

**Szegedi Tudományegyetem**  
**Informatikai Intézet**

**DIPLOMAMUNKA**

**Potyesz Máté**

**2024**

**Szegedi Tudományegyetem  
Informatika Intézet**

**Pénzügyi költségvetés kezelő keresztplatformos  
alkalmazás fejlesztése React Native-ben**

Diplomamunka

Készítette:  
**Potyesz Máté**  
programtervező informatikus  
szakos hallgató

Témavezető:  
**Dr. Bilicki Vilmos**  
egyetemi adjunktus

Szeged  
2024

## **Feladatkiírás**

A diplomamunka célja egy keresztplatformos pénzügyi költségvetés kezelő alkalmazás tervezése és fejlesztése React Native keretrendszer segítségével. Az alkalmazás kiemelt funkcionálitása a felhasználók személyes pénzügyeinek hatékony kezelése és jobb pénzügyi döntések támogatása. Az alkalmazás neve: "Credo". Az adatok tárolására a Google Firebase szolgáltatással történik.

Az alkalmazás feladata, hogy segítse a felhasználókat abban, hogy tudatosabbá váljanak a pénzügyeikkel kapcsolatban és minden egy helyen érhessenek el. Az analitikai ábrák, diagramok és egy AI chatbot segítségével a felhasználók könnyen nyomon követhetik a kiadásait, bevételiket és megtudhatják, hogyan alakulnak a pénzügyeik és befektetései az idő folyamán.

A Credo megfelel azoknak az üzleti és társadalmi szükségleteknek, amelyek a pénzügyi tervezés és felügyelet javítását igénylik. A felhasználók számára a hatékonyabb gazdálkodás támogatása mellett, a mesterséges intelligenciára épülő nyelvi modellekken alapuló fejlett chatbot bevonása is motiváló erőt jelenthet, melyek segítségével könnyedén tarthatják számon kiadásait és bevételiket.

## Tartalmi összefoglaló

- **A téma megnevezése:**

Pénzügyi Költségvetés Kezelő keresztplatformos alkalmazás fejlesztése React Native-ben.

- **A megadott feladat megfogalmazása:**

A diplomamunka célja egy keresztplatformos pénzügyi költségvetés kezelő alkalmazás tervezése és fejlesztése React Native keretrendszer segítségével, kiemelve a felhasználói élményt, személyre szabhatóságot, játékositást, pénzügyi tudatosságot, valamint a hozzáférhető nagy nyelvi modellek alkalmazását.

- **A megoldási mód:**

Az alkalmazás funkcionalitásának kidolgozása során React Native keretrendszert alkalmaztam a felhasználói felület kialakításához, ami szorosan kapcsolódik a Reacthoz, míg a szerveroldali funkciókhoz és az adattároláshoz a Firebase szolgáltatásait használtam. A fejlesztési folyamatot egy piackutatással indítottam, ahol elemzésre kerültek azok az alkalmazások, amelyek hasonló funkciókat látnak el majd a főbb funkciókat kiegészítettem saját ötletekkel.

- **Alkalmazott eszközök, módszerek:**

A fejlesztés során használt technológiák közé tartoznak a Typescript, Firebase (adatbázis- és felhőszolgáltatások), React Native, React, Expo a buildeléshez, Stripe az alkalmazáson belüli fizetéshez, néhány elemzési és analitikai eszközöket biztosító könyvtár, valamint pénzügyi és részvényekkel kapcsolatos API-k. A verziókövetést git-tel végeztem és githubra töltöttem fel a kódot. A technológiák elsajátítására a dokumentációkat, illetve egy React Native Udemy kurzust<sup>[1]</sup> vettettem igénybe. A mesterséges intelligencián alapuló chatbothoz az OpenAI nagy nyelvi modelljeit használtam.

- **Elért eredmények:**

Egy komplex, felhasználóbarát pénzügyi keresztplatformos alkalmazás, amely egyszerre kezeli és tartja számon a felhasználó összes pénzügyi adatait és modern technológiák felhasználásával egyedi válaszokat ad a kérdéseire, valamint támogatja a személyre szabható pénzügyi célok elérésében.

- **Kulcsszavak:**

Cross-platform, React Native, React, Firebase, Stripe, LLM

## Tartalomjegyzék

<i>Feladatkiírás.....</i>	<b>2</b>
<i>Tartalmi összefoglaló.....</i>	<b>3</b>
<i>Motiváció.....</i>	<b>7</b>
<b>1. Területi áttekintés .....</b>	<b>8</b>
1.1. MoneyCoach .....	8
1.2. Money Controll Spending Tracker.....	9
1.3. WalletApp .....	9
1.4. Dime .....	10
1.5. Monefy .....	10
<b>2. Felhasznált technológiák.....</b>	<b>11</b>
2.1. React Native .....	11
2.2. React.....	11
2.3. Typescript .....	12
2.4. JSX és TSX.....	12
2.5. Expo.....	12
2.6. Firebase .....	13
2.7. Verziókövetés .....	13
2.8. Stripe .....	14
2.9. ExpressJs .....	14
<b>3. Funkcionális specifikáció .....</b>	<b>15</b>
3.1. Autentikáció.....	16
3.1.1. Bejelentkezés .....	16
3.1.2. Regisztráció .....	16
3.2. Onboarding oldal.....	17
3.3. Főoldal.....	17
3.3.1. Tranzakciók.....	18
3.3.2. Költségkeret .....	18

3.3.3.	Kihívások és teljesítmény .....	19
3.4.	Ismétlődő tranzakciók.....	20
3.5.	Elemzések .....	21
3.5.1.	Költségvetés elemzés .....	21
3.5.2.	Ismétlődő tranzakciók elemzése .....	21
3.5.3.	Részvény, kriptovaluta és pénzáramlás elemzés .....	22
3.6.	Megtakarítások .....	22
3.6.1.	Pénzügyi célok.....	22
3.6.2.	Kriptovaluták és részvények.....	23
3.7.	Profil .....	24
3.8.	Beállítás .....	24
3.9.	Előfizetés.....	25
3.10.	Chatbot.....	26
4.	<i>Architektúra és adatmodellezés</i> .....	28
4.1.	Architektúra .....	28
4.2.	Adatmodellezés.....	29
4.2.1.	Felhasználó.....	30
4.2.2.	Tranzakciók.....	31
4.2.3.	Kriptovaluták.....	32
4.2.4.	Részvények .....	32
4.2.5.	Számlák.....	32
4.2.6.	Előfizetések .....	33
4.2.7.	Hitelek .....	33
4.2.8.	Pénzügyi célok.....	33
4.2.9.	Költségkeretek.....	34
5.	<i>Technológiai Implementáció bemutatása</i> .....	35
5.1.	Routing és Navigáció.....	35
5.2.	Jogosultságkezelés.....	37
5.3.	Autentikáció.....	37
5.3.1.	Regisztráció .....	38
5.3.2.	Bejelentkezés .....	38
5.3.3.	Elfelejtett jelszó és kijelentkezés.....	38

<b>5.4. Beállítások .....</b>	<b>39</b>
5.4.1. Péznem beállítása .....	39
5.4.2. Nyelv beállítása .....	40
<b>5.5. Fénykép feltöltése .....</b>	<b>40</b>
5.5.1. Kép Kiválasztása .....	40
5.5.2. Kép átméretezése .....	41
5.5.3. Kép feltöltése és a profil frissítése .....	42
<b>5.6. Adatkezelés firestore segítségével .....</b>	<b>42</b>
<b>5.7. Stripe integráció .....</b>	<b>43</b>
5.7.1. Szerver oldal .....	44
5.7.2. Kliens oldal .....	45
<b>5.8. Diagramok, elemzések készítése .....</b>	<b>46</b>
5.8.1. Adatok Szűrése és Feldolgozása .....	46
5.8.2. Diagram komponensek implementálása .....	47
5.8.3. Kripto és Részvényértékek Vizualizációja LineChart segítségével .....	48
<b>5.9. Kihívások implementálása .....</b>	<b>49</b>
<b>6. Nagy Nyelvi Modellek Integrációja .....</b>	<b>50</b>
6.1. Nagy Nyelvi Modellek Áttekintése .....	50
6.2. Prompt Engineering .....	51
6.3. Mesterséges Intelligencia és Chatbotok .....	51
6.4. Implementáció és Alkalmazás .....	52
6.4.1. Adatgyűjtés és Feldolgozás .....	52
6.4.2. Interakciós Logika és Promptok .....	53
<b>7. Tapasztalatok, továbbfejlesztési lehetőségek és összegzés .....</b>	<b>56</b>
7.1. Projekt Tapasztalatok .....	56
7.2. Továbbfejlesztési lehetőségek .....	56
7.3. Összegzés .....	57
<b>Irodalomjegyzék .....</b>	<b>58</b>
<b>Nyilatkozat .....</b>	<b>59</b>

## Motiváció

Az információs technológia gyors fejlődése új lehetőségeket teremt a minden napjai pénzügyi kezelés terén. Az online alkalmazások és platformok segítségével nyomon követhetjük kiadásainkat és befektetéseinket, valamint hatékonyabban tervezhetjük meg pénzügyeinket. Ebben a dinamikus környezetben merült fel az ötlet egy komplex, keresztplatformos pénzügyi kezelő alkalmazás létrehozására, ahol minden pénzügy egyszerre, egy alkalmazáson belül kezelhető.

A keresztplatformos alkalmazás tervezésekor a fő szempontok között szerepelt a felhasználói élmény, a személyre szabhatóság és a játékosítás beépítése. Célom egy olyan pénzügyi kezelő alkalmazás létrehozása volt, amely nemcsak hatékonyan segíti a felhasználókat a költségvetésük nyomon követésében, hanem egyben motivációt is nyújt a pénzügyi célok eléréséhez.

Az alkalmazás tervezésekor inspirációt merítettem a piaci igényekből és az eddigi pénzügyi alkalmazások korlátjainak felismeréséből. A cél az volt, hogy a fontosabb és hasznos eddigi megoldások beépítése mellett olyan újításokat vigyek be, amelyek valódi értéket jelentenek a felhasználók számára.

Egy olyan alkalmazás, amely nemcsak a költségvetéskezelés funkcionalitására összpontosít, hanem játékosítással, ösztönzéssel, illetve mesterséges intelligencia segítségével is motiválja a felhasználókat, egyedivé teheti a terméket. A tervezett multiplatform támogatás biztosítja, hogy a széles körű felhasználók számára elérhető legyen az alkalmazás, függetlenül attól, hogy milyen eszközt vagy platformot használnak.

Célom az is volt, hogy mélyebb betekintést nyerjek a modern keresztplatformos fejlesztési technológiákba, és egy olyan alkalmazást hozzak létre, amely kielégíti a pénzügyi tervezés és tudatosság terén tapasztalható növekvő igényeket. Az alkalmazás különösen hasznos lehet a jelenlegi pénzügyi válság idején, illetve szinte bármikor a diákoknak, pályakezdőknek, akik pénzt szeretnének spórolni és mindenhol egy helyen kezelni.

## 1. Területi áttekintés

Az első fejezetben részletesen bemutatom azokat az alkalmazásokat, amelyek szoros kapcsolatban állnak diplomamunkám témájával, azaz a pénzügyi kezeléssel. Az alkalmazások kiválasztásánál kiemelt szempont volt, hogy legalább alapvető ingyenes funkciókkal rendelkezzenek.

Az áttekintésben szerepel a MoneyCoach, Money Controll Spending Tracker, WalletApp, Dime és Monefy: Money Tracker amelyek a pénzügyek kezelésére fókusznak. A kiválasztott alkalmazások jellemzői és funkciói részletesen elemzésre kerülnek, különös hangsúlyt fektetve azokra, amelyek hasonló funkciókkal rendelkeznek, vagy ahonnan az ötleteket éppen merítettem.

Az összehasonlító elemzés segítségével kiemeltem azokat az elemeket, amelyek alapján az általam tervezett pénzügyi alkalmazás egyedivé és hatékonnyá teszi majd a felhasználói szemszögből.

Funkciók	Money Manager MoneyCoach	Money Control Spending Tracker	WalletApp	Dime	Monefy: Money Tracker	Credo (saját alkalmazásom)
Személyre szabható pénzügyi célok készítése és követése.	✓	□	□	□	□	✓
Fejlett analitikai eszközök a pénzügyek elemzéséhez és jelentések készítéséhez.	✓	✓	✓	✓	□	✓
Pénzügyi kihívások és játékos feladatok motivációs célok eléréséhez.	□	□	□	□	□	✓
Több pénznem támogatása	✓	✓	✓	✓	✓	✓
Költségek rögzítése, kategorizálása és elemzése.	✓	✓	✓	✓	✓	✓
Ismétlődő tranzakciók kezelése	✓	✓	✓	✓	✓	✓
Nagy nyelvi modell használása	□	□	□	□	□	✓
Befektetések, pénznemek, értékpapírok és kriptovaluták kezelése	□	□	✓	□	□	✓
Tervezett költségek és bevételi tételek naptárba való rögzítése.	✓	✓	✓	□	□	□
Lehetőség a felhasználói visszajelzés gyűjtésére.	✓	□	✓	✓	✓	✓
Költségkeretek létrehozása és nyomon követése.	✓	□	✓	✓	□	✓
Felhasználói profil és beállítások testreszabása.	✓	✓	✓	✓	✓	✓

1.1. ábra – Versenytársak elemzése

### 1.1. MoneyCoach

A MoneyCoach<sup>[2]</sup> applikációban is mint a legtöbb hasonló témájú alkalmazásban a leghasznosabb funkciók az előfizetések alatt érhető csak el. Ilyenek például az iCloud szinkronizáció, családdal való költségmegosztás, pénzügyi célok, és icon kusztomizáció számlaemlékeztetők és még sok más hasznos funkció. Az ingyenes verzióban továbbá megtalálható egy egész hasznos funkció is a fotó alapú nyugta beolvasás.

Alapvető funkciókat lehet meríteni belőle, de ennél már sokkal modernebb és hasznosabb alkalmazások is vannak a piacon. Hiányzik néhány funkció amely a pénzügyi tudatosságra ösztönözi a felhasználókat vagy arra, hogy ténylegesen tudjanak pénzt spórolni, mint például az ingyenes funkcióban a költségkeretek. Úgy vélem egy ilyen alkalmazásban a leglényegesebb funkciónak elérhetőnek kellene lenni az ingyenes verzióban is, ahogy az én alkalmazásomban is elérhető. Ötletet adott itt, hogy a diagramok

felett egy dátum választási lehetőség szerepeljen és dinamikusan tudjuk megnézni pl. a költségeken belüli kategóriák eloszlását az adott időszakon belül

Kiemelt funkció, amely szintén csak a prémium verzióban érhető el, a pénzügyi akadémia, amely oktató anyagokat biztosít a pénzügyekkel kapcsolatban. Összességében tehát a MoneyCoach egy megfelelő applikáció, de hátránya hogy az ingyenes verzióban nagyon kevés használható funkció található.

## 1.2. Money Controll Spending Tracker

A MoneyControl<sup>[3]</sup> felhasználói felülete kicsit régimódibb, cserébe az ingyenes verzióban már új lehetőségek, mint az ismétlődő tranzakciók kezelése vagy több pénzügyi fiók egyidejű kezelése is megjelennek, amik szintén ötletet adtak a saját alkalmazásomhoz. A tranzakciókat itt is, mint az általam elemzett összes alkalmazásban is fel lehet venni és nyomon követni. A prémium verzióban a szinkronizáció és csoportos megosztás mellett változatosabb diagramok is elérhetők.

## 1.3. WalletApp

A WalletApp<sup>[4]</sup> véleményem szerint a piacon elérhető egyik leghíresebb és leghasznosabb pénzügykezelő alkalmazás.

A saját funkcióimat leginkább ez az alkalmazás inspirálta, hiszen én személy szerint már használtam a prémium verzióját egy ideig. A legtöbb hasznos funkcionalitás megtalálható benne: költségkezelés, tranzakciókezelés, elemzések, statisztikák, pénzügyi jóslatok egyes helyeken, ismétlődő tranzakciók, költségkeretek, befektetések kezelése, stb. Azonban ebben az alkalmazásban sincs felhasználói ösztönzésre szolgáló funkció, pedig minden adott lenne hozzá. A prémium verzióban ugyanis itt már banki szinkronizálást is ígér az applikáció. Ez a funkció kétségekívül a leghasznosabb az összes vetélytársa közül. Az premium verziótól belül itt megjelennek az értékpapírok, befektetések tárolása is. Fontos megemlíteni, hogy itt sem lehet friss részvény és kriptovaluta adatokat rögzíteni, míg az én alkalmazásomban ez egy elérhető prémium funkció. A banki szinkronizálás biztonsági szempontból kritikus lenne számomra, így ennek megvalósítását elvettem, de mindenképpen az egyik leghasznosabb funkcionalitás egyike lenne.

## 1.4. Dime

A Dime<sup>[5]</sup> alkalmazás nem kínál sok új funkciót; alapvetően egy egyszerű alkalmazásról beszélünk. Előfizetői lehetőség nincs, és egy ember által lett fejlesztve. Az alkalmazás dizájnja viszont meglepően szép és használható elemeket tartalmaz, mint például a navigációs dizájn. Az alkalmazáson belüli tranzakciókeresés újdonságának számít. Ötletmerítés szempontjából az alkalmazásom kinézetét inspirálta, de a piaci vetélytársai sokkal hasznosabb funkciókkal rendelkeznek.

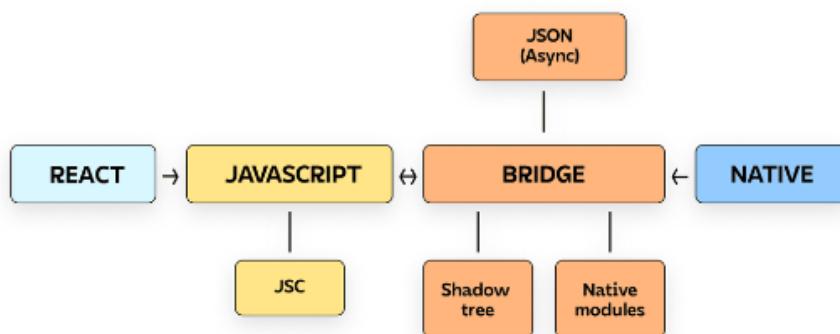
## 1.5. Monefy

A Monefy<sup>[6]</sup> alkalmazás ingyenes verziója sem rendelkezik sok új funkcióval. Alapvetően az alkalmazás a költségek és bevételek felvételére és kategorizálására, kategóriák személyre szabhatóságára törekszik. Megemlítendő egyedi funkció a fájlokba való exportálás vagy felhőtárhelyekkel való szinkronizáció. A fizetős funkció sem kínál nagyon sok újat, többek között több pénznemért vagy több fiók támogatásáért lehet fizetni az alkalmazáson belül

## 2. Felhasznált technológiák

### 2.1. React Native

A React Native<sup>[7]</sup> egy népszerű keretrendszer, amely lehetővé teszi alkalmazások keresztplatformos fejlesztését. A Facebook által kifejlesztett technológia abban különbözik a hagyományos mobilfejlesztési megoldásoktól, hogy egyetlen JavaScript kód bázissal natív alkalmazásokat hozhatunk létre iOS és Android platformokra illetve még webes alkalmazásként is tudjuk buildelni azt. A React Native előnyei közé tartozik a forráskód újra felhasználhatósága, ami jelentős időmegtakarítást eredményezhet a fejlesztés során. Továbbá, a „Hot Reloading” funkció lehetővé teszi a fejlesztési folyamat gyorsítását, mivel a változtatások azonnal láthatóak az alkalmazásban anélkül, hogy újra kellene indítani azt.



2.1. ábra - React Native architektúra<sup>[8]</sup>

### 2.2. React

A React<sup>[9]</sup> egy JavaScript könyvtár, amelyet felhasználói felületek fejlesztésére tervezett a Facebook. Ez a könyvtár segíti a dinamikus és interaktív felületek kialakítását, lehetővé téve a komponens-alapú architektúrát, amely javítja a kód újra felhasználhatóságát és a projekt skálázhatóit. A React virtuális DOM (Document Object Model) használata kulcsfontosságú jellemző, amely optimalizálja az alkalmazás teljesítményét, mivel csak a szükséges elemek frissülnek a felhasználói felületen, nem pedig az egész oldal.

A React Native közvetlenül épít a React alapelveire, lehetővé téve a React komponens alapú architektúrájának és kódjainak felhasználását a mobil alkalmazások fejlesztése során, miközben hozzáférést biztosít a mobil platformok natív funkcióihoz.

## 2.3. Typescript

Typescript<sup>[10]</sup> egy nyílt forráskódú programozási nyelv, amelyet a Microsoft fejlesztett ki, és a JavaScript szupersetjeként funkcionál. Az egyik legnagyobb előnye, hogy statikus típusosságot ad a különben dinamikusan típusos JavaScriptnek. Ez lehetővé teszi a hibák korai felismerését a fejlesztési folyamat során, javítva ezzel a kód minőségét és karbantarthatóságát. A Typescript széles körű támogatása és eszközei, mint például az automatikus kódkiigazítás és a kód refaktorálás, tovább növelik a fejlesztők produktivitását. A Typescript-tel írt kódok bizonyítottan jobban karbantarthatók és nagy kódbázison alapuló, piacon lévő alkalmazásokhoz is előnyben részestik a Javascripttel szemben, ahogy én is tettem.

