FAKULTA INFORMAČNÍCH TECHNOLOGIÍ ČVUT V PRAZE

Zadání diplomové práce

Název:	Mobilná aplikácia pre sledovanie financií
Student:	Bc. Natália Pohanková
Vedoucí:	Ing. Igor Rosocha
Studijní program:	Informatika
Obor / specializace:	Softwarové inženýrství
Katedra:	Katedra softwarového inženýrství
Platnost zadání:	do konce letního semestru 2024/2025

Pokyny pro vypracování

Cieľom diplomovej práce je návrh, implementácia a otestovanie iOS aplikácie, ktorá umožní používateľom udržiavať si prehľad o ich financiách. Aplikácia poskytne funkcie na zaznamenávanie mesačných príjmov a výdavkov, stanovovanie dlhodobých a krátkodobých finančných cieľov a sledovanie ich vývoja. Zároveň bude používateľovi umožnené vizualizovať si mesačné prehľady, nastaviť si notifikácie na rôzne udalosti a synchronizovať dáta na viacerých zariadeniach prostredníctvom iCloud.

Postupujte v týchto krokoch:
1. Popíšte riešený problém a porovnajte konkurenčné aplikácie na trhu, ktoré sa zaoberajú rovnakou problematikou.
2. Identifikujte a zdokumentujte kľúčové funkcionality a cieľového používateľa aplikácie.
3. Vytvorte návrh používateľského rozhrania a zvoľte vhodnú architektúru.
4. Po konzultácií s vedúcim práce implementujte navrhnutú aplikáciu pre operačný systém iOS.
5. Výslednú aplikáciu dôkladne otestujte.
6. Zhrňte výsledok práce, identifikujte jej prínos a navrhnite budúce vylepšenia aplikácie.

Elektronicky schválil/a Ing. Michal Valenta, Ph.D. dne 6. prosince 2023 v Praze.

FAKULTA
INFORMAČNÍCH
TECHNOLOGIÍ
ČVUT V PRAZE

Diplomová práca

Mobilná aplikácia pre sledovanie financií

Bc. Natália Pohanková

Katedra softwarového inženýrství
Vedúci práce: Ing. Igor Rosocha

6. mája 2024

Poďakovanie

Ďakujem vedúcemu diplomovej práce Ing. Igorovi Rosochovi za cenné rady, priateľský prístup a všetky pripomienky a usmernenia pri vypracovávaní tejto práce. Ďakujem mojej rodine a blízkym za podporu a pomoc počas celej doby môjho štúdia. Osobitná vďaka patrí Kláre a Michalovi, za ich pevné nervy, oporu a povzbudivé slová.

Vyhlásenie

Vyhlasujem, že som predloženú prácu vypracovala samostatne a že som uviedla všetky informačné zdroje v súlade s Metodickým pokynom o etickej príprave vysokoškolských záverečných prác.

Beriem na vedomie, že sa na moju prácu vzťahujú práva a povinnosti vyplývajúce zo zákona č. 121/2000 Sb., autorského zákona, v znení neskorších predpisov, a skutočnosť, že České vysoké učení technické v Praze má právo na uzavrenie licenčnej zmluvy o použití tejto práce ako školského diela podľa § 60 odst. 1 autorského zákona.

V Prahe dňa 6. mája 2024 Natália Pohanková

Odkaz na túto prácu

Pohanková, Natália. *Mobilná aplikácia pre sledovanie financií.* Diplomová práca. Praha: České vysoké učení technické v Praze, Fakulta informačních technologií, 2024.

Abstrakt

V súčasnom svete sa mnohí jednotlivci stretávajú s výzvami správy osobných financií a často sa ocitajú v situáciách, ktoré by mohli byť riešené efektívnejšie s dostatočnou znalosťou a nástrojmi. Táto diplomová práca sa zaoberá návrhom a implementáciou mobilnej aplikácie pre iOS slúžiacej na správu osobných financií. Cieľom práce bolo vytvoriť aplikáciu, ktorá umožní používateľom systematicky zaznamenávať mesačné príjmy a výdavky, vytvárať grafové prehľady a vytyčovať finančné ciele. Poskytuje aj funkcionality spojené s importovaním transakcií pomocou bankovej API služby, či exportovanie do súboru CSV. Práca detailne popisuje fázu analýzy, návrhu, implementácie a testovania aplikácie.

Kľúčové slová Swift, iOS, mobilná aplikácia, rozpočet, financie, bankové API

Abstract

In today's world, many individuals face challenges in managing their personal finances and often find themselves in situations that could be handled more effectively with sufficient knowledge and tools. This master's thesis deals with the design and implementation of a mobile application for iOS aimed at personal finance management. The aim of the thesis was to create an application that allows users to systematically record their monthly incomes and expenses, create graphical overviews and set financial goals. It also provides functionalities related to importing transactions using a bank API service and exporting to a CSV file. The thesis elaborates on the phases of analysis, design, implementation, and testing of the application.

Keywords Swift, iOS, mobile application, budget, finance, bank API

Obsah

Zoznam obrázkov

Zoznam tabuliek

Úvod

Finančná gramotnosť a efektívne nakladanie s financiami sú len príkladom tém, ktoré hýbu stránkami ekonomických časopisov a sú predmetom odborných diskusií ale aj bežných rozhovorov v práci, škole a v rodine. Je nepochybne existenčne dôležité, mať aspoň základné porozumenie problematiky týkajúcej sa šetrenia, plánovania krátkodobých a dlhodobých výdavkov, alebo pripravenia sa na neočakávané udalosti, a s tým spojené finančné náklady, formou dostatočnej peňažnej rezervy. Zároveň, sa čoraz viac aj medzi bežnými ľuďmi, a nielen medzi finančníkmi, hovorí o potrebách včasného investovania, o akciách, dlhopisoch alebo o kryptomenách a rizikách, ktoré sú s tým spojené.

Stavebným základom všetkého je ale byť schopný porozumieť a mať prehľad o svojej finančnej realite, od ktorej sa potom odvíjajú ďalšie možnosti. Pokiaľ človek nevie, aké množstvo peňazí mu na účet mesačne prichádza a aká čiastka z účtu odchádza, nie je schopný urobiť informované rozhodnutie o tom, koľko si môže dovoliť investovať alebo odložiť tak, aby neohrozil svoju finančnú stabilitu.

Väčšina veľkých bánk využíva internetové bankovníctvo, ale v dnešnej dobe ho má už takmer každý majiteľ účtu vo svojom mobilnom telefóne vo forme mobilnej aplikácie. Z potreby vedenia zápisníkov a mesačných kalkulácií vo fyzickej forme na papier sa už takmer úplne upustilo a poskytovanie možností ako si zaznamenávať informácie o svojich financiách do mobilného telefónu alebo tabletu je prirodzené.

Tému diplomovej práce považujem za aktuálnu a zaujímavú, v minulosti som osobne vyskúšala mnohé aplikácie s podobnou tématikou a nadobudla som pocit, že žiadna nemá všetky vlastnosti a funkcionality, ktoré by som ja ako používateľ potrebovala a očakávala.

Cieľ práce

Hlavným cieľom tejto diplomovej práce je navrhnúť a implementovať mobilnú aplikáciu, ktorá umožní svojím používateľom zaznamenávať si mesačné príjmy a výdavky, a tým si vytvárať prehľad o stave ich osobných financií.

V práci budem prechádzať všetkými hlavnými fázami vývoja softvérového produktu. Začnem analýzou, ktorej cieľom je identifikovať funkčné a nefunkčné požiadavky, popísať proces definovania persony a prípadov použitia. Následne sa budem venovať porovnaniu aplikácií, ktoré sa dotýkajú rovnakej problematiky a sú už používateľom dostupné na stiahnutie. Zhodnotím ich klady a zápory a prípadne identifikujem priestor na zlepšenie. V závere analýzy popíšem možnosti využitia služieb, ktoré banky technologickým firmám poskytujú.

Cieľom návrhovej časti je detailne popísať proces tvorby grafického rozhrania mobilnej aplikácie a navrhnúť riešenie pre ukladanie dát. Taktiež sa zameriam na porovnanie a charakterizovanie rôznych architektonických prístupov a technológií, ktoré budú použité v implementácií. V rámci kapitoly o implementácie následne bližšie priblížim implementačné aspekty, ako napríklad realizáciu vybranej architektúry a aplikovanie konkrétnych knižníc.

Záverom práce sa budem sústrediť na testovanie vytvorenej aplikácie, popíšem ako prebiehalo používateľské testovanie a zhrniem jeho výsledky.

Kapitola 1

Analýza

V tejto časti práce sa detailne venujem analýze aplikačných požiadaviek, definovaniu persony, vďaka ktorej bude jednoduchšie predstaviť si cieľovú skupinu používateľov, pre ktorú je aplikácia určená. Nasleduje prieskum trhu a analýza konkurenčných aplikácií. Kapitolu zakončím popisom niektorých termínov súvisiacich s financiami a bankovníctvom, ktoré budú potrebné pri tvorbe a neskôr používaní aplikácie. Bližšie sa zameriam najmä na koncept tzv. *open banking* a na súvislosti s ním spojené.

1.1 Požiadavky

Precízne definovanie aplikačných požiadaviek predstavuje neoddeliteľný základ každého kvalitného a úspešného softvérového produktu. Je nevyhnutné stanoviť a presne vymedziť, akými funkcionalitami daný softvér disponuje.

Spravidla ide o popis toho, čo produkt robí a aké funkcie používateľom poskytuje – to je charakteristické pre funkčné požiadavky. Naopak, to, ako to systém robí, čiže vlastnosti a špecifiká systému, napríklad výkonnosť, bezpečnosť či použiteľnosť, popisujú nefunkčné požiadavky[1].

Táto časť dokumentácie zohráva kľúčovú úlohu, keďže predstavuje detailný manuál, z ktorého vychádza vývojový a dizajnový tím. V nasledujúcej časti analyzujem nároky súvisiace s užívateľským rozhraním a funkcionalitami pre správu finančných údajov.

1.1.1 Funkčné požiadavky

Mesačný rozpočet

- V aplikácií bude možné vytvoriť mesačný rozpočet definovaním maximálnej sumy určenej na výdavkové transakcie. (F1)
- Pri zadávaní príjmových alebo výdavkových transakcií používateľ zadá popis, sumu a vyberie kategóriu. (F2)

- Systém umožní používateľovi manuálne zaraďovať výdavky (transakcie) do príslušných kategórií. (F3)
- Aplikácia transakcie zobrazí ako zoznam. V tomto zozname bude môcť používateľ filtrovať podľa typu transakcie a vyhľadávať podľa popisu. (F4)
- Aplikácia umožní mesačný rozpočet meniť – upravovať maximálnu sumu výdavkov. (F5)
- Používatelia si v aplikácii budú môcť synchronizovať svoje transakcie z podporovaného internetového bankovníctva za posledných 90 dní a zaradiť ich do mesačných štatistík. (F6)

Mesačné ciele

- Používateľ bude môcť definovať nový finančný cieľ s názvom, sumou a typom cieľa. (F7)
- Aplikácia poskytne možnosť sledovať aktuálny stav finančného cieľa pomocou grafického zobrazenia (progress bar, ukazujúci percentuálny pokrok). (F8)
- Používateľ si do systému bude môcť zaznamenať sumu, ktorú určil ako prostriedok k splneniu svojho vytýčeného cieľa. (F9)
- Aplikácia poskytne možnosť filtrovať finančné ciele podľa typu (mesačné, ročné). (F10)
- Používateľ bude mať možnosť cieľ zmazať. (F11)

Upozornenia

- Používateľ si bude môcť zapnúť, prípadne vypnúť dostávanie upozornení pre udalosti – týždenný prehľad stavu rozpočtu alebo pripomienky na pravidelné zaznamenávanie transakcií do aplikácie. (F12)

Nastavenia

- Aplikácia bude poskytovať jednoduchý návod, ktorý bude obsahovať zoznam hlavných funkcionalít. (F13)
- Používateľ si bude môcť vybrať a zmeniť menu, v ktorej sa budú zaznamenávať finančné hodnoty. (F14)

Vizualizácie

- Súčasťou aplikácie bude vytváranie grafových prehľadov z transakcií podľa časového obdobia. (F15)
- Zoznam transakcií bude môcť používateľ exportovať do súboru s formátom CSV (Comma-separated values). (F16)

1.1.2 Nefunkčné požiadavky

- Aplikácia bude určená pre zariadenia s operačným systémom iOS. (NF1)
- Bude dostupná možnosť synchronizácie dát a zálohovanie prostredníctvom iCloud na zaistenie bezpečnosti dát používateľa. (NF2)
- V aplikácií budú použité štandardné prvky a gestá typické pre iOS platformu. (NF3)
- Podporované jazyky aplikácie budú angličtina a slovenčina. (NF4)
- Aplikácia bude poskytovať podporu pre zobrazenie v tmavom režime. (NF5)

1.2 Persony a prípady použitia

Z pohľadu marketingu alebo návrhu a vývoja akéhokoľvek produktu či služby, je dôležité vedieť odpovedať najmä na tri otázky: Čo vytvárame? Pre koho to vytvárame? a Aký problém riešime? Práve koncept persony dáva odpoveď na druhú otázku. Persona je detailný opis fiktívnej osoby, pre ktorú je produkt, ktorý vytvárame, určený. Ide o fiktívneho predstaviteľa tzv. cieľovej skupiny. Vďaka podrobnému definovaniu tohto predstaviteľa vieme lepšie pochopiť potreby, ciele a správanie skutočných ľudí, ktorí budú využívať náš produkt alebo službu. [2]

Vytváranie persony, konkrétne pre mobilnú aplikáciu, je komplexný proces, ktorému by mal predchádzať kvalitatívny a kvantitatívny používateľský prieskum. Na základe dotazníkov a aj osobných rozhovorov, cieľom ktorých je získať čo najviac informácií o potencionálnych používateľoch, čo majú spoločné, prípadne aké sú medzi nimi rozdiely, budeme následne schopní vytvoriť hypotézu, ktorá by mala reflektovať základné potreby a charakteristiky nášho cieľového zákazníka. Na základe hypotézy formujeme spomínaný detailný popis, ktorý by mal obsahovať fiktívne meno, vek a popis záľub, typického dňa a krátkej histórie [3]. Od korektného popisu persony sa potom odvíja mnoho aspektov – návrh používateľského rozhrania, použité farby, písmo, platforma a v neposlednom rade samotný obsah.

V rámci konceptu persony sa stretávame aj s pojmom tzv. „antipersony“. Antipersona reflektuje používateľskú skupinu, ktorá s našou aplikáciou do

styku neprichádza a aplikácia nie je tvorená s ohľadom na preferencie a ciele tejto skupiny. Definovaním antipersony často optimalizujeme využitie zdrojov a znižujeme riziko neúspechu. Identifikácia tých, ktorí nie sú záujemcami o aplikáciu prispieva k efektívnemu cieleniu na skutočných potenciálnych zákazníkov. [4]

1.2.1 Definovanie persony

Persona A - ideálny používateľ

- meno: Petra
- vek: 29 rokov
- záľuby: cestovanie, nakupovanie, festivaly, gastronómia, chodenie do kina

Typický deň

Petra vstáva ráno do práce, doma neraňajkuje, keďže si cestou kupuje raňajky v miestnej pekárni. Počas pracovného dňa sa vo voľnej chvíli rada pozrie na rôzne stránky s oblečením, knihami alebo letenkami a takmer neustále má plný nákupný košík naprieč rôznymi webovými stránkami. Na obed ide do miestnej reštaurácie na denné menu. Po práci sa zastaví v supermarkete, kde málokedy nasleduje svoj nákupný zoznam a nechá sa zlákať akciami. Takmer každý večer trávi spoločensky - s priateľom v kine, alebo na večeri a nápoji v miestnom vychýrenom podniku s kamarátkami, kde diskutujú na akú dovolenku alebo víkendový výlet sa najbližšie spolu vyberú.

Krátka história

Petra pochádza z pomerne stabilného finančného prostredia, ale často má problémy s utrácaním, čo vedie k finančným obmedzeniam koncom mesiaca. Vie, že si platí predplatné mnohých streamovacích služieb a nemá presný prehľad, koľko mesačne minie peňazí na veci, ktoré až tak nevyužíva. V minulosti sa jej podarila väčšia investícia a kúpila byt, ktorý aktuálne prenajíma a tým spláca hypotéku. Rada by si časom našetrila na auto, ale nevie koľko si musí odkladať a celkovo jej chýba detailnejší prehľad o jej financiách a potrebuje nástroj, ktorý jej pomôže lepšie spravovať a plánovať jej výdavky.

Persona B - antipersona

- meno: Jozef
- vek: 60 rokov
- záľuby: šport, trávenie času s rodinou, práca v záhrade, turistika, čítanie kníh

Typický deň

Pán Jozef každý deň odštartuje šálkou kávy a čítaním jeho obľúbených novín. Najradšej má sekciu o športe. Po raňajkách, ktoré mu pripraví manželka, odchádza autom do práce. Pracuje ako učiteľ telesnej výchovy a geografie na miestnej strednej škole. Každý deň v priemere odučí 4 vyučovacie hodiny. Telesnú výchovu učí radšej, pretože za jeho mladých čias hrával futbal súťažne. Následne ide na obed do školskej jedálne. Poobede sa venuje príprave na ďalší deň, chodí na porady alebo známkuje písomky. V utorky a štvrtky vedie krúžok volejbalu. Po práci si rád ide zaplávať alebo zabehať, aby sa udržal v dobrej fyzickej kondícií. S manželkou v lete pracujú v záhrade alebo občas zájdu do divadla.

Krátka história

Pán Jozef je stará škola, býva v dome po rodičoch a o hypotéke nemusel ani uvažovať. Neinvestoval ani v mladosti a nevidí dôvod prečo by to mal robiť teraz. Má dobrý prehľad o rodinných financiách, manželkin a aj jeho príjem je každý mesiac stabilný a za tie roky spoločného života už vedia ako s financiami narábať. Na dôchodok si obaja odkladajú určitú čiastku z platu, tak ako aj na letnú dovolenku. Obe deti má už po vysokej škole a finančne nezávislé. Nepociťuje potrebu a ani nevidí zmysel svoje financie viacej monitorovať a nemá ani žiadne finančné ciele, ktoré by chcel dosiahnuť.

1.2.2 Prípady použitia - Use Cases - UC

Aby boli funkcionality produktu ľahšie uchopiteľné a zrozumiteľné pre všetky zainteresované subjekty a nielen pre analytikov, je možné využiť techniku prípadov použitia *(anglicky use cases)*, ktoré slúžia na opísanie a pochopenie funkcií alebo akcií, ktoré systém musí vykonať na základe interakcií s externými aktérmi (používateľmi, inými systémami atď.). Ich použitie je typické vo fáze analýzy požiadaviek a následne sú dôležité aj počas návrhu softvérových systémov.

Každý aktér ma na začiatku UC cieľ, ku ktorému musí na základe konkrétnych krokov dôjsť. Pre prípady použitia je typické uviezť názov, aktéra a hlavný scenár. Je možné pripojiť aj alternatívne scenáre, prípadne možné chybové stavy. [5]

- **UC1** – Nastavenie mesačného rozpočtu (F1)
- **UC2** – Zvýšenie mesačného rozpočtu (F5)
- **UC3** – Pridanie výdavkovej transakcie (F2, F3)
- **UC4** – Zmena údajov existujúcej transakcie (F3)

- **UC5** – Použitie filtra na zobrazenie príjmových transakcií a nájdenie konkrétnej platby (F4)
- **UC6** – Zapnutie notifikácií o rozpočte (F12)
- **UC7** – Zobrazenie mesačných finančných cieľov (F10, F8)
- **UC8** – Pridanie nového finančného cieľa (F7)
- **UC9** – Zmazanie finančného cieľa (F11, F8)
- **UC10** – Pridanie peňazí v rámci finančného cieľa (F9)
- **UC11** – Zmena meny v aplikácií (F14)
- **UC12** – Zobrazenie mesačného grafu a prehľad mesačných transakcií podľa kategórií (F15)
- **UC13** – Exportovanie dát do CSV súboru (F16)
- **UC14** – Zobrazenie návodu (F13)
- **UC15** – Pridanie tokenu z podporovaného bankovníctva (F6)
- **UC16** – Synchronizovanie platieb s podporovaným bankovníctvom (F6)

UC1 – Nastavenie mesačného rozpočtu

Používateľ na obrazovke *Rozpočet* klikne na text *Nastav mesačný rozpočet* v strede obrazovky. Zo spodnej časti obrazovky sa zobrazí list, kde používateľ zadá ručne čiastku alebo vyberie pomocou posúvača. V prípade, že chce, aby bol rovnaký rozpočet nastavený pre každý mesiac, pomocou prepínacieho tlačítka vyberie možnosť zapnutia. Ak si to rozmyslí, pohybom prsta z vrchnej časti listu, smerom k dolnej časti obrazovky, list zavrie. Ak chce rozpočet uložiť, učiní tak kliknutím na tlačidlo *Potvrdiť*. Následne aplikácia zobrazí obrazovku s kruhovým indikátorom pokroku a sumou, ktorú si používateľ nastavil.

UC2 – Úprava mesačného rozpočtu

V prípade, že chce používateľ znížiť alebo zvýšiť mesačný rozpočet, učiní tak na obrazovke *Rozpočet*, klikne na *Klepnutím upravte* a zo spodnej časti obrazovky sa zobrazí list, kde používateľ upraví čiastku alebo použije posúvač. Ak si úpravu rozmyslí, pohybom prsta z hornej časti listu, smerom k dolnej časti obrazovky, list zavrie. Pre uloženie klikne na tlačidlo *Potvrdiť*. Následne aplikácia zobrazí obrazovku s kruhovým indikátorom a upravenou maximálnou sumou rozpočtu.

UC3 – Pridanie výdavkovej transakcie

Alternatíva 1

Na obrazovke *Prehľad* používateľ klikne na tlačidlo *Pridať novú transakciu* v spodnej časti obrazovky. Na obrazovke sa zobrazí list, kde používateľ vyberie typ transakcie, napíše popis, prípadne vyberie deň, kedy sa daná transakcia uskutočnila. Má možnosť vybrať kategóriu transakcie a to kliknutím na príslušnú ikonu. V neposlednom rade zadá čiastku. V prípade, že si vytváranie transakcie rozmyslí, klikne na symbol krížiku v pravom hornom rohu a tým list zavrie. Ak chce transakciu uložiť, učiní tak kliknutím na tlačidlo *Potvrdiť*.

Alternatíva 2

Transakciu je možné pridať aj z obrazovky *Rozpočet*, kliknutím na tlačidlo so symbolom „+“ v časti *Transakcie*. Postup potom pokračuje rovnako ako v Alternatíva 1.

