VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

MOBILNÍ APLIKACE PRO ZAČÍNAJÍCÍ INVESTORY

MOBILE APPLICATION FOR BEGINNER INVESTORS

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — **MATEJ ČERNÝ**
AUTHOR

VEDOUCÍ PRÁCE — **Ing. MARTIN HRUBÝ, Ph.D.**
SUPERVISOR

BRNO 2025

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

Zadání bakalářské práce

165054

Ústav:	Ústav inteligentních systémů (UITS)
Student:	**Černý Matej**
Program:	Informační technologie
Název:	**Mobilní aplikace pro začínající investory**
Kategorie:	Elektronický obchod
Akademický rok:	2024/25

Zadání:

1. Prostudujte metodiku investování na akciových trzích (akcie, ETF). Prostudujte současné veřejně dostupné zdroje dat o akciových trzích. Prostudujte možnosti technického rozhraní na infrastrukturu akciových brokerů.
2. Navrhněte serverovou část aplikace pro zajišťování dat o trzích, jejich agregaci a zpřístupnění pro klientskou část aplikace. Navrhněte koncepci klientské mobilní aplikace (pro iOS), ve které může uživatel prohlížet aktuální stav svého portfolia (ceny akcií/ETF, složení ETF, data o aktivech, FOREX, apod).
3. Implementujte serverovou aplikaci. Implementujte klientskou aplikaci pro iOS.
4. Demonstrujte použití aplikace na modelovém portfoliu. Proveďte zhodnocení aplikace na vzorku respondentů.

Literatura:

- veřejné datové zdroje: Polygon.io, Yahoo Finance

Podrobné závazné pokyny pro vypracování práce viz https://www.fit.vut.cz/study/theses/

Vedoucí práce:	**Hrubý Martin, Ing., Ph.D.**
Vedoucí ústavu:	Kočí Radek, Ing., Ph.D.
Datum zadání:	1.11.2024
Termín pro odevzdání:	14.5.2025
Datum schválení:	4.11.2024

Abstrakt

Cieľom tejto bakalárskej práce je návrh a implementácia mobilnej aplikácie pre sledovanie investícií začiatočného investora na akciových trhoch s cieľom rozvíjať jeho vedomosti v oblasti investovania a podporovať osvedčené investičné návyky. Aplikácia pozostáva z iOS klienta implementovaného v jazyku Swift a serverovej časti, ktorá zabezpečuje získavanie, spracovanie a poskytovanie trhových dát prostredníctvom REST API. Používateľ môže sledovať stav a vývoj svojho investičného portfólia, hodnotu jednotlivých aktív a zloženie ETF fondov.

Aplikácia obsahuje aj edukačné prvky, ako sú interaktívne nápovedy, vysvetlenia základných investičných pojmov a odporúčania pre dlhodobé investovanie. Dôraz bol kladený na používateľskú prívetivosť, intuitívne rozhranie, vizuálnu konzistenciu a udržateľnosť architektúry s možnosťou budúceho rozširovania funkcionalít. Súčasťou práce je aj otestovanie aplikácie na modelovom portfóliu a vyhodnotenie používateľskej skúsenosti.

Abstract

The aim of this bachelor's thesis is to design and implement a mobile application for tracking investments by beginner investors in stock markets, with the goal of expanding their knowledge in the field of investing and promoting proven investment habits. The application consists of an iOS client implemented in Swift and a server component that handles the retrieval, processing, and serving of market data via a REST API. Users can monitor the status and performance of their investment portfolio, the value of individual assets, and the composition of ETF funds.

The application also includes educational features such as interactive guidance, explanations of basic investment concepts, and recommendations for long-term investing. Emphasis was placed on user-friendliness, an intuitive interface, visual consistency, and architectural sustainability with support for future feature expansion. The thesis also includes testing the application on a model portfolio and evaluating user experience.

Kľúčové slová

mobilná aplikácia, investovanie, akciové trhy, ETF, serverová architektúra, REST API, iOS aplikácia, Swift

Keywords

mobile application, investing, stock markets, ETF, server architecture, REST API, iOS application, Swift

Citácia

ČERNÝ, Matej. *Mobilní aplikace pro začínající investory.* Brno, 2025. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Ing. Martin Hrubý, Ph.D.

Mobilní aplikace pro začínající investory

Prehlásenie

Prehlasujem, že som celú túto bakalársku prácu vypracoval samostatne pod vedením pána Ing. Martina Hrubého, Ph.D. a v priloženom zozname som uviedol všetky potrebné literárne pramene, publikácie a ďalšie zdroje, z ktorých som čerpal.

.....
Matej Černý
12. mája 2025

Poďakovanie

Chcel by som sa poďakovať pánovi Ing. Martinovi Hrubému, Ph.D. za ochotu, odborné vedenie práce, návrh a rady z priebežných konzultácií.

Obsah

Kapitola 1

Úvod

Investovanie na akciových trhoch sa v posledných rokoch stáva čoraz dostupnejším a atraktívnejším spôsobom zhodnocovania svojich financií, najmä vďaka prudkému rozvoju digitálnych technológií a rozšíreniu mobilných aplikácií. Tento trend zásadne znižuje vstupné bariéry a otvára dvere do sveta investovania najmä pre začínajúcich investorov.

Napriek rýchlemu vývoju však mnohé existujúce riešenia vykazujú nedostatky z pohľadu použiteľnosti pre začiatočníkov. Investičné aplikácie bývajú často príliš komplexné, preplnené funkcionalitou alebo neprehľadné z pohľadu používateľa bez predchádzajúcich skúseností. Problémom môže byť aj rozdrobenosť informácií, napríklad ak má používateľ účty u viacerých brokerov a nemá k dispozícii nástroj, ktorý by mu umožnil sledovať všetky svoje investície na jednom mieste. Tieto skutočnosti vytvárajú priestor pre vznik aplikácií, ktoré kladú dôraz na jednoduchosť, prehľadnosť a zároveň ponúkajú edukačné funkcie pre rozvoj znalostí začiatočníkov.

