**Szegedi Tudományegyetem**

Informatikai Intézet

SZAKDOLGOZAT

Somogyvári Hanna

2024

**Szegedi Tudományegyetem**

Informatikai Intézet

Nyugtafelosztó

Szakdolgozat

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Készítette: |  | Témavezető: |  |
|  | Somogyvári Hanna |  | Dr. Bilicki Vilmos |  |
|  | programtervező informatikus szakos hallgató |  | egyetemi adjunktus |  |

Szeged

2024

# Feladatkiírás

A szakdolgozat keretein belül a cél egy olyan Angular[[1]](#Angular) alkalmazás elkészítése, ami nyugták végösszegének megadott résztvevők közötti felosztását teszi lehetővé. Emellett képes a feltöltött adatokat tárolni, listázni is.

A felhasználók képesek a vásárlás résztvevőit és őket tartalmazó csoportokat létrehozni. Ezeket a nyugta egyes termékeihez tudják rendelni. Az alkalmazás ez alapján kiszámolja az összeget, amit a termékek árainak megfelelően egy-egy személynek fizetnie kell.

A nyugtákat létre lehet hozni az adatok kézzel való begépelésével, valamint képfeltöltéssel is, ahonnan a szöveget az alkalmazás egy űrlapba illeszti. Az űrlap ezután szerkeszthető, valamint Excelbe vagy JSON formátumban is kiexportálható.

Az alkalmazás grafikus felhasználói felülete könnyű kezelhetőséget biztosít, és a reszponzív oldalak széleskörű eszközökön való elérhetőséget tesznek lehetővé.

# Tartalmi összefoglaló

A téma megnevezése:

Egy Angular[[1]](#Angular) keretrendszerben készült alkalmazás, ami nyugták végösszegének felosztását teszi lehetővé.

A megadott feladat megfogalmazása:

Az alkalmazás egy képként feltöltött vagy manuálisan létrehozott nyugtát képes felosztani résztvevők, csoportok között. Ezt a felhasználó a résztvevők és csoportok előzetes létrehozásával tudja megtenni, akiket utána a nyugta egyes termékeihez rendelhet. Az alkalmazás célja, hogy a megadottak alapján kiszámolja a végösszeget, és az adatokat tárolja.

A megoldási mód:

Az Nyugtafelosztó specifikációja után megterveztem az alkalmazást. Ez a fázis magába foglalja a piackutatást, a diagramok elkészítését és a használt technológiák alaposabb megismerését. A fejlesztést Angular[[1]](#Angular) keretrendszerben valósítottam meg, az adatok tárolásához pedig a Google Firebase[[2]](#GoogleFirebase) szolgáltatást használtam. A folyamatot az alkalmazás ellenőrzésével és tesztelésével zártam.

Alkalmazott eszközök, módszerek:

A diagramokat a draw.io[[3]](#drawio) weboldalon készítettem. A Visual Studio Code[[4]](#VisualStudioCode) kódszerkesztőben fejlesztettem Angular[[1]](#Angular) keretrendszer használatával. Emellett a CSS szintaxisához hasonló, de egyszerűbben használható Sass[[5]](#SCSS) CSS előfeldolgozót alkalmaztam SCSS[[5]](#SCSS) szintaxissal a stíluslapok elkészítéséhez. A GitLab[[6]](#GitLab) platformot használtam egyaránt verziókövetéshez és a fejlesztés feladatainak, mérföldköveinek kezeléséhez. Az adatokat a Google Firebase[[2]](#GoogleFirebase) szolgáltatás segítségével tároltam. A képen látható szöveg felismerését a Tesseract.js[[7]](#TesseractJs) könyvtárral valósítottam meg.

A fejlesztési folyamathoz az iteratív modellt választottam: először vázlatosan megterveztem az alkalmazást, ezt követően egy-két hetes iterációkra meghatároztam a feladataimat, amiket ez idő alatt részletesebben kidolgoztam és megvalósítottam.

**Elért eredmények:**

A kitűzött célokat sikeresen megvalósítottam, a terveknek megfelelő alkalmazást készítettem. Emellett mélyebb tudást szereztem az elkészítés során használt eszközökről.

**Kulcsszavak:**

Angular[[1]](#Angular), Google Firebase[[2]](#GoogleFirebase), adatkezelés, szövegfelismerés

# Tartalomjegyzék

[Feladatkiírás 3](#_Toc165329362)

[Tartalmi összefoglaló 4](#_Toc165329363)

[Tartalomjegyzék 5](#_Toc165329364)

[MOTIVÁCIÓ 9](#_Toc165329365)

[1. Területi áttekintés 10](#_Toc165329366)

[1.1. Alkalmazások nyugták tárolásához 11](#_Toc165329367)

[1.1.1. Zoho Expense 11](#_Toc165329368)

[1.1.2. Verify 11](#_Toc165329369)

[1.1.3. Expensify 11](#_Toc165329370)

[1.1.4. Shoeboxed 12](#_Toc165329371)

[1.1.5. Dext Prepare 12](#_Toc165329372)

[1.1.6. Invoice Maker 12](#_Toc165329373)

[1.2. Alkalmazások nyugták felosztásához 13](#_Toc165329374)

[1.2.1. EveryoneAllin 13](#_Toc165329375)

[1.2.2. Best Bill Splitter 13](#_Toc165329376)

[1.2.3. Kittysplit 14](#_Toc165329377)

[2. FUNKCIONÁLIS SPECIFIKÁCIÓ 15](#_Toc165329378)

[2.1. Autentikáció 16](#_Toc165329379)

[2.1.1. Regisztráció 16](#_Toc165329380)

[2.1.2. Bejelentkezés 16](#_Toc165329381)

[2.2. Profil 17](#_Toc165329382)

[2.3. Résztvevők 17](#_Toc165329383)

[2.3.1. Résztvevők listázása 18](#_Toc165329384)

[2.3.2. Résztvevők létrehozása 18](#_Toc165329385)

[2.3.3. Résztvevők részletes oldala 18](#_Toc165329386)

[2.4. Csoportok 19](#_Toc165329387)

[2.4.1. Csoportok listázása 19](#_Toc165329388)

[2.4.2. Csoportok létrehozása 19](#_Toc165329389)

[2.4.3. Csoportok részletes oldala 19](#_Toc165329390)

[2.5. Nyugták 20](#_Toc165329391)

[2.5.1. Nyugták listázása 20](#_Toc165329392)

[2.6. Nyugták létrehozása 20](#_Toc165329393)

[2.6.1. Képfeltöltés 20](#_Toc165329394)

[2.6.2. Új nyugta 21](#_Toc165329395)

[2.6.3. Csoportok, résztvevők megadása 21](#_Toc165329396)

[2.6.4. Termékek hozzáadása 21](#_Toc165329397)

[2.7. Nyugták részletes oldala 22](#_Toc165329398)

[2.7.1. Fejenkénti végösszeg 22](#_Toc165329399)

[2.7.2. Kiexportálás 22](#_Toc165329400)

[2.7.3. Nyugták szerkesztése, törlése 23](#_Toc165329401)

[2.8. Nyugták vendég felhasználóknak 23](#_Toc165329402)

[3. FELHASZNÁLT TECHNOLÓGIÁK 24](#_Toc165329403)

[3.1. Angular 24](#_Toc165329404)

[3.1.1. Angular Material 24](#_Toc165329405)

[3.1.2. AngularFire 24](#_Toc165329406)

[3.1.3. Ngx Image Cropper 24](#_Toc165329407)

[3.2. Google Firebase 25](#_Toc165329408)

[3.2.1. Authentication 25](#_Toc165329409)

[3.2.2. Cloud Firestore 25](#_Toc165329410)

[3.2.3. Cloud Storage 25](#_Toc165329411)

[3.2.4. Hosting 25](#_Toc165329412)

[3.3. Tesseract.js 25](#_Toc165329413)

[3.4. Sass 26](#_Toc165329414)

[3.5. Egyebek 26](#_Toc165329415)

[4. Architektúra 27](#_Toc165329416)

[4.1. Frontend 27](#_Toc165329417)

[4.2. Backend 27](#_Toc165329418)

[5. Belső felépítés 28](#_Toc165329419)

[5.1. Modulok 28](#_Toc165329420)

[5.2. Interfészek 29](#_Toc165329421)

[5.3. Szolgáltatások 30](#_Toc165329422)

[5.4. Routing 31](#_Toc165329423)

[6. Biztonság 33](#_Toc165329424)

[7. Adatmodell 36](#_Toc165329425)

[7.1. Pénznem 36](#_Toc165329426)

[7.2. Felhasználó 37](#_Toc165329427)

[7.3. Profilkép 37](#_Toc165329428)

[7.4. Résztvevő 38](#_Toc165329429)

[7.5. Csoport 38](#_Toc165329430)

[7.6. Termék 38](#_Toc165329431)

[7.7. Nyugta 39](#_Toc165329432)

[8. A rendszer magasszintű folyamatai, működése 40](#_Toc165329433)

[9. Fontosabb kódrészletek ismertetése 44](#_Toc165329434)

[9.1. Regisztráció 44](#_Toc165329435)

[9.2. Csoportok létrehozása 45](#_Toc165329436)

[9.3. Nyugta beolvasása képről 46](#_Toc165329437)

[9.4. Nyugta szerkesztése 49](#_Toc165329438)

[9.5. Nyugta részletes oldala 51](#_Toc165329439)

[10. Tapasztalatok, továbbfejlesztési lehetőségek 55](#_Toc165329440)

[10.1. Alkalmazás megtervezése 55](#_Toc165329441)

[10.2. Ütermterv, feladatok 55](#_Toc165329442)

[10.3. Szakmai ismeretek 55](#_Toc165329443)

[10.4. Továbbfejlesztési lehetőségek 56](#_Toc165329444)

[10.5. Összegzés 57](#_Toc165329445)

[Irodalomjegyzék 58](#_Toc165329446)

[Nyilatkozat 59](#_Toc165329447)

[Köszönetnyilvánítás 60](#_Toc165329448)

# MOTIVÁCIÓ

Többszemélyes bevásárlások során, amikor a boltban egy személy fizet, és az összeget a résztvevők utólag osztják fel egymás között, gyakran okoz problémát az egy-egy főre jutó végösszeg kiszámítása. A nyugták rendezetlenek és hosszúra nyúlhatnak, ami sokáig tartó fejtörést okoz, főleg, ha bizonyos termékek csak a csoport egy kisebb részéhez tartoznak. A papíralapú blokkok elkeveredhetnek, a kézzel végzett számításokba akár hiba is csúszhat.

Erre, és még sok hasonló problémára nyújt megoldást a Nyugtafelosztó. Az alkalmazás a blokkról feltöltött képről olvassa ki az adatokat, így nem szükséges azokat kézzel begépelni. Az így elkészült listaelemek mellé a felhasználó hozzárendelheti a vásárlás egyes résztvevőit.

Csoportok létrehozására is van lehetőség, amire leginkább akkor lehet szükség, ha a résztvevők több termékért csak bizonyos személyek fizetnek. Például ilyen lehet, amikor tofut csak a vegetáriánusok, gluténmentes kenyeret pedig csak a lisztérzékenyek vesznek. A csoportoknak köszönhetően a felhasználónak nem szükséges minden esetben felsorolni az összes személyt, aki az adott termékért fizet, így gyorsabban és egyszerűbben tudja elvégezni a felosztást.

A Nyugtafelosztó ezek után kiszámítja a személyenkénti végösszeget, így megkönnyíti a nyugta felosztását, lerövidíti az ezzel töltött időt, és megszünteti a kézi számolásból eredő hibákat.

# 1. Területi áttekintés

A piackutatás során a célom olyan alkalmazások megtalálása volt, amelyek kapcsolódnak a Nyugtafelosztó fontosabb funkcióihoz. Az elsődleges szempont a nyugták tárolása, valamint a végösszeg kiszámítása és felosztása volt. Emellett szövegfelismerésre és résztvevők hozzáadására lehetőséget adó alkalmazásokat is kerestem.

A talált alkalmazásokat az eredmények alapján két csoportra osztottam. Az első tagjai a feltöltött nyugták tárolására fókuszálnak, ide tartozik a Verify[[9]](#Verify) és a Shoeboxed[[11]](#Shoeboxed) alkalmazás. A második csoportba pedig azokat az oldalakat soroltam, amelyek célja a végösszeg felosztása, mint például az EveryoneAllin[[14]](#EveryoneAllin) és a Best Bill Splitter[[15]](#BestBillSplitter).

A kutatás során felmért alkalmazások teljes listáját és funkcióit az 1.1. ábra mutatja. Ezeknek az alkalmazásoknak köszönhetően sikerült felmérnem, hogy a Nyugtafelosztónak milyen felhasználói igényeket kell kielégítenie. Ötleteket adtak és példát mutattak a tervezett funkciók megvalósításához. Arra is rávilágítottak, hogy milyen hiányosságokat tudok orvosolni a fejlesztés során, és teljesebb képet kaptam az elérendő céljaimhoz.

A képen szöveg, képernyőkép, szám, tér látható

Automatikusan generált leírás

1.1. ábra – Piackutatás

## 1.1. Alkalmazások nyugták tárolásához

A Nyugtafelosztó főbb funkciói közé tartozik a nyugták szkennelése, és az így keletkezett adatok tárolása, listázása. A piackutatás során több olyan alkalmazást is találtam, amelyek ezeket megvalósítják.

A nyugták feltöltésére több különböző megvalósítást felfedeztem, és olyan példákat is láttam, ahol a digitalizált számla kiexportálására is lehetőség volt.

### 1.1.1. Zoho Expense[[8]](#ZohoExpenses)

A Zoho Expense nem csupán nyugták kezelésével foglalkozik, de mivel ez áll közel a szakdolgozatom témájához, csak az ehhez kapcsolódó funkcióit fogom részletezni. Az alkalmazás használatáért fizetni kell, de ingyenesen kipróbálható 14 napra, valamit demó is igényelhető.

Többféle lehetőség is van a nyugták feltöltésére: beszkennelhetőek telefon kamerájával, e-mailben továbbíthatók az alkalmazásra, valamint felhőről vagy a fájlkezelőből is feltölthetőek. A Nyugtafelosztóban csak a fájlként feltöltést tervezem megvalósítani, de a Zoho Expense[[8]](#ZohoExpenses) megmutatta, hogy nem csak ez az egy mód létezik.

A feltöltött nyugtákat fel lehet osztani, de nem résztvevők hozzáadásával, hanem például a százalék vagy a vásárlás napjának megadásával.

### 1.1.2. Verify[[9]](#Verify)

Ennek az alkalmazásnak az egyik erőssége a képi szövegfelismerés. A feltöltött nyugtáról az adatokat egy táblázatba rendezi, ami szerkeszthető, elmenthető, és később megtekinthető. A beolvasott információk kiexportálására is lehetőséget ad, JSON vagy OCR szöveg formátumban.

### 1.1.3. Expensify[[10]](#Expensify)

Az Expensify[[10]](#Expensify), a Nyugtafelosztóhoz hasonlóan, lehetőséget ad képek szkennelésére és manuális létrehozásra. Viszont az alkalmazás hátránya, hogy csak a nyugta nevét és végösszegét tárolja, valamint a szövegfelismerés sok időt vesz igénybe.

### 1.1.4. Shoeboxed[[11]](#Shoeboxed)

Az egyik legérdekesebb ebben az alkalmazásban, hogy a nyugták hagyományos feltöltési módjai (telefonos szkennelés, e-mail, fájlfeltöltés) mellett a felhasználók elpostázhatják őket a Shoeboxed[[11]](#Shoeboxed) létesítményébe, ahol az alkalmazottak szkennelik be őket.