Alternatíva 3

Na obrazovkách *Prehľad* a *Rozpočet* v dolnej časti sa kliknutím na *Zobraziť všetky transakcie* používateľ dostane na obrazovku so všetkými transakciami, kde v pravom dolnom rohu kliknutím na ikonu „+“ otvorí list a proces zadávania transakcie potom pokračuje rovnako ako v predchádzajúcich alternatívach.

UC4 – Zmena údajov existujúcej transakcie

Kliknutím na konkrétnu transakciu na obrazovke *Transakcie* používateľ otvorí list, ktorý má vyplnené údaje – názov, typ, kategóriu, dátum vytvorenia transakcie a sumu podľa vybranej transakcie. Všetky polia sú editovateľné. Používateľ potvrdí zmeny kliknutím na tlačidlo *Potvrdiť* alebo zmeny zahodí kliknutím na symbol krížika v pravom hornom rohu zobrazeného listu.

UC5 – Použitie filtra na zobrazenie príjmových transakcií a nájdenie konkrétnej platby

Na obrazovkách *Prehľad* a *Rozpočet* v dolnej časti sa kliknutím na *Zobraziť všetky transakcie* používateľ dostane na príslušnú obrazovku so všetkými transakciami. V hornej časti kliknutím na *Príjmy* vyberie filter, ktorý zobrazí len príjmové transakcie. Následne do vyhľadávacieho poľa zadá kľúčové slovo, podľa ktorého chce vyhľadávať. Na obrazovke aplikácie ostanú len transakcie, ktoré v názve obsahujú zadané slovo, alebo obrazovka ostane prázdna.

UC6 – Zapnutie notifikácií o rozpočte

Kliknutím na ikonu budíka na obrazovke *Prehľad*, sa používateľ dostane na obrazovku *Notifikácie*. Ak notifikácie nemá ešte povolené, najprv tak musí učiniť a to pomocou prepínacieho tlačidla. Pri prvom zapnutí sa na obrazovke zobrazí systémové dialógové okno, kde používateľ musí potvrdiť alebo odmietnuť zasielanie notifikácií. Po potvrdení aplikácia zobrazí na obrazovke zoznam notifikácií, ktoré poskytuje. Konkrétnu notifikáciu zapne kliknutím na ikonu zo zoznamu. Zapnutie signalizuje aj symbol začiarknutia (✓) v pravom dolnom rohu danej notifikácie.

UC7 – Zobrazenie mesačných finančných cieľov

Používateľ prejde na obrazovku *Ciele*. V hornej časti obrazovky klikne na *Mesačné ciele*. Na obrazovke sa mu zobrazujú len ciele, ktoré sú mesačné. Kliknutím na záložku *Ročné ciele*, môže medzi nimi prepínať.

UC8 – Pridanie nového finančného cieľa

Používateľ prejde na obrazovku *Ciele*. Po kliknutí na tlačidlo so symbolom „+“ a textom *Nastav nový cieľ* sa na obrazovke zobrazí list, kde používateľ vyplní popis cieľa, vyberie typ – mesačný cieľ alebo ročný a zadá sumu, ktorá sa k cieľu viaže. V prípade, že si vytváranie nového finančného cieľa rozmyslí, klikne na symbol krížiku v pravom hornom rohu a list tak zavrie. Ak chce cieľ uložiť, učiní tak kliknutím na tlačidlo *Potvrdiť*. Následne aplikácia zobrazí list všetkých existujúcich cieľov, vrátane novo pridaného.

UC9 – Zmazanie finančného cieľa

Používateľ prejde na obrazovku *Ciele*. Kliknutím a dlhším podržaním (aspoň 0,5 sekundy) na ikonu cieľa v zozname cieľov, vyberie ten, ktorý chce zmazať. Na obrazovke sa objaví vyskakujúce okno, ktoré overí, že používateľ chce naozaj daný záznam zmazať. Kliknutím na *Potvrdiť* dôjde k úspešnému zmazaniu cieľa a používateľovi sa zobrazí obrazovka so všetkými existujúcimi finančnými cieľmi, absentuje len ten, ktorý bol zmazaný.

UC10 – Pridanie peňazí v rámci plnenia finančného cieľa

Používateľ prejde na obrazovku *Ciele*. Kliknutím na ikonu cieľa vyberie ten, pri ktorom si chce zaznačiť svoj pokrok. V dolnej časti obrazovky sa mu zobrazí list, na ktorom pomocou posúvača alebo zadaním sumy do textového poľa nastaví sumu, ktorú si chce zaznamenať. Je možné pridávať len sumy väčšie ako 0. V prípade, že si celý proces rozmyslí, klikne na tlačidlo krížiku, v pravom hornom rohu prezentovaného listu alebo potiahne z hornej časti listu smerom k dolnej časti obrazovky. Ak chce pokračovať, vybranú sumu potvrdí kliknutím

na tlačidlo *Potvrdiť*. Následne sa zobrazí obrazovka s finančnými cieľmi, kde je k vybranému cieľu pripísaná hodnota, ktorú zadal.

UC11 – Zmena meny v aplikácií

Používateľ sa dostane na obrazovku *Nastavenia* z obrazovky *Prehľad*, ktorá má ikonu ozubeného kolieska. Následne klikne na menu s menami, kde sa mu zobrazia podporované meny. Kliknutím vyberie menu podľa svojej preferencie. Na obrazovke sa objaví vyskakujúce okno, ktoré overí, že používateľ chce naozaj zmeniť aktuálnu menu všetkých transakcií v aplikácií. Kliknutím na *Nie, zrušiť* červenej farby v dialógovom okne zostane mena aplikácie nezmenená a kliknutím na *Áno, zmeniť* sa mena zmení. V druhom popisovanom prípade používateľ po potvrdení v aplikácií vidí, že sa vo všetkých textoch s peňažnou hodnotou prepísal symbol meny.

UC12 – Zobrazenie mesačného grafu a prehľad mesačných transakcií podľa kategórií

Na obrazovke *Transakcie*, používateľ v pravom hornom rohu navigačného panelu klikne na ikonu pripomínajúcu stĺpcový diagram, čím prejde na obrazovku *Štatistiky*, kde si v hornej časti vyberie časové obdobie, ktoré ho zaujíma – denný, týždenný, mesačný alebo ročný súhrnný prehľad. Aplikácia podľa výberu zobrazí prislúchajúci stĺpcový graf. V grafe je farebne rozlíšený súčet príjmových (zelená) a výdavkových (červená) transakcií. Pod grafom sa nachádzajú tieto súčty aj v textovej podobe. Pod textom môže používateľ vidieť rozdelenie transakcií podľa ich príslušnosti ku kategóriám za vybrané časové obdobie.

UC13 – Exportovanie dát do CSV súboru

Kliknutím na ikonu listu a šípky dohora na obrazovke *Štatistiky* v pravom hornom rohu navigačného panela sa používateľovi vytvorí súbor vo formáte CSV obsahujúci v riadkoch všetky transakcie v tvare *Názov; Dátum; Cena* oddelené symbolom bodkočiarky „;“. Zároveň sa na obrazovke otvorí aj tzv. *ShareSheet* operačného systému iOS, ktorý poskytuje možnosti na uloženie alebo zdieľanie vytvoreného súboru.

UC14 – Zobrazenie návodu

Používateľ po prechode na obrazovku *Nastavenia*, vyberie z menu *Návod*. To zobrazí obrazovku, na ktorej sa používateľ oboznámi so základnými funkcionalitami aplikácie. V dolnej časti obrazovky je tlačidlo *Ďalšia strana*, ktorým prejde na ďalšiu z piatich obrazoviek návodu. V pravom hornom rohu stránky môže návod pomocou symbolu krížika kedykoľvek zavrieť.

UC15 – Pridanie tokenu z podporovaného bankovníctva

Alternatíva 1

Kliknutím na tlačidlo *Importovať z banky* na obrazovke *Prehľad* sa používateľ dostane na obrazovku, kde do textového poľa môže zadať token vygenerovaný zo svojho internetového bankovníctva. Obrazovka *Importovanie transakcií* s textovým poľom sa zobrazí len v prípade, ak ide o prvé zadanie tokenu. V prípade, že používateľ už token predtým zadal, zobrazí sa informačná obrazovka *Importovanie transakcií* s údajom dátumu posledného importu transakcií a tlačidlom, ktorým môže zahájiť ďalší import.

Alternatíva 2

Na obrazovku s tokenom alebo dátumom posledného importu sa používateľ dostane aj z obrazovky *Transakcie* kliknutím na ikonu listu a šípky dodola v hornom navigačnom paneli v pravej časti obrazovky. Následný priebeh zostáva rovnaký ako v Alternatíva 1.

Alternatíva 3

Scenár so zadaním tokenu sa opakuje aj v prípade, že sa používateľovi na obrazovke o iniciovaní importu kliknutím na tlačidlo *Získaj transakcie z Fio banky* na obrazovke *Importovanie transakcií* zobrazí list oznamujúci zlyhanie importu v dôsledku chyby alebo neplatnosti tokenu. Aplikácia používateľa vyzve na opakované zadanie tokenu a po kliknutí na tlačidlo *Vlož nový token* sa zobrazí obrazovka s textovým poľom, kde ho môže zadať.

UC16 – Synchronizovanie platieb s podporovaným bankovníctvom

Alternatíva 1

Kliknutím na tlačidlo *Importovať z banky* na obrazovke *Prehľad* sa používateľ dostane na obrazovku *Importovanie transakcií* s dátumom posledného importu transakcií a tlačidlom, ktorým môže zahájiť ďalší import. Po kliknutí na tlačidlo sa buď zobrazí obrazovka signalizujúca úspešný import s textom a počtom novo importovaných transakcií, alebo informácia o tom, že žiadne nové transakcie od posledného importu v bankovom účte spojenom s tokenom zadaným do aplikácie neexistujú. Používateľ tieto obrazovky zavrie pomocou symbolu krížika v pravej hornej časti obrazovky.

V prípade, že nastane nezhoda mien transakcií v aplikácií a importovaných transakcií z banky, aplikácia zobrazí obrazovku s varovaním a možnosťou pokračovať, (čím sa zmení mena aplikácie na menu importovaných transakcií a všetky existujúce sa vymažú) alebo proces importu ukončiť.

	UC1	UC2	UC3	UC4	UC5	UC6	UC7	UC8	UC9	UC10	UC11	UC12	UC13	UC14	UC15	UC16
F1	✓															
F2			✓													
F3			✓	✓												
F4					✓											
F5		✓														
F6															✓	✓
F7								✓								
F8							✓		✓							
F9										✓						
F10							✓									
F11									✓							
F12						✓										
F13														✓		
F14											✓					
F15												✓				
F16													✓			

Tabuľka 1.1: Tabuľka popisujúca pokrytie funkčných požiadaviek pomocou UC.

Alternatíva 2

Na obrazovku s tokenom alebo dátumom posledného importu sa používateľ dostane aj z obrazovky *Transakcie* kliknutím na ikonu šípky dodola v hornom navigačnom paneli v pravej časti obrazovky. Následný priebeh zostáva rovnaký ako v Alternatíva 1.

V tabuľke 1.1 môžeme vidieť pokrytie funkčných požiadaviek pomocou UC. Každá požiadavka má prislúchajúci UC, pomocou ktorého je realizovateľná.

1.3 Prieskum trhu

Problematika, ktorá sa dotýka monitorovania osobných financií ľudí nie je novinkou a to reflektuje aj množstvo aktuálne dostupných produktov na trhu mobilných aplikácií. Nakoľko táto práca je zameraná na vytvorenie aplikácie pre zariadenia od firmy *Apple*, a teda s operačným systémom iOS, skúmala som najmä aplikácie v *AppStore*, čo je opäť aplikácia, ktorá slúži ako distribučná platforma, na ktorej si ľudia môžu zakúpiť alebo zadarmo stiahnuť digitálny softvér.

Vybrala som si 4 aplikácie dotýkajúce sa podobnej problematiky. Všetko sú to aplikácie, ktoré sú dostupné používateľom na stiahnutie zadarmo a nachádzajú sa vo výbere populárnych a často sťahovaných aplikácií v kategórií *Financie*.

V subjektívnom porovnávaní som sa sústredila na aspekty ako sú:

- **funkčnosť** – od aplikácie na monitorovanie financií som očakávala minimálne možnosť zaznamenávania príjmov a výdavkov, možnosť kategorizácie transakcií, filtrovanie, zobrazenie celkového prehľadu a možnosť zaznamenávania finančných cieľov.

- **intuitívnosť** – odpovedala som na otázky, či sú jednotlivé funkcionality ľahko a priamočiaro vykonateľné, či sú použité typické prvky, ktoré sú používateľom iOS zariadení známe, ale zaujímalo ma aj, či je prítomný návod, ktorý aplikáciu a jej použitie ľuďom priblíži a vysvetlí.

- **estetickosť** – to je výrazne subjektívna kategória, nakoľko každý považuje za estetické niečo iné, sústredila som sa preto najmä na použitie farieb, typov a veľkostí písma, či použité ikony zobrazovali to, na čo slúžili atp.

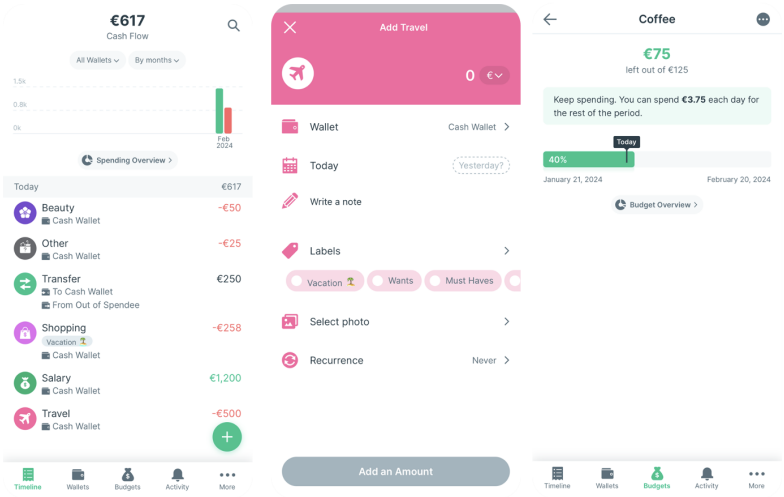
1.3.1 Spendee Money & Budget Planner

Aplikácia [6] hneď po spustení vyžaduje, aby sa používateľ prihlásil alebo zaregistroval. Prihlásenie je možné aj pomocou Google, Facebook alebo Apple účtu. Používateľovi je k dispozícií bezplatná verzia s obmedzeným množstvom funkcionalít. Z funkčného hľadiska je možné zadať transakciu, priradiť ju do kategórie, vybrať menu sumy a dátum transakcie, a dokonca aj pridať fotku (účtenky). Celkový prehľad je urobený dynamicky, je možné grafy meniť podľa vybranej kategórie a podľa toho, či sa jedná o príjem alebo výdavok. V aplikácií chýba možnosť vytýčiť si finančný cieľ a aktivity s tým spojené, ale je tu prítomný obdobný koncept, kedy si používateľ môže vytvoriť rozpočet, zadať mu dĺžku trvania, kategóriu a sumu, ktorú za daný čas môže minúť tak, aby vytýčený rozpočet neprekročil.

Aplikácia a jej používanie je intuitívne, obsahuje veľké množstvo funkcionalít a s tým súvisiacich prechodov na iné obrazovky, čo môže niekedy pôsobiť až rušivo. Absentuje návod pri prvom zapnutí aplikácie, ale aplikácia poskytuje možnosť obrátiť sa na zákaznícku podporu a na fórum s tipmi.

Oceňujem, ako aplikácia pracuje s farbami, každá kategória má priradenú unikátnu farbu a týmto spôsobom sú jednotlivé záznamy dobre rozlíšiteľné. K dispozícií je aj prepínanie medzi tmavým a svetlým režimom. Ikony ako peňaženka, banka, notifikácie odpovedajú použitiu v bežnom živote.

Ďalšie plusy aplikácie, ktorá je s rozpočtovaním a spravovaním financií na prvom mieste v *AppStore*, vidím v možnosti pridania si bankového účtu priamo do aplikácie, čo umožňuje, aby používateľ mal všetky svoje účty na jednom mieste. Tento koncept – *multibanking*, bližšie opíšem v ďalšej časti práce. Bohužiaľ, aj keď pochopiteľne, je toto jedna z funkcionalít, ktorá nie je prístupná v bezplatnej verzií. V aplikácií sa nachádza široké zastúpenie bánk, ktoré je možné do nej pridať. Zároveň je tu aj možnosť pridať krypto peňaženku. Používateľ môže filtrovať, vytvárať nové kategórie, prehľadávať transakcie, exportovať do CSV súboru a mnohé ďalšie. Niektoré obrazovky tejto aplikácie sú zobrazené na obrázku 1.1.



Obr. 1.1: Obrazovky aplikácie Spendee Money & Budget Planner

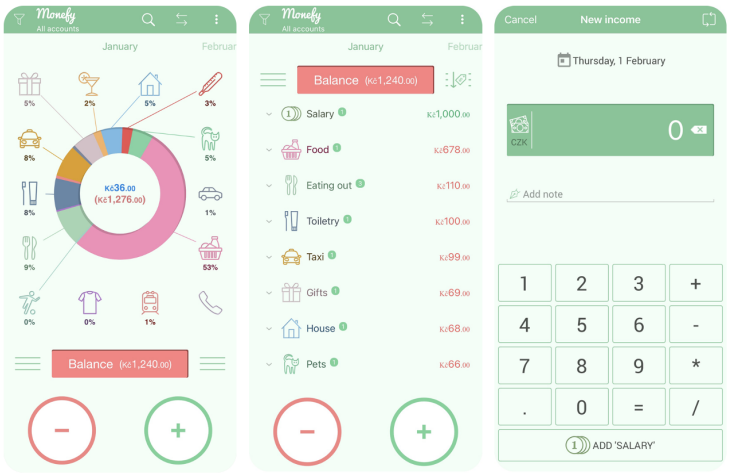
1.3.2 Monefy: Money Tracker

Druhá skúmaná aplikácia [7] je jednoduchšia a hlavný zreteľ sa kladie na rozpočtovanie a kruhový diagram, ktorý toto reflektuje. Používateľ si môže v bezplatnej verzii zaznamenávať príjmy a výdavky a ľubovoľne ich kategorizovať. Nové kategórie si používateľ môže vytvárať len v platenej verzii. Opäť absentuje možnosť vytýčenia finančných cieľov a ich sledovanie. Dominantou aplikácie je kruhový diagram, ktorým je graficky znázornené akú časť výdavkov, ktorá kategória predstavuje. Transakcie je možné prehľadávať, upravovať a radiť, ale nie filtrovať podľa kategórie. Podľa kategórií sú v zozname ale zhlukované čiže to čiastočne nahrádza chýbajúce filtrovanie.

Narozdiel od prvej aplikácie, v tomto prípade bola orientácia náročnejšia a použitie aplikácie nebolo tak intuitívne. Napríklad zobrazenie všetkých transakcií nebolo priamočiare, ako ani upravovanie už existujúcich záznamov. Pri prvom použití aplikácia ponúka vyskakujúce upozornenia a návestia, kde môže používateľ niečo ďalšie nastaviť alebo nájsť.

Prítomné sú aj piktogramy, ktoré zodpovedajú zaužívaným symbolom. Zapnutie tmavého režimu je funkcionalita platenej verzie.

Aplikácia poskytuje veľa možností, ako si ju môže používateľ prispôsobiť, ale pôsobí to miestami chaoticky. Dodatočné nastavenia zahŕňajú možnosť zvoliť prvý deň v týždni, menu, jazyk a či chce používateľ zapnúť ochranu heslom. Nachádza sa tu aj možnosť exportovať dáta, prípadne synchronizovať s Dropbox alebo Google Drive. Obrazovky tejto aplikácie zobrazuje obrázok 1.2.



Obr. 1.2: Obrazovky aplikácie Monefy: Money Tracker

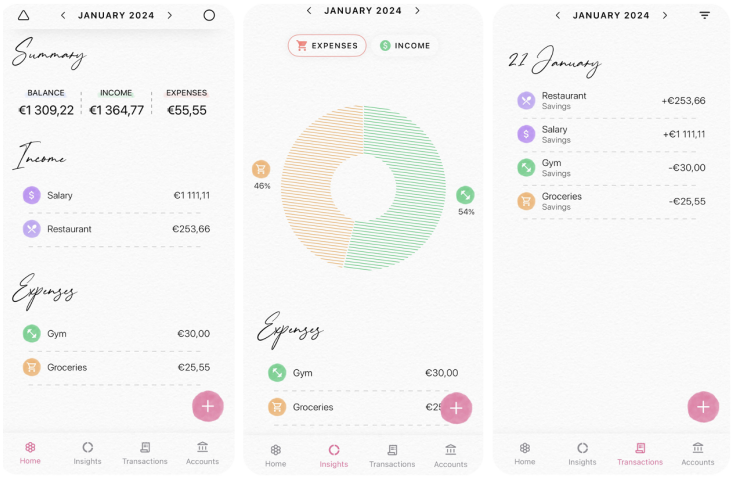
1.3.3 Fleur

Tretia aplikácia [8], na ktorú som sa zamerala v rámci prieskumu trhu, sa už na prvý pohľad odlišuje a sústredí na estetickosť a eleganciu. Používateľ má možnosť jednoducho vytvoriť transakciu, či už príjmovú alebo výdavkovú, priradiť jej kategóriu a aj nastaviť prípadné opakovanie. Toto je zaznamenávané do dvoch kruhových diagramov, príjmového a výdavkového a zároveň do číselného sumáru. V rámci tohto sumáru existuje pohľad na základe mesiaca a aj kategórie. Transakcie samotné je možné filtrovať podľa mnohých kritérií, ale nie je možné medzi nimi vyhľadávať. Chýba možnosť vytýčiť si finančné ciele.

Aplikácia je veľmi jednoduchá na používanie, ale ovládanie niektorých funkcií považujem za neintuitívne. Nastavenia nemajú typickú ikonku ozubeného kolieska ale len kruhu. Zadávanie hodnoty môže spôsobovať problémy, nakoľko je implicitne zapnutá možnosť desatinnej čiarky a používateľ sa môže jednoducho ukliknúť a pomýliť v zadávaní. Avšak to sa dá vypnúť. V aplikácií absentuje návod alebo tipy ako aplikáciu používať ale v nastaveniach je možnosť dostať sa k často kladeným otázkam alebo zákazníckej podpore.

Aplikácia pôsobí takmer až noblesne, čo je docielené typom písma a tlmenými farbami. Grafové diagramy sú zrozumiteľné a majú zaujímavý vizuál. Používateľ si môže prepnúť na tmavý režim, ale nie je možná varianta automatického prepínania medzi tmavým a svetlým režimom.