Z hľadiska návrhu a vývoja takéhoto riešenia vznikajú viaceré výzvy, ako napríklad výber spoľahlivých a verejne dostupných dátových zdrojov, návrh dátovej architektúry umožňujúcej efektívne spracovanie a ukladanie trhových informácií, alebo výber technológií, ktoré zabezpečia stabilitu, škálovateľnosť a možnosť budúceho rozširovania. Rovnako podstatný je návrh používateľského rozhrania s dôrazom na zrozumiteľnosť, intuitívnosť a podporu správnych investičných návykov.

Táto bakalárska práca reaguje na uvedené trendy implementáciou mobilnej aplikácie pre podporu začínajúcich investorov pri sledovaní svojich investícií, pričom ich zároveň vedie k lepšiemu pochopeniu princípov investovania. Riešenie sa zameriava na jednoduchosť používania a dostupnosť relevantných informácií. V práci sú tiež identifikované hlavné požiadavky cieľovej skupiny a navrhované spôsoby ich naplnenia prostredníctvom funkčného a technologicky udržateľného riešenia. Aplikácia je doplnená o vlastnú backendovú serverovú časť, ktorá zabezpečuje spracovanie a poskytovanie finančných dát.

Kapitola 2

Teoretický základ, poskytovatelia dát a použité technológie

Táto kapitola poskytuje teoretický rámec nevyhnutný pre pochopenie problematiky investovania na akciových trhoch. Súčasne predstavuje aj technológie a dátové zdroje využívané v rámci implementácie navrhovanej aplikácie.

2.1 Základné pojmy investovania na akciových trhoch

Investovanie predstavuje proces, pri ktorom jednotlivec alokuje svoje finančné prostriedky do rôznych finančných nástrojov s cieľom ich zhodnotenia v budúcnosti. Okrem očakávania výnosu je investovanie spojené aj s nezanedbateľným stupňom rizika, ktorý závisí od zvoleného nástroja, časového horizontu a trhovej situácie [5].

Dôležitým prvkom investičného rozhodovania je aj informovanosť investora a schopnosť diverzifikovať svoje investície rozložením zdrojov do viacerých aktív s cieľom zníženia rizika. Medzi základné investičné nástroje pre začiatočníkov patria najmä akcie a ETF fondy.

Akcie

Akcia je majetkový cenný papier, ktorý reprezentuje vlastnícky podiel v akciovej spoločnosti. Vlastník akcie (akcionár) získava právo na podiel zo zisku spoločnosti (tzv. dividendu), pokiaľ daná akciová spoločnosť dividendy vypláca, a na hlasovanie na valnom zhromaždení akcionárov, v prípade, že akcionár vlastní tzv. kmeňové akcie. Dividendy predstavujú podiel na zisku spoločnosti, ktorá ho môže — v pomere vlastnených akcií — vyplácať svojim akcionárom niekoľkokrát ročne. Nie je to však pravidlom a niektoré spoločnosti sa rozhodujú zisk investovať späť do rozširovania [8]. Akcie sa delia do viacerých typov, podľa ich charakteristík [1]:

- **Kmeňové akcie** – najbežnejší typ akcií; poskytujú hlasovacie práva a možnosť podielu na zisku spoločnosti formou dividendy
- **Prioritné akcie** – primárne bez hlasovacích práv, no majú stanovené fixné dividendy a taktiež majú vyšší nárok pri likvidácií spoločnosti
- **Konvertibilné akcie** – za vopred stanovených podmienok ich možno premeniť na kmeňové akcie, vďaka čomu kombinujú fixný výnos s možnosťou zhodnotenia pri náraste ceny akcií na trhu

- **Vykupiteľné akcie** – akciová spoločnosť, ktorá ich vydala, ich môže odkúpiť späť za vopred určených podmienok a často sú spojené s pevnou dividendou pokiaľ nie sú odkúpené

Hodnota akcií je ovplyvnená viacerými faktormi, ako je napríklad úspešnosť spoločnosti, ktorá ich vydala, ale aj pomer ponuky a dopytu a inými trhovými faktormi.

ETF fondy

ETF fondy (Exchange Traded Funds) sú moderným a obľúbeným investičným nástrojom s narastajúcou popularitou medzi investormi, ktorý im umožňuje získať expozíciu voči širokému spektru aktív prostredníctvom jedného cenného papiera. Ich hlavnou výhodou je, že kombinujú vlastnosti akcií a podielových fondov — sú diverzifikované ako fondy, ale obchodované na burze ako akcie. Väčšina ETF fondov sleduje určitý index (napr. S&P 500, Dow Jones Industrial, NASDAQ 100, atď.), čo znamená, že sú pasívne spravované a ich cieľom je napodobovať výkonnosť trhu. Keďže sa ich podiely obchodujú na burze, cena jednotlivých podielov sa počas dňa mení; to nemusí byť žiaduce, ak sa ETF fond výrazne odchyľuje od výkonnosti indexu, ktorý sa ETF fond snaží napodobovať. Preto sú súčasťou ETF aj tzv. autorizovaní účastníci (typicky veľké finančné inštitúcie), ktorí ho neustále monitorujú a pri veľkých rozdieloch medzi trhovou cenou ETF a hodnotou jeho podkladových aktív využívajú arbitrážne príležitosti na vyrovnanie ceny [6].

Príklady arbitráže:

- Ak je cena ETF vyššia ako aktuálna trhová cena podkladových aktív, autorizovaný účastník (AP) nakúpi jednotlivé akcie (komponenty fondu), vytvorí z nich nové ETF jednotky a predá ich na trhu so ziskom. Tým sa zvýši ponuka ETF na trhu a jeho cena začne klesať smerom k hodnote podkladových aktív.
- Ak je cena ETF nižšia ako aktuálna trhová cena podkladových aktív, AP kúpi lacné ETF jednotky na trhu, rozbalí ich na jednotlivé akcie a tie následne predá so ziskom. Tým sa zníži ponuka ETF na trhu a jeho cena stúpne smerom k hodnote podkladových aktív.