## 2.4. JSX és TSX

JSX és TSX kiterjesztések a JavaScript és Typescript nyelvekhez, amelyek lehetővé teszik a UI komponensek leírását, mintha HTML kódot írnánk közvetlenül a JavaScript-be. A JSX a React komponensek deklaratív szintaxisát biztosítja, amely segít átláthatóbbá és olvashatóbbá tenni a felhasználói felületek kódját. A TSX a JSX Typescript változata, amely hozzáadja a típusellenőrzés minden előnyét a JSX kifejezésekhez, így még robosztusabbá téve a fejlesztési folyamatot. Ezek a technológiák híd szerepet töltenek be a tervezés és a programozás között, megkönnyítve a komplex felhasználói felületek fejlesztését.

## 2.5. Expo

Expo<sup>[11]</sup> egy nyílt forráskódú platform, amely megkönnyíti a React Native alkalmazások fejlesztését, tesztelését és kiadását. Az Expo lehetővé teszi, hogy gyorsan elindíthassuk az alkalmazásunkat és tesztelhessük azt valós időben mobil eszközökön, anélkül, hogy bármilyen natív kódot manuálisan konfigurálniuk kellene. Az Expo számos beépített API-t és szolgáltatást kínál, amelyekkel a fejlesztők könnyedén hozzáadhatnak funkciókat az alkalmazásainkhoz, mint például helymeghatározás, videólejátszás, vagy értesítések kezelése. A fejlesztésem során elképesztően nagy hasznát vettettem az Expo Go applikációnak, melynek segítségével fizikális eszközön is pár másodperc alatt tudjuk futtatni az alkalmazásunkat. Ez fontos, hiszen például az IOS-es emulátort csak Mac OS-en tudunk futtatni, Windowson hivatalosan nem.

## 2.6. Firebase

Firebase<sup>[12]</sup> egy átfogó fejlesztési platform a Google-től, amely számos szolgáltatást és eszközt kínál a webes és mobilalkalmazás-fejlesztők számára. A platform segíti a gyors fejlesztést számos beépített funkcióval, mint például adatkezelés, felhasználokezelés, és analitika, így a fejlesztők több időt fordíthatnak az alkalmazások innovatív aspektusaira a backend műveletek helyett.

A projektben a Firebase Authentication szolgáltatást használtam az autentikáció kezelésére. Fontos megjegyezni, hogy itt nem az SDK-t<sup>[13]</sup>, hanem a REST API-t alkalmaztam a felhasználói hitelesítési folyamatok kezelésére. A firebase autentikációja lehetővé teszi a biztonságos autentikációs folyamatot. A REST API használatát azért tekintettem előnyösebbnek, mert így egy újra felhasználható autentikációs logikát építettem fel, amit bármikor át tudok írni egy másik backend környezettel való integráció esetén.

Firestore<sup>[14]</sup> egy NoSQL dokumentum alapú adatbázis, amely lehetővé teszi a strukturált adatok gyors és rugalmas tárolását és lekérdezését. A projektben a Firestore-t használtam a felhasználói adatok tárolására. A Firestore előnyei közé tartozik a valós idejű adatfrissítés, skálázhatóság, és a könnyű hozzáférés szabályozás. Offline támogatással is rendelkezik, így az adatok bizonyos része akkor is elérhető a felhasználó számára, ha nem rendelkezik internetkapcsolattal.

A Firebase Storage szolgáltatást arra használtam, hogy nagy méretű fájlokat, mint például képeket tároljak biztonságosan, így a felhasználók feltölthetik saját profilképüket az alkalmazásba. Ez a szolgáltatás integrálódik a Firebase többi részével, így a fejlesztők könnyen kezelhetik az adatokat és a médiafájlokat egyetlen, központosított helyről, amely támogatja a hatékony adatáramlást és hozzáférést.

## 2.7. Verziókövetés

A verziókövetés fontos része minden szoftverfejlesztési projektnek. Ebben a projektben a Git verziókezelő rendszert használtam, amely lehetővé teszi a kódváltozások nyomon követését, a munka verzióinak menedzselését és ha csapatban dolgozunk egy projekten, akkor a kollaborációt. A projekt kódbázisát<sup>[15]</sup> a GitHub-ra töltöttem fel, ami egy online platform a gites projektek (repository-k) tárolására, kezelésére és kollaborálására.

## 2.8. Stripe

Stripe<sup>[16]</sup> egy átfogó fizetési infrastruktúra, amelyet fejlesztők használhatnak az online fizetési tranzakciók kezelésére. A Stripe-ot a projektben az egyszeri előfizetések kezelésére használtam, bár érdemes megjegyezni, hogy a mobilos alkalmazásokban való használata korlátozott, mivel a legtöbb mobil áruház, mint az Apple App Store vagy Google Play, saját in-app előfizetési rendszereket igényel az előfizetések kezeléséhez, így ha ki szeretném adni az alkalmazásomat, akkor azokra kell váltanom majd.

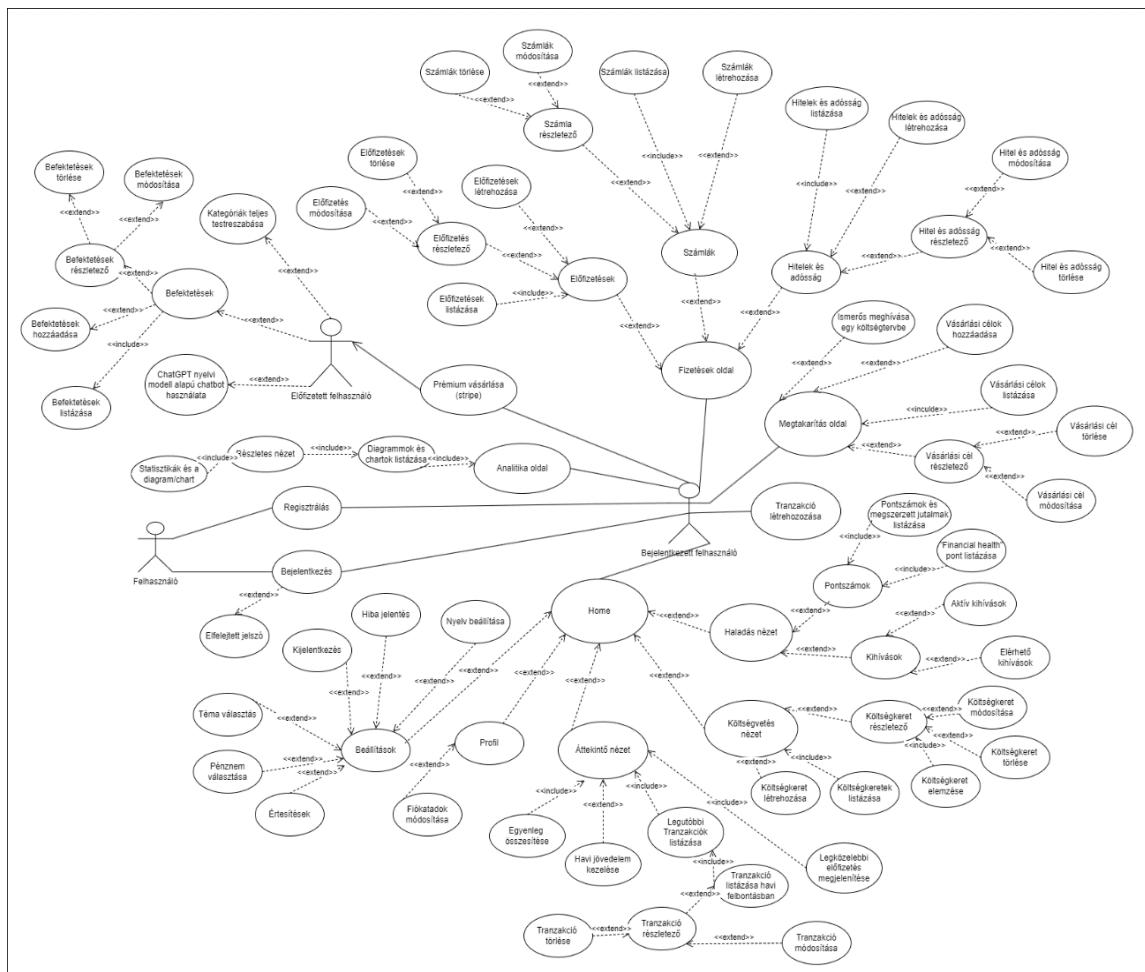
## 2.9. ExpressJs

Az ExpressJs egy rugalmas Node.js webalkalmazás keretrendszer, amelyet a projektben a Stripe integrációhoz használtam fel, egy mikroszerver létrehozására, amely biztonságosan kezeli a Stripe fizetési tranzakcióit.

### **3.      Funkcionális specifikáció**

Ebben a fejezetben bemutatom az alkalmazásom alapvető funkcióit, amelyek meghatározzák, hogy a felhasználók hogyan interaktálhatnak a rendszerrel és milyen műveleteket hajthatnak végre. Az itt leírt funkciók összessége adja az alkalmazás magját, amely a felhasználók számára biztosítja a szükséges eszközöket pénzügyei kezeléséhez. A funkcionális specifikáció célja, hogy áttekintést nyújtson az alkalmazás kínálta lehetőségekről és azoknak a felhasználói igényekre gyakorolt hatásairól. Ez a fejezet tehát azokat a főbb jellemzőket és tevékenységeket ismerteti, amelyeket a felhasználók elérhetnek az alkalmazás használata során.

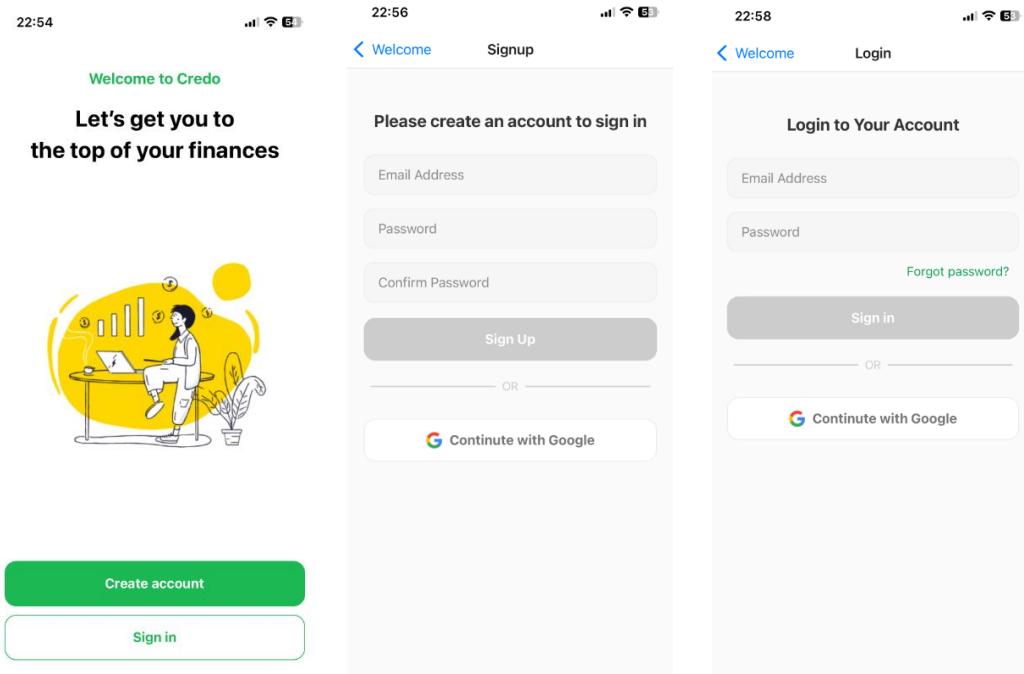
Az alkalmazás által nyújtott funkciókat részletesen, de lényegre törően tárgyalom, beleérve minden egyes funkció célját, a felhasználó által elérhető tevékenységeket, és azt, hogy ezek miként segítik elő a hatékonyabb pénzügyi döntéshozatalt. Ezáltal a fejezet célja, hogy világos képet adjon arról, hogyan is működik az alkalmazás felhasználói szemszögből.



### 3.1 ábra - Az alkalmazás Use-Case diagramja

### 3.1. Autentikáció

#### 3.1.1. Bejelentkezés



3.2. ábra - Az alkalmazás autentikációs oldalai

A bejelentkezési folyamat során a felhasználóknak lehetőségeük van az e-mail címük és jelszavuk megadására. Az alkalmazás csak akkor enged be, ha a megadott adatok helyesek és egyeznek az adatbázisban tárolt információkkal. A biztonság további növelése érdekében elérhető az új jelszó beállítása funkció, amely segít a felhasználóknak visszaállítani elfelejtett jelszavukat. Ekkor a felhasználó az emailcímére kap egy email-t amiben szerepel egy új jelszó beállítása link, ahol új jelszót tud beállítani. A bejelentkezési gomb kattinthatóvá és zöldé változik, ha a felhasználó kitölötté a mezőt.

#### 3.1.2. Regisztráció

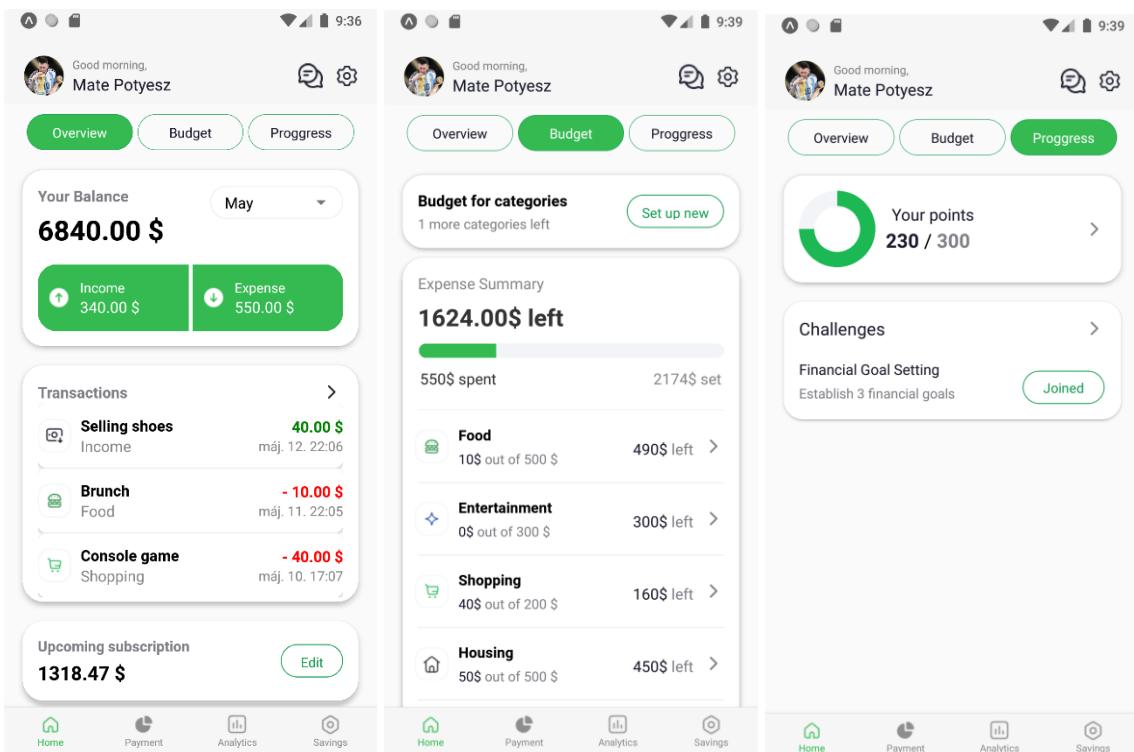
A regisztráció során a felhasználóknak biztosított egy egyszerű és átlátható felület, ahol megadhatják e-mail címüket és kétszeri jelszavukat. Az alkalmazás csak megfelelő hosszúságú és egyező jelszó esetén engedi a regisztrációt. Továbbá csak helyes formátumú email cím megadása szükséges a regisztráláshoz. A felhasználói élmény érdekében a regisztráció után a felhasználó azonnal be is lép az alkalmazásba, nem kell újra bejelentkezni.

### 3.2. Onboarding oldal

A regisztráció után az új felhasználó egy onboarding, azaz regisztráció utáni első lépések oldalra kerül. Itt a felhasználónak be kell írni az általános információit, hogy tudja használni az alkalmazást. Ilyen a neve, neme, születési dátuma, jelenlegi egyenlege és az átlagos havi jövedelme. Ezek után, ha helyesek a beírt adatok a felhasználó tudja használni az alkalmazást és a főoldalra kerül,

### 3.3. Főoldal

A főoldalon belül 3 nézetünk van. Az Áttekintés (Overview) nézetben a felhasználó megtekintheti az aktuálisan elérhető pénzösszegét, költségeit és jövedelmét egy rövid nézetben az aktuális hónapban. Alatta található az utolsó 3 tranzakció listázása



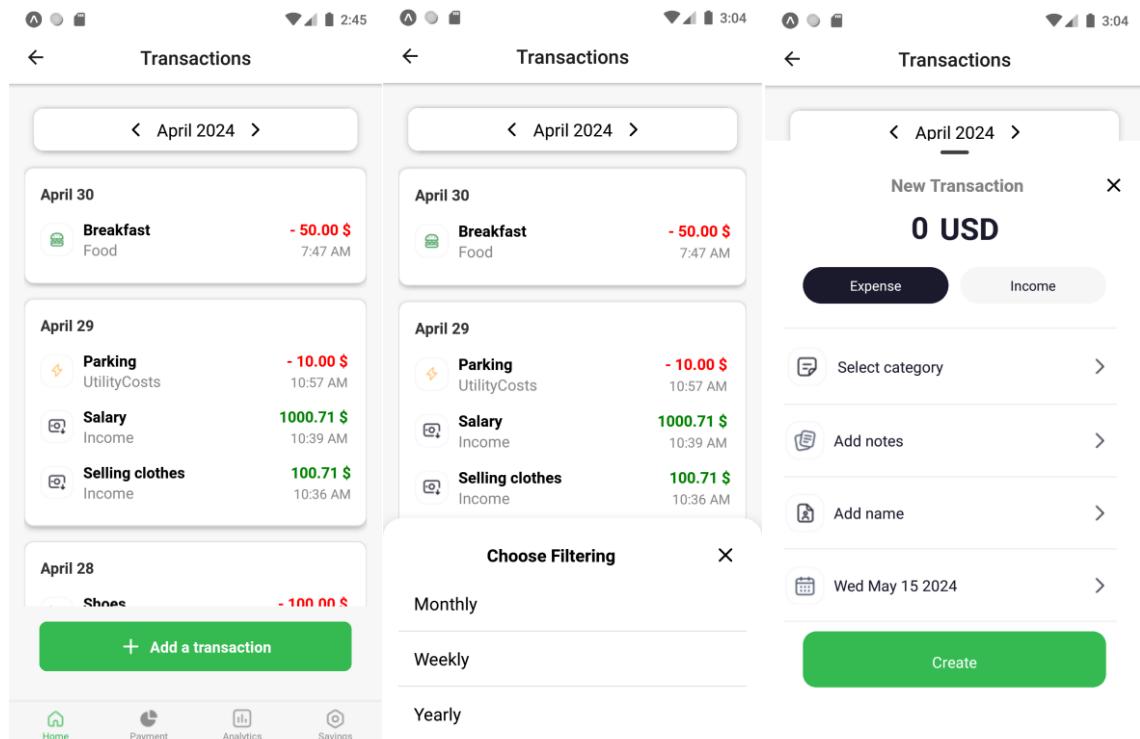
3.3. ábra - Az alkalmazás főoldala

A Költségkeret (Budget) nézetben található a költségkeretek listázása és kezelése. Egy költség kategóriára fel vehetünk egy pénzügyi keretet amibe be szeretnénk férfi az adott hónapban. Értelemszerűen minden kategóriára csak egy ilyen keretet lehet felvenni. Ezen az oldalon hónapot is lehet választani így megnézhetjük, hogy az előző hónapban hogyan fértünk bele egy aktuális kategória keretébe.

Az Előrehaladás (Progress) nézetben belül láthatjuk az aktuális pontjainkat amik az alkalmazás használata során és a kihívások során növekednek. Ez alatt található az aktuális kihívások amire ha rákattintunk elnavigálunk a kihívások oldalra.

