# Táplálkozási tanácsadó alkalmazás

Projektmunkám célja egy olyan alkalmazás kifejlesztése, melynek segítségével a felhasználó céljának megfelelő teljes értékű étrendet kap. A megfelelő paraméterek megadását követően az alkalmazás kiszámolja tápanyagigényeit, majd ez alapján állítja össze az optimális étrendet, mely teljes egészében módosítható. A szükséges paraméterek a nem, kor, magasság, tömeg, napi aktivitás, valamint cél (erősödés/ fogyás/ fenntartás), mely adatok szükségesek a napi tápanyag- és energiaszükséglet megállapításához. Továbbá a felhasználó megadja étel preferenciáit is, a prezentált ételek 10-es skálán való értékelése alapján. A fő cél, hogy az ételek egy olyan halmazát prezentálja a program, mely kielégíti a napi tápanyagszükségletet, minimális költséggel jár, és maximális össz-értékeléssel rendelkezik. Kutatómunkámban először kitérek a motivációra, majd megvizsgálom, milyen megoldás létezik már a problémára. Ezt követően megvizsgálom a lehetséges platformokat, melyeken az implementáció felmerülhet. Végezetül megvizsgálom a kérdést dietetikai szempontból és megadom a specifikációt.

# Motiváció

Ma Magyarországon ez egyik legsúlyosabb népbetegség az elhízottság. Az ország 16 évesnél idősebb lakosságának 35,8%[[1]](#footnote-1)-a elhízott. Az ülőmunkát végző lakosság 41,9%-a nem végez rendszeres testmozgást. Az elhízott embereknél nagyobb hajlam van a rák kialakulására, valamint 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]](#footnote-2). Ezen megfontolásból döntöttem úgy, hogy segíteni kívánok az egészséges táplálkozás irányába elmozduló embereknek ezen ingyenes szolgáltatással.

# Bevezetés

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[[3]](#footnote-3). 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 ételeket személyes ízlése alapján. Tehát a kiegészítő cél a felhasználói értékelés maximalizálása a napi étrendben. A területen már számos kutatás történt, melyeket át kívánok tekinteni.

# A kettős-célú étrend probléma

Aynur Kaharam és H. Aydolu Seven munkájukban[[4]](#footnote-4) 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:

* a napi tápanyagszükségletetek megfelelő
* minimális napi költséggel rendelkezik
* 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 limitációit 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 fogásból álló étkezésből áll.

A kettős-célú diéta probléma hátizsák problémaként[[5]](#footnote-5) modellezhető. Az egymegkötéses hátizsák probléma a következőképpen definiálható:

A hátizsák problémában5 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 a diéta problémában számos különböző megkötésünk van. Mindemellett a problémánknak 2 célja van. Emiatt a étrend problémánk egy több-célú többdimenziós (azaz többmegkötéses) hátizsák problémaként definiálható.

# A genetikus algoritmus(GA)

A többdimenziós hátizsák probléma(MKP) többcélú optimalizációját és megoldását fogom ismertetni a következőkben.

## Háttérkutatás

A genetikus algoritmus[[6]](#footnote-6) 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 stringként 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[[7]](#footnote-7) genetikus algoritmusa kiválóbb megoldást biztosít más heurisztikákhoz viszonyítva. A genetikus algoritmusuk egy stabil állapotú GA, mely bináris string kódolást alkalmaz, populáció nagysága 100, bináris verseny kiválasztást, egységes crosszovert, és bit mutációt alkalmaz. Nem enged egyed duplikációt 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 állítódik 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[[8]](#footnote-8)() 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[[9]](#footnote-9) helyettesítő kettősség megközelítését alkalmazták, az egyes megkötések árnyékárait az MKP lineáris programozás(LP) 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 megszegést. 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[[10]](#footnote-10) javított genetikus algoritmusa nagyon hasonló Chu és Beasley megodásához, 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.[[11]](#footnote-11) Van olyan módszer, mely valamely kritériumot előnyben részesíti másikkal szemben, van amelyik mindegyik célt el kívánja érni, van amelyik kompromisszumra alapszik a kritériumok között.11 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.11 Tehát a végső megoldás az optimalizációt és döntési folyamatot követően jön létre. [[12]](#footnote-12) A döntéshozó preferenciájának megfelelően a többcélkitűzésű evolúciós algoritmuson[[13]](#footnote-13) alapuló többcélú probléma[[14]](#footnote-14) 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.

