|  |  |  |
| --- | --- | --- |
|  | **NEUMANN JÁNOS**  **INFORMATIKAI KAR** | NIK_cimer.jpg |

|  |  |  |
| --- | --- | --- |
| **OE-NIK**  **2020/21** | Hallgató neve:  Hallgató törzskönyvi száma: | **Lindner Márton Tamás**  **T/005725/FI12904/N** |

**SZAKDOLGOZAT**

Tartalomjegyzék

Tartalomjegyzék

[1. Absztrakt 5](#_Toc58271628)

[2. Bevezetés 6](#_Toc58271629)

[2.1. Motiváció 6](#_Toc58271630)

[3. Az irodalom alapján a lehetséges megközelítési módok és megoldások áttekintése és elemzése 8](#_Toc58271631)

[3.1. Hasonló munkák áttekintése 8](#_Toc58271632)

[3.2. Genetikus algoritmus 10](#_Toc58271633)

[4. Probléma definíció 14](#_Toc58271634)

[5. A megoldási módszer kiválasztása, a választás indoklása 17](#_Toc58271635)

[5.1. Fitnesz 19](#_Toc58271636)

[6. Technológiai ismertetés 21](#_Toc58271637)

[6.1. Android 21](#_Toc58271638)

[6.2. Mi is az a webapplikáció? 22](#_Toc58271639)

[6.3. Web Architektúra 22](#_Toc58271640)

[6.3.1. Kliens-szerver 23](#_Toc58271641)

[6.3.2. Egységes interfész 23](#_Toc58271642)

[6.3.3. Rétegelt rendszer 24](#_Toc58271643)

[6.3.4. Gyorsítótár 24](#_Toc58271644)

[6.3.5. Állapotmentes 24](#_Toc58271645)

[6.3.6. Code-on-demand 24](#_Toc58271646)

[6.4. Webalkalmazás – Gyakorlati Megvalósítás 24](#_Toc58271647)

[6.5. Ruby on Rails 25](#_Toc58271648)

[6.6. Codeigniter 25](#_Toc58271649)

[6.7. Express.js 25](#_Toc58271650)

[6.8. ASP.NET 26](#_Toc58271651)

[6.8.1. Mi az a .NET platform? 26](#_Toc58271652)

[6.8.2. ASP.NET Core 26](#_Toc58271653)

[6.9. MVC tervezési minta és továbbfejlesztése 27](#_Toc58271654)

[6.9.1. Modell 27](#_Toc58271655)

[6.9.2. Nézet 27](#_Toc58271656)

[6.9.3. Részleges Nézet 28](#_Toc58271657)

[6.9.4. Kontroller 28](#_Toc58271658)

[6.9.5. Köztesréteg 29](#_Toc58271659)

[6.9.6. Függőség beinjektálása 30](#_Toc58271660)

[6.9.7. Kontroll megfordítása 31](#_Toc58271661)

[6.10. Entity Framework 31](#_Toc58271662)

[6.11. Front-end keretrendszerek 32](#_Toc58271663)

[6.11.1. React 32](#_Toc58271664)

[6.11.2. Angular 32](#_Toc58271665)

[6.11.3. Vue.js 33](#_Toc58271666)

[6.12. Választott technológia 33](#_Toc58271667)

[7. Specifikáció 34](#_Toc58271668)

[7.1. Funkcionális követelmények 34](#_Toc58271669)

[7.2. Nem-funkcionális követelmények 35](#_Toc58271670)

[8. A tervezés során végzett munkafázisok és tapasztalataik leírása 36](#_Toc58271671)

[8.1. Strukturális leírás 37](#_Toc58271672)

[9. A megvalósítás leírása 39](#_Toc58271673)

[9.1. Az adatok beszerzése 39](#_Toc58271674)

[9.2. Entity modell 39](#_Toc58271675)

[9.3. Backend projekt felépítése 40](#_Toc58271676)

[9.4. Frontend projekt felépítése 48](#_Toc58271677)

[10. Eredmények 51](#_Toc58271678)

[10.1. Optimális GA paraméterek 51](#_Toc58271679)

[10.2. Konvergencia gyorsasága az optimális megoldáshoz 52](#_Toc58271680)

[10.3. Továbbfejlesztési lehetőségek 53](#_Toc58271681)

[11. Összefoglalás 54](#_Toc58271682)

[12. Conclusion 55](#_Toc58271683)

[13. Melléklet 56](#_Toc58271684)

[14. Irodalomjegyzék 60](#_Toc58271685)

# Absztrakt

Munkám célja egy olyan alkalmazás kifejlesztése, melynek segítségével a felhasználó célkitűzésének megfelelő teljes értékű étrendet kaphat. A megfelelő paraméterek megadását követően az alkalmazás kiszámolja tápanyagigényeit, majd ez alapján ajánlja föl az optimális étrendet adott napra. Az étrend generálást a felhasználó valamennyi alkalommal megismételheti, a módszer heurisztikus mivoltának köszönhetően teljesen különböző étrendeket kap, melyek közül kiválaszthatja a számára legideálisabbat. A felhasználó megadja a napi étkezések számát, melyet figyelembe vesz az algoritmus. További szükséges paraméterek a nem, kor, magasság, tömeg, aktivitás, mely adatok szükségesek a napi tápanyag- és energiaszükséglet megállapításához. Továbbá a felhasználó megadja az egyes ételcsoportokra vonatkozó preferenciáit is, a prezentált ételek fő- és alkategóriáinak 10-es skálán való értékelése alapján. A fő cél, hogy az ételek egy olyan halmazát prezentálja a program, mely kielégíti a napi tápanyagszükségletet, és maximális összértékeléssel rendelkezik. Kutatómunkámban először kitérek a motivációra, majd megvizsgálom, milyen megoldás létezik már a problémára. Ezt követően pontosan definiálom a megoldandó problémát, majd meghatározom annak megoldási módszerét. Ezt követően megvizsgálom a lehetséges platformokat, melyeken az implementáció felmerülhet. Következő lépésként meghatározom a pontos specifikációt, és írok a tervezés során végzett munkafázisokról, valamint azok tapasztalatairól. Végezetül a megvalósított alkalmazást mutatom be és az elért eredményeket közlöm.

# Bevezetés

## Motiváció

Ma Magyarországon ez egyik legsúlyosabb népbetegség az elhízottság. Az ország 16 évesnél idősebb lakosságának 35,8% [1]-a elhízott. Az ülőmunkát végző lakosság 41,9%-a nem végez rendszeres testmozgást. Az elhízott embereknél nagyobb hajlam van a rák kialakulására. Továbbá rengeteg más egészségügyi kockázattal jár, főleg szív- és érrendszeri megbetegedések kockázatát növeli. Habár a barokk korban a túlsúly a gazdagság, a jólét jelképe volt, az elhízás lehangoltságot és súlyos esetben depressziót is okozhat. A túlsúly sokszor magas vérnyomáshoz is vezet, emellett érelmeszesedést, koszorúér-betegségeket, és agyérkatasztrófát is eredményezhet. Az elhízás kiválthatja a cukorbetegséget, érelzáródást okozhat, máj- és epehólyag betegségek előidézője lehet, valamint hozzájárulhat az ízületi porc-kopáshoz és a köszvényhez is [2]. Ezen megfontolásból döntöttem úgy, hogy segíteni kívánok az egészséges táplálkozás irányába elmozduló embereknek ezen ingyenes szolgáltatással

A számítógép által támogatott menü generálás az 1960-as évek óta egy nehézkes feladat. (Eckstein 1978). Mind a mai napig nem sikerült olyan alkalmazást készíteni, mely konzisztensen felülmúlja a dietetikus szakértők által készített étrendet. A feladat nem egyszerű: olyan étrendet generáljunk, mely nem csak kielégíti egy személy napi tápanyagszükségleteit, hanem étvágygerjesztő is. Egy étrend testre szabásához számos tápanyag korlátnak szükséges megfelelnie, az egyén személyes preferenciáit is figyelembe kell venni, valamint ki kell elégíteni az esztétikai követelményeket is.

Egy dietetikus számára egy étrend megtervezése egy egyén számára minimum 30 percet, de akár 3 órát is igénybe vehet. Több tucat tápanyagot kell figyelembe venni, valamint több ezer elemű étel halmazból kell kiválasztani a megfelelő kombinációt. Egy 1995-ös tanulmány kimutatta, hogy még a táplálkozási szakértők által tervezett étrendek sem felelnek meg a tápanyag követelményeknek. [3]

Az étrend tervezése során legalább 3 szempontot kell figyelembe vennem, melyek megnehezítik a feladatomat. Elsőként meg kell felelni számos tápanyag követelménynek, melyek közül több is konfliktusban áll egymással. Például ha visszaszorítjuk a húsfogyasztást az amerikai étkezési irányelveknek megfelelően, nehéz megfelelni az EDA által ajánlott cink bevitelnek. Továbbá egy étrendet mindaddig nem lehet kiértékelni, amig az nem ölti ki teljes formáját. Egy dietetikus egy étrendet nem ételenként vagy fogásonként értékel ki, hanem azt egészében vizsgálja. Mindemellett a józan ész követelményeinek is meg kell felelni. Vannak ételek, amik illenek egymáshoz (töltött csirkecomb), mások viszont nem. Kezdeti étrend tervező alkalmazások teljesen valószerűtlen ajánlásokat tettek. Például egyél egy nyers répát, igyál három liter limonádét , vagy reggelire fogyassz csokoládéba mártott mandulát párolt paradicsommal [4]. Ezen ételek habár megfeleltek a tápanyag követelményeknek, beláthatjuk, hogy nem túl valószerű ajánlások.

Egy személy napi étrendjének megállapítása a következők miatt érdekes probléma. Az étrenddel szemben számos követelmény megjelenhet, mint maximum és minimum energiatartalom, ásványi anyag, fehérje, zsír, és egy egyéb tápanyag tartalom. A klasszikus étrend probléma ezzel a kérdéskörrel foglalkozik [5]. Itt Stigler célja ételek egy olyan kombinációját megtalálni, mely minimális költséggel jár, miközben kielégíti az összes – egy személy számára szükséges – napi tápanyagigényt. Jelenlegi munkámban egy hasonló problémával foglalkozom, mindezt kiegészítve egy további céllal. Programomban a felhasználó értékelheti a felajánlott ételek kategóriáit személyes ízlése alapján. Tehát a kiegészítő cél a felhasználói értékelés maximalizálása a napi étrendben. A területen már számos kutatás történt, melyeket a következő fejezetben kívánok áttekinteni.

# Az irodalom alapján a lehetséges megközelítési módok és megoldások áttekintése és elemzése

## Hasonló munkák áttekintése

A Menütervezési probléma megoldására 1964-ben készült előszőr számítógép által támogatott megoldás. Balintfy [6] létrehozta az első menü tervező programot. A létrehozott menüvel szemben a következő követelményeket határozta meg: minimális költség valamint a napi tápanyagszükségletek elérése. Mindemellett figyelembe vette az menü sokszínűségét (szín, konzisztencia). Munkájában Lineáris Programozási technikát alkalmazott. A javasolt algoritmust egy kórházban alkalmazták, segítségével mintegy 30%-al csökkenteni tudták a menü költségét. Később Eckstein véletlenszerű keresést alkalmazott a tápanyag követelmények teljesítésére.

Ezt követően mesterséges intelligencia módszereket fejlesztettek ki leginkább precedens alapú[[1]](#footnote-1) [6] vagy szabály alapú[[2]](#footnote-2) [8] technikák mentén. Egy hibrid CBR-RBR rendszer, a CAMPER [9] integrálja a két független rendszer előnyeit: a precedens-alapú menü tervezőt - CAMP [10], valamint a PRISM [10] rendszert. Egy frissebb CBR megközelítés a MIKAS, menü konstrukció inkrementális tudás megszerzésen alapuló rendszer [12]. A MIKAS lehetővé teszi tudásbázisának inkrementális fejlesztését. Amennyiben az eredmény nem kielégítő, a szakértő manuálisan módosítja a rendszer által generált étrendet. Egy webes alapú rendszert készítettek Malajziában, mely a dietetikusok munkamenetet modellezi [13]. Célja az étrend generálás és annak menedzselése.

Aynur Kaharam és H. Aydolu Seven munkájukban [14]egy kettős célú étrend problémáját kívánják megoldani, mellyel egy egészséges ételt biztosíthatnak a felhasználónak a megadott paraméterek alapján. Itt a felhasználó kora és neme alapján határozzák meg annak napi tápanyag- és energiaigényét. Továbbá a felhasználó megadja az étel lehetőségek alapján preferenciáit. Céljuk a felhasználó számára az ételek egy olyan halmazát biztosítani, melyek kielégítik következő feltételeket: (i) a napi tápanyagszükségletetek megfelelő, (ii) minimális napi költséggel rendelkezik, (iii) maximális összértékeléssel rendelkezik.

Feltételezzük, hogy van n különböző fogásunk, melyeket ajánlani tudunk a felhasználó számára. Továbbá ismerjük a fogások egy adagra jutó tápanyagösszetételét és annak költségét. Megkapjuk a felhasználótól miden fogás értékelését. A napi tápanyagkövetelmények nagy része alsó határt szab meg, mint például a minimális vitamin- és tápanyagmennyiség. Mindemellett vannak felső korlátok is, mint például a maximális K vitamin mennyiség. Kaharam és Aydolku kutatásukban minden követelményt felső korlátként definiálnak, az alsó korlátok felső korláttá való áttranszformálásával. Céljuk meghatározni, hogy mely étel szerepeljen a napi tervben. Minden ételre egy választási flaget alkalmaznak, mely jelöli annak jelenlétét vagy hiányát (1, ha az étel szerepel, 0 ha az étel nem szerepel). Így a probléma a következő módon írható le:

Itt a j fogás a döntési változója; a felhasználó által megadott értékelés, annak a költsége; az i. megkötési érték; a j. fogás az i megkötésnek megfelelő tápanyagértéke.

A kutatás megemlíti annak imitációit is. A fentiekben egy fogásnak csak egy adagja lehet részese a napi étrendnek. Másik limitáció, hogy csupán egy fogást javasol a napra, miközben a valóságban egy egészséges napi étrend legalább három, különböző fogásokból álló étkezésből áll.

A kettős-célú diéta probléma hátizsák problémaként [15] modellezhető. Az egymegkötéses hátizsák probléma a következőképpen definiálható:

A hátizsák problémában az egyes elemek rendelkeznek profit értékekkel (), valamint egy bizonyos erőforrást igényelnek (). A probléma tárgya megtalálni az elemek azon kombinációját, mely maximális profittal rendelkezik és nem haladja meg a erőforrás korlátozást (c). A változók () adják meg, hogy az elem részét képezi- e a megoldásnak, vagy nem (értéke 0 vagy 1 lehet). A mi problémánkban az elemek a rendelkezésre álló fogások. A fenti hátizsák probléma csak egyetlen megkötéssel rendelkezik, viszont az étrend problémában számos különböző megkötésünk van. Mindemellett a problémánknak 2 célja van. Emiatt az étrend problémánk egy több-célú többdimenziós (azaz többmegkötéses) hátizsák problémaként[[3]](#footnote-3) definiálható.

Vassányi, Gaál és Kozmann munkájukban [16] sikeresen megterveztek és implementáltak egy automatizált menü generátort, melyet egy szív- és érrendszeri megbetegedések megakadályozását segítő online tanácsadó alkalmazásba integráltak. A MenuGene a heti menük megtervezéséhez genetikus algoritmust alkalmaz. A célértékeket egy online űralpon megadott adatok alapján, valamint általános táplálkozási irányelvek mentén dolgozták ki.

Nincsen egy jó menü generálásának általánosan elfogadott módja. Továbbá egy étrend (legyen az heti, napi, vagy csak egy étkezés) csak végleges állapotában értékelhető. Az értékelésnek legalább 2 szempontja van. Egyrészt figyelembe kell venni a tápanyagok mennyiségét. Egy személyre jól meghatározható tápanyag beviteli követelmények vonatkoznak (pl. fehérje, szénhidrát bevitel), melyek az illető kora, neme, tömege, munkájának típusa, betegségei alapján meghatározhatóak. Tehát a mennyiségre vonatkozóan az étrend megalkotása egy követelmény kielégítési és optimalizációs problémaként definiálható.