V rámci ďalších funkcionalít používateľ môže editovať kategórie, nastaviť prvý deň v týždni a zapnúť notifikácie alebo zabezpečenie pomocou biometrie – *FaceID*. Na obrázku 1.3 sú zobrazené ukážky obrazoviek tejto aplikácie.



Obr. 1.3: Obrazovky aplikácie Fleur

1.3.4 Finbot

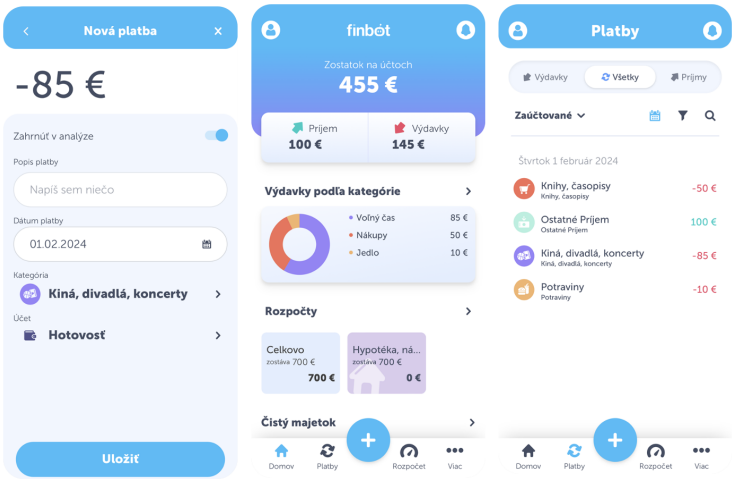
Posledná aplikácia [9], na ktorú som sa pozrela bližšie, je aplikácia od tímu slovenských vývojárov, ktorí stoja aj za aplikáciou alebo platformou, ktorá na slovenskom trhu sprostredkúva investovanie pre širokú verejnosť.

Pri spustení aplikácia vyzýva na registráciu alebo prihlásenie, používateľ si musí vytvoriť PIN kód prípadne zapnúť overenie pomocou *FaceID*. Následne aplikácia poskytuje možnosť pripojiť si bankový účet (tomu predchádza overenie identity) a je to obmedzené len pre niektoré krajiny (zväčša v rámci Európskej únie). Tak ako v predchádzajúcich prípadoch je v aplikácií možné vytvárať transakcie a kategorizovať ich. Aplikácia poskytuje aj funkcionality na vytváranie rozpočtov a grafických prehľadov. Avšak aj v tejto aplikácií chýba možnosť vytýčenia finančných cieľov a prehľad ich plnenia. Výdavkami je možné filtrovať podľa času a prehľadávať podľa názvu.

Aplikácia je prehľadná a intuitívna, piktogramy sú vhodne použité. Absentuje možnosť návodu pre nových používateľov, ale v aplikácií sú odkazy na ich podcasty, videá a ďalšie návody.

Ďalej oceňujem možnosť pridať účet z Finax `https://www.finax.eu/` (investičná platforma), nastaviť si upozornenia, zmeniť jazyk a menu, prípadne ak nastane nesprávna automatická kategorizácia transakcie, zmeniť to a vytvoriť tým pravidlo pre všetky nadchádzajúce prípady. Niektoré obrazovky tejto aplikácie sú zobrazené na obrázku 1.4.

Ďalšie aplikácie, ktoré som našla a zaujali ma sú *Finstat, WalletApp, MoneyStats, Accountit a Fin.*



Obr. 1.4: Obrazovky aplikácie Finbot

Zhrnutie

Všetky aplikácie poskytujú základné funkcie na vytváranie mesačných rozpočtov, pridávanie transakcií alebo výdavkov a ich kategorizovanie a možnosti vlastných nastavení. Žiadna z mnou skúmaných aplikácií nedisponovala funkciami na vytváranie finančných cieľov a sledovanie priebehu ich plnenia. Tento aspekt predstavuje príležitosť na diferenciáciu a pridanú hodnotu v navrhovanej aplikácií. Používateľské rozhranie bolo dobre navrhnuté a vo väčšine aplikácií bola orientácia intuitívna. Na záver hodnotenia dodávam, že prieskum prebiehal v decembri 2023 a aktuálny stav už tomu nemusí zodpovedať.

1.4 Osobné financie

Ako bolo spomenuté v úvode práce, financie nespochybniteľne hýbu svetom a predstaviť si to bez nich je v dnešnej dobe obzvlášť obtiažne. Prvý kontakt so zaobchádzaním s osobnými financiami prichádza pomerne skoro, v rodine alebo na základnej škole. Rozvíjanie finančnej gramotnosti býva často zakomponované do úloh na matematike alebo aj ostatných predmetov, čo tvorí základne povedomie o šetrení a nakladaní s príjmami a výdavkami. Neskôr sa k tomu pridáva koncept úrokovej miery, hypoték a pôžičiek. Deti na základnej alebo strednej škole sa stretnú aj s pojmami dlh alebo rozpočet aj keď ich širšie dôsledky im nemusia byť známe. S príchodom prvých brigád a prác sa ich znalosti a skúsenosti s financiami prehlbujú. Takto by sme mohli pokračovať cez hypotéky, úvery, investovanie, odkladanie na dôchodok, platenie zdravotného a sociálneho poistenia cez mnohé ďalšie. Mnohé z pojmov a procesov nie sú nevyhnutne súčasťou školského vzdelania ale nepochybne by mali doň patriť.

Pre ujasnenie pojmov uvádzam, že v tejto práci a aplikácií sa pod pojmom *rozpočet* rozumie množstvo alokovaných prostriedkov, ktoré môže používateľ minúť za daný mesiac. V súčte je to maximum sumy výdavkových transakcií. Na koncept šetrenia sa nahliada cez hodnotu finančného cieľa, na ktorý si používateľ môže postupne odkladať fyzické peniaze alebo peniaze, ktoré má na bankovom účte. V aplikácií vyplní sumu, ktorá sa priráta smerom k celkovej čiastke vytýčeného cieľa, a tým si zaznamená svoj progres.

1.4.1 Bankové API

Application Programming Interface (API)

API, skratka pre Application Programming Interface, v preklade aplikačné rozhranie, predstavuje súbor mechanizmov – protokolov, definícií a funkcií, vďaka ktorým si môžu softvérové aplikácie vymieňať dáta – komunikovať a interagovať medzi sebou. Vo webovej sfére sa jedná zväčša o dvojicu komunikujúcich – server a klient, kedy server zdieľa svoje funkcie a dáta s klientskými aplikáciami bez toho, aby museli mať prístup k jeho vnútornej implementácií a všetkým dátam. Ich komunikácia prebieha formou request-response (požiadavka-odpoveď), ktorá má vopred definovanú formu a štruktúru a je spravidla podrobne definovaná a popísaná v dokumentácií. Rozhranie by malo byť navrhnuté tak, aby zvládalo veľké množstvo požiadaviek a v prípade potreby bolo jednoducho škálovateľné. Zároveň je vhodné aby podporovalo široké spektrum formátov dát (json, xml, csv a ďalšie), čím bude prístupnejšie pre väčšiu skupinu produktov a vývojárov. [10] Ako pri takmer každej časti softvéru je nutné klásť dôraz aj na bezpečnosť rozhrania. Prístup k dátam a funkciám rozhrania by mal byť ošetrený buď pomocou bezpečnostných tokenov, protokolov alebo iných mechanizmov, aby nedochádzalo k ich zneužitiu. API hrá dôležitú úlohu pri integrácii rôznych softvérových systémov a umožňuje vytváranie nových aplikácií, ktoré môžu využívať už existujúce funkcionality. Existuje niekoľko architektúr rozhraní, medzi najznámejšie patrí SOAP (Simple object access protocol) API a REST (Representational state transfer) API.

Open banking

Načrtnime si situáciu, kedy máme niekoľko bankových účtov u rôznych bánk. Pravidelne máme na týchto účtoch rôzne príjmové alebo výdavkové transakcie. Každá banka poskytuje svoju vlastnú mobilnú aplikáciu pre prístup do bankovníctva, kde môžeme sledovať transakcie na príslušnom účte. Aby sme mali prehľad o všetkých účtoch a transakciách, ktoré sme vykonali za posledný mesiac, musíme buď sledovať notifikácie všetkých mobilných bankových aplikácií alebo obsah všetkých aplikácií pravidelne kontrolovať. Táto situácia môže byť časovo náročná a nepraktická, nehovoriac o tom, že nie všetky aplikácie musia byť prehľadné a poskytovať všetky funkcionality.

Jednou z možností, ako zlepšiť túto situáciu, je využitie konceptu nazývaného *Open Banking* v preklade otvoreného bankovníctva. V rámci otvoreného bankovníctva banky a iné finančné inštitúcie poskytujú prístup k informáciám o účtoch, jednotlivých transakciách a službách banky prostredníctvom rozhrania API [11]. Ak by sme mohli pristupovať k údajom o všetkých našich bankových účtoch cez jednu centrálne umiestnenú aplikáciu alebo platformu, mohli by sme získať prehľad o všetkých našich finančných transakciách na jednom mieste. Toto by nám ušetrilo čas a umožnilo mať lepší prehľad o našich financiách. A to práve popisuje *multibanking* [12], ktorý sa zameriava na centralizáciu účtov z rôznych bánk do jednej aplikácie alebo platformy.

Od januára 2018 sa v krajinách Európskej únie, stala záväznou smernica o platobných službách – **Payment Services Directive (PSD 2)**, ktorá si za cieľ kladie zvýšiť transparentnosť bankového sektoru, podporiť hospodársku súťaž a zlepšiť ochranu spotrebiteľov. Dôraz samozrejme dáva aj na bezpečnosť a autentifikáciu, pretože s prichádzajúcim uvoľneným prístupom k dátam je nutné myslieť aj na ich možné zneužitie [13]. Vďaka platnosti tejto smernice majú regulátorom schválené aplikácie tretích strán možnosť pristupovať k údajom o účtoch spotrebiteľov prostredníctvom rozhrania API, čo môže viezť k vzniku nových aplikácií a podnecovaniu inovácií v sektore platobných služieb. Prístup týchto aplikácií k dátam používateľov je regulovaný a je presne stanovené k akým dátam majú prístup. V prvom rade je nevyhnutný súhlas klienta. Na základe tohto súhlasu sa stretávame s dvoma typmi aplikácií tretích strán

- **AISP – Account Information Service Provider** – Poskytovateľ služieb informovania o účte – kedy aplikácia tohto poskytovateľa len zobrazuje informácie o transakciách a účte, ale platby alebo príkazy nemôže iniciovať.
- **PISP – Payment Initiation Service Provider** – Poskytovateľ platobných inicializačných služieb – v tomto prípade je poskytovateľovi používateľom udelený súhlas na iniciovanie platobných príkazov v danej banke. [14]

Následne to môže mať každá banka rôzne. Líšia sa prístupom k API, spravidla je nevyhnutná registrácia aplikácie tretej strany v ich prostredí a následne má v počiatočnej fáze prístup len k testovacím dátam. Líšia sa aj v spôsobe autentifikácie – tokeny, protokoly, napr. OAuth.

Na prínos *Open banking* sa z pohľadu biznisu môžeme pozerať ako na skvelú príležitosť vstúpiť na trh bankovníctva, ktorý bol dlho silne regulovaný a uzavretý. Banky dominujúce tomuto sektoru neumožňovali vstup menších podnikov a firiem na trh a všetky klientské dáta boli silne centralizované a úzko späté s jednou inštitúciou, čo by *Open banking* mal narušiť. Z pohľadu zákazníka to môže priniesť výhody ako pohodlnosť – všetky bankové dáta na

jednom mieste (v jednej aplikácií), personalizácia – vznikajú nové finančné riešenia, ktoré sa sústreďujú na konkrétnych používateľov a ich potreby. [15]

V rámci analýzy konkurencie som popísala niektoré aplikácie, ktoré túto funkcionalitu majú, aj keď je dostupná najmä v platenej verzií alebo si vyžaduje overenie klienta pomocou preukazu totožnosti a poskytnutia ďalších osobných údajov.

Prirodzene som sa bližšie zamerala na slovenský a český bankový trh. Nakoľko cieľom aplikácie nie je žiadne platby iniciovať ale len existujúce zobrazovať, popisovať budem problematiku AISP. Konkrétne budem porovnávať API rozhranie *Slovenská sporiteľna a.s.* [16] a *Fio banka a.s.* [17]. Obe poskytujú vývojárske prostredie pre aplikácie tretích strán, po registrácií prístupné testovacie prostredie s testovacími dátami a následne postup ako sa dostať k produkčným dátam. Avšak ako som spomenula vyššie, technické nuansy majú obe banky. Ponúka sa teda otázka, ako sa k tomuto problému postaviť z pozície návrhu a implementácie aplikácie. Je dostupná detailná dokumentácia API rozhrania. Na prvý pohľad veľmi podobné requesty, ale s inými parametrami a aj dáta, ktoré vracajú sa líšia v štruktúre, názvoch jednotlivých polí – čo vedie k záveru, že pre každú banku – predstavme si, že nebudú dve, ale budú ich desiatky, je nevyhnutné pripraviť konkrétnu logiku, ako dáta spracovávať. Akékoľvek zmeny v rozhraniach by pochopiteľne viedli k nefunkčnosti v aplikácií a nutným úpravám. Za ideálne by sa dalo považovať, keby requesty a dáta, ktoré vracajú, boli globálne regulované, aj keď z tohto hľadiska je to utopická myšlienka.

Predmetom tejto diplomovej práce nie je vytvorenie aplikácie tretích strán, ktorá by využívala vybrané klientské bankové dáta určitej banky a nad nimi mala postavené vlastné riešenie. Na vytvorenie takej aplikácie je mimo iné potrebná licencia od regulátora – v tomto prípade Českej národnej banky. Názornou ukážkou použitia konceptu *Open banking* je ale jedna z funkcionalít tejto aplikácie, a to riešenie na získavanie transakcií pomocou verejne dostupného API Fio banka a.s. [18]. Podmienkou, aby sa ktorýkoľvek používateľ (ktorý je nutne aj klient Fio Banky), vedel dostať k svojím pohybom na účte, je, aby si v internetovom bankovníctve tejto banky, ako majiteľ účtu alebo osoba s príslušným oprávnením k danému účtu vedeného vo Fio banke, vygeneroval token. Táto požiadavka na vygenerovanie tokenu je silno autorizovaná – čo znamená, že je vyžadované potvrdenie, buď pomocou zaslaného SMS kódu, alebo potvrdením v mobilnej aplikácií. Následne sa používateľovi zobrazí token, ktorý je možné použiť k získaniu informácií o platbách na účte (ale aj k zadávaniu platobných príkazov).

Z dokumentácie Fio API bankovníctva [18] ďalej vyplýva, že pre každý účet si musí majiteľ účtu vygenerovať nový token, čo predstavuje rozdiel medzi vývojárskym prostredím, kedy ako aplikácia tretej strany máte po splnení podmienok a získaní licencie prístup k všetkým účtom a ich vybraným bankovým dátam. Pre token je ďalej možné nastaviť, či má slúžiť len na získavanie informácií o platbách na účte, alebo aj na zadávanie platobných a inkaso

príkazov. Platnosť tokenu musí byť ohraničená, čo popisuje upozornenie z dokumentácie API, citujem: *„Každý nově založený token musí mít nastavenou platnost, nelze založit token bez doby platnosti nebo platnosti delší jak 180 dní. Pokud je zvoleno automatické prodlužování, tak při každém přihlášení do Internetbankingu nebo Smartbankingu se platnost tokenu prodlouží na 180 dnů ode dne přihlášení.“*

Ďalším bezpečnostným prvkom, ako ochrániť dáta, je možnosť získavať len transakcie, ktoré nie sú staršie ako 90 dní. V prípade, že by si majiteľ účtu na základe vygenerovaného tokenu, chcel stiahnuť dáta, ktoré sú staršie ako 90 dní, je nevyhnutná ďalšia autorizácia v Internetovom bankovníctve, ktorá dočasne sprístupní kompletnú históriu pohybov na účte ale len na 10 minút. API riešenie Fio Banky umožňuje získavanie dát v mnohých formátoch, ako napríklad v CSV, HTML, JSON alebo XML. V dokumentácií je uvedených niekoľko URL requestov, ktoré sú prístupné, na obrázku 1.5 je zobrazený popis requestu, ktorý je použitý v aplikácií. Dáta, ktoré vracia v zvolenom formáte Fio XML sú uvedené na obrázku 1.6.

5.2.1 Pohyby na účtu za určené období

Struktura:	https://www.fio.cz/ib_api/rest/periods/{token}/{datum od}/{datum do}/transactions.{format}	
	Token	unikátní vygenerovaný token
	Datum od	datum - začátek stahovaných příkazů ve formátu rok-měsíc-den (rrrr-mm-dd)
	Datum do	datum - konec stahovaných příkazů ve formátu rok-měsíc-den (rrrr-mm-dd)
	Formát	formát pohybů
Příklad:	Získání pohybů v období od 25.8.2012 do 31.8.2012 v xml	
https://www.fio.cz/ib_api/rest/periods/aGEMQB9Idh35fh1g51h3ekkQwyGlQ/2012-08-25/2012-08-31/transactions.xml		

Obr. 1.5: Popis štruktúry requestu z dokumentácie API

Samozrejme, generovanie tokenu pre všetky účty, prípadne potvrdzovanie v Internetovom bankovníctve, veľké zvýhodnenie používateľom nemusí priniesť. Vhodné použitie tohto rozhrania vidím napríklad v účtovníctve firiem alebo ako určitý druh evidencie.

Výsledek dotazu na pohyby v období od 1. 7. 2012 do 31. 7. 2012

```
<AccountStatement>
  <Info>
      <accountId>2111111111</accountId>
      <bankId>2010</bankId>
      <currency>CZK</currency>
      <iban>CZ7920100000002111111111</iban>
      <bic>FIOBCZPPXXX</bic>
      <openingBalance>7356.22</openingBalance>
      <closingBalance>7321.22</closingBalance>
      <dateStart>2012-07-01+02:00</dateStart>
      <dateEnd>2012-07-31+02:00</dateEnd>
      <idFrom>1147608196</idFrom>
      <idTo>1147608197</idTo>
  </Info>
  <TransactionList>
     <Transaction>
       <column_22 id="22" name="ID pohybu">1147608196</column_22>
       <column_0 id="0" name="Datum">2012-07-27+02:00</column_0>
       <column_1 id="1" name="Objem">-15.00</column_1>
       <column_14 id="14" name="Měna">CZK</column_14>
       <column_2 id="2" name="Protiúčet">2222233333</column_2>
       <column_3 id="3" name="Kód banky">2010</column_3>
       <column_12 id="12" name="Název banky">Fio banka, a.s.</column_12>
       <column_7 id="7" name="Uživatelská identifikace"> </column_7>
       <column_8 id="8" name="Typ">Platba převodem uvnitř
       banky</column_8>
       <column_9 id="9" name="Provedl">Novák, Jan</column_9>
       <column_25 id="25" name="Komentář">Můj test</column_25>
       <column_17 id="17" name="ID pokynu">2102392862</column_17>
     </Transaction>
     <Transaction>
       <column_22 id="22" name="ID pohybu">1147608197</column_22>
       <column_0 id="0" name="Datum">2012-07-27+02:00</column_0>
       <column_1 id="1" name="Objem">-20.00</column_1>
       <column_14 id="14" name="Měna">CZK</column_14>
       <column_2 id="2" name="Protiúčet">2222233333</column_2>
       <column_3 id="3" name="Kód banky">2010</column_3>
       <column_12 id="12" name="Název banky">Fio banka, a.s.</column_12>
       <column_7 id="7" name="Uživatelská identifikace"> </column_7>
       <column_8 id="8" name="Typ">Platba převodem uvnitř
       banky</column_8>
       <column_9 id="9" name="Provedl">Novák, Jan</column_9>
       <column_25 id="25" name="Komentář"></column_25>
       <column_17 id="17" name="ID pokynu">2102392863</column_17>
     </Transaction>
  </TransactionList>
</AccountStatement>
```

Obr. 1.6: Príklad response volania requestu na získanie informácií o platbách z Fio Api bankovníctva

Kapitola 2

Návrh

Nasledujúca kapitola obsahuje popis návrhu aplikácie, bližšie popisuje dátový model a spôsob ukladania dát. Venuje sa aj návrhu používateľského rozhrania, ako aj porovnaniu nástrojov, ktoré tento proces umožňujú. Kapitola je zakončená detailnou charakteristikou technických aspektov práce, zahŕňajúcich použitú architektúru, technológie a knižnice.

Medzi analýzou a implementáciou, ako časťami softvérového vývoja, stojí návrh. Kým v analýze sa na veci pozeráme abstraktne, z rôznych strán a skúmame možnosti, v návrhu sa tieto vízie stávajú realitou. Na rad prichádza dizajn systému zvonka ako aj vo vnútri. Pre mobilnú aplikáciu to znamená návrh obrazoviek, vytvorenie prototypu, voľba a popis vhodnej architektúry, či návrh databázových štruktúr. Výsledkom návrhovej fázy je spravidla detailný popis správania sa systému, jeho komponentov, ich vzájomnej komunikácie a návrh používateľského rozhrania, ktoré slúžia ako podklad pri ďalšom vývoji samotnej aplikácie. [19]

2.1 Ukladanie dát

Jedným z hlavných cieľov aplikácie je vytvorenie prehľadu o finančnej situácii používateľa. Tento prehľad by nebol možný bez ukladania dát a transakcií a prípadne následného vytvárania štatistických prehľadov. Je len naozaj málo aplikácií, ktoré by nepotrebovali žiadne dátové úložisko. Pre platformu iOS existuje hneď niekoľko riešení alebo frameworks, ktoré ukladanie a nakladanie s dátami riešia. V nasledujúcej časti bližšie popisujem natívne spôsoby od *Apple* medzi ktoré patrí *Swift Data*, možnosť ukladania do tzv. *User defaults* alebo do súborového systému.