## A kettős célkitűzésű étrend probléma genetikus algoritmusa

Kahraman és Seven genetikus algoritmusa gyakorlatilag ugyanaz, mint Chu és Beasley vagy Raidl genetikus algoritmusa. Különbség az inicializáló, javító, és optimalizáló módszerekben van. Az inizializáló módszer a Chu és Beasley által használthoz képest egy javított módszer. A C\* módszer, mely Gottlieb munkájában[[15]](#footnote-15) van részletesen kifejtve. A különbség az, hogy a C\* módszerben minden elemet megpróbálunk a megoldásba tenni, ezzel biztosítva, hogy a megoldás a határ mentén marad. Ezzel szemben Kahraman és Seven nem használja a lineáris relaxációt a a javító, és optimalizációs rutinokhoz. A javító és optimalizáló rutinok nagyon hasonlóak az inicializációhoz: az inicializáció során az egyes megoldásokhoz generált permutációk javító és optimalizációs rutinban felhasználásra kerülnek. A javító módszerben, az elemek a véletlenszerű permutáció sorrendjében kivonásra kerülnek a megoldásból addíg, míg nincs megkötés megszegés.15 Továbbá az optimalizációs módszerben az egyes elemek a véletlenszerű permutáció sorrendjében hozzáadásra kerülnek a megoldáshoz, feltéve, hogy nincs megkötés megszegés. Tehát a létrejövő genetikus algoritmus egy stabil állapotú, egyén duplikációt nem engedő, bit string kódolású, bitenkénti mutációjú(1/n valószínűséggel), egységes crossover rekombinációjú (0,9 valószínűséggel), 100-as populáció méretű, 106 loop-al rendelkező genetikus algoritmus. Továbbá probléma-specifikus inicializáció, javító, és lokális optimalizációs módszerek kerülnek alkalmazásra.

Mivel a problémában több célkitűzés van, a fitnesz modell értékelése másként történik. A súlyozott összeg megközelítést alkalmazzák a több célkitűzéses optimalizációhoz. A megközelítés szerint több célkitűzést áttranszformálunk egyetlen célkitűzéssé, mely az összes célkitűzés lineáris függvénye.11 A célkitűzéseknek azonos típusúaknak kell lennie, tehát vagy minimalizációnak, vagy maxiamlizációnak kell lennie mindnek. Ellenben a problémában a két célkitűzés közül az egyik minimalizació, a másik maximalizació. Tehát a minimalizációt maximalizációvá transzformálják a az célfüggvény reciprokának segítségével. Az egyes célkitűzéseket megszorozzák a hozzárendelt súlyértékkel, majd összeadják őket. Ennek az összegnek az értéke 1 kell legyen. Továbbá annak érdekében, hogy megakadályozzák, hogy az egyes célkitűzések befolyásolják egymást, a célkitűzés értékeket normalizálni kell. Tehát a problémához a célkitűzések, és a végső fitnesz függvény a következő:

Hozzárendelt értékeket 0,6 - 0,4 -re vagy fordítva variálhatók, vagy alapesetben 0,5 értékűek.

# Felhasznált adathalmaz

A probléma megoldásánál feltételezzük, hogy ismerjük ételek egy halmazának tápanyagtartalmát és költségét. Továbbá feltételezzük, hogy ismerjük egy egyén napi energia- és tápanyagigényeit. Utóbbira a Táplálkozás fejezetben térek ki. Az egyes ételek tápanyagtartalmát illetően számos adatbázis fellelhető, köztük például az USDA Sztenderd Referencia Nemzeti Tápanyag Adatbázis, 21. kiadás[[16]](#footnote-16).

# Platform

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.[[17]](#footnote-17)

Android applikációt Kotlin, Java, illetve C++ nyelveken lehet írni. Az Android SDK[[18]](#footnote-18) segítségével tudjuk a forráskódot, más adatokkal és forrásokkal együtt lefordítani egy APK[[19]](#footnote-19)-ba. A .apk állomány tartalmazza az összes telepítéshez szükséges információt. Minden Android applikáció egy saját biztonsági homokozóban üzemel, mely a következő biztonsági funkciókkal rendelkezik:

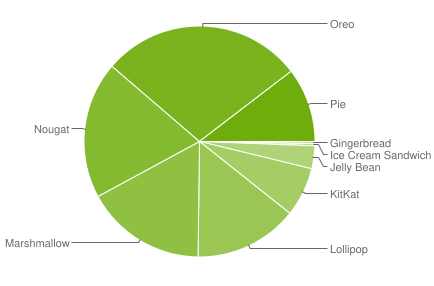
* Az Android operációs rendszer egy többfelhasználós Linux rendszer, melyben minden alkalmazás egy külön felhasználó.
* Alapesetben, a rendszer minden alkalmazáshoz hozzárendel egy egyedi Linux felhasználói azonosítót (melyet csak a rendszer használ, és ismeretlen az app számára). A rendszer beállítja 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.