Másrészt az ételek összetevőinek harmóniáját is figyelembe kell venni. Az étrendeknek mindamellett, hogy kielégítik a tápanyag követelményeket, étvágygerjesztőnek is kell lenniük. Az étkezések fogásainak illenie kell egymáshoz. A józan észnek megfelelően bizonyos fogások, tápanyagok nem illenek egymáshoz. Ezen mindennapos íz és főzésmódbeli megkötések egyszerű szabályokként megfogalmazhatóak. Ezen szabályok leírják, hogy mely komponensek illenek össze. Természetes módon a számbeli, valamint a harmóniát érintő követelmények között konfliktus keletkezhet. Egy tanulmány szerint szakértők által generált étrendek sem elégítik ki a tápanyag követelményeket. [3]

A tanulmányban meghatározzák a szív-, és érrendszeri megbetegedések tényezőit, melyeket a személyes paraméterek mellett figyelembe vesznek a célfüggvény meghatározásánál. A napi tápanyag szükségleteket egy DRI[[4]](#footnote-4)-hez hasonló táblázatból olvassa ki a program.

A kutatásban használt adatbázis egy magyar étkezési szokásoknak megfelelő étkezési adatbázis, mely 569 fogás receptjét, azok tápanyagösszetételét tartalmazzák. Az adatbázis a tápanyagok kategóriáit is tartalmazzák(gabona, zöldség, gyümölcs, tejtermék, hús, tojás, zsír, édesség). Ezt az osztályozást a validáció során alkalmazzák, annak megállapítására, hogy a teljes kompozíció (az összetevők arányának megfelelően) megfelel-e a tápanyag piramis irányelveinek.

## Genetikus algoritmus

Az általam vizsgált kutatások jelentős része alkalmazza a genetikus algoritmusok valamely fajtáját. Erre való tekintettel hadd mutassam be a genetikus algoritmusok általános jellemzőit a többcélú étrend probléma szempontjából.

A genetikus algoritmus [17] egy ismételt iterációkon keresztül működő keresési művelet, mely a következőn alapszik: elsőként produkál valamennyi potenciális megoldást (populáció inicializálása) egy értékelési módszert használva, mely megméri, hogy a megoldás mennyire szolgál egy bizonyos célt (fitnesz értékelése). Ezt követően a megoldások csoportjait szétválasztja bizonyos műveleteket használva (szelekció, rekombináció, mutáció). A művelet addig ismétlődik, amíg a teljes populáció nem konvergál, vagy a maximális iterációs számot el nem érjük. A genetikus algoritmus alkalmazásához meg kell határozni a megoldások reprezentációját, a fitnesz értékelő függvényt, a populáció méretét, a szelekciós függvényt, a generikus műveleteket, és a populáció inicializáló függvényt. Mivel az MKP értéke 0 vagy 1 lehet, ésszerű minden változón végig haladni és egy bit sztringként ábrázolni őket. Az MKP egy maximalizációs probléma, tehát minél nagyobb a cél függvény értéke, annál jobb. Kivéve azon eseteket, amikor a megoldás nem létezik. A fitnesz értékelő függvény a célfüggvény maga. Viszont a nem megoldható problémát érdemes úgy kezelni, hogy vagy büntetést, vagy egy javító algoritmust alkalmazunk. Az MKP megoldható szabványos genetikus algoritmussal, viszont a probléma specifikus heurisztika használata nagy mértékben megnöveli a hatékonyságot.

Számos genetikus algoritmus található a szakirodalomban, mely heurisztikával alkalmazza azt MKP megoldására. Chu és Beasley [18] genetikus algoritmusa kiválóbb megoldást biztosít más heurisztikákhoz viszonyítva. A genetikus algoritmusuk egy stabil állapotú GA, mely bináris sztring kódolást alkalmaz, populáció nagysága 100, bináris verseny kiválasztást, egységes keresztezést, és bit mutációt alkalmaz. Nem enged egyed ismétlődést a populációban. Továbbá egy MKP-specifikus inicializáló függvényt alkalmaznak, mely a pseudo-hasznosság elvet alkalmazza. Az inicializáló módszerükben elsőként az elemek egy random permutációja jön létre. Ezt követően minden elem értékét 1-re állítják a permutáció sorrendjében, egészen addig, amíg nem sért megkötés. Egy javító módszer segítségével a nem megvalósítható megoldás megfelelővé transzformálható. Az egy megkötésű hátizsák probléma esetén a j elem pseudo-hasznossága [19] () vagy . Minnél nagyobb az , annál nagyobb a valószínűsége, hogy az elem szerepelni fog a megoldásban. Azonban az MKP esetén több megkötés van, ezáltal nincs egyértelmű meghatározása a pseudo-hasznosságnak. Több módja van az MKP pseudo-hasznossági arányok kiszámításának. Chu és Beasley Pirkul [20] helyettesítő kettősség megközelítését alkalmazták, az egyes megkötések árnyékárait az MKP lineáris programozás[[5]](#footnote-5) relaxációban helyettes szorzóként alkalmazva. Tehát elsőként megoldják a MKP LP relaxációját, majd megtalálják a helyettes kettős szorzókat és kiszámolják az egyes elemek pseudo-hasznosság értékét. Chu és Beasley javító módszerükben először az elemek a nem megvalósítható megoldásból, értékük szerint növekvő sorrendben eltávolításra kerülnek addig, amíg egy megkötés sincs megszegve. Majd az elemek értékük szerint csökkenő sorrendben hozzáadásra kerülnek mindaddig, amíg nincs megkötés megszegés. Ezáltal a legalacsonyabb hasznossággal rendelkező elemek eltávolításra kerülnek, míg a legmagasabb hasznosságúak hozzáadásra kerülnek a megoldáshoz. Ez a javító algoritmus minden megoldást a megvalósíthatóság határán tart.

Raidl [21] javított genetikus algoritmusa nagyon hasonló Chu és Beasley megadásához, kivéve néhány különbséget az inicializációs, javító, és lokális optimalizációs függvényekben. A legfontosabb különbség, hogy értékekeit használja az MKP LP relaxációjában, mint rész-hasznosság arányok. Raidl javító és lokális optimalizációs módszeriben először n számú elem véletlenszerű permutációja kerül generálásra, majd az elemek azok rész-hasznosságuk alapján rendezésre kerülnek. Ezáltal az azonos rész-hasznossággal rendelkező elemek véletlenszerűen priorizálásra kerülnek. Raidl inicializáló függvénye mindemellett rész-hasznosság értékeket használ. Elsőként n számú elem véletlenszerű permutációja kerül generálásra. Majd minden elem számára egy véletlenszerű érték generálásra kerül a [0,1] intervallumból. Ha a random szám kisebb, mint az elem döntési változója az LP relaxációban, az elem hozzáadásra kerül, hacsak nem sértünk vele megkötést. Ezáltal nagyobb a valószínűsége, hogy a nagyobb értékkel rendelkező elem az LP relaxációban bekerül a kezdeti populációba. Viszont az kezdeti populáció sokszínű lett a véletlenszerűségnek köszönhetően. Chu és Beasley genetikus algoritmusa és Raidl genetikus algoritmusa is a keresést a megvalósíthatóság határán tartják. A teszteredmények alapján Raidl genetikus algoritmusa kevéssel jobban teljesít Chu és Beasley genetikus algoritmusához viszonyítva.

A többcélú optimalizációban a cél egy olyan megoldás megtalálása, melyben a célkitűzések vektora a legmegfelelőbb a döntéshozó számára. Számos megközelítés létezik a többcélú optimalizációra, mint az aggregált módok, nem pareto optimumon alapuló módok, és a pareto optimumon alapuló módok. Van olyan módszer, mely valamely kritériumot előnyben részesíti másikkal szemben, van amelyik mindegyik célt el kívánja érni, van amelyik kompromisszumra alapszik a kritériumok között [23]. Egy többcélú optimalizációs megoldást nem-domináltnak vagy nem-alárendeltnek nevezünk, ha nem létezik más legalább egyenértékű vagy kiválóbb megoldás, minden célt figyelembe véve.

Ahogy már azt említettem, a többcélú optimalizációs problémákban a cél egy olyan megoldás megtalálása, melyben a döntési változókból létrehozott vektor - mely kielégíti a megkötéseket és optimalizálja a célkitűzések vektorát - a legmegfelelőbb a döntéshozó számára. Tehát a végső megoldás az optimalizációt és döntési folyamatot követően jön létre [23]. A döntéshozó preferenciájának megfelelően a többcélkitűzésű evolúciós algoritmuson[[6]](#footnote-6) alapuló többcélú probléma[[7]](#footnote-7) megoldások a következő kategóriákba sorolhatók Hwang és Masud (1979) alapján:

* Priori Preferencia Artikuláció: több célkitűzést kombinál, melynek eredményeképp a probléma egycélkitűzésű problémává válik.
* Progresszív Preferencia Artikuláció: ezen kategória interaktív megoldásokat tartalmaz, melyekben a döntéshozás és optimalizáció páhuzamosan zajlik, s így a megoldások egy frissített változata biztosított a döntéshozó számára.
* Poszteriori Preferencia Artikuláció: pareto optimális jelölt megoldások halmaza adott, s a döntéshozó kiválasztja a megfelelő megoldást a halmazból.

Kahraman és Seven munkájukban a súlyozott összeg megközelítést alkalmazzák, mely az aggregációs módszerek egyike, s emellett MOEA megoldási technikákat is alkalmaznak.

# Probléma definíció

Ebben a szekcióban ismertetem a probléma megértéséhez szükséges koncepciókat, továbbá bemutatom a menütervezési probléma modelljét. Feladatom egy egészséges, felnőtt egyén számára egy heti étrendet generálni, mely megfelel az ő napi tápanyag szükségletlének, személyes preferenciáinak, és nem utolsó sorban tálalható.

Feltételezem, hogy van n különböző fogásom, melyeket prezentálhatok a felhasználó számára. Ezen fogásoknak ismerem az egy adagra vonatkozó tápanyag tartalmát. Az egyes fogásokra vonatkozóan a következő makro- és mikrotápanyagtartalmakat ismerem: szénhidrát zsír, telített zsír, rost, fehérje, nátrium, cukor, továbbá ismerem az energiatartalmat is. A fogások a teljesség igénye nélkül következő kategóriákba sorolhatók: tengeri étel, kenyér, ital, szárnyas hús, vörös hús, disznóhús, saláta, előételek és snackek, desszertek, reggeli, tészta és galuska, zöldség, gyümölcs, búza, tejtermék, édesség. Egy fogás természetesen több kategóriába is besorolható. A felhasználó által megadott értékelések alapján tehát ismerem minden fogás értékelését, melyet az egyes ételek összetevőinek kategória szerinti aránya szerint számolok ki.

Következő fontos lépés meghatározni, hogy hogyan modellezzem az egy hétre vonatkozó étrendet. A felhasználó számára lehetőséget biztosítok, hogy meghatározza napi étkezéseinek, valamint az étkezések közötti snackek számát. Egy főétkezés a következő lehetséges elemekből épül fel: leves, főétel, desszert, ital. A snack valamilyen gyümölcs lehet (alma, narancs, stb.).

Ennek fejében két célfüggvényt kell meghatároznom. Mindemellett egy sor előre megadott megkötésnek kell megfelelnie a generált étrendnek. Az első célfüggvény feladata a tápanyagra vonatkozó hiba minimalizálása. Ez a függvény kiszámítja a tápanyagok összegének valamint az ajánlott tápanyagok összegének abszolút különbségét. Munkámban az NHS[[8]](#footnote-8) által javasolt DRI[[9]](#footnote-9) ajánlásokat veszem figyelembe [24]. Az energia-, makrótápanyag- (fehérje, szénhidrát, zsír) vitamin-, ásványianyagbevitelre vonatkozóan nemre, korra, fizikai aktivitásra vonatkozóan könnyedén kiolvashatóak az adatok.

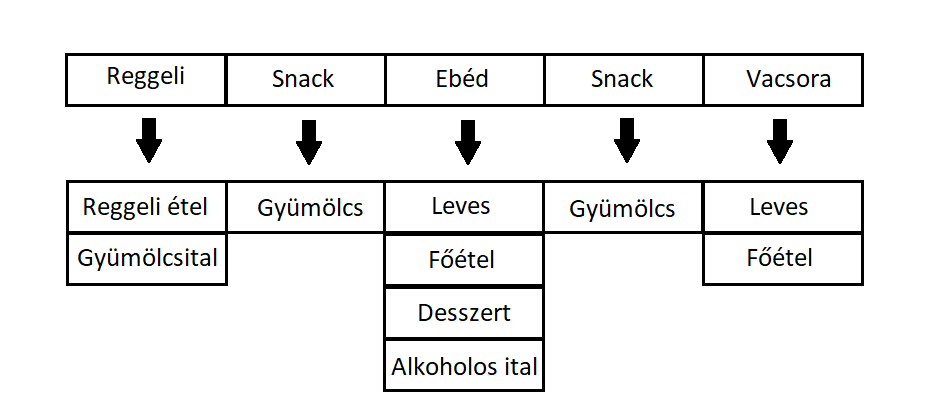
A másik célfüggvény feladata a felhasználói értékelés maximalizálása. Mivel egy-egy fogás több kategóriába is besorolható, ezen kategóriákra vonatkozó értékek összetevő szerinti súlyozott átlagát veszem figyelembe. Az így megkapott fogásonkénti átlagértékek összegének maximalizálására törekszik a függvény.

Mindemellett számos követelményt kell meghatároznom annak érdekében, hogy az ajánlott étkezések tálalhatóak és étvágygerjesztőek legyenek. Így figyelembe veszem a (i) változatosságot, (ii) konzisztenciát, (iii) számos találhatóság érdekében meghatározott szabályt.

* Tápanyag hiba korlát: a DRI által meghatározott ajánlott értékek – ahogy a nevében is van – ajánlások. Nem minden esetben fog a program minden tápanyag ajánlásnak megfelelő étrendet generálni. Ellenben a egy minimum határt megszabok, az adott DRI érték 70%-ban. Ha az étrend bármely tápanyag értéke nem éri el ezt a minimum korlátot, azt azonnal kiszűröm.
* Változatosság: Az étel táplálósága szempontjából elengedhetetlen annak sokfélesége. E célból ellenőrzöm egy fogás ismétlődését az étrendben. Ezen ismétlődéseket valamilyen módon büntetem az étrend generálás folyamán. Itt mind az adott napi ismétlődéseket, mind pedig a napon túli étel ismétlődéseket figyelembe kell vennem.
* Konzisztencia: az ételek textúrája annak minőségéről és elfogadhatóságáról győzi meg a fogyasztót. A konzisztenciát 2 csoportba sorolom: (i) folyadék/pépes és (ii) félkemény, kemény. Nem megengedett, hogy több, mint egy folyadék/pépes összetevője legyen egy ebédnek.
* Tálalhatósági és egyéb okokból meghatározott egyéb szabályok. Az egyes fogások további csoportokba vannak szedve. Ezen csoportokra szabályokat lehet meghatározni. Pl.: paradicsomos étel paradicsomlevessel → csökkentsd a fitnesz értéket a 60%-ra. Vagy magas cukortartalmú desszert → csökkentsd a fitnesz értéket az 50%-ra. A szabályok kiértékelése folyamán mindig a pontosabb szabályt fogom figyelembe venni.
* Egyes fogások száma. 1. ábrán bemutatott példa szerint megadott napi étrend struktúra alapján megkötésként meghatározom hány darabot kell generálni az egyes fogás típusokból. Tehát hány leves, főétel, stb. -re van szükség.

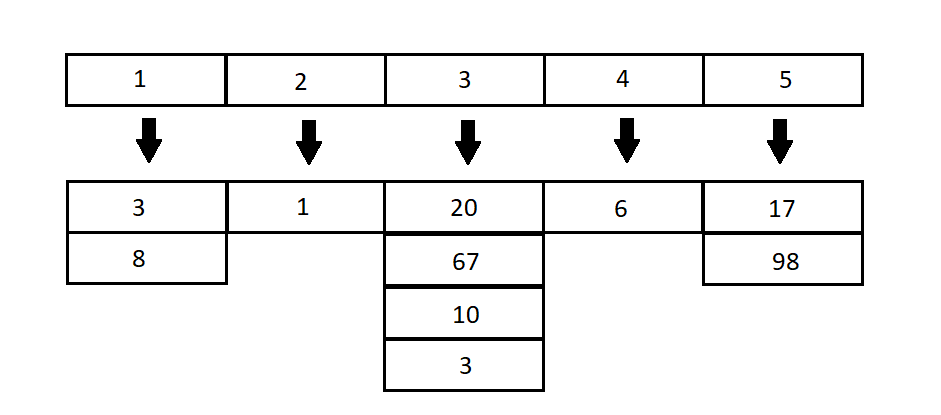
Annak ellenére, hogy a menütervezési probléma természetében kombinatorikai, nem kombinatorikai problémaként modellezem azt. A problémát kettős célú minimalizációs problémaként definiálok:

Itt a j fogás a döntési változója; a felhasználó által megadott értékelés alapján kiszámított étel értékelése, a referencia tápanyagmennyiségektől való eltérések összege; az i. megkötési érték; a j. fogás az i megkötésnek megfelelő tápanyagértéke.



