkalmazástól elvárás, hogy rendelkezzen a felhasználók számára könnyen értelmezhető, letisztult grafikus felhasználói felülettel. Biztosítani kell, hogy a felhasználók regisztrálhassanak, illetve beléphessenek az alkalmazásba. A megfelelő működéshez szükséges a kommunikáció biztosítása a szerveroldali alkalmazással.

A rendszertől további elvárás, hogy a felhasználók rögzíteni tudják saját receptjeiket és a háztartásukban levő alapanyagokat. Ezek az adatok elengedhetetlenek az optimalizált menü előállításához, mivel az optimalizálást végző algoritmus minden esetben figyelembe veszi a rendelkezésre álló alapanyagok lejárati dátumát. Az optimalizációnak változó paraméterek esetén is szükséges működnie, a személyre szabhatóság növelése érdekében, azaz az algoritmusnak a paramétereknek megfelelően kell meghatároznia az optimumot.

4. TERVEZÉS

4.1. Rendszerterv

Az elkészítendő webalkalmazás három fő komponensből épül fel. Ezen komponensek a kliens-, illetve a szerveroldali alkalmazás és a felhasználók adatait tároló adatbázis. A fő komponensek további egységekből épülnek fel, ezek szerveződését és a közöttük lévő kapcsolatokat a 4.1. ábra szemlélteti.



4.1. ábra. Rendszerterv

Az adatok tárolásáért egy, a Microsoft Azure felhőszolgáltatásban hostolt SQL adatbázis felel. Az Azure szolgáltatás választása mellett számos érv felsorolható, mint például a kényelem, egyszerű beállítás, nagyfokú személyre szabhatóság, hallgatói kedvezmény stb. A felhőszolgáltatásban lévő adatbázis könnyen elérhető a szerveroldali alkalmazás számára, amennyiben rendelkezik internetkapcsolattal, illetve megfelelő hitelesítéssel.

A szerveroldali alkalmazás ASP.NET Core keretrendszer segítségével kerül megvalósításra C# nyelven. A keretrendszerrel platformfüggetlen alkalmazást lehet készíteni, így a kész rendszer bármely platformon futtatható és hostolható, ami rendkívül rugalmas megoldást jelent. Az elkészítendő rendszer objektumokon alapul, ezáltal a C# nyelv objektum-orientált felépítése megkönnyíti a fejlesztést.

A 4.1. ábra szerinti *WEB API Controller* komponens biztosítja a HTTP alapú kommunikációt a kliens-, és a szerveroldali alkalmazás között JSON objektumok segítségével.

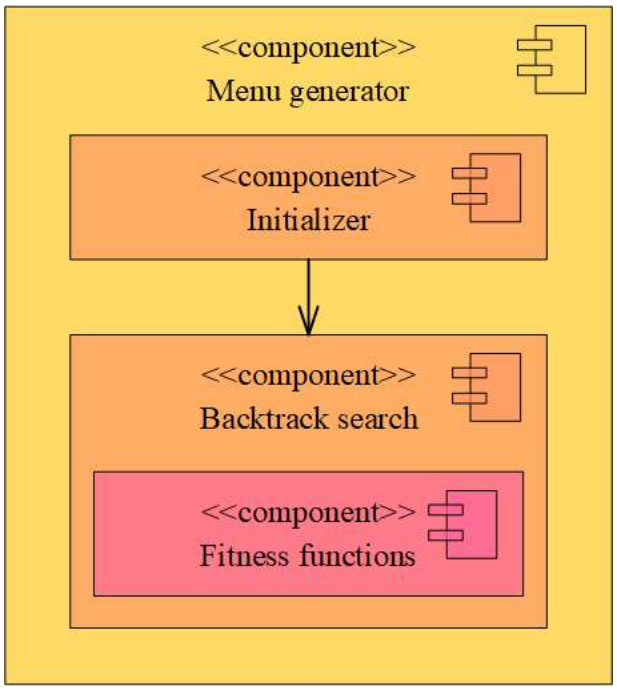
Az *EntityFrameworkCore* komponens felelőssége az adatbázis összekötése a szerveroldali alkalmazással. Az implementációja a Microsoft.EntityFrameworkCore csomag segítségével történik. Ezen csomag az Entity Framework adatelérési technológia könnyű, bővíthető, nyílt forráskódú és platformfüggetlen változata. Ezen változat egy ORM (*object-relational mapper*), amely lehetővé teszi, hogy egy adatbázissal .NET objektumokat használva lehessen dolgozni, illetve előre megírt adatelérési kóddal rendelkezik. [11]

A *CRUD for database entities* elem feladata új entitások létrehozása és elmentése az adatbázisba, meglévő entitások kiolvasása, módosítása, illetve törlése. Az előbbi műveleteket az *EntityFrameworkCore* komponens segítségével végzi el.

A *Shopping list* komponens felelősségi köre a bevásárlólista előállítása, amely kétféleképpen történhet. A felhasználó elkészíthet saját igényeinek megfelelő bevásárlólistát, illetve a menügenerálás során előállhat egy lista, amely tartalmazza azokat az alapanyagokat, amelyekkel a felhasználó az adatai alapján nem rendelkezik.

A *Menu generator* elem a 4.2. ábra alapján tartalmazza a következő komponenseket: *Initializer, BTS with fitness functions*. A *Menu generator* felelősségi köre az optimalizált menü előállítása. A komponens *Initializer* egysége végzi a beérkező paraméterek feldolgozását, illetve a *CRUD for database entities* komponenstől lekéri az adott felhasználóhoz tartozó, a kereséshez szükséges adatokat. Továbbá a *Menu generator* a menü előállítása után egy bevásárló listát is készít a *Shopping list* komponens segítségével.

A *User management* komponens felel a felhasználókezelésért, mint például regisztráció és belépés. A felhasználók hitelesítése JWT használatával történik. Ezen egység implementációja a Microsoft.AspNetCore.Identity.EntityFrameworkCore csomag segítségével történik. [12]



4.2. ábra. Menu generator szerveroldali komponense

A kliensoldali alkalmazás az Angular keretrendszer segítségével kerül megvalósításra, TypeScript nyelven. Az Angular egy frontend oldali JavaScript keretrendszer, amely alkalmas hatékony és kifinomult egyoldalas alkalmazások létrehozására, illetve . A TypeScript a Microsoft által fejlesztett erősen típusos nyelv, amely a JavaScriptre épül. A szerveroldali alkalmazás erősen épít a Microsoft által fejlesztett eszközökre, ezáltal a frontend alkalmazással való összeköttetés gördülékenyebb lehet, mivel az Angular és a TypeScript szintén támogatott a Microsoft által. A kliens öt nagyobb komponensből áll, amelyeket a 4.1. ábra szemléltet. Az alkalmazás a szerveroldallal HTTP alapú kommunikációval áll kapcsolatban, illetve JSON objektumokon keresztül kerülnek átadásra az adatok.

A kliensoldali alkalmazás egyik komponense a *Login and register*, amely feladata a felhasználókezelés frontend oldalon. A leendő felhasználóknak lehetősége van regisztrálni, illetve a már regisztrált felhasználók beléphetnek a rendszerbe. Továbbá ez a komponens biztosítja a backendtől kapott JWT elérését a kliens többi komponense számára.

Az *Ingredient inventory* elem az aktuálisan bejelentkezett felhasználó alapanyagait jeleníti meg. Ez a komponens ad lehetőséget a felhasználóknak új alapanyagok hozzáadására, meglévők módosítására, illetve törlésére.

A *Recipe books* komponens megjeleníti a felhasználók számára a saját receptkönyveiket, amelyek recepteket tartalmaznak. Itt van a felhasználóknak lehetősége új receptet vagy receptkönyvet létrehozni, illetve már meglévőket módosítani vagy törölni.

A *Shopping list view* elem biztosítja a bevásárlólista grafikus megjelenítését a felhasználó számára. A felhasználónak lehetősége van a bevásárlólistát módosítani, illetve

törölni. Menügenerálás után a menühöz szükséges, de hiányzó alapanyagokat a szerveroldali alkalmazás hozzáadja a listához, amely a kliensen megjelenik.

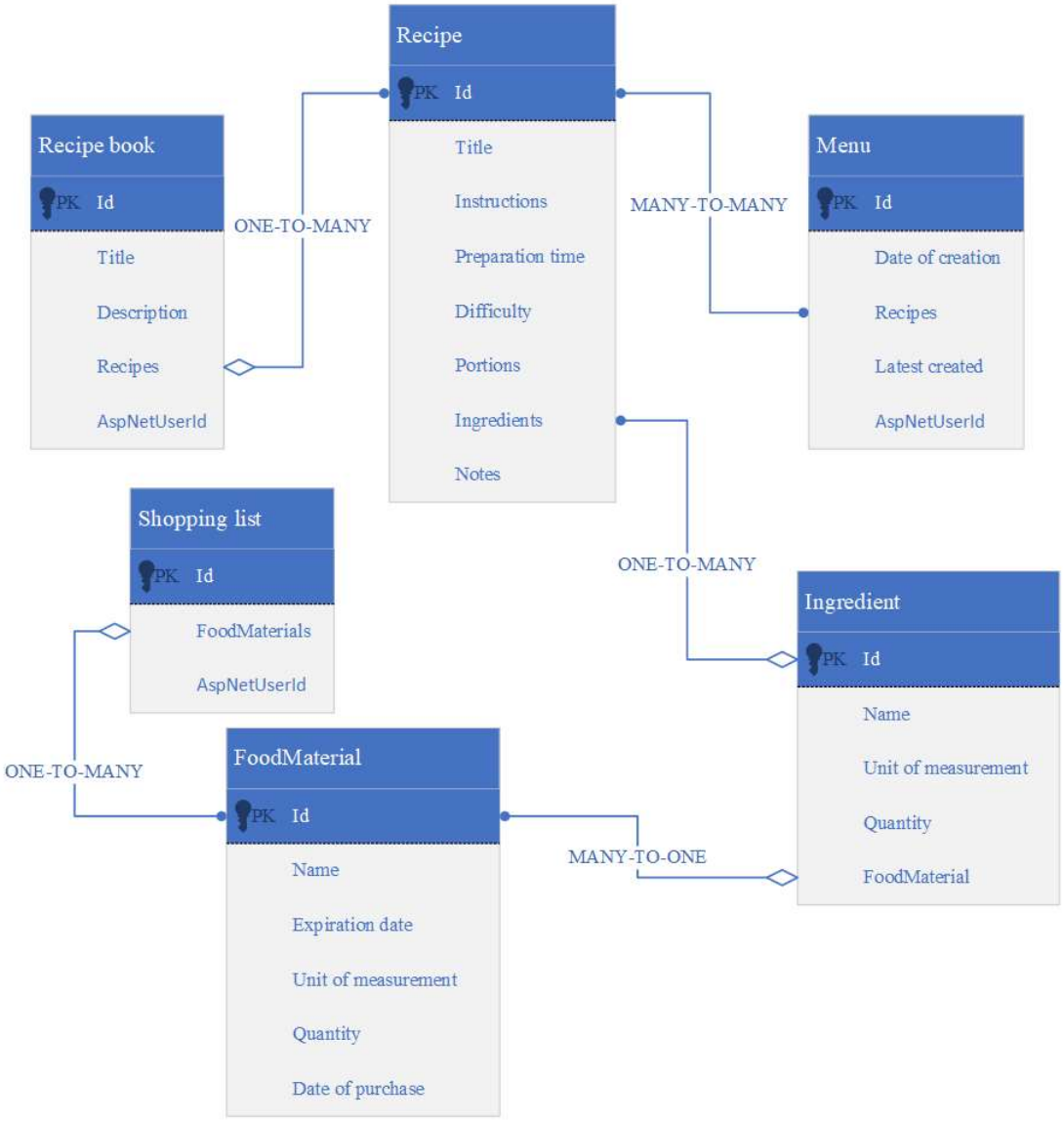
A *Menu view* komponens biztosítja a felhasználó számára a menügenerálást és az ahhoz tartozó paraméterek beállítását. Ha a menü előállt a felhasználó számára, azt itt tekintheti meg, illetve módosíthatja vagy törölheti is. Továbbá lehetősége van a felhasználónak manuálisan összeállítani egy menüt a saját receptjeiből válogatva.