## Az Android Stúdió

Android Stúdió egy integrált fejlesztői környezet, mely az Android fejlesztés de facto fejlesztő eszköze. Az Android Stúdiót speciálisan a Android fejlesztésre hozták létre. Fejlesztését és támogatását a Google biztosítja. A környezet magában foglal egy Android emulátort, melynek köszönhetően akár fizikai eszköz nélkül is tudunk fejleszteni. Ennek segítségével ki lehet próbálni bármilyen típusú eszközön az applikáció működését.

## Verziók

Az Android 1.0-t 2008-ban adták ki. Alig 11 év alatt 15 jelentős platformot adtak ki. A konvenció szerint minden új verzió egy édesség alapján, valamint abc rendben van elnevezve. Az Android fejlesztői oldalon[[20]](#footnote-20) megtalálható, hogy az Android felhasználók milyen arányban használják az egyes verziókat.

## Minimális SDK[[21]](#footnote-21)

De vajon milyen verziót válasszunk, amikor új applikációt tervezünk? Jogos gondolat lehetne a minimális, 1.0 verziót választani, melynek köszönhetően az – elméletben – összes lehető Android készülékkel kompatibilis lenne. Azonban sajnos ez nem járható út, hiszen rengeteg új technológia jelent meg az utóbbi 11 esztendőben, melyre maga a platform is természetszerűleg reagált. Példaként említhető a manapság egyre népszerűbb mobilos fizetés, mely NFC technológiát alkalmaz. Átalános cél az applikáció lehető legszélesebb körben való kompatibilitása. Azonban ennek ára van: új végrehajtási utak létrehozására van szükség az elavult vagy frissített API-k körül, vagy különböző UX tervezésére van szükség az egyes eszközökre. A teljesség igénye nélkül néhány új hardveres támogatás és funkció az új verziókhoz kötve: Home képernyő widgetek (Cupcake); több ujjas követés(Froyo); tablet (Honeycomb); Android Beam (Jellybean); Android TV, Auto, Wear (Lollipop); Pro Audio (Marshmallow). Egy optimális egyensúlyt kell létrehozni a széleskörű kompatibilitás és a ezen felhasználók költséges támogatása között.

## A Cél SDK

A minimális szoftverfejlesztő készlettel ellentétben a cél szoftverfejlesztő készlet nem egy felső korlát – csupán azt mutatja, hogy mely platformon teszteltük az applikációt. Egy meghatározott verziót célzó app a jövőben is kompatibilis lesz az új Android verzióval. A cél szoftverfejlesztő készletre csupán annak érdekében van szükség, hogyha egy új kiadásban egy jelentős változás történik a várt viselkedésben, az applikáció nem omlik össze a mobiltelefon frissítése következtében. Az Android Stúdió alapesetben a legfrissebb SDK-t célozza. Új applikáció fejlesztése esetén nem is lenne ésszerű más platformot beállítani. Továbbá, amikor az applikáció már kiadásra került, célszerű új Android verzió kiadása esetén az adott verzióra tesztelni és beállítani azt célplatformnak annak érdekében, hogy minél hamarabb kihasználhassuk az új verzió nyújtotta lehetőségeket.

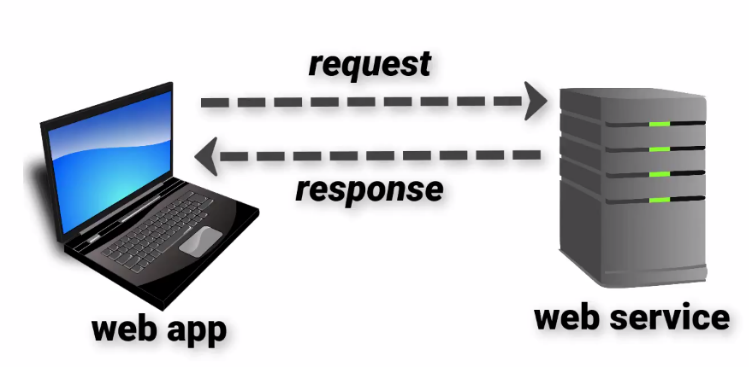
# Reszponzív Webalkalmazás

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[[22]](#footnote-22). 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.



## 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:

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.[[23]](#footnote-23)

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

A felhasználható platformok száma elég nagy, viszont 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 meg 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[[24]](#footnote-24) 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 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[[25]](#footnote-25)). 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(Például: 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.[[26]](#footnote-26)

# MVC tervezési minta

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[[27]](#footnote-27). 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 adatot, melyeknek köszönhetően 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 raktárhoz GitHubon 2012-ben[[28]](#footnote-28).