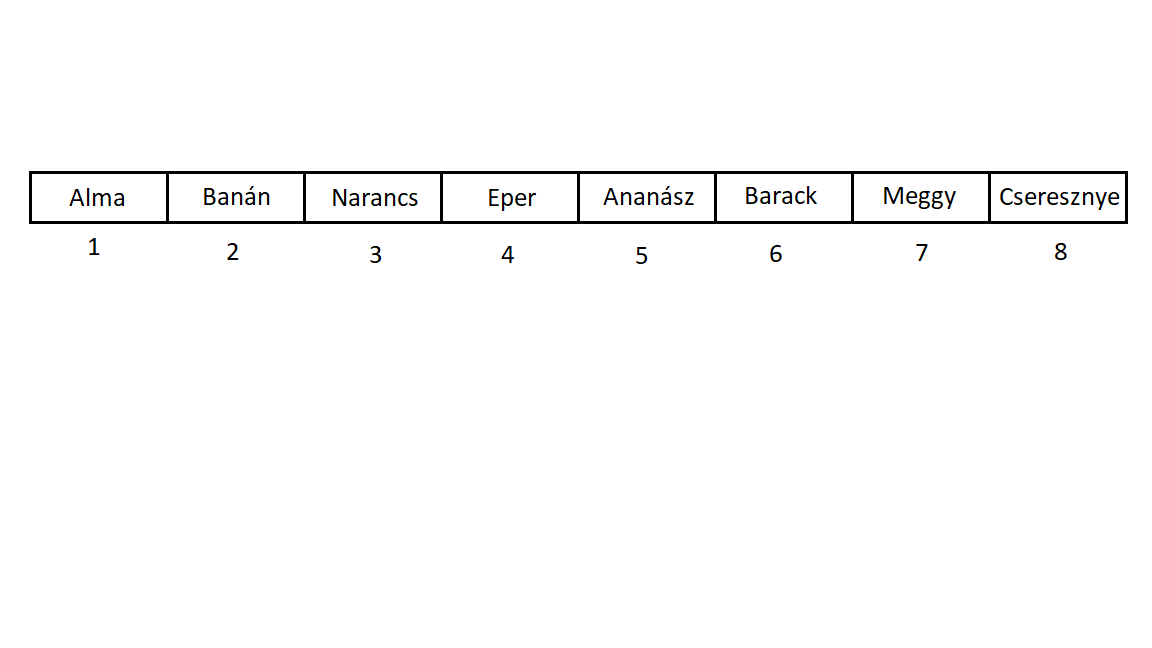
4.1 ábra, Napi menü lehetséges reprezentációja

Egy napi menü egyénenként komoly eltérést mutat, nem csak tartalmában, hanem struktúrájában is. Egy ilyen lehetséges felépítést mutat az 1. ábra. Ahogy azt az 2. ábra szemlélteti, az egyes étkezéseket külön listákként kezelem, melyeket egy egyedi egész szám értékkel reprezentálok. Az egyes étkezések alacsonyabb szintű elemekből épülnek föl.



4.2. ábra, Menü reprezentációja egész számokkal

Az értekezéseket fölépítő kategória elemek mindegyikéhez egy-egy lista tartozik. Egy ilyen listát találhatunk a 3. ábrán. A 2. ábra 2. étkezésének gyümölcs kategóriájú elemét egy ilyen listából választom ki. Jelen esetben 2. étkezés gyümölcs kategóriájú eleme az alma.



4.3. ábra, Gyümölcsök listája

# A megoldási módszer kiválasztása, a választás indoklása

A továbbiakban bemutatom a kiválasztott megoldási módszert.

Az étrend tervezési probléma megoldása során több, a fentiekben bemutatott megoldási módszert fogok ötvözni. Lineáris programozással együtt Genetikus algoritmust[[10]](#footnote-10) fogok használni, melynek általános jellemzőit 2.2 fejezetben bemutattam. Azért esett erre a GA módszerre a választásom, mert már számos hasonló kutatásban alkalmazták sikerrel [12] [14].

Kezdeti tesztként egy egyszerűbb genetikus algoritmust implementáltam online tananyagok segítségével, melyhez az eredeti adathalmazom töredéke méretű adathalmazt használtam. Azonban amint az algoritmust nagyobb adatmennyiségen futtattam (több ezer étel lehetőség), hamar megmutatkoztak a hatékonyságbeli hiányosságok. Egy genetikus algoritmust számos módon lehet javítani, köztük lokális optimalizációs technikával, vagy más heurisztikával. Vagy a GA végső eredményére alkalmazzuk ezen módszereket, vagy a GA által generált újabb és újabb populációkra alkalmazunk optimalizációs lépéseket. Egy másik kevésbé népszerű lehetőség a kezdeti populáció már előre optimalizált jelöltekből való felépítése. Ebben az esetben különösen kell ügyelni, nehogy túl hasonló megoldáshalmazokat állítsunk elő, megakadályozva a populáció diverzitását. Ilyenkor megnő annak az esélye, hogy a GA megakad egy lokális optimumban.

Az előző fejezetben általam definiált probléma megfeleltethető egy többdimenziós hátizsák problémának, melynek hatékony megoldására számos algoritmus született (lásd 2. fejezet). A Raidl által bemutatott módszert [21] fogom főképp felhasználni, mely a teszteredmények alapján kiemelkedően teljesített más hasonló heurisztikákhoz viszonyítva.

|  |  |
| --- | --- |
| Genetikus Algoritmus: | stabil állapotú,  egyed ismétlődés nem megengedett |
| Kódolás: | bit string |
| Kiválasztás: | verseny (k = 3) |
| Mutáció: | bitszerinti (pm = ~0.0337) |
| Rekombináció: | egyforma keresztezés[[11]](#footnote-11) (pr = ~0.698) |
| Populáció mérete: | 4527 |
| Terminálás: | fitnesz javulásának megállása esetén (43 iteráció) |
| További kiegészítési lehetőségek: | elő-optimalizált inicializáció,  javító függvény,  lokális javító függvény |

5.1. táblázat, GA jellemzői

A felhasznált genetikus algoritmust a fenti táblázat foglalja össze. A tervezés során ezen értékeket még nem ismertem, viszont az implementiációt követő további optimalizációs eljárás alkalmazásával (NSGA2 algoritmus) megtaláltam a paraméterek egy optimális halmazát. Az optimalizáció megkezdésekor már birtokában vagyok a napi étrend általános felépítésével. Itt értem: hány étkezést preferál a felhasználó, s ezen étkezések milyen alkategóriákból épüljenek fel. A kezdeti populáció generálásához az MKP probléma LP-relaxált alkalmazhatom Raidl munkája alapján [21]. Ezen megoldásjelöltek mind egy lehetséges napi étrendet reprezentálnak.

A napi étrendek felhasználó által preferált számú étkezésekből állnak, az étkezéseket pedig fogások alkotják. A genetikus algoritmus minden iterációjában egy sor evolúciós műveletet hajtok végre (keresztezés, mutáció, szelekció) az egyes elemeken. A hatékonyság javítása érdekében mindemellett javító és lokális javító1 függvényt tervezek alkalmazni. A javító függvény ugyancsak felhasználva az LP-relaxált MKP megoldást, ezen megoldások növekvő sorrendjében kapcsolják le a j. elem flagjét, amíg bármelyik megkötésbe ütközik a megoldás. A lokális javító függvény pedig ennek fordítottját alkalmazza, az LP-relaxált MKP megoldások csökkenő sorrendjében kapcsolja föl a j. elem flagjét, hacsak ezzel nem ütközünk bele valamelyik megkötésbe.

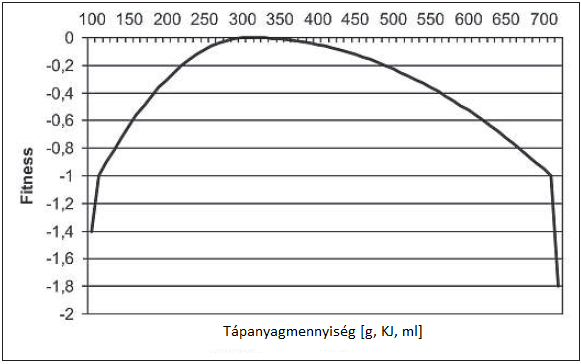
Az evolúciós folyamatot meghatározott számú iterációt követően vagy ha már nem történik szignifikáns javulás, megállítom. A végső populáció legjobb x elemét fogom megjeleníteni a felhasználónak, aki ezek közül személyes preferenciájának megfelelően választhat.

Tekintve, hogy a problémámban kettő célt fogalmaztam meg, másfajta fitnesz függvényt fogok használni. A súlyozott összeg módszerét fogom alkalmazni többcélú optimalizációhoz. Ennek megfelelően a célfüggvényeket egyetlen célfüggvénnyé transzformálom át, mely a célfüggvények lineáris függvénye [22]. A célfüggvényeknek ebből következően azonos típusúaknak kell lennie (azaz vagy minimalizáló, vagy maximalizáló). Ezzel szemben a problémámban az egyik célfüggvény maximalizáló, a másik pedig minimalizáló. Tehát a minimalizáló célfüggvényemet maximalizálóra transzformálom annak reciprok értékét véve. Minden célfüggvényt egy súly értékkel szorzok meg, melyeket így összeadok. A súly értékek összegének 1-nek kell lennie. Mindemellett a célfüggvények egymás fölötti dominanciáját megakadályozandó, normalizációt hajtok végre rajtuk. Így a problémám célfüggvényei és a végső fitnesz függvény a következő:

Szeretném bemutatni az evolúciós műveleteket. Egy tipikus magyar étkezés 4 összetevőből áll: leves, főétel, desszert, ital. A keresztezés (más néven rekombináció) folyamán 2 elemet használ fel. Egy véletlenszerű ponttól kezdődően ez elemek attribútumai kicserélődnek. Például, ha a kezdő pont a 2-es index, akkor a desszert és az ital attribútumokat cseréljük ki. Mutáció folyamán egy véletlenszerűen választott fogást cserélünk ki egy ugyanolyan kategóriájú másik fogásra (pl. egy főételt egy másik főétellel).

## Fitnesz

Egy új elem (utód) tehát kétféleképpen jöhet létre: keresztezés vagy mutáció által. A fitnesz függvénynek feladata, hogy megállapítsa az adott utód jóságát. A számszerű megkötéseket figyelembe véve, a fitnesz függvény feladata kiszűrni a nem megfelelő mennyiségű tápanyaggal rendelkező elemeket. A tápanyag hiba értékhez tartozó fitnesz érték meghatározásához a [16] kutatásban alkalmazott fitnesz függvény fogom alapul venni. Egy megoldásjelölt fitnesz értéke n (= a megkötések száma) darab négyzetes függvény összege, melyek maximum értéküket (0) a meghatározott optimumnál veszik föl. A függvény a meghatározott felső és alsó korlátoknál élesen letörik (lásd. 4. ábra). Ez egy büntetés-alapú fitnesz függvény, melyet gyakran alkalmaznak más többcélú optimalizációs technikákban [18]. A büntető függvény tervezésénél fogva a kis eltéréseket az optimumtól nem bünteti, de a meghatározott intervallumon kívül eső értékekkel szigorú.



5.. ábra Fitnesz függvény példa, Optimális érték: 300; Minimum érték: 100; Megengedett legmagasabb érték: 700

A fitnesz függvénybe a másik büntető függvény beépítésével tovább módosítom a fitnesz értéket. Ez a függvény azt vizsgálja, hogy mekkora az abszolút eltérés a potenciális megoldás összértékelése és a referencia értékelés összeg között. Ezen százalékos eltérés arányában fogom rontani a fitnesz értétkét.

Miután megállapítottam a az egyes elemek jóságát a számszerű megkötések és a felhasználói értékelés tekintetében, terveim szerint megvizsgálom a megoldás jelöltek további megkötéseknek való megfelelését. Ezen szabályok az ételek harmóniáját szolgálják.

Minden szabály két részből állhat össze, egy feltételből, egy feltételből és egy fitnesz módosító arányból. Az egyes fogások csoportokba vannak szedve. Ezen csoportokra szabályokat lehet meghatározni. Pl.: paradicsomos étel paradicsomlevessel → csökkentsd a fitnesz értéket a 60%-ra. Vagy magas cukortartalmú desszert → csökkentsd a fitnesz értéket az 50%-ra. A szabályok kiértékelése folyamán mindig a pontosabb szabályt fogom figyelembe venni.

# Technológiai ismertetés

A platform kiválasztása nem egyszerű feladat, hiszen többféle megközelítés is létezik. Kiemelten fontosnak tartom, hogy mobilon is teljes értékű felhasználói élményt nyújtson az alkalmazás. Fontos megvizsgálni annak a lehetőségét, hogy egy natív Androidos alkalmazást készítsek. Mindemellett nem ez az egyetlen járható út. Mivel célom a lehető legszélesebb felhasználó kör elérése, elengedhetetlennek tartom, hogy az alkalmazás böngészőből is elérhető legyen. A következőkben a lehetséges fejlesztői platformokat ismertetem. Elsőként az Android rendszer sajátosságaira térek ki.

## Android

Az Android egy mobil operációs rendszer a Linux kernel egy módosított verziójára és más szabad forráskódú szoftverre alapozva. Elsősorban érintőképernyős eszközökre tervezték, mint okostelefonok, tabletek.

Android applikációt Kotlin, Java, illetve C++ nyelveken lehet írni. Az Android SDK[[12]](#footnote-12) segítségével tudjuk a forráskódot, más adatokkal és forrásokkal együtt lefordítani egy APK[[13]](#footnote-13)-ba. A .apk állomány tartalmazza az összes telepítéshez szükséges információt. Minden Android applikáció egy saját biztonsági homokozóban üzemel, mely a következő biztonsági funkciókkal rendelkezik:

* Az Android operációs rendszer egy többfelhasználós Linux rendszer, melyben minden alkalmazás egy külön felhasználó.
* Alapesetben, a rendszer minden alkalmazáshoz hozzárendel egy egyedi Linux felhasználói azonosítót (melyet csak a rendszer használ, és ismeretlen az app számára). A rendszer beállítja az összes fájlhoz a hozzáférési jogosultságokat az appban, melynek köszönhetően csak a megfelelő felhasználói azonosítóval tud hozzáférni.
* Minden folyamatnak egy saját virtuális gépe van, s emiatt az app kódja izolálva fut a többi apptól.
* Alapesetben minden app a saját Linux folyamatát futtatja. Az Android rendszer akkor indítja a folyamatot, amikor az app bármelyik komponensét végre kell hajtani. A folyamatot pedig akkor állítja le, amikor arra már nincs szükség, vagy a rendszernek forrást kell felszabadítani memória szűke miatt.

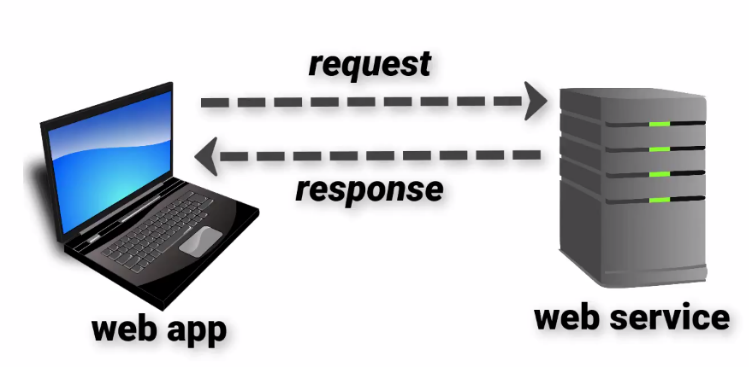
Az Android rendszer a minimális jogosultság elvén működik. Ennek értelmében minden alkalmazás csak a működéséhez szükséges komponensekhez fér hozzá. Mindennek köszönhetően egy nagyon biztonságos környezet jön létre, melyben egy applikáció nem férhet hozzá azokhoz a komponensekhez, melyekhez nincs engedélye. Ennek ellenére van lehetőség arra, hogy egy applikáció adatokat osszon meg más alkalmazással és rendszer szolgáltatásokhoz férjen hozzá.

Nem lehet elmenni szó nélkül a webalkalmazás lehetősége mellett. Manapság rengeteg cég mellőzi a natív Android alkalmazás kifejlesztését, s inkább egy reszponzív webalkalmazást készít, mely optimalizálható később mobilos felületre is. Továbbá, ha a backend oldali logikát REST API segítségével hozzuk létre, akár a későbbiekben van lehetőség azt felhasználni egy Android alkalmazás front-endjének kiszolgálásához.