4.2. Adatbázis felépítése

Az elkészítendő rendszer adatbázis komponensének terve a 4.3. ábrán látható, amely tartalmazza az alkalmazás által használt entitásokat és az azok közötti kapcsolatokat. A rendszerben megvalósításra kerül a felhasználókezelés is, de a 4.3. ábrán ez nem jelenik meg, mivel ez a funkció a Microsoft.AspNetCore.Identity.EntityFrameworkCore csomag segítségével történik. A csomag előre definiált adattáblákat, entitásokat és közöttük lévő kapcsolatokat definiál, ezáltal ezek bemutatása az alkalmazás tervezésének szempontjából nem releváns. Továbbá a 4.3. ábrán látható entitások csak az AspNetUsers táblával állnak közvetlen kapcsolatban a csomag által meghatározott entitások közül.

Az adatbázis hat entitásból és azok kapcsolatából áll. A *FoodMaterial* entitás az alapanyagokat hivatott reprezentálni, amelyeket az *Ingredient inventory* kliensoldali komponensén keresztül érhetnek el a felhasználók. Az *Ingredient* entitás a receptek hozzávalóit képviseli, amely tartalmazza, hogy mely alapanyaghoz köthető. Fontos megjegyezni, hogy a *FoodMaterial* és az *Ingredient* entitások között nagy a hasonlóság, de funkciójukat tekintve eltérnek egymástól. A *FoodMaterial* a kézzel fogható alapanyagokat reprezentálja, mint például 1 kilogramm krumpli, 3 fej hagyma stb., ezzel szemben az *Ingredient* objektumok absztrakt összetevői egy receptnek. A *Recipe* entitás egy receptet testesít meg, amely tartalmazza a hozzávalókat, azaz *Ingredient* objektumokat. Egy recepthez több *Ingredient* entitás is tartozhat, de egy hozzávaló csak egy recepthez tartozik. A *Recipe book* entitás egy receptkönyvet testesít meg, amelyet a felhasználók a *Recipe books* kliensoldali komponensen keresztül érhetnek el. Egy felhasználónak akár több receptkönyve is lehet. A *Shopping list* entitás a bevásárlólistát reprezentálja, amelyből a felhasználók számára csak egy elérhető. A bevásárlólista több *FoodMaterial* entitást is tartalmazhat. A *Menu* entitás a generált vagy a felhasználó által összeállított menü objektumokat képviseli. Egy menü objektumhoz tartozhat több recept is, illetve egy recept szerepelhet több menüben is. Emiatt a *Menu* és *Recipe* adattáblák között több a többhöz

kapcsolat áll fent, amely egy kapcsolótábla közbeiktatásával valósul meg. A rendszerhez használt *EntityFrameworkCore* komponens a kapcsolótáblát automatikusan legenerálja.

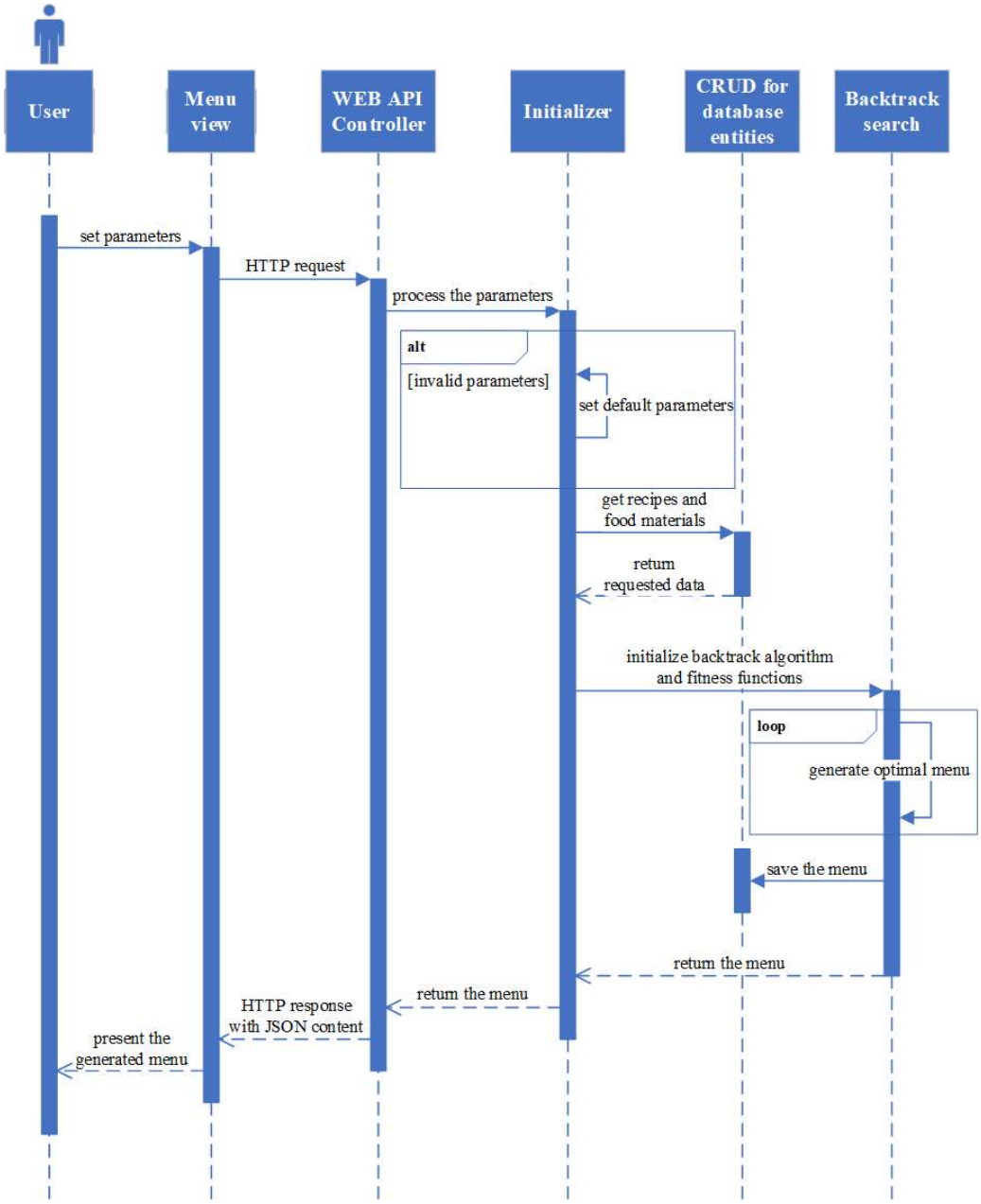


4.3. ábra. A rendszer adatbázisa EK diagrammal ábrázolva

4.3. Menügenerálás folyamata

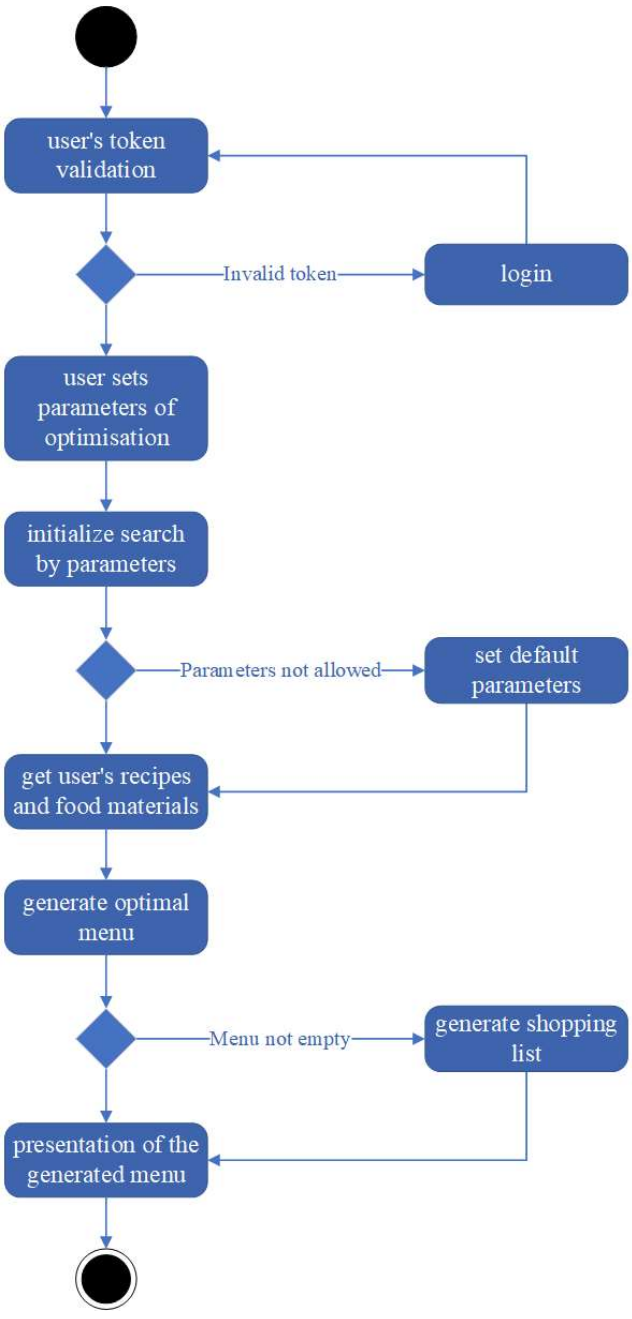
Az optimalizált menügenerálás folyamatát a 4.4. ábra szemlélteti egy szekvencia diagramon keresztül. Az optimalizáció első lépése, hogy a felhasználó beállítja a számára megfelelő paramétereket, mint például a receptek ismétlődése, legtöbb alapanyag felhasználása stb. A kliensoldali *Menu view* komponens egy HTTP kéréssel fordul a szerveroldali alkalmazáshoz. A kérés törzse tartalmazza a felhasználó által megadott paramétereket JSON objektumként. A backend *WEB API Controller* komponense fogadja a kérést,

és elindítja a feldolgozást. A *Menu generator* komponens *Initialize* egysége ellenőrzi az inputot. Amennyiben a bemenő adatok nem megfelelőek, akkor az alapértelmezett beállításokat használja. Ezt követően a *CRUD for database entities* komponenstől lekéri a felhasználó receptjeit, illetve alapanyagait. Amint az *Initializer* egység rendelkezésére állnak az adatok a komponens elindítja a visszalépéses keresést a beállított és lekért adatokkal. A menügenerálás végeztével a *Backtrack search* komponens a menüt átadja a *CRUD for database entities* egységnek, ami elmenti az adatbázisba, illetve visszaadja a menüt az *Initializer* komponensnek. A vezérlés visszakerül a *WEB API Controller* egységhez, ami HTTP válasz formájában eljuttatja a *Menu view* komponensnek a JSON-né konvertált menüt. Végül a *Menu view* egység a generált menüt megjeleníti.



4.4. ábra. A menügenerálás szekvencia diagramja

A menügenerálás folyamatát a 4.5. ábra is leírja, amely alapjaiban véve ugyanazt a folyamatot határozza meg, mint a 4.4. ábra, azzal a kivétellel, hogy a menügenerálás kapcsán más folyamatokat is szemléltet. A menügenerálás megkezdése előtt a kliens oldalon ellenőrizni kell, hogy az adott felhasználó rendelkezik-e megfelelő hitelesítéssel. Ha nem, akkor átirányításra kerül a bejelentkezés felületre, mivel csak hitelesített felhasználók vehetik igénybe a szolgáltatást. Továbbá a menü elkészítését követően a szerveroldali alkalmazás ellenőrzi, hogy az előállított menü tartalmaz-e recepteket. Amennyiben igen, akkor az ahhoz szükséges bevásárlólistát is előállítja, és hozzáfűzi a már meglévő *FoodMaterial* entitásokhoz.



4.5. ábra. A menügenerálás folyamatábrája

4.4. Funkciólista

- Optimalizált menügenerálás változtatható paraméterekkel.

  A menüt előállító algoritmus minden esetben figyelembe veszi a meglévő alapanyagok lejárati dátumát és azokat használja fel, amelyek a leghamarabb romlandók. További paraméterként megadható, hogy a menü tartalmazhat-e egy receptet többször, illetve azokat az alapanyagokat használja, amelyekből a legtöbb van stb.
- CRUD műveletek alapanyagok, receptek, hozzávalók, receptkönyvek, menük esetén.

  Az alapvető CRUD műveletek (*Create, Read, Update, Delete*) megvalósítása a különböző objektumok esetében.
- Bevásárlólista készítése.

  Amennyiben a generált menühöz hiányzik alapanyag, az automatikusan a bevásárlólistára kerül, illetve a felhasználó manuálisan is szerkesztheti a listát.
- Varázsló szerű adatbevitel bevásárlás után.