2.1.1 User defaults

Keď potrebujeme uložiť dáta, ktorých je málo a sú tvorené jednoduchými dátovými typmi, zvyčajne ide o Boolean hodnotu `true/false`, alebo celé

číslo (Integer), je vhodné použiť uloženie do tzv. *User defaults* [20], čo v preklade znamená, používateľské nastavenia/preferencie. Týmto spôsobom je jednoduché uložiť príznak napríklad, či používateľ chce použiť tmavý režim, aký jazyk aplikácie preferuje alebo základné informácie o tom, aká bola posledná relácia, keď používateľ použil danú aplikáciu. Všetky tieto dáta sú načítané pri každom spustení aplikácie a vo všeobecnosti nie je daný vrchný limit úložiska, ktoré je pre takýto typ dát k dispozícií, ale je lepšie keď sa jedná o nižšie stovky kilobajtov. [21] Toto úložisko je reprezentované ako slovník (*dictionary*), pod zadaným kľúčom, ktorý je vždy reťazec, si ukladáme jednoduchú hodnotu – reťazce, slovníky, celé či desatinné čísla alebo príznaky true/false či iné. Nasledujúci príklad ilustruje, ako získavame uložené hodnoty. Pomocou prvého príkazu nastavíme hodnotu `true` pod kľúč „PrefersDarkMode“ a druhým príkazom túto hodnotu z *User defaults* získame.

```
UserDefaults.standard.set(true, forKey: "PrefersDarkMode")
UserDefaults.standard.string(forKey: "PrefersDarkMode")
```

V prípade, že by sme chceli používateľské nastavenia zdieľať naprieč viacerými zariadeniami je nevyhnutné využiť *NSUbiquitousKeyValueStore*, nakoľko *User defaults* sú lokálne uložené na každom zariadení a túto funkcionalitu neposkytujú. *NSUbiquitousKeyValueStore* je založený na podobnom princípe akurát je úložisko v cloude. [22]

Obdobné riešenie ako ukladať kľúč-hodnota záznamy je ukladanie dát do tzv. *Property lists*.

2.1.2 File system

V prípade, že sa jedná o väčšie alebo komplexnejšie dáta, *Apple* poskytuje pre každú aplikáciu úložisko (časť súborového systému), kde je možné ukladať súbory. Prístup a manipulácia s týmto úložiskom je sprostredkovaná pomocou API a objektu *FileManager*. Takýmto spôsobom môžu byť ukladané aj veľké súbory ako napríklad fotografie, obmedzením je len veľkosť úložiska daného zariadenia. Výhodou tiež je, že použitie je možné aj s cloudovým úložiskom a je tak zabezpečená synchronizácia medzi viacerými zariadeniami. [23]

2.1.3 SwiftData

V aplikáciach, ktoré obsahujú objekty, ktoré je nevyhnutné mapovať na entity z reálneho života, v tomto prípade transakcia, či finančný cieľ, je vhodnejšie použiť komplexnejšie štruktúry ako len ukladanie dát do súborov, či používateľských preferencií. *Apple* na narábanie s takýmto typom dát poskytuje framework *CoreData*. Dôležité je zdôrazniť, že nejde o databázu ale len framework, ktorý napríklad ako dátové úložisko využíva na iOS zariadeniach zabudovanú SQLite databázu. [24] *CoreData* spravujú životný cyklus (vytvorenie, zmena, vyhľadanie, zmazanie) entít v objektovom grafe. Každá entita

je definovaná názvom, vlastnosťami a prípadnými vzťahmi s ďalšími entitami. Vďaka abstrakcií nemusia vývojári písať konkrétne SQL príkazy nad daným databázovým úložiskom, ale používajú *NSFetchRequest*, ktorý umožňuje definovať kritériá pre získanie dát z databázy. Tieto žiadosti sú vykonávané na základe definícií dátového modelu, filtrov a predikátov definovaných vo Fetch Requests. *CoreData* poskytuje možnosti synchronizácie lokálnych dát so vzdialenými zariadeniami vďaka integrácií s *CloudKit* framework, ako aj mechanizmy na zálohovanie a obnovu dát, prípadne jednoduchú migráciu. [25]

V lete 2023, *Apple* predstavil *SwiftData* [26], čo je nový framework určený na prácu s objektovými grafmi. Tento framework je postavený na pôvodných *CoreData*, ale poskytuje natívnejšie API, ktoré viac zapadá do súčasného použitia jazyka Swift. Predstavuje makrá, vďaka ktorým je definovanie entít jednoduchšie a nachádza sa priamo v kóde, čiže nie je nutné definovať schému pomocou vyplňovania formuláru v rozhraní. Oproti *CoreData* je výkonnejší, lepšie integrovateľný so *SwiftUI* ale poskytuje menej funkcionalít a nie je taký komplexný a nakoľko ide o mladý framework, ani vyspelý. [27]

Oba frameworks poskytujú možnosti asynchrónneho prístupu, čiže väčšie požiadavky nemusia zaťažovať hlavné vlákno aplikácie a spomaľovať UI (User Interface), sú použiteľné ako dočasná pamäť – cache a prinášajú podporu pre tzv. *Undo* tlačidlo, ktoré spôsobí odvolanie/zvrátenie určitej akcie napr. vymazania.

2.1.4 iCloud

Ak chceme zdieľať rôzne súbory ako dokumenty, fotky, hudbu alebo aj dáta aplikácií naprieč rôznymi zariadeniami s rovnakým *AppleID* (používateľský účet *Apple* platformy), *Apple* poskytuje cloudové úložisko nazývané *iCloud*. Aktuálne poskytuje svojím používateľom 5GB priestoru ku každému účtu zdarma. Ako bolo spomenuté vyššie, *CoreData* aj *SwiftData* poskytujú podporu na ukladanie dát práve do *iCloud* tak, aby boli synchronizované a zálohované, a tým pádom dostupné, naprieč zariadeniami.

2.1.5 Keychain

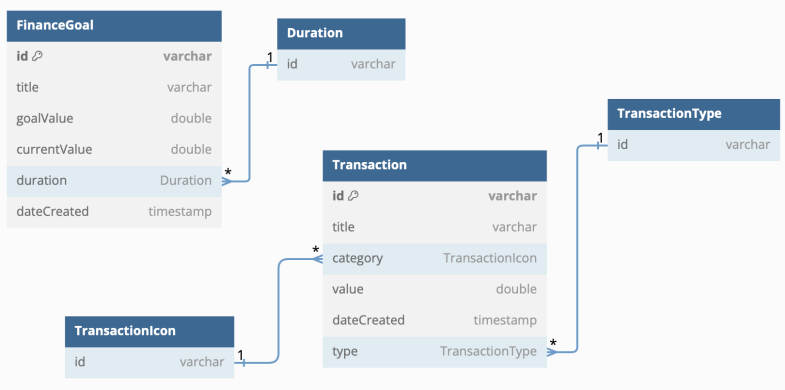
V niektorých aplikáciách je nevyhnutné aj ukladanie citlivých údajov ako sú heslá, čísla kreditných kariet a podobne. Bezpečné uloženie týchto údajov je kľúčové, pretože ich nesprávne zabezpečenie môže viesť k vážnym bezpečnostným rizikám pre používateľov. Preto nie je vhodné ukladať ich len ako text do databázy a odporúčané je využiť metódy na šifrovanie s dostatočne silnými kryptografickými metódami. *Apple* poskytuje na platforme iOS šifrované úložisko nazývané *Keychain*, ktoré je zabezpečené symetrickým šifrovaním AES–256–GCM s 256-bitovým kľúčom. Jedným z kľúčových prvkov bezpečnosti Keychain je jeho prístupový systém, ktorý umožňuje prístup len po zadaní používateľom definovaného hesla. [28]

Z pohľadu vývojára sa ku Keychain pristupuje pomocou *Keychain Services API* [29], ktoré rovnako poskytuje *Apple*. Toto API umožňuje interakciu s Keychain, vrátane pridávania, získavania, aktualizovania a mazania údajov. Programátor môže tiež nastavovať prístupové kontroly, ktoré určujú, ktoré aplikácie alebo služby majú prístup k uloženým údajom.

V tejto aplikácií bude Keychain použitý na prípadné uloženie tokenu, ktorý je súčasťou volaní bankového API a slúži na autentifikáciu používateľa.

2.1.6 Použitý dátový model

Dátový model v tejto aplikácií je pomerne jednoduchý, medzi tabuľkami nie sú žiadne vzťahy, čiže nie je nevyhnutné riešiť väzby. Ide o dve hlavné tabuľky, ktoré predstavujú entity finančných cieľov a transakcií – „FinanceGoal“ a „Transaction“. Ako obrázok 2.1 naznačuje, prítomné sú ešte pomocné tabuľky, ktoré sú nositeľom identifikátora použitého číselníku – výpočtového typu – typ transakcie nadobúda hodnoty *príjem* alebo *výdavok*, kategória majúca hodnoty napríklad: *Cestovanie, Jedlo, Zdravie* atp. a trvanie cieľa s možnými hodnotami *mesačný* alebo *ročný*.



Obr. 2.1: Databázový model aplikácie

Ďalej je v aplikácií použité spomínané úložisko *User defaults*, do ktorého je uložených niekoľko príznakov. Uloženie zvolenej meny aplikácie, informácia o zapnutých upozorneniach, ale aj hodnota (ak je nastavená) celkového rozpočtu pre aktuálny mesiac.

2.2 Používateľské rozhranie

Neoddeliteľnou súčasťou každého softvérového produktu, s ktorým prichádzajú ľudia do kontaktu, je používateľské rozhranie. Je to to prvé, s čím sa stretneme keď prvýkrát navštívime webovú stránku, použijeme mobilnú aplikáciu ale napríklad aj keď ovládame novú práčku. Je to všetko, čo sprostredkúva

komunikáciu medzi používateľom a systémom alebo aplikáciou. Používateľské rozhranie zahŕňa všetky prvky, ktoré používateľ vidí, počuje a používa na obrazovke alebo na inom zariadení, ako napríklad tlačidlá, ikony, textové polia, obrázky, zvuky, gestá a mnohé ďalšie. [30] Avšak používateľské rozhranie nie je len o samotnej estetike a interakcii, ale aj o uľahčení používateľskej skúsenosti a dosiahnutí používateľských cieľov. Jeho úlohou je minimalizovať záťaž a zmätok pre používateľa, čím zvyšuje jeho produktivitu a spokojnosť. Cieľom dizajnérov grafického rozhrania by malo byť dosiahnutie balansu medzi estetickosťou, intuitívnosťou a použiteľnosťou pre rôzne skupiny ľudí, ktorí prichádzajú s produktom do kontaktu.

Správny prístup k návrhu používateľského rozhrania vyžaduje úzku spoluprácu s našou cieľovou skupinou, ktorú som bližšie špecifikovala v predchádzajúcej kapitole. Tento prístup sa zakladá na dôkladnom definovaní persony, následných náčrtov, konceptualizácie a opakovaného vytvárania prototypov, či overovania záverov. Keď máme funkčný prototyp nasleduje tvorba grafiky a dizajnu vizuálnych prvkov – použitie farieb, typografia a ikony. V neposlednom rade používateľov zapojíme do testovania, aby sa identifikovali prípadné problémy alebo oblasti na zlepšenie. Na základe spätných väzieb sa vykonávajú iterácie a úpravy, ktoré vedú k zdokonaleniu rozhrania a zabezpečujú lepšiu použiteľnosť a spokojnosť používateľov. [31] Keď sme spokojní s výsledkom, finálnu verziu integrujeme do konkrétneho softvérového produktu alebo webovej stránky.

Existuje nespočet pravidiel a princípov, či odporúčaní, ktorých by sa dizajnéri používateľského rozhrania mali držať. Medzi najznámejšie nepochybne patrí *Nielsenova heuristika* [32]. Je to technika hodnotenia použiteľnosti softvérového produktu alebo webovej stránky, ktorá bola vyvinutá Jakobom Nielsenom a Rolfom Molichom, renomovanými odborníkmi na témy týkajúce sa použiteľnosti webu. Ide o sadu 10 všeobecných zásad, vďaka ktorým vieme identifikovať problémy v používateľskom rozhraní a naopak ich dodržanie pomáha vytvárať kvalitné, používateľsky orientované produkty, ktoré sú efektívne, spoľahlivé a spĺňajú potreby a očakávania používateľov.

2.2.1 10 pravidiel heuristickej analýzy Jakoba Nielsena

1. Viditeľnosť stavu systému – Systém by mal za každých okolností včas informovať používateľa o stave aplikácie napríklad pomocou *progress baru*, odpočtu času, atp. Aplikácia by nemala zamrznúť alebo dostať sa do stavu kedy nie je schopná reagovať na používateľské vstupy.

2. Zhoda medzi systémom a realitou – V aplikácií by sa mali používať pojmy/skratky/koncepty a princípy, ktoré korešpondujú s terminológiou cieľovej skupiny. Použitie metafor a konvencií musí zodpovedať použitiu v reálnom svete (príklad s košom, z ktorého sa dá aj vyberať, prehľadávať ho, nielen vysypať.) Ikony a ich použitie by malo byť samo popisujúce.

3. Kontrola a sloboda používateľa – Veľakrát používateľ vykoná akciu omylom, preto by v každej situácií mal byť jasne vyznačený spôsob ako nechcenú akciu zvrátiť. V prípade, že vyvolá nezvratnú akciu musí byť vopred upozornený a ideálne musí opakovane potvrdiť jeho úmysel. Používateľ musí mať pocit, že systém ovláda on (a nie naopak), aby sa vyhol frustrácií.

4. Konzistencia v rámci platformy a so štandardami – Toto pravidlo sa zameriava najmä na zhodu medzi systémom a platformou, na ktorej systém beží. Zhodné použite komponent, farieb, fontov platformy ako aj očakávané chovanie vyvolaných akcií. V jednoduchosti ide o zachovanie konvencií danej platformy.

5. Prevencia chýb – V prípade, že nastane chyba (aj keď dobre navrhnutý systém, by nemal používateľovi umožniť, aby sa do takého stavu dostal) by mala byť zobrazená jasná a zrozumiteľná chybová hláška. Ďalej, napríklad pri zadávaní vstupu používateľom, by mal mať formulár označené povinné polia, prípadne nároky, ktoré sú kladené (sila hesla) a zároveň, kde je to nutné, zobrazovať potvrdzovacie dialógy.

6. Uprednostniť rozpoznanie pred spomínaním – Cieľom tohto pravidla je sústrediť sa, resp. odbremeniť používateľa, aby si pamätal informácie, ktoré nemusí. Toto je jednoduché docieliť tým, že v rámci napríklad menu, sú zobrazené len akcie, činnosti či prvky, ktoré sú pre používateľa prístupné a nemusí ich držať v pamäti.

7. Flexibilita a efektivita použitia – Aplikácia by mala byť postavená tak, aby pre skúsených používateľov umožňovala zefektívnenie si používania. Čo znamená napríklad použitie klávesových skratiek, makra, šablóny, pokročilé nastavenia, ktoré bežný používateľ nepotrebuje.

8. Estetickosť a minimalizmus dizajnu – Menej je viac. Toto sa spája aj s informáciami v aplikácií, tj. malo by byť zobrazené len to, čo je potrebné a nevyhnutné na chod a používanie aplikácie. To isté platí aj pre rôzne akcie. Je zbytočné systém zahltiť množstvom akcií, ktoré môžu pôsobiť neprehľadne, čo má za dôsledok aj rýchlosť orientovania a používania systému používateľom.

9. Zrozumiteľné hlásenia o chybe – Ak nastane situácia (ktorá by v ideálnom systéme/svete nastať nemala), že je nevyhnutné informovať používateľa o chybovom stave, do ktorého sa aplikácia dostala, je dôležité urobiť to tak, aby to bolo zrozumiteľné – použitie bežného jazyka a nie nič nevypovedajúcich chybových kódov. Zároveň je dobré poskytnúť používateľovi presný popis prečo daná chyba/chybový stav nastal a pripojiť návrh na jeho opravu/zvrátenie.

10. Návod a dokumentácia – Použitie systému by malo byť natoľko intuitívne a priamočiare, že by nemalo vyžadovať žiaden návod na použitie. Aj napriek tomu je nevyhnutné ho pripraviť. Preferovaný je návod vo forme príkladov a kľúčová je tiež možnosť vyhľadávania a to, aby bola daná pomoc kontextová.

S pojmom používateľské rozhranie (*anglicky user interface* – skratka UI) je často spájaný pojem používateľská skúsenosť (*anglicky user experience* – skratka UX). Jeden pojem bez druhého vlastne nemôže existovať. Kým UI popisuje ako aplikácia a jej obrazovky vyzerajú, či ako s ňou používateľ interaguje, tak UX skúma celkový pocit a zážitok z používania danej aplikácie, a to, ako na používateľa jej ovládanie vplýva. Čiže nejedná sa len o vizuálny a interaktívny dizajn, ale aj o obsah aplikácie, jej funkčnosť, rýchlosť odpovede systému, ľahkosť navigácie a celkový dojem. [33]

2.3 Prototypovanie

Ako bolo už spomenuté, dizajn produktu sa odvíja od cieľovej skupiny a od očakávaných funkcionalít. Na začiatku vytvárania používateľského rozhrania má dizajnový tím určitú predstavu o tom, ako by mala aplikácia vyzerať. Táto predstava ale nemusí byť rovnaká akú majú ostatné zainteresované strany, ako napríklad klient, investor atp. Prototypovanie je proces, počas ktorého tímy vytvárajú fyzické alebo digitálne modely softvérového produktu, ktoré simulujú konečnú používateľskú skúsenosť. Slúžia najmä ako overenie, že predstavy dizajnérov sa zhodujú s predstavou používateľov a ďalších zahrnutých subjektov. Na základe spätnej väzby, sa prototypy vylepšujú dopĺňajú a opätovne testujú s používateľmi. Zároveň, znižujú riziko, pretože umožňujú identifikovať a riešiť problémy v raných štádiách vývoja, čo môže viesť k zníženiu nákladov a eliminácii nedorozumení v neskorších fázach projektu. [34]

Existujú rôzne prístupy k vytváraniu prototypov, môžu byť vytvorené pomocou rôznych nástrojov a techník, vrátane statických wireframes, klikateľných mockups, interaktívnych prototypov alebo dokonca funkčných modelov. Dôležité je, aby prototypy boli dostatočne reprezentatívne, aby umožnili účinné testovanie a validáciu návrhu produktu.

Rozlišujeme medzi dvoma základnými konceptami, prvým sú *Low fidelity (Lo-Fi)* prototypy – čo vo voľnom preklade znamená nízky level presnosti alebo vernosti, čiže ide o prototypy, ktoré sú pomerne vzdialené od reálneho výsledného riešenia a to buď tým, že sú len papierového charakteru, alebo ak sú v digitálnej forme, tak nie sú klikateľné a používateľ s nimi nemôže aktívne interagovať. Tento typ prototypov popisuje a zaznamenáva najmä rozmiestnenie a veľkosť jednotlivých textov alebo komponentov, nekladie dôraz na farby, obrázky ani reálne texty. Ich výhodou je, že sú rýchlo a lacno vytvoriteľné, nevyžadujú špeciálnu schopnosť ako ich vytvárať (znalosť softvéru) a ľahko reagujú na zmeny. Nevýhodou je, že sú nerealistické a akékoľvek testovanie

je svojím spôsobom kompromitované tým, že používateľ ich nemôže pohodlne vyskúšať, ale musí si veľa vecí predstavovať. [35]

Druhým typom sú *High fidelity (Hi-Fi)* prototypy, ktoré sú doplnením toho, čo v Lo-Fi prototypoch absentuje. Väčšinou ide o digitálne prototypy vytvorené za pomoci špeciálneho softvéru na to určeného, ako napríklad *Figma, Adobe XD* alebo ďalšie. Sú vysoko funkčné, interaktívne a od reálneho produktu sa takmer nelíšia. Testovanie sa tým pádom stáva presnejšie a pre používateľa prirodzenejšie, avšak vytváranie takýchto prototypov vyžaduje viac času, je finančne náročnejšie a je nevyhnutná znalosť a schopnosť ovládať určitý softvér alebo prípadne programovací jazyk. [36]

Oba prístupy majú svoj čas a miesto kedy sú vhodné na použitie. Kým *Lo-Fi* prototypy sa využívajú v skorých fázach návrhu, vo fáze brain-stormingu (skupinová metóda hľadania kreatívnych riešení) a kým sa všetky zainteresované strany zhodnú na jednej predstave, *Hi-Fi* prichádzajú v čase, keď je väčšina konceptov vyjasnená a blížime sa k implementácií.

2.3.1 Mockup

V rámci tejto diplomovej práce bol vytvorený prototyp, ktorý spadá do kategórie Lo-Fi prototypov, akurát nebol použitý papier a pero ale tablet a pero k tabletu. Počas hľadania vhodného softvéru, ktorý by som mohla použiť pri tvorbe prototypu som natrafila na mnohé aplikácie, z ktorých stojí za zmienku najmä tá, ktorú som si nakoniec vybrala a to je aplikácia pre iPad s názvom *Mockup* [37]. Veľkou výhodou aplikácie je, že v platenej verzií poskytuje mnohé natívne iOS prvky už vopred pripravené, a používateľ ich len jednoducho umiestňuje na obrazovku prototypu a nemusí ich hľadať a následne importovať. Čiže sa nemusí trápiť tvorbou napríklad navigačného panelu, nakoľko ho má predpripravený a jediné čo zmení sú ikony, názvy, prípadne farby a font. Aplikácia je jednoduchá na ovládanie, k dispozícií sú predkreslené rôzne formáty obrazoviek a k dispozícií je aj možnosť pridávať vlastné prvky.

2.3.2 Figma

Vzhľadom na rozšírenosť a obľúbenosť stojí za to spomenúť a bližšie popísať aj nástroj *Figma* [38], ktorý sa radí do kategórie prototypovacích nástrojov pre *Hi-Fi* prototypy. Použitie sa neviaže len na prototypy alebo návrhy mobilných či webových aplikácií, ale aj na rôzne reklamné predmety a materiály či mnohé ďalšie grafiky. Poskytuje možnosť kolaborácie s ostatnými členmi tímu v reálnom čase a pretože je celá v cloude, je možné pracovať na projektoch z ľubovoľného počítača. Dôvodom jej popularity je aj veľká knižnica plná existujúcich komponentov, pluginov alebo len inšpirácie pre ďalších dizajnérov. Platforma je dostupná v základnej forme zadarmo a pre pokročilejšie funkcionality poskytuje rôzne možnosti predplatného.

Medzi ďalšie komunitou obľúbené nástroje patrí *Balsamiq, Proto.io, Adobe XD, InVision, Justinmind* a iné.

2.3.3 Výsledný prototyp

Výsledný Lo-Fi návrh prototypu je na obrázku 2.2 a je v anglickom jazyku. Skladá sa z troch hlavných sekcií a to *Prehľad (Overview), Rozpočet (Budget)* a *Ciele (Goals).* Prechody medzi týmito obrazovkami sú možné pomocou dolného navigačného panelu. Jeho použitie je typické pre mobilné aplikácie, zväčša, ako aj v tomto prípade, ho tvorí tri až päť ikon, ktoré predstavujú jednotlivé obrazovky. Ich výber je dôležitý, musia byť výstižné, prípadne doplnené o popis. [39] Obrazovky sú potom dostupné na jedno kliknutie a ikona obrazovky, ktorá je práve prezentovaná (vybraná) je farebne odlíšená od ostatných.