## Mi is az a webapplikáció?

Számítástechnikában webalkalmazásnak nevezzük azt a kliens-szerver programot melyet a kliens (beleértve a felhasználói felületet és a kliens oldali logikát) futtat saját webböngészőjében. Példaként említhető a webmail, online kiskereskedelmi oldalak, online bankolás és online aukciók. A technológia által nyújtott talán legfontosabb előny, hogy jelentősen csökkenti a szerver és a kliens közötti kommunikáció mértékét hagyományos weboldalakhoz viszonyítva. A szerverünket érő terhelés csökkentésével jelentősen nagyobb mennyiségű felhasználót tudunk kiszolgálni valamint egy sokkal gördülékenyebb felhasználói élményt érhetünk el.

Egy hagyományos webapplikáció szerkezetét tekintve alapvetően 2 részre osztható. A frontend – melyhez jellemzően HTML, CSS, és JavaScript használatos – biztosítja a kliens oldali működést. Ehhez a felhasználó egy böngészőt használ, mely alkalmas az applikáció futtatására. A webapplikációt futtató környezet azonban jellemzően nem rendelkezik elegendő számítási teljesítménnyel, memóriával ahhoz, hogy egy komolyabb modellt képes legyen futtatni. Itt jön képbe a back-end webszolgáltatás, mely a modell logikáját biztosítja. A webszolgáltatás egy API-ként működik, melynek bármilyen applikációból küldhetünk HTTP kérést.



6.1. ábra, Webalkalmazás felépítés egyszerű szemléltetése

## Web Architektúra

1993-ban Roy Fielding és társai kételyüket fejezték ki a web skálázhatóságát illetően. Ezért kidolgozták a Web architektúrai stílusát, melyet 6 fő pontban lehet összefoglalni [27]:

1. Kliens-szerver
2. Egységes interfész
3. Rétegelt rendszer
4. Cache
5. Állapotmentes
6. Code-on-demand

### Kliens-szerver

A felelősségek szétválasztása a fő elemét képezi a Web kliens-szerver megkötözésének. A Web egy kliens-szerver alapú rendszer, melyben mindkettő különálló szerepet játszik. Implementációjuk függetlenül is megtörténhet bármilyen technológia vagy nyelv használatával egészen addig, amíg megfelelnek a Web egységes interfész követelményének.

### Egységes interfész

A Web komponenseinek közötti interakció – ezalatt értve a klienseket, szervereket, és közöttük lévő hálózati közvetítőket- az interfész egységességétől függenek. Ha bármely komponens nem teljesíti a sztenderdeket, a Web kommunikációja összeomlik.

A Web komponensei konzisztensen együttműködnek az egységes interfész négy követelménye mentén, melyek:

* A források azonosítása. Minden egyedi Web alapú koncepció forrásként ismeretes, melyet egy egyedi azonosító segítségével azonosíthatunk, mint például az URI.
* A források manipulációja a reprezentáción keresztül. A kliens manipulálja a források reprezentációját. Ugyanaz a forrás ábrázolható különböző kliensnek különböző módon. Például egy dokumentum reprezentálható, mint HTML, egy webböngésző számára, és mint JSON egy program számára. A fő gondolat a következő: a reprezentáció egy módja az interakciónak a forrással, de az nem a forrás maga. Ez az elvi megkülönböztetésnek köszönhetően a forrást különböző módokon és formátumokban reprezentálhatjuk, mindezt az azonosító megváltoztatása nélkül.
* Önmagukat magyarázó üzenetek. A forrás kívánt állapota leírható a kliens kérés üzenetében. Egy forrás jelenlegi állapota reprezentálható a válasz üzenetben, ami a szervertől érkezik vissza.
* Hipermédia az applikáció állapotának motorja. A forrás állapot reprezentációja hozzá kapcsolódó forrásokhoz tartalmaz linket. A linkek szálakként fonják össze az internetet azáltal, hogy a felhasználók értelmes és irányított módon járják keresztül információkat és alkalmazásokat.

### Rétegelt rendszer

A rétegelt rendszer megkötésnek köszönhetően megjelenhetnek a hálózat-alapú közvetítő, mint például a proxyk és átjárók. Ezek transzparens módon telepíthetőek a kliens és szerver között a Web egységes interfészét használva. A közvetítők jellemzően specifikus céllal jelennek meg, mint biztonság, válasz gyorsítótár, terheléskiegyenlítés.

### Gyorsítótár

A gyorsítótár a web architektúra legfontosabb követelménye, mely megköveteli, hogy a szerver deklarálja a válasz adat cachelehetőségét. A gyorsítótár használatnak köszönhetően csökkenthető a kliens által érzékelt késleltetés, növelheti az általános rendelkezésre állását és megbízhatóságát az alkalmazásnak és kontrollálja webszerver terhelését. Egy szóban a gyorsítótár olcsóbbá teszi a webet. A hálózati útvonal bármely pontján megjelenhet szerver és a kliens között.

### Állapotmentes

Az állapotmentességi követelmény szerint a szervernek nem feladata a kliens applikáció állapotát eltárolni. Ennek következtében minden egyes interakció esetén a kliensen kötelessége biztosítani az általa fontosnak tartott kontextus információt. A web szerver kéri a klienst, hogy kezelje a az applikáció állapot komplex kommunikációját. Ennek köszönhetően a szerver sokkal nagyobb számú klienst képes kiszolgálni.

### Code-on-demand

A web erősen kihasználja a kód-kérésre követelményt, melynek segítségével a webszerver átmenetileg átküldhet végrehajtható programokat, például szkripteket, plug-inokat a kliens részére. Ez az egyetlen követelmény, mely opcionális, hiszen egy technológiai függőséget követel a webszerver és a kliens között. A kliensnek tudnia kell értelmezni és végrehajtani a kódot, melyet a szervertől letölt. Webböngésző által hosztolt technológiákra példa: Java appletek, JavaScript, Flash.

## Webalkalmazás – Gyakorlati Megvalósítás

Egy fejlesztő számára ma rengeteg technológia adott egy webalkalmazás fejlesztésére. A tisztaság kedvéért egy webalkalmazás elkészítéséhez szükséges technológiákat 3 fő csoportba sorolandóik:

1. Backend nyelvek (pl.: Python, Ruby, C#, Java) – a webalkalmazás irányításáért felel
2. Web frontend (pl: HTML, CSS, JavaScript) – a kliens oldali logikáért, megjelenítésért felel
3. DevOps (pl.: Github, Jenkins, Azure) – az alkalmazás telepítéséért/ hosztingolásáért felel

A felhasználható platformok száma jelentős. Szükségszerű meghatározni hogy mely keretrendszert kívánom használni alkalmazásom fejlesztéséhez. A keretrendszerek egy támogató felületként szolgálnak, s képesek kiegészíteni a programozó által tervezett funkciókat. Érdemes azt is figyelembe venni, hogy milyen programozási nyelvben van előképzettsége a programozónak. A teljesség igénye nélkül tekintsünk át néhány keretrendszert, melyek manapság népszerűek.

## Ruby on Rails

A Ruby on Rails egy nyílt forráskódú, server-oldali web alkalmazás keretrendszer. Mindemellett a Ruby egy Modell-Nézet-Vezérlő[[14]](#footnote-14) keretrendszer. Alapértelmezett struktúrát biztosít az adatbázis, webszolgáltatás, weboldalak számára. Hosszú sikerre tekint vissza, a következő oldalak mögött található meg: Basecamp, GitHubt, AirBnb, Twicht és a SoundCloud. A Ruby-nak mindemellett egy nagyon segítő fejlesztői közössége van, mely kiemelkedően fontos egy kezdő fejlesztő számára.

## Codeigniter

Codeigniter egy robosztus PHP keretrendszer, melynek használata könnyű, s széleskörű funkciókínálattal rendelkezik webfejlesztés területén. Mindemellett rendkívül könnyű, maga a keretrendszer kiemelkedő minőségű dokumentációval rendelkezik, melyet igen egyszerű értelmezni. Egy komoly könyvtárkollekcióval rendelkezik, melynek köszönhetően könnyű megtanulni, adaptálni és telepíteni. A Codeigniter, mely egy MVC-alapú keretrendszer a rugalmasságáról és menedzselhetőségétől ismert. Codigniter megengedi, hogy saját szkripteket használjunk a saját magkönyvtáraink mellett. A keretrendszer komplex struktúráktól és fárasztó fejlesztői folyamatoktól mentes. Kezdő PHP fejlesztők számára a Codeigniter egy jó választás lehet.

## Express.js

Az Express.js egy előre megírt Node.js keretrendszer, melynek segítségével szerver-oldali alkalmazásokat gyorsabban, hatékonyabban, több rugalmassággal és skálázhatósággal fejleszthetünk. A keretrendszer a Node.js-nek megfelelő teljesítményt nyújt, valamint funkciók egy széles halmazát biztosítja mind web- és mobilalkalmazások területén. Segítségével készíthetünk egyoldalas, többoldalas vagy hibrid alkalmazásokat. Az Express JavaScriptben íródott, melynek köszönhetően könnyű benne fejleszteni. 5 éves múltra visszatekintve, érett platformnak mondható. Ez a keretrendszer főleg haladó programozók számára ajánlott.

## ASP.NET

Az ASP.NET egy Microsoft által készített, nyílt forráskódú keretrendszer modern webalkalmazások építéséhez .NET keretrendszerben. Az ASP.NET platformfüggetlen, mely fut Windows-on, Linux-on, macOS-en és Dockerben is.

Tekintve az ASP. NET Core számtalan előnyét, mint a platformfüggetlenség, megbízhatóság, támogató fejlesztői közösség, ezt a keretrendszert fogom választani webalkalmazásom fejlesztésére. Továbbá nem hagyható figyelmen kívül a keretrendszer teljesítménye, mely más keretrendszerekkel összehasonlítva az élmezőnyben teljesít (10. helyezett a Web Framework Benchmark tesztjében [28]). Mindemellett a későbbi telepítés szempontjából a rendszer integrációjának köszönhetően a Microsoft Azure felhővel könnyedén telepíthető az alkalmazás.

### Mi az a .NET platform?

A .NET egy különböző eszközökből, programozási nyelvekből és könyvtárakból álló fejlesztői felület, mely lehetőséget ad applikációk széles skáláját fejleszteni. Az alap platform olyan komponenseket biztosít, melyek alkalmazhatóak bármilyen alkalmazásban. További keretrendszerek, mint az ASP.NET, kiszélesítik a .NET komponenseket annak érdekében, hogy speciális célokra építsünk alkalmazásokat. A .NET platform magában foglalja a következőket:

* A C# programozási nyelv és annak fordítója
* Alap könyvtárak (például: string, dátum, file I/O kezeléshez)
* Szerkesztők és más eszközök Windows, Linux, macOS és Docker platformon

Az ASP.NET kiterjeszti a .NET platformot webalkalmazások fejlesztéséhez használható eszközökkel és könyvtárakkal. Az ASP.NET a következőkkel egészíti ki a .NET platformot:

* Alap keretrendszer web kérések feldolgozásához
* Weboldal sablonozó szintaxis. Razor, dinamikus weboldalak készítéséhez.
* Könyvtárak a népszerű web mintákhoz (pl.: MVC)
* Hitelesítési rendszer, mely magában foglal könyvtárakat, adatbázist, és minta oldalakat a bejelentkezések kezelésére. Beleértve a többlépcsős hitelesítést, valamint a külső hitelesítést (pl.: Google vagy Twitter).
* Szerkesztő kiegészítések, mely szintaxis kiemelést, kódkiegészítést és egyéb funkciókat biztosítanak weboldalak fejlesztéséhez.

### ASP.NET Core

ASP.NET Core egy nyílt forráskódú és platformfüggetlen verziója az ASP.NET-nek. A Microsoft ajánlása szerint minden új fejlesztést ASP.NET Core-ban indokolt elkészíteni. A kizártan Windows-t támogató ASP.NET verziókra ASP.NET-ként hivatkozunk. A fejlesztések nagy része ASP.NET Core-ban történik, viszont a Microsoft előreláthatólag még hosszú ideig támogatni fogja ezen verziókat is [29].

## MVC tervezési minta és továbbfejlesztése

Az MVC az egyik legismertebb és legnépszerűbb tervezési minta, mely a modell, nézet, kontroller szavak rövidítéséből épül fel. Elmozdul a másik népszerű mintától, a Page-Controllertől. Célja a vonatkozások szétválasztása. Segítségével projektünket átláthatóbbá tudjuk tenni. Az ASP.NET MVC eszközök egy széles (és növekvő) tárházával rendelkezik, melynek köszönhetően modern webapplikációkat készíthetünk. Tekintsük át az MVC tervezési mintát.

### Modell

A modell képviseli azokat az adatokat, melyek segítségével a felhasználó számára biztosítani tudjuk a funkcionalitást. A felhasználók egy adat vezérelt oldalon navigálnak, s a modell az adatot képviseli. A modellt fölhasználva jelenítjük meg a nézetet. A modell nem az entitás, amit az adatbázisba maradandóan tároljuk. Ez egy nagyon fontos szempont, hiszen ha ellenkezőleg készítjük a modellt, biztonsági kockázatnak tehetjük ki alkalmazásunkat. Egy rosszindulatú felhasználó képes lehet hozzáférni olyan adattagokhoz, melyeket eredetileg nem szántunk a külvilág számára. Pontosan ily módon jutott adminisztrátori jogosultsághoz Egor Homakov a Ruby on Rails repositoryhoz GitHubon 2012-ben [30].

Legtöbb példában a nézet modellekre az entitásokat közvetlenül használják, mint modell a nézet számára. Azonban ez a hozzáállás nem segíti elő a szoftverfejlesztés más aspektusait, mint például a tesztelés vagy a vonatkozások szétválasztását a kontrollereinkben. Az entitás közvetlen használata a nézetben azt jelenti, hogy egy nem kívánt mértékű csatolást értünk el az adatbázistól egészen a nézetig. A modell az, amire szükségünk van ahhoz, hogy az oldalt megjelenítsük miután végrehajtottuk az üzleti logikát és (adott esetben) egy kilapított denormalizált rekordot az adatbázis több táblájából.

### Nézet

A nézetek ASP.NET Core MVC-ben olyan fájlok, melyekben a modell bizonyos részeit összeillesztünk megfelelő HTML kóddal annak érdekében, hogy a felhasználó számára biztosítsuk a megfelelő felületet.

Razor nézetmotor segítségével könnyedén válthatunk az irányítási folyamat vagy modell és szolgáltatások eléréséhez szükséges szintaxis, valamint a HTML leíró szintaxis között. Razor segítségével soron belül tudunk C# kódot használni, amit kifejezetten okosan tud értelmezni. Egy egyszerű @ jellel tudjuk jelezni a fordító számára, hogy C# kódra váltottunk. Továbbá mivel hegyes zárójel nem használható C# kifejezés kezdetére, tudja hogy mikor váltottunk vissza HTML kódra.

### Részleges Nézet

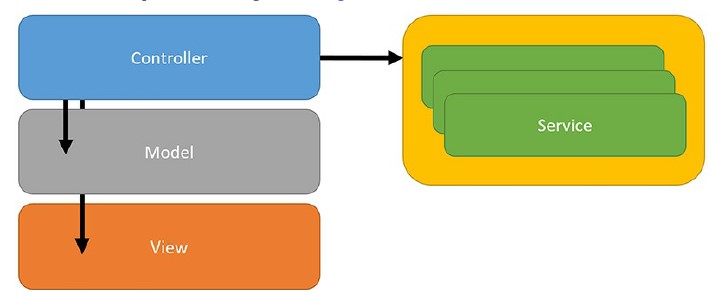
Eszköztárak, azonosító felületek, vásárlási kosarak, irányítópult bizonyos részei, és más hasonló komponensek az applikációkban sok esetben több vagy akár az összes oldalon is megjelennek. A DRY[[15]](#footnote-15) elvet figyelembe véve lehetőség van részleges nézetek létrehozására. Ezeket így több oldalon is fölhasználhatjuk. A részleges nézetek jellemzően nem önmagukban rendereljük, hanem más nézetekkel együtt. Lehetőség van létrehozni bejelentkezési állapotot jelző részleges nézetet. Továbbá eszköztárak, vásárlói kosarak is létrejöhetnek itt. Gyermek tevekénységek korábbi MVC Framework verziókban aszinkron módon voltak renderelve. Mostanra viszont aszinkron renderelésre is van lehetőség a teljesítmény növelése érdekében. Viszont még mielőtt a felhasználók megkaphatják a nézet kimenetét, valamint ahhoz, hogy betölthessünk bármilyen modellt a nézet motorba, szót kell említenünk a kontrollerekről.