  A szükséges adatok manuális bevitelének könnyítése a funkció célja. A felhasználónak nem szükséges az *Ingredient inventory* komponensen megadnia az adatokat, vásárlás után a *Shopping list* komponens biztosítja ezt a lehetőséget.
- Összesítő statisztikák készítése.

  A szeméttermelés nyomon követhetősége a felhasználó által, illetve a „fejlődés” vizualizálása.
- Jutalom rendszer alkalmazása.

  Mínusz, illetve plusz pontok gyűjtése, ezzel motiválva a felhasználót a tudatos étkezésre, illetve a kevesebb étkezési hulladék termelésére.

4.5. Fejlesztés tervezett menete

A készítendő rendszer fejlesztésére a 4.6. ábra szerinti terv alapján 12 hét áll rendelkezésre. Az első héten a három fő komponens, a szerver-, illetve kliensoldali alkalmazás és az adatbázis inicializálása a cél. Ehhez a fázishoz tartozik a verziókövető rendszer beállítása, és a *remote repository* szinkronizálása, annak érdekében, hogy a projekt előrehaladása követhető legyen. A második héttől kezdve a szerveroldali CRUD műveletek megvalósítása, illetve a hozzájuk tartozó kliensoldali megjelenítés elkészítése a cél. A negyedik hét folyamán szükséges elkészíteni az alkalmazás gerincét adó optimalizáló függvényt és az ahhoz tartozó komponenseket, mint például a jóság függvény(ek). Ebben a fejlesztési fázisban el kell készülnie a menügeneráláshoz szükséges kliensoldali megjelenítésnek is. Ezt követően a bevásárlólistát megvalósító backend oldali logika és a kliensoldali megjelenítését szükséges implementálni. Amennyiben ezek a fejlesztési fázisok

elkészültek, a rendszert tesztelni szükséges. Végül a fejlesztés és tesztelés befejezésével el kell készíteni a rendszer dokumentációját.

Feladat		Backend és frontend projekt inicializálás	Modellek és az adatbázis létrehozása	CRUD megvalósítása	BTS alapú optimalizálás megvalósítása	Bevásárlólista komponens elkészítése	Tesztelés	Dokumentálás
Kezdés		2022.02.07	2022.02.09	2022.02.14	2022.02.28	2022.03.28	2022.04.11	2022.05.02
Befejezés		2022.02.08	2022.02.11	2022.02.27	2022.03.27	2022.04.10	2022.04.24	2022.05.08
Időtartam		2 nap	3 nap	14 nap	28 nap	14 nap	14 nap	7 nap
2022. Február	1. hét							
	2. hét							
	3. hét							
2022. Március	4. hét							
	5. hét							
	6. hét							
	7. hét							
	8. hét							
2022. Április	9. hét							
	10. hét							
	11. hét							
	12. hét							
2022. Május	13. hét							

Tervezett időtartam
Tervezett csúszás

4.6. ábra. Fejlesztés tervezett menete

5. FEJLESZTÉS

A fejlesztés során számos probléma felmerült, melyeket ebben a fejezetben fogok bemutatni. A rendszer megvalósítását a CRUD műveletek elkészítésével kezdtem, a szerver-, és kliensoldalon egyaránt.

5.1. Entitások elhelyezése a szerveroldali alkalmazásban

A CRUD műveletek implementálása során az első felmerült probléma az volt, hogy hol helyezzem el az adatbázisban megtalálható entitásokat leíró osztályokat. Több lehetőség is felmerült:

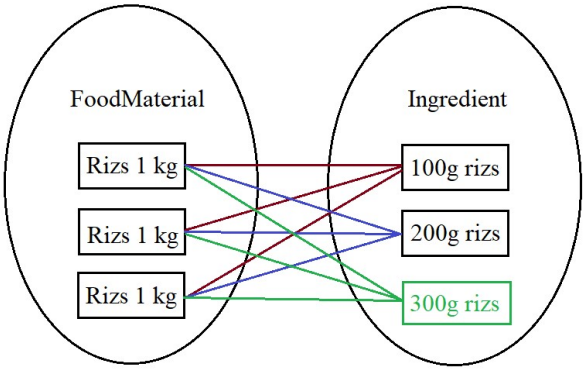
- minden projektben újra és újra megadni a leíró osztályokat,
- az egyik projektben meghatározni az entitásokat, majd minden egyéb projekt erre hivatkozik,
- teljesen szeparált projektben megadni az entitásokat leíró osztályokat.

Az felsorolt megoldási lehetőségek közül az első biztosítaná a projektek közötti függetlenséget, de nehezen kezelhetővé tenné a kódot. A második lehetőség nem megengedett függősékeket eredményezne, így a harmadik opciót választottam megoldásnak. Ennek előnye, hogy a modelleket leíró projekt csak abban az esetben töltődik be a memóriába, amennyiben arra tényleg szükség van, továbbá a leíró osztályokat nem kell duplikálni tehát a kód is könnyebben kezelhetővé válik.

5.2. FoodMaterial és Ingredient entitások közötti kapcsolat

Az adatbázis felépítésének tervezése során úgy gondoltam, hogy a FoodMaterial és az Ingredient entitások között közvetlen kapcsolat építhető fel. Az implementálás során azonban kiderült, hogy a közvetlen kapcsolat rengeteg adatbázis műveletet eredményezne olyankor is, amikor arra nem feltétlenül lenne szükség. Az adatbázis műveletek megsokszorozódása mellett további problémát jelentene a bonyolult logika megírása, amivel a hibák száma is jelentősen megnőhet.

A problémát az alábbi példával tudnám szemléltetni:



5.1. ábra. FoodMaterial és Ingredient entitások közötti közvetlen kapcsolat

Az 5.1. ábra alapján, az adatbázis FoodMaterial táblájában szerepel 3 darab 1 kg-os rizs entitás és az Ingredient táblában 2 db rizs entitás különböző tömegekkel. A közvetlen kapcsolat miatt mind a két Ingredient táblában található rizs az összes FoodMaterial táblában lévő rizs entitással kapcsolatban áll. Amennyiben egy új rizs entitás jön létre az Ingredient táblában, akkor hozzá kell kötni ezt a FoodMaterial táblában található összes rizs entitáshoz a közvetlen kapcsolat fenntartásához, ezeket az ábrán zölddel jelöltem.

Ezen probléma feloldására egy új adattábla bevezetését tartottam a legmegfelelőbbnek. Az új adattábla a FoodType, ami tartalmazza az alapanyagok és hozzávalók lehetséges típusát, hiszen ez az a tulajdonság, ami alapján kapcsolat van az alapanyagok és hozzávalók között. Az új adattábla bevezetésének köszönhetően az entitások közötti kapcsolatok rendezettebek és logikailag is jobban elkülönülnek.

5.3. A menügenerálás megvalósítása

A menügenerálást végző komponens két alkomponensből tevődik össze, az inicializáló egységből és a visszalépéses keresésen alapuló optimalizáló algoritmusból. Az implementáció során az inicializáló egység elkészítése volt az első lépés. Ehhez meg kellett határozni a generálás paramétereit, mivel az inicializáló egység feladata ezen paraméterek validálása. Az előkészítési feladatok körébe tartozott, hogy nem megfelelő paraméterek megadása esetén a generálás is zökkenőmentesen tudjon működni, illetve a felhasználó számára biztosítson visszajelzést a bemenő adatok hibáiról.

A visszalépéses keresés megvalósítása lépcsőzetesen történt. Elsőként a 0-1 hátizsák problémát megoldó algoritmust alkalmaztam egyetlen menü előállításához. Ezen fázisban a jóság függvény azt határozta meg, hogy az aktuálisan kiválasztott receptek hozzávalóinak a lejárati dátuma milyen mértékben tér el az aznapi dátumtól. Ehhez a következő függvényt alkalmaztam, amelynek a görbéje kék színnel szerepel az 5.2. ábrán:

$$\textbf{Jóság}_{\textbf{LejáratiDátum}}(\textbf{x}) = \frac{\textbf{1400}}{\textbf{x+1}},$$

ahol *x* a vizsgált alapanyag lejárati dátumának és az aktuális dátumnak a különbsége napban megadva. A függvényt számos körülmény figyelembevételével határoztam meg. A kiindulási függvény az $\frac{1}{x}$ volt, mivel a jóság és az argumentum között fordított arányosságnak kell fennállnia, azaz minél korábban jár le az alapanyag, annál nagyobb jósággal kell bírnia. Továbbá a számlálóban 14 * 100 szerepel, mivel a cél az volt, hogy a két héten belül romlandó alapanyagok jósága kiemelkedően magas legyen, hogy azok minél nagyobb valószínűséggel felhasználásra kerüljenek. Az *x*-hez azért volt szükséges hozzáadni egyet a nevezőben, hogy az x=0 helyen is értelmezett legyen a függvény, mivel az argumentum értéke nulla is lehet, ha a generálás napján jár le a szavatossága egy alapanyagnak.

Második lépésként az egyes receptek többszöri kiválasztásának lehetőségét adtam hozzá a meglévő generáláshoz. Ennél a fázisnál egy segéd entitásra volt szükség, amely azt is nyilvántartja, hogy egy adott recept hányszor került kiválasztásra. Továbbá a lejárati dátumokat vizsgáló jóság függvényt szükséges volt kiegészíteni az ismétlődések kezelésével abban az esetben, ha a felhasználó engedélyezi a receptek ismétlődését a generálás során.

Harmadik lépésként két további jóság függvényt kellett meghatározni annak érdekében, hogy a menügenerálás figyelembe vehesse minél több alapanyag felhasználását, illetve a receptek elkészítési idejét. A függvények meghatározása során fontosnak tartottam, hogy azok értéke ne befolyásolja túl nagy mértékben az összesített jóságot abban az esetben, ha a recept hozzávalói között szerepel olyan, amelynek a lejárati ideje 14 napon belüli.

Az elkészítési időhöz tartozó jóság függvény görbéje az 5.2. ábrán piros színnel szerepel, aminek a meghatározása a következő:

$$\textbf{Jóság}_{\textbf{ElkészítésiIdő}}(\textbf{x}) = \textbf{-0,56x+101},$$

ahol *x* a vizsgált recept elkészítési ideje percben megadva. A függvény meghatározása az alapján történt, hogy a 180 percnél több időt igénylő receptekhez negatív jóság szerepel, illetve az x=14 helyen megegyezzen a jóság értéke a lejárati dátumhoz tartozó jóság függvény értékével.

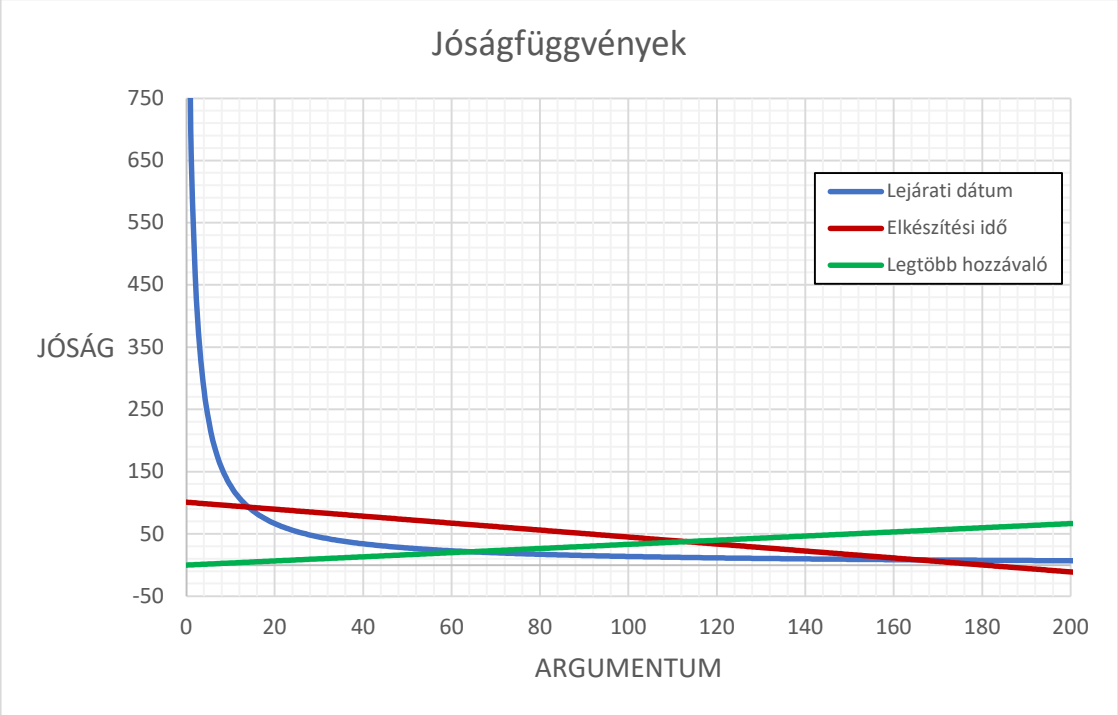
A legtöbb hozzávaló felhasználásához tartozó jóság függvény görbéje az 5.2. ábrán zöld színnel szerepel, aminek meghatározása a következő:

$$\textbf{Jóság}_{\textbf{LegtöbbHozzávaló}}(\textbf{x}) = \frac{\textbf{x}}{\textbf{3}},$$

ahol *x* a vizsgált hozzávaló mennyisége deciliterben vagy dekagrammban megadva, amennyiben az lehetséges. Ez a függvény a végtelen felé tart, ezért a meghatározása során

a cél az volt, hogy csak akkor legyen magas az értéke, ha rengeteg alapanyag kerül felhasználásra.