### 3.3.1. Tranzakciók

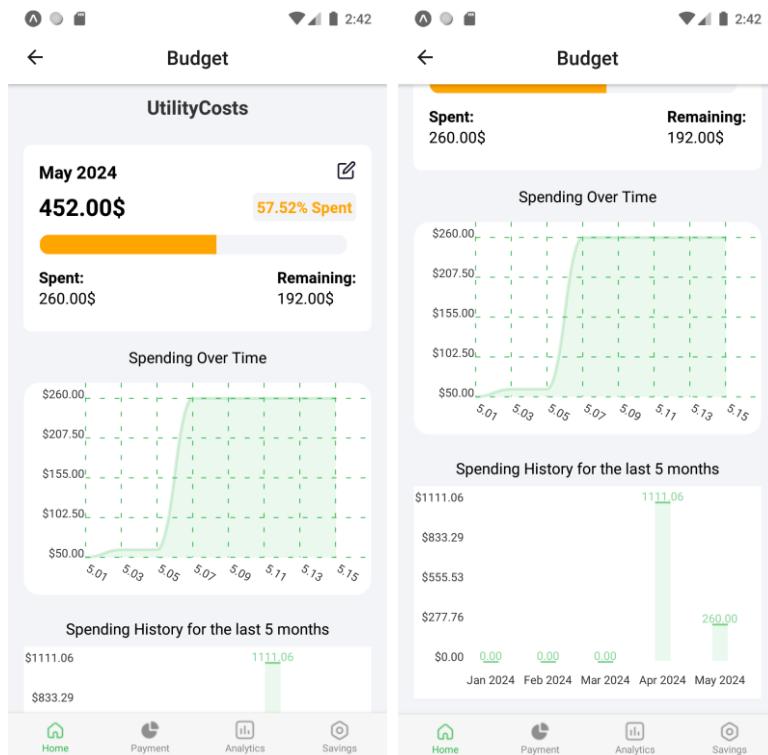
A tranzakciók oldalon a felhasználó tranzakciói jelennek meg kategorizálva, havi, heti és éves lebontásban. Ezen az oldalon tudja a felhasználó egy új tranzakciót felvenni, illetve módosítani vagy törölni. Egy tranzakcióhoz kötelező megadni a hozzá tartozó kategóriát, nevet és összeget, valamint a dátumot, de leírást is adhatunk opcionálisan hozzá.



3.4. ábra - Tranzakciók oldal

### 3.3.2. Költségkeret

A felhasználó az aktuális költségkeretére lépve részletezve megnézheti annak alakulását a kijelölt hónap szerint. Értesülhet arról, hogy mennyit költhet a kategóriából még, illetve havi lebontásban nézheti meg a költségei alakulását azt akutálisan kiválasztott kategóriákban, továbbá az utolsó 5 hónap költségeit is meg tudja nézni szintén egy diagram formájában.

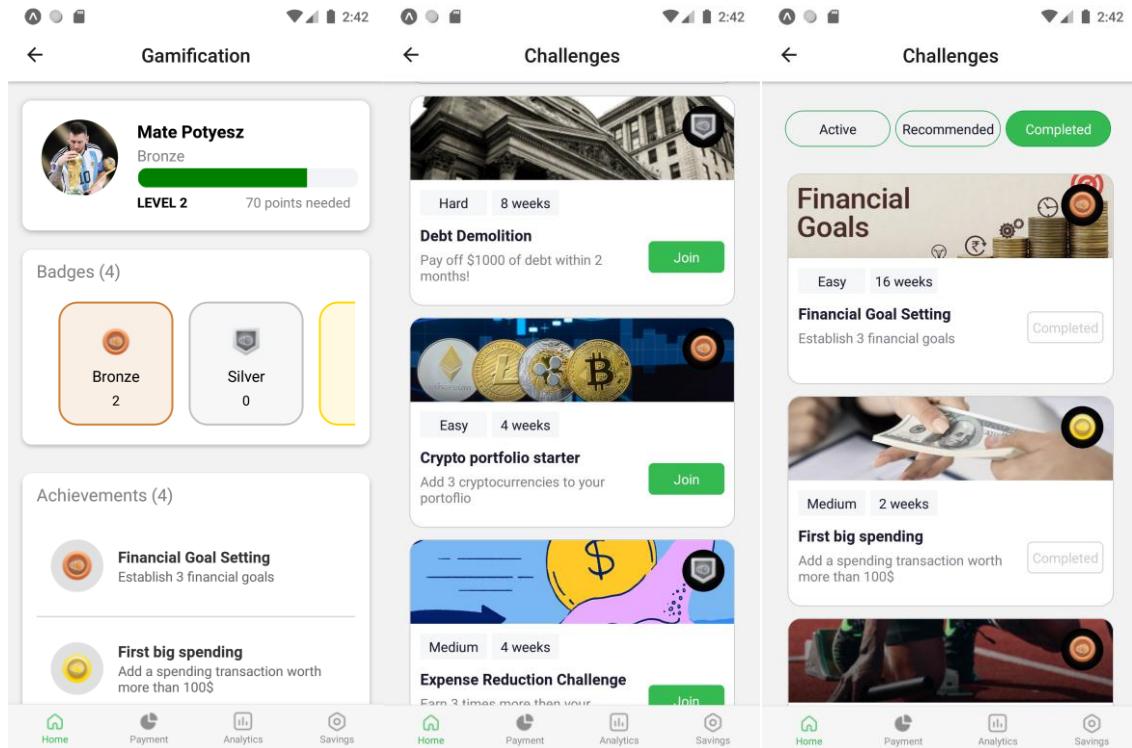


3.5. ábra - Költségkeret oldal

### 3.3.3. Kihívások és teljesítmény

A kihívások oldalon megnézhetjük, hogyan állunk az aktuálisan felvett kihívásokkal. Az ajánlott nézetben válogathatunk az aktuálisan elérhető kihívások közül. Egy kihívás akkor érhető el, ha az ahhoz szükséges szinttel rendelkezik a felhasználó. A teljesített nézetben pedig az összes teljesített kihívást láthatjuk. A kihívásra kattintva csatlakozhatunk ahhoz és megnézhetjük a kihívás részleteit. A teljesítmény nézetben a felhasználó pedig megnézheti az aktuális szintjét, tapasztalati pontját, medáljait és teljesített kihívásait.

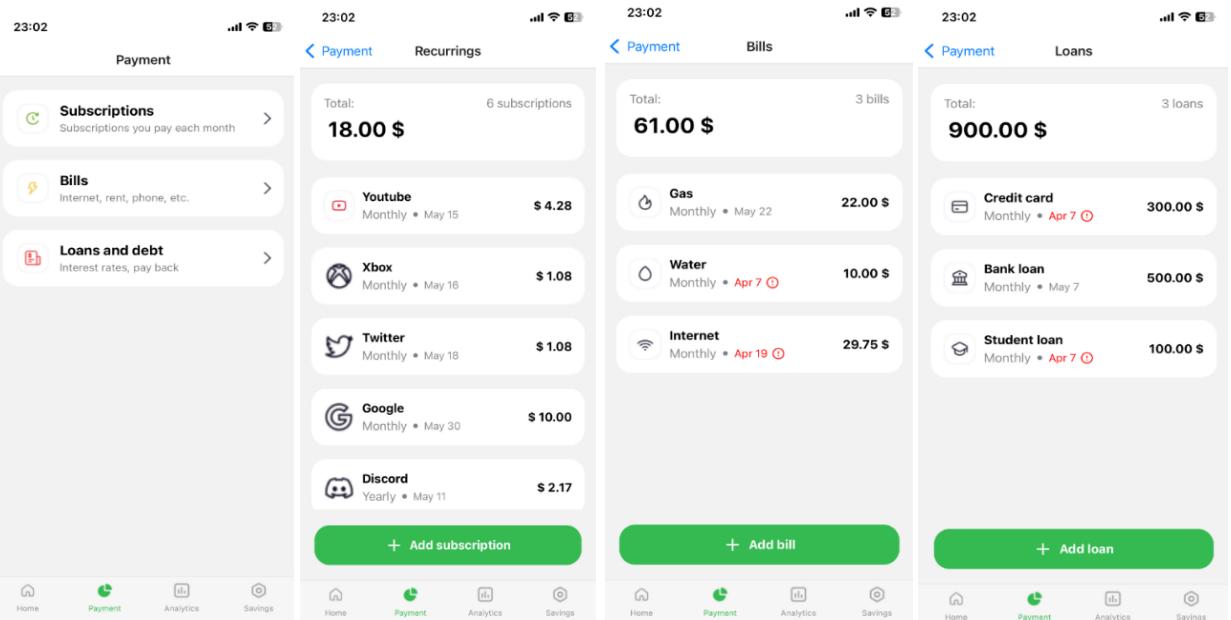
## Pénzügyi költségvetés kezelő keresztpalatmos alkalmazás fejlesztése React Native-ben



3.6. ábra - Teljesítmény és küldetések oldalak

### 3.4. Ismétlődő tranzakciók

Az alkalmazásban lehetőségünk van szinte az összes ismétlődő tranzakciót nyomon követni. Több mint 10 féle előfizetés, 5 féle számla és 6 féle hitel közül lehet választani melyhez egyedi ikon tartozik. Persze egyéb kategóriát is meg lehet adni, ehhez viszont nem tartozik ikon.



3.7. ábra - Ismétlődő tranzakció oldalak

Ezeken az oldalakon tehát az előfizetést, számláinkat valamint hiteleinket, tartozásukat tudjuk nyomon követni. Csakúgy mint a költségnél és költségkeret nél is egy felugró ablakban tudunk hozzáadni, módosítani illetve törlni egy-egy ismétlődő fizetést kategória, érték dátum illetve fontosság szerint.

### 3.5. Elemzések

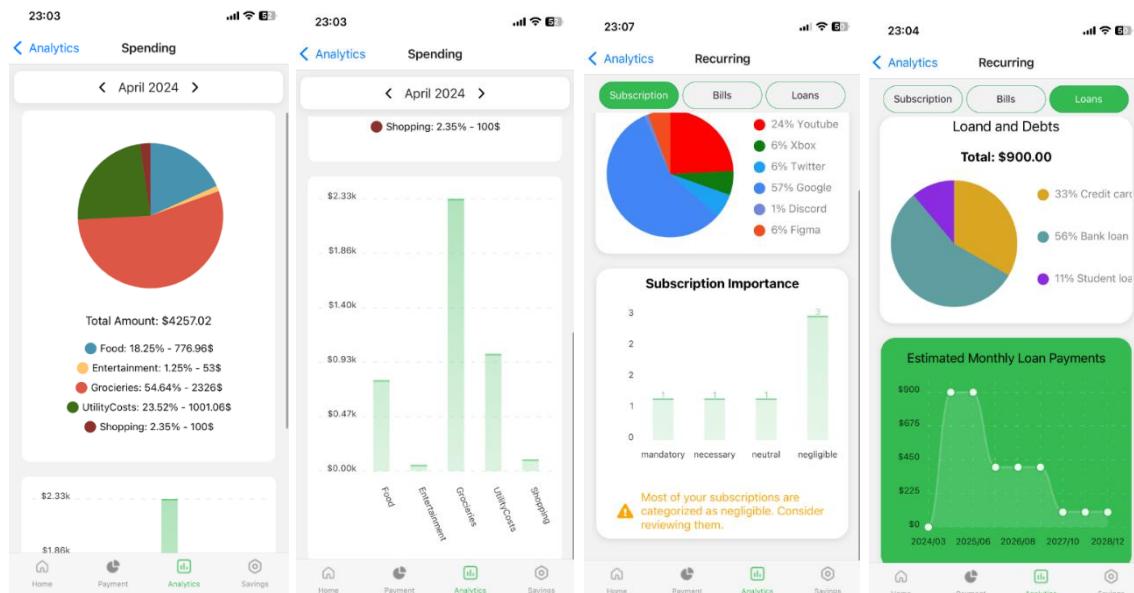
#### 3.5.1. Költségvetés elemzés

Ezen az elemzési oldalon a felhasználó a költségeinek elemzését tudja megtekinteni. Lehet szűrni dátum alapján (havi, heti, éves lebontásban) és lépkedni közöttük. Ezen az oldalon egy egyszerű és biztos képet kaphat arról a felhasználó, hogyan és milyen lebontásban alakultak a költségei a kiválasztott időszakban, kategóriákra bontva.

#### 3.5.2. Ismétlődő tranzakciók elemzése

Az ismétlődő tranzakciók elemzési oldalon a felhasználó a 3 ismétlődő költségtípusról találhat elemzéseket. Az első oldalon a felíratkozások szerepelnek összeg lebontásban, illetve egy oszlopdiagram is kimutatja, milyen fontosságú feliratkozásai vannak, illetve értesül arról, ha a nélkülözhetők vannak túlsúlyban.

A számlák és a hitelek oldalon is hasonló elemzéseket láthat a felhasználó annyi különbséggel, hogy a hiteleknél a jövőbe tud tekinteni és meg tudja nézni, hogy mennyi lesz a havi költsége a hiteleit tekintve az évek/hónapok során ha rendesen fizeti azokat.

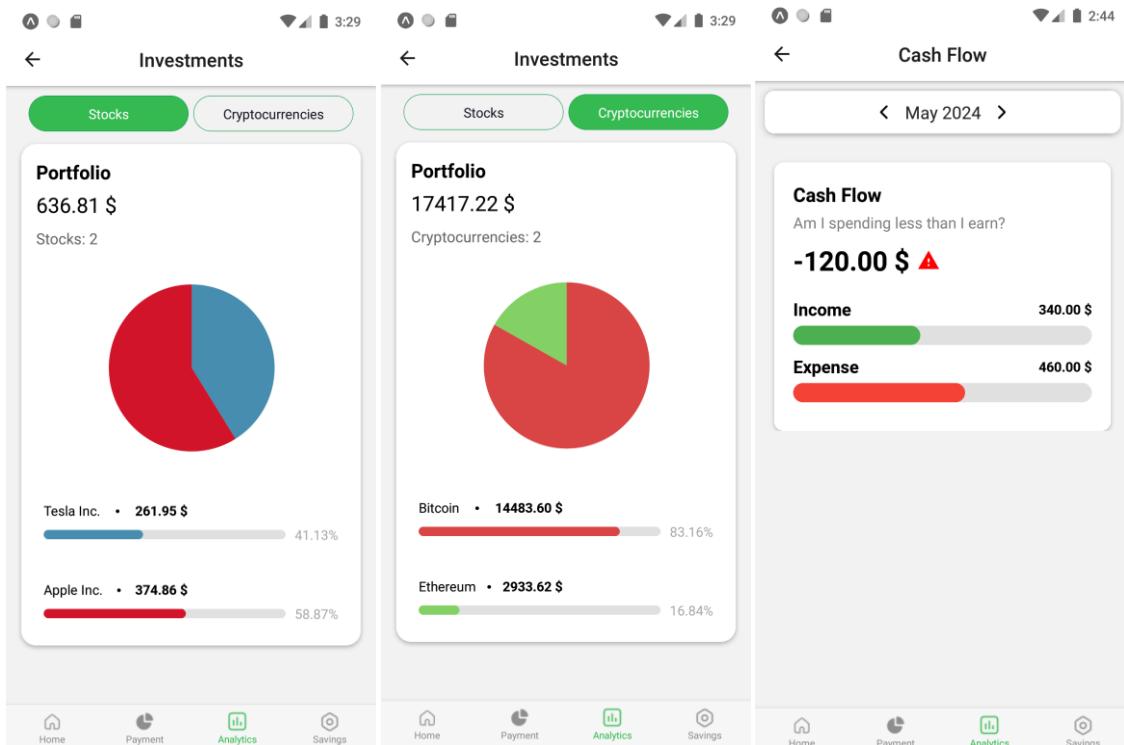


3.8. ábra - Költségek és ismétlődő tranzakciók elemzési oldalak

### 3.5.3. Részvény, kriptovaluta és pénzáramlás elemzés

A részvények elemzési oldalon a felhasználó nyomon tudja követni az aktuális portfóliójának eloszlását mind részvények, mind pedig a kriptovaluták terén százalékos lebontásban és portfólióösszegzést nézve.

Az utolsó elemzési oldalon a felhasználó időintervallumokra szűrve tudja megnézni, hogyan alakult a pénzáramlása a kiválasztott intervallumban.

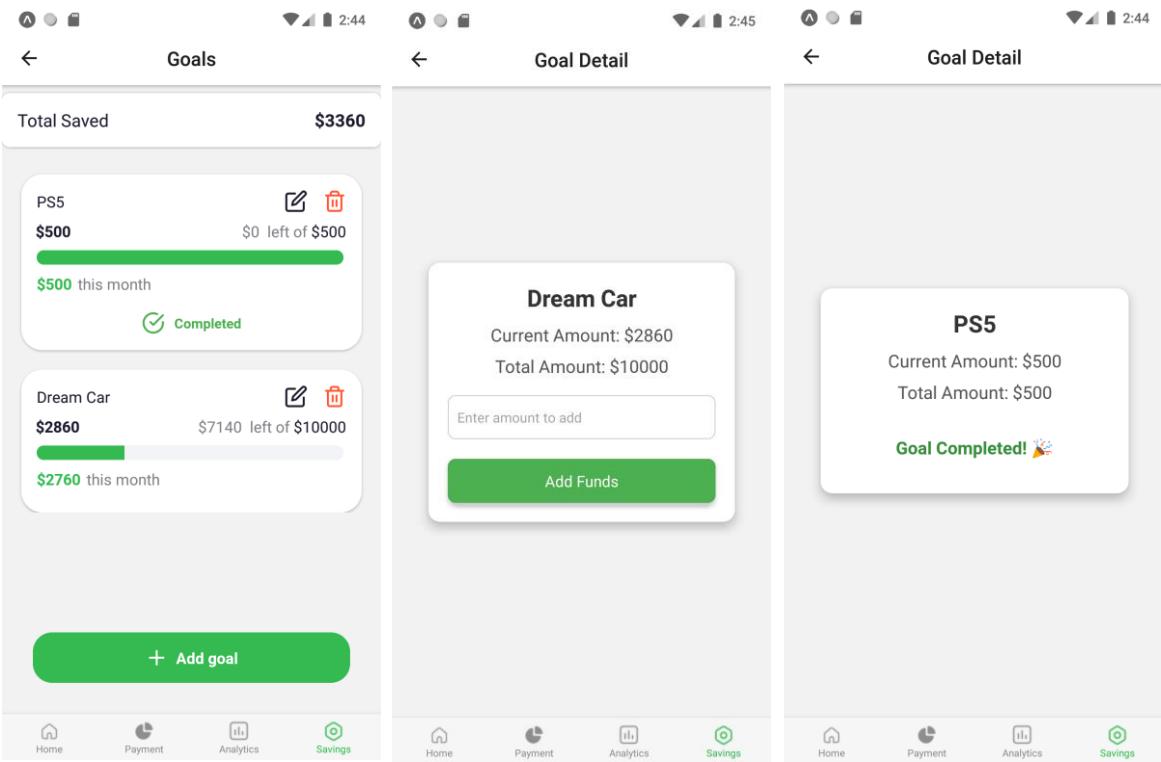


3.9. ábra - Részvény, kriptovaluta és pénzáramlás elemzési oldalak

## 3.6. Megtakarítások

### 3.6.1. Pénzügyi célok

A pénzügyi célok oldalon a felhasználó hozzá tudja adni és nyomon tudja követni az aktuálisan megvásárolni kívánt termékeit név, szükséges összeg, és várható dátum szerint. Az aktuális célra kattintva pedig meg tudja azt részleteiben is nézni, illetve hozzá tud adni egy kívánt megspórolt összeget az elérhető keretéből.

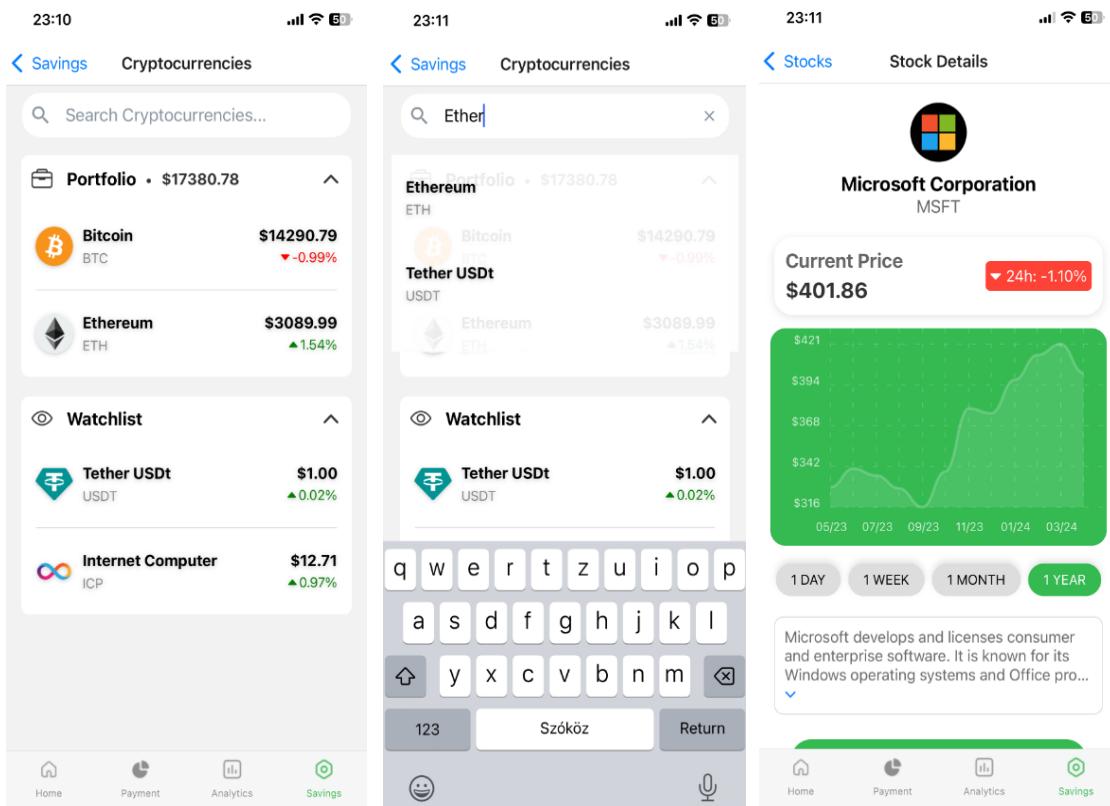


3.10. ábra - Pénzügyi célok

### 3.6.2. Kriptovaluták és részvények

A kriptovaluták oldalon a felhasználó a legfrissebb adatok alapján tud böngészni több ezer kriptovaluta között. A kiválasztott kriptovalutára kattintva megtekintheti annak aktuális áringadozását, jelenlegi piaci árát, illetve hozzá tudja adni azt a portfóliójához, el tud adni belőle, illetve a megfigyeltek közé tudja azt rakni.