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

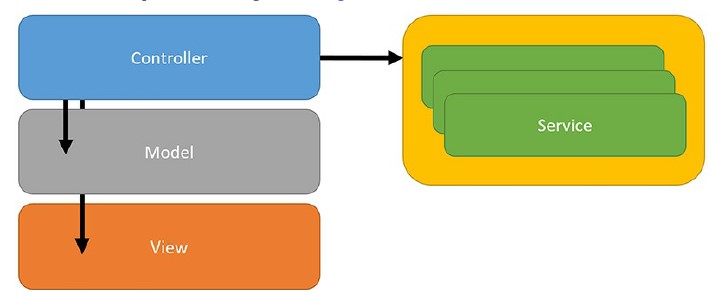
Eszköztárak, azonosító felületek, vásárlási kosarak, irányítópult bizonyos részei, és más hasonló komponensek az applikációkban sok esetben több vagy akár az összes oldalon is megjelennek. A DRY[[29]](#footnote-29) elvet figyelembe véve lehetőség van részleges nézetek létrehozására. Ezeket így több oldalon 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[[30]](#footnote-30) 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.[[31]](#footnote-31)

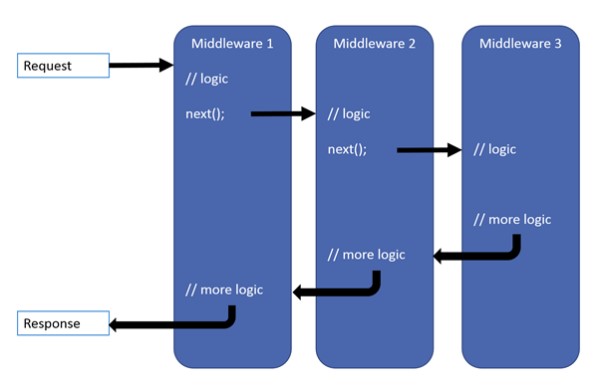
## Köztesréteg[[32]](#footnote-32)

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.

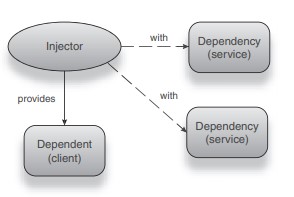
Az ASP.NET Core kérés szállítórendszer kérés delegáltak sorozatából áll, melyek egymás után kerülnek meghívásra. A következő diagramm demonstrálja az elképzelést. A végrehajtás a fekete nyilak mentén folyik.

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.[[33]](#footnote-33)

## Függőség beinjektálása

A függőség injektálás[[34]](#footnote-34) 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[[35]](#footnote-35) 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. [[36]](#footnote-36)

# Entity Framework

Az első és egyik legfontosabb lépés alkalmazásom tervezésénél az adatmodell megalkotása. Ennek elkészülését követően létrehozom a POCO[[37]](#footnote-37) objektumokat. Az objektumok leképezését a mögöttes adatbázisba az O/RM[[38]](#footnote-38) 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[[39]](#footnote-39) 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[[40]](#footnote-40). 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[[41]](#footnote-41)

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

# Táplálkozás

Sokszor halljuk a mondást: a fogyás egy nagyon egyszerű képlet alapján működik: bevitt kalória, szemben az elégetett kalóriákkal. Sajnos azonban a probléma mögött rengeteg tényező áll.

Természetesen nem létezik varázsformula a fogyásra, de a kutatók számos matematikai modellt alkottak annak érdekében, hogy megértsük hogyan veszít testünk a súlyából. Mint már azt bevezetőmben említettem, Magyarországon az elhízott egyének száma meghökkentő, több mint a lakosság egyharmada. A jelenségnek számos negatív hatása van, kezdve az egészségügyre rakódó többlettehertől, egészen az egyének mentális és fizikai egészségéig.

Fontosnak tartom leszögezni, hogy egy illető napi energiaszükségletének meghatározása nagyon nehéz feladat, melyet rengeteg tényező befolyásol. Az egyik legfontosabb fitnessz statisztika az anyagcsere gyorsasága pihenő állapotban[[42]](#footnote-42). A teljes napi energiafelhasználásunk[[43]](#footnote-43) legjelentősebb hozzájárulója ez az érték, ezért az RMR megemelése egy logikus lépés fogyás szempontjából. Például megpróbálhatunk izomtömeget növelni, hogy megemeljük az RMR értékünket körülbelül 7-8%-al vagy az edzés utáni többlet energiafogyasztásunkat[[44]](#footnote-44).

Egy másik fontos tényező, hogy a közepes mértékű edzés növelheti a kalóriafogyasztásunkat akár 383 kilokalóriával naponta[[45]](#footnote-45). Ez a többletkalóriafogyasztás oka lehet a többletenergiafogyasztásból adódó éhség csillapítása és az energiaraktárak visszatöltése, vagy pszichológiai eredetű: a jutalom realizálása.