Ehhez a számításhoz szükséges volt az aktuálisan vizsgált hozzávalók mennyiségének átváltására a mértékegységük függvényében. Emiatt elkészítettem egy olyan statikus konvertert, ami a tömegeket dekagrammra, a térfogatokat pedig deciliterre váltja. Mivel a receptek megadása során előfordulhatnak ettől eltérő mértékegységek is, mint például evőkanál, bögre, darab stb., ezért szükség volt olyan általános átalakítóra is, amely képes ezeket is elfogadható értékekké alakítani. A megvalósítása ennek úgy történt, hogy az egyedi mértékegységű hozzávalókat először dekagrammra próbálja átváltani. Ha ez sikertelen akkor megpróbálja deciliterben megadott mennyiségre átalakítani. Abban az esetben, ha az átváltás mindkét esetben sikertelen, akkor a mértékegységtől függetlenül a mennyiséggel számol a jóság függvény.



5.2. ábra. Menügeneráló algoritmus jóságfüggvényei

Negyedik lépésként hozzáadásra került a menügenerálás paramétereihez az az opció, hogy a generált menü tartalmazzon alternatívákat is. A generálást ezért úgy kellett átalakítani, hogy az több lehetséges menüt is tartalmazzon. A visszalépéses keresés milyenségéből adódóan ezt úgy oldottam meg, hogy minden érvényes megoldás esetén a már korábban előállt menük közül a legalacsonyabb értékűvel hasonlítottam össze az aktuálisan vizsgált megoldás értékét. Amennyiben az új megoldás jósága nagyobb, akkor az kerül az addigi legkisebb értékű menü helyére. Az előállt menük a jóságuk szerinti csökkenő sorrendben kerülnek visszaadásra a hívó félhez.

Az algoritmus elkészítése során többször is felmerültek olyan problémák, amelyek miatt módosítani kellett entitásokon, logikán vagy kontrolleren. Ezeket a módosításokat könnyedén el tudtam különíteni a menügeneráló komponens implementációjától a *git* verziókövető rendszer *git stash* parancsának használatával.

5.4. Receptek kezelése

A menügeneráláshoz szorosan kapcsolódik a menüben található receptek kezelése. A menü generálását követően, ha a felhasználó kiválasztotta a számára megfelelő menüt, az alkalmazásnak lehetőséget kell biztosítania a bevásárlólista frissítésére és a menüben található receptek elkészítettként való megjelölésére.

A generált menü mentése után a szerveroldali alkalmazásnak szükséges megvizsgálnia a menüben található recepteket és azok hozzávalóit. A cél az, hogy a recept elkészítéséhez szükséges, hiányzó hozzávalók felkerüljenek a bevásárlólistára. Ehhez ellenőrizni kell, hogy a raktárban mely élelmiszerek állnak rendelkezésre. A fő problémát a különböző mértékegységek jelentik, mivel egy receptben szerepelhet olyan hozzávaló, aminek mértékegysége eltér a raktárban található megegyező típusú élelmiszer mértékegységétől. Továbbá megoldandó problémát jelent az, hogy a raktárban egy adott típusú élelmiszerből több is előfordulhat, illetve azok lejárati dátuma is különbözhet egymástól.

A mértékegységek eltérését részben sikerült kiküszöbölni egy statikus konverter használatával, ami képes dekagrammra váltani a grammban vagy kilogrammban megadott élelmiszer mennyiségét, illetve deciliterre váltani a centiliterben, milliliterben vagy literben megadott mennyiséget. Amennyiben a vizsgált hozzávaló mennyisége átváltható dekagrammra vagy deciliterre, abban az esetben a még nem lejárt szavatosságú azonos típusú élelmiszereket kell tovább vizsgálni. Ha ezek átválthatóak a megfelelő mértékegységű mennyiségre és az nem haladja meg a szükséges hozzávaló mennyiségét, akkor a bevásárlólistára kerül az adott típusú élelmiszer a hiányzó mennyiséggel és a számítás közben használt mértékegységgel. Azok a hozzávalók, amelyek mértékegysége nem váltható át dekagrammra vagy deciliterre, mint például evőkanál, csokor, bögre stb., azok egy külön listára kerülnek és nem befolyásolják a bevásárlólistát. Ez alól kivételt képeznek a darab, a csomag és a fej mértékegységek, mivel ezeknél nem szükséges a konvertálás, ha a raktárban is ugyanezzel a mértékegységgel szerepelnek az azonos típusú élelmiszerek.

A receptek késznek jelölése esetében is a fő problémát az eltérő mértékegységek jelentik. Ebben az esetben az elvárás az, hogy a hozzávalók mennyisége levonásra kerüljön a raktárban fellelhető megfelelő típusú élelmiszerek mennyiségéből. Ennek a megvalósítása hasonlóan történt, mint a bevásárlólista frissítése, azzal a különbséggel, hogy a folyamat végén a megfelelő típusú élelmiszer mennyiségéből kerül levonásra a hozzávaló mennyisége. Amennyiben a receptnek bármely hozzávalója hiányzik, vagy nem áll rendelkezésre elegendő mennyiség a raktárban, akkor a recept késznek jelölése és az élelmiszerek mennyiségének csökkentése nem történik meg.

5.5. Statisztika készítése

A rendszertől elvárás, hogy a felhasználó számára készüljenek statisztikák, amelyek segítségével nyomon követheti a szeméttermelését. A statisztikai adatokat szükséges vizuálisan megjeleníteni annak érdekében, hogy a felhasználók könnyedén értelmezhessék azokat. A nyomon követhetőség miatt a statisztikákat szükséges elmenti az adatbázisba, hogy azok a későbbiekben is elérhetőek legyenek. Az elvárások kielégítése érdekében a statisztikák havi szinten készülnek, és tartalmazzák az adott hónapban felhasznált, lejárt és elérhető élelmiszerek mennyiségét.

A statisztika feltöltése adatokkal kétféle eljárás szerint történik. Az egyik eljárás során a felhasznált élelmiszerek mennyisége kerül meghatározásra, amikor a felhasználó késznek jelöl egy olyan receptet, ami a rendelkezésére álló élelmiszerek alapján elkészíthető. Az elkészített recept hozzávalóinak mennyiségét megpróbálja dekagrammra vagy deciliterre átváltani és az annak megfelelő mennyiséggel növelni a felhasznált élelmiszerek értékét. Ha az átalakítás sikertelen, mert a mértékegység nem átváltható akkor a hozzávaló eredeti mennyisége kerül hozzáadásra. Emellett az elérhető élelmiszerek mennyisége a felhasznált mennyiség mértékével csökken a statisztikában.

A másik eljárás a lejárt élelmiszerek mennyiségét tartja számon, azonos mennyiségi konverzió mentén, mint a felhasznált élelmiszerek esetén. A metódus megvizsgálja a felhasználó raktárában található élelmiszereket és az aktuális dátumhoz képest régebbi lejárati dátummal rendelkező élelmiszerek mennyiségével növeli a lejárt élelmiszerek számát a statisztikában. Ezen eljárás nem ütemezetten végzi el az ellenőrzést, azért, hogy az esetlegesen inaktív felhasználók esetében a szerver oldali alkalmazás ne legyen feleslegesen túlterhelve. Az ellenőrzés indítását a kliens oldali alkalmazás kezdeményezi, amikor a felhasználó a saját statisztikáját kívánja megtekinteni. Ebből a működésből fakadóan szükséges eltárolni a statisztikában, hogy az mikor volt legutoljára frissítve, különben a lejárt élelmiszerek mennyisége többször is hozzá lenne adva a statisztikához. Így a meglévő élelmiszerek lejárati dátumának ellenőrzése során figyelembe kell venni, hogy a már lejárt szavatosságú élelmiszerek lejárati dátuma későbbi legyen a legutóbbi vizsgálati dátumnál. Továbbá az implementációból adódóan szükséges kezelni egy speciális esetet, amikoris az első statisztika jön létre. Ebben az esetben a legutolsó frissítési dátumot figyelmen kívül kell hagyni, hogy a korábbi lejárt alapanyagok megfelelő mennyisége is bekerüljön a statisztikába.

Mindkét eljárás az aktuális dátumnak megfelelő statisztikát tölti fel az adatokkal. Ha a dátumnak megfelelő statisztika nem létezik, akkor azt automatikusan létrehozza a két eljárás közül az, amelyik előbb kerül meghívásra. A statisztika létrehozásakor a legutolsó frissítési dátum az aktuális hónap első napjának éjfélét kapja értékül.

A kliens oldali alkalmazásban a statisztika megjelenítéséhez kördiagrammot alkalmaztam, amelyhez az *ngx-charts* külső kódkönyvtárat használtam. [13]

5.6. Visszajelzés a felhasználónak

A kliensoldali alkalmazás használatának könnyítése miatt elengedhetetlen érthető visszajelzést kapnia a felhasználónak abban az esetben, ha valami hiba történik a szerveroldali végrehajtás közben. Ennek érdekében a szerveroldali alkalmazásban hiba esetén érthető hibaüzenettel rendelkező kivételek váltódnak ki. Ahhoz, hogy ezek a hibaüzenetek eljussanak a felhasználóhoz, szükséges volt egy az „ExceptionFilterAttribute" osztályból származó saját osztály elkészítése, ami felülírja az „OnException(ExceptionContext context)” metódust. Ezen a felülírt metóduson keresztül a szerveroldali alkalmazásban bekövetkezett hibák által kiváltott alapértelmezett HTTP válasz tetszőlegesen felülírható, így a hibaüzenet szövegét el lehet juttatni a kliensoldali alkalmazáshoz.

A kliensoldali alkalmazás a kapott hibaüzenetet az *ngx-toastr* külső kódkönyvtár segítségével jeleníti meg. [14]

5.7. Shopping list komponens elkészítése

A tervezési fázisban meghatározott rendszerterv szerint a *Shopping list* komponensnek egy különálló egységben lenne a helye a szerveroldali alkalmazásban. A megvalósított komponens 80%-a CRUD műveletekből áll, ezért nem tartottam indokoltnak egy külön egység létrehozását, mivel a *CRUD for database entities* feladatkörével egybeesik az elkészített logika legnagyobb részének a működési elve.

6. TESZTELÉS

Az elkészült alkalmazás tesztelése unit tesztekkel és manuális teszteléssel történt. Legnagyobb mértékben a unit tesztek lettek kidolgozva az üzleti logika minél nagyobb lefedettségének érdekében a szerveroldali alkalmazásban. A kliensoldali alkalmazás működésének ellenőrzése manuális tesztekkel történt.

6.1. Unit tesztek

Az elkészült szerveroldali alkalmazás üzleti logikája két komponensből áll: a *Menu generator* és a *CRUD for database entities*-ből. A *CRUD for database entities* komponens lett először tesztelve, mivel erre a komponensre épít a *Menu generator*. Ezután következett a *Menu generator* komponens, azon belül az *Initializer* és a *Backtrack search* egységek is külön-külön kerültek tesztelésre. Az üzleti logika tesztelése során 164 különböző teszteset készült, ezért a továbbiakban csak a legfontosabbak kerülnek részletes leírásra. Továbbá a különböző objektumokhoz tartozó CRUD művelet tesztjeinek leírása is összevonásra kerül a könnyebb áttekinthetőség érdekében. A *CRUD for database entities* fontosabb teszteseteit a 6.1. táblázat tartalmazza. A komponens unit teszt lefedettsége több, mint 88%, ennek megoszlása osztályonként látható a 6.1. ábrán.