Prvá obrazovka reprezentovaná ikonou doláru ($), je *Overview*, na ktorej sa nachádza líniový graf, ktorý predstavuje vývoj príjmových a výdavkových transakcií za aktuálny mesiac. Graf je doplnený aj o textový súčet príjmov a výdavkov, ktorý sa nachádza priamo pod ním. Pod grafom sú zobrazené aj tlačidlá na pridanie novej transakcie, import transakcie alebo možnosť zobraziť všetky transakcie. V hornom navigačnom paneli sú dve ikony odkazujúce na obrazovky *Upozornenia* (*Notifications*) a *Nastavenia* (*Settings*). V rámci obrazovky *Notifications* si používateľ môže povoliť pravidelné notifikácie ohľadom rozpočtu a pripomienok na zaznamenávanie údajov do aplikácie a na obrazovke *Settings* napríklad prispôsobiť menu aplikácie alebo prečítať návod.

Ďalšou obrazovkou, v poradí druhou, v navigačnom paneli je *Budget*, reprezentovaný ikonou kalendára s výkričníkom. Na tejto obrazovke je zobrazený rozpočet pre daný mesiac, ako aj tri posledné transakcie a tlačidlo s odkazom na obrazovku so všetkými transakciami. Používateľ má opäť možnosť pridať transakciu ako aj upraviť mesačný rozpočet. Po kliknutí na príslušné akcie sa na obrazovke zobrazí list s príslušným formulárom, ktorý používateľ vyplní a potvrdí, prípadne zruší.

Poslednou obrazovkou z navigačného panelu je *Goals* s ikonou iskier, čo má v používateľoch vzbudiť pozitívny pocit a povzbudiť, či motivovať ich k plneniu svojich vytýčených cieľov. Na tejto obrazovke sú zobrazené mesačné a ročné finančné ciele rozdelené do záložiek, medzi ktorými sa dá prepínať. Je tu možnosť pridania nového cieľa, čo opäť zobrazí na obrazovke list s formulárom, ako aj prehľad cieľov a graficky zobrazené ich napĺňanie alebo splnenie. V prípade pridania transakcie k cieľu, sa po kliknutí na konkrétny cieľ otvorí list z dolnej časti obrazovky s formulárom.

Dôležitou obrazovkou, na ktorú sa dá dostať z *Overview* ale aj z *Budget* je obrazovka *Transakcie (Transactions).* Na tejto obrazovke sú zobrazené všetky transakcie, je možné vyfiltrovať len príjmové alebo výdavkové pomocou filtra a zároveň medzi transakciami vyhľadávať pomocou vyhľadávacieho poľa v hornej časti. V dolnej časti obrazovky sa nachádza tlačidlo so symbolom „+“.

Po kliknutí sa otvorí list s formulárom na pridanie novej transakcie. V hornej časti navigačného panela tejto obrazovky sa na pravej strane nachádza ikona stĺpcového grafu, ktorá odkazuje na ďalšiu obrazovku *Štatistiky (Statistics)*, kde je používateľovi zobrazený stĺpcový graf príjmov a výdavkov a pomocou sekcií v hornej časti je možné upravovať časové obdobie zobrazovaných dát. V dolnej časti pod grafom je prehľad výdavkov rozdelený do kategórií. V hornom navigačnom paneli je na pravej strane ikona, ktorá umožňuje exportovať transakcie do súboru s formátom CSV, po kliknutí na túto ikonu sa zobrazí natívny spôsob ukladania a zdieľania súborov na iOS platforme tzv. ShareSheet. Na obrazovke *Transactions* je ďalej v hornom navigačnom paneli vedľa ikony odkazujúcej na štatistiky, možnosť prejsť na obrazovku *Importovať transakcie (Import transactions)*. Na tejto obrazovke je používateľovi umožnené zadať token z internetového bankovníctva Internetbanking Fio banky (ak ešte nebol zadaný predtým), a tým využiť ich API rozhranie a importovať transakcie. Návrh zahŕňa aj obrazovky, ktoré sa zobrazia v prípade úspešného alebo chybného importu.

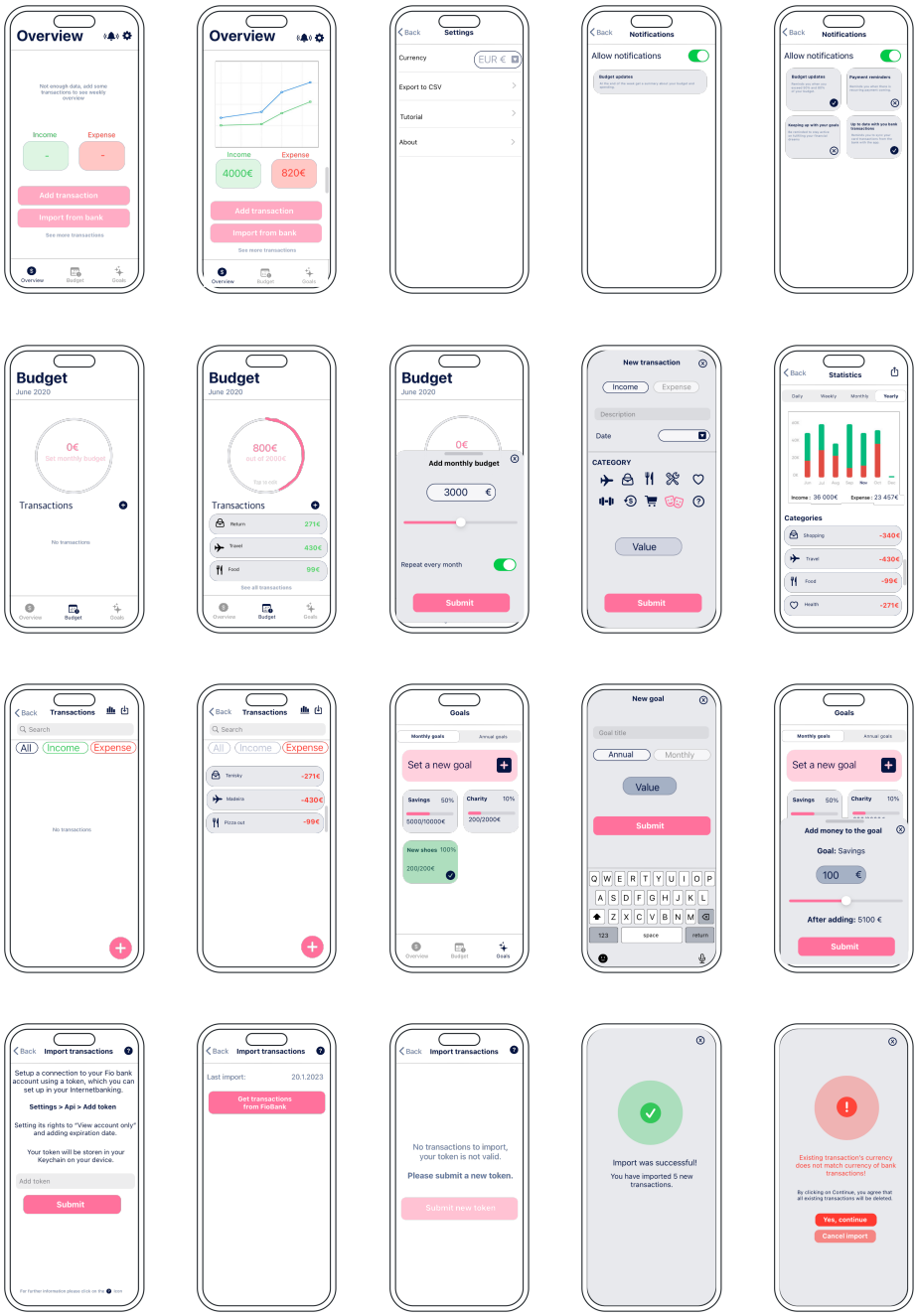
2.4 Použitá architektúra a technológie

V nasledujúcej časti sa zameriam na kľúčové aspekty programovania pre operačný systém iOS. Bližšie priblížim programovacie jazyky, ktoré sú vhodné na vývoj aplikácií pre zariadenia od spoločnosti *Apple*. Následne predstavím vývojové prostredie ako aj frameworks, ktoré sú najčastejšie používané pri tvorbe iOS aplikácií. Na záver tejto časti opíšem rôzne architektúry, typické pre mobilné aplikácie.

2.4.1 Vývoj iOS aplikácií

V súčasnosti je na vývoj aplikácií pre platformu iOS nevyhnutné mať aplikáciu *Xcode* a ďalšie jej knižnice. Aplikácia *Xcode* [40] je integrované vývojové prostredie (IDE) vyvinuté spoločnosťou *Apple* pre vývoj softvéru pre ich platformy, ako sú iOS, macOS, watchOS a tvOS. Je k dispozícií zadarmo a vývojárom poskytuje všetky potrebné nástroje na navrhnutie, vytváranie, ladenie a nasadzovanie aplikácií. Veľkým plusom je iPhone simulátor, ktorý slúži vývojárom počas tvorby aplikácie na ladenie a to tak, že simuluje správanie na reálnom zariadení. Táto aplikácia je bohužiaľ spustiteľná len na operačnom systéme macOS, čiže pokiaľ developer nemá prístup k zariadeniu od *Apple*, jeho jediná možnosť je využiť služby virtualizácie. [41]

Hlavným programovacím jazykom je **Swift** [42], taktiež vyvinutý spoločnosťou *Apple*. Vychádza z pôvodného (ale stále používaného) Objective-C, ktorý bol do roku 2014 hlavným používaným programovacím jazykom na vývoj aplikácií pre platformy iOS, macOS, watchOS a tvOS. Narozdiel od Objective-C, Swift používa jednoduchšiu syntax a viac sa orientuje na bezpečnosť a rýchlosť kódu. Ide o open source jazyk, čiže jeho zdrojový kód je verejne dostupný



Obr. 2.2: Lo-Fi model návrhu aplikácie

a je rozvíjaný komunitou vývojárov. [43] Kým vývojové prostredie Xcode je nevyhnutné pre vývoj, programovací jazyk iOS aplikácií nie je viazaný len na Swift alebo Objective-C, ale iOS aplikácie je možné vyvíjať aj v jazykoch ako je JavaScript a React Native alebo Dart a Flutter. Voľba nepoužiť jazyk Swift môže mať rôzne motivácie. Väčšinou je rozhodujúce, či chceme aplikáciu vyvinúť len pre jednu platformu a využiť jej potenciál a funkcionality naplno, alebo chceme, aby bolo jej použitie širšie. Pokiaľ cieľom nie je tvorba natívnej iOS aplikácie ale medziplatformovej aplikácie, ktorá pobeží nielen na operačnom systéme iOS ale aj Android alebo Windows či Linux, je vhodnejšie napríklad použitie jazyka Dart v kombinácií s Flutter framework alebo JavaScript v kombinácií s React Native. [44] Kľúčovým môže byť aj predchádzajúca znalosť konkrétnej technológie.

V tejto diplomovej práci je na vývoj aplikácie zvolený programovací jazyk *Swift*. Jednak preto, že aplikácia nemá za cieľ byť spustiteľná na iných operačných systémoch ako na iOS a zároveň preto, že jeho použitie je vhodnejšie v kombinácií s natívnymi frameworks a z hľadiska výkonnosti je na tom lepšie ako Objective-C [45].

2.4.2 Frameworks

Na začiatku tejto kapitoly som bližšie popísala známy framework *CoreData*, ktorý sa používa na prácu s dátami pri vývoji iOS aplikácií. V tejto časti opíšem ďalšie frameworks, ktoré boli použité vo výslednej mobilnej aplikácií. Ako prvému sa budem venovať porovnaniu UIKit a SwiftUI.

UIKit, SwiftUI

Tieto frameworks sú hlavnými stavebnými prvkami v oblasti tvorby používateľského grafického rozhrania iOS aplikácií. *UIKit* je pôvodným nástrojom, ktorý bol predstavený v roku 2008. Je postavený na jazyku Objective-C ale kompatibilita s jazykom Swift je zabezpečená. Napísaný je imperatívnym spôsobom, čo znamená, že každý krok vytvorenia a následnej aktualizácie grafického prvku či komponentov, je popísaný sériou príkazov, ktoré systém vykoná v určitom poradí. Toto zabezpečuje, že developer má veľký stupeň kontroly nad prevedením, ale zároveň, voľne povedané, čo si nenapíše, to nemá. Dôsledkom veľkej kontroly však môže byť aj vznik mnohých chýb, ak programátor niečo zabudne explicitne naprogramovať. Tým, že sa jedná o starší framework, je k dispozícií veľké množstvo dokumentácie a zdrojov, ale aj množstvo UI vstavaných komponentov, ktoré si developer nemusí písať od začiatku. Nevýhodou môže byť to, že tým, že je starší a tradičnejší, nereflektuje na nové návrhové vzory. Napríklad nie je kompatibilný s frameworks tretích strán, ktoré podporujú reaktívne programovanie. [46]

Hlavným komponentom je *UIViewController*, ktorý má niekoľko funkcií, ktoré si programátor môže prispôsobiť tak, aby spĺňali jeho požiadavky. De-

finovať môže farby textu, pozadia, animácie, prechody, rôzne komponenty atp. Avšak môže využiť aj tzv. *Storyboards*, čo je grafický nástroj v Xcode, ktorý umožňuje programátorom jednotlivé obrazovky vysskladať v grafickom rozhraní bez písania kódu, napojiť navigáciu a to všetko v jednom súbore. Výhodou tohto prístupu je dostupnosť pre začiatočníkov a schopnosť vytvoriť prototyp za krátky čas. Nevýhodou je náročné debuggovanie či udržiavanie verzií a pri veľkej aplikácií sa súbor môže stať chaotickým a neprehľadným.

Apple predstavil *SwiftUI* v roku 2019, narozdiel od *UIKit* používa deklaratívnu syntax, ktorá je intuitívna a vychádza z moderných princípov. Developeri nemusia poznať všetky detaily implementácie jednotlivých komponentov a veľa vecí framework rieši za nich. Základným stavebným prvkom je v tomto prípade *View*. Absentuje možnosť použitia *Storyboards* a všetko musí byť napísané v kóde. Programátor ale môže použiť tzv. *Previews*, ktoré zabezpečia, že počas písania kódu sa všetky zmeny na View v reálnom čase prepisujú a vidíme ich na obrazovke. Veľkou výhodou je aj podpora reaktívneho programovania, ktorá zabezpečuje automatické aktualizácie tzv. binding objektov. Naopak nevýhodou môže byť nedostatočná kompatibilita so staršími verziami iOS (iOS 12 a mladšie), ako aj nie tak široké množstvo predpripravených komponent ako poskytuje *UIKit*. Vďaka možnosti skladania *UIKit* prvkov do *SwiftUI* za pomoci *UIHostingController* je to ale ľahko riešiteľný problém. [47]

V aplikácií tejto diplomovej práce používam kombináciu *SwiftUI* s navigáciou pomocou *UIKit ViewControllers*.

SwiftCharts

Ďalší framework, ktorý som použila bol rovnako od firmy *Apple* a slúži na vytváranie grafov. *SwiftCharts* [48] je pomerne nový framework, bol predstavený v iOS16 v roku 2022. Rovnako ako *SwiftUI* (s ktorým je kompatibilný, s *UIKit* nie) využíva deklaratívnu syntax. Základným komponentom je *Chart*, čo je *SwiftUI View*, ktoré slúži na zobrazovanie grafových dát. Následne je len na fantázií programátora a funkčných požiadavkách systému, ako tieto dáta najlepšie vizualizovať. Framework poskytuje podporu pre stĺpcové, kruhové a aj čiarové diagramy. Samozrejmosťou je aj nastavovanie x, y osí, škálovanie, nastavovanie symbolov a anotácií na grafe, prípadne ovládanie pomocou gest, ktoré sú typické aj pre ostatné *SwiftUI Views*. Dáta, ktoré graf vykresľuje môžu byť dátového typu *Int, String, Double, Date, a Decimal*, prípadne také, ktoré spĺňajú požiadavky protokolu (v *Swift* koncept podobný interface) *Plottable*.

V aplikácií je použitý kruhový, čiarový a aj stĺpcový diagram na zobrazenie rozpočtu a transakcií.

UserNotifications

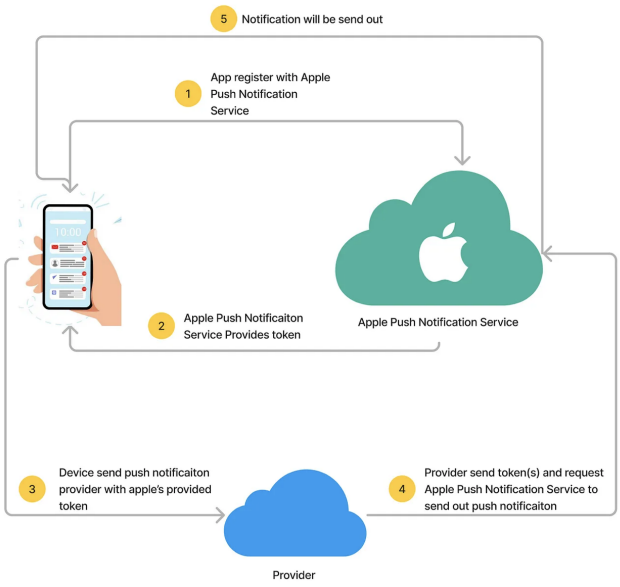
Notifikácie, či pripomienky, sú štandardnou súčasťou mnohých mobilných aplikácií. Rozlišujeme dva typy, prvým sú vzdialené tzv. *Push notifikácie*, ktoré aplikácia prijíma zo vzdialeného servera a druhou kategóriou sú lokálne notifikácie, ktorých vytvorenie a nastavenie nie je závislé od žiadnej externej inštancie a typicky reagujú na zmenu lokality alebo času.

Prvým krokom, aby zariadenie mohlo prijímať notifikácie je získanie súhlasu od používateľa. Zväčša pri prvom spustení aplikácie sa zobrazí na obrazovke zariadenia dialógové okno, ktoré používateľ buď potvrdí alebo zamietne. Po získaní súhlasu so zasielaním notifikácií je v nastaveniach ďalej možné zmeniť typ notifikácií, ktoré chce používateľ dostávať – odznak na ikone aplikácie, banner, notifikačné centrum, prípadné nastavenie zvukov notifikácie.

Pre lokálne notifikácie je možné nastaviť opakovanie po určitom čase, napr. každý piatok o 8:00, alebo po uplynutí konkrétneho časového intervalu. Zároveň programátor môže zabezpečiť ich manuálne zrušenie, keď ich zasielanie už nie je relevantné, nastaviť nadpis a telo notifikácie, prípadne pridať ikonu. Po kliknutí na notifikáciu sa používateľovi otvorí aplikácia a opäť je možné nastaviť na akej obrazovke sa otvorí. [49]

Postup pre nastavenie *Push notifikácií* je zložitejší. V prvom rade, je nevyhnutné mať developerský účet od *Apple*, ktorý stojí 99$ na rok. Bez toho sa v Xcode ani nedá funkcionalita *Push notifikácií* povoliť. Nasleduje krok pre registrovanie zariadenia, prebehne komunikácia so službou Apple Push Notification service (APNs) [50], ktorá zabezpečuje doručenie notifikácií na zariadenia používateľa. Výsledkom tejto komunikácie je unikátny token zariadenia a ďalšie konfigurácie nevyhnutné pre fungovanie notifikácií. Tento token sa musí dostať na vzdialený server, ktorým chceme ovládať vytváranie a časovanie notifikácií. Môže ísť o službu tretích strán napríklad populárne je použitie *Firebase*, alebo o vlastné riešenie. Dôležité je aj ukladanie tokenov v prípade, ak chceme notifikácie cieliť na konkrétnych používateľov, je nevyhnutné ich ukladať do zabezpečeného úložiska spolu s napríklad užívateľským menom, alebo emailom ako identifikátorom v našej doméne. Keď chceme zaslať notifikáciu, vytvoríme request a zašleme ho do APNs. Tento request musí minimálne obsahovať spomínaný token a obsah zasielanej notifikácie vo forme JSON objektu. [51] Obrázok 2.3 zachytáva popisovaný postup.

Doručenie notifikácie závisí od viacerých faktorov. Notifikácia sa zobrazí podľa zvolených preferencií používateľa a toho, či je aplikácia práve spustená. Môže dôjsť aj k oneskoreniu doručenia v prípade chýb alebo nedostupnosti služby, či vplyvom vypnutého zariadenia. Vplyv na doručenie môže mať aj zapnutý režim „Nerušiť“ alebo iné používateľom vytvorené obmedzenia. Narozdiel od lokálnych notifikácií sa nedá nastaviť presný čas doručenia notifikácie.



Obr. 2.3: Postup pri nastavovaní push notifikácií [52]

Swift Package manager – SPM

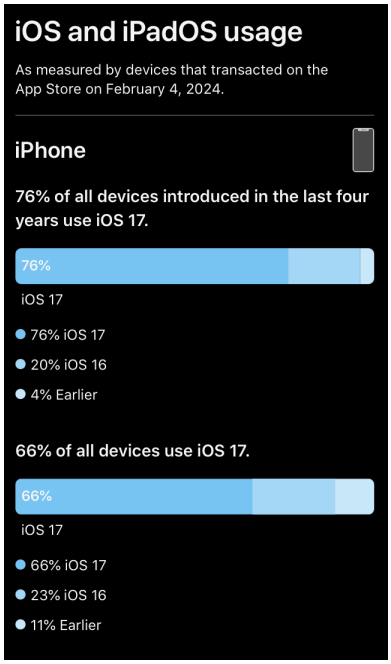
Pri robustnejších systémoch s mnohými závislosťami s rôznymi verziami je vhodné využiť nástroj na spravovanie a integrovanie externých knižníc a závislostí. Jedným príkladom je *Swift Package manager*, natívne riešenie zakomponované v Xcode. To je aj jeho hlavnou výhodou. Používateľ si jednoducho v rozhraní vyberie framework alebo externú závislosť, ktorú chce integrovať a zbytok zabezpečí SPM, vrátane použitia správnej verzie podľa preferencií používateľa. Nevýhodou oproti iným manažérom závislostí ako sú napríklad *CocoaPods či Carthage* je menšie množstvo dostupných závislostí, ktoré sú integrovateľné pomocou SPM, či výlučná podpora Swift projektov (nepodporuje Objective-C). [53]

Pomocou SPM sú v tejto aplikácií integrované externé závislosti na GitHub repozitáre, napríklad **ACKategories** `https://github.com/AckeeCZ/ACKategories`, ktorý obsahuje rôzne komunitou často používané rozšírenia na prácu s dátovými typmi, s *UIKit* komponentami, čí napríklad inicializáciu farieb pomocou HEX hodnôt atp.