### Kontroller

A kontrollerek az MVC applikáció forgalomirányítói, melyek biztosítják, hogy a megfelelő bitek a megfelelő helyekre utaznak. Kontrollerek jellemzően a Controller alaposztályból öröklődnek, továbbá ajánlott a konvenciónak megfelelően az osztály nevét „Controller” végződéssel ellátni(pl.: OrdersController).

Az alapértelmezett konvenció szerint a kontrollereinket a „Controllers” mappába kell helyezni a projekt gyökerébe. Azonban ez már nem követelmény Core MVC-ben, mert egy assembly felderítés történik az elnevezési és öröklődési konvenciók alapján. Viszont továbbra is ajánlott így eljárni az átláthatóság érdekében.

Szoftverfejlesztőként a kontrollereket konténerként használjuk, melyek kezelik a bejövő kéréseket. Ezek a kezelőket[[16]](#footnote-16) metódusként implementáljuk a kontroller osztályunkban. Minden metódus nulla vagy több paramétert fogadhat el, melyek automatikusan (amennyiben jelen vannak) kitöltésre kerülnek a modell kötő lépésben a végrehajtási szállítórendszerben.

Néha szükség van a modell osztály egy példányát létrehozni, lekérni, vagy egy megfelelő HTTP státusz kódon alapuló választ legyártani. Érdemes elkerülni minden jellegű üzleti logikát a kontrollerben, melyért a modell vagy más komponensek felelnek. Mindemellett ajánlott az adathozzáférést vagy külső hívásokat kihagyni az kezelőkből, melyek az applikáció szolgáltatások része kell legyenek [29]. 

6.2. ábra, MVC tervezési minta felépítése

### Köztesréteg[[17]](#footnote-17)

6.3 ábra, Middleware működése

A köztesréteg egy olyan szoftver, mely az applikáció szállítórendszerébe van beillesztve annak érdekében, hogy a kéréseket és a válaszokat kezelje. Minden komponens eldöntheti, hogy

1. Tovább küldi a kérést a következő komponensnek a szállítórendszerben, vagy
2. feladatokat hajt végre a következő komponens előtt és után a szállítórendszerben.

Kérés delegáltakat arra használjuk, hogy kérés szállítórendszert építsünk. A kérés delegáltak kezelik az összes HTTP kérést.

A kérés delegáltak a Run, Map, Use kiegészítő metódusok által konfigurálhatóik. Egy egyéni kérés delegált névtelen metódusként meghatározható soron belül, vagy egy újrafelhasználható osztályban is definiálható. Ezeket az újrahasznosítható osztályokat és soron belüli névtelen függvényeket nevezzük köztesrétegnek vagy köztesréteg komponenseknek. Minden köztesréteg komponens a kérés szállítórendszerben a következő komponens meghívásáért felel, vagy lehetősége van a szállítórendszer rövidzárjára. Terminális köztesrétegnek nevezzük, amikor a köztesréteg rövidzárásra kerül, mert ez megakadályozza további köztesrétegeket a kérések feldolgozásától.

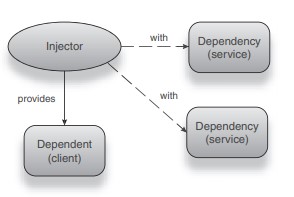
Az ASP.NET Core kérés szállítórendszer kérés delegáltak sorozatából áll, melyek egymás után kerülnek meghívásra. A 6.3 diagramm demonstrálja az elképzelést. A végrehajtás a fekete nyilak mentén folyik.

Minden delegált végrehajthat műveleteket a következő delegált előtt és után. A hibakezelő delegáltak meghívása hamar kell, hogy történjen a szállítórendszerben annak érdekében, hogy el tudják kapni szállítórendszerben később keletkező kivételeket.

### Függőség beinjektálása

A függőség injektálás [30] egy objektum orientált tervezési minta melynek segítségével lazán csatolt kódot tudunk fejleszteni. A DI segítségével megszabadulhatunk a szorosan csatolt szoftver komponensektől. A tervezési minta célja, hogy könnyen karbantartható és továbbfejleszthető kódot hozzunk létre. A technika segítségével a függőséget vagy függőségeket azok felhasználási helyén kívül hozzuk létre. A függőségeket beinjektáljuk onnan, ahol a osztályt hívjuk. Ezáltal minden információt a függőségek létrehozásával kapcsolatban elrejtünk. Ezért mondhatjuk azt, hogy a függőség injektálás egyúttal a kontroll megfordítása[[18]](#footnote-18) elvet is követi. Annak érdekében, hogy megérthessük a függőség injektálás elvet, a kontroll megfordítása elvet is meg kell ismerni.

### Kontroll megfordítása

Normál esetben a program végrehajtásának logikáját az egymáshoz kötött objektumok határozzák meg. A kontroll megfordítása esetén a végrehajtás a definiált absztrakciókon múlik, melyeket implementálunk. Az elvet jól leírja a Hollywood elv, mely a következő: „Ne hívj minket, majd mi hívunk téged!” Pont ahogy a hollywoodi ügynökök használják ezt az elvet, hogy meghallgatásokat szervezzenek a színészeknek. Ugyanúgy a DI könyvtárak használják ezt az elvet annak érdekében, hogy az objektumoknak biztosítsák a szükséges függőségeket. Az injektálást egy külső keretrendszer végzi el, melyet függőség injektáló keretrendszernek nevezünk. Ennek feladata a függőségek létrehozása, összeállítása és behuzalozása a objektum gráfba. [33]

6.4 ábra, IOC működése

## Entity Framework

Az első és egyik legfontosabb lépés alkalmazásom tervezésénél az adatmodell megalkotása. Ennek elkészülését követően létrehozom a POCO[[19]](#footnote-19) objektumokat. Az objektumok leképezését a mögöttes adatbázisba az O/RM[[20]](#footnote-20) végzi, mely .NET keretrendszerben az Entity Framework. Microsoft jelenleg 2 támogatott Entity keretrendszere van, az EF 6 és a EF Core. Működésüket és funkciójukat tekintve kifejezetten hasonlóak, azonban fontos megjegyezni hogy az EF Core rendszer - frissebb technológia lévén – még rendelkezik licitációkkal, főleg a komplexebb LINQ[[21]](#footnote-21) lekérdezések esetén. Az EF feladata, hogy minimalizálja az impedancia eltérést az objektum-orientált és relációs világ között. Ennek köszönhetően a fejlesztők olyan alkalmazást írhatnak, melyben erősen típusos .NET objektumokat használva relációs adatbázisban tárolt adatokkal interaktálhatnak. Ezek az objektumok reprezentálják az alkalmazás tartományát[[22]](#footnote-22). Mindennek köszönhetően kiküszöbölhetjük az adathozzáféréshez szükséges kód nagy részét. Az Entity Framework számos funkcióval rendelkezik, ezek között:

* POCO entity osztályok leképezése, melyek nem függenek EF típusoktól
* Automatikus változás nyomon követés
* Identitás feloldás és Unit of Work
* Mohó, lusta és explicit betöltés
* Erősen típusos lekérdezések fordítása LINQ segítségével
* Gazdag leképezési képességek, mely támogatja:
  + Egy az egyhez, egy a többhöz, és a több a többhöz típusú kapcsolatokat
  + Öröklődést
  + Komplex típusokat
  + Tárolt eljárásokat
* „Kód először” funkcionalitás[[23]](#footnote-23)

A EF Core számos adatbázis motort támogat , én az SQL szervert fogom használni.

## Front-end keretrendszerek

A backend oldali technológiai ismertetést követően szeretném bemutatni a frontend oldalon elérhető technológiákat.

### React

A React egy Facebook által fejlesztett frontend framework. A fejlesztőcsapat célja egy magas hatékonyságú User Interface (UI elkészítése volt. A React különlegessége, hogy a programozó a fejlesztés során egy ún. virtuális DOM-al interaktál, ami rengeteg funkcionalitást biztosít. A React egy stabil platform, mely képes nagy forgalom esetén is megbízhatóan működni. Single-page web application[[24]](#footnote-24) építése estén ajánlott használat. Előnyei között említhető a komponensek újrafelhasználhatósága, magas hatékonyság, széles fejlesztői eszközkészlet. Hátrányi között említhető az átfogó dokumentáció hiánya.

### Angular

Az Angular egy Google által fejlesztett frontend keretrendszer. Programozási nyelve a Typescript, mely típusossággal egészíti ki a JavaScriptet A típusosság rengeteg előnnyel jár (kódkiegészítés, bugok korai felismerése), melyek nagyban könnyíti a fejlesztő munkáját. A Reactal ellentéteben, kétirányú adatkötést alkalmaz, ezáltal nincsen valós idejű szinkronizáció a modell és a nézet között. Mobil vagy webes alkalmazás fejlesztése esetén kitűnő választás az Angular. Mindemellett a többoldalas, valamint progresszív webalkalmazások fejlesztésére is alkalmas. Előnyei között említhető beépített funkcionalitás a modell és a nézet közötti szinkronizációra, függőségek szétválasztását külső elemek definiálásával, komponensek újrafelhasználhatósága, átfogó dokumentáció, támogató közösség. Hátrányai között említhető a komplexitásából adódó hosszasabb tanulási folyamat, nem megfelelő optimalizáció esetén lehetséges teljesítménycsökkenés.

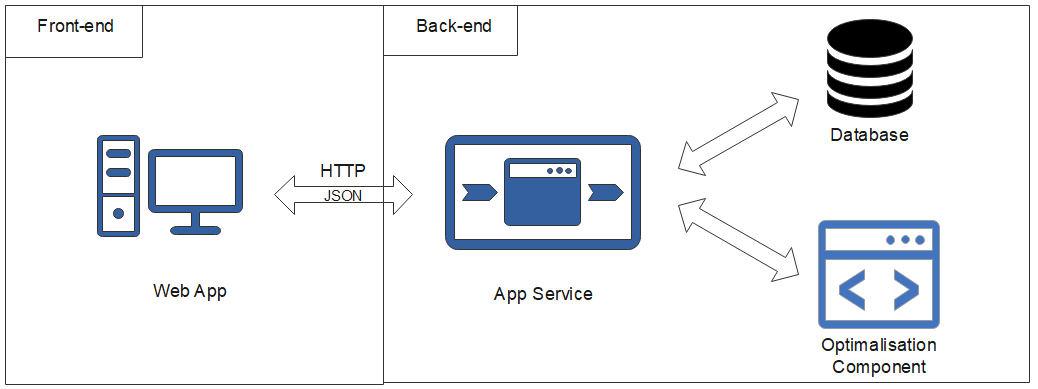
### Vue.js

Az egyik legnépszerűbb frontend keretrendszer, a Vue egyszerű és egyértelmű. Komplexitását tekintve egyszerűbb, mint az Angular. Mérete kicsi, továbbá komponens alapú és virtuális DOM-al funkcionál. Mindemellett kétirányú kötést alkalmaz. Habár alkalmazás optimalizáció és a komplex feladatok megoldása a fő célja, nem tekinthető népszerű keretrendszernek a nagyvállalatok körében. Előnyi között említhető az átfogó dokumentáció, egyszerű szintaxis, tervezési rugalmasság, Typescript támogatás. Hátrányai között említhető a komponenesek instabilitása, viszonylag kicsi támogató közösség, nyelvi korlátok (sok plugin kínaiul nyelven készült).

## Választott technológia

A fenti szempontokat figyelembe véve frontend oldalon Angular keretrendszert fogok alkalmazni. Komponens alapú működésének köszönhetően jól strukturált, hatékony kódot lehet írni. Kétirányú adatkötést tekintve széleskörű funkcionalitást biztosít. A Typescript nyelvi támogatásának köszönhetően megkímélhetem magamat a JavaScript sokszor kiszámíthatatlan működésétől.

Backend oldalon ASP.NET Core alapon fogok egy REST servicet implementálni, mellyel a frontend HTTP protokoll segítségével tud kommunikálni. Az ábrán látható az alkalmazásom magasszintű architektúrája.



6.5 ábra, Alkalmazásom magasszintű felépítése

# Specifikáció

A modern társadalom egyik legnagyobb kihívása a túlsúly. Ez a népbetegség nem csak Magyarországot, hanem az egész nyugati civilizációt sújtja. Sokakban megjelenik az igény az életmódváltásra, azonban megfelelő szakmai segítség hiányában mégsem tudják azt optimálisan véghez vinni. Alkalmazásom segítségével a felhasználó könnyedén meghatározhatja, milyen étrend lenne számára megfelelő jelenlegi életstílusa szerint.

Kifejezetten fontos szempontnak tartom a testre szabhatóságot, melynek érdekében számos személyes paramétert kell megadnia a felhasználónak az étrend generálásához. Mindemellett kiemelt szempontok a következők: reszponzivititás, könnyű használhatóság, magas fokú átláthatóság, gyorsaság, hatékony működés. Tekintsük át főbb pontokba szedve az alkalmazással szemben támasztott funkcionális és nem-funkcionális követelményeket.

Funkcionális követelmények:

1. Egyéni tápanyagigények meghatározása
2. Személyre szabott étrend generálása
3. PDF export készítése

Nem-funkcionális követelmények:

1. Hordozhatóság
2. Gyors működés
3. Könnyű használhatóság
4. Személyes adatok védelme

## Funkcionális követelmények

A program a felhasználó következő személyes paramétereivel dolgozik:

* Nem (M/F)
* Kor
* Tömeg
* Magasság
* Fizikai aktivitás (ülő, alacsony, közepes, magas)

A fenti paraméterek ismeretében az alkalmazás meghatározza a felhasználó napi tápanyagszükségleteit, felhasználva az Amerikai Egészségügyi Minisztérium által kiadott táplálkozási irányelveket [24]. Itt táblázatos formában megtalálható adott nemre, korra, tömegre, magasságra, fizikai aktivitásra, vonatkozóan az ajánlott napi energia-, sóbevitelt; intervallum formában megadva, továbbá hogy ennek mekkora része legyen fehérje, zsír, szénhidrát.

A felhasználónak további információval is kell szolgálnia az étrend megtervezéséhez:

1. megadja az egyes ételcsoportokra vonatkozó preferenciáit 1-től 10-ig tartó skálán
2. megadja a tervezni kívánt étrend általános struktúráját (napi hány étkezés)

Az ételcsoportok a következők: tejtermék, gyümölcs, zöldség, gabona, hús, édesség. Ezen kategóriák további alkategóriákra vannak bontva. A felhasználó akár főkategóriára, akár alkategóriákra vonatkozóan egyenként megadhatja preferenciáját.

Az így megkapott adatok alapján az alkalmazás elvégzi az optimalizációt, figyelembe véve a felhasználó tápanyagszükségleteit, valamint személyes preferenciáját. Az alkalmazás a lehető legkisebb hibára törekszik mindkettő megkötés esetén. z optimalizáció eredménye egy napi étrend, melynek generálását a felhasználó valamennyi alkalommal megismételheti. A felhasználó a számára nem kívánatos étrendet szabadon törölheti. Ezáltal a felhasználó képes egy hétre, vagy akár egy egész hónapra étrendet generálni, melyek sorrendje szabadon módosítható. Bármely kiválasztott étrendet a felhasználó PDF formátumba exportálhatja a böngésző beépített funkcionalitása segítségével.

## Nem-funkcionális követelmények

Célom egy széles körben alkalmazható táplálkozási tanácsadó webes alkalmazás készítése. Azért esett választásom a böngészőből elérhetőségre, mert így az alkalmazás a platformok lehető legszélesebb skáláján használható lesz, legyen az Android, IOS, MacOS, vagy Windows. A webalkalamzás további előnye, hogy nincs szükség telepítésre, az könnyedén elérhető egy böngészőből, mely szinte minden okos elektronikai eszközön adott.

Mindemellett fontos szempont a gyors működés, megfelelve a kor webes trendjének. Ismert tény, hogy a felhasználói élmény nagyban függ a válaszidő hosszától, melynek akár kis mértékű növekedése hatalmas felhasználói veszteséggel járhat. Mivel még nem vagyok ismeretében a probléma megoldásának pontos számítási igényével, pontos követelményeket nehéz meghatározni. Ugyanakkor a fejlesztés során folyamatosan elsődleges szerepet kell adni az alkalmazás hatékonyságának.