### 1.1.5. Dext Prepare[[12]](#DextPrepare)

A Dext Prepare[[12]](#DextPrepare) is képes képről felismerni a nyugta szövegét, legyen az telefonról, e-mailből vagy fájlként feltöltött. Ezután az alkalmazás kiszűri a lényeges információt, és feltölti a rendszerbe. Az adatokat 7 évig tárolják, bankszintű titkosítással.

Arra is lehetőséget nyújt, hogy a felhasználó szinkronizálja a Dext Prepare[[12]](#DextPrepare)-t másik könyvelési alkalmazásokkal, mint az IRIS KashFlow[[17]](#IRIS) és xero[[18]](#xero). Így az itt tárolt adatok a szinkronizált alkalmazásokra is automatikusan feltöltésre kerülnek.

A nyugtákból készült riportokat ki is lehet exportálni, CSV és PDF formátumban, valamint e-mailben is továbbíthatók.

### 1.1.6. Invoice Maker[[13]](#InvoiceMaker)

Az Invoice Maker[[13]](#InvoiceMaker), mint ahogy arra a neve is utal, főként nyugták készítésével foglalkozik. A Nyugtafelosztóval ellentétben itt még nem létezik nyugta, amivel számolni kell, hanem a felhasználó készíti el őket az ügyfelei számára.

A különböző célok ellenére a két alkalmazásban találhatók hasonlóságok, mégpedig az oldal Incvoice Calculator[[13]](#InvoiceMaker) részén. Itt manuálisan lehet megadni a nyugtán szereplő termékeket, ami alapján a végösszeg kiszámításra kerül.

Ezután az elkészített nyugta továbbítható az Invoice Generatorra[[13]](#InvoiceMaker), ami segítségével az adatok megszokott nyugta formátumba szerkeszthetők, például cím, dátum és logó megadásával. A folyamat leegyszerűsítése érdekében az alkalmazás személyre szabható sablonokat is biztosít a nyugtákhoz, amiket a felhasználó letölthet, többek között Word dokumentumként vagy Excel táblázatként. Az elkészült, formázott nyugtát PDF formátumban szintén le lehet tölteni.

## 1.2. Alkalmazások nyugták felosztásához

Több olyan alkalmazás is létezik, ahol a nyugtát egyenlő részekre vagy bizonyos arányban lehet elosztani egymás között. A piackutatás során viszont az volt a célom, hogy a Nyugtafelosztóhoz hasonló funkcióval rendelkező alkalmazásokat keressek, ahol résztvevők hozzáadására is van lehetőség, akik között az alapján oszlik meg a végösszeg, hogy ki melyik terméket vásárolta.

Sajnos a talált alkalmazások között a résztvevők elmentésére egyik esetében sincs lehetőség, ahogy csoportokat sem lehet létrehozni. Viszont képesek rá, hogy a résztvevőket termékekhez rendeljük, és ezt figyelembe is veszik a végösszeg számításakor.

### 1.2.1. EveryoneAllin[[14]](#EveryoneAllin)

Az EveryoneAllin[[14]](#EveryoneAllin) egyszerű felhasználói felületet nyújt, ahol egy nyugta könnyen felosztható a résztvevők között. A számolás során az első lépés a résztvevők létrehozása, majd ezután a termékek hozzáadása következik, egyesével. Itt meg kell adni a termék árát és nevét, hogy ki fizetett érte, és kinek kell fizetnie, ahol kijelöléssel több személy is kiválasztható.

A felhasználó választhat, hogy csoportonként vagy személyenként ossza fel az alkalmazás az adott termék árát. Az előbbi esetében az árat a megadott résztvevők között osztja fel, míg az utóbbi során minden résztvevő egy megadott összeget fizet.

Miután a termékek hozzáadásra kerültek, a felhasználó megtekintheti a teljes számlát, ahol egy lista résztvevőnként mutatja, hogy kinek mennyit kell fizetnie, esetleg mennyivel tartoznak neki. Az alkalmazás hátránya, hogy itt nem jelzi, ki kinek tartozik, ami problémát jelent, ha többen is fizettek.

Az EveryoneAllin[[14]](#EveryoneAllin) telefonos alkalmazással is rendelkezik, ahol a résztvevőket nem csak kézzel lehet létrehozni, hanem más alkalmazásokból is meg lehet hívni őket, mint például a WhatsApp, vagy az elmentett telefonszámok használatával.

### 1.2.2. Best Bill Splitter[[15]](#BestBillSplitter)

A Best Bill Splitter[[15]](#BestBillSplitter) több funkciót kínál a nyugták felosztására, és a korábban létrehozott adatok megtekintésére is lehetőséget nyújt. Ehhez a nyugták készítésekor nincs külön mentés gomb, minden automatikusan tárolásra kerül. Egyesével nem lehet törölni ezeket az elmentett nyugtákat, csupán az egész előzmény eltávolítására van lehetőség.

A termékeket egyesével, kézzel kell megadni. Ide lehet képet feltölteni a nyugtáról, de az csak adatként tárolódik, az alkalmazás nem ismeri fel rajta a szöveget.

Résztvevőket termékek létrehozása előtt, de akár közben is hozzáadhat a felhasználó. Ezután az alkalmazás megkérdezi, hogy ki fizetett az adott termékért, itt több résztvevő is megadható, akár különböző összeggel. Az ár elosztása során beállítható, hogy egyenlő félben, megadott százalék szerint, vagy fix értékben számítsa ki a személyenkénti összeget.

Az alkalmazás ezek után kiírja, hogy ki kinek mennyivel tartozik. Az adatok szövegként is kimásolhatók, valamint exportálhatók CSV formátumban. Az eredményt linkként is meg lehet osztani, ahol beállítható, hogy a megtekinthetők csak olvashatják vagy szerkeszthetik is a nyugtát.

### 1.2.3. Kittysplit[[16]](#Kittysplit)

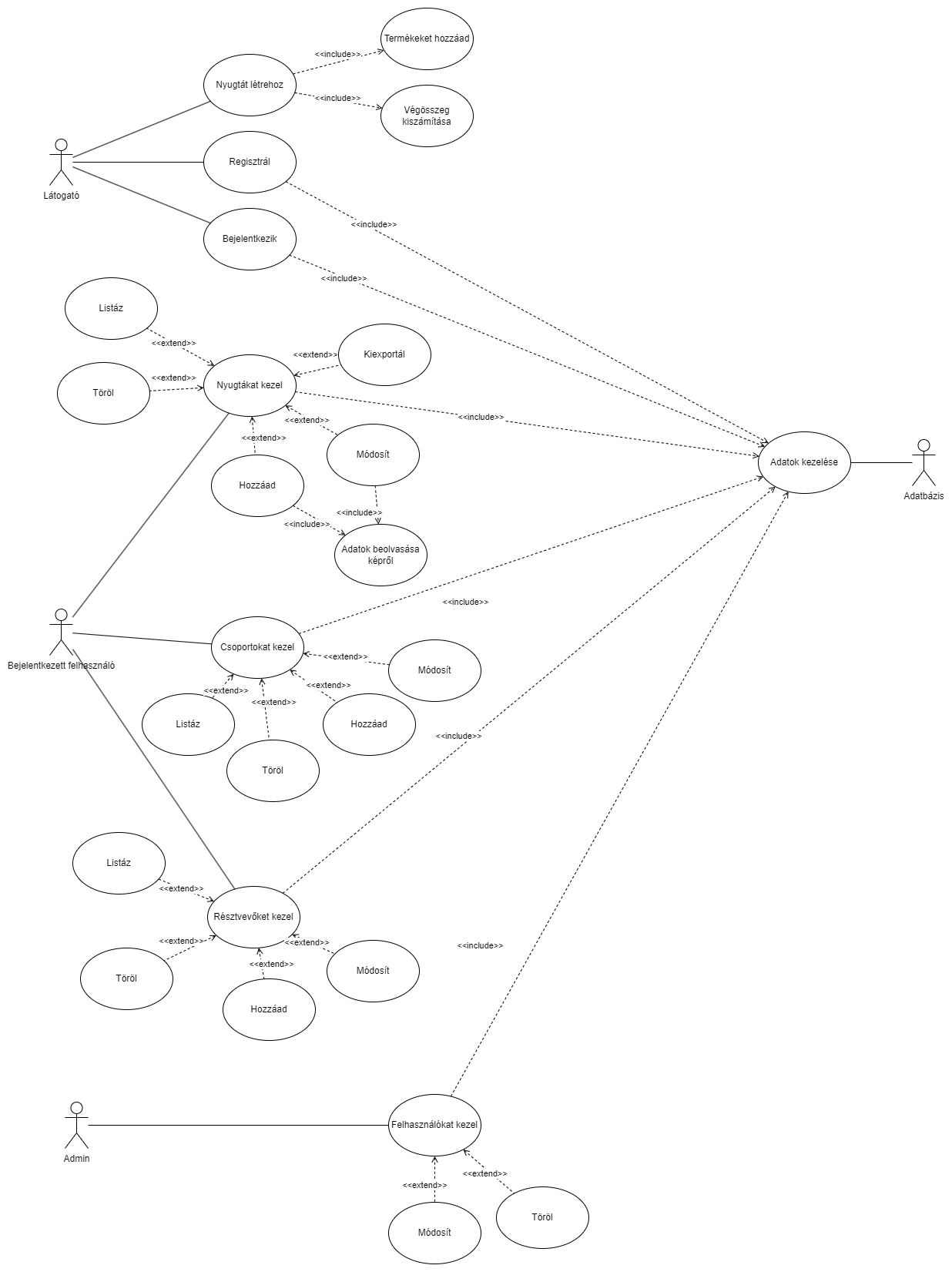
A Kittysplit[[16]](#Kittysplit) a Best Bill Splitterhez[[15]](#BestBillSplitter) hasonlóan automatikusan elmenti a nyugtákat, amiket a felhasználók létrehoztak. Ezek a főoldalon listázásra kerülnek, és egyenként törölhetőek.

Egy nyugta létrehozásához először meg kell adni a résztvevőket. Ezután a termékeket egyesével lehet hozzáadni, ahol ki kell választani a résztvevőt, aki fizetett, és beírni az árat. Alapértelmezetten az összeg egyenlően kerül elosztásra mindenki között, de van lehetőség másik módot is választani. Az utóbbi esetben az ár elosztható egyenlően a kijelölt résztvevők között is, de van lehetőség százalék vagy fix összeg szerinti elosztásra is.

Az alkalmazás mutatja, hogy ki kinek mennyivel tartozik. A nyugtát linkként meg is lehet osztani, aminek megnyitását követően ki kell választani, hogy a résztvevők közül kik vagyunk. A válaszunk alapján kijelez egy személyre szabott összegzést a nyugtáról a teljes összeggel, a mi részünkkel, és hogy ebből mennyivel tartozunk még. A link által a nyugta az ingyenes verzióban szerkeszthető is.

# 2. FUNKCIONÁLIS SPECIFIKÁCIÓ

A 2.1. ábrán látható az alkalmazás Use Case diagramja. Az alkalmazás felhasználói három csoportba tartoznak: látogató, bejelentkezett felhasználó és admin. Ezek a csoportok más-más funkciókat érnek el, különböző jogosultsági szinttel. A funkciókat ebben a fejezetben fejtem ki, valamint a 2.1. ábrán is megtekinthetők.



2.1. ábra – Use Case diagram

A Use Case diagramon a funkciók mellett a különböző jogosultsággal rendelkező felhasználók is jelen vannak.

A vendégek regisztráción és bejelentkezésen kívül hozhatnak létre nyugtákat is, de ezeket csak manuálisan, tárolt résztvevők és csoportok használata nélkül, valamint mentésre sincs lehetőségük.

A bejelentkezett felhasználók hozzáférnek a saját profiljukhoz, résztvevőikhez, csoportjaikhoz és nyugtáikhoz. Ezeket létrehozhatják, módosíthatják és törölhetik, valamint listázásra is van lehetőségük. Ők a nyugtákat képfeltöltéssel is létrehozhatják, valamint ki is exportálhatják azokat Excel táblázatba vagy JSON formátumban.

Az adminisztrátor felhasználók ezen felül képesek a többi felhasználót kilistázni és akár adminisztrátorrá tenni őket.

## 2.1. Autentikáció

Az alkalmazás használatához nem kötelező bejelentkezni, viszont a legtöbb funkció enélkül nem elérhető. Az autentikációs felületen van lehetőség regisztrációra és bejelentkezésre, amit a Google Firebase[[2]](#GoogleFirebase) Authentication szolgáltatással valósítottam meg.

### 2.1.1. Regisztráció

Fiók létrehozásához egy e-mail cím és egy jelszó megadása kötelező. A jelszónak legalább 6 karakter hosszúnak kell lennie, és egy szem ikonra kattintva megjeleníthető, ha a felhasználó leellenőrizné. E-mail cím esetében ellenőrzésre kerül a megfelelő formátum, és a regisztrálás megpróbálása után a felhasználó arról is tájékoztatást kap, ha a megadott cím már foglalt. Sikeres regisztráció esetében egy felugró ablakban jelenik meg. Ezt követően egy igazoló e-mail kerül kiküldésre a megadott fiókba.

A regisztrált felhasználók a Cloud Firestore[[2]](#GoogleFirebase) adatbázisban is tárolásra kerülnek.

### 2.1.2. Bejelentkezés

A bejelentkezéshez előzetes regisztráció és az e-mail cím visszaigazolása szükséges. Csak regisztrált, visszaigazolt e-mail címmel és a hozzá tartozó jelszóval lehet bejelentkezni, ha valamelyik helytelen, arról egy hibaüzenet jelenik meg. A jelszó itt is megtekinthető a szem ikon használatával.

Ha a felhasználó nem emlékszik a jelszavára, az *Elfelejtett jelszó?* gomb segítségével újat igényelhet. Ekkor megnyílik egy felugró ablak, ahová az e-mail cím beírása szükséges. Ezután az alkalmazás a megadott fiókba küld egy e-mailt. Az üzenetben szereplő linkre kattintva a Nyugtafelosztó bejelentkezés oldalán egy újabb felugró ablak jelenik meg, ahová az új jelszó írható.

Az oldalon egy *Emlékezz rám!* jelölőnégyzet is található, aminek kipipálásával az oldal elmenti a megadott e-mail címet a local storage-ba. A bejelentkezési felület következő meglátogatása során az így tárolt e-mail cím jelenik meg a mezőben.

Bejelentkezés után az alkalmazás a főoldalra navigál, megváltozik a fejléc, és újabb funkciók válnak elérhetővé. A bejelentkezett felhasználó ID-ja is a local storage-ban tárolódik, ahonnan az kijelentkezés esetén törlésre kerül.

## 2.2. Profil

Bejelentkezett felhasználók képesek a saját profiluk megtekintésére. Itt megjelenik az e-mail címük és a profilképük. Lehetőségük van a profilkép és jelszó módosítására, valamint a fiók törlésére is.

Profilkép módosítása során egy fájlt tudnak feltölteni. Ha a fájlformátum nem megfelelő, arról egy hibaüzenetből értesülnek. Kép feltöltése esetén egy felugró ablakban kell kiválasztaniuk a képnek egy négyzet alakú részét, ami ezután kivágásra és feltöltésre kerül, majd megjelenik a profil oldalon és a fejlécen.

Ha a jelszó módosítását választják, ahhoz egy felugró ablakban meg kell adniuk a régi, valamint az új jelszavukat. Ellenőrzésre kerül, hogy a régi jelszó megfelelő-e, valamint hogy az új hossza eléri-e a 6 karaktert.

A felhasználó fiókjának törlését egy megerősítő felugró ablak előz meg, valamint itt is szükséges megadni a felhasználó jelszavát.

## 2.3. Résztvevők

A résztvevők olyan személyeket takarnak, akik egy vásárlásban részt vettek, és az összeget közöttük szeretnénk szétosztani. Ezeket a résztvevőket egy nyugta feltöltése előtt létre kell hozni.