Tento mechanizmus zabezpečuje, že cena ETF sa udržiava veľmi blízko hodnote aktív, ktoré fond drží, a tým realizuje pasívnu investičnú stratégiu. Tá spočíva v tom, že fond sa nesnaží prekonať výkonnosť trhu, ale iba kopíruje jeho vývoj. Na rozdiel od toho bežné podielové fondy často využívajú aktívnu stratégiu, pri ktorej správca fondu neustále analyzuje trh a rozhoduje sa, aké aktíva kúpiť alebo predať s cieľom dosiahnuť vyšší výnos než trh. Z dlhodobého hľadiska však táto stratégia býva spojená s vyššími nákladmi a nie vždy prináša lepšie výsledky ako pasívne investovanie.

Vďaka tejto pasívnej stratégii dosahujú výrazne nižšie poplatky v porovnaní s aktívne riadenými fondmi, čo zvyšuje ich atraktivitu najmä pri dlhodobom investovaní. ETF fondy sú zároveň veľmi transparentné — investor presne vie, do akých aktív jeho peniaze smerujú — a ich likvidita je vysoká, keďže sa obchodujú počas celého obchodného dňa na burze za aktuálnu trhovú cenu. Okrem toho sú ETF fondy dostupné aj pre menších investorov a umožňujú jednoduchým spôsobom diverzifikovať svoje portfólio bez potreby vysokej vstupnej investície.

Za potenciálnu nevýhodu však možno považovať skutočnosť, že v prípade menej likvidných ETF fondov môže dochádzať k výraznejšiemu rozdielu medzi nákupnou a predajnou cenou podielu (tzv. *spread*), čo zvyšuje transakčné náklady investora [6].

Likvidita

Likvidita je schopnosť predať aktívum rýchlo a bez výraznej straty jeho trhovej hodnoty. Vysoká likvidita znamená, že aktívum možno kedykoľvek predať za cenu blízku jeho reálnej hodnote, zatiaľ čo nízka likvidita môže spôsobiť, že investor bude nútený aktívum predať za menej výhodnú cenu, alebo bude musieť čakať dlhší čas, kým sa nájde vhodný kupca. Príkladom vysoko likvidných aktív sú akcie veľkých spoločností alebo meny (napríklad euro), zatiaľ čo nehnuteľnosti, umelecké diela alebo podiely v malých firmách patria medzi aktíva s nižšou likviditou.

Portfólio

Pod pojmom portfólio investora sa rozumie súhrn všetkých aktív, do ktorých daný jednotlivec alebo inštitúcia investoval finančné prostriedky. Zvyčajne obsahuje investície do finančných nástrojov, ako napríklad akcie či komodity, no môžu ho dopĺňať aj alternatívne aktíva, napríklad nehnuteľnosti, umelecké diela, kryptomeny a podobne. Portfólio môže mať rôzne podoby v závislosti od zvolenej investičnej stratégie, časového horizontu a miery rizika, ktorú je investor ochotný podstúpiť. Vybudovanie kvalitného a vyváženého portfólia nie je jednoduchý proces a vyžaduje jasne definované investičné ciele, dôkladný prieskum trhu a výber aktív, ktoré sú v súlade s týmito cieľmi.

Metodiky investovania

Pri budovaní kvalitného investičného portfólia je dôležité zvoliť si vhodnú, konzistentnú investičnú metodiku. Konzistentný prístup zabezpečuje, že sa minimalizuje riziko nejasností pri analýze výkonnosti portfólia, ktoré by mohli vzniknúť pri miešaní rôznych investičných stratégií. Bez jasne definovanej metodiky môže byť veľmi náročné správne vyhodnotiť, ktoré investičné rozhodnutia boli efektívne a prečo nejaké kroky zlyhávajú.

Začiatočníci veľmi často preferujú jednoduché stratégie, ktoré nevyžadujú aktívne monitorovanie trhu a zároveň predstavujú nižšiu úroveň rizika. Tieto metodiky umožňujú rýchly začiatok investovania a zároveň dávajú investorom priestor na postupné budovanie znalostí a skúseností. Vďaka tomu môžu časom prejsť k pokročilejším a personalizovanejším formám investovania, ktoré lepšie zohľadnia ich ciele.

Medzi takéto stratégie patrí napríklad **indexové investovanie**, ktoré spočíva v nákupe indexových fondov kopírujúcich vývoj časti trhu. Tým ponúka vysokú mieru diverzifikácie a je vhodné primárne pre dlhodobé investovanie. Ďalšou možnosťou je napríklad **pravidelné investovanie** (tzv. *dollar-cost averaging*), pri ktorom si investor vopred stanoví fixnú sumu, ktorú investuje v pravidelných intervaloch nezávisle od aktuálnej trhovej ceny aktíva. Tento prístup pomáha znižovať riziko nesprávneho načasovania nákupu a taktiež podporuje disciplínu investora a dlhodobé investičné návyky.

2.2 Zdroje a poskytovatelia dát

Kvalitné dáta predstavujú základ každého analytického rozhodovania. V tejto časti sú uvedené hlavné zdroje dát, ktoré aplikácia využíva. Keďže cieľom nie je vytvoriť nástroj na sledovanie trhu v reálnom čase, bol tomu prispôsobený aj výber poskytovateľov dát, ako aj voľba predplatných plánov pri platených zdrojoch.

Finančné dáta sú veľmi žiadané a väčšinou využívané na komerčné účely, preto nie je veľa kvalitných zdrojov, ktoré by poskytovali spoľahlivé, komplexné a aktuálne dáta zadarmo. V tejto práci bol zvolený kompromis medzi cenou prístupu k dátam a ich rozsahom.

Polygon.io

Ako primárny zdroj finančných dát bola zvolená služba Polygon.io, ktorá patrí medzi lídrov v tejto oblasti. Polygon.io poskytuje veľmi kvalitnú dokumentáciu, vysokú dostupnosť a vlastné trhové dáta, ktoré získava priamym napojením na burzy — na rozdiel od iných poskytovateľov, ktorí ich často získavajú od tretích strán [9].

Zvolený plán **Stocks Starter**, určený primárne pre individuálne použitie, poskytuje prístup ku všetkým americkým tickerom, neobmedzený počet API požiadaviek, historické dáta za obdobie posledných 5 rokov a trhové dáta s oneskorením len 15 minút, čo je postačujúce pre potreby tejto práce [9]. Navyše architektúra projektu je navrhnutá tak, aby bolo možné poskytovateľa dát kedykoľvek vymeniť bez potreby veľkých zásahov do viacerých častí systému, ako je podrobnejšie opísané v kapitole 3.