A könnyű használhatóság egy olyan követelmény, melyet nehéz konkrét, objektív mérőszámok alapján meghatározni. Ide köthető követelmény a széles körben használt webböngészők teljes körű támogatása (Firefox, Chrome, Safari). Az alkalmazás használatának magától értetődőnek kell lennie, használata bármilyen háttérrel rendelkező felhasználó számára egyszerű kell legyen.

A felhasználó személyes adatainak megóvása érdekében azokat titkosított csatornán szállítom a hálózaton. Az alkalmazás használata regisztrációhoz kötött, a jelszó megadása során számos követelményt meghatározok (minimum 8 karakter, nagybetű, szám, speciális karakter használata). Ezáltal minimalizálni tudom a szenzitív információk kiszivárgásának kockázatát.

# A tervezés során végzett munkafázisok és tapasztalataik leírása

A következőkben áttekintem a tervezés során végzett munkafázisokat, valamint hogy milyen következtetéseket vontam le azokból.

A tervezés első és talán legfontosabb lépése a megfelelő adathalmaz beszerzése, annak megfelelő struktúrába rendezése. Alapos kutatómunkát követően arra a következtetésre jutottam, hogy az adathalmaz beszerzése az én feladatom lesz, melyet web scraping technikával hajtok végre. Sajnos nem elérhető szabadon felhasználható átfogó tápanyaginformációkkal rendelkező recept adatbázis az interneten, mely számomra megfelelő lenne. Ebből kifolyólag beleástam magam a webscraping technológiába, s így megismerkedtem a Scrapy nevű Python keretrendszerrel. Ebben a környezetben írtam egy scrapert, mellyel beszereztem több mint 7000 receptet egy széles körben használt recept weboldalról. Az adathalmaz számos információval szolgál az egyes receptekről: a teljesség igénye nélkül tartalmazza a nevet, kategóriát, leírást, összetevőket (mennyiségekkel), értékelést, átfogó tápanyag információkat. Az utóbbi kiemelendő, hiszen más weboldalakkal ellentétben az oldalon található receptek 19 tápanyagra vonatkozó információt tartalmaznak. Ezek között található a kalória, zsír, telített zsír, szénhidrát, nátrium, koleszterin, kálium, cukor, A vitamin, B vitamin, C-vitamin, kalcium, vas, stb. Ez az adathalmaz kielégítő információt biztosít az optimalizációs technikám kidolgozásához, annak teszteléséhez.

A következő lépés a specifikáció pontos meghatározása. Tehát mit várok el a programtól? Itt mind funkcionális, mind nem-funkcionális követelményekre kitértem. Igyekeztem minél több szempontból megvizsgálni a kérdést, valamint ihletet merítettem más hasonló munkákból. Figyelembe kellett vennem, hogy milyen követelményei lehetnek egy tipikus felhasználónak, legyen az teljesítménybéli, design, vagy biztonságra vonatkozó elvárás. Így született meg a pontos specifikáció.

Egy másik fontos lépés volt a lehetséges technológiák áttekintése. Elsőként a fő technológiai csapásvonalat kellett meghatároznom. Tehát: asztali vagy webes vagy mobil alkalmazást írjak. Az asztali alkalmazás lehetőségét hamar elvetettem, hiszen ezáltal számos kompatibilitási problémába ütköznék. Egy alkalmazás, melyet Windows platformra fejlesztenék, nem működne macOS-en vagy Linuxon. A mobilalkalmazás és a webalkalmazás közötti választás nagyobb kihívást jelentett. Számos előnye van egy mobilos alkalmazásnak, s mára számos keretrendszer biztosít lehetőséget az app fejlesztésre a natív irány mellett. Ugyanakkor a területen való jártasságom hiánya miatt inkább webes alkalmazás fejlesztése mellett döntöttem. Ezen a területen is számos technológia áll rendelkezésre, melyeket korábban bemutattam. Választásom az Angular keretrendszerre esett, többek között modularitása, Typescript kompatibilitása, átfogó dokumentációja miatt.

A front-end fontossága mellett kiemelt szerepet kap a backend rendszer, hiszen alkalmazásom eredményessége nagyban függ a szerveroldali rendszer megfelelő működésétől. Itt fog zajlani a munka jelentős része, hiszen magát az optimalizációt a szerver fogja elvégezni. Ezen szolgáltatását REST API segítségével fogja nyújtani, melynek köszönhetően a front-end rendszerből könnyedén tudok kéréseket indítani. A hálózati kommunikáció természetesen http(s) protokoll szerint fog zajlani, formátumát tekintve JSON adatok segítségével. A szerver oldali rendszert, a Rest Api-t C#-ban fogom implementálni. Ez a web komponens egy másik komponenssel fog kommunikálni, mely az optimalizációért felel.

Tehát az implementáció fázisa áll előttem, melynek során folyamatos finomhangolások mentén fogom fejleszteni alkalmazásomat.

## Strukturális leírás

Szeretném bemutatni alkalmazásom magasszintű strukturális felépítését ebben a fejezetben. Szemléltető ábra a mellékletben található (9.2 ábra).

Webalkalmazásról lévén szó, a rendszer két fő részre osztható: front-end és back-end oldali komponensek. A kliens számítógép futtatja böngészőjében a webalkalmazást, mely folyamatos http kérések indításával, válaszok fogadásával kommunikál a szerver-oldali rendszerrel. Az étrend generálásának szekvencia diagramja a 9.2 ábrán látható. Ezen kérések kezeléséért egy kijelölt MealPlanner REST komponens felelős. E komponens központi szerepet játszik a szerver oldalon, hiszen ő biztosítja a külvilág felé az interfészt, mellyel kommunikálunk, s mindemellett a többi back-end oldali komponens hívásáért felel. Gyakorlatilag vezénylőként irányítja a szerver-oldali folyamatokat: optimalizáció, perzisztencia, exportálás, autentikáció. A felsorolt komponensek mind az alkalmazás egy-egy funkcionalitásáért felelnek.

Az optimalizációs komponens felelős az adott paramétereknek megfelelő optimális étrend generálásáért. A komponens legfontosabb része a genetikus algoritmus, mely az egyes generációkat egymást követő iterációkban fejleszti. A kommunikációt az optimization interfészen keresztül biztosítja. Ez az interfész több metódust tartalmaz, melyekkel különböző étrendeket kérhetünk le. A metódusok paraméterként fogadják ez étrenddel szemben támasztott megkötéseket, és a fitnesz függvény meghatározásához szükséges paraméterket. A perzisztencia komponens felelős az adatbázis műveletek meghívásáért, melyekhez egy persistence interfészt biztosít. Gyakorlatilag facadeként működik, egy leegyszerűsített interfészt biztosít a mögöttes komplex rendszerhez, mely esetünkben az adatbázis. A biztonsági komponens felelős a felhasználók autorizációjáért. A webalkalmazásba való bejelentkezés esetén MealPlanner REST komponens a felhasználó adatbázisból kinyerését követően ezen komponenst hívja meg bármilyen kérést megelőzően, hogy meghatározza, hogy van-e a felhasználónak jogosultsága az adott funkcióhoz.

Így néz ki magas szinten alkalmazásom felépítése. Tervezés során törekedtem az erős kohézióra, és a függéségek szétválasztására.

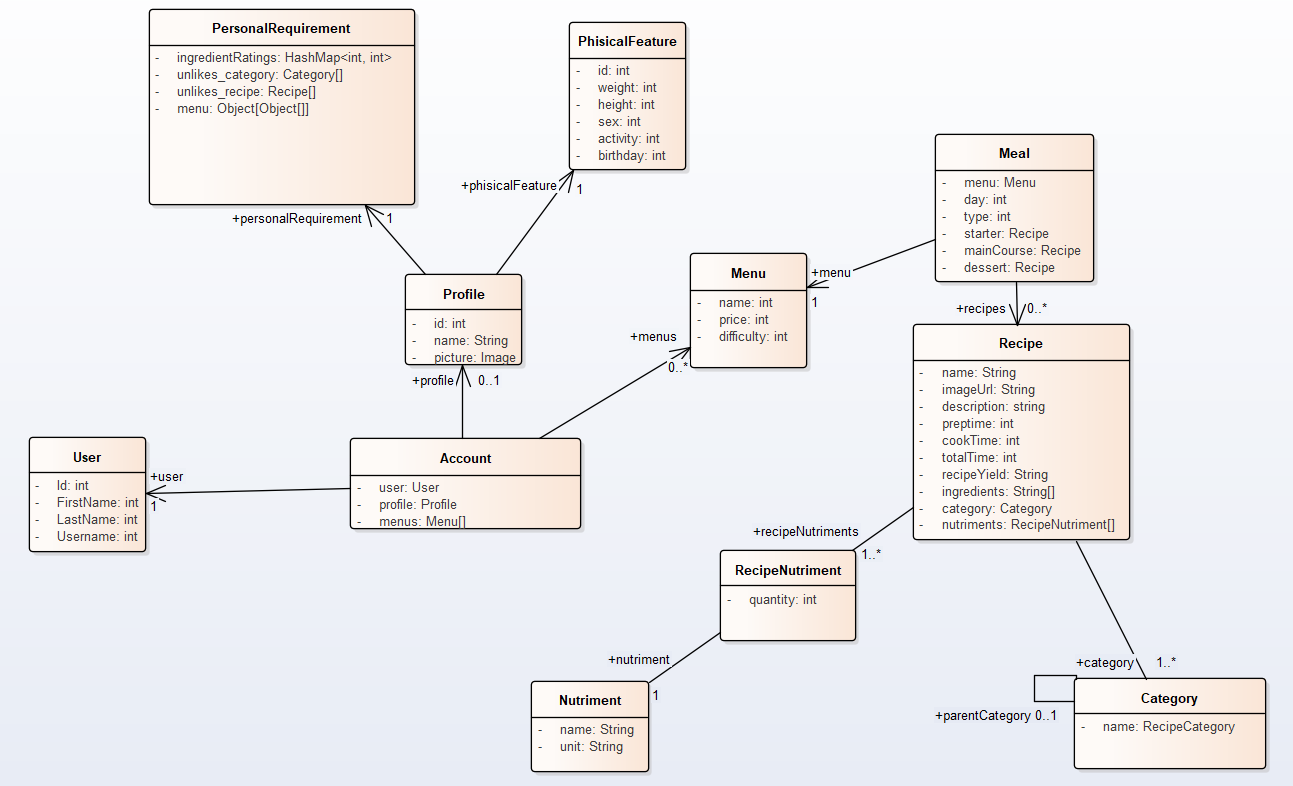
# A megvalósítás leírása

Az alkalmazás implementálása során törekedtem a mind a frontend mind a backend projektet átláthatóan és logikusan felépíteni, függőségeket szétválasztani, valamint az egyetemi tanulmányim során elsajátított szoftverfejlesztési praktikák mentén dolgozni.

## Az adatok beszerzése

Az optimalizációhoz szükséges adathalmazt egy közismert online recept weboldalról crawloltam le. Ehhez a Python nyelven használható Scrapy keretrendszert alkalmaztam. A kezdeti hosszas kutatómunka során számos szempontot kellett figyelembe vennem, a recepteknek átfogó információval kell szolgálnia tápanyagokat illetően (makro- és mikrotápanyagok), katgóriákba kell hogy legyenek sorolva (a felhasználói preferencia figyelembe vétele miatt), ételek széles skáláját kell tartalmazniuk. Ezen adatokat egy angol eredetű recept weboldalon találtam meg, melyek beszerzéséhez a már említett Scrapy keretrendszert alkalmaztam. Itt figyelembe kellett vennem a webcrawling általánosan elfogadott irányelveit annak elkerülése érdekében, hogy túlzott terhelést támasszak a szerverek felé.

## Entity modell



9.1 ábra, Entity modell

A beszerzett adatok, valamint a pontos specifikáció ismeretében megkezdtem az adatbázis struktúra kialakítását. Az entity modell kialakítása során figyelembe vettem a négy normálforma irányelveit, logikus struktúra kialakítására törekedtem. Minden alkalmazás megtervezése során ez egyik legfontosabb lépés az entity modell létrehozása. Az struktúra kialakítása során tisztázódik, hogy pontosan milyen entitásokkal fog az alkalmazás működni. Ezen osztályok az ASP NET Core Code first megközelítés segítségével könnyen leképezhetőek adatbázis táblákká. Mindemellett deployoltam egy MSSQL szervert gépemen, melyhez a későbbiek során csatlakozom az alkalamazásomból.

## Backend projekt felépítése

Következő lépésként létrehoztam mind a frontend mind a backend vázát. Elsőként a a backend felépítése: az alkalmazás egy OpimalDietAPI nevű solutionben található, mely 9 darab különálló projektet tartalmaz. A projektek a következők: Commons, Interfaces, Maps, WebAPI, MealOptimizer, Models, Repositories, Services, ViewModels. Legmagasabb szinten a WebAPI projekt található, mely a http kéréseket kiszolgáló kontrollereket, továbbá az alkalmazás egészéhez kapcsolódó általános konfigurációkat tartalmaz. Ez a projekt funkcionál az alkalmazás startup projektjeként. A projekt 4 darab kontrollert tartalmaz. A UsersController a felhasználó regisztrációjáért és authntikációjáért felel, valamint profiljához kapcsolódó és egyéb felhasználóhoz kapcsolódó végpontokat tartalmaz. A RecipesController egyetlen végpontja segítségével hajthatjuk végre a kezdeti migrációt az adatbázisba. Itt a már korábban általam említett körülbelül 7500 receptről van szó, melyek json formában találhatóak meg a repository data könyvtárában. A MenuController központi szerepet foglal el az alkalmazás működésében, hiszen a core logika előtt lévő végpontokat tartalmazza. Itt van lehetőség menü generálására a /generatemenu végponton keresztül, valamint az elkészült menü lekérésére, annak törlésére. A menü módosítását nem találtam ésszerű funkcionalitásnak, hiszen ezáltal az alkalmazás elveszítené a lényeges funkcionalitását, ami a személyre szabott optimális menü előállítása. Az utolsó kontroller a CategoriesController, mely ugyancsak egyetlen végpontot tartalmaz. A categories/getcategories végponton keresztül a frissen létrehozott profilhoz kérhetőek le itt a felhasználó által értékelhető kategóriák. A kontrollerekhez tartozó endpointokat a mellékletben a 12.3 ábrán vizuálisan szemléltetem.

Az alkalmazás modellje a Models projektben található, felépítését részletesebben taglaltam feljebb. Továbbá itt található az adatbázis működéséhez szolgáló DbContext osztály. Itt definiálom az adatbázisból Entity Framework segítségével elérhető DbSeteket, mely referenciák segítségével tudom manipulálni az adatbázisban található adatokat. Mindemellett további adatbázis konfigurációk találhatóak itt: loggolás, connectionstring beállítása, stb.



9.2., A DataContext osztály

A kontrollerek kiszolgálásáért a Services projekt felelős, mely egy köztes kiszolgálóként funkcionál az adatbázis és a kontrollerek között. Minden kontrollerhez tartozik egy service, mely végrehajtja a kapcsolódó üzleti logikát, valamint tovább hív a perzisztenciáért felelős Repositories projektbe. MenuService osztály emelendő ki, mely a magasabb szintű logikát leszámítva tovább delegálja a menü generálásának feladatát a MealPlanner projektben található optimalizációs komponensnek. Itt található meg az alkalmazás fő logikáját végrehajtó genetikus algoritmus (9.3 kódrészlet). Az algoritmus későbbi optimalizációja céljából statisztika generáló, valamint egy NSGA2 algoritmus (9.4. kódrészlet) is implementálásra került ebben a komponensben.





9.3., A generikus genetikus algoritmus, rövidítve







9.4., GA paramétereket optimalizáló NSGA algoritmus, rövidítve

További projektek a Commons, a Maps, Interfaces, ViewModels. A Commons projektben a globális konfigurációs paraméterek mellett más komponensek által közösen használt osztályok találhatóak, mint például a a tápanyagszükséglet számoló osztály. David Frankenfield tanulmánya [30] megmutatja, hogy a Mifflin St Jeor formula bizonyul a legpontosabbnak, számításomhoz ezt veszem alapul. Mindemellett a genetikus algoritmus statisztika osztálya, objektum konverter, alkalmazásszintű kivétel osztály, és további segéd osztály található ebben a komponensben.