A részvények nézet hasonló funkciókkal működik, ezzel támogatva a felhasználói élményt és egyszerűséget. A felhasználó itt szintén megtekintheti és a portfoliójába veheti a több ezer elérhető részvények valamelyikét, így nem szükséges egy részvény applikációba mennie, hanem elég csak ide felvennie a portfólióját és egy helyen nyomon tudja követni az aktuális, legfrissebb értékkel számított portfóliót.



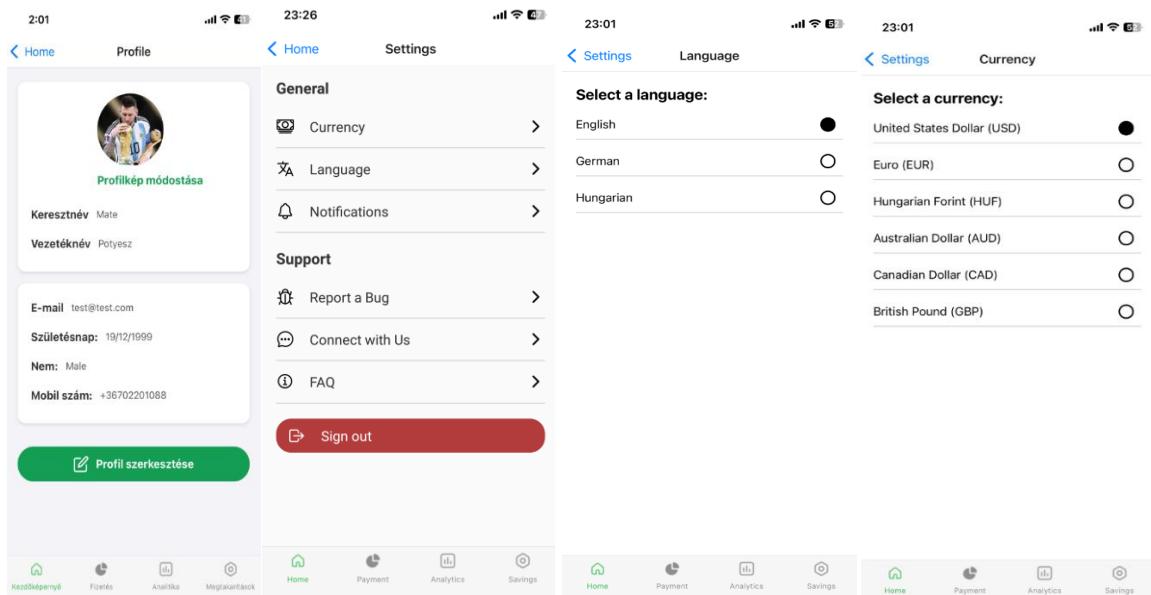
3.11. ábra - Kriptoállatuk oldal és egy részvény-részletező oldal

### 3.7. Profil

A profil oldalon elérhető a felhasználó adatai, illetve módosítani is tudjuk azokat. A bejelentkezett felhasználó továbbá profilképet is tölthet fel magáról. A módosítható adatok között szerepel a telefonszám, születési dátum, név és nem.

### 3.8. Beállítás

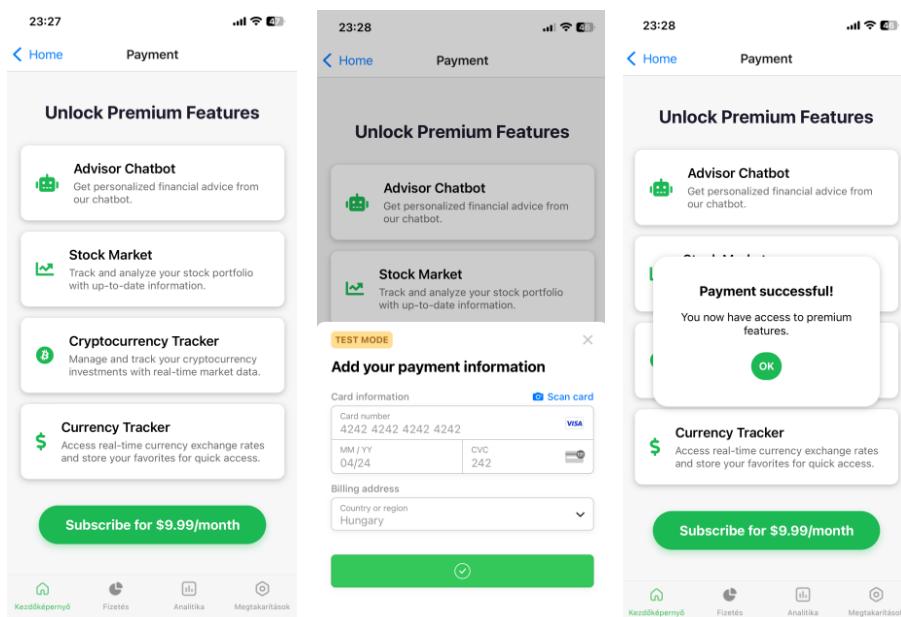
Egy fontos oldal még a beállítások oldal. Itt a felhasználó tud választani pénznemek között és hivatalos, akutálisan elérhető, legfrissebb váltószámok alapján jelennek meg a pénzügyi értékei az alkalmazásban, ha pénznemet váltott. Elérhető továbbá 3 nyelv is az alkalmazásban, a német, az angol illetve a magyar. A nyelv alatt ki lehet választani miről szeretnénk értesítéseket kapni és miről nem. A felhasználó visszajelzéseket is küldhet az alkalmazásról hibajelentés formájában, illetve elolvashatja a gyakran kérdezett kérdéseket is, és az alkalmazás elérhetőségeit is elérheti, mint például a közösségi platformok vagy emailcím. Végül ezen a felületen tudunk kijelentkezni a profilunkból.



3.12. ábra – Beállítások és profil oldalak

### 3.9. Előfizetés

Az előfizetés kezelése egy fontos funkciója az alkalmazásnak. A felhasználó ugyanis csak akkor éri el a mesterséges intelligenciára épülő chatbotot vagy a kriptovaluta és részvénypiacot, ha egy egyszeri előfizetést visz véghez. Fizetés után prémium felhasználóvá tudunk válni. Előtte az aktuális funkciókra lépve egy fizetési oldalra kerülünk, illetve a főoldalon is van egy kártya, amire kattintva ide navigálhatunk.



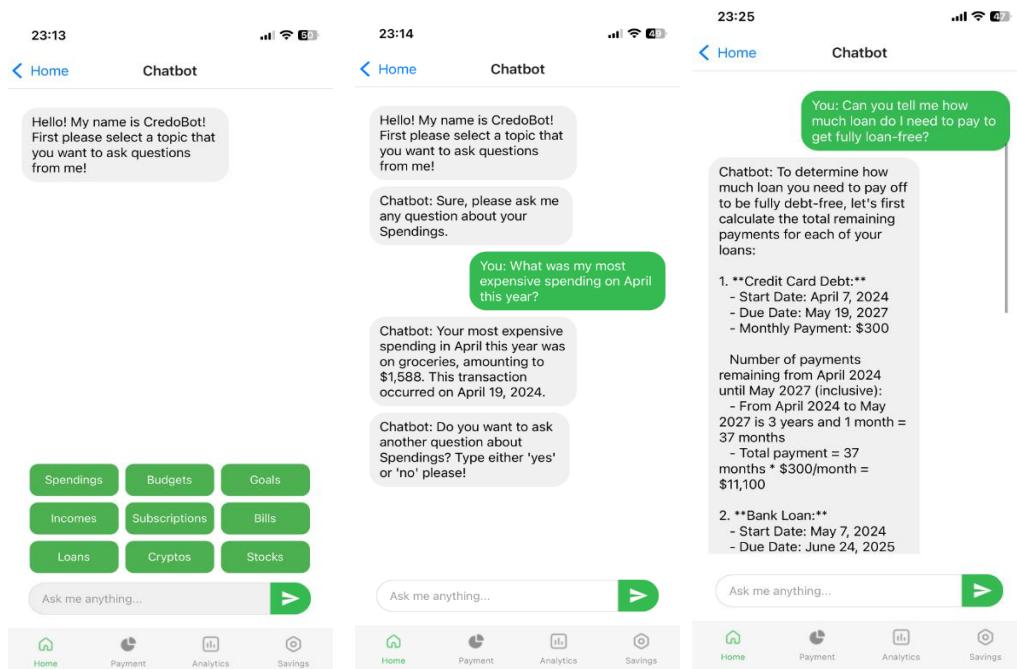
3.13. ábra - Fizetési oldal

Fizetni helyes bankkártyaadatokkal és fedezettel lehet, pont úgy, mint minden más fizetéssel rendelkező alkalmazásban. Sikeres fizetés után már elérhetőek lesznek a prémium funkciók is.

### 3.10. Chatbot

Az alkalmazás egyik leghasznosabb funkciója a költségvetési kisegítő, a mesterséges intelligenciára épülő chatbot. Az előfizetett felhasználó 9 kategóriában is tudja kérdezni a chatbotot, ami az alkalmazásban lévő adatok alapján képes helyesen válaszolni a kérdéseire. Például kérhetünk tőle elemzéseket a költségeinkkel kapcsolatban, megkérdezhetjük mi volt a legdrágább költségünk egy aktuális időszakban, miből kell kevesebbet költeni, stb. De akár segít a pénzügyi céljainkat is elérni. Tudja, hogy mennyit érdemes félrerakni havonta, hogy időben elérjük a kívánt céltunkat és meg tudjunk eleget takarítani.

Kérdezhetjük a feliratkozásainkkal kapcsolatosan is ahol az aktuális feliratkozások fontosságát is figyelembe veszi és ajánlani tud, mit érdemes nélkülözni, min érdemes spórolni. Hiteleinkben választ kaphatunk, hogy körülbelül mennyi idő múlva tudjuk kifizetni őket, ha betartjuk a havi törlesztőrészleteket, vagy azt is, hogy összesen mennyi pénzt kell még visszafizetnünk, hogy teljesen hitelmentesek legyünk. Informálódhatunk továbbá a kriptovaluta és részvény portfóliónkról is, ezáltal egy személyre szabott szakértői elemzést és tanácsadást is kérhetünk. Továbbá a bevételeinkről is kérdezhetjük, ahol pedig elemez a név, dátum és összeg alapján, hogy milyen bevételeink voltak, miket érdemes a jövőben is csinálni, illetve a bevételeink összegzését is kérhetjük tőle dátumra bontva, vagy éppen a legnagyobb bevételünket is meg tudja válaszolni a chatbot, ezáltal hiteles képet kapva arról, milyen bevételeket igyekezzünk folytatni



3.14. ábra - Chatbot

A chatbot egy hasznos funkció, mely a piacon elérhető legfrissebb, legnagyobb, legfejlettebb nyelvi modellel dolgozik, így ha elegendő adatunk van, akkor hiteles választ kaphatunk minden pénzügyi kérdéseinkre. A nagy nyelvi modell amit használ az alkalmazás, az OpenAI<sup>[17]</sup> legújabb nyelvi modellje, a GPT-4o. Viszont mint mindenhol itt is előfordulhatnak olyan esetek, hogy a mesterséges intelligencia rosszul válaszol. Ilyenkor megpróbálhatunk újból kérdezni tőle ugyan abban a témaban reménykedve a helyes válaszban.

## 4. Architektúra és adatmodellezés

### 4.1. Architektúra

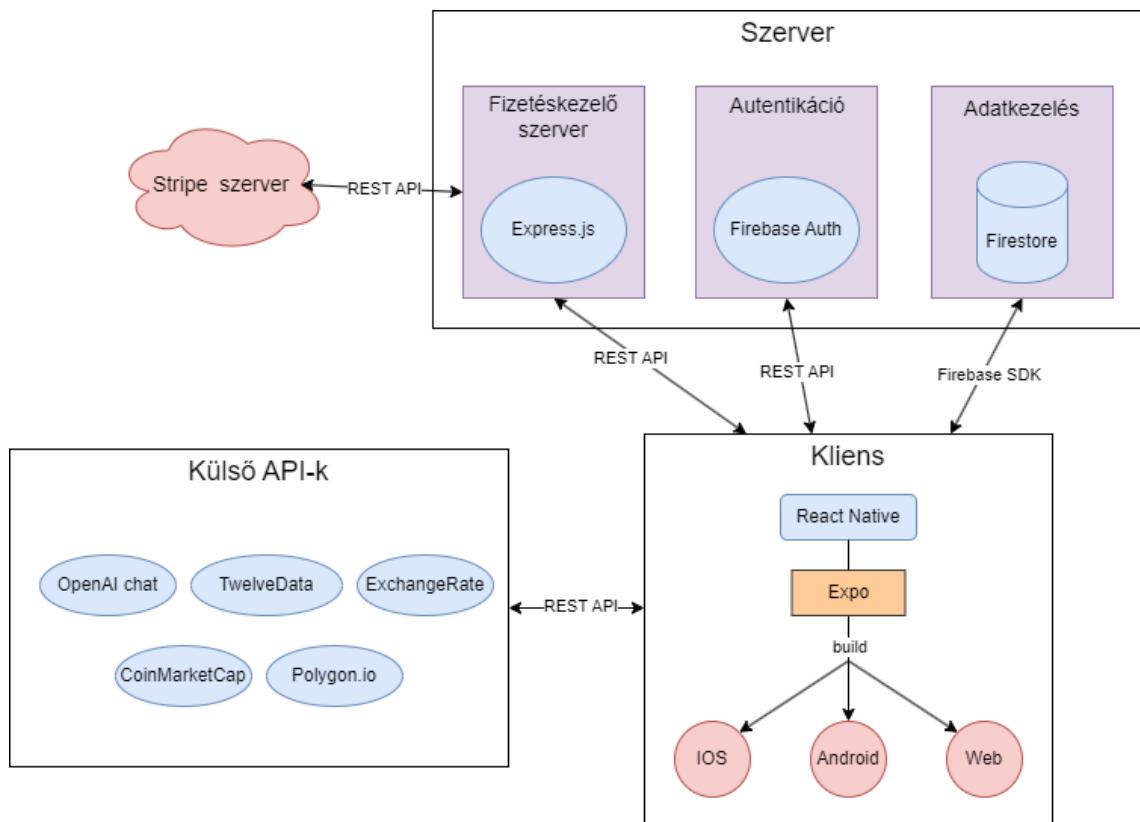
Az alkalmazás architektúrája két fő komponensre bontható: kliensoldal és szerveroldal. A kliensoldal, amelyet a felhasználók közvetlenül használnak, React Native technológiával készült, kihasználva a React komponens-alapú fejlesztési modelljét. Ez lehetővé teszi a felhasználói felület dinamikus és reszponzív kezelését, ahol a komponensek állapota (state) és életciklusa (lifecycle) szorosan kezelt és optimalizált a felhasználói interakciók gyors és hatékony kezelése érdekében. A kliensoldal integrálja a modern felhasználói élményt elősegítő technológiákat, mint például a Context API-t a globális állapotkezelésre, vagy például az Axios-t a külsős API hívások kezelésére.

A szerveroldali komponenseket a Google Firebase biztosítja, amely több szolgáltatást is magában foglal, mint például a Firestore adatbázist a perzisztens adattárolásért és a Firebase Authentication szolgáltatást a felhasználók hitelesítéséért. Az autentikációs műveletek REST API-n. Az adatkezelés a Firestore adatbázissal a Firebase SDK-n keresztül történik.

Az alkalmazás architektúrája továbbá magában foglal több külső API integrációját is, amelyek kritikusak a pénzügyi adatok kezeléséhez. Például, a CoinMarketCap<sup>[18]</sup> API a kriptovaluták aktuális árfolyamainak lekérésére, az ExchangeRate-API<sup>[19]</sup> a pénznemek konverziójához, a Polygon.io<sup>[20]</sup> a részvényekhez, vagy a TwelveData<sup>[21]</sup> a pénzügyi diagramokhoz. Továbbá az openAI chat api-ját használtam a chatbotos integrációhoz. Ezek mind közvetlenül a kliensoldalról vannak integrálva főleg Axios technológia segítségével. A Stripe fizetési műveletek kezelésére egy külön Express.js alapú mikroszervert hoztam létre, amely a frontenddel és a Stripe szolgáltatással közvetlen kapcsolatban áll, így biztosítva a tranzakciók biztonságos kezelését.

A projekt forráskódjának verziókezelése Git segítségével történt, és a GitHub-on tároltam, amely lehetővé tette a fejlesztési folyamat hatékony nyomon követését és a változatok kezelését.

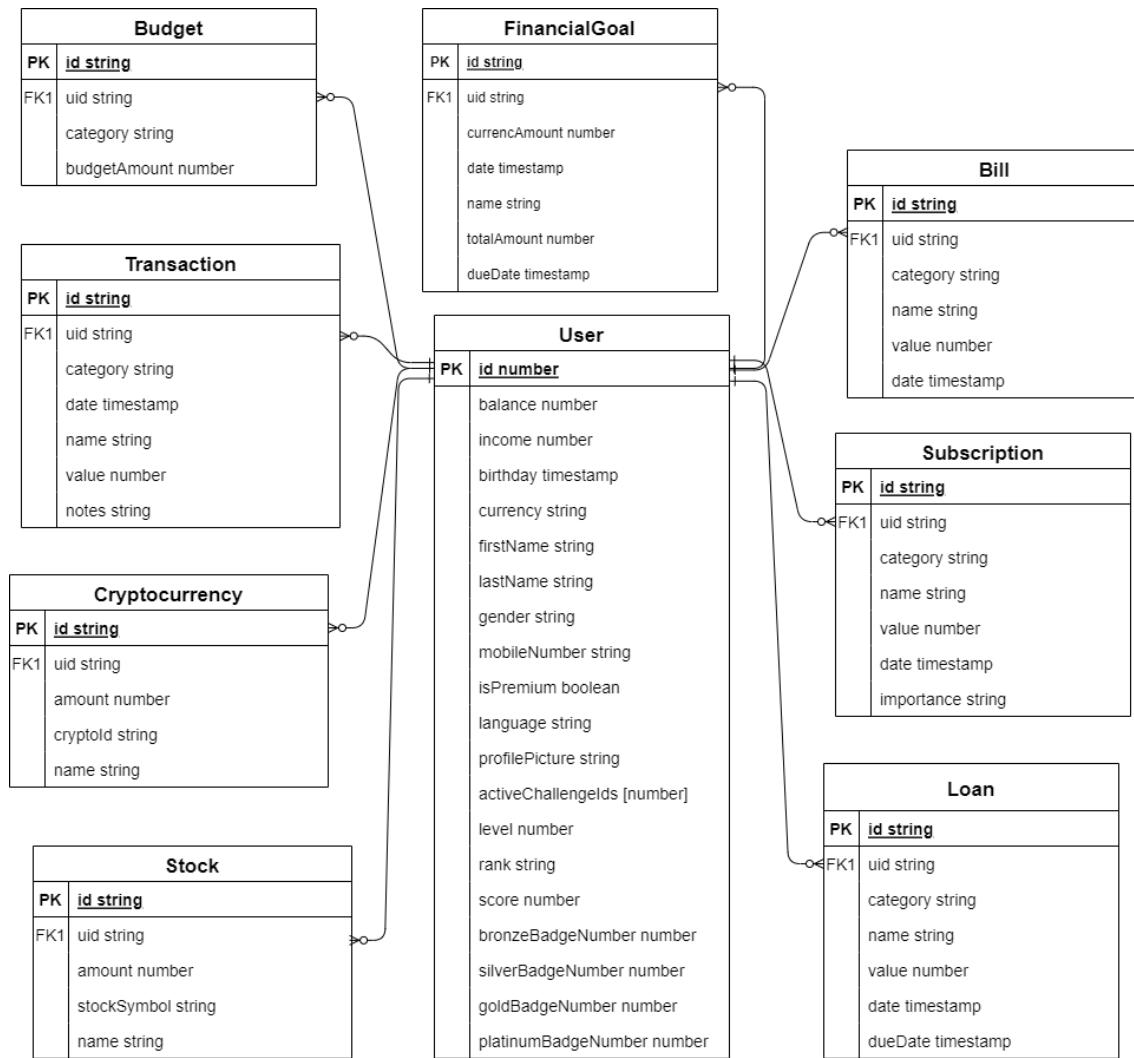
Összességében az alkalmazás architektúrája a modern technológiákat használja ki, hogy biztosítsa a rugalmasságot, a biztonságot és a skálázhatóságot. Az integrációk és a technológiák kiválasztása oly módon történt, hogy az alkalmazás megfeleljen a mai digitális kor felhasználói és üzleti követelményeinek.