### 2.3.1. Résztvevők listázása

A résztvevők nevei egy listában jelennek meg, ami alap esetben ABC szerint rendezett. Ebben a listában csak az adott felhasználó által létrehozott résztvevők láthatók, más felhasználó résztvevői nem érhetőek el.

A listában keresésre is van lehetőség. Ez a résztvevők nevére szűr, és nem érzékeny a kis- vagy nagybetűkre. Ha nincs a keresésnek megfelelő elem, arról a felhasználót egy üzenet tájékoztatja.

Ha a felhasználó egy listaelemre kattint, azzal megnyílik egy résztvevő részletes oldala.

### 2.3.2. Résztvevők létrehozása

Résztvevőket a lista fölött található űrlap segítségével lehet létrehozni. A folyamat felgyorsítása érdekében csak a résztvevő neve adható meg.

Két résztvevő neve nem egyezhet. Ha mégis így történne, arról egy hibaüzenet tájékoztatja a felhasználót.

### 2.3.3. Résztvevők részletes oldala

Ez az oldal a listában a résztvevő nevére kattintva érhető el. Ha a felhasználó az URL módosításával egy olyan résztvevőt próbálna elérni, amihez nem fér hozzá vagy nem létezik, az alkalmazás visszanavigál a résztvevők listájához.

Az oldalon megtekinthető, hogy a résztvevő melyik csoportoknak a tagja, és melyik nyugtákban vett részt. Ezekre kattintva az alkalmazás átnavigál az adott csoport vagy nyugta oldalára.

A nyugták listáján belül a felhasználó azt is megtekintheti, hogy a résztvevő kinek fizet és mennyit, valamint hogy ebből mennyit fizetett már ki. Egy jelölőnégyzet segítségével megadható, hogy a résztvevő kifizette a tartozását. A még ki nem fizetett összegek piros színnel jelennek meg.

Törölni is itt lehet őket, az oldalon megjelenő gomb segítségével. Ha egy olyan résztvevő kerül törlésre, aki szerepel egy csoportban, a csoport tagjai közül is el lesz távolítva. Ha egy nyugtában is szerepel, onnan nem kerül törlésre, hanem a neve helyett a *\*Törölt résztvevő\** szöveg jelenik meg.

## 2.4. Csoportok

A csoportok több résztvevőt tartalmaznak, és azt a célt szolgálják, hogy felgyorsítsák a nyugta felosztását. Erre leginkább akkor lehet szükség, ha több termékért többször ugyanazok a személyek együtt fizetnek. Csoport létrehozása után ezeket a személyeket nem kell mindegyik terméknél felsorolni, elég csak a csoportot megadni.

Csoportok lehetnek például speciális étrendű vagy valamilyen érzékenységgel rendelkező emberek, mint például a vegetáriánusok vagy a gluténérzékenyek.

### 2.4.1. Csoportok listázása

Csoportok a résztvevőkhöz hasonlóan, ABC sorrendben kerülnek listázásra. Csak az adott felhasználó által létrehozott csoportok láthatóak. Itt is van lehetőség keresésre, a megadott nevű csoportok alapján. Egy listaelemre kattintva nyitható meg az adott csoport részletes oldala.

### 2.4.2. Csoportok létrehozása

A lista felett látható *Új* gomb megnyomásával hozható létre új csoport. Ekkor egy felugró ablakban meg kell adni a csoport nevét, ami akár tagok nélkül is elmenthető.

A résztvevőket egy lenyíló listából lehet kiválasztani, ahol az összes létrehozott résztvevő látható. Egy résztvevő a mellette található x gombbal távolítható el a csoportból. A létrehozás során minden résztvevő csak egyszer kerül a tagok közé, még abban az esetben is, ha a felhasználó többször adta meg.

Ha a létrehozni kívánt csoport neve már foglalt, egy hibaüzenet jelenik meg.

### 2.4.3. Csoportok részletes oldala

Egy csoportot törölni vagy módosítani annak részletes oldalán van lehetőség. A felületen a csoport neve és egy listában annak tagjai láthatók, a listaelemek a résztvevők részletes oldalára vezetnek.

Szerkesztés esetén a létrehozás során már megismert felugró ablak jelenik meg, ahol módosíthatók a csoport adatai.

A törlést egy megerősítő felugró ablak előzi meg.

## 2.5. Nyugták

A nyugták termékek listájából állnak, amik névvel, darabszámmal és árral rendelkeznek. Ezekhez a termékekhez felhasználók és csoportok rendelhetők. A Nyugtafelosztó ezek alapján kiszámolja, hogy a résztvevőknek egyenként mennyit kell fizetniük.

### 2.5.1. Nyugták listázása

A felhasználó által létrehozott nyugták dátum szerinti csökkenő sorrendben jelennek meg. A rendezés a fejlécre kattintva módosítható a bolt nevére vagy a végösszegre. Lehetőség van keresésre a nyugta boltja alapján. Egy-egy listaelem az adott nyugta részletes oldalára vezet.

## 2.6. Nyugták létrehozása

A lista felett egy *Új* gomb található, aminek megnyomása egy új oldalra vezet. Ha a felhasználó a *Képfeltöltés* gombot választja, akkor a létrehozás oldalát egy képfeltöltési folyamat előzi meg.

### 2.6.1. Képfeltöltés

A felhasználót a kép feltöltése előtt egy dialógus figyelmezteti a szövegfelismerés hátrányairól. Az itt használt Tesseract.js[[7]](#TesseractJs) képes felismerni magyar szöveget, de sokszor hibásan. Emellett azt tapasztaltam, hogy a kissé gyűrött, saját nyugtákról készített képek esetében sem teljesít jól. Így szükséges, hogy a feltöltött képen a szöveg olvasható legyen, és a kép ne legyen elforgatva.

További hátrány, hogy rengeteg formájú, elrendezésű nyugta létezik, így az alkalmazás a megszokott nyugtáktól eltérőket nem tudja megfelelően felismerni. Ha a képen a nyugta mellett más szöveg is szerepel a háttérben, a Tesseract.js[[7]](#TesseractJs) azt is beolvashatja, ezután a Nyugtafelosztó ezeket a hibás adatokat megpróbálja majd az űrlapba illeszteni.

Ha a felhasználó nem képet próbál feltölteni, arról egy hibaüzenet tájékoztatja. Megfelelő formátumú fájl feltöltése után a nyugta adatai egy űrlapba kerülnek: a bolt neve, a vásárlás dátuma, végösszege és pénzneme, valamint egyenként a termékek neve, darabszáma és ára.

A felhasználónak a további adatokat kézzel kell megadnia.

Az említett hibák miatt fontos, hogy a felhasználó átnézze az így elkészült nyugtát, mivel az adatok pontatlanok lehetnek. Ennek ellenére a képfeltöltés így is felgyorsítja egy nyugta létrehozásának folyamatát, hiszen egyszerűbb elírásokat kijavítani, mint az összes adatot kézzel begépelni.

### 2.6.2. Új nyugta

Képfeltöltés nélkül egy üres űrlap fogadja a felhasználót. Itt szükséges megadni a nyugta pénznemét, ami egy lenyíló listából választható ki; a résztvevőket, és külön azt a résztvevőt, aki a teljes nyugtát kifizette. A vásárlás dátuma és a bolt neve is rögzíthető.

### 2.6.3. Csoportok, résztvevők megadása

Egy nyugtához tetszőleges mennyiségű résztvevő hozzáadható, mindegyik csak egyszer. A résztvevők egy lenyíló listából választhatók ki, és a mellettük szereplő x-el törölhetők, ha véletlenül lettek hozzáadva.

A nyugtáért fizető résztvevőt csak az után lehet megadni, miután a résztvevők mező ki lett töltve, mivel a fizetőt közülük kell kiválasztani.

Azok a csoportok automatikusan hozzáadásra kerülnek, amikből legalább egy tag szerepel már a nyugtán. Ezek a csoportok nem láthatóak a felhasználó számára a résztvevők mezőben, hanem a termékekért fizető résztvevők kiválasztásakor játszanak szerepet.

### 2.6.4. Termékek hozzáadása

Egy terméket az *Új termék* gombbal adhat hozzá a felhasználó. Ekkor új mezők jelennek meg az űrlapban, ahol a termék neve, darabszáma, ára és az érte fizető résztvevők adhatóak meg.

A darabszám itt csak az adatok rögzítése céljából szerepel, az árnál a már ezzel felszorzott érték megadása szükséges. Ennek oka az, hogy a legtöbb nyugtán a termékek mellett is a felszorzott összeg szerepel, és a darabszám külön szerepel – ezért ha a felhasználó a képfeltöltés mellett döntött, a termékek mellől azok teljes összege lesz kiolvasva, ami a darabszámmal már fel van szorozva.

A *Fizet* mezőben több résztvevő vagy csoport megadható, vegyesen is, amiket egy lenyíló listából lehet kiválasztani, ahol csak a nyugtához hozzáadott csoportok és résztvevők láthatók. Ha valamelyik véletlenül lett hozzáadva, a mellette látható x gomb segítségével törölhető. A listából a *Mindenki* elem is kiválasztható, amivel az összes résztvevő automatikusan hozzáadódik az adott termékhez.

Ha egy résztvevő vagy csoport többször kerül megadásra ugyanahhoz a termékhez, az alkalmazás csak egyszer fogja figyelembe venni. Ugyanez lép fel akkor, ha a felhasználó megad egy résztvevőt egy olyan csoport mellé, aminek tagja.

A nyugta elmentésekor a csoportok felbontásra kerülnek, és csak annak tagjai kerülnek tárolásra.

## 2.7. Nyugták részletes oldala

A nyugták részletes oldalán van lehetőség a nyugta kiexportálására, szerkesztésére és törlésére. Emellett a résztvevőkre felbontott végösszeg is itt tekinthető meg egy táblázatban. A táblázat egy-egy elemére kattintva megtekinthető az adott résztvevő részletes oldala.

Megjelenik a nyugta boltja, dátuma, és a teljes nyugtáért fizető neve is.

### 2.7.1. Fejenkénti végösszeg

Egy termék ára egyenlően kerül elosztásra a megadott résztvevők és csoportok tagjai között, majd az egyes termékek felosztása után az alkalmazás az így kapott értékeket összegzi. A táblázatban az így kiszámított összeg jelenik meg a résztvevők neve mellett.

A *Fizetett* oszlopban a felhasználó azt is megtekintheti, hogy az adott összegből mennyit fizetett a résztvevő. Ez a nyugta elmentése után minden résztvevő esetében 0, kivéve azé, aki a teljes nyugtáért fizetett – az ő része már ki van fizetve. Az emellett lévő jelölőnégyzettel beállítható, ha valaki kifizette az összeget. Ekkor az oszlop értéke a résztvevő végösszegére módosul, és a jelölőnégyzet deaktiválttá válik. Azon résztvevők sora, akik még nem fizették ki a teljes összeget, pirossal ki van emelve.

A táblázat fejlécében található szerkesztés gombbal is módosítható az egy-egy résztvevő által kifizetett összeg. Ekkor egy felugró ablakban megjelenik az összes résztvevő az eddig fizetett összeggel. A kifizetett mennyiség itt bármennyire módosítható, még a már kifizetett összegek is megváltoztathatók.

### 2.7.2. Kiexportálás

A nyugta kiexportálható Excel táblázatba CSV formátumban. A fájlban két táblázat szerepel. Az egyik a nyugta alapadatait tartalmazza: a bolt nevét, a dátumot, a pénznemet, a végösszeget, és a fizető személyt. Emellett egy külön táblázatban mutatja a résztvevők nevét, hogy mennyit kell fizetniük, és hogy ebből mennyit fizettek már ki.

Exportálás esetén JSON formátum is választható. Ez ugyanezeket az adatokat tartalmazza, csak táblázat helyett kulcs-pár értékekként.

### 2.7.3. Nyugták szerkesztése, törlése

A szerkesztés gombra kattintva a létrehozásról ismert űrlap jelenik meg, ahol a korábban leírtakkal megegyező módon lehet módosítani, majd elmenteni az adatokat.

Törlésre is itt van lehetőség a megfelelő ikonra kattintva, amit egy megerősítő dialógus előz meg.

## 2.8. Nyugták vendég felhasználóknak

Nyugtákat vendég felhasználók is képesek létrehozni. Bejelentkezés nélkül a főoldalon csak az *Új nyugta* gomb jelenik meg, ami a nyugták létrehozási űrlapját nyitja meg. Ők csak manuálisan hozhatnak létre nyugtákat, képfeltöltésre nincs lehetőségük.

A pénznemek listájához a vendégek hozzáférhetnek, de résztvevőket és csoportokat nem hozhatnak létre. Ezért a *Résztvevők* mezőben nincs lenyíló lista, amiből választhatnak, ehelyett a résztvevők nevét tudják beírni, amit vesszővel vagy az enter billentyű lenyomásával választhatnak el.

Az általuk létrehozott lista nem tárolódik az adatbázisban, csak a local storage-ba lesz elmentve. Külön résztvevők sem jönnek létre, hanem a local storage-ban található nyugtában tárolódnak ideiglenesen.

A létrehozott nyugta részletes oldalát meg tudják tekinteni, ahol a nyugta boltja, dátuma és fizetője látszik. A fizetők táblázatában csak a résztvevők nevét és a fizetendő összeget látják, a többi funkció előlük rejtve marad. A nyugtát kiexportálni nem képesek. A felhasználói felületen módosításra, törlésre sincs lehetőségük.

Ha új nyugtát hoznak létre, a korábbi felülíródik a local storage-ban, bejelentkezéssel pedig törlődik.

# 3. FELHASZNÁLT TECHNOLÓGIÁK

## 3.1. Angular[[1]](#Angular)

Az Angular[[1]](file:///C:\Users\hanna\OneDrive\Dokumentumok\Szakdoga\nyugtafeloszto\szakdolgozat\SomogyváriHanna_GLXGEU_szakdolgozat.docx#Angular) egy webfejlesztési, nyílt forráskódú keretrendszer, ami a TypeScript nyelven alapul. A fő célja single-page (egy oldalas) alkalmazások fejlesztése, ahol új weblapok betöltése helyett egyetlen oldal megjelenő tartalmát frissíti dinamikusan a web szerveren tárolt adatokkal.

A keretrendszer többek között egy struktúrát is biztosít az alkalmazások elkészítéséhez. Ennek alapjai a komponensek, amik a rendszer kisebb elemei, mint egy lista vagy egy bejelentkezési űrlap. Egy komponens egy TypeScript osztályból és egy HTML fájlból áll, előbbi a működést, utóbbi pedig a megjelenést definiálja. Szolgáltatások létrehozására is van lehetőség, amik megjelenítéshez közvetlenül nem kapcsolódó funkciókat valósítanak meg, például adatbáziselemekkel való interakciókat. Egy szolgáltatást több komponens is használhat, amit előtte injektálni kell. A szorosan összekapcsolódó komponensek, szolgáltatások és egyéb kódfájlok modulokba szervezhetők.

Fejlesztés során az Angular[[1]](file:///C:\Users\hanna\OneDrive\Dokumentumok\Szakdoga\nyugtafeloszto\szakdolgozat\SomogyváriHanna_GLXGEU_szakdolgozat.docx#Angular) 16-os verziójával dolgoztam. Emellett az Angular[[1]](file:///C:\Users\hanna\OneDrive\Dokumentumok\Szakdoga\nyugtafeloszto\szakdolgozat\SomogyváriHanna_GLXGEU_szakdolgozat.docx#Angular) lentebb részletezett segédkönyvtárait is használtam.

### 3.1.1. Angular Material[[19]](#AngularMaterial)

Az Angular Material[[19]](#AngularMaterial) egy komponenseket tartalmazó könyvtár. Weboldalak egyszerűbb, gyakran használt elemeiből áll, mint gombok, menük, űrlapmezők. Ezek az alap stílusukkal is alkalmazhatók, de a személyre szabásra is van lehetőség.