A *Menu generator* komponens tesztelése során az *Initializer* és a *Backtrack search* is egy-egy metódus tesztelését jelentette, mivel csak ezek publikusak, a segédmetódusaik privát láthatóságúak. A privát metódusok tesztelése a különböző tesztesetek meghatározásával volt elérhető, de a számos teszteset miatt ezek összefoglalva szerepelnek a 6.2. táblázatban. A komponens unit teszt lefedettsége 100%-os, aminek részletei a 6.2. ábrán láthatók.

#	Követelmény	Tesztleírás	Mérés célja	Eredmény
1	Lehessen regisztrálni még nem létező felhasználónévvel és e-mail címmel és megfelelően erős jelszóval.	Új felhasználó létrehozása az autentikációs logika segítségével.	Sikeres regisztrációt követően a felhasználók száma egyel nő.	✓
2	Lehessen bejelentkezni a rendszerbe megfelelő felhasználónév/e-mail és jelszó párossal.	Létező felhasználónév/e-mail cím és jelszóval belépés az autentikációs logika által.	Sikeres bejelentkezést követően a logika a megfelelő felhasználónevet és token-t adja vissza.	✓
3	Hibás adatokkal történő bejelentkezési kísérlet megfelelő hibaüzenetet adjon.	Hibás felhasználónév/e-mail cím és jelszó kombinációval megpróbálni belépni.	Nem létező felhasználó vagy hibás jelszó feldolgozása során az autentikációs logika kivételt dob megfelelő hibaüzenettel.	✓

#	Követelmény	Tesztleírás	Mérés célja	Eredmény
4	Lehessen létrehozni alapanyagot, receptet, receptkönyvet, és menüt megfelelő tulajdonságokkal.	A megadott típusú objektum létrehozása és mentése a típusnak megfelelő logika által.	Sikeres létrehozást követően a megfelelő típusú objektumok száma egyel nő.	✓
5	Alapanyagot, receptet, receptkönyvet, és menüt csak a tulajdonosa tudjon törölni.	A kiválasztott objektum törlése, ha a törlést az objektum tulajdonosa végzi.	Sikeres törlést követően a tulajdonos megfelelő típusú objektumainak száma egyel csökken.	✓
6	A felhasználó le tudja kérni az összes olyan alapanyagot, receptet, receptkönyvet vagy menüt, aminek ő a tulajdonosa.	A megfelelő típusú összes objektum lekérése, létező felhasználó adataival.	Sikeres lekérést követően a visszaadott objektumok száma megegyezik a felhasználó összes azonos típusú objektumával.	✓
7	Lehessen lekérni egy alapanyagot, receptet, receptkönyvet és menüt azonosító alapján.	A tulajdonos megadott típusú objektumának lekérése.	Sikeres lekérést követően a felhasználó megfelelő tulajdonságokkal rendelkező objektuma kerül visszaadásra.	✓
8	Adott felhasználó ne tudjon lekérni olyan alapanyagot, receptet, receptkönyvet vagy menüt, aminek nem ő a tulajdonosa.	Létező objektum lekérése olyan felhasználóval, aki nem tulajdonosa az objektumnak.	Jogosulatlan hozzáférési kísérlet miatt a megfelelő logika kivételt dob értelmezhető hibaüzenettel.	✓
9	Lehessen módosítani egy tetszőleges alapanyagot, receptet vagy receptkönyvet.	A tulajdonos megadott típusú objektumának frissítése.	Sikeres módosítást követően a módosított objektum új tulajdonságai mentésre kerülnek és a felhasználó objektumainak száma nem változik.	✓
10	Lehessen módosítani egy tetszőleges recept hozzávalóinak listáját.	Létező felhasználó egy receptjében a hozzávalók listájának módosítása.	Egy recept hozzávalóinak listája legyen módosítható.	✓
11	Lehessen egy menühöz hozzáadni olyan tetszőleges receptet, ami még nem szerepel a menüben.	Létező felhasználó egy receptjének hozzáadása a felhasználó menüjéhez, amiben még nem szerepel a recept.	A felhasználó vizsgált menüjében a receptek száma egyel nő.	✓

#	Követelmény	Tesztleírás	Mérés célja	Eredmény
12	Lehessen egy menühöz hozzáadni olyan receptet, ami már szerepel a menüben.	Létező felhasználó egy receptjének hozzáadása a felhasználó menüjéhez, amiben már szerepel a recept.	A felhasználó vizsgált menüjében a receptek ismétlődésének összege egyel nő.	✓
13	Ne lehessen archivált menühöz hozzáadni receptet.	Létező felhasználó egy receptjének hozzáadása a felhasználó archivált menüjéhez.	A menüt kezelő logika kivételt dob megfelelő hibaüzenettel.	✓
14	Lehessen egy menüből eltávolítani egy olyan receptet, ami nem szerepel többször.	Menü tulajdonosával nem ismétlődő recept eltávolítása a menüből.	A menü receptjeinek száma egyel csökken.	✓
15	Ismétlődő recept eltávolítása a menüből az ismétlődések számát csökkenti.	Menü tulajdonosával ismétlődő recept eltávolítása a menüből.	A menüben az eltávolított recept ismétlődésének száma egyel csökken.	✓
16	Lehessen generált menüt elmenteni.	Tetszőleges felhasználóval egy olyan menü mentése, amelyben már szerepelnek receptek.	Az adott felhasználó menüjeinek száma egyel nő.	✓
17	Lehessen egy menüben egy receptet késznek jelölni és a recept hozzávalóit vonja le a felhasználó alapanyagai közül.	Egy olyan recept késznek jelölése, amelyhez rendelkezésre áll elegendő alapanyag.	Az elkészített recept hozzávalóinak mennyiségével csökkent a megfelelő alapanyagok mennyisége.	✓
18	Lehessen hozzáadni új alapanyagot a bevásárlólistához.	Olyan típusú alapanyag felvétele a bevásárlólistára, ami még nem létezik sem a bevásárlólistában sem a felhasználó raktárában.	A bevásárlólistán szereplő alapanyagok száma egyel nő.	✓
19	Lehessen törölni egy alapanyagot a bevásárlólistáról.	Létező felhasználóval bevásárlólistán szereplő alapanyag törlése, aminek a felhasználó a tulajdonosa.	A bevásárlólistán szereplő alapanyagok száma egyel csökken.	✓

#	Követelmény	Tesztleírás	Mérés célja	Eredmény
20	Generált menü mentésekor a hiányzó alapanyagok kerüljenek fel a bevásárlólistára.	Olyan menü mentése, amelyben a receptek összes hozzávalójának mértéke meghaladja a raktárban található alapanyagok mennyiségét.	A bevásárlólistán szereplő alapanyagok mennyisége nő a raktárban található alapanyagok és a mentett menüben található hozzávalók mennyiségének különbségével.	✓
21	Egy recept késznek jelölése után a hozzávalóinak mennyisége legyen hozzáadva az aktuális statisztika felhasznált mennyiségeihez.	Egy hozzávalókat tartalmazó lista átadása a statisztikát kezelő logikának.	A statisztikában szereplő felhasznált mennyiség nő a hozzávalólista összesített mennyiségével.	✓
22	Legyenek ellenőrizve a felhasználó lejárt és felhasználható alapanyagai, illetve azok mennyisége kerüljön hozzáadásra a statisztika megfelelő tulajdonságához.	Olyan alapanyag lista ellenőrzése, ami tartalmaz az aktuális dátumnál régebbi és későbbi lejárati dátumú alapanyagokat is.	A statisztika lejárt élelmiszereket számon tartó mennyisége a lejárt alapanyagok mennyiségével nőtt, míg az elérhető mennyiségeket számon tartó mennyiség a még nem lejárt alapanyagok mennyiségével nőtt.	✓

6.1. táblázat. CRUD for database entities komponens unit tesztjei összefoglalva

#	Követelmény	Tesztleírás	Mérés célja	Eredmény
1	Az inicializáló egység legyen képes feldolgozni a bemeneti paraméterek valamennyi kombinációját és azok alapján elindítani a menügenerálás folyamatát. Továbbá a paraméterek és a rendelkezésre álló receptek alapján adjon megfelelő visszajelzést a felhasználónak a paraméterek változtatásáról, amennyiben az szükséges.	Egy tetszőleges felhasználó receptjei és több, különböző menügenerálási paraméterek átadásra kerülnek az inicializáló egységnek. A tényleges menügenerálást végző egység helyettesítve van egy egyszerűsített metódussal.	Az inicializáló egység a paraméterek és a receptek függvényében meghatározott darabszámú menüt generál, illetve a paraméterekhez megfelelő üzenetet is tartalmazza az eredmény.	✓

#	Követelmény	Tesztleírás	Mérés célja	Eredmény
2	Az inicializáló egység abban az esetben kivételt dob, ha a felhasználónak nincsenek receptjei.	Egy üres recept lista és tetszőleges paraméterek kerülnek átadásra az inicializáló egységnek.	Az inicializáló egység kivételt dob, ami tartalmazza a megfelelő hibaüzenetet.	✓
3	A felhasználók a saját receptjeik alapján tudjanak menüt generálni tetszőleges paraméterekkel.	Helyes menügenerálási paraméterekkel, létező receptekkel és alapanyagokkal elindul a menügenerálás. Egy segédmetódus meghatározza az összes recept értékét a paramétereknek megfelelően, majd ezek közül a legnagyobb értékűek kerülnek összegzésre.	A menügeneráló egység megoldása annyi alternatívát tartalmaz, amennyit a paraméterek meghatároztak. Továbbá az összes kiválasztott recept értéke megegyezik a segédmetódus által meghatározott összértékkel.	✓

6.2. táblázat. Menu generator komponens unit tesztjei összefoglalva

	Line coverage				
Name	Covered	Uncovered	Coverable	Total	Percentage
− Logic	713	93	806	1255	88.4%
Logic.Classes.AuthLogic	43	3	46	89	93.4%
Logic.Classes.FoodMaterialLogic	52	0	52	92	100%
Logic.Classes.FoodTypeLogic	57	0	57	99	100%
Logic.Classes.MenuLogic	179	21	200	275	89.5%
Logic.Classes.RecipeBookLogic	56	2	58	102	96.5%
Logic.Classes.RecipeLogic	64	4	68	117	94.1%
Logic.Classes.ShoppingListLogic	137	16	153	214	89.5%
Logic.Classes.StatisticsLogic	84	17	101	149	83.1%
Logic.Helpers.AutoMapperProfiles	41	0	41	64	100%
Logic.Services.TokenService	0	30	30	54	0%

6.1. ábra. CRUD for database entities komponens unit teszt lefedettsége

	Line coverage				
Name	Covered	Uncovered	Coverable	Total	Percentage
− MenuGenerator	266	0	266	393	100%
MenuGenerator.BTS.MenuGeneratorBTS	128	0	128	178	100%
MenuGenerator.Helpers.MenuWithValue	2	0	2	14	100%
MenuGenerator.Helpers.RecipeSelected	6	0	6	20	100%
MenuGenerator.Initializer.MenuGeneratorInitializer	130	0	130	181	100%

6.2. ábra. Menu generator komponens unit teszt lefedettsége

6.2. Manuális tesztek

A manuális tesztelés során a szerveroldali és a kliensoldali alkalmazás egyaránt ellenőrzésre került. A szerveroldali alkalmazás manuális teszteseteit nem kívánom részletesen bemutatni, mivel azok megegyeznek a unit tesztelés során megvizsgált tesztesetekkel. Emellett a kliensoldali alkalmazás manuális tesztjei egyúttal a szerveroldali alkalmazás helyes működését is alátámasztják. A manuális tesztesetek részletes leírását a 6.3. táblázat tartalmazza.