4.1. ábra - Az alkalmazás architektúrája

## 4.2. Adatmodellezés

A Firebase adatbázisban tárolt dokumentumok struktúrája kulcsfontosságú az alkalmazásom működésében. Az adatmodell támogatja az alkalmazás funkcióinak széles körét, kezdve a felhasználói profilok kezelésétől egészen a pénzügyi tranzakcióig. Az alábbiakban az alkalmazás legfontosabb adattípusainak leírását szemléltetem, amelyek magukban foglalják a mezők típusait és a lehetséges értékeket.



4.2. ábra - Az alkalmazás Egyed-kapcsolat diagramja

#### 4.2.1. Felhasználó

A felhasználó dokumentum tartalmazza a regisztrált felhasználók alapvető adatait, amelyek lehetővé teszik az egyéni beállításokat és a pénzügyi tevékenységek nyomon követését az alkalmazásban. Létrejön a felhasználó regisztrációja utáni Onboarding beállítások után.

- id: A felhasználó egyedi azonosítója (string).
- balance: A felhasználó jelenlegi egyenlege (number).
- income: A felhasználó havi jövedelme, becsült érték (number).
- birthday: A felhasználó születésnapja (timestamp).
- currency: Az alkalmazásban használt pénznem, alapértelmezett érték: USD (string).
- firstName: A felhasználó keresztnéve (string).

- lastName: A felhasználó vezetékneme (string).
- gender: A felhasználó neme (string).
- mobileNumber: A felhasználó telefonszáma, opcionális (string).
- isPremium: A felhasználó prémium státusza, (boolean).
- language: A felhasználó által használt nyelv, (string).
- profilePicture: A felhasználó profilképének URL-címe, opcionális (string).
- activeChallengeIds: Az aktív kihívások egyedi azonosítói, opcionális (array).
- level: A felhasználó jelenlegi szintje, (number).
- rank: A felhasználó rangja, (string).
- score: A felhasználó aktuális pontszáma (number).
- bronzeBadgeNumber: A felhasználó bronz jelvényeinek száma (number).
- silverBadgeNumber: Az ezüst jelvények száma (number).
- goldBadgeNumber: Az arany jelvények száma (number).
- platinumBadgeNumber: A platina jelvények száma (number).

#### 4.2.2. Tranzakciók

A tranzakció dokumentumok a felhasználók pénzügyi műveleteit rögzítik, beleértve bevételeket és kiadásokat, amelyek alapját képezik a pénzügyi menedzsment funkcionak. Létrejön, ha a felhasználó felvesz egy bevételt vagy költséget.

- id: A tranzakció egyedi azonosítója (string).
- category: A tranzakció kategóriája, értékei: "Income" vagy 10 különböző kiadási kategória (string).
- date: A tranzakció dátuma, csak múltbeli vagy aktuális dátumok érvényesek (timestamp).
- name: A tranzakció neve (string).
- value: A tranzakció értéke USD-ben (number).
- paymentMethod: A tranzakció fizetési módja, opcionális, értékei: "Készpénz", "Bankkártya", "Hitelkártya" (string).
- notes: A tranzakció leírása, opcionális (string).
- uid: A tranzakciót létrehozó felhasználó egyedi azonosítója (string).

#### 4.2.3. Kriptovaluták

Ez a dokumentumok a felhasználók által birtokolt kriptovaluták adatait tartalmazzák, lehetővé téve a kriptovaluta portfólió kezelését. Létrejön, ha a felhasználó egy új kriptovalutát ad a portfóliójába.

- Id: Az egyedi azonosító (string)..
- amount: A felhasználó birtokában lévő valutaegység (number).
- cryptoId: A kriptovaluta egyedi azonosítója az API-ból (string)..
- name: A kriptovaluta neve (string)
- uid: A felhasználó egyedi azonosítója (string).

#### 4.2.4. Részvények

A részvény dokumentumok azon részvények adatait rögzítik, amelyekkel a felhasználók kereskednek, így segítve őket a befektetési portfólió kezelésében. Létrejön, ha a felhasználó egy új részvénnyt ad a portfóliójába.

- Id: A részvény egyedi azonosítója, (string).
- amount: A birtokolt részvényegységek mennyisége, (number).
- stockSymbol: A részvény egyedi azonosítója az API-ból, (string)..
- name: A részvény neve, (string).
- uid: A felhasználó egyedi azonosítója, (string).

#### 4.2.5. Számlák

Számla dokumentumok a rendszeres számlákat, mint például közművek vagy rezsi díjakat tartalmazhatják. Létrejön, ha a felhasználó felvesz egy előfizetést

- Id: A számla egyedi azonosítója (string).
- date: A fizetési dátum (timestamp).
- category: A számla kategóriája, értékei: "monthly", "yearly" (string).
- name: A számla neve (string).
- value: A számla összege (string).
- uid: A felhasználó egyedi azonosítója (string).

#### 4.2.6. Előfizetések

Előfizetés dokumentumok azokat a szolgáltatásokat tartalmazzák, melyekre a felhasználó rendszeresen előfizetett, beleértve azok gyakoriságát és díjait. Létrejön, ha a felhasználó felvesz egy előfizetést

- Id: Az előfizetés egyedi azonosítója (string).
- date: Az előfizetés fizetési dátuma (timestamp).
- category: Az előfizetés kategóriája, értékei: "weekly", "monthly", "yearly" (string).
- name: Az előfizetés neve (string).
- value: Az előfizetés díja (number).
- importance: Az előfizetés fontossága, értékei: "important", "necessary", "neutral", "negligible". (string).
- uid: A felhasználó egyedi azonosítója (string).

#### 4.2.7. Hitelek

Hitel dokumentumok a felhasználók által felvett hitelek részleteit tartalmazzák. Létrejön, ha a felhasználó felvesz egy hitelt.

- Id: A hitel egyedi azonosítója (string).
- date: A hitel fizetési dátuma (timestamp).
- category: A hitel kategóriája, értékei: "monthly", "yearly" (string).
- name: A hitel neve (string).
- value: A havi vagy éves fizetési összeg (number).
- dueDate: A fizetési határidő (number).
- uid: A felhasználó egyedi azonosítója (string).

#### 4.2.8. Pénzügyi célok

A pénzügyi cél dokumentumok a felhasználók által kitűzött hosszú- és rövidtávú pénzügyi céljaikat rögzítik, lehetővé téve a haladás nyomon követését és motivációt nyújtva a célok eléréséhez. Létrejön, ha a felhasználó beállít egy pénzügyi célt.

- id: A pénzügyi cél egyedi azonosítója (string).
- currentAmount: A cél jelenlegi összege (number).
- date: A cél létrehozásának dátuma (timestamp).

- name: A pénzügyi cél megnevezése (string).
- totalAmount: A cél eléréséhez szükséges teljes összeg (number).
- dueDate: A cél teljesítésének határideje (timestamp).
- uid: A cél létrehozójának egyedi azonosítója (string).

#### 4.2.9. Költségkeretek

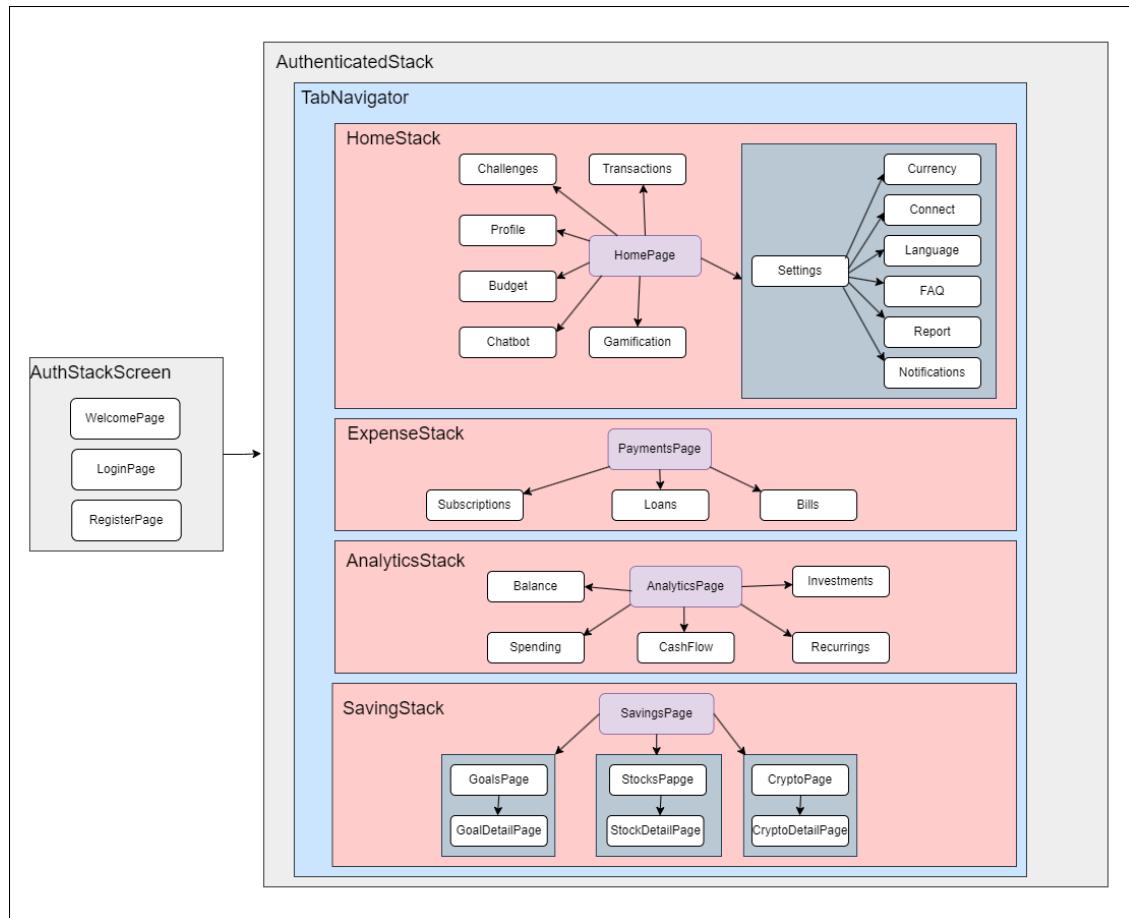
Költségkeret dokumentumok az egyes költségkategóriákra vonatkozó pénzügyi korlátokat tartalmazzák, amelyek segítenek a felhasználóknak kontroll alatt tartani kiadásaikat a megadott kereteken belül. Létrejön, ha a felhasználó beállít egy költségkeretet.

- id: A költségkeret egyedi azonosítója (string).
- category: A költségkeret kategóriája, értékei megegyeznek a tranzakciók kategória értékeivel (string).
- budgetAmount: A keretben szabott összeg (number).
- uid: A költségkeretet beállító felhasználó egyedi azonosítója (string).

Ezek a dokumentumok és az attribútumai kulcsszerepet játszanak az alkalmazás működésében, támogatva a felhasználói interakciókat, adatkezelést és a felhasználói élmény javítását. Az egységes és jól strukturált adatmodellezés biztosítja, hogy az alkalmazás hatékonyan kezelje és tárolja a felhasználók pénzügyi és személyes adatait.

## 5. Technológiai Implementáció bemutatása

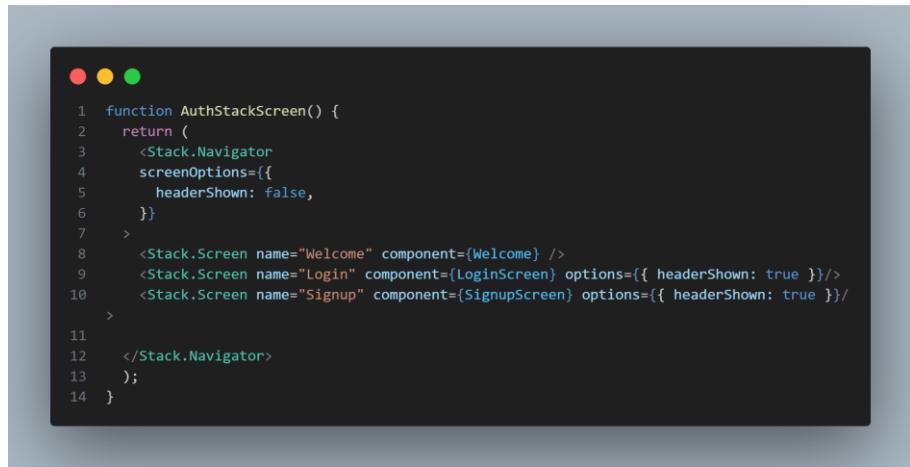
### 5.1. Routing és Navigáció



5.1. ábra - Az alkalmazás routingja

A React Native alkalmazásoknál a navigáció kritikus eleme a felhasználói élménynek. A navigációs rendszer megtervezésekor különféle módszerek közül választhatunk, mint például a stack, tab, és drawer navigátorok. Ezek a navigátorok lehetővé teszik, hogy könnyedén kezelhessük az alkalmazásban való navigációt, állapotokat, és a felhasználó által végzett interakciókat.

Az AuthStackScreen az autentikációs folyamatért felelős, amely az alkalmazás indításakor jelenik meg. Ez a stack navigátor kezeli az összes kezdő képernyőt, mint a Welcome, Login és Register oldalak. Ez egy lineáris navigációs folyamatot biztosít, ahol a felhasználó a bejelentkezési vagy regisztrációs folyamatot követően juthat tovább.



```
1 function AuthStackScreen() {
2   return (
3     <Stack.Navigator
4       screenOptions={{
5         headerShown: false,
6       }}
7     >
8       <Stack.Screen name="Welcome" component={Welcome} />
9       <Stack.Screen name="Login" component={LoginScreen} options={{ headerShown: true }}/>
10      <Stack.Screen name="Signup" component={SignupScreen} options={{ headerShown: true }}/>
11    </Stack.Navigator>
12  );
13}
```

5.2. ábra - *AuthStackScreen*

A TabNavigator a sikeres autentikáció után válik elérhetővé. Ez a komponens az alkalmazás fő navigációs rendszere, ami a Home, Payments, Analytics, és Savings főbb területeket teszi elérhetővé tabok formájában. Ezek a tabok gyors hozzáférést biztosítanak az alkalmazás különböző szekcióihoz anélkül, hogy újra kellene tölteni a képernyőket.



```
1 <Tab.Navigator
2   screenOptions={({ route }) => ({
3     tabBarStyle: { backgroundColor: '#F5F6F5', paddingBottom: 6, paddingTop: 6
4   },
5     tabBarActiveTintColor: '#35BA52',
6     tabBarIndicatorStyle: [
7       backgroundColor: '#35BA52',
8       height: '110%',
9     ],
10    headerShown: route.name === 'Welcome' ? true : false,
11  })}
12  >
13    <Tab.Screen
14      name={languages[selectedLanguage].welcome}
15      component={HomeStack}
16      options={({ navigation, route }) => ({
17        tabBarIcon: ({ color }) => (
18          <Image
19            source={require('./assets/home.png')}
20            style={{ tintColor: color, width: 24, height: 24 }}
21          />
22        ),
23        headerShown: route.name === 'Welcome' ? true : false,
24      })}
25    />
```

5.3. ábra - *Tab.Navigator* és *Tab.Screen*

Az egyedi Stack Navigátorok, mint például a HomeStack, ExpensesStack, SavingsStack, és AnalyticsStack, további részletességet adnak minden egyes fő terület navigációjához. Például a HomeStack kezeli a Home főoldalt és további aloldalakat, mint a Profile, Chatbot, vagy a Settings. Ez lehetővé teszi az alkalmazáson belüli mélyebb szintű navigációt azáltal, hogy minden fő terület saját kontextusban kezeli az aloldalakat. Egy Stack-ben pedig Stack Navigatorok vannak azon belül pedig a StackScreenekben kell megadni azt a komponens ahova el szeretnénk jutni az adott Stack-en.

## 5.2. Jogosultságkezelés

Az alkalmazáson belüli adatokat csak bejelentkezés után lehet elérni. Ezt biztosítja a routing, ami megakadályozza az összes oldal elérését a nem bejelentkezett felhasználóknak, kivéve az autentikációs oldalakat. A chatbot, kriptovaluta és részvények oldalak kizárolag az előfizetett felhasználóknak érhetőek el. Ezekre az oldalakra való navigáláskor a felhasználó isPremium boolean adattagját vizsgáljuk. Ha igaz, akkor a felhasználó elő van fizetve és elérheti az oldalakat, különben nem. Az 5.4-es ábra az adattagokhoz való hozzáférést írja le. A chatbot-ot is hozzátettem, amire ugyan nincs külön adat az adatbázisban természetesen, de külön adatforrásként is tekinthetünk rá.

C = létrehozás R = olvasás U = módosítás D = törlés	Költségkeret	Tranzakciók	Eltörlések	Számíták	Hitelek/adósságok	Pénzügyi célok	Küldetések	Kriptovaluta	Részvények	Chatbot
Bejelentkezett felhasználó	CRUD	CRUD	CRUD	CRUD	CRUD	CRUD	R	-	-	-
Bejelentkezett prémium felhasználó	CRUD	CRUD	CRUD	CRUD	CRUD	CRUD	R	CRUD	CRUD	R

5.4 ábra - Adatokhoz való hozzáférési jogok

## 5.3. Autentikáció

Az autentikáció két fő részből áll: a regisztráció és a bejelentkezés, mindenkor szorosan integrálva a Firebase Authentication szolgáltatással és a React Context API-val. Az auth.ts-ben létrehoztam egy authenticate metódust, ami paraméterben várja, hogy bejelentkezni, vagy éppen regisztrálni szeretnénk. A firebase SDK helyett a REST API-t hívom, a végpontnak csak meg kell adni, hogy bejelentkezni vagy regisztrálni kell.



```

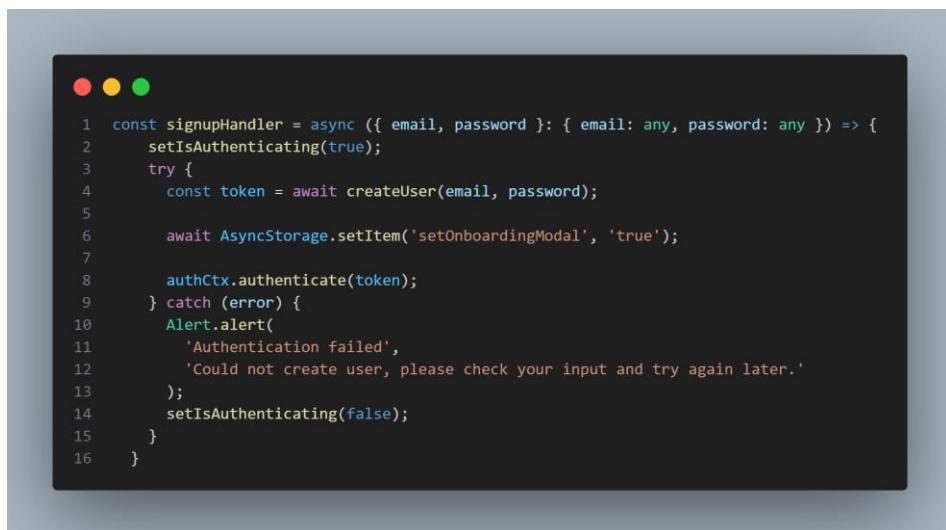
1  async function authenticate(mode: any, email: any, password: any) {
2    const url = `https://identitytoolkit.googleapis.com/v1/accounts:${{mode}}?key=${{apiKeys.FIREBASE_API_KEY}}`;
3
4    const response = await axios.post(url, {
5      email: email,
6      password: password,
7      returnSecureToken: true,
8    });
9
10   const token = response.data.idToken;
11
12   return token;
13 }

```

5.5. ábra - autentikációért felelős API hívás

### 5.3.1. Regisztráció

A regisztrációs folyamat a createUser függvény segítségével történik, amely az e-mail cím és jelszó alapján hoz létre új felhasználói fiókot a Firebase rendszerében. A felhasználó regisztrációja után azonnal egy token is generálódik, amely az autentikációs státusz azonosítására szolgál. A token elmentésre kerül az AsyncStorage-ba, ami lehetővé teszi a felhasználói munkamenet folytonosságát az alkalmazás újraindítása után is.



```
1 const signupHandler = async ({ email, password }: { email: any, password: any }) => {
2   setIsAuthenticating(true);
3   try {
4     const token = await createUser(email, password);
5     await AsyncStorage.setItem('setOnboardingModal', 'true');
6     authCtx.authenticate(token);
7   } catch (error) {
8     Alert.alert(
9       'Authentication failed',
10      'Could not create user, please check your input and try again later.'
11    );
12    setIsAuthenticating(false);
13  }
14}
15}
16}
```

5.6. ábra - *signupHandler* regisztrációs metódus

A createUser tehát meghívja a már fentebb említett authenticate metódust, ami a regisztrációs api-t hívja meg. Sikeres regisztráció és a kapott token lementése után lekérjük és beállítjuk az authContext segítségével a userId-t, azaz a felhasználó azonosítóját.