Mindamellett, hogy az étvágyat kontrollálni szükséges, az éhség figyelmen kívül hagyása komoly hiba. Abban az esetben, ha éhségünk ellenére nem csillapítjuk azt, akár 20%-al is lecsökkenhet az RMR értékünk, mely megakadályozza a fogyást. Például egy 2000 kcal napi bevitellel rendelkező személy számára egy 20%-os RMR csökkenés 240-300 kcal-t jelent naponta, azaz 11-14 kilogrammot évente.

Látva ezt a rengeteg statisztikát, a matematikai számítás a hagyományos súlykezelés filozófiájának magját képezi. Habár több szempontot érdemes figyelembe venni a téma vizsgálatánál, mint a pszichológiai, érzelmi összetevőit a elhízottságnak valamint a szociális és kulturális környezet hatásait, nem lehet figyelmen kívül hagyni a formulák jelentőségét.

## A Formulák

Előszöris jegyezzük meg, hogy ezek a formulák mindamellett, hogy általánosságban valósnak tekinthetők, potenciálisan komoly hibaaránnyal működnek, mind elhízott, mind nem elhízott személyek esetén.

|  |  |  |  |
| --- | --- | --- | --- |
|  | Pontos becslés | Alul becslés | Túlbecslés |
| **Harris-Benedict** |  | | |
| Nem elhízott(BMI alapján) | 45-81% | 23% | 42% |
| Elhízott | 36-64% | 35% | 43% |
| **Miffin-St. Jeor** |  | | |
| Nem elhízott(BMI alapján) | 82% | 18% | 15% |
| Elhízott | 70% | 20% | 15% |

A Harris-Benedict egyenlet, melyet 1919-ben alkottak sokáig sztenderdnek minősült, mely akár 81%-ban pontos nem elhízott egyéneknél, viszont akár 42%-ban túlbecsülhet. A Mifflin-St. Jeor egyenlet, melyet 1990-ben alkottak, kisebb hibarátával működik, mely a legpontosabb matematikai formulának számít az Egyesült Államok populációjában vizsgálva[[46]](#footnote-46). Komoly korlátja ezen formuláknak, hogy nem veszi figyelembe a sovány izomtömeg és zsír arányát, mely nagymértékben befolyásolja az RMR értékét. Ezzel szemben a Katch-MCArdle formula felhasználja a sovány izomtömeg arányát, mely növeli az egyenlet pontosságát. Viszont így a számítás függ a pontos testzsír arány meghatározásától.

Tehát egy személy energiafelhasználásának komponensei:

TDEE = RMR + TEF[[47]](#footnote-47) + TEPA[[48]](#footnote-48)

Ezen komponensek a:

* TDEE a teljes napi energiafelhasználás.
* RMR a nyugalmi anyagcsere mértéke, azaz a pihenés közben felhasznált energia mértéke. Az RMR jellemzően 60-70%-át teszi ki a TDEE-nek.
* TEF az élelmiszer termikus hatása, azaz a élelem megrágásának, lenyelésének, megemésztésének, és elraktározásának költsége. Jellemzően a TDEE 10%-át teszi ki.
* TEPA a fizikai tevékenység termikus hatása, azaz az edzés, fizikai tevékenység, valamint a nem edzéshez köthető aktivitás energiaigénye. A TEPA jellemzően a TDEE 15-30%-át teszi ki.

## A TDEE kiszámolása

Mind a Harris-Benedict formula, mind Miffin-St. Jeor formula 2 lépésben számolja ki TDEE értékét:

1. RMR kiszámítása
2. RMR megszorzása az aktivitási faktor pontszámmal, melyet a TEF és a TEPA határoz meg.

|  |  |
| --- | --- |
| **Harris-Benedict** | |
| Férfiak | RMR = 88,362 + (13,397 \* kg) + (4,799 \* cm) – (5,677 \* kor) |
| Nők | RMR = 447,593 + (9,247\* kg) + (3,098\* cm) – (4,33 \* kor) |
| TDEE = RMR \* aktivitási faktor pontszám | |
| **Miffin-St. Jeor** | |
| Férfiak | RMR = (9,99 \* kg) + (6,25 \* cm) – (4,92 \* kor) + 5 |
| Nők | RMR = (9,99 \* kg) + (6,25 \* cm) – (4,92 \* kor) - 161 |
| TDEE = RMR \* aktivitási faktor pontszám | |

### 2.lépes: Az aktivitási faktor pontszám kiszámítása

Ez a lépés határozza meg a fizikai aktivitásból(TEPA) és élelmiszer emésztéséből(TEF) adódó további kalóriafelhasználást[[49]](#footnote-49). Számos másik számtalanszámítási mód létezik, melyekhez szükség lenne az adott személy többnapi 24 órás rutinjának monitorozására. Ezekre praktikus okokból nem térek ki, hiszen ez nem elvárható a táplálkozási tanácsadó alkalmazásom egy tipikus felhasználójától.