#	Követelmény	Tesztleírás	Mérés célja	Eredmény
1	Bejelentkezés után a felhasználó Raktár nézete töltsön be.	Létező felhasználóval belépés az alkalmazásba böngészőn keresztül.	A „Belépés” gombra kattintás után a Raktár nézet tölt be.	✓
2	Regisztráció után a felhasználó legyen beléptetve és a Raktár nézete töltsön be.	Új felhasználó regisztrálása böngészőn keresztül.	A „Mehet!” gombra kattintás után a Raktár nézete tölt be az új felhasználónak, a jobb felső sarokban pedig megjelenik az imént regisztrált felhasználónév.	✓
3	Bejelentkezés nélkül a regisztráción és a bejelentkezésen kívül ne legyenek elérhetőek az alkalmazás funkciói.	Nem hitelesített felhasználóként az URL kiegészítése a „/inventory”-val.	A Raktár nézet nem tölt be, helyette a főoldalra kerül átirányításra a felhasználó.	✓
4	„Új alapanyag hozzáadása” gombra kattintás után jelenjen meg egy form és az oldal fókusza kerüljön rá.	Bejelentkezett felhasználóval „Új alapanyag hozzáadása” gombra kattintás.	Az oldal alján megjelenik egy form és a böngésző automatikusan oda helyezi a fókuszt.	✓
5	A Menü fül alatt a „Menü generálása” gombra kattintva megjelenik egy form ahol beállíthatók a generálás paraméterei.	Bejelentkezett felhasználóval „Menü generálása” gombra kattintás.	A kattintás után megjelenik az elvárt form.	✓

#	Követelmény	Tesztleírás	Mérés célja	Eredmény
6	A menügenerálási paraméterek kiválasztását követően a „Mehet!” gombra kattintás után a megfelelő minőségű menüt/menüket kapjuk.	Bejelentkezett felhasználóval a „Több menü készítése” beállítása +10-re, az „Egy recept többször is előfordulhat” beállítása 2-re majd a „Mehet!” gombra kattintás.	A generálás 11 menüt ajánl fel mentésre, amik között előfordul olyan, ahol egy recept 2-szer is szerepel.	✓
7	A Raktárban csak akkor jelenik meg az aktuális havi statisztika, ha van felhasznált vagy lejárt élelmiszer.	Újonnan regisztrált felhasználóval hozzáadni egy olyan alapanyagot, ami később jár le, majd generálni egy menüt és késznek jelölni egy receptet.	Az alapanyag hozzáadása után a Raktár nézetben nem jelenik meg a statisztika, de egy recept késznek jelölése után a statisztika megjelenik a frissített adatokkal.	✓
8	A „Kimutatások” menüpont alatt megtekinthetőek az aktuális és a korábbi statisztikák.	Bejelentkezett felhasználóval „Kimutatások” menüpont megnyitása.	Az oldal betöltése után az aktuális statisztika jelenik meg, illetve jobbra és balra mutató nyilakkal lehet navigálni a korábbi statisztikák között.	✓

6.3. táblázat. Kliensoldali alkalmazás manuális tesztjei

7. ÉRTÉKELÉS

A rendszer elkészítésének és tesztelésének befejezésével kijelenthető, hogy az alkalmazással szemben támasztott követelmények többnyire teljesülnek. A 4.4. fejezetben meghatározott funkciólista alapján a fejlesztést követően az alábbiak valósultak meg.

Az optimalizált menügenerálás változtatható paraméterekkel funkció teljes mértékben megvalósításra került. A visszalépéses keresésen alapuló algoritmus a felhasználó saját receptjei alapján készíti el a lehető legnagyobb jósággal rendelkező menüt. A jóság meghatározásában minden esetben számít a recepteket alkotó hozzávalók lejárati dátuma. Egy menü jóságának meghatározását két további paraméterrel lehet befolyásolni, a receptek elkészítési idejének a figyelembe vételével, illetve a minél több alapanyag felhasználásával. Ezen felül a felhasználónak lehetősége van további paramétereket beállítania a számára megfelelő menük előállításához. Megadható, hogy egy menü hány receptet tartalmazzon, hány főre készüljön, illetve hány alternatív menü készüljön. Beállítható még, hogy egy recept maximum hányszor fordulhat elő egy menüben, továbbá lehetőség van szűrni a kiválasztandó receptekre a nehézségük alapján.

Az alkalmazással szemben elvárás még a CRUD műveletek elkészítése különböző objektumok esetében. Ennek a funkciónak is maradéktalanul teljesült az implementációja. A felhasználóknak lehetőségük van a saját alapanyagaikat, receptjeiket, receptkönyveiket és menüiket kezelni más felhasználók adataitól szeparáltan. Az előbb felsorolt objektumok közül bármelyik létrehozható, törölhető, lekérhető és módosítható, amennyiben az adott objektumnak a műveletet végző felhasználó a tulajdonosa. Értelemszerűen a létrehozásnál a műveletet végző felhasználó válik az objektum tulajdonosává.

További funkcionális elvárás még a bevásárlólista készítése. Egy felhasználó regisztrálása során a szerveroldali alkalmazás inicializál egy bevásárlólistát, aminek a felhasználó lesz a kizárólagos tulajdonosa. Erre a listára tetszőleges alapanyagokat vehet fel a tulajdonosa, illetve törölni is tudja azokat. Amikor a felhasználó egy generált menüt elment, a szerveroldali alkalmazás a menü sikeres mentését követően ellenőrzi a felhasználó rendelkezésre álló alapanyagait. Amennyiben a menü receptjei között szerepel olyan hozzávaló, aminek a mennyisége meghaladja a megfelelő típusú rendelkezésre álló alapanyag mennyiségét, akkor a megfelelő típusú alapanyag a hiányzó mennyiséggel automatikusan felkerül a bevásárlólistára. Tehát ezen funkció is teljes mértékben megvalósításra került a fejlesztés során.

A rendszerrel szemben támasztott elvárás még, hogy összesítő statisztikák készüljenek, amelyek segítségével a felhasználó nyomon követheti a szeméttermelését és a fejlődését. A funkció a következők szerint került megvalósításra: a statisztikák a felhasználó alapanyagait vizsgálja havi bontásban. A statisztika tartalmazza a felhasznált, a lejárt és az elérhető élelmiszermennyiségeket. A kliensoldali alkalmazásban két helyen szerepelnek a statisztikák, a *Raktár* és a *Kimutatások* menüpontok alatt. A Raktár

menüpont alatt mindig az aktuális hónap statisztikája látható, amennyiben az adott hónapban van már lejárt vagy felhasznált élelmiszer. A *Kimutatások* menüpontban az aktuális és a korábbi statisztikák is megtekinthetők, ezzel biztosítva a fejlődés szemléltetését. Amikor a kliensoldali alkalmazás lekéri az összes vagy az aktuális statisztikát, a szerveroldali alkalmazás ellenőrzi a felhasználó alapanyagait ezáltal naprakészen tartva az aktuális havi lejárt és elérhető élelmiszerek mennyiségét. Továbbá egy menü sikeres késznek jelölésekor a felhasznált élelmiszerek mennyisége is hozzáadásra kerül a statisztikához.

A funkciólista szerint elvárás még az alkalmazástól a varázsló szerű adatbevitel bevásárlás után és a jutalomrendszer alkalmazása. A rendszer fejlesztése során ezekre a funkcióra nem jutott elegendő idő a szoros határidő miatt ezért nem kerültek megvalósításra, de az alkalmazás legfontosabb célkitűzését ezen implementációk hiánya nem befolyásolja számottevően.

A rendszer komponenseinek felépítése és a közöttük lévő kapcsolatok a tervezettnek megfelelően alakultak pár módosítást leszámítva. Első sorban a szerveroldali alkalmazás *Shopping list* komponense nem került külön megvalósításra, mivel az elkészült logika nagy része CRUD műveletekből áll, így a *CRUD for database entities* komponensbe került be.

További szerkezeti eltérés kivitelezésére az adatbázis táblái között volt szükség. Elengedhetetlen volt bevezetni egy FoodType táblát, hogy a kapcsolat meglegyen a FoodMaterial és az Inventory táblák között, de ne kelljen minden azonos típusnak összeköttetésben állnia, mivel így számos entitás esetében az adatbázis könnyen átláthatatlanná válna.

A menügenerálás tervezett megvalósítása is eltér a tényleges implementációtól, mégpedig abban, hogy a generált menü mentését nem a *Backtrack search* egység végzi, hanem a felhasználó kezdeményezi azt a kliensoldali alkalmazás felől. Erre azért volt szükség, mivel alternatív menük generálásakor nem szükséges elmenteni az összes előállított menüt, csak azt amelyiket a felhasználó szeretne menteni.

A lefejlesztett alkalmazások megfelelő működésének ellenőrzésére számos unit teszt és manuális teszt eset készült. A megvalósított komponensek tesztjei mind sikeresek voltak, ezért elég nagy biztonsággal állítható, hogy az elkészült rendszer a tervezettnek megfelelően működik.

8. TOVÁBBFEJLESZTÉSI LEHETŐSÉGEK

A rendszer elkészítése során számos olyan funkcióra nem jutott elegendő fejlesztési idő, amelyek jelentősen javítanák az alkalmazás kezelhetőségét és minőségét. Ilyen funkciók a tervezésben meghatározott varázsló szerű adatbevitel és a jutalomrendszer alkalmazása, amelyek növelnék a rendszer használhatóságát.

További fejlesztési lehetőség még, egy vonalkód beolvasó komponens beépítése a rendszerbe. Ennek segítségével az alapanyagok felvitele számottevően kényelmesebb lenne a felhasználók számára, különösen a regisztrációt követően, amikor a meglévő élelmiszereiket a mobiljuk használatával egyszerűen adhatnák hozzá az alkalmazáshoz. A vonalkód beolvasás segítségével a bevásárlás utáni raktár feltöltés is egyszerűbbé válna.

Hasznos lenne egy közösségi platform kialakítása is, ahol a készételeket vagy a nem kívánt élelmiszereket tudnák felajánlani vagy elcserélni egymás között a felhasználók. Ez a funkció illene az alkalmazás legfontosabb célkitűzéséhez, ami az élelmiszerhulladék mennyiségének csökkentése.

Számottevő minőségnövelő fejlesztés lenne a felhasználók étkezési szokásainak nyomon követése. Az ebből származó adatok feldolgozásának segítségével a menügenerálás kiegészíthető lenne úgy, hogy a generált menü elősegítse a változatos étrend kialakítását. Ezzel az élelmiszerpazarlás problémája mellett az egyoldalú étkezésből fakadó nehézségek kiküszöbölését is támogatná az alkalmazás.

Az elkészült rendszer jelenleg nem rendelkezik különböző szerepkörökkel, hanem felhasználó orientáltan működik. A szerepkörök bevezetésével le lehetne képezni a több fős háztartásokat, ahol mindenki ugyanahhoz a felülethez férne hozzá eltérő jogosultságokkal. Emellett érdemes lenne implementálni a *signalR*-t, ami biztosítaná, hogy az egy háztartáshoz tartozó felhasználók mindig naprakész információkat kapjanak az adataik állapotáról.

A bevásárlólista kiszervezése külön komponensbe és annak továbbfejlesztése is ígéretes lehet. Manapság már számos lehetőség van otthonról intézni a bevásárlást interneten keresztül. Ebből kifolyólag előnyös lenne egy olyan funkció elkészítése is, amely a Mealime-hoz hasonlóan a kiválasztott kereskedő oldalán lépésről lépésre végig vezetné a felhasználót a bevásárlólistája alapján az adott kereskedőnél található szükséges alapanyagokon, amiket így könnyedén meg tud vásárolni.

Az imént felsorolt lehetőségek is mutatják, hogy az elkészült alkalmazásban rengeteg továbbfejlesztési potenciál van, amennyiben rendelkezésre állnak a funkciók elkészítéséhez szükséges erőforrások és idő.

9. IRODALOMJEGYZÉK

[1] Supercook, (https://www.supercook.com), utoljára megtekintve: 2021.09.27.
[2] Mealime, (https://www.mealime.com), utoljára megtekintve: 2021.10.01.
[3] Food Planner, (http://www.foodplannerapp.com), utoljára megtekintve: 2021.10.03.
[4] Szénási, S.: Algoritmusok, adatszerkezetek II. *ÓE-NIK 5013*, 2018
[5] De St. Germain, H. J.: Recursion, (https://www.cs.utah.edu/~germain/PPS/Topics/recursion.html), utoljára megtekintve: 2021.10.26.
[6] Skiena, S. S.: The Algorithm Design Manual. *Springer*, 2008
[7] Bradley, S. P., Hax, A. C., Magnanti, T. L.: Applied Mathematical Programming. *Addison-Wesley*, 1977
[8] Neapolitan, R. E.: Foundations of Algorithms. *Jones & Bartlett Learning*, 2015
[9] GeeksforGeeks, (https://www.geeksforgeeks.org/0-1-knapsack-problem-dp-10/), utoljára megtekintve: 2021.11.12.
[10] Kasza, G., Dorkó, A., Kunszabó, A., Szakos, D.: Quantification of Household Food Waste in Hungary: A Replication Study Using the FUSIONS Methodology. *Sustainability*, Vol. 12., 2020, pp. 1-3.
[11] Microsoft Entity Framework Core, (https://docs.microsoft.com/en-us/ef/core/), utoljára megtekintve: 2021.12.04.
[12] Microsoft ASP.NET, (https://github.com/dotnet/aspnetcore), utoljára megtekintve: 2021.12.04.
[13] ngx-charts, (https://swimlane.gitbook.io/ngx-charts/), utoljára megtekintve: 2022.04.24.
[14] ngx-toastr, (https://www.npmjs.com/package/ngx-toastr), utoljára megtekintve: 2022.04.24.