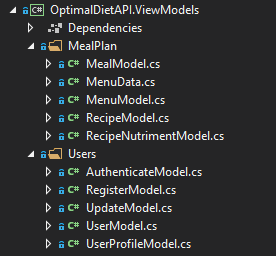
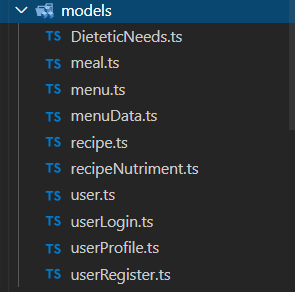
A Maps projekt biztosítja az alkalmazás domainje és a viewmodel közötti leképezést (illeszkedve az MVVM tervezési mintához). Két osztály található itt, melyek egyenként előbbi művelet egyes irányokba való elvégzéséért felelnek. A mappeléshez a Jimmy Bogard által fejlesztett AutoMapper csomagot használom. Konvenció-alapú működésének köszönhetően könnyen és tömören implementálható a viewmodel és a domain osztályok közötti leképezés.

Az Interfaces projekt – elnevezésének megfelelően – az alkalmazás interfészeit tartalmazza. Minden service-hez tartozik egy interfész, mely elengedhetetlen a dependency injection használatához. Mindemellett még két fontos interface található itt, a menügeneráló komponens interface, valamint egy generikus repository interface, mely az alapvető CRUD műveleteket kényszeríti ki a megvalósító repository osztályokból. Ezen interfacek csupán a service publikus függvényeinek fejléceit tartalmazza, így gyakorlatilag egy kontraktusként szolgálnak. A Startup osztályban ezen serviceket és a hozzájuk tartozó osztályokat regisztrálom a dependency injection keretrendszer számára, ahogy a következő kódrészletben az látható.



9.5., A Startup osztály ConfigureServices metódusában a dependency injection beállítása

A ViewModels projekt határozza meg a frontend és a backend közötti kommunikáció során alkalmazott típusokat (9.6. ábra). Egy alkalmazás működése során nem biztonságos, és sokszor nem logikus egy az egyben felfedni az entity típusokat. Ezen objektumok szenzitív információt tartalmazhatnak (pl.: id, passwordhash), melyeket nem küldök ki a frontend számára. Mindemellett a frontenden elvárt logikai felépítés sok esetben nem felel meg a entity modell felépítésének, ezért szükséges a viewmodel osztályok kilapítása. Ezen típusok az Angular alkalmazás models könyvtárában is megtalálhatóak.



9.6. ábra, A létrehozott viewmodel osztályok backenden (bal) és frontenden (jobb)

## Frontend projekt felépítése

Az Angular projektben hasonlóképpen törekedtem a logikus felépítésre, illetve a keretrendszer által adott lehetőségek minél széleskörűbb kihasználására.

Elsőként egy általános elrendezést határoztam meg három fő komponens segítségével, melyet az app komponensbe való beágyazással értem el (9.7. kódrészlet). A layout könyvtárban találhatóak meg: app, dashboard, footer, head. Az app komponensben természetesen megtalálható egy router-outlet, mely lebonyolítja a routingot, azaz segítségével renderelődnek ki az egyes komponensek az oldalra. Továbbá beágyazásra került egy register, illetve egy login komponens is ide. Az autentikáció lebonyolítása érdekében segítő service-t, valamint authguardot hoztam létre (9.10. kódrészlet). A segítő service kezeli a http kérésekben megtalálható jwt tokent, mely az azonosított user minden kérésében benne kell lennie. Az autentikációs és authorizációs authguard segítségével bármely kompones elérését korlátozhatom. Az autentikáció néhány komponens kivételével mindíg elvárt. További komponensek a componsents könyvtárban találhatóak. Itt találhatóak többek között a menu, menus, nutritional-info, recipe, user-profile komponensek.



9.7. kódrészlet, A frontend gyökér html oldala

A backendel való kommunikáció lefolytatása érdekében számos service-t implementáltam, melyek az http kérések küldéséért felelnek. Négy fő servicet hoztam létre: category, menu, token, user. A categoryservice felelős az osztályozandó kategóriák lekéréséért a user-profile komponensben (9.8. kódrészlet). Továbbá mivel az adatokat más struktúrában kapom meg, mint ahogy azokkal dolgozok a frontend alkalmazásban, átalakító függvényeket is implementáltam a categoryserviceben. A menuservice a felhasználóhoz köthető menü generálását kezdeményező, és egyéb menühöz kapcsolódó művelet elvégzését kezdeményező http kérésekért felelős. A tokenservice egyetlen publikus függvénye segítségével történik az autentikáció a backend felé. További service a baseservice, mely hibakezelést, kérések headerjébe az authentikációs token hozzáadását, és egyéb segítő funkcionalitást végeznek el.



9.8. kódrészlet, A menüket kezelő service

Az app-routing modul definiálja az alkalmazásban található routokat. Egy route meghatározásához definiálhatunk kell egy relatív elérési útvonalat, esetleges redirect útvonalat, path azonosításának módját, esetleges autentikációt, a cél komponsenst, illetve számos egyéb lehetséges paramétert (9.9 kódrészlet). Az app modul az egyetlen gyökér modul az alkalmazásban, melyben definiálásra kerülnek a deklarált, az importált kompnensek. Továbbá definiálásra kerülnek a providerek, valamint meghatározásra kerül a bootstrap komponens. További megemlítendő elem az alklamazásban a config.ts file, melyben kettő konfigurációs paraméter került meghatározásra. Ezek a API path és az environment típusa. Ezen változókat hivatkozásként használom az alkalmazásban, esetleges cseréjüket ezzel jelentősen megkönnyítve.



9.9. kódrészlet, Az alkalmazás routejai (url végpontok)

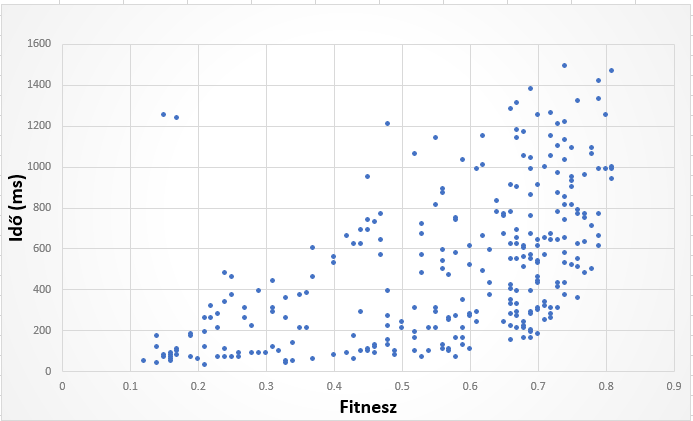


9.10. kódrészlet, Az autentikációt ellenőrző guard, mely a helper osztály isAuthenticated függvénye segítségével határozza meg, hogy a felhasználó böngészőjében a szükséges token megtalálható-e

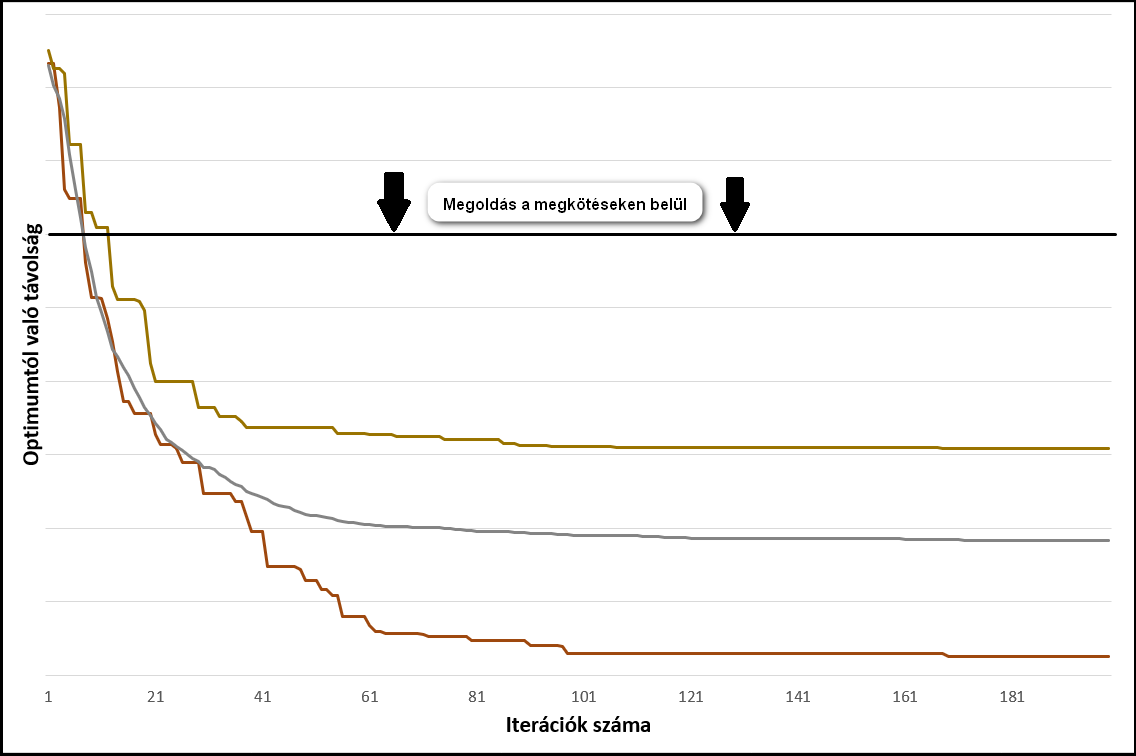
# Eredmények

## Optimális GA paraméterek

Számos tesztet hajtottam végre annak érdekében, hogy meghatározzam az ideális kombinációját a GA paramétereknek. Ezen paraméterek a következők: kiválasztás során versenyeztetésnél az elemek száma, mutációs ráta, rekombinációs ráta, populáció mérete, kilépési feltétel. Kezdeti próbálkozásként a szakirodalomban megtalálható ajánlások alapján a paraméterek kézzel állítottam folyamatosan. Hamar rájöttem, hogy ezen módszer nem vezet sikerre, hatékonysága rendkívül kis fokú. Emiatt konzulensem ajánlása alapján áttekintettem a Haladó Algoritmusok tantárgy keretei belül tanult algoritmusokat, mely lenne megfelelő a GA paraméterhalmaz optimalizálására. Választásom az NSGA2 algoritmusra esett több okból kifolyólag is. Egyrészt viszonylag hatékony (NSGA algoritmushoz képest számos javítást tartalmaz), másrészt többcélú optimalizációra kitűnően alkalmas. Továbbá nem egyetlen megoldást ad, hanem egy ún. pareto frontot prezentál számomra, melyek közül kiválaszthatom a számomra megfelelőt. Az optimalizáció során 2 célfüggvényt alkalmaztam, maximális fitneszre és minimális futási időre törekedtem. Az algoritmus implementációja a genetikus algoritmus projektjében található. Az újonnan létrehozott végpont hívásához egy egyszerű kliens projektet hoztam létre, mellyel meghatározott számú menü generálási kérést tudok indítani. Az adathalmaz teljes véletlenszerűségét elkerülendő, egy személyes fizikai paraméter adathalmazt töltöttem le a kaggle.com-ról [31]. A tesztek végrehajtása mindig ugyanazt a 20 paraméterhalmazt alkalmazva generáltam étrendet, ezen futások statisztikáit pedig összesítettem.



10.. ábra, Az optimalizációs algoritmus futási ideje különböző konfigurációs paraméterek esetén. Minden pont 20 futás átlagát mutatja (magasabb fitnesz jobb).



0.2. ábra, Az optimumtól való távolság a legrosszabb (felső görbe), az átlagos (középső görbe) és a legjobb (alsó görbe) megoldás 10 futásból, az iterációk száma függvényében

## Konvergencia gyorsasága az optimális megoldáshoz

Mivel az alkalmazás webes környezetben fog működni, a futási idejű teljesítmény kiemelt szerepet kap. A parméterhalmaz optimalizációja során egy olyan megoldást kerestem, mely a lehető leggyorsabban konvergál az optimális megoldáshoz. Mindemellett a legfontosabb szempont a megoldás minősége. Ennek érdekében az implementált NSGA2 algoritmus segítségével a paraméterek egy széles skáláját tudtam megvizsgálni. A beállítandó paraméterek a következők: populáció mérete, elitizmus értéke, lokális optimumba való beakadás esetén további iterációk száma, mutációs ráta, versenyeztetés esetén alkalmazott elemszám, keresztezési arány. Az NSGA2 algoritmus által felajánlott pareto frontból a számomra legideálisabb paraméterhalmazt tudtam kiválasztani. Ahogy említettem, mind a futási idő, mind pedig a megoldás minősége kiemelt szerepet kap az optimalizációs algoritmus kidolgozása során. A futási időt a generációnként végrehajtott műveletek száma határozza meg. Nem meglepő módon a mutációs és keresztezési ráta erős lineáris korrelációt mutat a futási idővel. A tesztek futtatása során kiderült, hogy viszonylagosan nagy populáció méret az ideális, 3-4000 elemmel. Ebből kifolyólag a mutációs ráta értéke viszonylag alacsony értékre csökken (0,1 alatt) a populáció méretét ellensúlyozva. A keresési tér nagy mértéből adódóan, annak érdekében, hogy a lehető legtöbb alléljelölt kombinációt megvizsgáljunk a megoldáshoz, magas keresztezési arányra van szükség (~0,6).

Ahogy a 10.1 ábra mutatja, a megoldás minősége hosszabb futási idő estén javul, melyet részben a beállított paraméterek okoznak. Ezzel szemben az optimumhoz való konvergencia jelentősen lelassul egy idő után ezekben az esetekben, mindemellett viszonylag magas minőségű megoldások jönnek létre ezen idő során. Ebből az következik, hogy elegendő a az algoritmust addig futtatni, amíg a boríték görbe el nem kezd sűrűsödni.

Az algoritmus konvergenciáját étrend generáló teszt futásokkal vizsgáltam. A megkötések a következők voltak (egy napi menüre): 2148 kcal energiabevitel, 69,81 – 187,95 g közötti fehérjebevitel, 241,65 – 349,05 g közötti szénhidrátbevitel. A populáció mérete ~4500 volt, a keresztezés és mutáció valószínűsége ~0,7 és ~0.03. Az eredmények azt mutatják, hogy négy megkötés esetén átlagosan 8,5 iteráció alatt sikerült a elégséges megoldást találni (legrosszabb esetben pedig 14 iteráció alatt). Ahogy az a 10.2 ábrán látható, körülbelül 100 iterációt követően alig tapasztalható javulás, tehát egy közel optimális megoldás megtalálható ezen idő alatt.

## Továbbfejlesztési lehetőségek

A jövőben meg fogom fontolni a Raidl által kidolgozott inicializáló, optimalizációs, javító módszereket a probléma LP relaxált változatának megoldásához. A genetikus algoritmus optimalizációjához felhasznált NSGA2 algoritmus nem csak erre a célra lehet alkalmas, hanem akár a problémát megoldandó algoritmusként is lehetne használni a jövőben.

# Összefoglalás

Ezen dolgozatban egy automatizált, személyre szabott menü tervező alkalmazás fejlesztésének folyamatát írtam le.

Kezdeti lépéként megvizsgáltam a fejlesztéshez felhasználható technológiákat. A lehetőségek áttekintését követően választásom egy kliens-szerver architektúrát megvalósító webes alkalmazásra esett. Az implementáció során törekedtem az alacsony csatolásra és a magas kohézióra. A két rendszer egy közös interfacen kommunikál egymással JSON üzentek formájában.

A feladat pontos definiálása, valamint annak megoldási módjának megtalálása volt az első fő feladatom. A hasonló problémákat megoldani kívánó tudományos munkák áttekintése során kiderült, hogy feladatomat egy kombinatorikai problémaként lehet definiálni. Számos kutatás áttekintését követően választottam a genetikus algoritmust a probléma megoldásához, mely a nagy keresési térrel rendelkező problémák esetén hatékonyan, gyorsan megtalálja az optimális megoldást. Törekedve a későbbi újrafelhasználhatóságra, az algoritmust generikus módon implementáltam. Az algoritmushoz implementáltam egy fitnesz függvényt (a célfüggvények kombinálásával), mely az egyes étrendeket értékeli tápanyagösszetétel, felhasználó személyes preferenciája, és sokszínűség alapján. Ezen szempontok közül a tápanyagösszetétel veszem legnagyobb súllyal figyelembe. Ezen célfüggvény egy intervallumnak való megfelelést vizsgál (minimális, optimális és maximális tápanyagmennyiség), melynek megfelelően értékelést ad az adott napi étrendre.