A kutatók meghatároztak az aktivitási szorzók egy készletét, melyeket Katch-McArdle szorzóknak nevezünk[[50]](#footnote-50). Ahhoz, hogy kiszámolhassuk a teljes napi energiafelhasználásunkat (TDEE), a következő szorzókat kell alkalmazni a BMR értékére:

* Ülő(kevés vagy semennyi edzés és ülőmunka) = 1,2
* Enyhén aktív(enyhe edzés 1-3 alkalommal hetente) = 1,375
* Közepesen aktív(közepes edzés 3-5 alkalommal hetente) = 1,55
* Nagyon aktív(Erős edzés 6-7 alkalommal hetente) = 1,725
* Kiemelkedően aktív(nagyon erős edzés, fizikai munka, 2 edzés naponta) = 1,9

Fontos megjegyzés: ezek a számítások egy becslést adnak a napi kalóriaszükségletre. Tehát a valós TDEE lehet magasabb és alacsonyabb valamivel. Mindemellett egy jó kiindulási alapot tud biztosítani egy táplálkozási terv kialakítása esetén.

## A TDEE manipulálása izomtömegnöveléshez és fogyáshoz

Tehát amint kiszámoltuk az egyén TDEE értékét, ebből kiindulva ki tudunk tűzni egy napi kalória célt. De hogyan kell módosítani a TDEE értékét? Rengeteg információ terjed arról, hogy vajon mi az optimális módja a test átformálásának. Két dolog azonban biztos:

* Abban az esetben, ha fogyni szeretnénk (pontosabban zsírtömeget veszíteni), kevesebb kalóriát kell bevinnünk a TDEE értékünknél. Ennek hatására testünket arra kényszerítjük, hogy energiát vonjon el zsírsejtjeinktől, hogy kompenzálni tudja a szükséges kalóriákat. Ezt metabolikus folyamatnak nevezzük. Ha ezt konzisztens módon tesszük, veszíteni fogunk a zsírtömegünkből.
* Ha célunk az izomtömeg növelése, értelemszerűleg a TDEE értékünknél több kalóriát kell bevinnünk. Ha tömeget szeretnénk növelni, kalóriatöbbletes étrendet kell folytatnunk. Ha mindezt erős intenzitású erősítő edzés kíséretében tesszük, követve a progresszív túlterhelés elvét, testünk izomzatot fog építeni. Ezt anabolikus folyamatnak nevezzük.

## Mindez a gyakorlatban

A felhasználó számára lehetőséget fogok biztosítani a számára ideális mértékű kalóriatöbblet vagy kalóriadeficit kiválasztására (Max. TDEE +- 25%). Így megkapjuk az adott egyén számára ideális napi kalóriabevitel értékét. A következő lépésben el kell döntenünk, hogy ezen energiamennyiséget milyen arányban alkossák a különböző makró tápanyagok.

## Makró tápanyagok

Köztudottan három nagy csoportját különböztetjük meg a makró tápanyagoknak, melyek minden táplálékunkat alkotják: fehérjék, szénhidrátok, és a lipidek (melyet a köznyelv zsírnak nevez). Szervezetünk számára tipikusan a lipidek és szénhidrátok biztosítják a szükséges energiát, a fehérjék pedig testünk építőköveinek tekinthetők. Emiatt is érdemes a táplálkozási terv összeállításánál elsőként a szükséges mennyiségű fehérje kiszámítása, ezt követően megnézzünk a szükséges zsírmennyiséget, végül pedig a napi kalóriabevitel fennmaradó részéből ki tudjuk számolni a szükséges szénhidrátmennyiséget.

Peter W.R. Lemon tanulmánya alapján egy intenzív erősítő edzést folytató személy számára a napi szükséges fehérjemennyiség körülbelül 1,6-1,8 gramm testsúly kilogrammonként. Az edzést nem folytató egyének számára ez a szám körülbelül ennek az értéknek a fele(0,8g/kg).[[51]](#footnote-51) Viszont amint azt Lemon tanulmányban is leírja a gyakorlat sokszor mást mutat, az atléták ennél lényegesen nagyobb mennyiségű fehérjét fogyasztanak. Emiatt az intenzív edzést folytató egyének számára a 2g/kg értéket fogom alkalmazni, mely kicsit több mint 10%-al több, mint a Lemon által ajánlott.