10. ÁBRAJEGYZÉK

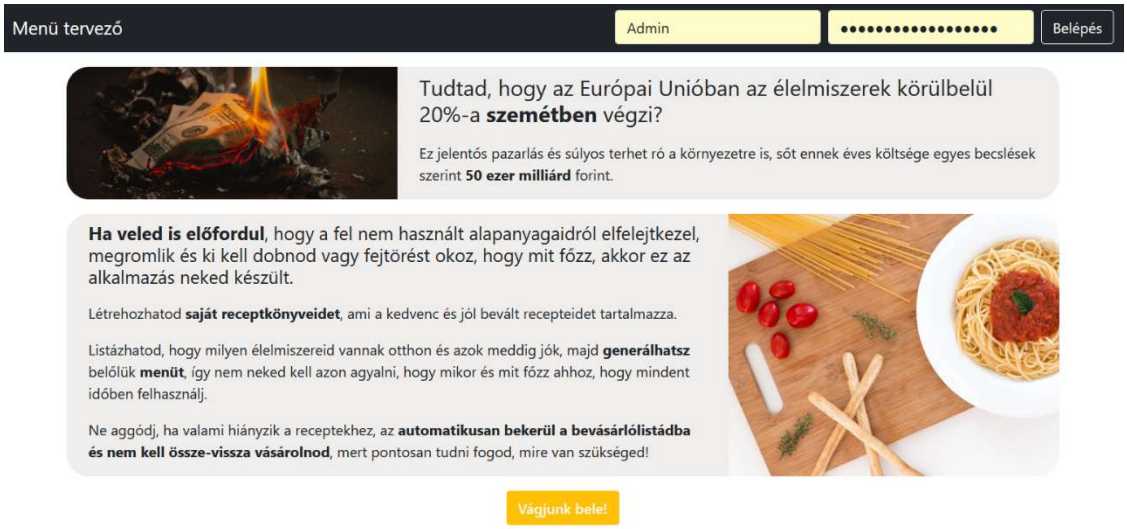
11. TÁBLAJEGYZÉK

12. MELLÉKLETEK

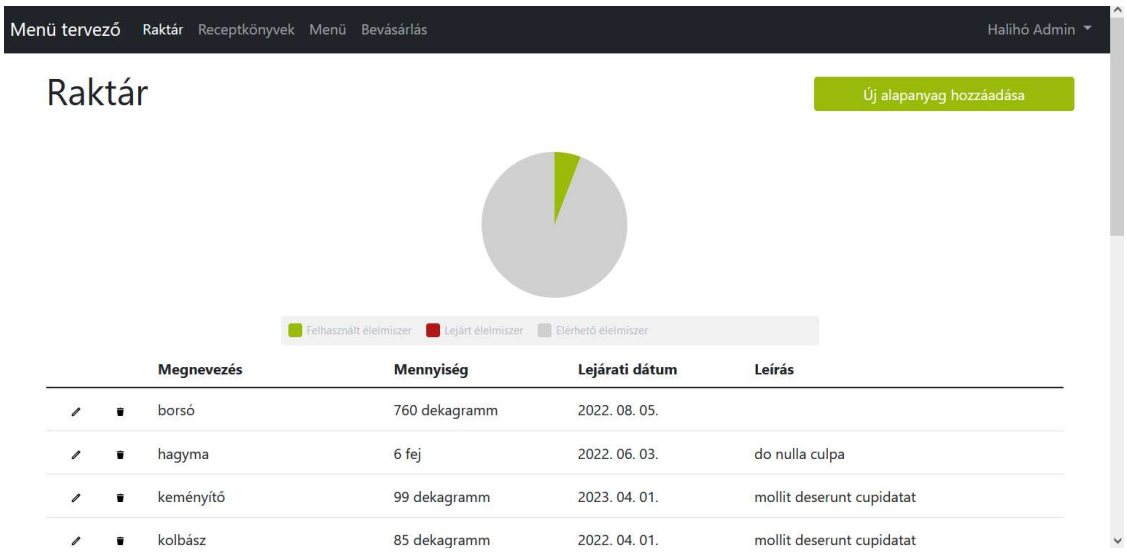
Az elkészült alkalmazás itt érhető el: https://menugeneratortest.azurewebsites.net
Bejelentkezési adatok a beépített felhasználóhoz:

- Fehasználónév/e-mail: Admin/test@mail.hu
- Jelszó: veryStrongPa$$w0rd

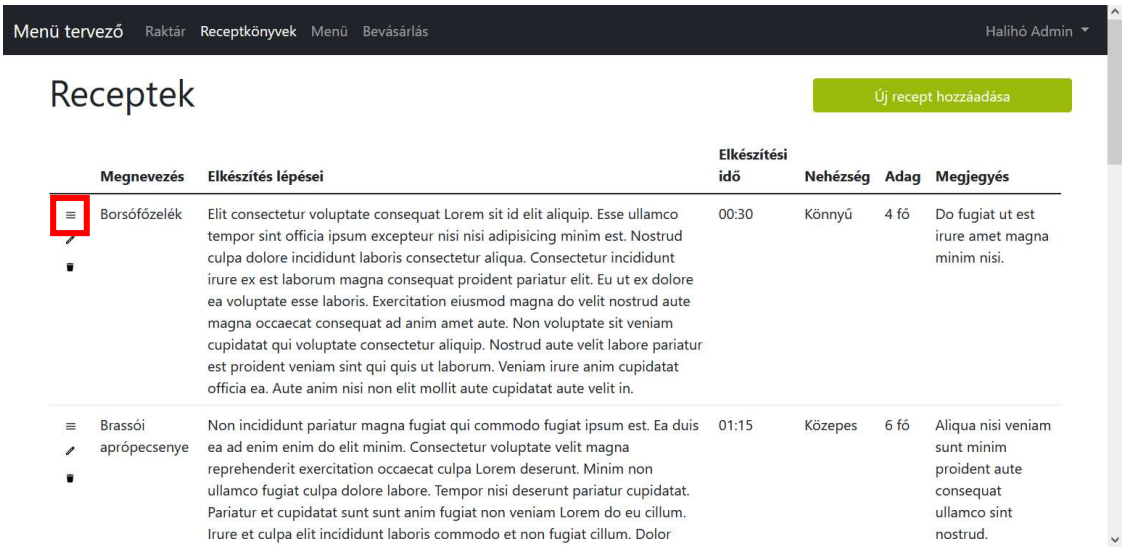
Képek az alkalmazásról:



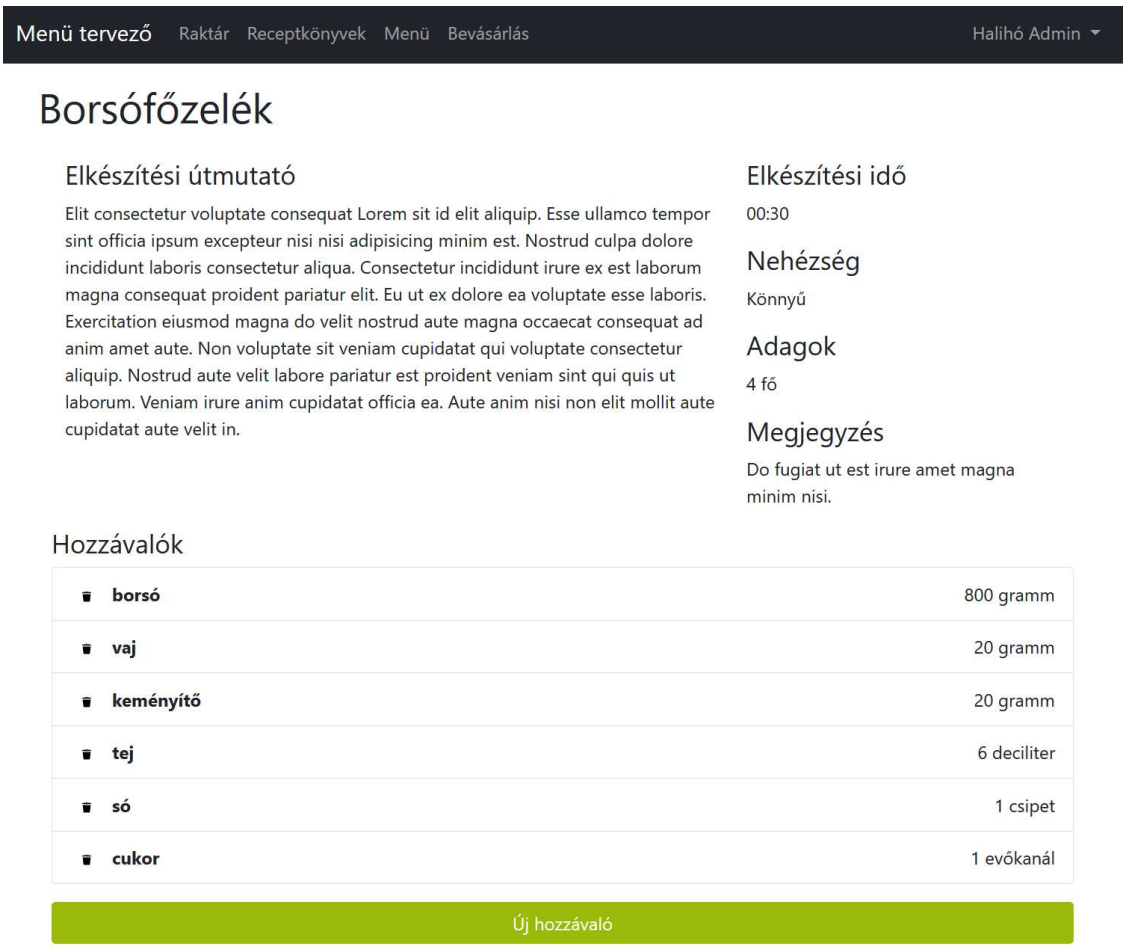
A fenti képen az alkalmazás nyitóoldala látható, ahol a *Vágjunk bele!* gombra kattintással megjelenik a regisztrációs form.



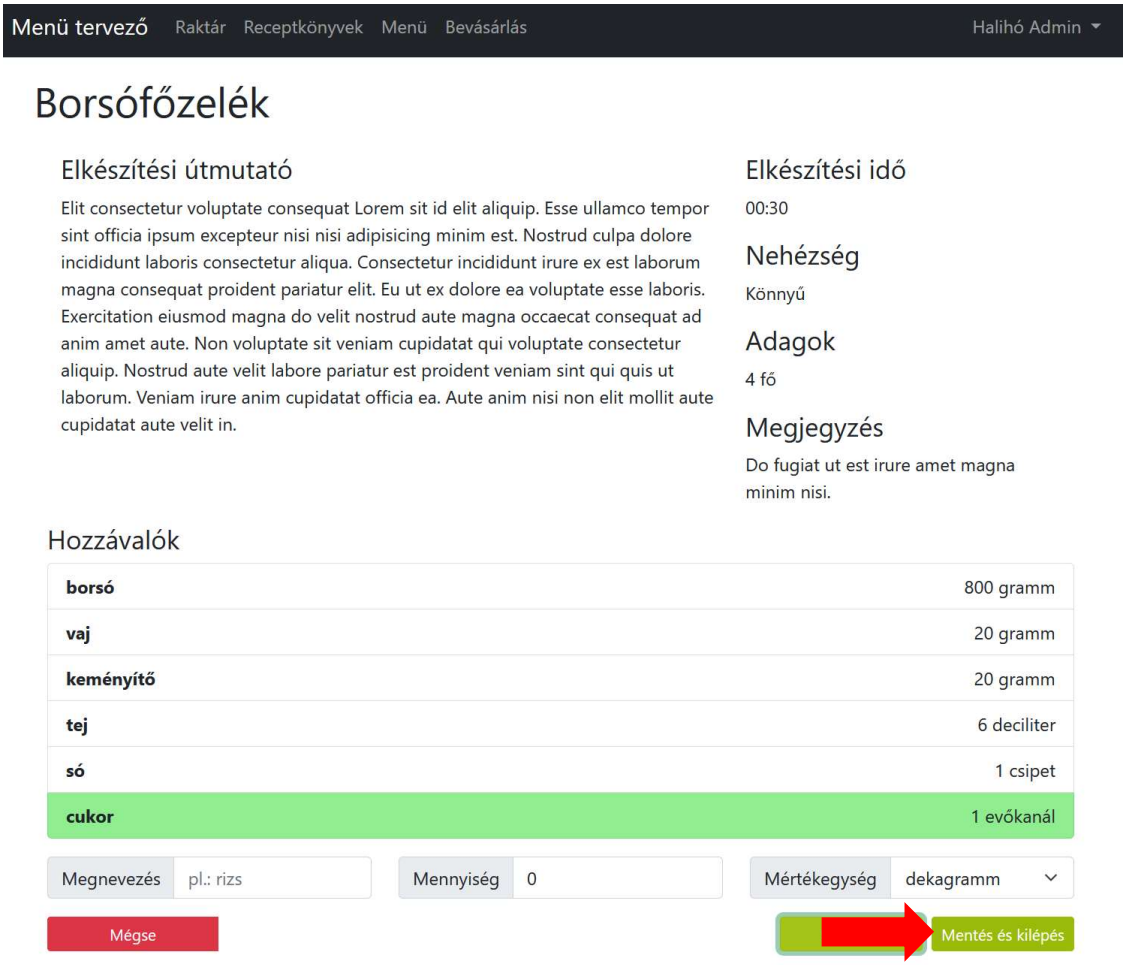
A képen az **Admin** *Raktár* nézete látható, ahol az aktuális havi statisztika jelenik meg, illetve az **Admin** hozzávalóinak listája.