### 5.3.2. Bejelentkezés

A bejelentkezási folyamat hasonlóképpen működik, mint a regisztrálás. A metódus ugyan az, csupán login kulcsszóval hívjuk az autentikációs API-t. Sikeres bejelentkezés esetén a Firebase-től kapott token segítségével a felhasználó azonosítója (userId) és e-mail címe (email) lekérésre kerül, amit a Context API segítségével globálisan elérhetővé teszik az alkalmazáson belül.

### 5.3.3. Elfelejtett jelszó és kijelentkezés

A Firebase Authentication szolgáltatása támogatja a jelszó-visszaállítási folyamatot, amely lehetővé teszi a felhasználók számára, hogy biztonságos módon állítsanak vissza

jelszavakat egy e-mail címen keresztül. Az alkalmazás egy egyszerű, de hatékony módon kezeli ezt:

- E-mail cím ellenőrzése: A felhasználónak meg kell adnia az e-mail címét, amellyel regisztrált. A rendszer ellenőrzi, hogy az megfelel-e az e-mail cím formátumának.
- Jelszó-visszaállító e-mail küldése: Amennyiben a megadott e-mail cím érvényes, a Firebase sendPasswordResetEmail függvényén keresztül egy jelszó-visszaállítási linket küldünk erre a címre.
- Felhasználói visszajelzés: A felhasználó értesítést kap arról, hogy ellenőrizze postafiókját a további instrukciókért, vagy tájékoztatást kap, ha a megadott e-mail cím nem található a rendszerben.

A kijelentkezési folyamat során eltávolítjuk a felhasználó tokenét, azonosítóját és e-mail címét az állapotból, valamint az AsyncStorage-ból, így biztosítva, hogy a munkamenet adatok ne legyenek elérhetőek az alkalmazás újbóli indítása után. Az AuthContext segítségével a kijelentkezés funkciót globálisan elérhetővé tesszük így az alkalmazásban bárhol könnyedén meghívhatjuk azt, például a beállítások oldalon.

## 5.4. Beállítások

A legfontosabb beállítási lehetőségek közé tartozik a nyelv és a pénznem beállítása, amelyek közvetlenül befolyásolják, hogyan jelenik meg az információ és hogyan történik a tranzakciók kezelése az alkalmazásban.

### 5.4.1. Pénznem beállítása

Az updateUserCurrency metódus felelős a pénznem frissítéséért. Ez a folyamat magában foglalja a felhasználó adatbázisban történő pénznem frissítését, valamint a konverziós arány és a pénznem szimbólumának az AsyncStorage-ba való mentését. Az ExchangeRate API segítségével lekérjük az aktuális konverziós rátát az USD-ről a kiválasztott pénznemre illetve adatfeltöltéskor pedig fordítva, mert ott az általános pénznemre, azaz USD-re váltunk és azzal az értékkel mentjük el az összegeket az adatbázisra. Erre azért van szükség mert az adatbázisba minden egy egységes pénznemként kell menteni az értékeket.



```
1 const updateUserCurrency = async (newCurrency: string) => {
2     try {
3         const settingsQuery = query(collection(db, 'users'), where('uid', '==', userId));
4         const querySnapshot = await getDocs(settingsQuery);
5
6         if (!querySnapshot.empty) {
7             for (const doc of querySnapshot.docs) {
8                 await updateDoc(doc.ref, { currency: newCurrency });
9
10            const newSymbol = getCurrencySymbol(newCurrency)
11            // OpenExchangeRate API call from USD to the chosen currency
12            const result = await convertCurrencyToCurrency('USD', newCurrency);
13            await AsyncStorage.setItem('conversionRate', result.toString());
14            await AsyncStorage.setItem('symbol', newSymbol);
15        }
16    } else {
17        console.error('No user document found.');
18    }
19 } catch (error) {
20     console.error('Error updating user currency:', error);
21 }
22};
```

5.7. ábra - *updateCurrency* metódus

### 5.4.2. Nyelv beállítása

A updateUserLanguage funkció lehetővé teszi a felhasználók számára, hogy a kívánt nyelvre váltsanak, amely az alkalmazásban megjelenő szövegeket érinti. Az új nyelv beállítása után a felhasználói adatbázisban is frissül az aktuális nyelv, illetve a választott nyelv mentésre kerül az AsyncStorage-ban, így biztosítva a kiválasztott nyelvi beállítások állandóságát. A nyelveket JSON fájlok segítségével kezelem, amelyek a különböző nyelvek szövegeit tartalmazzák. Ezen fájlok importálása és használata lehetővé teszi, hogy dinamikusan változtassuk az alkalmazás nyelvét a felhasználói beállításoknak megfelelően.

## 5.5. Fénykép feltöltése

Ebben az alfejezetben bemutatom, hogyan valósítottam meg a felhasználói profilkép feltöltését a Firebase Storage használatával. A képfeltöltési folyamat során több technológiát és könyvtárat használtam, beleértve az Expo ImagePicker-t, az Expo ImageManipulator-t és a Firebase Storage-et. Az alábbiakban részletesen bemutatom az implementáció lépéseit és a kapcsolódó metódusokat.

### 5.5.1. Kép Kiválasztása

Az első lépés a kép kiválasztása a felhasználó galériájából az Expo ImagePicker<sup>[22]</sup> segítségével. Az alkalmazás először engedélyt kér a felhasználótól a média

hozzáféréséhez. Ha az engedély megadása megtörtént, a képválasztó elindul, és a felhasználó kiválaszthat egy képet. Ha a felhasználó kiválasztott egy képet, az URI-t (Uniform Resource Identifier) használjuk a további feldolgozáshoz.



```
1 const handleSelectImage = async () => {
2   setIsLoading(true);
3
4   const permissionResult = await ImagePicker.requestMediaLibraryPermissionsAsync();
5   if (!permissionResult.granted) {
6     Alert.alert("You've refused to allow this app to access your photos!");
7     return;
8   }
9
10  const pickerResult: any = await ImagePicker.launchImageLibraryAsync();
11  if (pickerResult.cancelled) {
12    setIsLoading(false);
13    return;
14  }
15
16  if (pickerResult.assets && pickerResult.assets.length > 0) {
17    const uri = pickerResult.assets[0].uri;
18
19    const resizedUri = await resizeImage(uri);
20    if (resizedUri) {
21      const uploadUrl = await uploadImage(resizedUri, userId);
22      if (uploadUrl) {
23        await updateUserProfileImage(uploadUrl);
24      }
25    } else {
26      setIsLoading(false);
27      console.error("No assets found in picker result");
28    }
29  }
30};
```

5.8. ábra - `handleSelectImage` metódus a fénykép kiválasztására

### 5.5.2. Kép átméretezése

Miután a felhasználó kiválasztotta a képet, az Expo ImageManipulator segítségével átméretezzük azt. Az átméretezés célja, hogy optimalizáljuk a fájl méretét, így csökkentve a feltöltési időt, tárolási költségeket és a méretproblémákból következő esetleges problémákat. Az átméretezés során a kép szélességét 800 pixelre állítjuk, és PNG formátumban tároljuk.

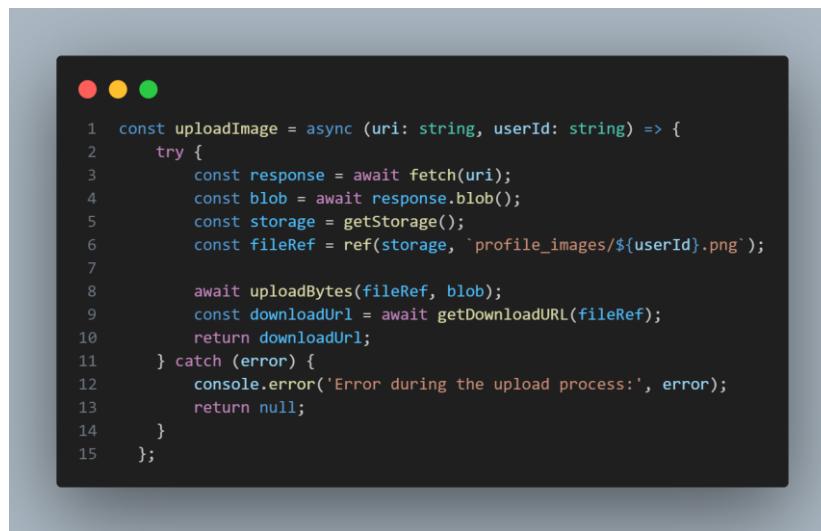


```
1 export const resizeImage = async (uri: string) => {
2   try {
3     const result = await ImageManipulator.manipulateAsync(
4       uri,
5       [{ resize: { width: 800 } }],
6       { compress: 0.7, format: ImageManipulator.SaveFormat.PNG }
7     );
8     return result.uri;
9   } catch (error) {
10     console.error("Error resizing image:", error);
11     return uri;
12   }
13};
```

5.9. ábra - `resizeImage` metódus

### 5.5.3. Kép feltöltése és a profil frissítése

Az átméretezett képet feltöltyük a Firebase Storage-ba az uploadImage() metódus segítségével. A képet a felhasználói azonosító alapján egyedileg azonosítjuk, és tároljuk. Miután a kép sikeresen feltöltésre került, a kapott URL-t elmentjük a Firestore adatbázisban a felhasználói dokumentumban, ezért az updateUserProfileImage() metódus felel, amely egy szimpla dokumentummódosítás a megfelelő profilePicture mezővel. Ez biztosítja, hogy a felhasználó profilja frissüljön az új képpel.



```
1 const uploadImage = async (uri: string, userId: string) => {
2     try {
3         const response = await fetch(uri);
4         const blob = await response.blob();
5         const storage = getStorage();
6         const fileRef = ref(storage, `profile_images/${userId}.png`);
7
8         await uploadBytes(fileRef, blob);
9         const downloadUrl = await getDownloadURL(fileRef);
10        return downloadUrl;
11    } catch (error) {
12        console.error('Error during the upload process:', error);
13        return null;
14    }
15};
```

5.10. ábra - uploadImage metódus

Ez a három lépés biztosítja, hogy a felhasználók könnyedén feltölthessék és frissíthessék profilképüket az alkalmazásban, miközben a képek biztonságosan tárolódnak a Firebase infrastruktúráján. A megfelelő hibakezelés és az aszinkron műveletek használata biztosítja a zökkenőmentes felhasználói élményt.

## 5.6. Adatkezelés firestore segítségével

Az alábbiakban ismertetem, hogy az alkalmazásban hogyan valósulnak meg a CRUD (olvasás, írás, módosítás, törlés), vagyis az adatkezelési műveletek Firestore használatával. Ezekre a műveletekre a firebase SDK-t használom. A Firebase SDK (Software Development Kit) egy fejlesztői eszközkészlet, amely lehetővé teszi a fejlesztők számára, hogy könnyen integrálják a Firebase szolgáltatásokat alkalmazásaiikba.

A Firestore adatbázisból történő adatok olvasása aszinkron módon történik. A fetchTransactions függvény például az aktuális felhasználó azonosítóját (uid) használja, hogy lekérje az összes hozzá tartozó tranzakciót a getDocs beépített függvény segítségével. Előtte azonban az adott query-t kell lekérnünk a collection metódus segítségével, amivel egy adott collection-ben tudunk dokumentumok között keresni.



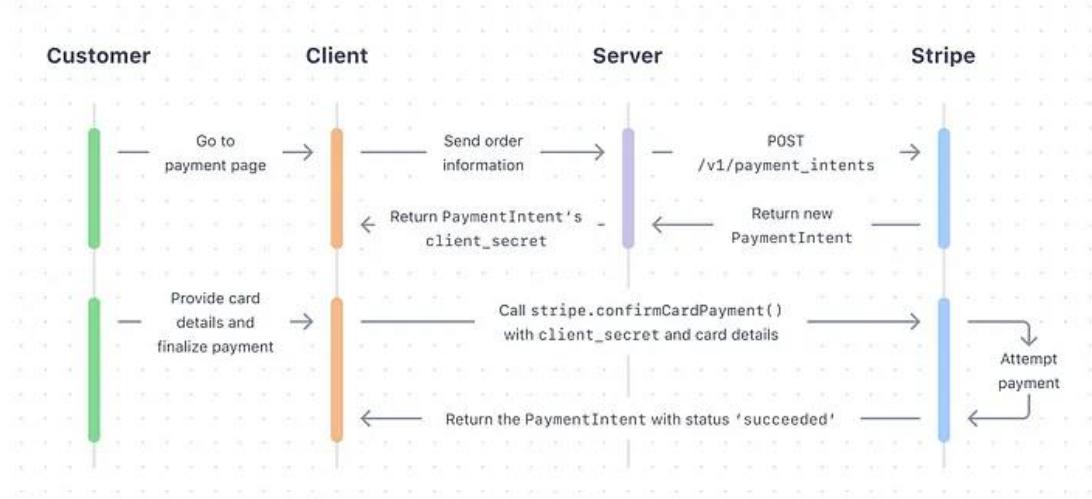
5.11. ábra - Adatok lekérése a firestore-ból

Új adatok hozzáadása a Firestore adatbázishoz az addDoc függvény segítségével történik. Az adatok törlése specifikus dokumentum azonosító alapján történik. A deleteDoc függvény segítségével törölhetünk egy adott tranzakciót. Meglévő adatok frissítése az updateDoc függvénnyel történik. Ebben az esetben egy már meglévő dokumentum adatait frissítjük. Itt szükségünk lesz a referenciahoz az adott dokumentum egyedi azonosítójára is.

Összességében, a Firestore segítségével történő adatkezelés nagyon hatékony módszer az alkalmazásadatok szervezésére és kezelésére. Segítségével nem kell külön foglalkozni az adatbázis kezeléssel, backend szerver megírásával, illetve ezek összekötésével.

## 5.7. Stripe integráció

A Stripe egy népszerű fizetési platform, amely lehetővé teszi az alkalmazások számára a biztonságos fizetési tranzakciók kezelését. Implementációja egyszerű, gyorsan lefejleszthető és karbantartható, biztonságos technológiát nyújtva a felhasználók számára.



5.12. ábra - Stripe fizetési folyamat ábrázolása<sup>[23]</sup>

### 5.7.1. Szerver oldal

A Stripe integráció az alkalmazásba jelentősen egyszerűsíti a fizetési folyamatokat, különösen a mikroszolgáltatások architektúrájának részeként használva, mint például az Express.js alapú mikroszerverek esetében. Az Express.js alkalmazás beállítása magában foglalja a fizetési útvonalak definiálását.

```

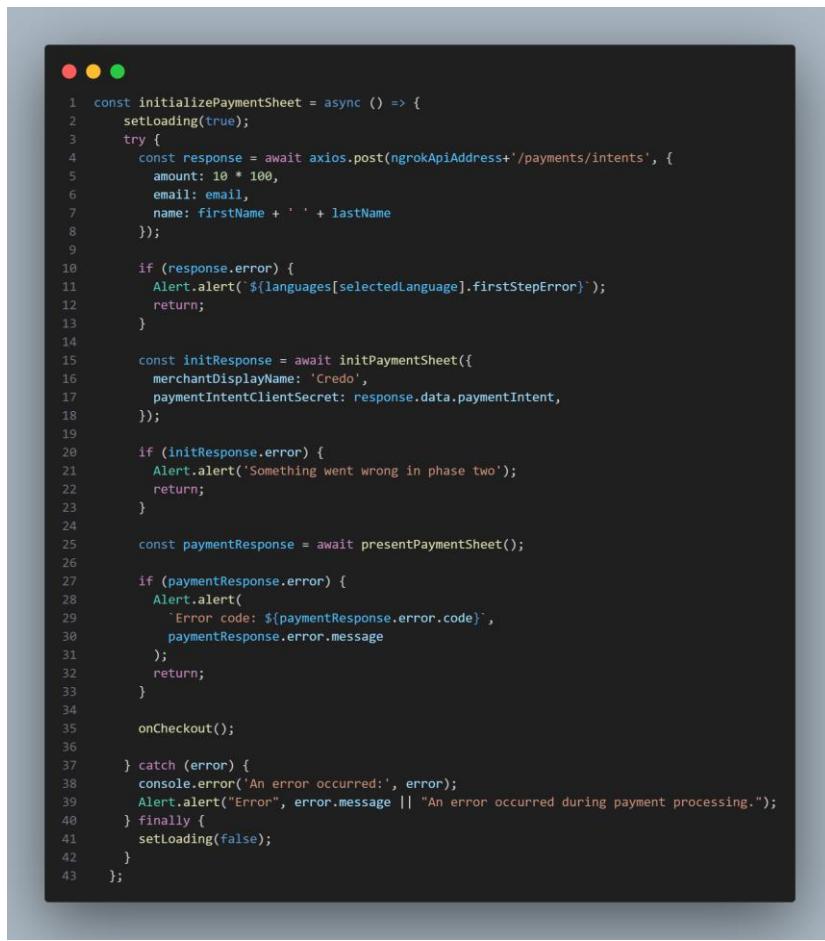
1 router.post('/intents', async (req, res) => {
2   try {
3     const customer = await stripe.customers.create({
4       email: req.body.email,
5       name: req.body.name,
6     });
7     const paymentIntent = await stripe.paymentIntents.create({
8       amount: req.body.amount,
9       currency: 'usd',
10      customer: customer.id,
11      automatic_payment_methods: {
12        enabled: true,
13      },
14    });
15    res.json({ paymentIntent: paymentIntent.client_secret });
16  } catch (e) {
17    res.status(400).json({
18      error: e.message,
19    });
20  }
21});
  
```

5.13. ábra - Payment Intent megvalósítása szerver oldalon

Ez a kód a /intents végponton fogadja a fizetési szándékok létrehozására vonatkozó kéréseket, ahol egy új Stripe ügyfél kerül létrehozásra, majd egy fizetési szándék jön létre az adott ügyfél számára. Ez az adatstruktúra tartalmazza a tranzakció összegét és pénznemét, valamint engedélyezi az automatizált fizetési módszereket.

### 5.7.2. Kliens oldal

A frontend integráció során a Stripe PaymentSheet komponensét használom, amely egy előre összeállított UI elemet biztosít a fizetési adatok biztonságos gyűjtésére és feldolgozására. A fizetési folyamat három fő lépésből áll: a fizetési szándék létrehozása, a fizetési adatok gyűjtése a PaymentSheet segítségével, és a tranzakció véglegesítése. Hibakezelés is be van építve minden lépésnél, hogy a felhasználókat tájékoztassa, ha valami hiba lép fel a folyamat során. Az API kommunikációhoz az Axios HTTP kliens könyvtárat használtam, melyek segítségével egyszerűen kezelhetőek a HTTP kérések, támogatja a promisz-alapú aszinkron műveleteket, így lehetővé teszi az alkalmazás számára, hogy hatékonyan kommunikáljon a backend szolgáltatásokkal.