Yahoo Finance

Ďalším využívaným zdrojom dát je bezplatná verejná služba Yahoo Finance, ktorá patrí medzi najznámejšie online finančné portály na svete. Poskytuje prístup k základným trhovým údajom, novinkám zo sveta investovania, historickým dátam a ďalším finančným informáciám.

Dáta z Yahoo Finance sa v rámci tejto práce využívajú pre zobrazovanie informácií o hlavných častiach zloženia ETF fondov a doplnkových informácií o aktívach. Tieto informácie sú na platforme voľne dostupné, zatiaľ čo u iných poskytovateľov bývajú súčasťou drahých plánov určených primárne pre profesionálne alebo komerčné použitie. Pre účely tejto aplikácie predstavuje Yahoo Finance vhodný doplnkový zdroj dát s postačujúcou úrovňou presnosti a dostupnosti.

2.3 Technológie používané na strane servera

V tejto kapitole sú bližšie špecifikované všetky hlavné technológie využívané na strane servera, ktoré zabezpečujú chod backendovej časti aplikácie. Sú tu spomenuté princípy ich fungovania a ich prípadné výhody a nevýhody vzhľadom na požiadavky projektu.

Serverová aplikácia je implementovaná v jazyku TypeScript pre zaistenie bezpečnejšieho a rýchlejšieho vývoja vďaka typovej kontrole a automatickým nápovedám.

Serverless architektúra

Serverless architektúra predstavuje moderný a čoraz populárnejší spôsob prevádzky softvérových aplikácií. Namiesto potreby spravovať vlastné servery aplikácia využíva cloudovú infraštruktúru poskytovanú tretími stranami (v prípade tohto projektu konkrétne platformu Vercel), ktoré automaticky zabezpečujú konfiguráciu, škálovanie, dostupnosť a alokáciu zdrojov. Hlavnými výhodami serverless prístupu sú [11]:

- **Flexibilita a škálovateľnosť** – systém sa automaticky prispôsobuje aktuálnej záťaži

- **Nižšie náklady na prevádzku** – pre veľa projektov je platobný model poplatkov len za využitú výpočetnú kapacitu výhodnejší ako fixné náklady spojené s prevádzkou vlastného servera
- **Znížená komplexita** – vývojári sa nemusia starať o konfiguráciu serverov, zabezpečenie ani iné činnosti spojené s prevádzkou serverov
- **Vysoká dostupnosť** – aplikácie môžu byť jednoducho distribuované po sieti tak, aby boli čo najbližšie ku koncovým používateľom, čím sa zvyšuje ich dostupnosť a skracuje sieťová odozva

Medzi nevýhody patrí skutočnosť, že nie všetky štandardné technológie sú v tomto prostredí dostupné, a preto treba túto architektúru zohľadniť už pri vývoji aplikácie.

Nuxt

Nuxt je fullstack JavaScript framework zameraný na výkonnosť, spoľahlivosť a efektivitu pri vývoji webových aplikácií. Je postavený na open–source ekosystéme **UnJS** (unified JavaScript ecosystem), ktorý tvoria modularizované knižnice a nástroje. UnJS vychádza z filozofie Unixu, ktorá kladie dôraz na jednoduché, znovupoužiteľné, prehľadné a dobre kombinovateľné komponenty. Nuxt vďaka univerzálnosti použitia jednotlivých stavebných prvkov získal silnú komunitnú podporu, čo prispieva k jeho spoľahlivosti. Najdôležitejšou súčasťou Nuxtu z pohľadu serverovej architektúry sú balíčky **h3** a **Nitro**, ktoré zabezpečujú jeho HTTP vrstvu a serverový runtime.

Z pohľadu tohto projektu Nuxt slúži ako integračná vrstva medzi viacerými UnJS komponentami, čím urýchľuje vývoj a uľahčuje nasadzovanie do produkčného prostredia.

Nitro a h3

Nitro a h3 tvoria jadro serverového prostredia v ekosystéme UnJS, kde Nitro slúži ako výkonný a univerzálny runtime, zatiaľ čo h3 slúži ako minimalistický HTTP framework s dôrazom na jednoduchosť, ktorý poskytuje modulárne rozhranie pre prácu s HTTP požiadavkami a odpoveďami. V porovnaní s inými frameworkami, ako napríklad Express, je výrazne efektívnejší, rýchlejší a prispôsobený na použitie v moderných serverless prostrediach.

Nitro stavia na h3 a rozširuje ho o pokročilé funkcie, ako je napríklad podpora plánovaných úloh, vytváranie HTTP endpointov na základe súborovej štruktúry v projekte a ďalšie. Jedným z najväčších prínosov je však to, že jeho cieľom je byť platformovo nezávislý. Preto poskytuje abstrakciu nad rozdielnymi typmi prostredí (napr. Node.js, serverless funkcie a iné) a umožňuje vytvárať jednodnú serverovú logiku bez ohľadu na cieľovú platformu.

PostgreSQL

PostgreSQL je populárny open–source relačný databázový systém, ktorý sa vyznačuje efektivitou a širokými možnosťami rozširovania. Vďaka podpore databázových transakcií, výbornej komunitnej podpore a jednoduchej integrácií s JavaScriptovými ORM nástrojmi je častou voľbou pre viaceré webové aplikácie.

PostgreSQL taktiež podporuje komplexné dátové typy, indexovanie, replikáciu, fulltextové vyhľadávanie a veľa ďalších pokročilých funkcií. Z pohľadu tohto projektu slúži ako perzistentné úložisko finančných dát od poskytovateľov za účelom cachovania.

Drizzle ORM

Drizzle ORM je minimalistický TypeScript ORM nástroj navrhnutý s dôrazom na výkonnosť a jednoduchosť použitia. Na rozdiel od iných riešení, zachováva syntax veľmi podobnú čistému SQL, čo umožňuje vývojárom začať s jeho používaním bez nutnosti študovať novú vrstvu abstrakcie. Drizzle automaticky generuje typy databázovej schémy pre TypeScript z jej definície, čím znižuje počet chýb počas vývoja a zlepšuje produktivitu. Vďaka Drizzle je spracovanie databázových operácií na strane servera efektívnejšie, bezpečnejšie a udržateľnejšie.