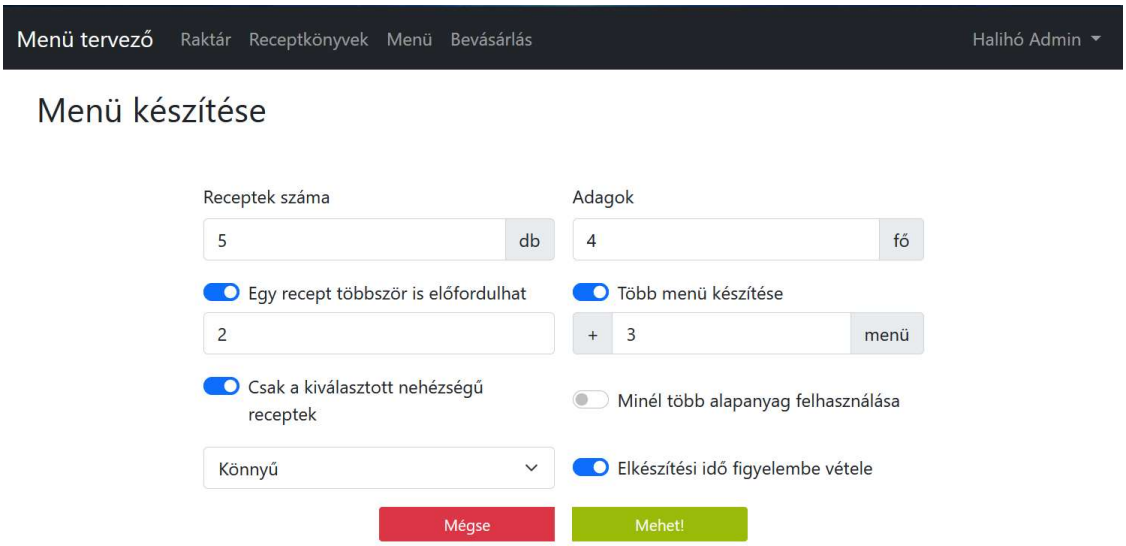
A képen az **Admin** receptjeinek listája látható. A piros négyzettel megjelölt gombra kattintással érhető el a receptek részleteinek megtekintése.



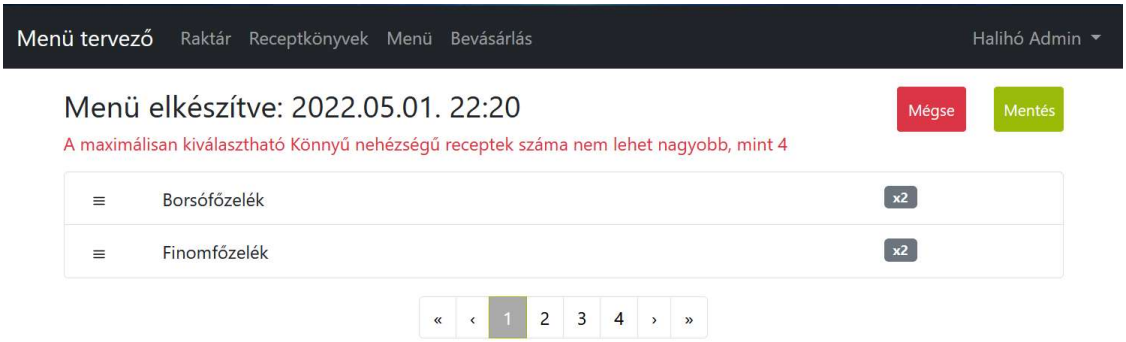
A képen az **Admin** egy receptje látható, a recepthez tartozó adatokkal és a hozzávalók listájával.



A képen egy recepthez új hozzávaló hozzáadásának folyamata látható. A zöld háttérrel kiemelt „cukor” hozzá lett adva a recept hozzávalóihoz, de még nem lett elmentve. A piros nyíllal jelzett gomb lenyomásával lehet menteni a módosításokat.

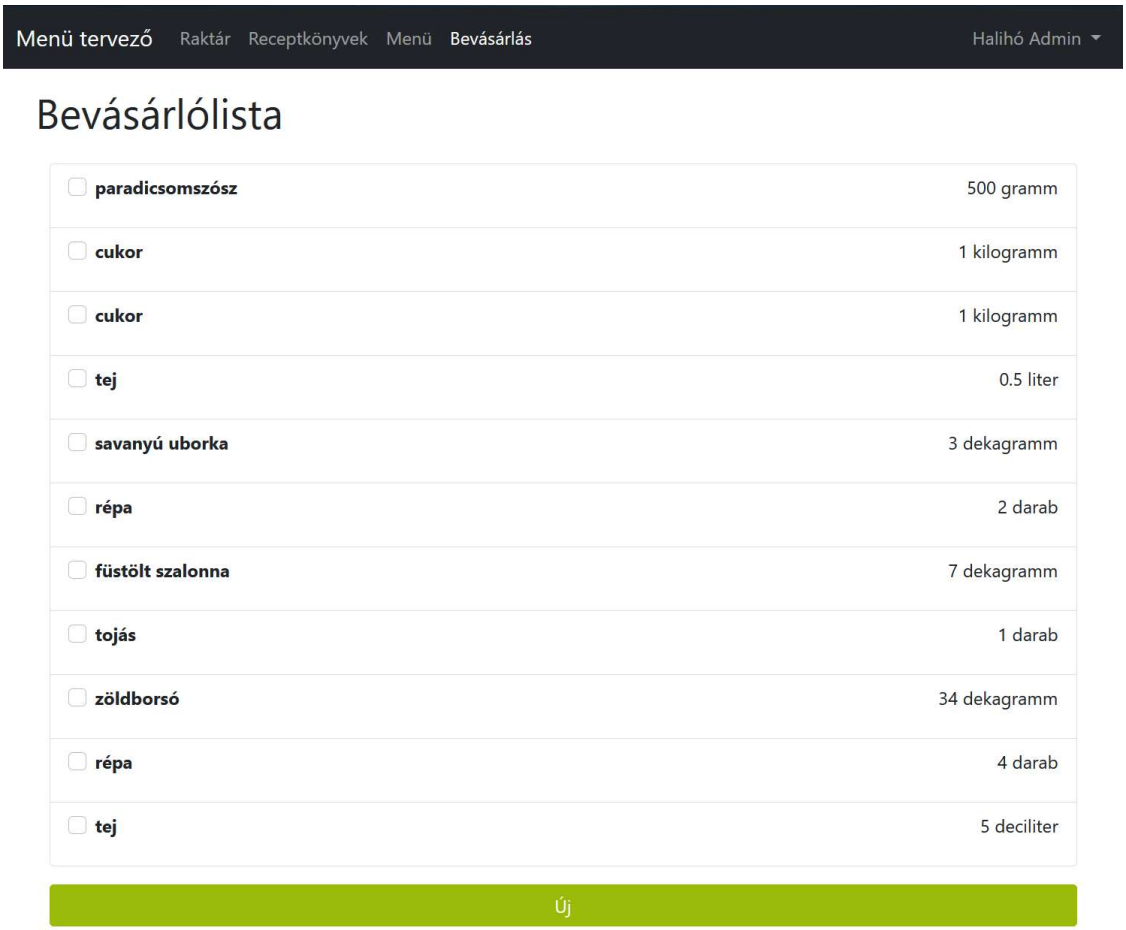


A képen a menügenerálás paramétereinek beállítása látható. A kívánt paraméterek megadása után a *Mehet!* gombra kattintva elindítható a generálás folyamata.

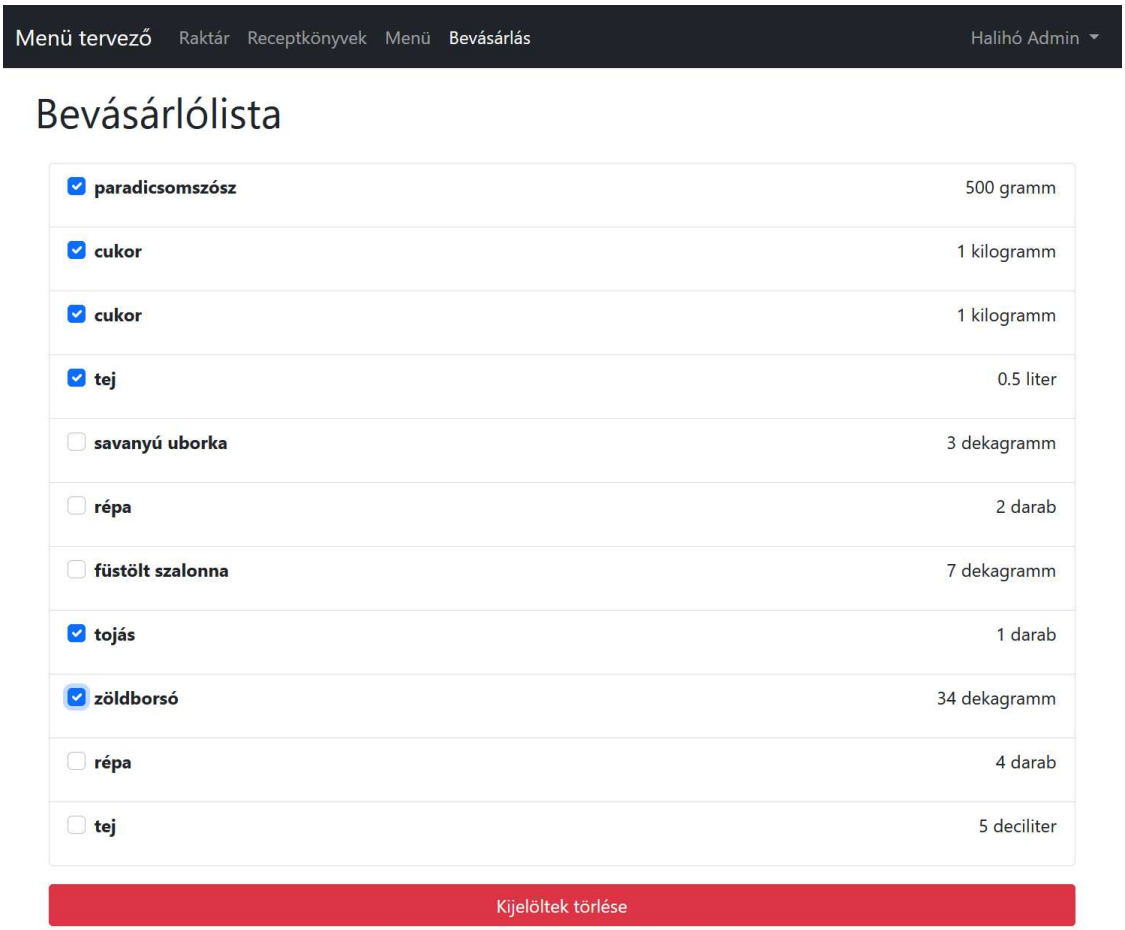


A képen a fentebb meghatározott paraméterekkel indított menügenerálás eredménye látható.

A képen a *Kimutatások* nézete látható, amelyet a piros nyíllal jelzett menüponton keresztül érhet el a felhasználó.



A képen az **Admin** bevásárló listája látható. Az *Új* gombra kattintva hozzáadható egy tetszőleges új elem a listához.



A képen a bevásárlólistáról való törlés folyamata látható. A *Kijelöltek törlése* gombra kattintva a kijelölt elemek törlésre kerülnek a listából.