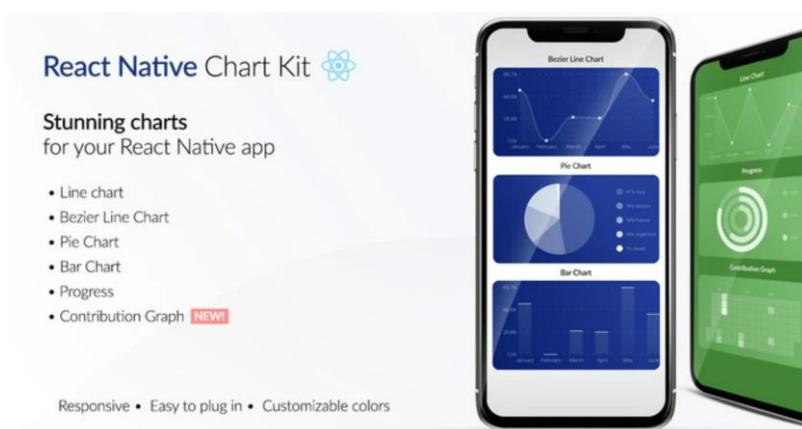
```
1 const initializePaymentSheet = async () => {
2   setLoading(true);
3   try {
4     const response = await axios.post(`https://api.ngrok.com/payments/intents`, {
5       amount: 10 * 100,
6       email: email,
7       name: firstName + ' ' + lastName
8     });
9
10    if (response.error) {
11      Alert.alert(`${languages[selectedLanguage].firstStepError}`);
12      return;
13    }
14
15    const initResponse = await initPaymentSheet({
16      merchantDisplayName: 'Credo',
17      paymentIntentClientSecret: response.data.paymentIntent,
18    });
19
20    if (initResponse.error) {
21      Alert.alert('Something went wrong in phase two');
22      return;
23    }
24
25    const paymentResponse = await presentPaymentSheet();
26
27    if (paymentResponse.error) {
28      Alert.alert(
29        `Error code: ${paymentResponse.error.code}`,
30        paymentResponse.error.message
31      );
32      return;
33    }
34
35    onCheckout();
36
37  } catch (error) {
38    console.error('An error occurred:', error);
39    Alert.alert("Error", error.message || "An error occurred during payment processing.");
40  } finally {
41    setLoading(false);
42  }
43};
```

5.14. ábra - Stripe integrációs metódus frontenden

Mivel a fejlesztési folyamat egy fizikai mobilkészüléken történt tesztelését is magában foglalta, szükségessé vált egy ngrok szervert állítani. Az ngrok<sup>[24]</sup> egy fordított proxy szolgáltatás, amely lehetővé teszi a localhoston futtatott szerverek elérését az internetről. Ez különösen hasznos a mobilfejlesztés során, amikor a helyi fejlesztési környezet közvetlen elérése nem lehetséges a külső hálózatokról.

## 5.8. Diagramok, elemzések készítése

Az alkalmazásban implementált diagramok és elemzések kulcsfontosságúak a felhasználói élmény és az adativizualizáció szempontjából. A diagramok segítségével a felhasználók könnyen átláthatják pénzügyi helyzetüket, észlelhetik a mintázatokat, és jobban megérthetik, hogyan oszlanak meg kiadásaik. Az implementáció során több külső könyvtárat is felhasználtam, amelyeket konfigurálni kellett az adatok megfelelő formázásával, szűrésével, hogy azok megfelelően jelenjenek meg. Az egyik ilyen könyvtár a react-native-chart-kit, amely széles körű támogatást nyújt a mobilalkalmazások számára különböző típusú grafikonok, többek között vonal-, sáv- és kördiagramok készítéséhez.



5.15. ábra - React Native Chart Kit<sup>1[25]</sup>

### 5.8.1. Adatok Szűrése és Feldolgozása

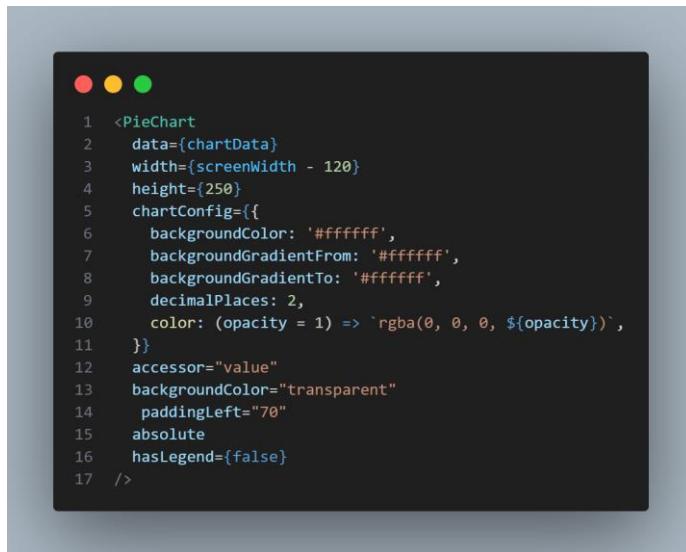
A következő példa a tranzakciós adatok kategória szerinti eloszlása, az aktuálisan, a felhasználó által dátum szerint filterezve, a tranzakciók pénzértéke szerinti százalékos eloszlást reprezentálva. A tranzakciós adatok hatékony szűrése kulcsfontosságú az adatok releváns vizualizációjához. A tranzakciók dátum szerinti szűrését a következő logika alapján végezem. Ezért a dátum szerinti szűrésért a filterTransactionsByDate metódus felel. Három szűrési típus létezik a heti, havi és éves szűrés.

A useMemo hook használata növeli a teljesítményt azzal, hogy memoizálja a szűrt tranzakciók listáját, csökkentve ezzel a szükségtelen újraszámításokat mikor éppen a hónapok vagy hetek, esetleg évek között váltunk. Ezt a filterezett és memorizált, szűrt listát adom át az aktuális Diagram komponenseknek.

### 5.8.2. Diagram komponensek implementálása

A vizualizációk megvalósításához saját DonutChart, BarChart, PieChart és egyéb komponenseket használtam, amelyekbe a memoizált adatokat áadtam.

A következő példában a PieChart komponenst használtam, amely a már említett react-native-chart-kit könyvtárból származik. Ez a komponens a kategóriánkénti kiadásokat mutatja be színes kördiagram formájában:



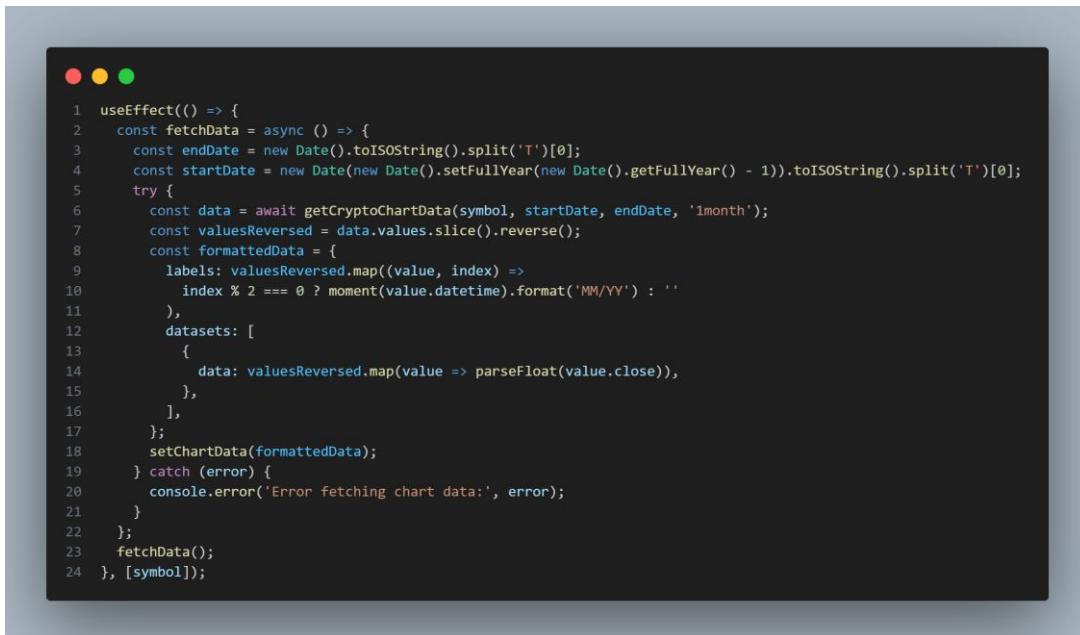
5.16. ábra - PieChart komponent konfigurálása

A komponens adatkészlete egy összesítő adaton való map-eléssel jön létre. Az adatok aggregálása a kategória szerinti osztályozáshoz pedig a következő módon történik:

A reduce függvény második paramétereként egy üres tömböt adok meg, ami az acc kezdeti értéke. Ez fogja tartalmazni az összesített adatokat. A data tömb minden egyes eleme, amely egy tranzakciót reprezentál, sorra kerül feldolgozásra. minden iteráció során a függvény meghívódik, és az aktuális tranzakció (transaction) és az eddigi akkumulált érték (acc) a paraméterei. Mindig megkeressük, hogy van-e már olyan elem, amelynek a name attribútuma megegyezik az aktuális tranzakció category attribútumával. Ezt a find metódus segítségével történik, ami visszaadja az első találatot, ha talál egyezést, különben undefined (nincs érték) értéket ad vissza. Ha találunk már létező kategóriát (existingCategory), akkor hozzáadjuk az aktuális tranzakció értékét (transaction.value) az adott kategória értékéhez. Ha nem találunk meglévő kategóriát, akkor létrehozunk egy új objektumot, amely tartalmazza a tranzakció kategóriáját (name) és értékét (value), majd hozzáadjuk ezt az objektumot a tömbhöz.

### 5.8.3. Kripto és Részvényértékek Vizualizációja LineChart segítségével

A LineChart alkalmazása szintén a react-native-chart-kit könyvtárból származik, amely lehetővé teszi, hogy a felhasználók áttekinthető és intuitív módon láthassák a kriptovaluták illetve a részvények értékeinek időbeli változását.



5.17. ábra - LineChart komponenshez szükséges adatbetöltés

Az adatlekérés egy useEffect hookban történik, ahol először meghatározom az aktuális dátumtól számított egy évvel korábbi dátumot kezdőpontként, hogy a felhasználók betekinthessék az éves tendenciákba. Ez a folyamat biztosítja, hogy a felhasználók a legfrissebb éves adatokkal rendelkezzenek. A getCryptoChartData függvényt használom az adatok lekérésére egy külső, twelvedata nevű API-tól, amely az adott szimbólumot, a kiválasztott intervallumot, valamint a kezdő és záró dátumot veszi figyelembe és visszaadja a kiválasztott kriptovaluta, vagy részvény értékeinek eloszlását az aktuális két dátum között, a megadott intevallumegységek szerint.

A LineChart komponens konfigurálása történik ezután. A LineChart bezier tulajdonsága simított vonalakat eredményez, amelyek esztétikailag vonzóbbá teszik a grafikont. A formatYLabel függvény dinamikusan formázza az Y tengelyen megjelenő értékeket a nagyságuk alapján, ami segít a különböző értéktartományok jobb áttekinthetőségében. A grafikon adatkészletei pedig egy az egyben az általam összerakott és leformázott adatok lesznek.

## 5.9. Kihívások implementálása

Az alkalmazáson belül a felhasználók tudnak csatlakozni különböző hihívásokhoz. Ezek különböző témaújak és sok van belőlük, de egy példán keresztük egyszerűen bemutatom azok kezelését. Ebben a példában egy olyan egyszerű kihívást implementálok, ami akkor sikeres a felvétel után, ha a felhasználónak van összesen 3 sikeresen teljesített pénzügyi célja.

Minden kihívásért egy metódus felel, amelyek nagyrészt hasonló struktúrával rendelkeznek. A `checkAndCompleteFinancialGoalSettingsChallenge()` metódus felel ezért a kihívásért. A metódus elején először is lekérjük, hogy a felhasználó rendelkezik-e ezzel a kihívással, azaz csatlakozott-e az adott nevű kihíváshoz. Ha nem akkor kilépünk a metódusból, ha igen akkor tovább megyünk. minden kihívásnak van egy teljesítési intervalluma, amely rendelkezésre áll a felvételtől kezdődően. Ez hetekben van minden megadva. Egy küldetéshez való csatlakozáskor eltároljuk annak a csatlakozási idejét, így könnyedén meg tudjuk állapítani, hogy meddig kell teljesíteni az adott kihívást. Ha benne vagyunk ebbe az intervallumba és az adott feltétel sikerült akkor egy kis ablakban jelezük a felhasználónak, hogy sikeresen teljesítette a kihívást, és pontot adunk neki, amivel később adott összeg elérésével szintet tud lépni. Továbbá fontos, hogy a csatlakozott küldetés `isActive` adattagján hamisra állítsuk, azaz teljesítetre állítjuk, így már nem a csatlakozott küldetések között fog megjelenni, hanem a teljesítettek között.

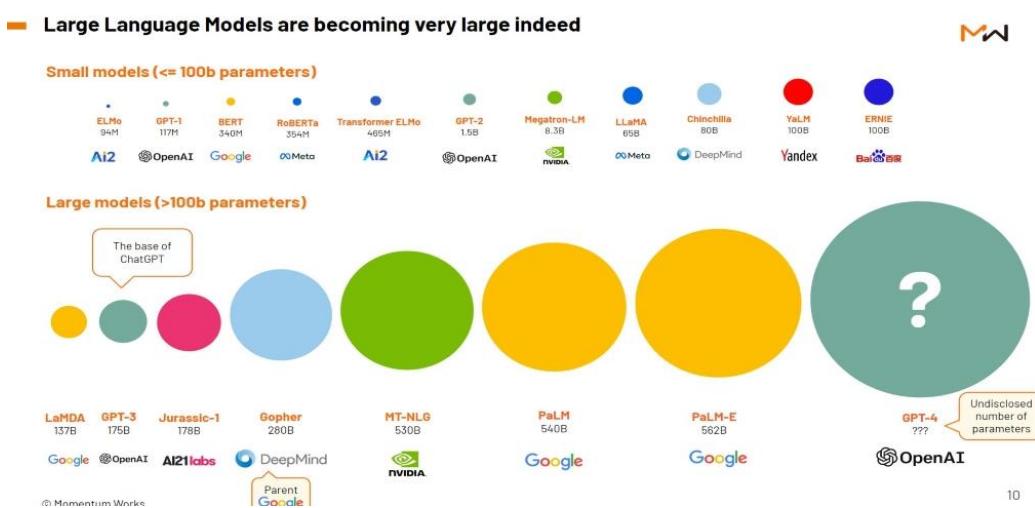
Az adott feltételek minden küldetés függők, de ebben az esetben elegendő minden a pénzügyi célok frissítésekor megnézni, hogy hány teljesített célja van eddig a felhasználónak. Ha ez nagyobb egyenlő, mint a kihívásban szerepel, akkor meghívjuk ezt a `checkAndCompleteFinancialGoalSettingsChallenge()` metódust abban a metódusból, ahol listázzuk a pénzügyi célokat.

## 6. Nagy Nyelvi Modell Integrációja

A modern mobilalkalmazások egyre inkább támaszkodnak mesterséges intelligencia (AI) technológiákra, hogy növeljék interaktív képességeiket és javítsák a felhasználói élményt. Az OpenAI nagy nyelvi modelljeinek, mint a GPT sorozatnak az integrálása lehetővé teszi, hogy az alkalmazások emberhez hasonló nyelvi képességekkel rendelkezzenek. Az alábbi alfejezetek részletesen bemutatják, hogyan hajtható végre és használható fel ez a technológia az alkalmazásomban. Fontos, hogy az AI fel fogja használni a bejelentkezett felhasználó saját adatait és azokra specifikus válaszokat, elemzések, ajánlásokat is fog tudni adni egész pontos keretek között.

### 6.1. Nagy Nyelvi Modellek Áttekintése

A nagy nyelvi modellek olyan előre kiképzett, mély tanulási modellek, amelyek képesek a természetes nyelv feldolgozására és generálására. Ezek a modellek, mint például a GPT-3, több milliárd paramétert tartalmaznak, amelyek a nagy adathalmazokon való tanulás eredményeképpen képesek összetett nyelvi feladatok megoldására. A nyelvi modellek felhasználhatók szövegértelmezésre, összefoglalásra, fordításra, valamint emberi természetű interakciókra, amelyek jelentősen javíthatják a felhasználói élményt. Ezekből tehát több is létezik, nem csak az openai modellje, ilyen például a google PALM modelljei vagy az NVIDIA modellje.

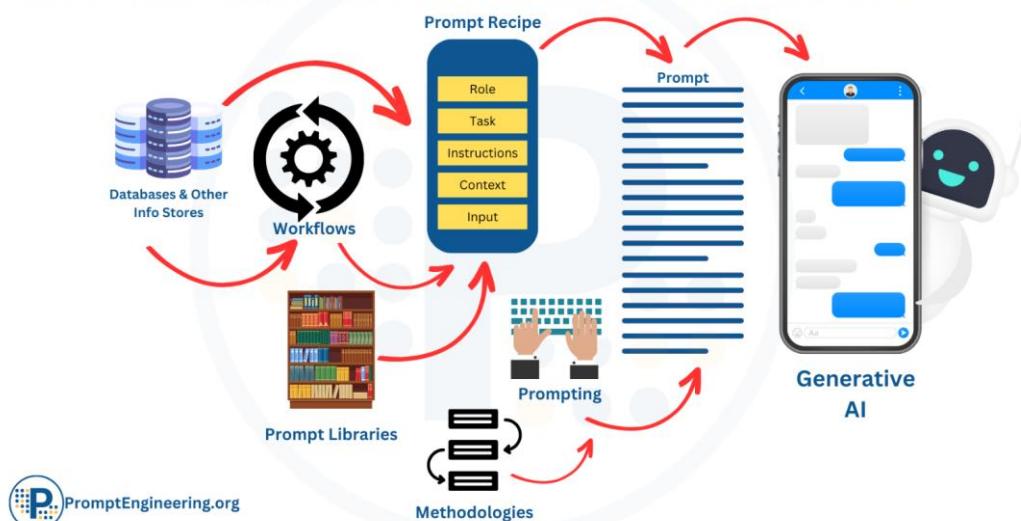


6.1. ábra - Nagy nyelvi modellek méretének összehasonlítása<sup>[26]</sup>

## 6.2. Prompt Engineering

A prompt engineering egy művészeti és tudományos terület, amely arra koncentrál, hogy a mesterséges intelligenciától pontos és releváns válaszokat kapjunk azáltal, hogy gondosan megformáljuk a neki feltett kérdéseket vagy utasításokat. Egy jól megtervezett prompt növeli a modell pontosságát, segít elkerülni a félreértéseket és növeli a kimenet relevanciáját. A promptok hatékony tervezése esszenciális az AI-alapú rendszerekben, ahol a kontextus és a kérdés pontos megfogalmazása közvetlenül befolyásolja a válasz minőségét. Fontos, hogy ne legyen se túl részletes egy kérdés, de se túl egyszerű. Ennek az arany középútnak a megtalálása nem egyszerű, főleg ha nem tudjuk előre, mit is fogunk kérdezni, illetve ha nagy mennyiségű adatot fogunk átadni a modellnek. Éppen ezért a saját példámra nem feltétlen mindenkor fogunk kapni pontos választ elsőre, de a felhasználónak lehetősége lesz majd ezek korrigálására, újra kérdezésére.

What is Prompt Engineering? Everything that goes before the prompt



6.2. ábra - Prompt engineering leírására szolgáló ábra<sup>[27]</sup>

## 6.3. Mesterséges Intelligencia és Chatbotok

Az AI-alapú chatbotok egyre fontosabb szerepet töltnek be az ügyfélszolgálatban, az oktatásban vagy bármilyen interaktív alkalmazásban. Ezek a chatbotok képesek természetes párbeszédet folytatni a felhasználókkal, valamint reagálni és segítséget nyújtani kérdéseikre. Az integrált nyelvi modellek lehetővé teszik a chatbotok számára, hogy kontextusérzékeny válaszokat adjanak. Az AI-chatbotok használata jelentősen csökkenheti a humán erőforrásokra gyakorolt terheket, miközben javítják a felhasználói élményt és növelik az elégedettséget.