Az algoritmus implementációját követően egy újabb optimalizálási problémával szembesültem. A genetikus algoritmusnak számos indítási paraméterre van szüksége, melyek manuális kalibrálása meglehetősen nagy kihívás. Az optimális paraméter kombináció megtalálásának érdekében újabb optimalizációs algoritmust kerestem. A haladó algoritmusok kurzus során megismert algoritmusok áttekintését követően az NSGA2 algoritmusra esett választásom. Az optimalizáció végrehajtása érdekében egy egyszerű konzolos kliens alkalmazást készítettem, melynek segítségével egy időben akár több száz étrend generálási kérést indíthatok az API-m felé. A konzisztencia érdekében létrehoztam több száz teszt adatot felhasználói fizikai paramétereket és személyes preferenciát illetően. Mindemellett statisztikai segítő osztályokat is implementáltam, melyeknek köszönhetően vizuálisan is képes voltam szemléltetni az futások hatékonyságát, futási idejét, generált étrendek minőségét.

Az teszt futtatások során kiderült, hogy viszonylag nagy méretű populációra van szükség, alacsony mutációs rátával, és magas keresztezési rátával. Mindemellett kiderült, hogy egy maghatározott időt követően a megoldás minősége már csak jelentéktelen mértékben javul. Az alkalmazás sokszínű, megkötéseknek megfelelő és harmonikus étrendet generál.

# Conclusion

The aim of my thesis project was the implementation of a custom meal planner application.

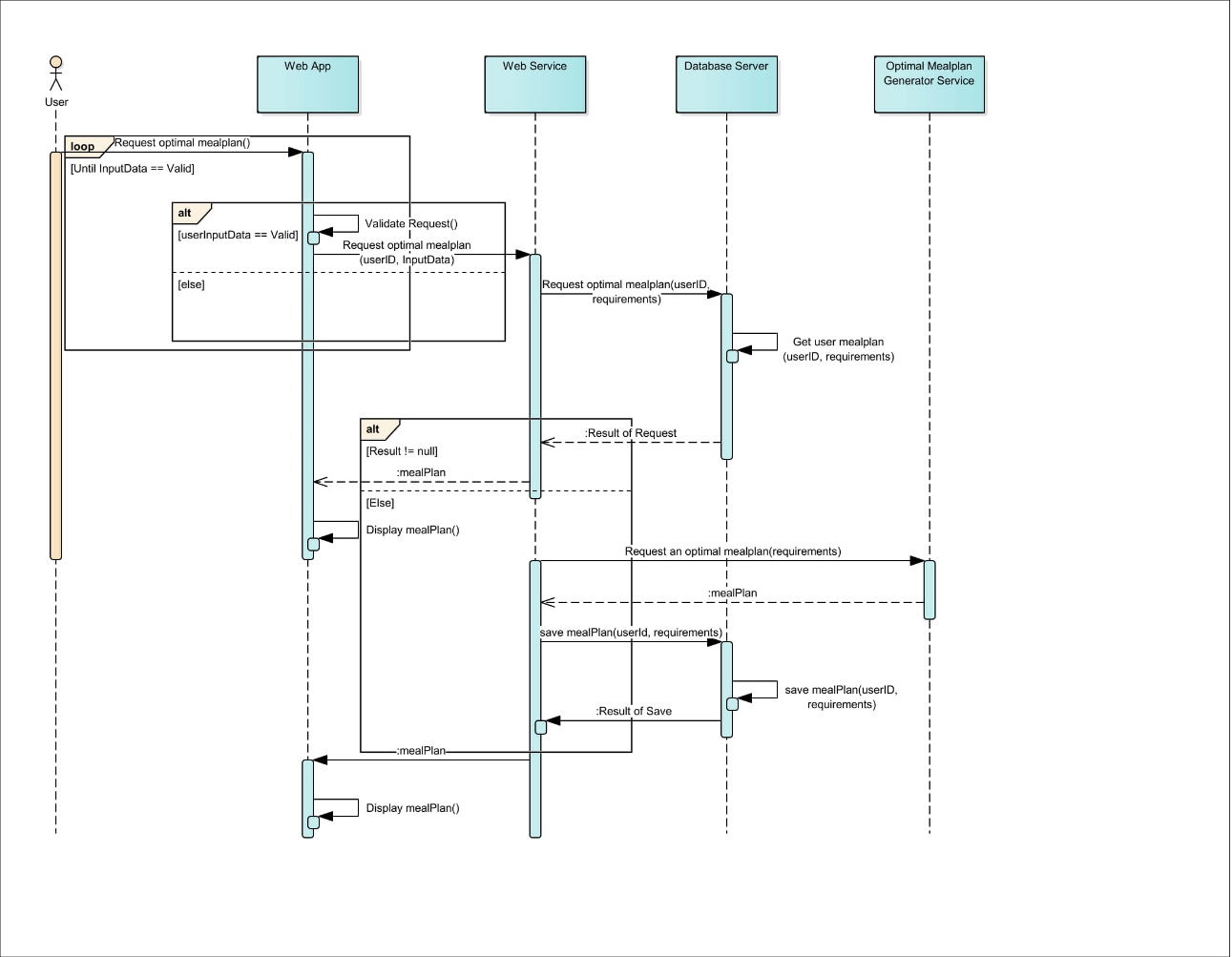
A thourough research has preceded my implementation during which I have examined the possible technologies I could use for development. I have decided to implement a webapplication building upon the client-server architecture. During the implementation process I have aimed for high cohesion and low coupling. The two systems (fronted and backend) communicate on a common interface using JSON messages.

During this phase my main goal was defining my problem specifically and finding the most efficient way to solve it. After overviewing scinteific papers with similar problem sets I have came to the conclusion that my task can be specified as a combinatorical problem. Additionally I have decided to implement a genetic algorithm, which could find quickly and effficiently the optimal solution even in large search spaces. With reusability in mind, I have implemented the algorithm in a generic way. I have implemented a fitness function for the algorithm by combining the aim functions. This function evaluates each possible meal combination by taking into account the dietetic content, user’s personal needs and variety. These aim functions are not taken into account with the same weight: the dietetic needs are the most important aspect of a mealplan. This aim function examines how well the mealplan meets each dietetic constraint taking into account the minimal, maximal and optimal nutrient content.

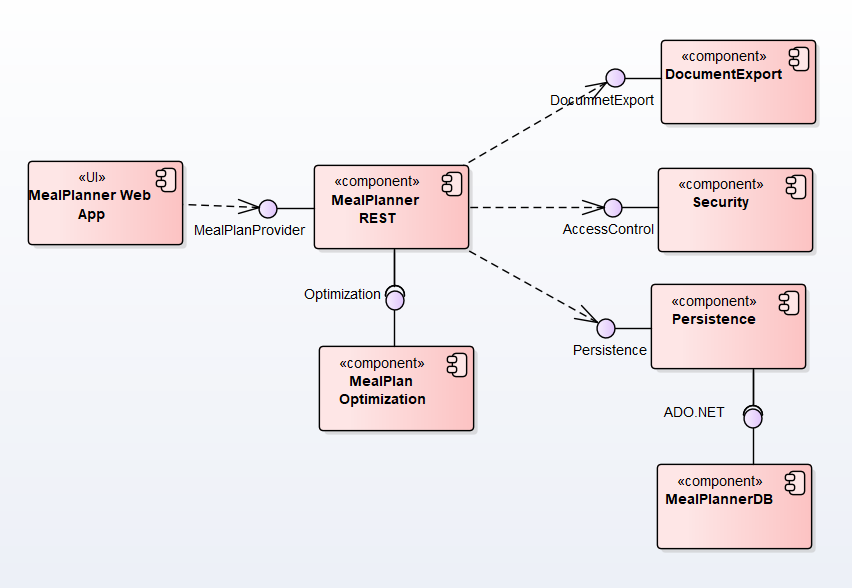
After implementing the genetic algorithm I have faced another optimalisation problem. The genetic algorithm needs a number of initialisation parameters. Calibrating these values manually is a huge challage. With the aim of finding the optimal parameter combination I have started my research over in search of the right optimalisation algorithm. After overviewing the algorithms learned in the advanced algorithms course I have chosen the NSGA2 algorithm. To execute the optimalisation I have implemented a simple console application. With this app I could initiate several hundred meal planning requests to my API. For consistency reasons, I have created several hundred test datasets with user’s phisical parameters and personal preferences. Furthermore, I have created statistical helper classes, which I could use to visually illustrate efficiency, runtime, and the generated meal plan’s quality during runs.

From test runs I have concluded that a relaitevely large population is needed with a a low mutation rate and a high crossover rate. Furthermore, I have concluded that after a given time period the quality of the solution only improves by a negligable amount. The application produces a harmonic and diverse mealplan wich meets the constraints.

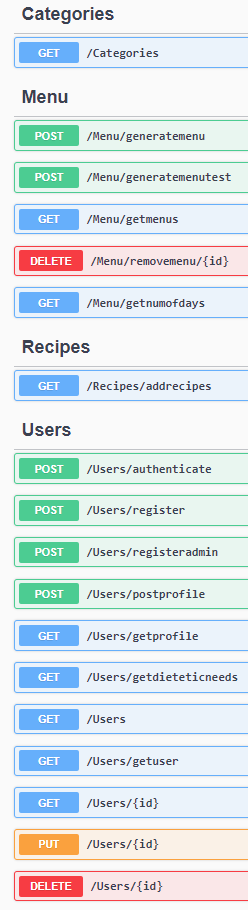
# Melléklet



12.. ábra, Optimális étrend generálásának szekvencia diagramja



12.2 ábra, A rendszer komponens diagramja



2.3. ábra, A létrehozott API endpointok

# Irodalomjegyzék

|  |  |
| --- | --- |
| [1] | „KSH,” 23 július 2018. [Online]. Available: https://www.ksh.hu/docs/hun/xftp/stattukor/egeszsegallapot1617.pdf. [Hozzáférés dátuma: 14. május 2020]. |
| [2] | L. Halmy, „Az elhízás gyakorlati kérdései,” *HIPPOCRATES,* pp. 130-136., 2017. |
| [3] | J. Dollathite, D. Franklinms és R. McNew, „Problems encountered in meeting the Recommended Dietary Allowances for menus designed according to the Dietary Guidelines for Americans,” *Journal of the American Dietetic Association,* pp. 341-347, 1995. |
| [4] | L. Sterling, G. Petot, C. Marling és K. Kovacic, „The role of common sense knowledge in menu planning,” *Expert Systems with Applications,* p. 303, 1996. |
| [5] | G. J. Stigler, „The Cost of Subsistence,” *Journal of Farm Economics,* pp. 303-314, 1945. |
| [6] | J. L. Balintfy, „Menu Planning by Computer,” *Communication of the ACM,* pp. 255-259, 1964. |
| [7] | J. Kolodner, Case-based reasoning, San Mateo, USA, 2014. |
| [8] | D. Frye, P. D. Zelazo és T. Palfai, „Theory of mind and rule-based reasoning,” *Cognitive Development,* pp. 483-527, 1995. |
| [9] | C. R. Marling, G. J. Petot és L. S. Sterling, „Integrating case‐based and rule‐based reasoning to meet multiple design constraints,” *Computational Intelligence,* 1999. |
| [10] | G. J. Petot, C. Marling és L. Sterling, „An artificial intelligence system for computer-assisted menu planning,” *Journal of the American Dietetic Association,* pp. 1009-1014, 1998. |
| [11] | K. Kovacic, Using common-sense knowledge for computer menu planning, USA: UMI, 1995. |
| [12] | A. Hoffmann és A. Salam Khan, „An advanced artificial intelligence tool for menu design,” *Nutrition and health,* 2003. |
| [13] | N. Mohd-Yusoff, N. S. Shafii és Z. Abdul-Manaf, „DietPal: a Web-based dietary menu-generating and management system,” *Journal of medical Internet Research,* 2004. |
| [14] | A. Kahraman és H. Aydolu Seven, „Healthy Daily Meal Planner,” in *GECCO*, Washington, D.C. USA, 2005. |
| [15] | H. Kellerer, U. Pferschy és D. Pisinger, Knapsack Problems, Springer-Verlag, 2004. |
| [16] | B. Gaál, I. Vassányi és G. Kozmann, „A Novel Artificial Intelligence Method for Weekly Dietary Menu Planning,” *Methods of Information in Medicine,* pp. 655-664, 2005. |
| [17] | A. Eiben és J. Smith, Introduction to Evolutionary Computing, Németország: Springer, 2015. |
| [18] | P. Chu és J. Beasley, „A Genetic Algorithm for the Multidimensional Knapsack Problem,” *Journal of Heuristics,* pp. 63-86, 1998. |
| [19] | A. Wieczorek, „Pseudo-Utilities,” in *Springer-Verlag*, Németország, 1986. |
| [20] | H. Pirkul, „A heuristic solution procedure for the multiconstraint zero‐one knapsack problem,” *Naval Research Logistics,* pp. 161-162., 1987. |
| [21] | G. Raidl, „An improved genetic algorithm for the multiconstrained 0-1 knapsack problem,” in *IEEE*, USA, 1998. |
| [22] | C. Coello Coello, „A Comprehensive Survey of Evolutionary-Based Multiobjective Optimization Techniques,” *Knowledge and Information Systems,* p. 269–308, 1999. |
| [23] | D. Kalyanmoy, Multi-Objective Evolutionary Algorithms, Berlin: Springer, 2015. |
| [24] | 2015–2020 Dietary Guidelines for Americans. 8th Edition, U.S. Department of Health and Human Services and U.S. Department of Agriculture., 2015. |
| [25] | M. Masse, REST API Design Rulebook, USA: O'Reilly Media, Inc., 2011. |
| [26] | „TechEmpower,” [Online]. Available: https://www.techempower.com/benchmarks/#section=data-r18&hw=ph&test=plaintext. [Hozzáférés dátuma: 14. május 2020.]. |
| [27] | Microsoft, [Online]. Available: https://dotnet.microsoft.com/learn/aspnet/what-is-aspnet. [Hozzáférés dátuma: 14. május 2020.]. |
| [28] | J. Leyden, „GitHub reinstates Russian who hacked site to expose flaw,” *The Register,* 05. március 2012. |
| [29] | D. Paquette, J. Chambers és S. Timms, ASP.NET Core Application Development: Building an Application in Four Sprints, USA: Microsoft Press, 2015. |
| [30] | R. Anderson és S. Smith, „docs.microsoft.com,” Microsoft, 06. május 2020. [Online]. Available: https://docs.microsoft.com/en-us/aspnet/core/fundamentals/middleware/?view=aspnetcore-3.1. [Hozzáférés dátuma: 14. május 2020]. |
| [31] | D. R. Prasanna, Dependency Injection, USA: Manning, 2009. |
| [32] | R. P. C. Moreira, E. F. Wanner, F. V. C. Martins és J. F. M. Sarubbi, „CardNutri: A software of Weekly Menus Nutritional Elaboration for Schoolar Feeding applying Evolutionary Computation,” in *Springer, Cham*, Svájc, 2018. |

1. Case-Based Reasoning (CBR) [↑](#footnote-ref-1)
2. Rule-Based Reasoning (RBR) [↑](#footnote-ref-2)
3. Multidimensional Knapsack Problem - MKP [↑](#footnote-ref-3)
4. Dietari Reference Intake – Ajánlott tápanyag bevitel [↑](#footnote-ref-4)
5. Linear Programming(LP) [↑](#footnote-ref-5)
6. Multiobjective Evolutionary Algorithm (MOEA) [↑](#footnote-ref-6)
7. Multiobjective Problem (MOP) [↑](#footnote-ref-7)
8. National Health Service – (Amerikai) Nemzeti Egészségügyi Szolgálat [↑](#footnote-ref-8)
9. Daily Reference Intakes – Napi Ajánlott bevitel [↑](#footnote-ref-9)
10. GA [↑](#footnote-ref-10)
11. uniform crossover [↑](#footnote-ref-11)
12. Software Development Kit – Szoftverfejlesztő készlet [↑](#footnote-ref-12)
13. Android Package File – telepítőfájlok (szabványos JAR archívum, egyéb metaadatokkal) [↑](#footnote-ref-13)
14. Model-View-Controller, MVC [↑](#footnote-ref-14)
15. Don’t Reapeat Yourslf – ne ismételd önmagadat [↑](#footnote-ref-15)
16. angolul: action [↑](#footnote-ref-16)
17. angolul: middleware [↑](#footnote-ref-17)
18. angolul: Inversion of Controll - IoC [↑](#footnote-ref-18)
19. Plain Old CLR Object – külső függőségekkel nem rendelkező objektum [↑](#footnote-ref-19)
20. Object Relational Mapper – Objektum relációs leképező [↑](#footnote-ref-20)
21. Language Integrated Query – Integrált nyelvi lekérdezés [↑](#footnote-ref-21)
22. domain [↑](#footnote-ref-22)
23. entity modellek létrehozása kód írásával [↑](#footnote-ref-23)
24. egy weboldallal rendelkező webalkalmazás [↑](#footnote-ref-24)