A használatuk egyszerű, ami megkönnyíti a fejlesztési folyamatot, és egységes kinézetet biztosít az alkalmazásnak.

### 3.1.2. AngularFire[[20]](#AngularFire)

Az AngularFire[[20]](#AngularFire) könyvtár a Firebase[[2]](#GoogleFirebase) adatbázissal való kapcsolatot valósítja meg. Az itt definiált metódusok használhatók többek között autentikációra, valamint adatbáziselemek lekérésére és manipulációjára.

### 3.1.3. Ngx Image Cropper

A segédkönyvtárból származó *ImageCropperModule* használatával valósítottam meg a feltöltött profilkép kivágását. A modul tartalmazza a HTML komponenst, ahol a kép kivágása megtörténik, emellett komponens és CSS tulajdonságokat, valamint metódusokat is. Ezáltal a képfeltöltés személyre szabható, például megadható a kivágás mérete, vagy hogy mi történjen az esemény után.

## 3.2. Google Firebase[[2]](#GoogleFirebase)

A Firebase[[2]](#GoogleFirebase) egy Google által fejlesztett backend platform alkalmazások fejlesztéséhez. Széleskörű szolgáltatásokat nyújt a felhasználók számára, és különféle platformokon való alkalmazások fejlesztését támogatja.

### 3.2.1. Authentication

A Firebase Authentication szolgáltatás a felhasználók autentikációját könnyíti meg. Többféle bejelentkezési módot is lehetővé tesz, az e-mail címen kívül használható például telefonszám, Google és Facebook fiók is. Nemcsak felgyorsítja az alkalmazáson belüli autentikáció fejlesztését, de a felhasználói adatok biztonságos tárolásáról is gondoskodik.

### 3.2.2. Cloud Firestore

A Cloud Firestore adatbázis NoSQL dokumentumokként tárolja az adatokat. A lekérdezésekhez beépített metódusok használhatók, amik egyszerűbbé teszik az adatbázissal való kommunikációt.

### 3.2.3. Cloud Storage

A képek tárolásához a Cloud Storage szolgáltatást használtam, ami egy adatbázis fájloknak.

### 3.2.4. Hosting

Az alkalmazást a Hosting szolgáltatás segítségével hosztoltam. Ehhez a Firebase[[2]](#GoogleFirebase) által biztosított subdomaint is igénybe vettem.

## 3.3. Tesseract.js[[7]](#TesseractJs)

A Tesseract.js[[7]](#TesseractJs) egy JavaScript nyelvre épülő OCR motor. Több, mint 100, köztük magyar nyelven is képes szöveget felismerni, emellett az automatikus szövegelrendezést és a szkriptek detektálását is támogatja.

A kódba szkript tagként beimportálható, és ezek után egy *Workerként* vagy egy időzített feladat elvégzésére is használható.

## 3.4. Sass[[5]](#SCSS)

A Sass[[5]](#SCSS) egy CSS bővítmény és előfeldolgozó, ami a CSS-hez hasonló, de hatékonyabban használható szintaktikával rendelkezik. A stíluslapokba új funkciókat vezet be, mint a változók, az egymásba ágyazott szabályok és az importálások.

Két szintaktikával rendelkezik, amik közül és az SCSS[[5]](#SCSS)-t használtam. A kettő közül az SCSS[[5]](#SCSS) jobban hasonlít a CSS-hez, mivel itt is ugyanúgy szükséges a kapcsos zárójelek és a pontosvesszők használata, míg a SASS[[5]](#SCSS) szintaxis esetében ezek szóközökkel és vesszővel helyettesítettek.

## 3.5. Egyebek

A diagramokat a draw.io[[3]](#drawio) oldal segítségével készítettem. Itt több szoftverfejlesztéshez kapcsolódó diagram sablonja elérhető, valamint előre elkészített elemeket is biztosít.

A fejlesztést a Visual Studio Code[[4]](#VisualStudioCode) szerkesztőben végeztem. Az alkalmazás átláthatóvá teszi a kódot, rendelkezik beépített terminállal, lehetőséget ad bővítmények hozzáadására, és rengeteg olyan funkcióval rendelkezik, ami felgyorsítja a fejlesztés folyamatát.

A GitLab[[6]](#GitLab) platformját egyaránt használtam verziókövetésre és a feladatok kezelésére is. Itt követtem nyomon a fejlesztés haladását, valamint a feladatok használatával a következő lépéseimet is előre el tudtam tervezni.

A GitLab Issue board funkcióját is használtam, ahol a feladatokat táblába rendezett formában lehetett követni. Itt létrehoztam a fejlesztésnek megfelelő oszlopokat (hátralévő, folyamatban, tesztelés alatt és kész), így a feladataim mozgatásával mindig pontosan láttam, hogy éppen hol tartok a fejlesztés alatt.

# 4. Architektúra

A Nyugtafelosztó architektúrája két részből, a frontendből és a backendből áll. Az előbbit az Angular[[1]](#Angular) keretrendszer segítségével valósítottam meg, míg az utóbbihoz a Google Firebase[[2]](#GoogleFirebase) szolgáltatást használtam.

A két oldal HTTP kérések küldésével és fogadásával kommunikál egymással. A frontend oldal küld egy kérést a backendnek, ha adatokra van szüksége vagy adatok módosultak. A backend erre küld egy választ, aminek eredménye a frontend oldalon jelenik meg. Ilyen például egy nyugta adatainak lekérése vagy a felhasználó által módosított nyugta adatainak elküldése.

## 4.1. Frontend

A frontend az a része az alkalmazásnak, amivel a felhasználó interakcióba lép. Ide tartoznak a weboldalon megjelenő elemek, mint a fejléc, az autentikációs oldalak vagy például a nyugták listája.

A HTML kód határozza meg a struktúrát és az oldal elemeit, ezek stílusát pedig az SCSS lapok írják le. Az oldal működéséért felelő logika a TypeScript kódban található, amik például akkor futnak le, ha egy komponens betöltődik vagy ha a felhasználó egy elemre kattint.

## 4.2. Backend

A backenden tárolódnak az alkalmazás adatai. Ez a rész felelős a felhasználó autentikációért, az adatok kezeléséért és a hosztolásért.

A frontendről érkező kérésekre a megfelelő adatokat visszaküldi vagy azokat módosítja, létrehozza, törli. A felhasználó ezt az oldalát nem látja az alkalmazásnak, és nem is tud közvetlenül interakcióba lépni vele.

A Google Firebase[[2]](#GoogleFirebase) nem tekinthető hagyományos backendnek, hanem a Backend-as-a-Service (BaaS) kategóriába tartozik. Ez azt jelenti, hogy a hagyományos megoldással ellentétben nem szükséges külön backendet implementálni, helyette a Firebase[[2]](#GoogleFirebase) által biztosított eszközök, mint a szolgáltatások és metódusok használhatók.

# 5. Belső felépítés

## 5.1. Modulok

Egy Angular[[1]](#Angular) alkalmazás több modulból épül fel, amik az alkalmazás szorosan összetartozó részeit foglalják magukba. Tartalmazhatnak másik modulokat, komponenseket, szolgáltatásokat és más egyéb kódot. Más modulok funkcióit is képesek használni, ha azok exportálva vannak.

A Nyugtafelosztóban az alábbi modulokat valósítottam meg:

– *AppModule*:

Ez a gyökér modul az alkalmazásban, ami az összes többi modult tartalmazza. Egyetlen komponensében, az *AppComponentben* megjeleníti az alkalmazás fejlécét és *RouterOutlet* segítségével dinamikusan tölti be a router állapota alapján szükséges komponenseket.

– *AuthModule*:

A felhasználó autentikációért felelős modul. Tartalmazza a bejelentkezési és regisztrációs komponenseket, valamint az elfelejtett jelszó esetén megjelenő jelszó visszaállító komponenst, ami egy dialógusként jelenik meg a bejelentkezési felületen.

– *HomeModule*:

Itt található a *HomeComponent*, ami az alkalmazás főoldala. Ez jelenik meg az alkalmazás megnyitása, valamint bejelentkezés után.

– *MemberModule*:

A résztvevőkhöz kapcsolódó komponenseket egybefogó modul. Komponensei a résztvevők listája és azok részletes oldala.

– *GroupModule*:

Ez a modul tartalmazza a csoportokhoz tartozó kódrészeket, mint azok listája, részletes oldala és a létrehozáskor valamint módosításkor megjelenő szerkesztő dialógus.

– *ReceiptModule*:

Itt találhatók a nyugtákhoz kapcsolódó komponensek. A nyugták listája, részletes és szerkesztő oldala mellett a résztvevők kifizetett összegeit szerkesztő dialógus komponense is ide tartozik.

– *SharedModule*:

Ide tartozik az összes olyan kódrészlet, amit a többi modul megosztva használ, amik több helyen is importálva vannak. A lentebb részletezett szolgáltatások és interfészek itt vannak megvalósítva. Emellett ide tartozik az oldal fejlécében mindig megjelenő *MenuComponent*, valamint a *DialogComponent*, ami a neki átadott adatoknak megfelelő tartalmat jeleníti meg dialógusként. Az *authGuard* is itt van definiálva, ami biztosítja, hogy csak a bejelentkezett felhasználók férhessenek hozzá bizonyos oldalakhoz.

## 5.2. Interfészek

TypeScriptben interfészek segítségével definiálhatjuk, hogy egy objektumnak milyen tulajdonságokkal kell rendelkeznie, és ezek milyen adattípusba tartozzanak. Ezzel megkönnyítik az objektumok típusának ellenőrzését, hiszen egy helytelen típusba tartozó érték megadása egyből hibát jelez. Emellett egységesítik is a kódot, hiszen az azonos interfészek azonos struktúrával fognak rendelkezni.

Az általam definiált interfészek a Google Firebase[[2]](#GoogleFirebase)-ben tárolt kollekciókkal párhuzamba vonhatók, hiszen az adatbázisból lekért adatokat a megfelelő típusú interfészekben tárolom. Bizonyos eltérések vannak az interfészek és a Firebase[[2]](#GoogleFirebase) kollekciók között, amiket lentebb részletezek.

– *User*:

A felhasználókat meghatározó interfész. Az adatbázisban tárolt mezőkön kívül rendelkezik egy formázott utolsó bejelentkezés mezővel, ami csak a frontenden van használva, a backendre nincs feltöltve.

– *Currency*:

Az alkalmazásban használható pénznemek tulajdonságait definiáló interfész.

– *Member*:

Ez határozza meg egy résztvevő tulajdonságait. Az adatbázis mezőin felül itt definiálva van egy fizet és fizetett tulajdonság is, amik egy adott nyugtában tárolt adatok alapján vannak kiszámolva.

– *Group*:

A csoportok tulajdonságait leíró interfész. A tagok tulajdonsága egy listát definiál, aminek elemei a *Member* interfészbe tartoznak.

– *Product*:

Ennek az interfésznek nincs párja az adatbázisban, helyette a termékek a nyugtákon belül egy listában tárolódnak. A könnyebb kezelés érdekében frontenden külön interfészt hoztam létre a termékek egységesítése érdekében, ami egy névből, árból, darabszámból és a fizető résztvevők ID-ját tartalmazó listából áll.

– *Receipt*:

A nyugtákat ez az interfész definiálja. A *User* interfészhez hasonlóan itt is létezik egy formázott dátum tulajdonság, ami az adatbázisban nincs definiálva.

## 5.3. Szolgáltatások

A szervizek önálló osztályok, amik egy-egy specifikus célt valósítanak meg. Jelen esetben a szervizek a backenddel való kommunikációt biztosítják.

A használatukhoz először injektálni kell őket a megfelelő komponensekbe, és csak egyszer lesznek példányosítva.

Az általam megvalósított szervizek nagyrésze a Cloud Firestore[[2]](#GoogleFirebase) adatbázissal lép kapcsolatba.

– *AuthService*:

Az autentikációhoz kapcsolódó funkciókat megvalósító szolgáltatás, a Firebase Authentication[[2]](#GoogleFirebase) szolgáltatásainak használatával. Itt található például a be- és kijelentkezésért, az e-mail küldésért és a fiók törléséért felelős logika.

– *UserService*:

Ez a szerviz kezeli a felhasználókat érintő adatok lekérését vagy módosítását. Definiálja többek között a felhasználókat lekérő funkciókat, a létrehozást, a módosítást és a törlést.

– *CurrencyService*:

Mivel a pénznemek csak olvashatóak a felhasználók számára, ezért ez a szerviz csak az összes vagy adott ID-val rendelkező pénznem lekérését biztosítja.

– *MemberService*:

A résztvevők lekérését, létrehozását, módosítását és törlését megvalósító funkciókat definiálja.

– *GroupService*:

Ez a szerviz felelős a csoportok lekéréséért, létrehozásáért, módosításáért és törléséért.

– *ReceiptService*:

A *MemberService* és *GroupService* funkcióihoz hasonlókat tartalmaz, nyugták kezelésére.

– *ImageService*:

A Firebase Storage[[2]](#GoogleFirebase)-ban tárolt képek lekérését, feltöltését és törlését biztosítja.

## 5.4. Routing

A képen szöveg, diagram, Tervrajz, Párhuzamos látható

Automatikusan generált leírás

5.4.1. ábra – Routing fa

A Nyugtafelosztóban az Angular[[1]](#Angular) Router kezeli a navigációt. Mivel ez egy single-page (egyoldalas) alkalmazás, az új oldalak nem külön-külön töltődnek be, hanem egy oldalon megjelenő komponensek dinamikusan frissülnek. A Router az URL tartalma alapján változtatja meg a látható elemeket.

Az alkalmazásra navigálva a *HomeModule* töltődik be. Az 5.4.1. ábra mutatja, hogy az URL megváltoztatása hatására melyik komponensek töltődnek be.

Bizonyos oldalak *Auth Guard* segítségével vannak megvédve az illetéktelen hozzáférés ellen. Az *Auth* *Guard* navigáció előtt ellenőrzi, hogy a felhasználónak van-e joga az adott oldalhoz hozzáférni, és átirányítja egy másik oldalra, ha a jogok nem megfelelők.

A Router osztály *CanActivateFn* funkcióját implementáltam, ami ellenőrzi, hogy az adott felhasználó be van-e jelentkezve, és ha nem, akkor a főoldalra (*HomeComponent*) irányítja át. Az alkalmazásban a *MemberModule*, *GroupModule* és *ProfileModule* modulokba tartozó oldalak védettek Auth Guard használatával.

# 6. Biztonság

A Cloud Firestore[[2]](#GoogleFirebase)lehetőséget biztosít biztonsági szabályok írására, ami korlátozhatja az adatbázisban tárolt adatokhoz való hozzáférést külön olvasás, létrehozás, módosítás és törlés szinten. A Security Rules szolgáltatás használatával megadtam, hogy egyes kollekciókban milyen szabályok legyenek érvényesek az adatok kezelése során.

A képen szöveg, képernyőkép, Betűtípus, szám látható

Automatikusan generált leírás

6.1. ábra – Objektumokhoz és erőforrásokhoz való hozzáférés

A 6.1. ábrán láthatóak a vendég, a bejelentkezett felhasználó és az adminisztrátor jogai az egyes objektumokhoz vagy erőforrásokhoz való hozzáférés során.

Ez alapján a pénznemekhez mindenkinek adtam olvasási jogot. Bár a vendég felhasználó képes új nyugtákat készíteni, de azok csak a local storage-ban tárolódnak, így az adatbázisban csak a már regisztrált felhasználók hozhatják létre őket. Felhasználót csak a vendégek képesek létrehozni, a bejelentkezett felhasználók pedig képesek olvasni, módosítani vagy törölni a saját profiljukat. Résztvevők, csoportok, nyugták és profilképek csak bejelentkezés után olvashatók, hozhatók létre, törölhetők és módosíthatók, és minden felhasználó csak a saját maga által létrehozottakhoz fér hozzá.