Verzia iOS

Pri vytváraní mobilnej aplikácie pre operačný systém iOS je dôležité zvážiť aj verziu operačného systému, ktorú aplikácia bude podporovať. Rôzne verzie iOS môžu mať odlišné funkcie, obmedzenia a podporu pre určité technológie, či knižnice. Vo všeobecnosti platí, alebo by malo platiť, že novšie verzie sú

spravidla optimalizovanejšie a bezpečnejšie. Prirodzene chceme, aby čo najviac potencionálnych používateľov našej aplikácie zvolený podporovaný operačný systém mali nainštalovaný na zariadeniach. Štatistiky, ktoré *Apple* za týmto účelom poskytuje, v čase písania práce naznačovali, že viac ako 65% zariadení používa operačný systém iOS17. Presné štatistiky sú na obrázku 2.4. Z obrázku ďalej vyplýva, že zariadenia zakúpené v posledných štyroch rokoch majú ešte väčšiu podporu (75%) tejto verzie OS.



Obr. 2.4: Štatistiky používania iOS [54]

V aplikácií som zvolila verziu **iOS17** ako najnižšiu podporovanú verziu operačného systému, pretože počet používateľov, ktorí majú nainštalovaný práve iOS17 je vysoký a očakávam, že bude rásť najmä s príchodom iOS18 v septembri 2024. Oproti starším verziám je použitie najaktuálnejšej vždy bezpečnejšie a má väčšiu podporu od *Apple* ako aj kompatibilitu s novými frameworks ako napríklad *SwiftData*.

2.4.3 Architektúra

Tak ako budova musí byť postavená na dobrých základoch a šetrenie na materiáloch, či obchádzanie niektorých procesov sa zväčša nevypláca, tak to isté platí pri architektúre mobilných aplikácií či informačných systémov vo všeobecnosti. Dobre navrhnutý systém s použitím zaužívaných architektonických vzorov či princípov by sa následne mal ľahšie škálovať, udržiavať a aj

testovať. Známy je akronym **SOLID** [55], ktorý reprezentuje 5 zaužívaných princípov ako pristupovať k architektúre a kódu samotnému.

1. **S**ingle Responsibility – Princíp jednej zodpovednosti – táto zásada hovorí o tom, že každá väčšia časť kódu – funkcia, trieda či modul, by mala niesť zodpovednosť len za jednu úlohu či funkcionalitu.

2. **O**pen/Closed – Otvorený pre rozšírenie, uzavretý pre modifikovanie – Prí vývoji softvéru platí, pokiaľ to nie je chyba a funguje to, tak to neopravuj. Tento princíp hovorí o tom, že jednotlivé entity by mali byť rozšíriteľné takým spôsobom, aby nebolo nevyhnutné meniť a modifikovať existujúci kód.

3. **L**iskov Substitution – Tento princíp je dôležitý najmä v kontexte dedičnosti alebo polymorfizmu v objektovo orientovaných jazykoch. Objekty podtypu by mali byť schopné nahradiť objekty nadtypu bez toho, aby to ovplyvnilo správanie programu. To znamená, že podtypy musia implementovať všetky funkcie nadtypu, môžu ale navyše poskytovať aj vlastné špecifické funkcie.

4. **I**nterface Segregation – Rozdeľovanie do rozhraní – V súvislosti s prvým princípom, je tento zásadný pre správny návrh rozhraní, ktoré by nemali byť nadmerne veľké a obsahovať príliš veľa funkcií. Naopak, mali by byť dostatočne špecifické, malé a orientované na potreby svojich klientov. Programovanie oproti rozhraniu a nie konkrétnej implementácii znižuje previazanosť (*anglicky coupling*) medzi rôznymi časťami systému. To znamená, že zmeny v jednej časti systému nebudú mať taký veľký dopad na ostatné časti. Tento prístup tiež umožňuje vymeniť implementáciu za inú bez toho, aby bolo nutné meniť celý kód, ktorý na ňu odkazuje, čo vedie k vyššej flexibilite a udržateľnosti kódu. Minimalizovanie závislosti, vedie k zvýšenej modularite a robí ho testovateľnejším a flexibilnejším.

5. **D**ependency Inversion – Spomínaný pojem *previazanosť*, súvisí aj s ďalším princípom, a to princípom reverzného zapojenia závislostí. Moduly v aplikáciách, či väčších systémoch, spolu neodmysliteľne potrebujú interagovať a vymieňať si dáta. Tomu sa vyhnúť nedá, avšak tento princíp hovorí, že jednotlivé moduly by nemali byť závislé na konkrétnych implementáciách iných modulov, ale na abstrakcií a využívať rozhrania na odstienenie. Tento princíp tiež podporuje používanie *Dependency Injection*, ktoré je jedným z najčastejších spôsobov implementácie tohto princípu. Pomocou *Dependency Injection* sa závislosti poskytujú externými zdrojmi, čo umožňuje získať inštancie objektov, ktoré spĺňajú dané rozhrania a abstrakcie.

Vo svojej podstate všetky tieto princípy určitým spôsobom aplikujú aj jedny z najpoužívanejších architektonických vzorov v oblasti mobilných ap-

likácií. Patrí medzi ne **Model View Controller (MVC)**, jeho variácia **Model View ViewModel (MVVM)**, **The Composable Architecture (TCA)** alebo tzv. **Coordinator pattern**.

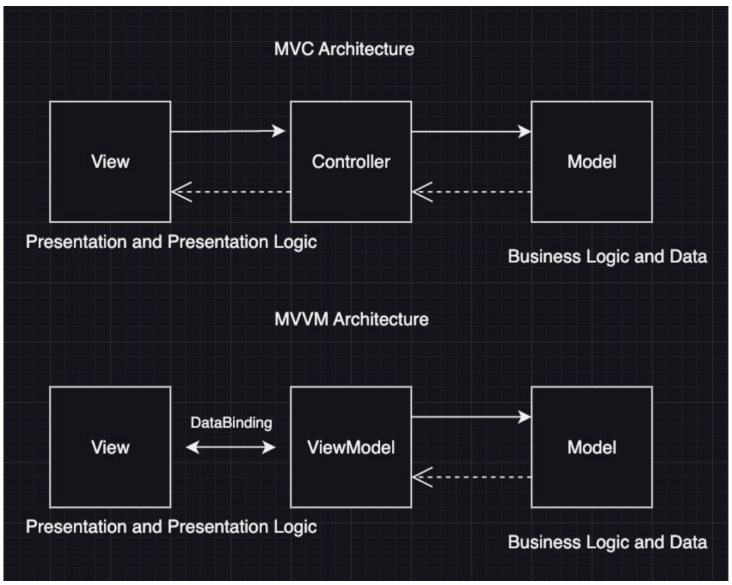
Model View Controller (MVC), Model View ViewModel (MVVM)

Architektonický vzor MVC rozdeľuje aplikáciu na tri časti - Model, View a Controller. Každá časť zodpovedá za určitú funkcionalitu aplikácie. Model má na starosti všetky úkony, ktoré sa týkajú dátovej logiky. To zahŕňa komunikáciu s databázovými objektami, validáciu vstupov, či samotnú biznisovú logiku aplikácie. View obsahuje všetky UI komponenty, je zodpovedný za logiku zobrazovania dát z Modelu a prijíma používateľský vstup. Controller stojí medzi nimi ako rozhranie a sprostredkúva ich komunikáciu – obdrží používateľský vstup z View, spracuje ho a komunikuje s Modelom. Táto interakcia vedie k zmene Modelu, a túto zmenu opäť sprostredkuje na View. Nevýhodou tohto vzoru je veľký dôraz a sústredenie na samotný Controller, ktorý vplyvom zväčšovania sa aplikácie môže narásť do veľkých rozmerov a postupne obsahovať aj časti biznis logiky, čo jednak porušuje *Single responsibility principle*, ale aj zhoršuje testovanie či údržbu. [56]

Model View ViewModel nahrádza Controller časťou ViewModel. Hlavný rozdiel medzi MVVM a MVC je práve v tejto časti, čiže v spôsobe, ako sú spravované interakcie medzi View a Modelom. V časti ViewModel sa nachádza aktuálny stav dát, ktorý je zobrazený na View. Pomocou metód reaktívneho programovania alebo tzv. *binding frameworks*, sa všetky zmeny dát automaticky synchronizujú (viažu) s ich vizuálnou reprezentáciou. Vďaka tomuto dochádza k zamedzeniu úzkeho previazania medzi biznisovou a prezentačnou vrstvou aplikácie. ViewModel sa stará o prípravu dát a logiku zobrazenia, zatiaľ čo View sa stará len o zobrazenie týchto dát. To vedie aj k lepšej testovateľnosti nakoľko ViewModel je nezávislý na konkrétnej implementácii View a môže byť testovaný nezávisle od neho [57]. Naopak Controller je priamo závislý na View a Model a môže byť ťažké ho testovať bez prítomnosti týchto komponentov. Na obrázku 2.5 môžeme vidieť porovnanie interakcie jednotlivých častí v MVC a MVVM.

Coordinator pattern

Ako bolo spomenuté vyššie, v zväčšujúcich sa aplikáciách často nastáva problém robustného Controlleru. Jeden aspekt, ktorý je toho príčinou, je navigácia a tok spojený s jednotlivými obrazovkami mobilnej aplikácie. Na každej z nich môže byť hneď niekoľko možností ako prejsť na inú obrazovku. Toto nevyhnutne vytvára v samotnom View závislosť a previazanosť na konkrétny Controller, nehovoriac o porušení *Single responsibility principle*, kde View je zodpovedné nielen za zobrazovanie dát, ale aj navigáciu medzi časťami aplikácie.



Obr. 2.5: Diagram pre porovnanie MVC a MVVM [58]

Koordinátorový vzor (anglicky *Coordinator pattern*) je architektonický vzor používaný v mobilných aplikáciách a iných softvérových systémoch na riadenie navigácie a toku aplikácie. Hlavným cieľom tohto vzoru je oddelenie navigačnej logiky od ostatných častí aplikácie. Tento vzor často pracuje s metódami push a pop. Keď používateľ vykoná nejakú akciu, ako je napríklad kliknutie na tlačidlo, koordinátor určí, aká obrazovka má byť zobrazená ďalej a prevezme navigačnú logiku potrebnú na presun na túto obrazovku. [59]

The Composable Architecture (TCA)

Do kontrastu iný prístup k rozdeleniu aplikácie poskytuje framework TCA [60], postavený na základoch funkcionálneho programovania, za vývojom ktorého stojí **Brandon Williams** a **Stephen Celis**, ktorí sú známi z platformy `http:/pointfree.co`. Jedná sa o open-source knižnicu, ktorá obsahuje mnohé nástroje, ktoré programátorov vedú k tomu, ako štruktúrovať kód do určitých častí tak, aby výsledkom bol dobre testovateľný a modulárny kód. Zatiaľ čo MVC alebo MVVM sú aplikovateľné na rôzne platformy, TCA je knižnica napísaná v jazyku *Swift* a výlučne navrhnutá na integráciu so *SwiftUI* (alebo *UIKit*) od *Apple* a tým pádom je jej použitie limitované.

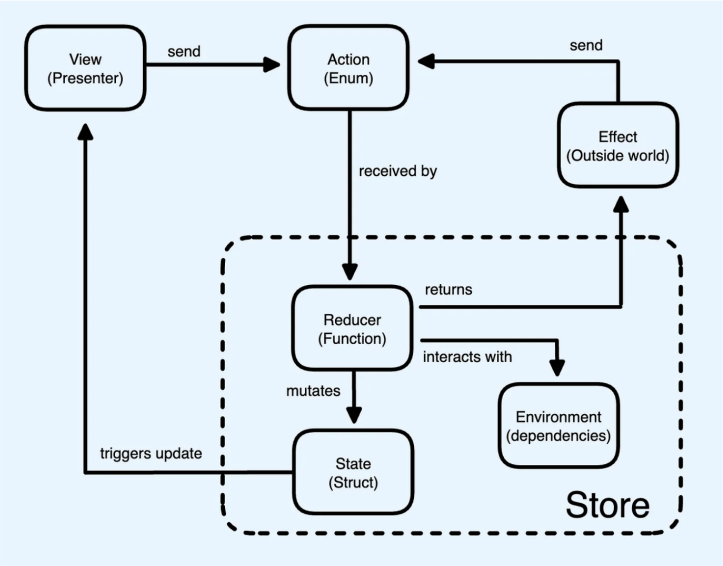
Každý modul alebo funkcionalita aplikácie je tvorená niekoľkými hlavnými komponentmi. [61]

- **Stav (State)** obsahuje všetky premenné, dáta, ktoré charakterizujú aktuálny stav danej obrazovky.

- **Akcie (Actions)** definujú všetky udalosti alebo funkcionality, ktoré sú vykonateľné a svojím spôsobom menia Stav aplikácie.
- Ďalším dôležitým komponentom je **Reduktor (Reducer)**, ktorý ako vstup, prijíma aktuálny stav a akciu a produkuje nový stav ako výstup. Je zodpovedný za spracovanie akcií a aktualizáciu Stavu aplikácie na základe týchto akcií.
- Výsledkom akcie nemusí byť nevyhnutne len zmena Stavu, ale aj **Efekt (Effect)**, ktorý akcia môže vyvolať – väčšinou predstavuje ďalšiu asynchrónnu akciu.
- Jednotlivé moduly nie sú v aplikácií izolované, ale je nevyhnutné doplniť aj **Závislosti** (**Environment**), ktoré predstavujú ďalší komponent v tejto architektúre, ktorý umožňuje pristupovať k vonkajším zdrojom a prostriedkom.
- Všetky tieto časti sú dostupné v komponente **Úložisko (Store)**, ktorý ich spája dohromady a umožňuje im spolupracovať na správe stavu aplikácie. Je zodpovedný za spracovanie akcií, aktualizáciu stavu a riadenie toku dát v aplikácii alebo jej časti.

Prezentačná logika modulu je obsiahnutá vo View, ktoré ma odkaz na Store, a tým pádom prístup ku všetkým komponentom. Nakoľko každá funkcionalita je oddeliteľná od ostatných jej testovateľnosť je priamočiara a jednoduchá. Následná kompozícia do celku je závislá na správnom nastavení reduktorov a úložísk. Výsledná hierarchia aplikácie je grafová. V koreni sa nachádza hlavný *AppStore*, obsahujúci *AppState, AppAction a AppReducer*. Pod hlavným *AppStore* môžu existovať rôzne moduly alebo časti aplikácie (potomkovia), ktoré zodpovedajú za rôzne funkcionality alebo obrazovky aplikácie. Všetci potomkovia obsahujú subkomponenty, ktoré obsahujú stav, akcie a reduktory pre danú funkcionalitu. Tieto komponenty sú integrované do seba pomocou zdieľania stavu a definovania akcií, ktoré môžu ovplyvniť tento stav. Akcie sú následne spracované reduktormi, ktoré aktualizujú stav aplikácie na základe prijatých akcií. Tok dát je riadený pomocou toho, ako sú definované akcie a reduktory v jednotlivých komponentoch. Akcia môže byť vyvolaná v jednej časti aplikácie a následne spracovaná reduktorom v inej časti, čo umožňuje komunikáciu a prenos dát medzi rôznymi celkami aplikácie. Výsledok je smerovaný smerom do hlavného *AppStore* – do koreňa. [62]

Tento postup je pomerne zložitý a pre menšie a jednoduchšie aplikácie zbytočne robustný a zdĺhavý. Naopak pre veľké aplikácie predstavuje výhody v jednoducho prevediteľnej škálovateľnosti, modularite a testovateľnosti. Kód je prehľadný a jasne štrukturovaný. Z popisuje je zrejmé, že tento prístup aplikuje všetky vyššie popísané SOLID princípy tvorby softvérového produktu.



Obr. 2.6: Diagram znázorňujúci fungovanie TCA [63]

Zvolená architektúra pre aplikáciu tejto diplomovej práce je MVVM s použitím návrhového vzoru Coordinator. Použitie TCA v tomto kontexte považujem za príliš robustné vzhľadom na veľkosť aplikácie a nakoľko aplikácia bude obsahovať značné množstvo biznis logiky, použitím MVC by mohlo dôjsť k spomínanému zahlteniu Controllera. MVVM umožní oddelenie logiky do ViewModel, čo bude mať za dôsledok lepšie možnosti otestovania jednotlivých častí.

Zhrnutie

Jazyk implementácie – Swift
Verzia OS – iOS17
Frameworks – ACKategories, Snapshot-Testing integrované pomocou SPM
Apple Frameworks – SwiftData, SwiftCharts, UIKit, SwiftUI, UserNotifications, Foundation, Observation, XCTest
Architektúra – MVVM s použitím návrhového vzoru Coordinator

Kapitola 3

Implementácia

V tejto kapitole sa bližšie venujem niektorým implementačným detailom, ktoré boli použité v kóde aplikácie.

3.1 Implementácia MVVM

Na konkrétnom príklade funkcionality zoznamu finančných cieľov predstavím, ako je implementovaná architektúra MVVM. Všetky ostatné funkcionality ako napríklad zoznam transakcií, pridanie transakcie, vytvorenie rozpočtu, nastavenia, majú obdobnú štruktúru a členenie kódu.

3.1.1 Štruktúra kódu

Súbory sú rozdelené podľa logických celkov do priečinkov. Každý priečinok obsahuje aspoň tri hlavné súbory. Súbor **View*, v ktorom sa nachádza definícia používateľského rozhrania vytvoreného pomocou *SwiftUI* a binding premenná na ViewModel, ktorá slúži na riadenie zobrazovaných dát a interakcií medzi používateľom a aplikáciou. Konkrétny ViewModel je vždy definovaný pomocou vzoru (protokolu). Protokol obsahuje definície premenných a funkcií, ktoré predstavujú minimálne nároky pre konkrétny ViewModel. Pre zoznam finančných cieľov existuje protokol *FinanceGoalsViewModeling*, kde je pre každú premennú definovaný jej názov, dátový typ a prístup (`get, set`). Samotný ViewModel obsahuje okrem týchto premenných a implementovaných funkcií z *FinanceGoalsViewModelingActions* aj iné `private` premenné či funkcie, ktoré sú zodpovedné za biznis logiku danej funkcionality.

Ak má používateľ po kliknutí na tlačidlo na obrazovke možnosť prejsť na ďalšiu obrazovku, súbor obsahuje aj definíciu *FlowDelegate*, ktorý je súčasťou vzoru *Coordinator* a obsahuje definície metód zodpovedných za riadenie toku navigácie. Tieto metódy sú následne implementované v príslušnom *FlowCoordinator*, ktorý opisujem v ďalšej časti. Posledným hlavným súborom v priečinku je **ViewController*, ktorý vytvorí zo *SwiftUI* View s príslušným

ViewModel *UIViewController* (*UIKit*), a to vložením do *UIHostingController*. Použitie tohto postupu je typické v situácií keď chceme spolu použiť *SwiftUI* a *UIKit*.

Na obrázku 3.1 je zobrazené ako vyzerá definícia protokolu pre *FinanceViewModel*, ako aj definícia *FinanceGoalsFlowDelegate*.

```
public protocol FinanceGoalsFlowDelegate: AnyObject {
    func newGoal()
    func addToGoal(goal: FinanceGoal)
}

public protocol FinanceGoalsViewModeling {
    var actions: FinanceGoalsViewModelingActions { get }

    var duration: Duration { get set }
    var monthlyGoals: [FinanceGoal] { get }
    var annualGoals: [FinanceGoal] { get }
    var isPressing: Bool { get set }
    var hasNoGoals: Bool { get }
    var currency: String { get }
}

public protocol FinanceGoalsViewModelingActions {
    func newGoal()
    func addToGoal(goal: FinanceGoal)
    func fetchData()
    func delete(goal: FinanceGoal)
}

public extension FinanceGoalsViewModeling where Self: FinanceGoalsViewModelingActions {
    var actions: FinanceGoalsViewModelingActions { self }
}
```

Obr. 3.1: Definícia FinanceViewModel a FinanceGoalsFlowDelegate v jazyku Swift

Dátová časť sa nachádza v samostatnom priečinku. Model pre *FinanceGoal* ako databázovú entitu je v súbore, kde je definovaná trieda s anotáciou `@Model` obsahujúca konštruktor a premenné. Týmto spôsobom sa veľmi jednoducho v *SwiftData* definujú tabuľky a o zbytok sa postará framework. Manipuláciu s databázou má na starosti `DatabaseService`, ktorá je implementovaná pomocou *Dependency Injection*, ktorú popisujem v časti 3.6.

Aby *View* reflektovalo na zmeny, ktoré sú na *Model*, musí tieto zmeny realizované ViewModel „sledovať“. Toto je zabezpečené pomocou *ObservableObject protokolu* a `@ObservedObject`, ktoré sú súčasťou *SwiftUI* framework. Objekt, ktorý implementuje *ObservableObject*, môže obsahovať premenné označené `@Published`, čo signalizuje, že ak sa hodnota týchto premenných zmení, má upozorniť všetky Views, ktoré objekt sledujú, na ich aktualizáciu.

`@ObservedObject` je obal (*anglicky property wrapper*), ktorý sa používa vo Views na sledovanie zmien premenných, ktoré sú definované v *ObservableObject*. Ak chceme sledovať zmeny v objekte implementujúcom *ObservableObject*, označíme inštanciu tohto objektu ako `@ObservedObject` vo View, ktoré je závislé na týchto dátach. Pri použití `@ObservedObject` SwiftUI automaticky

sleduje zmeny v *ObservableObject* a vyvoláva aktualizácie v UI, keď zmeny nastanú.

Apple v iOS17 predstavil framework *Observation* [64], ktorý veľkú časť tejto logiky zastieňuje a zjednodušuje. Namiesto použitia protokolu *ObservableObject* použijeme property wrapper nad ViewModel `@Observable`, všetky `@Published` property wrappers z ViewModelu odstránime a chovanie zostane zachované s ušetrenými riadkami kódu. [65] Táto implementácia je použitá aj v aplikácií.

Nakoľko ide o mladý framework (predstavený na jeseň 2023), niektoré koncepty s jeho použitím nie sú kompatibilné a musia byť implementované pôvodným spôsobom. Problém spôsobuje použitie premennej *UserDefaults*, kde pomocou property wrapper `@AppStorage` sme čítanie a následný zápis mali s automatickými aktualizáciami, ale to pri použití `@Observable` nie je na *ViewModel* zatiaľ možné.