A lipidek ugyancsak létfontosságú tápanyagok szervezetünk számára. Sajnos a múltban számtalan tanulmány egyenesen egészségtelennek és károsnak kiáltotta ki a zsírt, mely számos tévképzethez vezetett és vezet ma is. Mint általában, itt is a változatos étrendet, és a minél kevésbé feldolgozott élelmiszereket ajánlott előnyben részesíteni. Néhány évtizeddel ezelőtt a dietetikusok között erős konszenzus volt az alacsony zsírtartalmú étrend egészségügyi előnyével kapcsolatban. Azonban a 2015-2020 Egyesült Államok Dietétikai Irányelve már nem tartalmaz felső határértéket a napi zsírbevitelre.[[52]](#footnote-52) Így a legfontosabb szempont, amit érdemes figyelembe venni, a teljes napi kalóriabevitel. Alkalmazásom felhasználója számára lehetőséget biztosítok választani a magas vagy közepes zsírtartalmú étrend lehetőségeiből, s ennek megfelelően alakítom táplálkozási tervét. A magas zsírtartalmú vagy ketogén étrendben jellemzően a napi bevitt kalóriamennyiség 50-75%-a zsírból áll. A közepes zsírbevitel esetén az energiabevitel tipikusan 35-40%-át alkotja a zsír. Ezt mediterrán diétának is nevezzük.

Tekintve, hogy jelenleg már adott a szükséges fehérje- és zsírbevitel, egyszerűen kiszámolható a fennmaradó kalóriamennyiségből az ajánlott napi szénhidrátmennyiség.[[53]](#footnote-53)

# Specifikáció

## Személyre szabott étrend összeállítása

A felhasználó paraméterei alapján való étrend összeállítás funkció a következő. A felhasználó megnyitja a programot, majd a regisztráció opciót kiválasztva megadja a következő személyes paramétereket:

* Email
* Jelszó
* Név
* Kor
* Tömeg(kg)
* Nem
* Magasság(cm)
* Napi átlagos alvás(óra)
* Heti testmozgás(óra)
* Testzsírszázalék
* Edzési gyakorlat
* Étrend preferencia
* Cél(erősödés/fogyás)

A program ezt követően a legfontosabb paramétereket fölhasználva kiszámolja az egyén napi energiaszükségletét, valamint makrótápanyagigényét. A kiszámolt adatokból összeállít egy személyre szabott étrendet, melyet azonnal megjelenít a „táplálkozási terv” nézetben. A felhasználó továbbá itt láthatja a kiszámolt tápértékszükségleteit is.

## Étrend testre szabása

A felhasználó saját étrendjét módosíthatja. Akár törölhet étrendjéből, vagy hozzáadhat új ételeket. Két nézetben láthatja az ételeket: „A hét ételei”, valamint az „Összes étel”. Amint a felhasználó eldöntötte, mely ételt kívánja hozzáadni étrendjéhez, legördülő menüben kiválasztja a napot, majd az „Étrendhez hozzáadás” gomb lenyomása után a módosított étrendjének nézete jelenik meg.

# Összegzés

Tehát az alkalmazást .NET keretrendszerben ASP .NET Core segítségével fogom lefejleszteni. Mobilon való megjeleníthetőség érdekében szem előtt tartom a rezponzív megjelenítését. Általános célom, hogy a felhasználó egy teljes körű, személyre szabott étrendet legyen képes összeállítani, mindössze néhány egyszerű lépéssel. Továbbá információval látom el a személyes tápanyag szükségleteit illetően.