Az adminisztrátorok képesek mindenre, amire egy bejelentkezett felhasználó. Ezen felül ők képesek másik felhasználók adatait olvasni, emellett módosíthatják azok adminisztrátor státuszát.

A képen szöveg, képernyőkép, Betűtípus látható

Automatikusan generált leírás

6.2. ábra – Nyugta létrehozásának szabályai

A 6.2. ábra mutat egy szabályt, ami a nyugták létrehozására alkalmaz megkötéseket. A *request.auth != null* biztosítja, hogy a kérést bejelentkezett felhasználó küldte. A nyugta *user* mezőjének, valamint a kérést küldő felhasználó UID-jának összehasonlítása azt határozza meg, hogy egy felhasználó csak saját maga számára képes nyugtákat létrehozni. Adatok módosítása során ezeken felül az is ellenőrzésre kerül, hogy az objektum *user* értéke nem változott-e meg, így a felhasználók nem képesek mások objektumait a sajátjukká tenni.

A szabályok definiálásával nem csak a különféle jogosultságokat állítottam be, hanem az adatok típusára, értékére is kikötéseket tettem. A 6.2. ábrán látható, hogy a nyugta ID-jának szövegnek kell lennie, és a *request.resource.data.id != null* sor miatt ennek a mezőnek kötelezően értéket kell adni. Emellett például olyan szabályokat is definiáltam, mint hogy az adat ID-ja nem változtatható meg, és a nyugta boltjának hossza nem lehet több 100 karakternél.

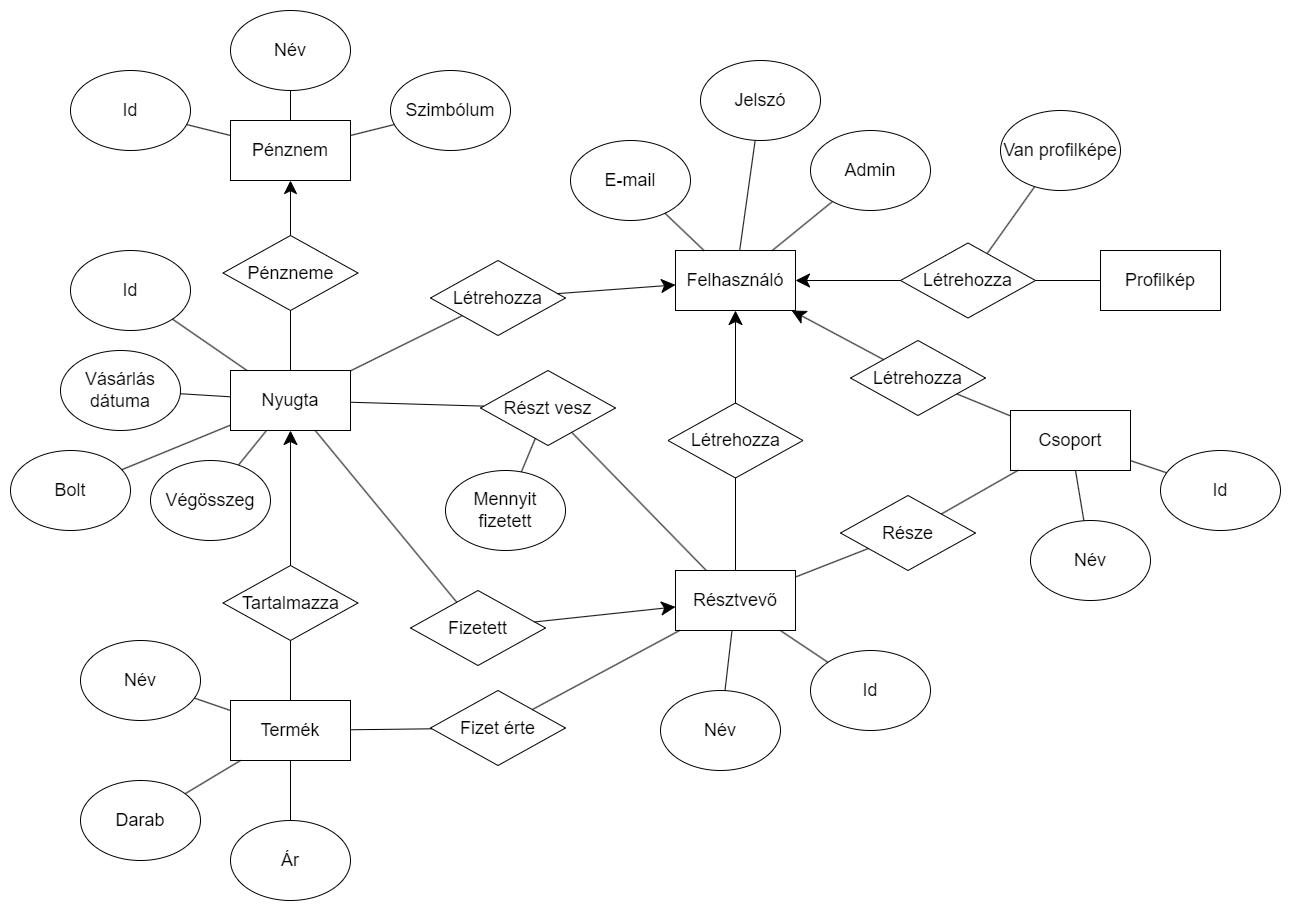
A Cloud Firestore[[2]](#GoogleFirebase)-hoz hasonlóan a fájlok tárolásáért felelős Cloud Storage[[2]](#GoogleFirebase)-ban is megadhatók szabályok. Itt a főoldalon megjelenő képekhez bárki hozzáférhet, de a profilképeket csak annak tulajdonosa érheti el.

A biztonságról a frontend oldalon is gondoskodtam, ahol a backendnek megfelelő szabályok érvényesülnek. Helytelen adatokról, mint a túl hosszú szöveg megadásáról a felhasználó hibaüzenetben értesül. A szolgáltatásokon belül adatok lekérése során szűrés történik az adott felhasználóra, így azok mindenképpen csak a saját objektumaikat és erőforrásaikat képesek megkapni. Az egyes oldalak hozzáférésének korlátozásáról a fentebb részletezett Auth Guard felel.

# 7. Adatmodell

A Google Firebase[[2]](#GoogleFirebase) Cloud Firestore szolgáltatását vettem igénybe az adatok tárolásához, a fájlok pedig a Firebase Storage-ba kerülnek elmentésre. A Cloud Firestore egy NoSQL adatbázis, ahol az objektumokat dokumentumok képviselik, amik kollekciókban tárolódnak.

A hagyományos adatbázisokkal ellentétben egy dokumentumon belül lehetnek beágyazott objektumok is. Ezért a termékek számára nem hoztam létre külön kollekciót, hanem azok a megfelelő nyugtán belül tárolódnak, amit a köztük lévő 1:n kapcsolat is megenged.



7.1. ábra – Egyed-kapcsolat diagram

A 7.1. ábrán látható a rendszer egyed-kapcsolat diagramja. Az itt szereplő egyedek a Cloud Firestore adatbázisban tárolódnak, a profilképet kivéve, ami egy képfájl a Firebase Storage-ban.

## 7.1. Pénznem

A pénznemek konstans adatok, amiket ID segítségével lehet beazonosítani. Nem bejelentkezett felhasználók is elérik őket.

Rendelkeznek egy névvel (pl.: HUF), ami a keresésüket könnyíti meg, és egy szimbólummal (p.: Ft), ami összegek mellett jelenik meg.

Minden nyugtának egy pénzneme lehet. A két objektumot a nyugtákban található pénznem mező köti össze, ami a pénznem azonosítóját tartalmazza.

Az adatbázisban elmentett pénznemek megfelelő pénzjeléhez (szimbólum) és ISO kódjához (név) a Wikipédia forgalomban lévő pénznemek listáját[[23]](#WikipediaCurrencies) használtam.

## 7.2. Felhasználó

A felhasználók a Cloud Firestore mellett a Google Firebase[[2]](#GoogleFirebase) Authentication szolgáltatásának segítségével is tárolásra kerülnek. Itt az autentikációhoz kapcsolatos adatok találhatóak, mint az ID, az e-mail cím és hogy az megerősített-e, valamint az utolsó bejelentkezés dátuma. A jelszavak biztonságos kezeléséről is ez a szolgáltatás gondoskodik. Új tulajdonságok ehhez az objektumhoz nem adhatók hozzá, ezért hoztam létre a felhasználó kollekciót a Cloud Firestore-on belül.

Az adatbázisban is mentésre kerül a felhasználó ID-ja és e-mail címe, valamint egy-egy logikai érték jelképezi, hogy a felhasználó admin-e és van-e profilképe.

Lehetőség lenne rá, hogy a profilkép URL hivatkozását a Firebase Authentication felhasználójában tároljam, de a fájl elérési útvonalának egyértelműsége miatt nem ezt a módszert alkalmaztam.

## 7.3. Profilkép

A profilképek képfájlok, amik a Firebase Storage-ban a *profile/* elérési útvonalon belül tárolódnak. A fájlok neve megegyezik a feltöltő felhasználó ID-jával, így azok mindig egyértelműen beazonosíthatók. Új fájl feltöltése során a korábbi kép törlésre kerül, így egy felhasználóhoz mindig csak egy profilkép tartozik. Emellett bármilyen formátumú kép feltölthető, amit az alkalmazás az adatbázisba kerülés előtt PDF formátumra konvertál.

Ha egy felhasználó nem rendelkezik profilképpel, a *default.png* nevű kép kerül használatra, amit minden bejelentkezett felhasználó képes olvasni, de törölni vagy módosítani nem lehet.

## 7.4. Résztvevő

A résztvevők rendelkeznek egy ID-val, ami az egyedi azonosítójuk, és egy névvel. Az őket létrehozó felhasználó azonosítója a *user* nevű mezőben tárolódik, ami biztosítja, hogy az objektumhoz csak a tulajdonosuk férjen hozzá.

## 7.5. Csoport

A felhasználóhoz hasonlóan a csoportok esetében is tárolásra kerül az ID-juk, nevük és a létrehozó felhasználójuk. Emellett rendelkeznek egy tömb adattaggal, ami azon résztvevők ID-ját tartalmazza, akik tagjai a csoportnak. Ez a lista lehet üres is, és folyamatosan frissül, ha egy új résztvevőt hozzáadnak vagy eltávolítanak. Ha egy résztvevő törlésre kerül, a csoport tagjai közül is törlődik.

## 7.6. Termék

A termék objektumok nem rendelkeznek saját kollekcióval, hanem nyugtákon belül tárolódnak. Rendelkeznek névvel, árral és darabszámmal, valamint a fizetők tömbjével. Itt résztvevők azonosítói tárolódnak.

## 7.7. Nyugta

A képen szöveg, képernyőkép látható

Automatikusan generált leírás

7.7.1. ábra – Példa egy nyugta felépítésére

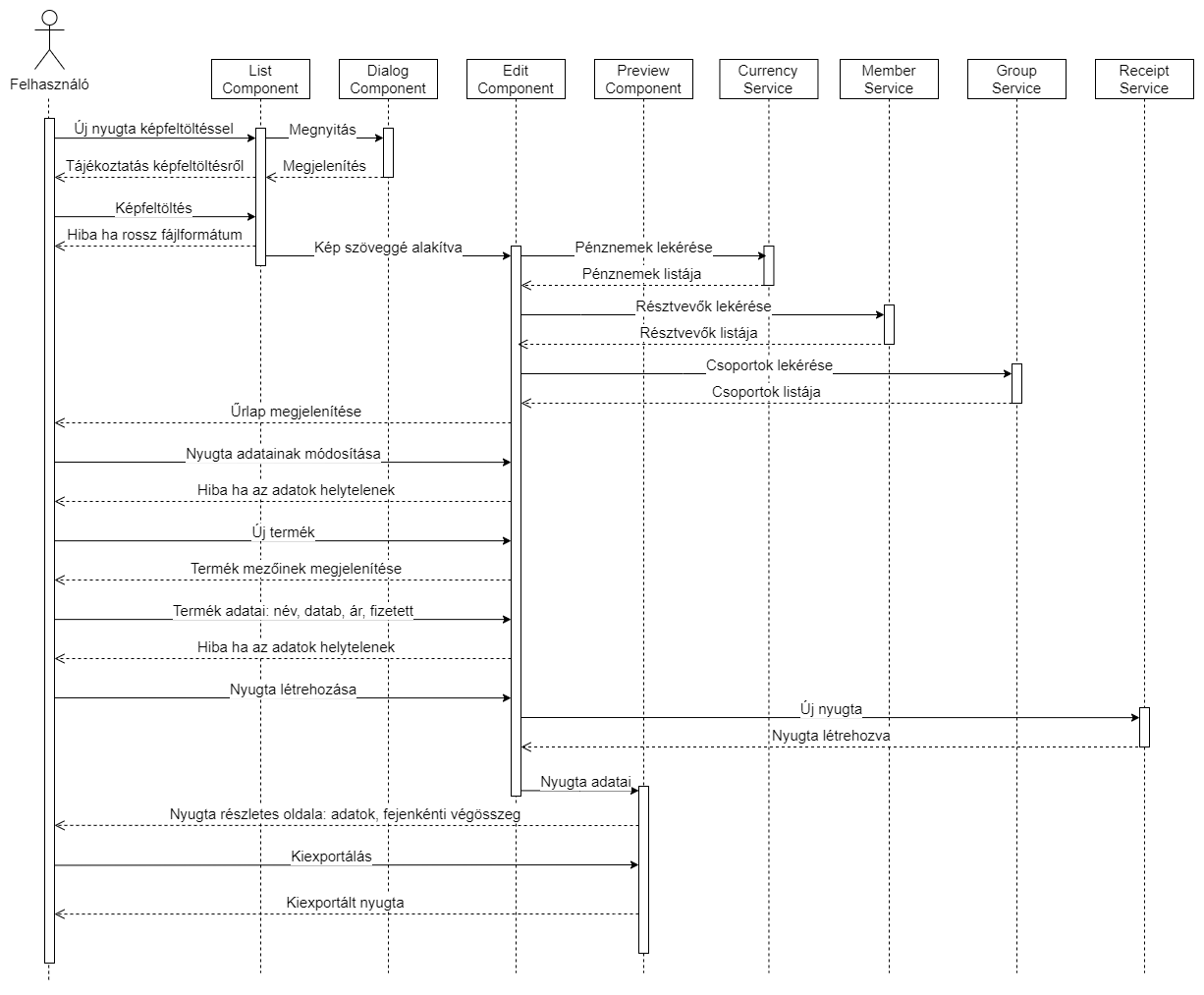
Nyugtákat az ID-juk azonosít, és rendelkeznek egy bolttal, végösszeggel és egy időbélyegként tárolt dátummal. A létrehozó felhasználó ID-ja is rögzítésre kerül, valamint a pénznem és a teljes nyugtáért fizetett résztvevő azonosítója is külön mezőben tárolódik. A termékek egy tömbben találhatóak, de itt hivatkozás helyett a teljes objektum tárolásra kerül.

A nyugta résztvevőit egy olyan tömb tartalmazza, ami szintén objektumokat tárol. Itt nem csak a résztvevő azonosítója található, hanem egy szám is, ami azt jelzi, hogy az adott résztvevő a nyugta rá eső részéből eddig mennyit fizetett ki.

A 7.7.1. ábrán látható egy példa egy nyugtáról a Cloud Firestore adatbázisban.

# 8. A rendszer magasszintű folyamatai, működése

Az alkalmazás egyik magasszintű folyamata a nyugta létrehozása, ahogy ez a Nyugtafelosztó fő funkciója is. Ebben a fejezetben ezt a folyamatot fogom részletezni.