3.2 Implementácia vzoru Coordinator

FlowDelegate predstavuje definíciu metód, ktoré zahajujú navigáciu na ďalšie obrazovky. Na počiatku cyklu aplikácie je jej spustenie, ktoré riadi tzv. *AppDelegate*. Je to kľúčová trieda, ktorá poskytuje spoločný bod pre riadenie a správu behu iOS aplikácie. Táto trieda obsahuje *AppFlowCoordinator*, ktorý naštartuje navigáciu, konkrétne v tejto aplikácií definíciu a spustenie obrazoviek, ktoré sú v dolnom navigačnom paneli – TabBar. Na väčšinu logiky navigácie je použitá závislosť *ACKategories*, ktorá poskytuje metódy na vytvorenie navigačného toku alebo pridanie či odstránenie z neho. Funkcionalita *FinanceGoals* je prístupná z navigačného panelu. Jej hlavná funkcia `start()` obsahuje definíciu ikony a popisu z navigačného panelu a nadpisu obrazovky. ViewController, ktorý je vytvorený pomocou *UIHostingController* je vložený do *UINavigationController* a ten do TabBar-u. Časť tohto popisu je možné vidieť na obrázku 3.2. Definície metódy `start()` môžu byť odlišné v závislosti od spôsobu začatia toku (na celú obrazovku, modálne okno, pridanie do existujúceho toku atp.)

Za zaujímavejšie považujem implementáciu definícií, ktoré som opísala pri popise MVVM a FlowDelegate. V prípade FinanceGoals, ide o metódy `func newGoal(), func addToGoal(goal: FinanceGoal)`, z obrázku 3.1. Ich implementácia, ktorú je vidieť na obrázku 3.3 presne popisuje, že obrazovka, kde si používateľ k cieľu zaznamená čiastku je naprezentovaná ako tzv. „sheet“ z dolnej časti obrazovky, kým obrazovka s pridaním nového finančného cieľa je zobrazená na celú obrazovku. Ich zavretie a potvrdenie je definované v ich príslušných FlowCoordinators.

Implementovaním tohto návrhového vzoru sa značné množstvo navigačnej logiky presunie z ViewModel, kde zostáva len krátka metóda, volajúca konkrétnu metódu na FlowDelegate.

```
import ACKategories

final class FinanceGoalsFlowCoordinator: Base.FlowCoordinatorNoDeepLink {

    // MARK: - Base.FlowCoordinator methods

    override func start() -> UIViewController {
        super.start()

        let vc = createFinanceGoalsVC(
            flowDelegate: self
        )
        vc.title = "Goals".localized()

        let navVC = UINavigationController(
            rootViewController: vc
        )

        navVC.tabBarItem.title = "Goals".localized()
        navVC.tabBarItem.image = UIImage(systemName: "sparkles")

        navigationController = navVC
        rootViewController = navVC

        return navVC
    }
}
```

Obr. 3.2: Ukážka definície Coordinator triedy – FinanceGoalsFlowCoordinator

3.3 Notifikácie

V aplikácií sú použité lokálne notifikácie, používateľ ich môže povoliť na obrazovke *Notifications*. Možnosti sú dve – upozornenia o stave rozpočtu, ktoré sú používateľovi doručené každý piatok o 8:00 ráno alebo notifikácie, opakujúce sa každé tri dni, ktoré pripomínajú používateľovi, aby zaznamenával svoje transakcie do aplikácie.

3.4 Lokalizácia

Aplikácia je lokalizovaná do dvoch jazykov – anglický a slovenský. Ich prepínanie je systémové a možné zmeniť pomocou aplikácie *Nastavenia*, kde je okrem iných nastavení, zoznam všetkých nainštalovaných aplikácií s ich dodatočnými konfiguráciami. V aplikácií sa potom nachádzajú dva lokalizačné súbory vo formáte `"kľúč" = "hodnota"` a podľa vybratého jazyka sa pre konkrétny kľúč vyberie príslušná hodnota.

```
extension FinanceGoalsFlowCoordinator: FinanceGoalsFlowDelegate {
    func newGoal() {
        let vc = createNewGoalVC(
            flowDelegate: self
        )

        vc.modalPresentationStyle = .fullScreen
        navigationController?.present(vc, animated: true)
    }

    func addToGoal(goal: FinanceGoal) {
        let vc = createAddToGoalVC(
            flowDelegate: self,
            goal: goal
        )

        vc.modalPresentationStyle = .pageSheet
        vc.sheetPresentationController?.detents = [
            .medium(), .large()
        ]
        vc.sheetPresentationController?.preferredCornerRadius = 40
        vc.sheetPresentationController?.prefersGrabberVisible = true

        navigationController?.present(vc, animated: true)
    }
}
```

Obr. 3.3: Implementácia metód FinanceGoalsFlowDelegate

3.5 iCloud synchronizácia

Ak chceme synchronizovať dáta s cloudovým úložiskom iCloud, je podobne ako pri push notifikáciách nevyhnutné mať developerský účet. Ich nastavenie je potom v kombinácií so *SwiftData* jednoduché. [66] V konfiguračnom súbore v Xcode povolíme schopnosť synchronizovať s iCloud a aktualizovanie na pozadí (*anglicky Backround modes - Remote notifications*). Framework *SwiftData* vyžaduje, aby všetky premenné (čiže stĺpce v tabuľke) boli *Optional* tj. mohli nadobúdať hodnotu `nil` alebo majú defaultnú hodnotu. Pre *User Defaults* bola synchronizácia implementovaná pomocou vlastného property wrapper a *NSUbiquitousKeyValueStore*, ktoré som spomínala v časti 2.1.1.

Synchronizácia dát aplikácie s iCloud sa dá zapnúť alebo vypnúť v systémovej aplikácií *Nastavenia*, priamo v nastaveniach AppleID účtu. Dostať sa k týmto nastaveniam nie je možné priamo odkazom z našej aplikácie, ale používateľ sa tam musí dostať sám, čo považujem za nevýhodu a používateľsky neprívetivé.

3.6 Services – Dependency Injection (DI)

Jeden zo SOLID princípov spomínaných v časti 2.4.3, hovorí, že softvér by mal byť závislý na abstrakcií a nie na konkrétnej implementácií. To je dôležité najmä z hľadiska testovania a údržby aplikácie. Tento koncept je v aplikácií implementovaný pomocou protokolov a hlavnej triedy *AppDepencency*, ktorá udržiava všetky referencie na závislosti, ktoré sa používajú naprieč aplikáciou. Táto trieda je *singleton* – existuje len jedna inštancia v celej aplikácií. Použitie demonštrujem na triede *FioAPIService*, ktorá implementuje protokol *ApiServicing*. Tento protokol obsahuje definíciu všetkých metód a premenných, ktoré trieda *FioAPIService* musí implementovať. Zároveň, keď chceme túto triedu (závislosť), ktorá obsahuje metódu na získanie dát z Fio API, použiť, do konkrétnej triedy (ViewModel) ju predáme práve vďaka využitiu DI. Definujeme ďalšie protokoly, pre *FioApiService* bude definovaný ako „HasApiDependency“ a bude obsahovať premennú `apiService: ApiServicing`. V spomínanom singletone nainštanciujeme všetky závislosti ako premenné
`lazy var apiService: ApiServicing = FioAPIService()` a následne potom v každom mieste kde potrebujeme danú závislosť, ju predáme pomocou konštruktoru s parametrom týchto závislostí. *Lazy* zabezpečí, že sa inštancia danej triedy vytvorí, až keď je zavolaná. Aby každý ViewModel vedel, aké má závislosti, je v ňom definovaný `typealias`, konkrétne pre *ImportViewModel* `typealias Dependencies = HasApiDependency`, ktorý vynúti, že v danej triede je vyžadovaná premenná `apiService: ApiServicing`, ktorá je v konštruktore potom iniciovaná z *AppDependecy* singletonu. Toto sa dá potom využiť pri testovaní, kedy cez konštruktor predáme závislosť na implementáciu protokolu, ktorá ale obsahuje testovacie dáta a objekty.

Presná definícia a implementácia je ukázaná na obrázku 3.4, kde sú spojené viaceré časti kódu z rôznych súborov. Týmto spôsobom sú ešte implementované závislosti *HasDatabaseDependency* a *HasNotificationDependency*.

3.7 Biznis logika

Logika ohľadom zaznamenávania financií je priamočiara a intuitívna. Za spomenutie stojí najmä implementácia rozpočtu. Pri každom príchode na obrazovku *Rozpočet* sa zavolá funkcia, ktorá kontroluje, že v prípade, že má používateľ nastavený rozpočet, tak sa nezmenila hodnota „mesiaca“ od posledného nastavenia. Ak sa zmenila, rozpočet zresetuje a podľa toho, či má používateľ nastavenú rovnakú hodnotu pre všetky mesiace, zvolí buď danú hodnotu, alebo zobrazuje hlášku „Nastav rozpočet“. V prípade, že v danom mesiaci, najprv zadával transakcie a až potom nastavil rozpočet, existujúce výdavkové transakcie v danom mesiaci sa po nastavení rozpočtu automaticky započítajú do hodnoty, ktorú už v mesiaci vyčerpal. V prípade editovania súm v transakciách na to rozpočet rovnako reflektuje úpravou hodnôt. Príjmové

(plusové) transakcie na rozpočet nemajú vplyv, do rozpočtu sa započítavajú len výdavkové transakcie.

```
let appDependencies = AppDependency() // Singleton AppDependency

final class AppDependency { // Všetky závislosti
    lazy var apiService: ApiServicing = FioAPIService()
    //...
}

protocol HasApiDependency { // Definuje predávanú závislosť a jej protokol
    var apiService: ApiServicing { get }
}

protocol ApiServicing { //
    func makeRequest(token: String, from: String, to: String) async throws
        -> FioApiResponse?
}

final class FioAPIService: ApiServicing {
    func makeRequest(token: String, from: String, to: String) async throws
        -> FioApiResponse?
    {
        //... implementácia
    }
}
final class ImportViewModel: ImportViewModeling, ImportViewModelingActions {
    // indikuje, aké závislosti má daná trieda (v prípade väčšieho množstva)
    typealias Dependencies = HasApiDependency

    //...

    private let apiService: ApiServicing

    public init(
        dependecies: Dependencies // dependencies: HasApiDependecy
    ) {
        self.apiService = dependecies.apiService
        //...
    }
    //...
}
```

Obr. 3.4: Ukážka implementácie závislosti FioApiService

3.7.1 Import transakcií z API

Získavanie transakcií z FioAPI prebieha volaním requestu, ktorý je vidieť aj na obrázku 1.5 v časti 1.4.1. Tento request má 3 parametre.

```
/periods/{token}/{dateFrom}/{dateTo}/transactions.json
```

Ak sa jedná o prvý import transakcií, parameter `dateFrom` je vypočítaný ako aktuálny deň mínus 90 a parameter `dateTo` je vždy aktuálny deň podľa systémového času. Parameter `token` je nahradený hodnotou, ktorú pri prvom importe zadá používateľ, alebo je uložená v *Keychain*. Po úspešnom importe sa vždy uloží hodnota identifikátora poslednej importovanej transakcie a dátum importu. Tieto hodnoty sa potom použijú pri nasledujúcom importe, aby nedošlo k importovaniu už uložených transakcií.

V prípade, že jedna z importovaných transakcií má inú menu ako je mena aplikácie, používateľ je vyzvaný na reakciu – buď zmení menu aplikácie, čím dôjde k vymazaniu existujúcich dát, alebo zastaví import.

Kapitola 4

Testovanie

Hlavný cieľom testovania je zabezpečiť a overiť, že softvér spĺňa základné nároky na bezpečnosť (nedochádza k nepovoleným operáciám ako je napríklad delenie nulou, prístup mimo alokovanú pamäť atp.), poskytuje očakávané funkcionality definované v analýze a tieto funkcionality sú implementované správne – dávajú správne výsledky. Testovanie je, a malo by byť, súčasťou celého vývoja softvérového produktu. V rámci analýzy a návrhu je to testovanie a ujasňovanie si konceptov a ich porozumenia na strane medzi zákazníkom a dodávateľom, prípadne testovanie s cieľovou skupinou a potvrdenie si záverov prototypovania grafického rozhrania. Počas vývoja je to jednak testovanie každej funkcionality zvlášť (tzv. unit testovanie) ako aj testovanie, že vykonané zmeny nerozbijú funkcionality zbytku softvéru a správne fungovanie častí softvéru je zachované (integračné testovanie). Systémové testy nastupujú pred dodaním produktu zákazníkovi na strane vývojového tímu, overí sa správne fungovanie dodávaných funkcionalít, ktoré potvrdí na svojej strane akceptačnými testami zákazník. [67]

Z pohľadu vývoja je hlavne pri väčších aplikáciách prirodzene ideálne, aby spúšťanie testov prebiehalo automaticky a v najlepšom prípade zakaždým, čo developer dokončí svoju prácu na danej funkcionalite. Za týmto účelom je možné využiť *Continuous Integration* (CI) [68], čo je metóda, ktorá spočíva v pravidelnom a automatickom integrovaní zmien z kódu do hlavného repozitára (hlavnej vetvy) verzovacieho systému. Súčasťou tejto metódy je aj spúšťanie automatických testov, ktoré odhalia problémy a chyby predtým ako sú zmeny integrované.

V tejto kapitole popisujem ako prebiehalo a ako je implementované testovanie mobilnej aplikácie – jej biznisovej logiky, či používateľského rozhrania. Ďalej opisujem ako prebiehalo testovanie s potencionálnymi používateľmi a aké závery a opravy z toho vyplynuli.

4.1 Testovanie aplikácie

4.1.1 Unit testy

Jednotlivé obrazovky a funkcionality boli dôkladne otestované pomocou natívnych unit testov a framework *XCTest*. Základom bolo využitie DI, kedy každú závislosť ako DatabaseService, či ApiService a NotificationService, som naimplementovala podľa protokolu, ale samotné metódy vracali mnou predpripravené testovacie dáta a nevolali sa žiadne requesty na API službu alebo nepristupovalo sa k údajom v databáze. Tieto implementácie som následne predala ako parameter do konštruktoru, ktorý sa volá pred každým spustením testu.

Základom všetkého je písať kód, ktorý je testovateľný. Počas tvorby unit testov som narazila na problém s aktuálnym dňom. V aplikácií niekoľkokrát volám funkciu `Date()`, ktorá vracia systémový čas. Nakoľko ale potrebujem, aby testy vždy vracali rovnakú hodnotu, musela som podobne aj túto závislosť (závislosť na systéme a aktuálnom dni) implementovať pomocou protokolu a následne zvoliť vhodnú inštanciu (*CurrentDateMock*) pri volaní testov tak, aby vždy vracala mnou definovaný deň.

Výsledné pokrytie kódu unit testami predstavuje približne 70%, čo je veľmi priaznivé číslo, keď zoberieme do úvahy to, že veľkú časť kódu tvorí definícia používateľského rozhrania ako sú farby, písmo a komponenty, alebo navigácia medzi obrazovkami.

4.1.2 UI testy

V aplikácií som implementovala aj testovanie grafického rozhrania. Pri pridaní novej funkcionality alebo zmene časti dizajnu, chceme vedieť spozorovať nechcené zmeny na obrazovkách. Na toto slúži tzv. *Snapshot testovanie*, v aplikácií je pomocou SPM integrovaná závislosť na GitHub repozitár `https://github.com/pointfreeco/swift-snapshot-testing`, za ktorým stojí rovnaká dvojica, ktorú som v rámci kapitoly o architektúre a popise framework TCA už spomínala, **Brandon Williams** a **Stephen Celis** z `http://pointfree.co`. Tento framework poskytuje oporu pre vytváranie *snapshots* (snímok obrazoviek) s vopred definovanými dátami. Spúšťanie prebieha rovnako ako pri unit testoch, s tým rozdielom, že pri každom spustení (okrem prvého) sa verzia vytvoreného snapshot-u porovnáva s tým uloženým na disku (alebo v cloude).

4.1.3 Testovanie používateľského prostredia

Testovanie používateľského rozhrania je kľúčovou fázou vývoja mobilných aplikácií, ktorá nám umožňuje získať dôležité poznatky o tom, ako používatelia interagujú s aplikáciou. Často odhalí problémy, ktoré nie sú evidentné pre programátorov alebo iných členov tímu, ktorí s aplikáciou prichádzajú do styku počas vývoja. Zisťovanie, ako sa používatelia orientujú, na čo klikajú

alebo ako dlho im trvá vykonať určitú úlohu, nám poskytuje dôležité informácie o tom, ako efektívne je navrhnuté používateľské rozhranie, či je nutné niečo pridať alebo zmeniť. Dôležitý je aj aspekt používateľskej skúsenosti – UX, čiže ako na používateľov aplikácia vplýva z emocionálneho hľadiska a ako sa cítia pri jej používaní.

Existujú dva typy testovania, prvá metóda je bez používateľov, testovanie vykonáva expert za použitia napríklad heuristiky (Nielsenova heuristika popísaná v časti 2.2.1), ktorú je dobré mať na pamäti už počas vývoja a tvorby návrhu. Druhým typom je prirodzene testovanie s používateľom. Typickým prístupom je vytvorenie testovacích scenárov, ktoré používateľovi definujú úlohy, ktoré by mal v aplikácii vykonať. Počas toho používateľa pozorujeme, a snažíme sa identifikovať miesta, kedy systém nie je pre používateľa intuitívny, alebo sa správa inak ako by očakával. Dobré je na konci zaradiť aj dotazník, ktorým sa dopýtame na spätnú väzbu, napríklad ako na používateľa aplikácia vplývala, ktorá úloha bola náročná a čo by na aplikácií vylepšil. [69]

4.1.4 Priebeh

Testovania sa zúčastnilo 10 účastníkov, ktorých úlohou bolo:

- zorientovať sa v aplikácií,
- vytvoriť ľubovoľnú transakciu,
- zmeniť menu aplikácie,
- nastaviť mesačný rozpočet,
- vytvoriť a následne zmazať ročný finančný cieľ.

Žiadny z používateľov nemal problém identifikovať miesta kde sa v aplikácií vytvára transakcia, kde sa nastavuje rozpočet alebo vytvára finančný cieľ. Pri zmene meny aplikácie boli dvaja účastníci prekvapení, že nedošlo k prepočítaniu hodnôt podľa aktuálne platného kurzu, čo značí, že nečítali upozornenie, ktoré je súčasťou potvrdzovacieho dialógu pri zmene meny.

Traja účastníci overili aj schopnosť aplikácie reagovať na chybné vstupy a do všetkých polí, kde sa zadáva hodnota (číslo), vkladali textové hodnoty, čo v jednom prípade odhalilo neošetrený vstup a vyústilo do delenia nulou. Používateľské testovanie taktiež odhalilo chybu pri využití vyhľadávacieho poľa na obrazovke s transakciami, kedy po kliknutí na „x“ v pravej časti vyhľadávacieho poľa nedošlo k zatvoreniu klávesnice a odstráneniu už zadaného textu.

Na základe spätnej väzby od účastníkov boli implementované nasledujúce zmeny:

- farebne zvýraznený typ transakcie – červenou výdavok a zelenou farbou príjem, ktorý sa nachádza v hornej časti listu s formulárom pre vytvorenie novej transakcie,
- pridanie tzv. „back gesture“, ktoré je typické pre iOS zariadenia, ale vplyvom implementácie vlastného tlačidla „Späť“, bola táto funkcionalita potlačená,
- skrátená dĺžka podržania transakcie alebo finančného cieľa pri pokuse o odstránenie a pridanie vibrácie,
- upravené tlačidlo „Potvrdiť“ na obrazovke „Pridať transakciu“, nakoľko jeho pôvodná implementácia – že je vždy nad klávesnicou, nevyhovovala väčšine používateľom.
- pridanie možnosti interagovať s grafom na obrazovke „Prehľad“, kedy keď používateľ klikne na graf a konkrétny bod, zobrazí sa mu súčet transakcií pre vybraný deň a takto môže ťahaním prsta po grafe prechádzať a vždy vidí aktuálne dáta
- na obrazovke „Rozpočet“ bol upravený text, ktorý zachytával časť rozpočtu, ktorú používateľ už vyčerpal. Používatelia to nepovažovali za dostatočne intuitívne a navrhli zmenu, aby zobrazený text zachytával sumu, ktorú ešte môže v danom mesiaci minúť

Používatelia v rámci spätnej väzby ocenili vhodné použitie farieb, implementáciu tmavého režimu a možnosť exportovať dáta do súboru CSV.

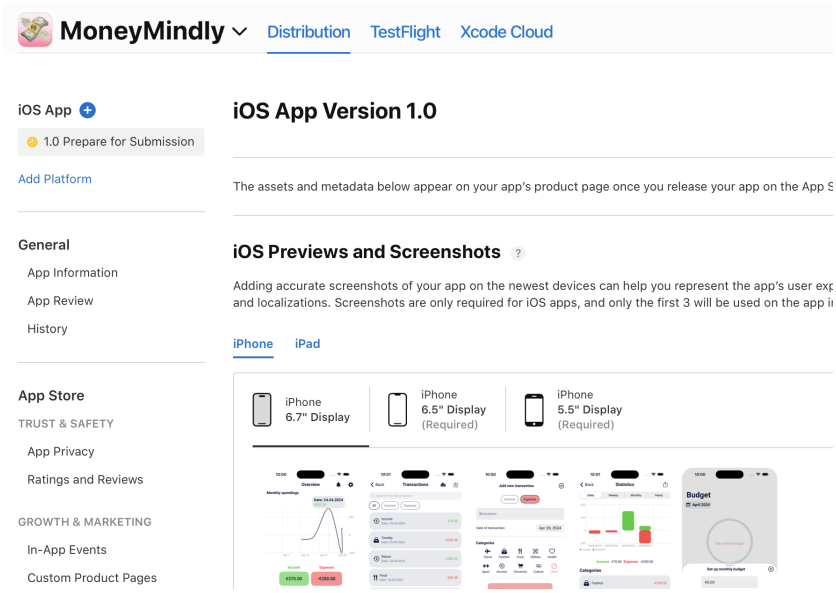
Záloha do iCloud bola náročnejšia na otestovanie, nakoľko používatelia nemali viac zariadení s rovnakým AppleID. Na tento účel testovania ale postačilo aplikáciu vymazať a znova nainštalovať. V prípade, že v aplikácií *Nastavenia*, bola iCloud synchronizácia dát aplikácie povolená, po nainštalovaní boli dáta opäť dostupné.

Podobný problém nastal pri testovaní notifikácií, nakoľko sú časovo závislé, len dvaja používatelia dostali notifikáciu v dobe testovania. Ich možnosť zapnutia ocenili ale všetci.