1. <https://www.ksh.hu/docs/hun/xftp/stattukor/egeszsegallapot1617.pdf> [↑](#footnote-ref-1)
2. <https://premiumorvos.hu/files/Dr.HalmyLszlcikke.pdf> [↑](#footnote-ref-2)
3. <https://www.jstor.org/stable/1231810?seq=1> [↑](#footnote-ref-3)
4. <http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.511.2965&rep=rep1&type=pdf> [↑](#footnote-ref-4)
5. <http://bioinfo.ict.ac.cn/~dbu/AlgorithmCourses/Lectures/Lec6-Knapsack-Problems-Kellerer2003.pdf> [↑](#footnote-ref-5)
6. <https://link.springer.com/book/10.1007%2F978-3-662-44874-8> [↑](#footnote-ref-6)
7. <https://link.springer.com/article/10.1023/A:1009642405419> [↑](#footnote-ref-7)
8. <https://link.springer.com/chapter/10.1007/978-3-642-46618-2_17> [↑](#footnote-ref-8)
9. <https://onlinelibrary.wiley.com/doi/abs/10.1002/1520-6750(198704)34:2%3C161::AID-NAV3220340203%3E3.0.CO;2-A> [↑](#footnote-ref-9)
10. <https://ieeexplore.ieee.org/abstract/document/699502> [↑](#footnote-ref-10)
11. <https://link.springer.com/article/10.1007/BF03325101> [↑](#footnote-ref-11)
12. D. A. V. Veldhuizen, G. B. Lamont: Multiobjective Evolutionary Algorithms: Analyzing the State-of-the-Art, Massachusetts Institute of Technology Evolutionary Computation, 8. fejezet , 125-147., 200. oldal [↑](#footnote-ref-12)
13. Multiobjective Evolutionary Algorithm (MOEA) [↑](#footnote-ref-13)
14. Multiobjective Problem (MOP) [↑](#footnote-ref-14)
15. <https://ci.nii.ac.jp/naid/10019556432/> [↑](#footnote-ref-15)
16. <https://www.ars.usda.gov/research/publications/publication/?seqNo115=199178> [↑](#footnote-ref-16)
17. <https://en.wikipedia.org/wiki/Android_(operating_system)> [↑](#footnote-ref-17)
18. Software Development Kit – Szoftverfejlesztő készlet [↑](#footnote-ref-18)
19. Android Package File – telepítőfájlok (szabványos JAR archívum, egyéb metaadatokkal) [↑](#footnote-ref-19)
20. https://developer.android.com/about/dashboards/index.html#Platform [↑](#footnote-ref-20)
21. Software Development Kit - SDK [↑](#footnote-ref-21)
22. <https://en.wikipedia.org/wiki/Web_application> [↑](#footnote-ref-22)
23. <https://www.oreilly.com/library/view/rest-api-design/9781449317904/ch01.html> [↑](#footnote-ref-23)
24. angolul: Model-View-Controller, MVC [↑](#footnote-ref-24)
25. <https://www.techempower.com/benchmarks/#section=data-r18&hw=ph&test=plaintext> [↑](#footnote-ref-25)
26. <https://dotnet.microsoft.com/learn/aspnet/what-is-aspnet> [↑](#footnote-ref-26)
27. angolul: separation of concerns [↑](#footnote-ref-27)
28. GitHub reinstates Russian who hacked site to expose flaw, John Leyden, March 5, 2012, *http://www.theregister.co.uk*. [↑](#footnote-ref-28)
29. Don’t Reapeat Yourslf – ne ismételd önmagadat [↑](#footnote-ref-29)
30. action [↑](#footnote-ref-30)
31. James Chambers, David\_Paquette, Simon Timms: ASP Application Development [↑](#footnote-ref-31)
32. angolul: middleware [↑](#footnote-ref-32)
33. <https://docs.microsoft.com/en-us/aspnet/core/fundamentals/middleware/?view=aspnetcore-3.1> [↑](#footnote-ref-33)
34. angolul: Dependency Injection - DI [↑](#footnote-ref-34)
35. angolul: Inversion of Controll - IoC [↑](#footnote-ref-35)
36. 2009, Dhanji R. Prasanna – Dependency Injection [↑](#footnote-ref-36)
37. Plain Old CLR Object – külső függőségekkel nem rendelkező objektum [↑](#footnote-ref-37)
38. Object Relational Mapper – Objektum relációs leképező [↑](#footnote-ref-38)
39. Language Integrated Query – Integrált nyelvi lekérdezés [↑](#footnote-ref-39)
40. domain [↑](#footnote-ref-40)
41. entity modellek létrehozása kód írásával [↑](#footnote-ref-41)
42. RMR – Resting Metabbolic Rate [↑](#footnote-ref-42)
43. TDEE – Total Daily Energy Expenditure [↑](#footnote-ref-43)
44. EPOC - Excess Postexercise Energy Consumption [↑](#footnote-ref-44)
45. Whybrow et al. 2008 [↑](#footnote-ref-45)
46. Weijs 2008 [↑](#footnote-ref-46)
47. Thermic Effect of Food [↑](#footnote-ref-47)
48. Thermic Effect of Physical Activity [↑](#footnote-ref-48)
49. <https://www.ideafit.com/personal-training/the-energy-equation/> [↑](#footnote-ref-49)
50. <https://completehumanperformance.com/2013/10/08/calorie-needs/> [↑](#footnote-ref-50)
51. <https://www.researchgate.net/profile/Peter_Lemon/publication/12300829_Beyond_the_Zone_Protein_Needs_of_Active_Individuals/links/58b0513992851cf7ae8ba414/Beyond-the-Zone-Protein-Needs-of-Active-Individuals.pdf> [↑](#footnote-ref-51)
52. <https://health.gov/dietaryguidelines/2015/> [↑](#footnote-ref-52)
53. Az egyes makrotápanyagok kalóriatartalma: zsír – 9kcal/g, szénhidrát – 4kcal/g, fehérje – 4kcal/g [↑](#footnote-ref-53)