8.1. ábra – Szekvencia diagram nyugta létrehozásáról

A 8.1. ábra mutatja be egy nyugta létrehozásának szekvencia diagramját. Ez a nyugták listáján a *Képfeltöltés* gombra nyomással kezdődik, és a kiexportált nyugta letöltésével fejeződik be. A folyamatot bizonyos lépések megelőzik, hiszen résztvevők létezése szükséges egy nyugta létrehozásához, ahogy a korábbi nyugták kilistázása is megtörténik a lista komponensen.

Ahogy a diagramon látható, miután a felhasználó a nyugta létrehozásához a képfeltöltést választotta, egy dialógus komponens nyílik meg a listán, ami tájékoztatja a felhasználót a szövegfelismerés hibáiról és javaslatokat tesz a feltöltött képről a legjobb eredmény elérése érdekében. Ezt követően tudja a képet feltölteni, amit a rendszer ellenőriz. Nem megfelelő fájlformátum vagy egyéb hibából eredő sikertelen fájlfeltöltésről üzenetet kap a felhasználó.

Megfelelő fájl feltöltése esetén a képen található betűk felismerése következik. A felismert szövegből a rendszer kiszűri a releváns információt, és az adatokat egy új nyugta objektumba írja.

A szerkesztési oldal megnyitása után az *EditComponent* lekéri a pénznemek listáját a *CurrencyService* segítségével. Ez azért szükséges, mivel a nyugta pénznem mezőjében a felhasználó ezek közül képes választani. Hasonló módon kéri le a *MemberService*-n és a *GroupService*-n keresztül a felhasználó által létrehozott résztvevők és csoportok listáját. A résztvevők a fizető kiválasztása során jelennek meg egy lenyíló listában, ahogy a listáért fizető személyt is közülük lehet kiválasztani – ebben az esetben viszont egy szűrt listából, ahol csak a nyugta már hozzáadott résztvevői szerepelnek. Csoportokra csak a későbbiekben lesz szükség, termékek fizetőjének kiválasztásakor, ahol résztvevőkkel vegyesen szerepelnek.

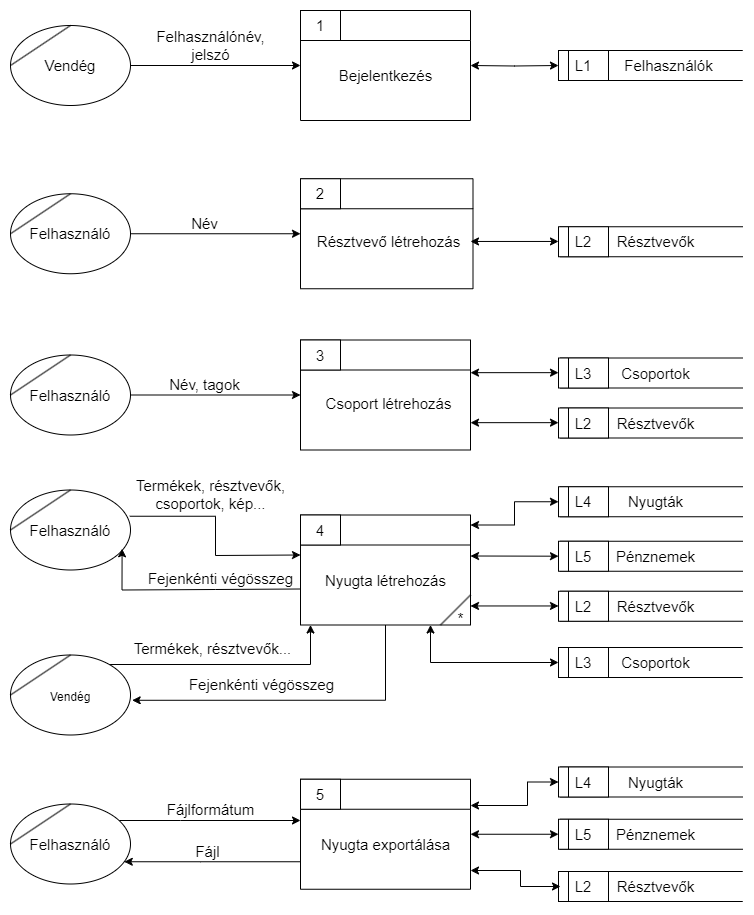
Miután minden szükséges adat rendelkezésre áll, a rendszer kitölti az űrlap mezőit a beolvasott kép adataival, amit utána megjelenít a felhasználónak. Sikeres szövegfelismerés után a mezők nagy részében már szerepelnek adatok, ide tartozik a bolt, a dátum, a pénznem, és egyes termékek neve, ára, darabszáma. Ám előfordulhat, hogy ezek üresen maradtak vagy hibásak, így a felhasználó képes módosítani őket. A mezők változtatása után az űrlap ellenőrzése történik, és hibaüzenet jelenik meg nem megfelelő adatok vagy hiányzó értékek esetén.

Az adatok módosítása folyamatba tartozik a nyugta résztvevőinek és fizetőjének, valamint az egyes termékekért fizető személyek, csoportok megadása, mivel ezeket képfeltöltéssel nem lehet megadni. Ebben az esetben is ellenőrzésre kerül az űrlap helyessége.

A felhasználónak lehetősége van további termékek hozzáadására az *Új termék* gombbal. Ekkor további mezők jelennek meg a felületen, ahol a név, darabszám, ár és a fizetők adhatók meg. Ha a felhasználó ezeket is kitöltötte, ismét ellenőrzés történik az értékek helyességéről.

Megfelelő adatok megadása után a nyugta elmenthető. Az űrlap értékei egy nyugta objektumba kerülnek, ami a *ReceiptService*-n keresztül jön létre az adatbázisban. Ezután megnyílik a nyugta részletes oldala, ahol az új objektum adatai szerepelnek. Itt tekintheti meg a felhasználó az egyes résztvevőkre eső, fizetendő összegeket is.

A kész nyugtát többek között kiexportálni is lehetséges. Amint a felhasználó kiválasztotta a kívánt fájlformátumot (CSV vagy JSON), a *PreviewComponenten* belül a nyugta szükséges értékei formázva fájlba íródnak, ami felhasználó eszközén letöltésre kerül.



8.2. ábra – Logikai adatfolyam diagram

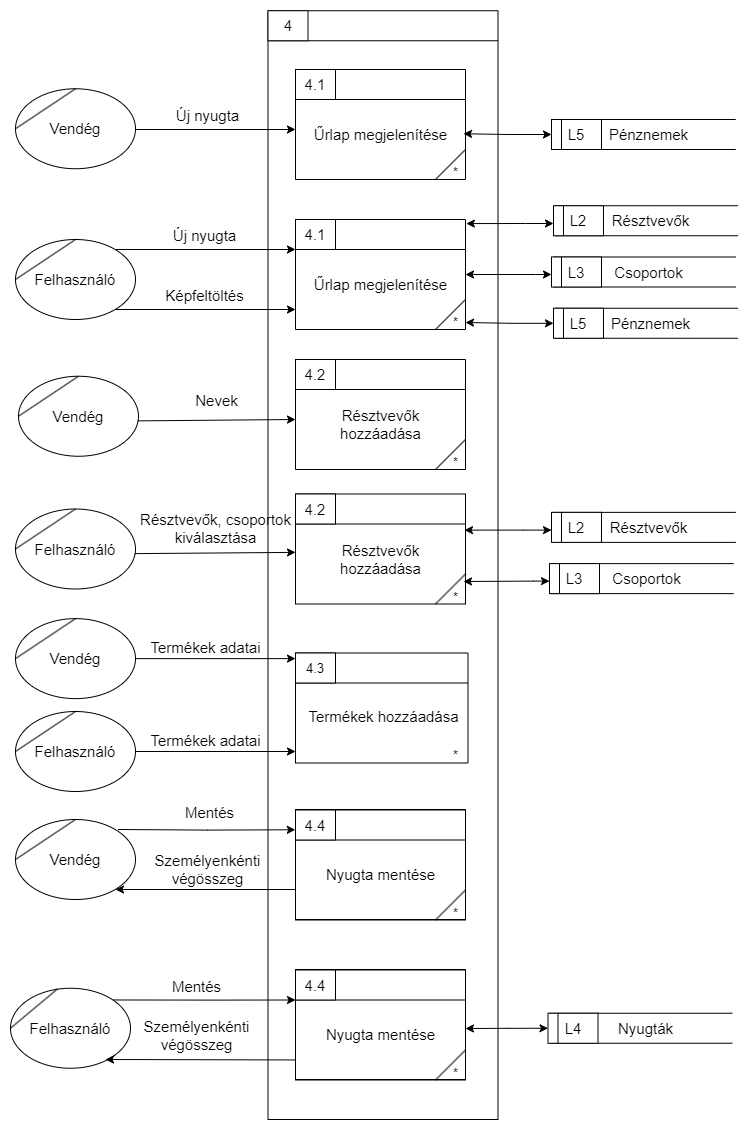
A 8.2. ábrán látható egy logikai adatfolyam diagram, ami nagyobb kitekintést enged a rendszerre. Itt a nyugta létrehozását megelőző lépések is láthatók, valamint a kiexportálás a létrehozáson kívül van feltüntetve.

A diagram fontos tényezője, hogy látható, a folyamatok mely adattárakkal kommunikálnak. Például adatok áramolnak egy nyugta létrehozásának folyamata és a nyugták, pénznemek, résztvevők, valamint csoportok adattárai között.

Egy nyugta létrehozása összetett folyamat, ezért részfolyamokra bontható. Erről egy második szintű adatfolyam diagramot is készítettem, ami megtekinthető a 8.3. ábrán. Ezen a diagramon bizonyos folyamatok kétszer is szerepelnek, mivel a rendszer máshogy viselkedik attól függően, hogy az adatokat bejelentkezett felhasználó vagy vendég küldte-e.

Az ábrán 4.1. számmal jelölt űrlap megjelenítése folyamat csak a pénznemeket kéri le, ha vendégről van szó. Bejelentkezett felhasználó esetén a résztvevőket és csoportokat is kilistázza, hiszen a nyugta résztvevőit, a teljes összeget fizető személyt és a termékekért fizetőket is közülük kell kiválasztani – ez látható a 4.2. folyamat esetében is. Ez utóbbi folyamat során a vendégek csak a résztvevők neveit használhatják, nem a tárolt objektumokat.

A mentés is különbözik számukra, mivel a vendég adatai csak a local storage-ban tárolódnak, míg a felhasználó esetében az adatbázisba kerülnek. A folyamat végén mindketten megkapják a személyenként felosztott nyugtát, ami a bejelentkezett felhasználó esetében további lehetőségekkel és egyéb adatok megjelenítésével is jár.



8.3. ábra – Második szintű logikai adatfolyam diagram

# 9. Fontosabb kódrészletek ismertetése

## 9.1. Regisztráció

A *RegisterComponenten* belül található regisztrációs űrlap kitöltésével kezdődik a folyamat. A mezők értékei Angular[[1]](#Angular) *Validators* osztályának metódusaival kerülnek ellenőrzésre. A hibaüzenetek *mat-error* tagokként jelennek meg. Helyes kitöltés esetén a *Regisztráció* gomb elérhetővé válik, ami kattintásra meghívja a 9.1.1. ábrán látható *onSubmit* függvényt.

A képen szöveg, képernyőkép, szoftver látható

Automatikusan generált leírás

9.1.1. ábra – *onSubmit* függvény

A függvény először igazra állítja a *progressBar* értékét, ami egy töltődést jelző csíkot eredményez a fejléc alatt, ezáltal a felhasználó tudhatja, hogy adatok feldolgozása történik. Az *AuthService* *signup* metódusának átadja az űrlapból származó e-mail cím és jelszó értéket, ami az *AngularFireAuth*-on belüli *createUserWithEmailAndPassword* segítségével hoz létre egy új felhasználót, és egy *Promise*-szal tér vissza.

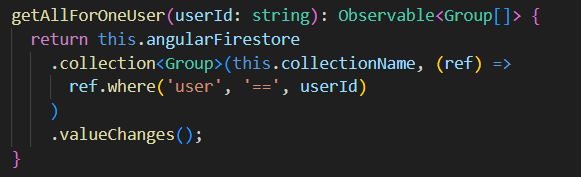
Ezután a létrjött felhasználó e-mail címére elküldi a visszaigazoló üzenetet, és egy *User* objektumot is létrehoz, amit a *UserService* *create* metódusának használatával ment el. A *createUserWithEmailAndPassword* lefutása után a felhasználó automatikusan bejelentkezik, de mivel ennek megerősített e-mail cím nélkül nem szabad megtörténnie, ezért a rendszer egyből ki is jelentkezteti őt az *AuthService* *logOut* függvényével, és a localStorage-ban létrejövő *use*r értéket is kitörli. Végül az *openDialog* hatására megnyílik a *DialogComponent*, ami *data* tulajdonságában átadott adatokat jeleníti meg dialógus formában.

Ha a felhasználó létrehozása nem sikerülne, és közben hiba dobódik, akkor azt a rendszer elkapja, és meghívja a *handleError* függvényt. Itt a hibakód ellenőrzése történik, ami alapján a megfelelő hibaüzenet állítódik be.

A folyamat végén az adatfeldolgozást jelző csík is megszűnik, így tájékoztatva a felhasználót, hogy folytathatja az alkalmazás használatát.

## 9.2. Csoportok létrehozása

Csoportok létrehozása a *GropuModule*-on belüli *ListComponent* felületén történik. A lista megnyitása után a *GroupService getAllForOneUser* metódusával az összes csoport, amit a felhasználó létrehozott listázásra kerül. Ez az *AngularFirestore* *collection* függvényének segítségével történik, aminek átadásra kerül a kollekció neve, majd szűrés történik a felhasználó ID-jára, mint ahogy az a 9.2.1. ábrán is látható.



9.2.1. ábra – *getAllForOneUser* függvény csoportok lekérésére

A megkapott csoportokból két *MatTableDataSource* osztálypéldány jön létre, az egyik a teljes listát tartalmazza, míg a másik annak egy másolatát, amin szűrések történhetnek. Az adatok táblázatba kerülése után arra egy név alapján csökkenő rendezést alkalmaz a rendszer.

A lista fölött látható *Új* gombra nyomás által hívódik meg az *openEdit* függvény. Ez az *editComponentet* nyitja meg dialógus formában, ahol a csoportok listája mellé a felhasználó által létrehozott összes résztvevő is lekérésre kerül.

A megjelenő űrlap név mezőjét a *Validators* osztály segítségével ellenőrzi a rendszer, emellett saját validátor függvényeket is létrehoztam. Például az *existenceValidator* felel azért, hogy ne létezzen egy felhasználónak két egyforma nevű csoportja – ehhez a lekért csoportokon végigiterál, és ha egyezést talál, egy hibaüzenetet jelenít meg.

A csoport tagjainak kiválasztásához az Angular Material[[19]](#AngularMaterial) Chips és Autocomplete komponensét használtam. Ezek rendelkeznek saját tulajdonságokkal, amik meghatározzák, hogy melyik függvény hívódjon meg, ha például a felhasználó belekattint a mezőbe, esetleg kiválaszt vagy eltávolít egy elemet. Szöveges érték megadása esetén a *filter* függvény hívódik meg, ami szűri a résztvevők nevét az alapján, hogy melyik tartalmazza a beírt értéket, így ezután csak a megfelelő adatok jelennek meg a lenyíló listában. Résztvevők hozzáadására két függvény, az *addMember* és a *selectedMember* áll rendelkezésre, mindkettő hozzáad egy új elemet a kiválasztott résztvevők listájához. Az előbbi az Autocomplete (szöveg beírása, majd enter vagy vessző billentyű lenyomása), az utóbbi a Chips (a lenyíló listából kattintással egy elem kiválasztása) komponenssel történt interakció hatására fut le.