Návrhový vzor repository

Návrhový vzor *repository* slúži na oddelenie aplikačnej logiky od dátovej vrstvy. Umožňuje abstrahovať konkrétny spôsob prístupu k dátam, čím zvyšuje udržateľnosť a testovateľnosť projektu.

2.4 Technológie používané na strane klientskej aplikácie

Klientská aplikácia je implementovaná pre mobilnú platformu iOS s využitím moderných technológií od Apple. Využíva sa programovací jazyk **Swift**, ktorý poskytuje bezpečnosť a pohodlie pri vývoji vďaka silnej kontrole a modernému syntaxu. Pre tvorbu používateľského rozhrania je použitý deklaratívny framework **SwiftUI**. Ako perzistentné úložisko aplikácia využíva framework **CoreData**, ktorý zabezpečuje ukladanie dát v zariadení používateľa. Celková architektúra aplikácie je postavená na princípe **MVVM** (Model-View-ViewModel), ktorý zabezpečuje prehľadnú komunikáciu medzi dátovou vrstvou a prezentačnou vrstvou aplikácie.

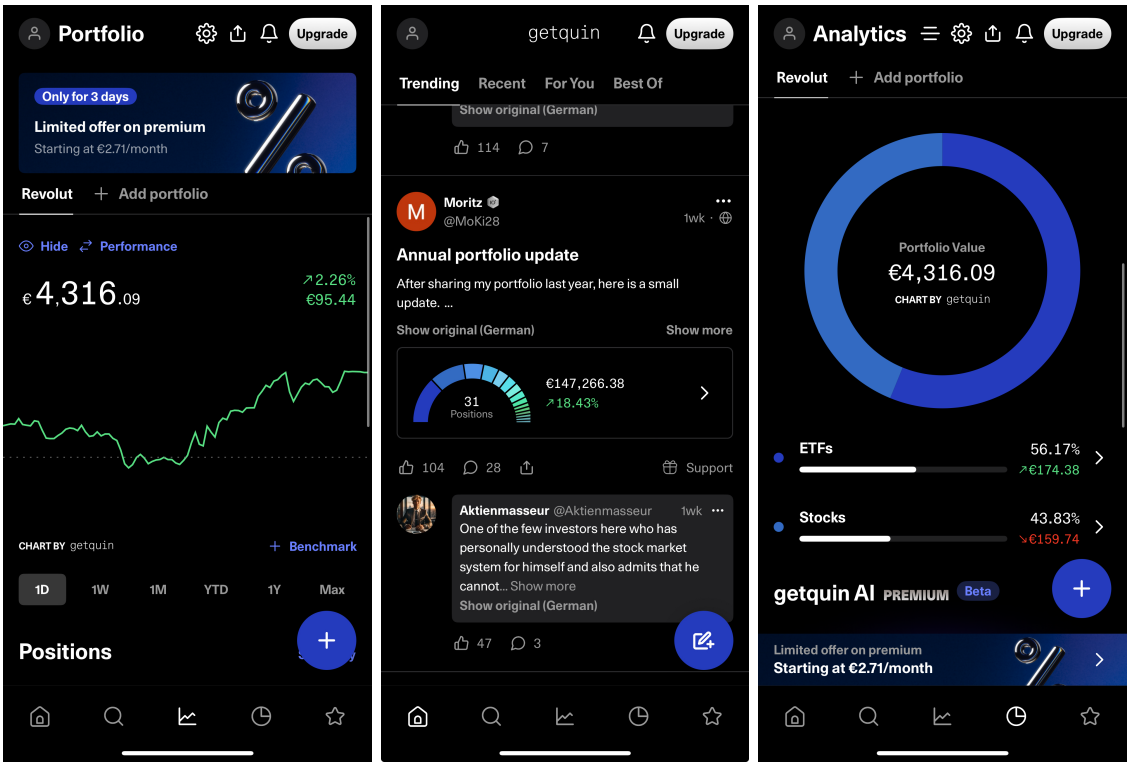
2.5 Analýza existujúcich riešení

Pred vytvorením návrhu aplikácie bola vykonaná analýza existujúcich riešení. Cieľom bolo preskúmať aktuálne dostupné možnosti pre sledovanie investičného portfólia a identifikovať ich výhody a nedostatky z pohľadu začiatočného investora. Analýza sa predovšetkým zameriavala na celkovú používateľskú skúsenosť a použiteľnosť aplikácie pre začiatočníkov. Výsledky tejto analýzy slúžili ako základ pri definovaní požiadaviek pre návrh vlastného riešenia prispôsobeného potrebám cieľovej skupiny.

getquin

Getquin je multiplatformná aplikácia určená na sledovanie investičného portfólia v reálnom čase, ktorá je okrem mobilnej aplikácie dostupná aj prostredníctvom webového rozhrania. Aplikácia umožňuje pridávať vlastné transakcie, sledovať výkonnosť portfólia v rôznych časových obdobiach a zisky z dividend. Súčasťou sú aj nápovedy k investičným termínom priamo v používateľskom rozhraní, čo môže byť veľkým prínosom pre začiatočníkov. Výraznou nevýhodou však je, že aplikácia kladie veľký dôraz na svoju komunitnú časť pripomínajúcu sociálnu sieť, ktorá veľmi znižuje prehľadnosť a narúša sústredenie na hlavný účel aplikácie. Ďalším nedostatkom je nedodržiavanie zvyklostí vývoja aplikácií pre iOS platformu, čo je pravdepodobne dôsledkom multiplatformného základu. Kvôli tomu aplikácia pôsobí v porovnaní s natívnymi aplikáciami neprirodzene. Aplikácia taktiež funguje na báze

predplatného, pričom vo verzii zdarma sú dostupné len veľmi obmedzené funkcie a používateľské rozhranie obsahuje veľmi rušivé reklamy na platené plány, čo výrazne zhoršuje použiteľnosť aplikácie. Navyše, prístup do aplikácie vyžaduje registráciu, čo môže pre mnohých začiatočníkov predstavovať ďalšiu bariéru. Hlavné obrazovky aplikácie sú zobrazené na obrázku 2.1.