## 6.4. Implementáció és Alkalmazás

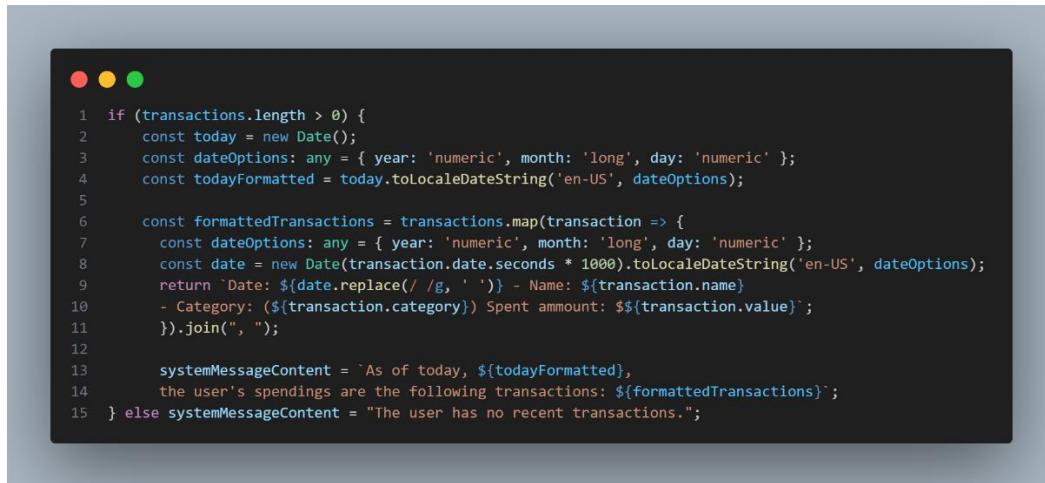
### 6.4.1. Adatgyűjtés és Feldolgozás

A chatbot funkcionálitásának egyik alapja a felhasználói adatok hatékony gyűjtése és feldolgozása. Az alkalmazásban először összegyűjtöm a felhasználói tranzakciókat, jövedelmeket és egyéb pénzügyi adatokat amikkel kapcsolatban kérdezhetjük az AI-t. Fontos, hogy a chatbot mindenkor a legfrissebb adatokkal dolgozzon. A useEffect() hook lekéri az összes friss adatot.

```
1  useEffect(() => {
2    async function fetchData() {
3      setIsLoading(true);
4      try {
5        const [transactions, incomes, subscriptions, loans, bills, stocks, cryptos, goal, budgets, ] = await Promise.all([
6          fetchTransactions(userId), fetchIncomes(userId), fetchSubscriptions(userId), fetchLoansAndDebts(userId),
7          fetchBills(userId), fetchStocks(userId), fetchCryptos(userId), fetchGoals(userId),
8        ]);
9        setTransactions(transactions);
10       setIncomes(incomes);
11       setSubscriptions(subscriptions);
12       setLoansAndDebts(loans);
13       setBills(bills);
14       setStocks(stocks);
15       setCryptocurrencies(cryptos);
16       setGoals(goal);
17     } catch (error) {
18       setIsLoading(false);
19     }
20   } finally {
21     setIsLoading(false);
22   }
23 }
24 }
25
26 fetchData();
27 }, [userId]);
```

6.3. ábra - Felhasználói adatok lekérése a chatbot működéséhez

Ezt követően ezeket szegmentálja és strukturálja, hogy az AI megfelelően elemezni tudja őket. Ebben a részben már az ehhez az adathoz tartozó alapvető prompt üzenet megadása is megtörténik. Továbbfejlesztési lehetőség lehet itt még, hogy adott kategóriákon belül további részletekig menjünk és a felhasználó által feltett kérdést is elemezzük és adott kontextus előfordulása esetén egyedi promptot adjunk. Például ha a felhasználó olyan üzenetet adna, amiben szerepel valamilyen formában az „ebben a hónapban” kifejezés, akkor ne az összes tranzakciót adjuk be a chatbotnak, hanem szűrjük le az adatot adott hónapban felvettekre. Ezzel növelnénk a hatékonyságot, a gyorsaságot és a költséget is.



```
1 if (transactions.length > 0) {
2     const today = new Date();
3     const dateOptions: any = { year: 'numeric', month: 'long', day: 'numeric' };
4     const todayFormatted = today.toLocaleDateString('en-US', dateOptions);
5
6     const formattedTransactions = transactions.map(transaction => {
7         const dateOptions: any = { year: 'numeric', month: 'long', day: 'numeric' };
8         const date = new Date(transaction.date.seconds * 1000).toLocaleDateString('en-US', dateOptions);
9         return `Date: ${date.replace(/\ /g, ',')}, Name: ${transaction.name}
10        - Category: ${transaction.category} Spent amount: ${transaction.value}`;
11     }).join(", ");
12
13     systemMessageContent = `As of today, ${todayFormatted},
14     the user's spendings are the following transactions: ${formattedTransactions}`;
15 } else systemMessageContent = "The user has no recent transactions.";
```

6.4. ábra - Felhasználói adat formázása a promphoz

#### 6.4.2. Interakciós Logika és Promptok

A chatbot intelligenciájának központi eleme az interakciós logika, amely meghatározza, hogyan reagáljon a bot a különböző felhasználói parancsokra és kérdésekre. Ez magában foglalja a különböző pénzügyi kategóriákra (kiadások, jövedelmek, hitelek, befektetések, célok, stb.) vonatkozó specifikus promptok kialakítását és alkalmazását, amelyek segítik a felhasználót a pénzügyi helyzetének jobb megértésében. Az alábbi kód részlet a chatbot válaszainak előkészítését mutatja be, amelyek a felhasználó által választott pénzügyi témahez kapcsolódnak. A chatbot intelligenciáját a felhasználó választásai és a promptok alapján szabályozott interakciók jellemzik. A felhasználó először kiválaszt egy témat, majd arról tudja kérdezni a chatbotot. A téma kiválasztásakor pedig már tudni fogjuk, milyen kategória adataira szűrhetünk és milyen promptokból választhatunk majd.



```
1 const handleTopicSelection = (topicKey: string) => {
2     const topic = topicsTranslations[selectedLanguage][topicKey];
3     const messageTemplate = topicMessages[selectedLanguage].topicPrompt;
4     const message = messageTemplate.replace('{topic}', topic);
5
6     setConversation(prev => [...prev, `Chatbot: ${message}`]);
7     setCurrentTopic(topicKey);
8     setShowTopics(false);
9};
```

6.5. ábra - Téma beállításáért felelős handleTopicsSelection metódus

A 6.5-ös ábrán lévő handleTopicSelection() függvény lehetővé teszi a felhasználó számára, hogy kiválassza az érdeklődési körébe tartozó pénzügyi témat. Ezután a chatbot egy nyitó üzenettel reagál, amely meghívja a felhasználót, hogy tegyen fel kérdéseket a kiválasztott témaban.

A különböző pénzügyi témahez tartozó specifikus promptok a felhasználó által végzett interakciók alapján aktiválódnak. Az alábbi példa bemutatja, hogyan alkalmazok egy promptot a felhasználói kérdésekre adott válaszok generálására. A chatbot logikájának magja a handleSelect() helper metódus amely minden a 9 kategóriára felépít a szükséges promptokat, elküldi az API híváshoz szükséges megfelelő kéréseket és beállítja az üzeneteket. Ebben a példában a felhasználó a költségeivel kapcsolatban akart kérdezni.

Ha a chatbot nem vár további interakciót, vagyis az isAwaitingFollowUp nincs beállítva, akkor feldolgozza a felhasználói inputot, és hozzáadja azt a beszélgetéshez. A isAwaitingFollowUp igazra állítódik és nem nézünk választott kategóriát, ha a chatbot egy megválasztott kérdése után felteszi azt a kérdést, hogy új kategóriára akarunk kérdezni, vagy megint fel szeretnénk tenni egy adott kérdést az aktuálisan kiválasztott kategóriával kapcsolatban. Ha igen vagy nem választ adunk, akkor kilépünk ebből az állapotból és vagy új kérdésre vár a chatbot (ha igen választ kapott), vagy az új kategória kiválasztását várja (ha nem választ kaptunk).

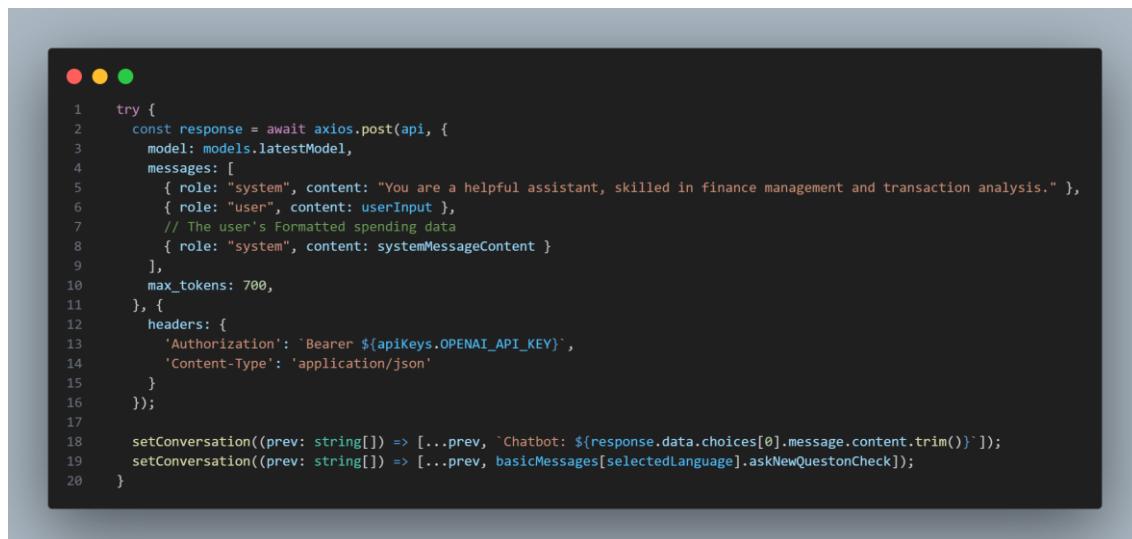


```
1 if (isAwaitingFollowUp) {
2   setIsAnswerLoading(false);
3   const userResponse = userInput.trim().toLowerCase();
4   if (userResponse === 'yes') {
5     setConversation((prev: string[]) => [...prev, `You: ${userInput}`,
6     `Chatbot: What else would you like to know about ${currentTopic}?`]);
7     setUserInput('');
8   } else if (userResponse === 'no') {
9
10    setShowTopics(true);
11    setConversation((prev: string[]) => [...prev, `You: ${userInput}`, basicMessages[selectedLanguage].selectFromTopics]);
12    setCurrentTopic('');
13    setUserInput('');
14  } else {
15    setConversation((prev: string[]) => [...prev, `You: ${userInput}`,
16    `Chatbot: I didn't really catch that, can you answer again with 'yes' or 'no', please?`]);
17    setUserInput('');
18    return;
19  }
20 }
```

6.6 ábra - *isAwaitingFollowUp* eset kezelése

Miután a felhasználó kérdezett az aktuális témahez, a téma (currentTopic) alapján megfelelően formázzuk a rendszerüzenetet a már említett 6.4-es ábrán lévő kóddal. Ebben az esetben a Spendings téma alapján a költési tranzakciókat dolgozzuk fel. A költési tranzakciókat a megfelelő formátumba alakítjuk, hogy azok könnyen érhetőek legyenek. A tranzakciók dátumát, nevét, kategóriját és értékét összegezzük, illetve a mai dátumot is átadjuk a promptnak, mivel ez sok kérdést megkönnyít és esetleges hibás válaszokat akadályoz meg.

A már említett OpenAI API segítségével kérünk választ a chatbot számára, a felhasználói kérdés és a rendszerüzenet alapján. A modell a legújabb elérhető változatát használom (`models.latestModel`), amely nem más mint a `gpt-4o`. Az API kérésben szereplő System Message a modell számára biztosít kontextust, meghatározva, hogy a chatbot egy segítőkész asszisztens, aki jártas a pénzügyi menedzsmentben és tranzakciók elemzésében (mivel a felhasználó a költségek kategóriát választotta ki). A másik system message pedig a már említett formázott adat amiben szerepel többek között a felhasználói formázott adat. A User Message üzenet pedig a felhasználói inputot tartalmazza, amit a felhasználó kérdezett a chatbottól. A kérés során meghatározzuk a maximum tokenek számát is, ami 700 ebben az esetben, hogy korlátozzuk a válasz hosszát. Ez a szám elég nagy, de így pontos és részletes választ kapunk. Ezek után a chatbot válaszát a `setConversation` metódussal hozzáadjuk a beszélgetéshez és várjuk az újabb felhasználói reakciót arra, hogy új kérdést akar-e felenni az adott témaban, vagy új témát akar-e választani.



```
1  try {
2    const response = await axios.post(api,
3      {
4        model: models.latestModel,
5        messages: [
6          { role: "system", content: "You are a helpful assistant, skilled in finance management and transaction analysis." },
7          { role: "user", content: userInput },
8          // The user's Formatted spending data
9          { role: "system", content: systemMessageContent }
10        ],
11        max_tokens: 700,
12      },
13      {
14        headers: {
15          'Authorization': `Bearer ${apiKeys.OPENAI_API_KEY}`,
16          'Content-Type': 'application/json'
17        }
18      });
19      setConversation((prev: string[]) => [...prev, `Chatbot: ${response.data.choices[0].message.content.trim()}`]);
20      setConversation((prev: string[]) => [...prev, basicMessages[selectedLanguage].askNewQuestionCheck]);
21    }
22  }
```

6.7. ábra - Egy teljes system és user prompt beadása, OpenAI API meghívása és a válasz beállítása

Ez a logika biztosítja, hogy a felhasználói kérdésekre adott válaszok pontosak és relevánsak legyenek, miközben a rendszer zökkenőmentesen kezeli a különböző pénzügyi témahez kapcsolódó interakciókat. Ehhez hasonlóan további 8 kategóriára oldottam meg a promptok kezelését, amely hasonló struktúrában épül fel, de természetesen más promptokkal és adatformázással.

## 7. Tapasztalatok, továbbfejlesztési lehetőségek és összegzés

Az elkészült projekt egy keresztplatformos pénzügyi költségvetés-kezelő alkalmazás, amely React Native-ben készült, és lehetővé teszi a felhasználók számára, hogy hatékonyan kezeljék pénzügyeiket különböző eszközökön. A pénzügyek iránt minden is nagy érdeklődést mutattam, ezért öröömre szolgált, hogy diplomamunkám keretében egy olyan projekten dolgozhattam, ami lehetővé teszi számomra, hogy mélyebb betekintést nyerjek a pénzügyi alkalmazások fejlesztésébe.

### 7.1. Projekt Tapasztalatok

A diplomamunkám során sikerült mélyrehatóan megismernem és elsajtítani a React Native technológiát, ami kulcsfontosságú volt az alkalmazás keresztplatformos fejlesztése szempontjából. Ennek eredményeképpen létrehoztam egy jól működő, felhasználóbarát pénzügyi költségvetés-kezelő alkalmazást, amely hatékonyan támogatja a felhasználókat pénzügyeik menedzselésében és a pénzügyi tudatosságban való fejlődésben. Az alkalmazás stabilan fut mind iOS, mind Android platformokon, biztosítva ezzel a széles körű elérhetőséget.

A projekt során számos kihívásnak néztem elébe, amelyek megoldása során jelentősen bővült a tudásom nem csak a mobilalkalmazás-fejlesztés területén, hanem a felhasználói élmény finomításában. Az alkalmazás fejlesztése során szerzett tapasztalatok és a projekt dokumentációjának elkészítése közben számos továbbfejlesztési ötlet fogalmazódott meg bennem, amelyek jelentősen növelhetik az alkalmazás piaci potenciálját és felhasználói elégedettségét.

### 7.2. Továbbfejlesztési lehetőségek

Jövőbeli terveim között szerepel az alkalmazás publikálása az App Store-ban és a Google Play Áruházban, ami lehetővé teszi, hogy szélesebb közönséghez juttassam el a terméket. Ezen felül tervezem az in-app fizetési lehetőségek bevezetését a Stripe-os fizetés helyett, hiszen előfizetés kezelésre éles környezetben erre lesz szükség. Emellett a banki szinkronizációs lehetőségek bevezetése is felmerült ötletnek, valamilyen harmadik féltől származó szolgáltatással, amely lehetővé tenné a felhasználók számára, hogy valós időben kövessék banki tranzakcióikat és egyenlegeiket az alkalmazáson keresztül, ennek persze biztonsági követelményei vannak, így ez még csak tervben van.

### 7.3. Összegzés

Úgy érzem, sikerült egy olyan alkalmazást létrehoznom, amely megfelel az elvárásoknak, és amelyet a felhasználók könnyedén és gyorsan tudnak használni a pénzügyeik kezelésére. A fejlesztés során tapasztalt nehézségek ellenére sikerült jelentősen bővíteni a technológiai ismereteimet, és számos új képességet sajátítottam el, amelyek a jövőben is hasznosak lesznek számomra. A megszerzett tapasztalatokat és tudást reményeim szerint a jövőbeni projektek során is kamatoztatni tudom majd.

A diplomamunkám során szerzett tudás és tapasztalat nemcsak szakmai fejlődésemet támogatta, hanem magabiztosságot is adott a hasonló, jövőbeni kihívások kezeléséhez. Biztos vagyok benne, hogy ezek a tapasztalatok és ismeretek segítenek majd abban, hogy a jövőben még sikeresebb projekteket valósítsak meg a keresztplatformos fejlesztések területén, és további funkciókat valósítsak meg a pénzügyi alkalmazásomhoz.

## Irodalomjegyzék

- [1] Maximilian Schwarzmüller - <https://www.udemy.com/course/react-native-the-practical-guide/learn/lecture/31404530?start=120#reviews> - (Utolsó megtekintés: 2024. 05. 15.)
- [2] MoneyCoach hivatalos weboldal - <https://moneycoach.ai> - (Utolsó megtekintés: 2024. 05. 15.)
- [3] MoneyControll Spending Tracker elérhetősége - <https://apps.apple.com/us/app/moneycontrol-spending-tracker/id465909912> - (Utolsó megtekintés: 2024. 05. 15.)
- [4] WalletApp by budgetbackers hivatalos oldala - <https://budgetbakers.com> - (Utolsó megtekintés: 2024. 05. 15.)
- [5] Dime elérhetősége - <https://apps.apple.com/sg/app/dime-budget-expense-tracker/id1635280255> - (Utolsó megtekintés: 2024. 05. 15.)
- [6] Monefy hivatalos weboldala - <https://monefy.me> - (Utolsó megtekintés: 2024. 05. 15.)
- [7] React Native hivatalos weboldala - <https://reactnative.dev> - (Utolsó megtekintés: 2024. 05. 15.)
- [8] React Native architektúra - <https://blog.stackademic.com/react-native-vs-native-which-one-is-the-better-option-for-your-mobile-app-development-project-3b134571f70> - (Utolsó megtekintés: 2024. 05. 15.)
- [9] React hivatalos weboldala - <https://react.dev> - (Utolsó megtekintés: 2024. 05. 15.)
- [10] Typescript dokumentációja - <https://www.typescriptlang.org> - (Utolsó megtekintés: 2024. 05. 15.)
- [11] Expo hivatalos weboldala - <https://expo.dev> - (Utolsó megtekintés: 2024. 05. 15.)
- [12] Firebase hivatalos weboldala - <https://firebase.google.com> - (Utolsó megtekintés: 2024. 05. 15.)
- [13] Firebase SDK dokumentációja - <https://firebase.google.com/docs/firestore/client/libraries> - (Utolsó megtekintés: 2024. 05. 15.)
- [14] Firestore weboldala - <https://firebase.google.com/docs/firestore> - (Utolsó megtekintés: 2024. 05. 15.)
- [15] Az alkalmazás github repositoryja -  
[https://github.com/potyeszmate/Potyesz\\_Mate\\_Szakdolgozat\\_Credo/tree/main](https://github.com/potyeszmate/Potyesz_Mate_Szakdolgozat_Credo/tree/main) - (Utolsó megtekintés: 2024. 05. 15.)
- [16] Stripe hivatalos weboldala - <https://stripe.com/en-hu> - (Utolsó megtekintés: 2024. 05. 15.)
- [17] OpenAi fejlesztői dokumentáció - <https://platform.openai.com/docs/overview> - (Utolsó megtekintés: 2024. 05. 15.)
- [18] CoinMarketCap API dokumentációja - <https://coinmarketcap.com/api/documentation/v1/> - (Utolsó megtekintés: 2024. 05. 15.)
- [19] ExchangeRate API dokumentáció - <https://www.exchangerate-api.com/docs/overview> - (Utolsó megtekintés: 2024. 05. 15.)
- [20] Polygon.io API dokumentációja - <https://polygon.io/docs/stocks> - (Utolsó megtekintés: 2024. 05. 15.)
- [21] Twelvedata API dokumentációja - <https://twelvedata.com/docs#getting-started> - (Utolsó megtekintés: 2024. 05. 15.)
- [22] React Native image picker dokumentációja - <https://www.npmjs.com/package/react-native-image-picker> - (Utolsó megtekintés: 2024. 05. 15.)
- [23] Rohit Kumar - Stripe Payment Integration in React-Native -Getting Started - [https://medium.com/@rohit.kumar\\_54330/stripe-payment-integration-in-react-native-getting-started-7a31cd6101f4](https://medium.com/@rohit.kumar_54330/stripe-payment-integration-in-react-native-getting-started-7a31cd6101f4) - (Utolsó megtekintés: 2024. 05. 15.)
- [24] Ngrok dokumentációja - <https://ngrok.com/docs> - (Utolsó megtekintés: 2024. 05. 15.)
- [25] React Native Chartkit dokumentációja - <https://www.npmjs.com/package/react-native-chart-kit> - (Utolsó megtekintés: 2024. 05. 15.)
- [26] The emergence of Large Language Models (LLMs) - <https://thelowdown.momentum.asia/the-emergence-of-large-language-models-llms/> - (Utolsó megtekintés: 2024. 05. 15.)
- [27] Sunil Ramlochan - What is Prompt Engineering? - <https://promptengineering.org/what-is-prompt-engineering/> - (Utolsó megtekintés: 2024. 05. 15.)

## **Nyilatkozat**

Alulírott Potyesz Máté programtervező informatikus MSc szakos hallgató, kijelentem, hogy a dolgozatomat a Szegedi Tudományegyetem, Informatikai Intézet Szoftverfejlesztés Tanszékén készítettem, programtervező informatikus MSc 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 a diplomamunkámat a Szegedi Tudományegyetem Diplomamunka Repozitóriumában tárolja.

2024. 05. 15.

*Potyesz Máté*