A *Mentés* gomb megnyomása az *onSubmit* függvényt hívja meg. Itt létrejön egy új tömb, amibe a mezőben szereplő tagok ID-i kerülnek. Ezt a rendszer átalakítja *Set* adattípusra, aminek tulajdonsága, hogy minden értéket csak egyszer tárol, így kerülnek eltávolításra azok a résztvevők, akik esetleg többször lettek megadva. Utána ezt visszaalakítja tömbbé, amit átad egy új *Group* típusú objektumnak. A csoport többi mezőjének is értéket ad, a nevet az űrlap *Név* mezőnek értéke alapján, a létrehozó felhasználót pedig a local storage-ban tárolt ID alapján állítja be.

Végezetül a *GroupService create* metódusának segítségével a létrehozott csoportot a rendszer az adatbázisba is elmenti, a dialógus ablakát pedig bezárja. A *ListComponent* a *MatDialogRef* osztály *afterClosed* metódusának köszönhetően értesül az eseményről, és az új csoportot figyelembe véve rendezi a táblázatot.

## 9.3. Nyugta beolvasása képről

A képfeltöltés a *ReceiptModule ListComponent* oldalán történik a *Képfeltöltés* gombra kattintással. Ekkor az *openDialog* függvény fut le, ami megnyitja a *DialogComponentet*, hogy tájékoztassa a felhasználót a képfeltöltés hátrányairól és a felismerést segítő kép feltöltéséről. Ez a komponens rendelkezik egy *submitEvent* nevű *EventEmitterrel*, ami akkor bocsát ki eseményt, ha a felhasználó a dialóguson is a *Képfeltöltés* gombot választotta. Ennek hatására a rendszer a listán belüli fájlfeltöltő HTML input elemre nyom, ami a felhasználó elől rejtve van.

Ezután az *uploadImage* függvény kerül meghívásra. Itt ellenőrzésre kerül, hogy a felhasználó ténylegesen töltött-e fel fájlt, és annak megfelelő-e a fájlformátuma; ha nem, akkor egy dialógus nyílik meg, ami tájékoztatja őt a hibáról.

A rendszer létrehoz egy üres nyugtát, aminek a felhasználó mezőjét a local storage *user* értéke alapján inicializálja, a többi mezőbe pedig a típusnak megfelelően üres szöveg vagy lista, esetleg 0 kerül.

A képen szöveg, képernyőkép, szoftver, Multimédiás szoftver látható

Automatikusan generált leírás

9.3.1. ábra – Tesseract.js[[7]](#TesseractJs) *Worker* használata

Mint ahogy az a 9.3.1. ábrán is látható, a Tesseract.js[[7]](#TesseractJs) *Worker* osztályát a *createWorker* metódussal példányosítottam, *’hun’* paraméter átadásával, aminek köszönhetően a *worker* magyar nyelvet képes felismerni. A *target.files[0]* tartalmazza a feltöltött képfájlt, ami – miután a létezése ellenőrzésre került – átadódik a *Worker recognize* metódusának. Ez felel a képen való szöveg felismeréséért, ami utána a *ret* változóban tárolódik.

A *ret.data* tárolja a képfelismeréshez kapcsolódó adatokat. Ezen belül több tulajdonság is található, például a *confidence*, ami egy százalékos érték a felismert szöveg helyességének biztosságáról. A szöveg többféleképpen is elérhető, a *text* attribútum egyetlen sztringkén, a *paragraphs*  a bekezdésenként, a *lines* a soronként, a *words* szavanként a *symbols* pedig betűnként egy-egy tömbben tárolja azt.

Én a nyugták felépítése miatt a *lines* attribútumot választottam, aminek elemein a program egy for each ciklussal iterál végig. A 9.3.1. ábrán a cikluson belüli kódrészlet helytakarékosság miatt nem szerepel. Itt kapnak értéket a nyugta tulajdonságai.

A bolt neve az első nem üres sor tartalma, hiszen egy hagyományos nyugtán is itt helyezkedik el.

Ha a sor tartalmaz pontot, akkor annak szavait a rendszer egyenként megpróbálja dátummá alakítani. Abban az esetben, ha sikerrel jár, ezt az értéket menti el a vásárlás dátumaként.

A végösszeget általában az *összesen*, esetleg az *összeg*  vagy *fizetendő* szavak előzik meg, így az ezt követő szám és szöveg lesz a nyugta végösszege valamint pénzneme. Ehhez a *converToSumAndCurrency* metódust használja, ami eltávolítja a sorból a felesleges karaktereket, majd végigiterál a szavakon. Ha számot talál, akkor azt hozzáadja a szöveg típusú *number* változóhoz. Azért szükséges, hogy szövegként tárolja, mivel előfordulhat, hogy a nyugtán a végösszeg ezresenként szóközökkel van tagolva, és így külön szavakban találhatók az összeg számjegyei. Számmá konvertálás csak az összes számjegy megtalálása után történik. A pénznemet úgy állítja be, hogy minden talált szót, ami nem szám, megpróbál megkeresni a pénznemek szimbólumai és nevei között – ha sikerül megtalálnia, azt az értéket menti el.

A nyugtában található bizonyos szavakat, mint a *cikkszám*, *sorszám* és *részösszeg* a rendszer figyelmen kívül hagy. A termékek általában a *nyugta* szó után helyezkednek el, így ha ez a szöveg megjelenik, a *beginning* érték hamisra állítódik, ami jelzi, hogy a feldolgozás elhagyta a nyugta elejét. A végösszeg és a pénznem beállítása a termékek végét jelzi, így ekkor az *end* változó kap igaz értéket, mert ettől kezdve már a nyugta vége következik.

Termékek beolvasása akkor történik, ha elhagytuk a *nyugta* szót, de még nem értük el a végösszeget, tehát a *beginning* és az *end* is hamis. Megvizsgálásra kerül, hogy a sor számmal kezdődik-e, és ha igen, ez lesz a következő termék darabszáma. Egyéb esetben termékről van szó, ami a *convertToProduct* funkció segítségével kerül feldolgozásra, ami a sort és a darabszámot is megkapja. Ha az előtte lévő sorban nem darabszám szerepelt, akkor az automatikusan 1 lesz.

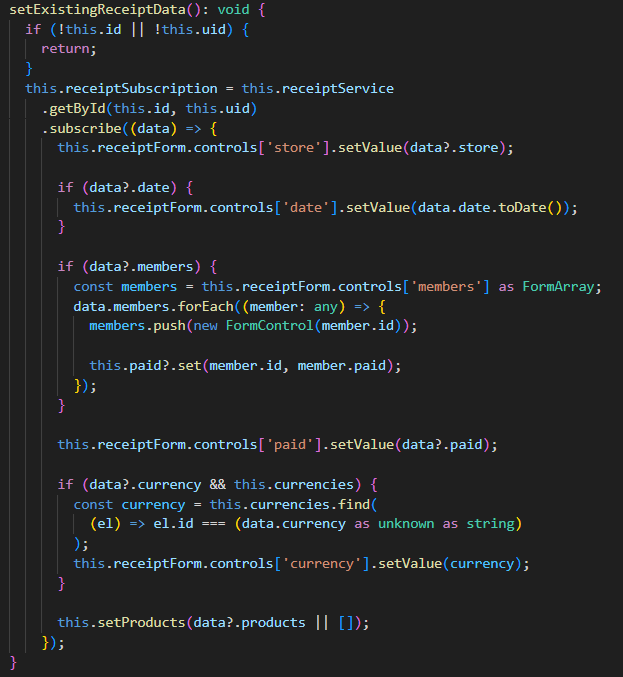
A *convertToProduct* először a termék árát olvassa be, a soron visszafelé haladva, hiszen az ár a termékek jobb oldalán található. A végösszegnél említett indok miatt a számok itt is szövegként kerülnek először tárolásra. A beolvasás addig halad, amíg el nem ér egy olyan szöveget, amit nem lehet számmá konvertálni, és a szöveg nem mínusz jel, hiszen negatív számok is szerepelhetnek a nyugtán. Emellett a beolvasás folytatódik, ha még nem sikerült egy számot sem beolvasni, mivel előfordulhat, hogy az ár mellett a pénznem is szerepel. A termék neve az árat megelőző szavak összekapcsolásából jön létre. Ha mindkét értéket sikerült kiolvasni, a rendszer egy *Product* objektumba menti őket, a metódus ezzel tér vissza. Az így felismert termékek a nyugta *product* adattagjában tárolódnak.

A beolvasás végét a *Worker terminate* metódusa által visszaadott *Promise* teljesülése jelzi. Ezt követően a létrejött nyugta objektum ideiglenesen tárolásra kerül a *local storage*-ban *receipt* néven, majd megnyílik a nyugta *EditComponent* komponense a */receipt/upload* URL használatával.

## 9.4. Nyugta szerkesztése

Ez a folyamat a *ReceiptModule EditComponent* komponensén belül történik. Ez az oldal több esetben megnyílhat: nyugta létrehozásakor bejelentkezett vagy vendég felhasználó által, képfeltöltéskor és szerkesztéskor. Ekkor más az URL utolsó tagja is (a fenti sorrendben: *new*, *guest*, *upload*, *edit:id*), valamint a komponens viselkedésében is vannak kisebb eltérések.

A pénznemek listája minden esetben lekérésre kerül a *CurrencyService getAll* metódusa által, de résztvevők és csoportok listája csak bejelentkezett felhasználó esetén kap értéket. Szerkesztés (*edit:id*) és képfeltöltés (*upload*) esetén már létezik egy nyugta objektum, aminek értékei előbbi esetén a *setExistingReceiptData*, utóbbinál pedig a *setUploadedReceiptData* meghívásával kerülnek az űrlap mezőibe. A kettő közötti fő különbség, hogy a *setExistingReceiptData* a *ReceiptService getById* metódusa segítségével kéri le a nyugtát, míg a *setUploadedReceiptData* a local storage *receipt* értéke alapján dolgozik. Ezen kívül az adatokat hasonlóan állítják be, és a *setProducts* függvényt mindkettő meghívja, ahol minden termék számára külön *FormGroup* jön létre a *formBuilder.group* futtatásával. A *setExistingReceiptData* függvény megtekinthető a 9.4.1. ábrán.



9.4.1. ábra – *setExistingReceiptData* függvény

Mivel ezen az oldalon is szerepelnek az Angular Material[[19]](#AngularMaterial) Chips és Autocomplete komponensei, ezért itt is definiáltam csak ezekhez kapcsolódó funkciókat. A *filter* a pénznemek listáját szűri a mezőbe írt adatok alapján – ehhez hasonló a *filetPayer*, a *filterMember* és a *filterMembersAndGroups*, amik a *Fizetett*, a *Résztvevők* és a termékek *Fizet* mezőjének lenyíló listáit szűrik. Emellett minden Chips komponenshez tartozik egy-egy függvény, ami lekéri a megjelenítendő adatokat, valamint ami akkor fut le, amikor egy érték hozzáadásra vagy eltávolításra kerül.

A *Mentés* gomb megnyomása hatására az *onSubmit* funkció fut le, ami a nyugta objektum létrehozásáért és adatbázisba való feltöltéséért felelős.

A nyugta tagjai egy *Setbe* kerülnek, hogy a duplán szereplő résztvevők eltávolításra kerüljenek. Bejelentkezett felhasználó esetén a kiválasztott résztvevők ID-ja, vendég esetén a mező értéke kerül tárolásra.

Az űrlapban szereplő termékeken a rendszer for each segítségével megy végig. Ezen belül a termék fizetőin is végigiterál, amik a résztvevőkhöz hasonlóad egy *Setben* tárolódnak. Bejelentkezett felhasználó esetén ellenőrzésre kerül, hogy csoportról van-e szó, ebben az esetben a csoportot a rendszer felbontja, és a résztvevőit egyesével a termék fizetői közé adja, ellenkező esetben a résztvevő ID-ja kerül a *Setbe*. Vendég felhasználónál a *Fizet* mező értékei lesznek a fizetők. A termékekből ezután egy-egy *Product* objektum készül, ahol a név, mennyiség és ár a megfelelő mezőkből származnak, a fizetők pedig a *Set* tömbbé alakításából keletkeznek. Emellett minden termék ára hozzáadódik a nyugta teljes végösszegéhez.

Miután a termékek listája értéket kapott, a résztvevők tömb formátumúra alakítása is megtörténik. Ezt követően a duplikátumokat már nem tartalmazó résztvevőkön is végigmegy a rendszer. Egy-egy objektum kerül hozzáadásra a nyugta résztvevői közé, ami bejelentkezett felhasználó esetében az ID-t és az eddig kifizetett összegét tartalmazza, vendég esetén pedig egy ID-t és egy nevet, amik megegyeznek a mező értékével.

A résztvevők kifizetett összege új nyugta esetén 0, kivéve azét, aki a nyugtáért fizetett, ő a nyugta rá eső értékét már kifizette. Már létező nyugtánál ez az érték nem változik szerkesztésnél, hanem a korábban beállított értéke marad.

A *Receipt* objektum a nyugta fent részletezett adataiból, valamint a további mezők értékeből jön létre. Vendég esetén a local storage-ban egy *receipt* objektum jön létre, míg bejelentkezett felhasználónál a *ReceiptService* *create* (új nyugta) vagy *update* (létező nyugta frissítése) metódusa segítségével az adatok az adatbázisban is rögzítésre kerülnek.

A mentés befejeztével navigáció történik a nyugta részletes oldalára.

## 9.5. Nyugta részletes oldala

A *ReceiptModule PreviewComponent* komponense reprezentálja a nyugta részletes oldalát. A felület megnyitása után bejelentkezett felhasználó esetén a *ReceiptService getById* metódusával kéri le a rendszer a megjelenítendő nyugtát, aminek azonosítóját az URL-ből olvassa ki. Ha a keresett nyugta nem létezik, a nyugták listájára történik átirányítás. Vendég felhasználó esetén a local storage-ban szereplő nyugta objektum kerül átkonvertálásra. Amennyiben a vendég úgy próbálná megnyitni a felületet, hogy a local storage-ban nem szerepel nyugta objektum, vagy a nem megfelelő URL-t próbálta használni, őt a nyugta létrehozásának oldalára irányítja át a rendszer.

A nyugta egy *Receipt* objektumként tárolódik az oldalon. A *currency* mezőjében található pénznem ID-ját használva a rendszer a pénznem objektumot is lekéri a *CurrencyService getById* metódusának használatával.

A *calculatePrices* függvény fut le ezek után, ami a tagok fizetendő összegét állítja be. Ehhez először a *needToPay* *Map*nek kulcs-érték párokat állít be, a kulcs anyugta egyes résztvevőinek ID-ja, az ehhez tartozó értékek pedig kezdetben mind 0-k. Ezt követően végigiterál a termékeken, azokon belül pedig az egyes fizető résztvevőkön. Minden egyes résztvevő esetében a termékek árát elosztja annak fizetőinek számával, és ezt hozzáadja a *needToPay* adott résztvevőhöz tartozó, eddig számolt értékhez.

Következőként a *getMembersFromReceipt* függvény kerül meghívásra. Itt is végigmegy a nyugta résztvevőin, ahol vendégek esetén mindegyikhez hozzáadja a *needToPay* megfelelő értékét, kerekítve. Bejelentkezett felhasználók esetén a résztvevők itt kerülnek lekérésre, a *MemberService getById* metódusának használatával. A résztvevők *pays* attribútuma itt kerül beállításra a *needToPay* értékeinek alapján, valamint a *paid* mezőjük is értéket kap a nyugtában tárolt azonos nevű adattag alapján. Előfordulhat, hogy adott ID-jú résztvevő nem létezik, ha az törlésre került, ebben az esetben egy új *Member* objektum jön létre, aminek neve *\*Törölt résztvevő\**. Az így beállított résztvevők egy listához lesznek adva, ami az oldalon egy táblázatként jelenik meg.