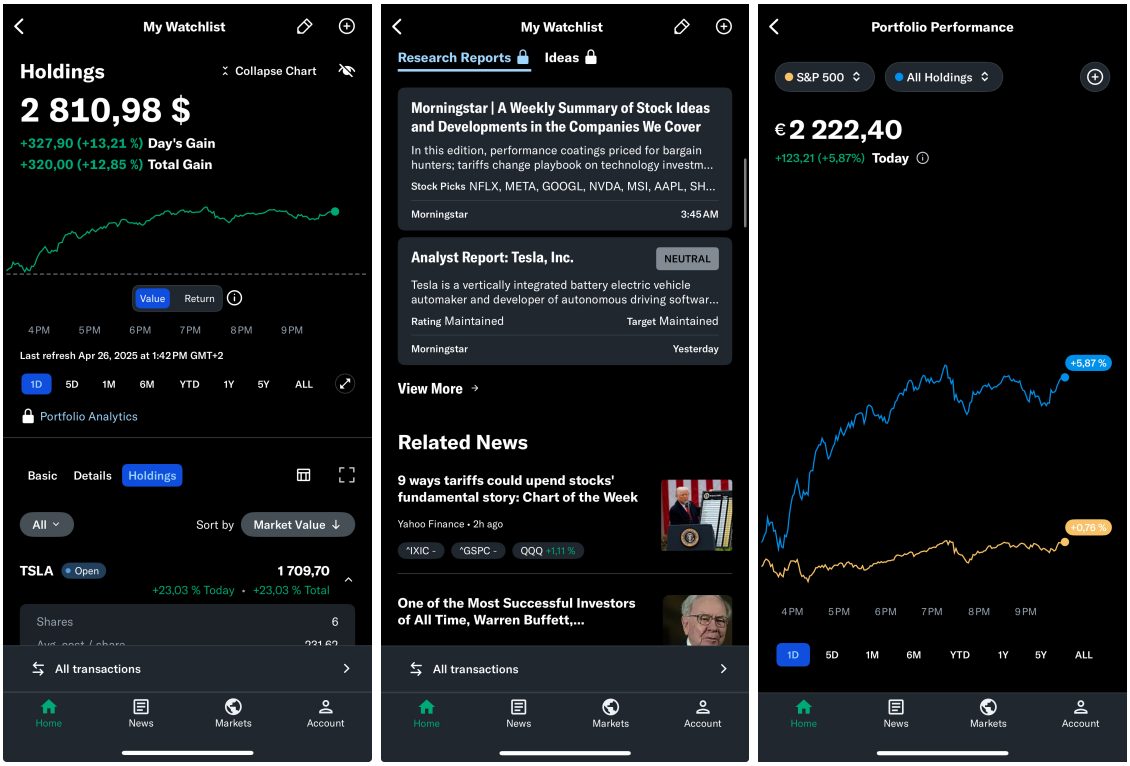


(a) Prehľad portfólia (b) Komunitná časť (c) Analýzy portfólia

Obr. 2.1: Ukážka aplikácie getquin.

Yahoo Finance

Služba Yahoo Finance patrí medzi najznámejšie platformy na sledovanie finančných trhov a ponúka aj mobilnú aplikáciu pre sledovanie vlastných portfólií. Výhodou je možnosť prepojenia s vybranými brokerskými účtami pre automatický import obchodov. Aplikácia zároveň poskytuje veľmi široké spektrum investičných metrík. Pre plný prístup k analytickým nástrojom je však potrebné predplatné. Z pohľadu začiatočníka môže byť zložité a preplnené používateľské rozhranie (zobrazené na obrázku 2.2) vnímané ako negatívum, ktoré okrem portfólia obsahuje aj množstvo správ z finančných trhov, čo je vhodnejšie skôr pre pokročilých používateľov. Zadávanie nových obchodov je neintuitívne a vyžaduje viacero krokov. Aplikácia navyše zobrazuje rozsiahle tabuľky, ktoré si vyžadujú horizontálne posúvanie, čo nie je optimálne pre zobrazenie na mobilných zariadeniach. Funkcia porovnávania aktív je obmedzená iba na fixné porovnanie so S&P 500 indexom, čím sa znižuje flexibilita pri analýze portfólia.