4.2 Publikácia v AppStore

Keď je aplikácia otestovaná, a z pohľadu vývojárov hotová, je čas ju dostať medzi reálnych používateľov. Ako už bolo spomenuté, inštalovanie aplikácií na zariadeniach s iOS sprostredkúva aplikácia *AppStore*. Prerekvizitou uverejnenia aplikácie v *AppStore* je opäť developerský účet, ktorý stojí 99$ na rok. Následne je pomocou platformy *AppStore Connect*, kde má developer prístup k všetkým jeho vytvoreným aplikáciam a informáciám o nich, nutné vytvoriť žiadosť o posúdenie aplikácie (*anglicky App review*). Tomu

predchádza dôkladné vyplnenie niekoľkých formulárov zahŕňajúcich mimo iné údaje ako názov, popis, snímky obrazoviek v rôznych veľkostiach, verziu, informácie o spoplatnení, nakladaní s osobnými údajmi atp. Aplikácia musí mať unikátny názov – v tomto prípade je názov **Money Mindly** a ikonu. Po vyplnení všetkých formalít je čas vytvoriť *build* aplikácie a nahrať ju do *AppStore Connect*. Toto prebieha napríklad v Xcode, kde sa nastaví (zdvihne) verzia, vytvorí build, archivuje sa a po potvrdení dialógového okna nahrá automaticky do Connect-u daného developerského účtu. [70] Tento proces je v prípade pravidelného nahrávania a vydávania vhodné automatizovať. Na obrázku 4.1 je vidieť formulár, ktorý obsahuje vyplnené údaje o aplikácií, ktorú nahrávame k posúdeniu.



Obr. 4.1: Formulár v AppStore Connect

Nasleduje fáza, kedy je aplikácia overovaná [71]. Proces *App Review* je naozaj dôkladný, aplikácia je preverovaná v mnohých ohľadoch vrátane dizajnu, biznisu, či právnych náležitostí. [72] Vo výsledku sú schválené len aplikácie, ktoré všetky kritéria spĺňajú a sú svojou funkcionalitou skutočne originálne a pre používateľov prínosné.

V mojom prípade som dostala dodatočnú otázku na fungovanie FioApi, testerov zaujímalo, či je platby v aplikácií po synchronizácií s účtom možné iniciovať. Po zodpovedaní na otázku proces posúdenia pokračoval a po približne štyroch hodinách bola aplikácia schválená a v čase písania tohto textu je dostupná na stiahnutie v *AppStore*, obrázok 4.2.

Obr. 4.2: Screenshot aplikácie v zozname v AppStore

Budúcnosť vývoja

Za významnú súčasť práce považujem analýzu a následné implementovanie možnosti synchronizácie platieb vďaka využitiu bankovej API služby, čo predstavuje základ pre integráciu ďalších služieb do budúcnosti, aby používatelia mohli spravovať viacero účtov z rôznych bánk z jedného miesta.

Ďalšiu možnosť ako rozšíriť rozsah mobilnej aplikácie vidím v pridaní funkcionalít, ktoré dovolia používateľom si aplikáciu personalizovať podľa ich individuálnych potrieb a preferencií. Napríklad vytváranie nových kategórií transakcií umožní používateľom lepšie štruktúrovať svoje výdavky a získať detailnejší prehľad o tom, ako spravujú svoje financie.

V súčasnej dobe je veľmi aktuálne použitie umelej inteligencie a jej predikcií, ktoré môžu poskytnúť používateľom odporúčania týkajúce sa ich financií na základe ich transakčnej histórie a správania. Napríklad navrhnúť spôsoby, ako ušetriť peniaze, investovať či plánovať veľké nákupy. Integrácia umelej inteligencie by mohla výrazne zvýšiť prínos aplikácie a poskytnúť používateľom ešte sofistikovanejší nástroj na sledovanie ich financií.

Samozrejme, priestor na zlepšenie funkcionalít je takmer nekonečný a dotýka sa aj vylepšení grafického rozhrania, rozšírenia podporovaných jazykov, implementácie tzv. widgetov, pridanie ďalších upozornení a v neposlednom rade implementovania zmien na základe spätnej väzby od nových používateľov.

Záver

Hlavným cieľom tejto diplomovej práce bolo navrhnúť a implementovať mobilnú aplikáciu pre operačný systém iOS, ktorá bude poskytovať používateľom funkcionality spojené so sledovaním osobných financií.

Výsledkom je aplikácia pre správu osobných financií, ktorá poskytuje používateľom nástroje na systematické zaznamenávanie ich finančných tokov, vytváranie prehľadných grafov a nastavovanie finančných cieľov, čo im môže pomôcť lepšie riadiť ich financie. Okrem toho, integrácia s bankovými API službami umožňuje jednoduché a bezpečné importovanie transakcií, čo uľahčuje užívateľom sledovanie ich finančných pohybov na bankovom účte. Dôležitým prínosom je aj zabezpečenie zálohovania dát v iCloud, čo zaručuje bezpečnosť a dostupnosť údajov pre používateľov. Aplikáciu považujem za úspešne implementovanú, nakoľko spĺňa definované funkčné požiadavky.

Práca dokumentuje celý proces návrhu a implementácie mobilnej aplikácie. V prvej kapitole popisujem identifikáciu požiadaviek, cieľového používateľa, prípady použitia a hodnotím konkurenčné aplikácie na trhu. V závere kapitoly som popísala koncept otvoreného bankovníctva a jeho možných aplikácií.

V ďalšej kapitole som predstavila možnosti ukladania dát a návrh používateľského rozhrania. Kapitola zahŕňa popis často používaných architektonických prístupov k tvorbe mobilných aplikácií, charakteristiku použitých technológií a externých závislostí.

V rámci kapitoly o implementácií som demonštrovala implementačné detaily a použitie konkrétnych technológií. V poslednej kapitole som sa venovala testovaniu.

Použitie každej aplikácie či metódy, ktorá pomáha sledovať osobné financie a núti nás zamýšľať sa nad tým, ako s nimi nakladáme, môže pozitívne vplývať na rozvoj finančnej gramotnosti a schopnosti chovať sa finančne zodpovedne, v čom vidím aj hlavný prínos a potenciál tejto práce.

Literatúra

[1] ALTEXSOFT. *Functional and Non-functional Requirements: Specification*. In: AltexSoft [online]. 2023-11-30. [cit. 2023-12-17]. Dostupné z: https://www.altexsoft.com/blog/functional-and-non-functional-requirements-specification-and-types/

[2] LOUISE BRUTON. *What are UX personas and what are they used for? - UX Design Institute*. In: www.uxdesigninstitute.com [online]. 2022-05-25. [cit. 2023-12-17]. Dostupné z: https://www.uxdesigninstitute.com/blog/what-are-ux-personas/

[3] DAM, Rikke a Teo SIANG. *Personas – A Simple Introduction*. In: The Interaction Design Foundation [online]. 2019-03-29. [cit. 2023-12-17]. Dostupné z: https://www.interaction-design.org/literature/article/personas-why-and-how-you-should-use-them

[4] CHAPMAN, Tom. *Why your strategy must involve anti-personas*. In: Medium [online]. 2019-06-07. [cit. 2023-12-20]. Dostupné z:https://uxplanet.org/why-your-strategy-must-involve-anti-personas-858ef1900464

[5] SOEGAARD, Mads. *How to Design Use Cases in UX*. In: The Interaction Design Foundation [online]. 2024-04-02. [cit. 2024-02-20]. Dostupné z: https://www.interaction-design.org/literature/article/use-case-ux

[6] *Spendee Money & Budget Planner*. [cit. 2023-12-23]. Dostupné z: https://www.spendee.com

[7] *Monefy: Money tracker*. [cit. 2023-12-23]. Dostupné z: https://monefy.me

[8] *Fleur*. [cit. 2023-12-23] Dostupné z: https://fleurbudget.com

[9] *Finbot*. [cit. 2023-12-23]. Dostupné z: https://finbot.eu/sk

[10] IBM. *What is an Application Programming Interface (API) — IBM*. In: www.ibm.com [online]. 2024-04-09. [cit. 2024-04-10]. Dostupné z: https://www.ibm.com/topics/api

[11] ESTEVEZ, Eric. *Open Banking*. In: Investopedia [online]. 2019. [cit. 2024-04-01]. Dostupné z: https://www.investopedia.com/terms/o/open-banking.asp

[12] MALY, Sabrina. *Multibanking explained simply - with advantages — amnis.* In: Amnis [online]. 2022-10-18. [cit. 2024-04-01]. Dostupné z: https://amnistreasury.com/multibanking-explained-simply-with-advantages/

[13] BANK, European Central. *The revised Payment Services Directive (PSD2)*. In: European Central Bank [online]. 2018-10-05. [cit. 2024-04-11]. Dostupné z: https://www.ecb.europa.eu/press/intro/mip-online/2018/html/1803_revisedpsd.en.html

[14] Tatra banka. *Nová smernica PSD2*. In: Tatra banka [online]. 2017-12-14. [cit. 2024-04-11]. Dostupné z: https://www.tatrabanka.sk/sk/blog/tlacove-spravy/vsetko-co-ste-mali-vediet-novej-smernici-psd2/

[15] EWIN, Brad. *What is open banking: Everything you need to know*. In: gocardless.com [online]. 2023. [cit. 2024-04-11]. Dostupné z:https://gocardless.com/guides/posts/open-banking/#what-are-the-benefits-of-open-banking

[16] Dokumentácia API Slovenská sporiteľna a.s. [cit. 2024-04-11]. Dostupné z: https://developers.erstegroup.com/docs/apis/bank.egb/bank.egb.v1%2Faisp

[17] Dokumentácia API Fio banka a.s. [cit. 2024-04-11]. Dostupné z: https://developers.fio.cz/swagger-ui/index.html#/Connection%20test%20controller%20V2/getCertInfoUsingGET_1

[18] Fio APi bankovnictví. [cit. 2024-04-11]. Dostupné z: https://www.fio.cz/docs/cz/API_Bankovnictvi.pdf

[19] CLARK, Hannah. *6 Stages Of The Software Development Life Cycle (SDLC)*. In: The Product Manager [online]. 2022-05-20. [cit. 2024-04-30]. Dostupné z: https://theproductmanager.com/topics/software-development-life-cycle/

[20] APPLE. *UserDefaults - Foundation — Apple Developer Documentation*. In: Apple.com [online]. 2019. [cit. 2024-03-23]. Dostupné z: https://developer.apple.com/documentation/foundation/userdefaults

[21] HUDSON, Paul. *Storing user settings with UserDefaults*. In: Hacking with Swift [online]. 2023-10-29. [cit. 2024-04-30]. Dostupné z: https://www.hackingwithswift.com/books/ios-swiftui/storing-user-settings-with-userdefaults

[22] WALKER, Eric. *Data Persistence — NSUbiquitousKeyValueStore*. In: Medium [online]. 2019-08-05. [cit. 2024-04-30]. Dostupné z: https://betterprogramming.pub/data-persistence-nsubiquitouskeyvaluestore-9cf4f97cd50c

[23] FLORIAN. *Data Persistence in iOS apps with Swift - Overview - Swift Tutorial*. In: iOS App Templates [online]. 2021-11-07. [cit. 2024-03-30]. Dostupné z: https://iosapptemplates.com/blog/ios-development/data-persistence-ios-swift

[24] ASSUMANI, Medi. *An Introduction to Core Data, Part 1*. In: Medium [online]. 2021-04-30. [cit. 2024-02-23]. Dostupné z: https://betterprogramming.pub/a-light-intro-to-core-data-part-un-e344f9d1528

[25] APPLE. *Core Data*. In: Apple Developer Documentation [online] [cit. 2024-04-30]. Dostupné z: https://developer.apple.com/documentation/coredata/

[26] KHADASE, Nayan. *SwiftData: The Future of Data Persistence in Swift*. In: Medium [online]. 2023-08-12. [cit. 2024-03-25]. Dostupné z: https://medium.com/@nayankhadase97/swiftdata-the-future-of-data-persistence-in-swift-7f23659ff0b6

[27] AMANGUPTA. *Swift Data VS Core Data*. In: Medium [online]. 2023-06-20. [cit. 2024-04-30]. Dostupné z: https://medium.com/@amangupta007/swift-data-vs-core-data-2caa5d907a8d

[28] SAIBAA, Omar. *Local Storage in iOS: Keychain*. In: Medium [online]. 2023-08-29. [cit. 2024-03-30]. Dostupné z: https://medium.com/@omar.saibaa/local-storage-in-ios-keychain-668240e2670d#:~:text=1-

[29] APPLE. *Storing Keys in the Keychain*. In: Apple Developer Documentation [online] [cit. 2024-04-30]. Dostupné z: https://developer.apple.com/documentation/security/certificate_key_and_trust_services/keys/storing_keys_in_the_keychain

[30] CORRIGAN, Stephanie. *Introduction to User Interface Design: 6 Important Principles*. In: www.flux-academy.com [online]. 2023. [cit. 2024-03-29]. Dostupné z: https://www.flux-academy.com/blog/introduction-to-user-interface-design-6-important-principles

[31] PUCHKOV, Dmytro. *UI Development Process: All You Should Know in 2024*. In: Software Development Blog & IT Tech Insights — Django Stars [online]. 2017-05-24. [cit. 2024-04-30]. Dostupné z: https://djangostars.com/blog/ui-development-flow/#:~:text=This%20process%20includes%20understanding%20the

[32] NIELSEN, Jakob. *10 Heuristics for User Interface Design*. In: Nielsen Norman Group [online]. 1994-04-24. [cit. 2024-04-20]. Dostupné z: https://www.nngroup.com/articles/ten-usability-heuristics/

[33] COURSERA. *UI vs. UX Design: What's the Difference?* In: Coursera [online]. 2023-05-17. [cit. 2024-04-30]. Dostupné z: https://www.coursera.org/articles/ui-vs-ux-design#

[34] DESHPANDE, Nikhil. *Prototyping in UI/UX Design.* In: Medium [online]. 2023-10-16. [cit. 2024-03-30]. Dostupné z: https://medium.com/@thenikhildeshpande/prototyping-in-ui-ux-design-5f73ee8fcbf8

[35] ESPOSITO, Emily. *Low-fidelity vs. high-fidelity prototyping*. In: Invisionapp.com [online]. 2018-05-29. [cit. 2024-02-20]. Dostupné z: https://www.invisionapp.com/inside-design/low-fi-vs-hi-fi-prototyping/

[36] INTERACTION DESIGN FOUNDATION. *What is Prototyping?* In: The Interaction Design Foundation [online]. 2019. [cit. 2024-02-27]. Dostupné z: https://www.interaction-design.org/literature/topics/prototyping

[37] *Mockup*. [cit. 2023-12-23]. Dostupné z: https://getmockup.app

[38] *Figma*. [cit. 2023-12-23]. Dostupné z: https://www.figma.com

[39] TOLOKNOVA, Ksenia. *Navigation patterns in mobile applications. How to make the right choice?* In: Medium [online]. 2023-12-10. [cit. 2024-04-30]. Dostupné z: https://uxdesign.cc/navigation-patterns-in-mobile-applications-how-to-make-the-right-choice-fa3c228e5097

[40] *Xcode*. [cit. 2024-04-30]. Dostupné z: https://developer.apple.com/xcode/

[41] IBM. *iOS App Development — IBM.* In: www.ibm.com [online] [cit. 2024-02-07]. Dostupné z: https://www.ibm.com/topics/ios-app-development

[42] *Swift.* [cit. 2024-04-15]. Dostupné z: https://developer.apple.com/swift/

[43] JOHNS, Robert. *5 Best iOS Programming Languages for App Development [2023].* In: Hackr.io [online]. 2023-12-14. [cit. 2024-02-15]. Dostupné z: https://hackr.io/blog/ios-programming-languages

[44] GALIYA, Jaimin. *Best iOS App Development Programming Languages 2024.* In: Radixweb [online]. 2023-12-18. [cit. 2024-02-15]. Dostupné z: https://radixweb.com/blog/best-programming-languages-for-ios-app-development

[45] INC, AltexSoft. *The Good and the Bad of Swift Programming Language.* In: Medium [online]. 2018-02-23. [cit. 2024-05-05]. Dostupné z: https://blog.prototypr.io/the-good-and-the-bad-of-swift-programming-language-2adc4b7e0d1a

[46] NDUNGU, Dedan. *SwiftUI vs UIKit: The best choice for iOS in 2024.* In: Sendbird [online] [cit. 2024-02-18]. Dostupné z: https://sendbird.com/developer/tutorials/swiftui-vs-uikit

[47] ARORA, Smriti. *Swift UI or StoryBoard??* In: Technology at Nineleaps [online]. 2022-01-28. [cit. 2024-04-05]. Dostupné z: https://medium.com/technology-nineleaps/swift-ui-or-storyboard-675ff2b40829

[48] *SwiftCharts.* [cit. 2024-04-05].m Dostupné z: https://developer.apple.com/documentation/charts

[49] APPLE. *Scheduling a notification locally from your app.* In: Apple Developer Documentation [online] [cit. 2024-03-19]. Dostupné z: https://developer.apple.com/documentation/usernotifications/scheduling-a-notification-locally-from-your-app

[50] SIWAL, Sachin. *Integrating iOS Push Notifications Using Swift.* In: Medium [online]. 2024-02-20. [cit. 2023-04-03]. Dostupné z: https://medium.com/@sachinsiwal/integrating-ios-push-notifications-using-swift-209d99862ec2

[51] SIWAL, Sachin. *Integrating iOS Push Notifications Using Swift.* In: bugfender.com [online]. 2024-02-09. [cit. 2024-03-15]. Dostupné z: https://bugfender.com/blog/ios-push-notifications/

[52] ACHSUTHAN MAHENDRAN. *How does iOS push notification work?*. In: Medium [online]. 2022-06-18. [cit. 2024-04-20]. Dostupné z:https://achsuthan.medium.com/how-does-ios-push-notification-work-bcedc1bcf37b

[53] VENTAYOL, Aleix. *Top 10 iOS Swift Libraries for 2024: Stay Ahead of the Game*. In: bugfender.com [online]. 2023-12-04. [cit. 2024-02-26]. Dostupné z: https://bugfender.com/blog/top-ios-swift-libraries/

[54] APPLE. *App Store - Support - Apple Developer*. In: developer.apple.com [online] 2024-02-04. [cit. 2024-02-26]. Dostupné z: https://developer.apple.com/support/app-store

[55] MILLINGTON, Sam. *A Solid Guide to SOLID Principles.* In: Baeldung [online]. 2019-02-05. [cit. 2024-02-26]. Dostupné z: https://www.baeldung.com/solid-principles

[56] DOBREAN, Dragos a Laura DIOSAN. *Model view controller in iOS mobile applications development*. 2019. [cit. 2024-03-01]. Dostupné z: https://doi.org/10.18293/SEKE2019-048

[57] ORLOV, Bohdan. *iOS Architecture Patterns*. In: Medium [online]. 2018-05-29. [cit. 2024-03-01]. Dostupné z: https://medium.com/ios-os-x-development/ios-architecture-patterns-ecba4c38de52

[58] KRSTANOVIĆ, Marko. *Understanding the Differences: MVC vs. MVVM*. In: BrightMarbles [online]. 2023-07-07. [cit. 2024-04-20]. Dostupné z: https://brightmarbles.io/blog/differences-between-mvc-and-mvvm/

[59] SRIVASTAVA, Shwetansh. *SwiftUI Navigation Simplified with the Coordinator Pattern*. In: Medium [online]. 2023-08-03. [cit. 2024-05-01]. Dostupné z: https://medium.com/@shwetansh_srivastava/swiftui-navigation-simplified-with-the-coordinator-pattern-67204667f398

[60] *The Composable Architecture*. In: GitHub [online]. [cit. 2024-05-01]. Dostupné z: https://github.com/pointfreeco/swift-composable-architecture

[61] RAJKUMAR, Kanagasabapathy. *Getting Started with The Composable Architecture (TCA)*. In: Medium [online]. 2024-01-07. [cit. 2024-03-03]. Dostupné z: https://sabapathy7.medium.com/getting-started-with-the-composable-architecture-tca-7369f6ee4e4d

[62] LUPICH, Dmitry. *The Composable Architecture: Swift guide to TCA*. In: Medium [online]. 2024-04-15. [cit. 2024-05-01]. Dostupné z: https://medium.com/@dmitrylupich/the-composable-architecture-swift-guide-to-tca-c3bf9b2e86ef

[63] ADEGOKE, Ade. *Exploring the Composable Architecture Framework*. In: Medium [online]. 2023-04-06. [cit. 2024-04-20]. Dostupné z: https://insight.conjure.co.uk/the-composable-architecture-2eae60963248

[64] *Observation*. [cit. 2024-04-20]. Dostupné z: https://developer.apple.com/documentation/observation

[65] FATBOBMAN. *A Deep Dive Into Observation: A New Way to Boost SwiftUI Performance — Fatbobman's Blog*. In: fatbobman.com [online]. 2023-06-19. [cit. 2024-05-01]. Dostupné z: https://fatbobman.com/en/posts/mastering-observation/

[66] HOSSAIN, Jakir. *Sync SwiftData with iCloud using CloudKit*. In: Medium [online]. 2023-10-21. [cit. 2024-03-30]. Dostupné z: https://medium.com/@jakir/sync-swiftdata-with-icloud-using-cloudkit-34764a46ba54

[67] KITNER, Radek. *Typy testování software*. In: Radek Kitner [online]. 2017-09-20. [cit. 2024-05.01]. Dostupné z:https://kitner.cz/testovani_softwaru/typy-testovani-software-trideni-testu/

[68] CODERAMA. *Continuous integration — Coderama*. In: coderama.com [online]. [cit. 2024-05-01]. Dostupné z: https://coderama.com/slovnik/continuous-integration

[69] INTERACTION DESIGN FOUNDATION. *What is Usability Testing?* In: The Interaction Design Foundation [online]. 2019. [cit. 2024-05-01]. Dostupné z: https://www.interaction-design.org/literature/topics/usability-testing

[70] KUTS, Nikolay. *How to Publish an iOS App on the App Store in 9 Steps*. In: orangesoft.co [online]. 2022-11-06. [cit. 2024-05-01]. Dostupné z: https://orangesoft.co/blog/how-to-submit-an-ios-app-to-the-app-store

[71] *App Review*. [cit. 2024-05-06]. Dostupné z: https://developer.apple.com/distribute/app-review/

[72] APPLE. *App Store Review Guidelines - Apple Developer*. In: Apple.com [online]. 2019. [cit. 2024-04-05]. Dostupné z: https://developer.apple.com/app-store/review/guidelines/

Dodatok **A**

Zoznam použitých skratiek

CSV – Comma-separated values

UC – Use case

API – Application Programming Interface

PSD2 – Payment Services Directive

AISP – Account Information Service Provider

PISP – Payment Initiation Service Provider

UI – User interface

UX – User experience

Lo-Fi – Low fidelity

Hi-Fi – High fidelity

IDE – Integrated development environment

APNs – Apple Push Notification service

SPM – Swift Package manager

DI – Dependency Injection

MVC – Model View Controller

MVVM – Model View ViewModel

TCA – The Composable Architecture

CI – Continuous Integration