A felhasználónak lehetősége van szerkeszteni a nyugtát, ekkor az *editReceipt* függvény megnyitja a szerkesztési oldalt. A törlés ikonra kattintva a *deleteReceipt* hívódik meg, ami egy *DialogComponent* dialógust nyit meg figyelmeztető üzenettel. Ha a felhasználó itt az *Ok* gombot választja, a *submitEvent* hatására kitörlődik a nyugta a *ReceiptService delete* függvényének használatával, majd navigáció történik a nyugták listájára.

A képen szöveg, képernyőkép, Betűtípus, szoftver látható

Automatikusan generált leírás

9.5.1. ábra – Nyugta letöltéséért felelő kód

A nyugtát ki is lehet exportálni, a letöltés ikonra kattintva a felhasználó egy lenyíló menüből választhatja ki a CSV vagy JSON fájlformátumot. A fájl kiterjesztése átadásra kerül a *downloadReceipt* metódusnak. Ez először kiszűri a nyugta objektumból a kimentendő adatokat, ami az egyes résztvevők neve, fizetett és fizetendő összege, valamint a nyugta boltja, formázott dátuma, pénzneme, végösszege és a fizető neve. Ezután készít egy URL-t, ami az adattípust, a karakterkészlet, és az *encodeURIComponent* metódussal kódolt adatokat tartalmazza.

CSV választása esetén a kódolás a *createCsv* metódus által visszaadott szövegen történik. Itt először definiálásra kerül a CSV fájl fejléce, ami a nyugta kulcsait tartalmazza nagy kezdőbetűssé alakítva, majd ezek pontosvesszővel összefűzve a *rows* tömbbe kerülnek. A kulcsokhoz tartozó értékek hasonló módon pontosvesszővel elválasztva lesznek hozzáadva a tömbhöz. Ezután egy üres szöveg is a tömbbe kerül, ezzel lesz egy sor hely a nyugta alapadatai és a résztvevők táblázata között. A résztvevők esetében ismét pontosvesszővel lesznek összefűzve a kulcsok, majd az értékek. Végül a *rows* tömb elemei egy-egy új sor karakterrel elválasztva lesznek visszaadva egyetlen szövegként.

A HTML kódban található egy elrejtett link elem *download* ID-val, a letöltendő fájl URL-jét a rendszer ennek a *href* attribútumának adja értékül. A fájl neve is itt kerül beállításra a link elem *download* tulajdonságaként, ami a következő formátumot követi: *bolt\_dátum.típus*.

Az URL létrehozása, a *download* HTML link elem attribútumainak megadása és az arra való kattintással a letöltés megkezdése látható a 9.5.1. ábrán, ahol a *downloadData* a nyugta kimentendő adatait tartalmazza.

A nyugta részletes oldalán lehetőség van a résztvevők által kifizetett összeg módosítására is. A résztvevők táblázatában minden sorban található egy jelölőnégyzet, aminek megnyomása a *payForMember* funkciót hívja meg, paraméterként a hozzá tartozó résztvevővel. Itt a nyugta résztvevői közül megkeresi a megfelelőt, akinek a *paid* attribútumát a *needToPay Mapben* hozzá tartozó értékre állítja. A nyugtán ezután átalakítást végez a *transformReceipt* függvénnyel, ami eltávolítja a nyugtából a csak a frontenden használt értékeket, mint a formázott dátumot, a résztvevők nevét, felhasználóját és fizetendő összegét, valamint a *currency* mezőben is csak annak ID-ját tárolja. Az így átalakított nyugtát a *ReceiptService update* metódusával tölti fel a szerverre. Végül újraszámolja a *paidSum* értéket, ami a résztvevők által összesen kifizetett pénzmennyiség, a résztvevők *paid* attribútuma alapján.

Ha egy résztvevő nem a teljes összeget fizette ki, hanem annak csak egy részét, a felhasználó ezt is be tudja állítani a résztvevők táblázatában látható szerkesztés gombra nyomva. Ekkor az *openPaidDialog* hívódik meg, ami a *PaidComponentet* nyitja meg dialógus formában. A komponensnek átadódnak a nyugta tagjai és a pénznem szimbóluma.

A *PaidComponenten* belül először minden résztvevőnek létrejön egy űrlapmező, aminek kezdeti értéke az általa eddig fizetett összeg. Az űrlap mentésekor az *onSubmit* függvény hívódik meg, ami a résztvevőknek beállítja az új fizetett értékét, elküldi a *PreviewComponentnek* a *submitEvent* eseményt a résztvevőkkel, és bezárja a dialógust.

A *PreviewComponenten* belül a *submitEvent* hatására a rendszer végigiterál a nyugta résztvevőin, és beállítja nekik az új fizetett értéküket a megkapott résztvevők tömbje alapján, valamint az összesen fizetett mennyiséget is újraszámolja. Ezt követően a korábban már részletezett *transformReceipt* használatával átalakítja a listát, amit a *receiptService update* metódusával küld el az adatbázisba.

A *PaidComponent* megnyitása és az eredmény hatására a résztvevők fizetett értékének módosítása látható a 9.5.2. ábrán.



9.5.2. ábra – *openPaidDialog* függvény

# 10. Tapasztalatok, továbbfejlesztési lehetőségek

## 10.1. Alkalmazás megtervezése

Az Nyugtafelosztó fejlesztését a tervezési fázissal kezdtem, amikor még egészen más elképzeléseim voltak az alkalmazásról. Elsőként számos diagramot elkészítettem, amik akkor útmutatást adtak a kód alapjának elkészítésével, de ahogy haladtam az implementálással, egyre kevesebb részletet tartottam meg belőlük.

Az egyik funkció, ami sok változáson ment keresztül, az a nyugta létrehozása volt. Az első ötletem alapján képfeltöltéssel csak termékeket lehetett volna hozzáadni, és a nyugta nem kapott volna külön részletes oldalt, hanem a szerkesztési oldal alján egy dinamikusan frissülő táblázatban jelentek volna meg a fejenkénti végösszegek. Emellett a vendégek résztvevőket sem adhattok volna hozzá a nyugtájukhoz.

Bár a kezdeti tervek nem mindenhol egyeznek a végeredménnyel, ez nem jelent problémát az elkészült alkalmazás tekintetében. A korai ötletek felülvizsgálásával végezetül sikerült olyan megoldásokat kitalálnom, amik jobban illeszkednek az alkalmazás motivációjához és céljához.

## 10.2. Ütermterv, feladatok

A fejlesztési fázis elején készítettem egy ambiciózus ütemtervet, amit sajnos nem mindig sikerült tartanom. Ennek egyik fő oka az volt, hogy az implementálás során további új funkciók jutottak eszembe, amik az alkalmazás javára válhatnak, és ezek megtervezésével valamint megvalósítása nem számoltam az ütemterv elkészítése során.

Ami az időm megtervezésében sokat segített, az a GitLab[[6]](#GitLab)Issue board funkciója volt. Ide vettem fel a feladatokat, amiket el kellett végeznem, amiket próbáltam kisebb részekre, esetleg részfeladatokra osztani. Ez által mindig tudtam, hogy nagyjából mennyi van még hátra a fejlesztésből, emellett egy-két hetes időszakokban mindig előre el tudtam tervezni, hogy melyikeket fogom megoldani.

## 10.3. Szakmai ismeretek

A szakdolgozatomhoz azért választottam az Angular[[1]](file:///C:\Users\hanna\OneDrive\Dokumentumok\Szakdoga\nyugtafeloszto\szakdolgozat\SomogyváriHanna_GLXGEU_szakdolgozat.docx#Angular) keretrendszert Google Firebase[[2]](file:///C:\Users\hanna\OneDrive\Dokumentumok\Szakdoga\nyugtafeloszto\szakdolgozat\SomogyváriHanna_GLXGEU_szakdolgozat.docx#GoogleFirebase)backenddel, mivel már rendelkeztem tapasztalattal ezek terén, emellett a tudásom továbbfejlesztésére is lehetőség nyílt.

A fejlesztésben nehézséget okozott bizonyos Angular Material[[19]](#AngularMaterial) komponensek használata. Ennek oka az volt, hogy a komponensek dokumentációja általában csak az alap használati eseteket fedi le, én viszont bonyolultabb megoldásokat szerettem volna megvalósítani, mint az Autocomplete és Chip komponensek egymásba ágyazása, és ezekben adatbázisból lekért elemek megjelenítése. Akkor is problémákba ütköztem, amikor a Material komponenseket személyre akartam szabni, mivel azok előre meghatározott kinézettel rendelkeznek, aminek bizonyos tulajdonságait bonyolult megváltoztatni.

A Google Firebase[[2]](file:///C:\Users\hanna\OneDrive\Dokumentumok\Szakdoga\nyugtafeloszto\szakdolgozat\SomogyváriHanna_GLXGEU_szakdolgozat.docx#GoogleFirebase) esetén a felhasználók hozzáférését korlátozó biztonsági szabályok implementálása volt újdonság számomra. Itt kezdetben nehezen ment ezek meghatározása, hiszen nem voltam tisztában a lehetőségekkel és a megfelelő módszerekkel. Végül sikerült elsajátítanom a szabályok írását, és örülök, hogy ezzel a tudással is gazdagodhattam.

## 10.4. Továbbfejlesztési lehetőségek

Számos ötletem van, amivel az alkalmazást tovább lehetne fejleszteni, és ami jobb felhasználói élményt biztosítana.

Az egyik a fordítások bevezetése lenne, hogy a Nyugtafelosztó szélesebb közönség körében elérhetővé váljon. Ezzel kapcsolatban az egyik probléma a nyugták formátuma, ami országonként más-más lehet, ami megnehezítené a nyugtákról a megfelelő adatok kinyerését.

A képfelismeréssel kapcsolatban is van lehetőség a továbbfejlesztésre. Jelenleg ezt a Tesseract.js[[7]](#TesseractJs) végzi, ami az elképzelés bemutatásához megfelelő, de a korábban említett hibái miatt nem tekinthető teljeskörű megoldásnak. A probléma kiküszöbölhető lenne saját OCR motor fejlesztésével, ami kifejezetten nyugták felismerésére fókuszál, és nemcsak azok szövegét lenne képes felismerni, hanem a struktúra alapján azt is képes lenne kikövetkeztetni, hogy melyik karakterlánc mit reprezentál.

Az alkalmazás azzal is továbbfejleszthető lenne, ha a felhasználók képesek lennének interakcióba lépni egymással. Így nyugtákhoz nem csak résztvevő objektumokat lehetne hozzáadni, hanem más felhasználókat is, akik így a saját profiljukon látják, ha tartozásuk van, valamint erről értesítést is kapnak. Emellett az összeg kifizetését nem csak egy jelölőnégyzettel lehetne jelezni, hanem az alkalmazásban egy fizetési szolgáltatás igénybe vételével akár a tartozás átutalására is lehetne lehetőség.

A Nyugtafelosztó használatát tovább egyszerűsítené, ha mobil applikációként is rendelkezésre állna a felhasználók számára. Így a szövegfelismerésre nem csak kép feltöltésével lenne lehetőség, hanem a telefon kamerájával beszkennelt fotókat is képes lenne egyből feldolgozni. Egy mobilos applikáció offline használatra is lehetőséget nyújtana.

## 10.5. Összegzés

Összességében úgy érzem, végül sikerült egy olyan alkalmazást fejlesztenem, aminek funkciói kielégítik az elvárásokat, és egyszerűen, gyorsan lehet a használatával egy nyugtát felosztani.

Bár a fejlesztés során voltak nehézségeim, de ezek hatására sikerült tovább bővítenem a tudásom a használt technológiák körében. Remélem, hogy a megszerzett tapasztalatokat a későbbiekben is kamatoztatni tudom, és ezek által tovább fejleszthetem a Nyugtafelosztó alkalmazást.

# Irodalomjegyzék

1. Angular: <https://angular.io/> (Utolsó megtekintés: 2024. 04. 30.)
2. Google Firebase: <https://firebase.google.com/> (Utolsó megtekintés: 2024. 04. 30.)
3. draw.io: <https://app.diagrams.net/> (Utolsó megtekintés: 2024. 04. 30)
4. Visual Studio Code: <https://code.visualstudio.com/> (Utolsó megtekintés: 2024. 01. 18.)
5. Sass, SCSS: <https://sass-lang.com/> (Utolsó megtekintés: 2024. 01. 18.)
6. GitLab: <https://about.gitlab.com/> (Utolsó megtekintés: 2024. 04. 30.)
7. Tesseract.js: <https://tesseract.projectnaptha.com/> (Utolsó megtekintés: 2024. 01. 18.)
8. Zoho Expense: <https://www.zoho.com/expense/> (Utolsó megtekintés: 2024. 01. 21.)
9. Verify: <https://www.veryfi.com/> (Utolsó megtekintés: 2024. 01. 21.)
10. Expensify: <https://www.expensify.com/> (Utolsó megtekintés: 2024. 01. 21.)
11. Shoeboxed: <https://www.shoeboxed.com/> (Utolsó megtekintés: 2024. 01. 21.)
12. Dext Prepare: <https://dext.com/en/products/prepare> (Utolsó megtekintés: 2024. 01. 21.)
13. Invoice Maker, Invoice Calculator, Invoice Generator: <https://saldoinvoice.com/invoice-calculator/> (Utolsó megtekintés: 2024. 01. 21.)
14. EveryoneAllin: <https://everyoneallin.com/bill-splitter/online-bill-splitter/> (Utolsó megtekintés: 2024. 04. 25.)
15. Best Bill Splitter: <https://bestbillsplitter.com/> (Utolsó megtekintés: 2024. 01. 21.)
16. Kittysplit: <https://www.kittysplit.com/en/> (Utolsó megtekintés: 2024. 01. 21.)
17. IRIS KashFlow: <https://www.kashflow.com/> (Utolsó megtekintés: 2024. 01. 21.)
18. xero: <https://www.xero.com/> (Utolsó megtekintés: 2024. 01. 21.)
19. Angular Material: <https://material.angular.io/> (Utolsó megtekintés: 2024. 04. 30.)
20. AngularFire: <https://firebaseopensource.com/projects/firebase/angularfire/> (Utolsó megtekintés: 2024. 01. 22.)
21. Flaticon – alapértelmezett profilkép, készítette Freepik : <https://www.flaticon.com/free-icons/user> (Utolsó megtekintés: 2024. 02. 27.)
22. Ngx Image Cropper: <https://www.npmjs.com/package/ngx-image-cropper> (Utolsó megtekintés: 2024. 04. 25.)
23. Wikipédia – Forgalomban lévő pénznemek listája: <https://hu.wikipedia.org/wiki/Forgalomban_l%C3%A9v%C5%91_p%C3%A9nznemek_list%C3%A1ja> (Utolsó megtekintés: 2024. 04. 28.)

# Nyilatkozat

Alulírott Somogyvári Hanna programtervező informatikus szakos hallgató, kijelentem, hogy a dolgozatomat a Szegedi Tudományegyetem, Informatikai Intézet Szoftverfejlesztés Tanszékén készítettem, programtervező informatikus BSc diploma megszerzése érdekében.

Kijelentem, hogy a dolgozatot más szakon korábban nem védtem meg, saját munkám eredménye, és csak a hivatkozott forrásokat (szakirodalom, eszközök, stb.) használtam fel.

Tudomásul veszem, hogy szakdolgozatomat a Szegedi Tudományegyetem Diplomamunka Repozitóriumában tárolja.

Dátum

Aláírás

# Köszönetnyilvánítás

Szeretnék köszönetet mondani a témavezetőmnek, Dr. Bilicki Vilmos egyetemi adjunktusnak, aki útmutatást adott, felügyelte a haladásom, és tanácsokkal látott el.

Emellett szeretnék köszönetet mondani a családomnak a támogatásukért, megértésükért és az ötletemért való lelkesedésért.

Végezetül hálával tartozok a páromnak is, aki mellettem volt, ha nehézségekbe ütköztem, és velem együtt örült, amikor sikereket értem el.