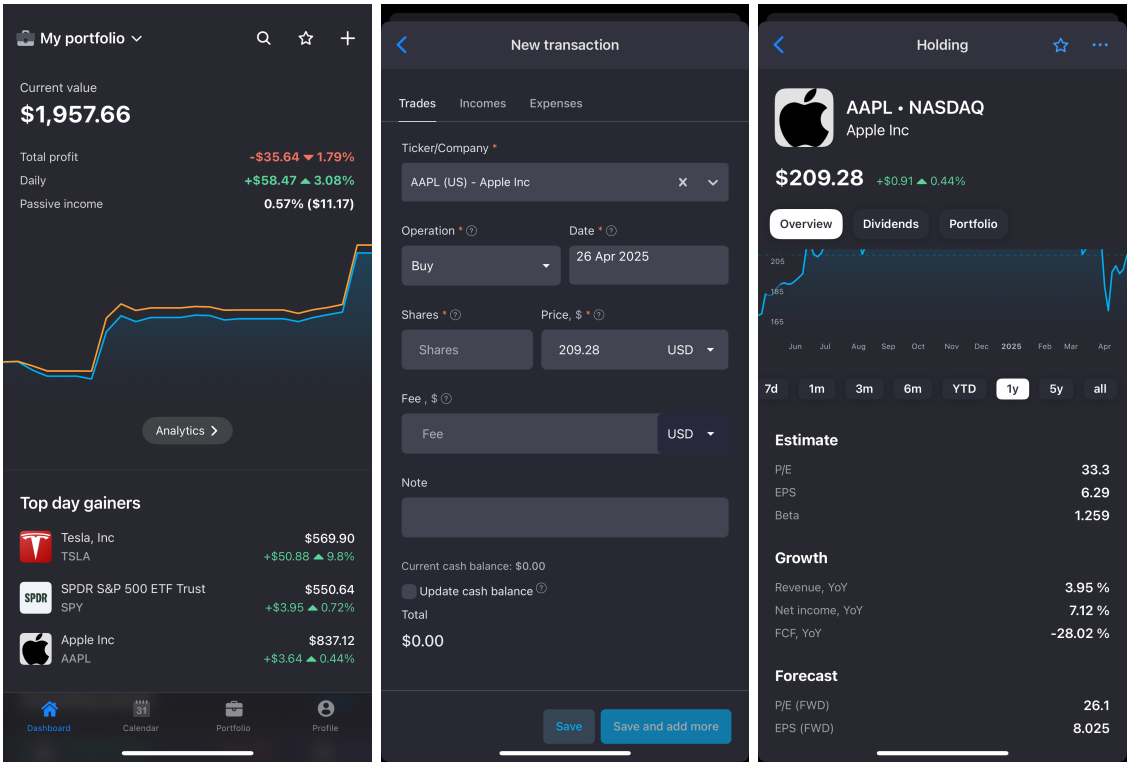


(a) Prehľad portfólia (b) Spravodajské informácie (c) Porovnanie aktív

Obr. 2.2: Ukážka aplikácie Yahoo Finance.

Snowball

Snowball poskytuje multiplatformné riešenie zamerané primárne na jednoduché sledovanie investičného portfólia a jeho analýzu bez pridávania rušivých rozšírení. Tento produkt je najbližší tomu, čo sa snaží dosiahnuť vlastná aplikácia popísaná v tejto práci — vytvoriť jednoduchý, prehľadný a účelový nástroj pre začiatočníkov. Používateľské rozhranie je vo všeobecnosti prehľadné a poskytuje široký rozsah informácií o aktívach, vrátane podrobného zobrazenia informácií o dividendoch. Aplikácia má konzistentné dizajnové spracovanie, no miestami pôsobí skôr ako webová aplikácia než natívne iOS riešenie (napríklad zobrazenie validácie vo formulároch, nevhodné typy klávesnice pre vstupné polia či občasná potreba manuálneho obnovenia používateľského rozhrania). Z pohľadu použiteľnosti je tiež problematické, že niektoré bežne využívané funkcie vyžadujú viacero krokov. Asi najväčšou nevýhodou je však veľmi limitovaná bezplatná verzia, ktorá umožňuje sledovať len maximálne desať aktív vo svojom portfóliu a poskytuje len obmedzené možnosti porovnávania či zobrazovania štatistík. Na rozdiel od vyššie spomenutých riešení táto aplikácia neposkytuje žiadne edukačné prvky, ktoré by mohli pomôcť začiatočníkom lepšie sa orientovať v oblasti investovania. Ukážka aplikácie je zobrazená na obrázku 2.3.



(a) Prehľad portfólia (b) Pridávanie nového obchodu (c) Detail aktíva

Obr. 2.3: Ukážka aplikácie Snowball.

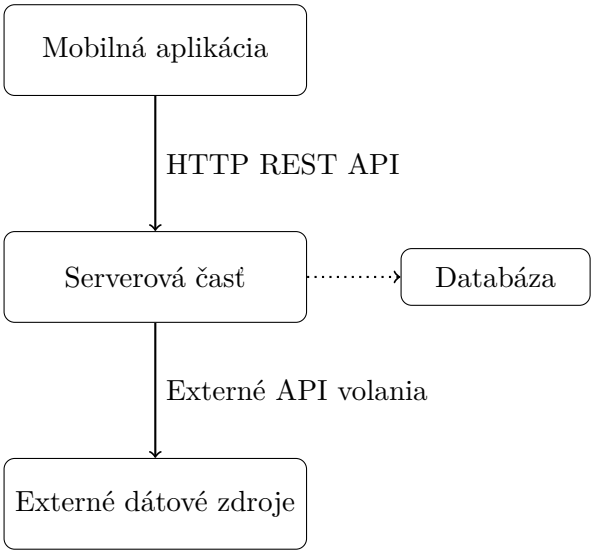
Zhrnutie analýzy a motivácia pre vlastné riešenie

Analýza existujúcich riešení ukázala, že väčšina dostupných aplikácií kladie dôraz na multiplatformnosť na úkor optimalizácie a dodržiavania zvyklostí natívnych iOS aplikácií. Výsledkom sú používateľské rozhrania, ktoré pôsobia pre dlhodobých používateľov platformy iOS neintuitívne a neprakticky, keďže nezohľadňujú ich očakávania. Navyše, mnohé z týchto riešení nemusia byť vhodné pre začiatočníkov alebo im chýbajú edukačné prvky, ktoré by používateľom pomáhali priebežne získavať nové investičné znalosti pri práci s aplikáciou. Tieto zistenia slúžia ako základ návrhu vlastného riešenia, ktorý je inšpirovaný silnými stránkami existujúcich riešení, ponúka intuitívne používateľské rozhranie, podporuje vzdelávanie začiatočníkov a drží sa štandardov natívneho spracovania pre platformu iOS.

Kapitola 3

Návrh riešenia

Táto kapitola sa zameriava na technologický návrh riešenia, ktoré pozostáva z dvoch hlavných komponentov – serverovej časti a klientskej aplikácie. Neoddeliteľnou súčasťou architektúry projektu sú aj externí poskytovatelia dát, vďaka ktorým dokáže server poskytovať klientskej aplikácii finančné dáta. Cieľom kapitoly je detailne popísať architektúru jednotlivých častí (stručne zobrazenú na obrázku 3.1), ich vzájomnú integráciu, spôsob komunikácie a rozhodnutia, ktoré ovplyvnili návrh z pohľadu použiteľnosti, výkonnosti a rozšíriteľnosti.



Obr. 3.1: Vysokúrovňová architektúra riešenia.

3.1 Architektúra serverovej časti

Dôležitým faktorom pri návrhu riešenia bolo navrhnúť vlastnú backendovú vrstvu, ktorá má slúžiť ako určitá forma abstrakcie nad rôznymi poskytovateľmi finančných dát. Tento prístup zabezpečí vyššiu flexibilitu pri výbere, rozširovaní alebo nahrádzaní externých služieb v budúcnosti. Ďalšou výhodou je aj možnosť začať vývoj s cenovo dostupnými prístupovými plánmi poskytovateľov dát a v prípade potreby prechodu do produkčného prostredia jednoducho prejsť na plány s komerčnými licenciami.

Vlastný backend tiež zabezpečuje konzistentnosť a validitu dát poskytovaných pre klientskú aplikáciu. Klient sa tak nemusí starať o validáciu, transformáciu či mapovanie dát a ani o spracovanie rôznych typov odpovedí z aplikačných rozhraní tretích strán. Ide o aplikáciu princípu oddelenia zodpovedností (*separation of concerns*), ktorý je základom udržateľného návrhu moderných softvérových riešení.

Ďalšou výhodou tohto riešenia je aj možnosť pracovať so službami, ktoré umožňujú len obmedzený počet API požiadaviek za určitú dobu, alebo nedisponujú dostatočnou výkonnosťou. Backend môže obsahovať perzistentné úložisko za účelom cachovania dát tretích strán. Postupným ukladaním dát do tejto vlastnej databázy sa výrazne znižuje záťaž na externé API a zároveň je tak možné zabezpečiť rýchlejšiu odozvu pre klienta. Tento prístup umožňuje optimalizovať výkonnosť aplikácie nezávisle od výkonnosti služieb tretích strán, ktorú nemožno ovplyvniť.

Bezpečnosť predstavuje ďalší kľúčový dôvod, prečo je implementácia samostatného backendu nevyhnutná. Mnohé služby, ako napríklad Polygon.io, využívajú na autentifikáciu súkromné prístupové kľúče, ktoré nie sú určené na distribúciu do klientských zariadení. Prístup k týmto dátam preto musí byť sprostredkovaný cez server, kde je možné zabezpečiť ochranu prístupových údajov a kontrolovanú komunikáciu s týmito externými službami.

Hlavnou úlohou servera v tomto projekte je teda sprostredkovanie dynamického prístupu k externým dátam pre mobilnú aplikáciu. Pri požiadavke na dáta server najskôr vždy overí ich dostupnosť v internej databáze. V prípade, že požadované dáta sa v nej nenachádzajú, získa ich z externého API, spracuje a uloží ich pre budúce použitie.

Dáta o aktívach

Pri požiadavkách na detaily o konkrétnom aktíve môžu nastať 2 situácie:

- **Dáta sú už dostupné v databáze**

  V tomto prípade sú dáta získané z databázy ihneď vrátené klientovi

- **Dáta nie sú v databáze**

  Ak ide o dopyt na nové aktívum, ktoré sa ešte v databáze nenachádza, server automaticky získa požadované údaje z rôznych služieb podľa typu aktíva (napríklad akcia alebo ETF). Po úspešnom získaní vráti tieto informácie klientovi a zároveň ich uloží do databázy. Takýmto spôsobom dochádza k dynamickému rozširovaniu databázy finančných dát na základe reálnych požiadaviek používateľov s výhodou ukladania len skutočne žiadaných dát.

Od servera sa v tomto prípade očakáva poskytnutie požadovaných dát pre klienta bez potreby poznať spôsob fungovania dátových zdrojov na pozadí a kombinovania rôznych služieb. Údaje o existujúcich tickeroch sú priebežne aktualizované prostredníctvom periodických plánovaných cron úloh. Pri týchto aktualizáciách, ako aj pri prvotnom vytváraní záznamu o aktíve v internej databáze, sa získavajú detaily ako názov, symbol, primárna burza a podobne, ako aj často meniace sa údaje, napríklad objem obchodovania za posledný týždeň alebo uzatváracia cena predchádzajúceho obchodného dňa. Výhodou tohto dynamického prístupu pri plnení databázy je znížená náročnosť na úložisko v porovnaní s predbežným naplnením databázy veľkým objemom dát. Zároveň sa takto neobmedzuje rozsah dostupných aktív len na konkrétnu podmnožinu, ale je umožnený prístup ku všetkým aktívam, ktoré daný dátový poskytovateľ podporuje. Aktualizácia údajov je taktiež efektívnejšia, pretože sa priebežne obnovujú iba tie dáta, o ktoré bol záujem zo strany klienta.

Agregácie obchodov

Z hľadiska požadovaného časového rozmedzia záznamov o obchodných agregáciách sú v tomto projekte využívané dve rozlíšenia. Toto riešenie bolo zvolené kvôli optimalizácii nárokov na úložný priestor, pričom sa zároveň prihliadalo na potrebu zabezpečiť dostatočný počet dátových bodov pre grafické zobrazenie v aplikácii s prihliadaním na konkrétne prípady použitia.

- **Časové rozmedzie do 7 dní**

  Pre časové obdobie kratšie alebo rovné siedmim dňom sa pracuje s agregáciami po štyroch hodinách. Táto veľkosť agregačného okna bola zvolená ako optimálna po zvažovaní jemnejšej granularity (napr. jedna hodina), ktorá sa ukázala ako príliš náročná na objem dát a zbytočná s ohľadom na požiadavky projektu. Štvorhodinové agregácie predstavujú vyvážený kompromis medzi množstvom dátových bodov pre vizualizáciu a nárokmi na úložisko.

- **Časové rozmedzie viac ako 7 dní**

  Pri vyžiadaní väčšieho časového rozmedzia sa automaticky používa veľkosť agregačného okna jeden deň. Tento prístup výrazne znižuje objem dát pri dlhodobých prehľadoch a poskytuje dostatočnú informačnú hodnotu o vývoji ceny.

Dáta o obchodných agregáciách sú tiež ukladané na strane servera s cieľom umožniť rýchlejší prístup pri budúcich požiadavkách — podobne ako v prípade detailov o aktívach. Rozdiel však spočíva v tom, že tieto dáta majú aj časovú dimenziu, čo vyžaduje pokročilejšie spracovanie a dodatočnú réžiu.

Pre efektívne cachovanie a minimalizáciu počtu požiadaviek na externé služby sa popri dátach agregácií uchovávajú aj dodatočné záznamy o časových rozmedziach, ktoré uložené agregácie pokrývajú. Na základe týchto záznamov systém dokáže identifikovať, pre ktoré časové úseky je možné získať agregované dáta priamo z databázy a kedy je potrebné doplniť chýbajúce dáta z externého zdroja.

Komunikácia so službami tretích strán

Pre komunikáciu so službami poskytujúcimi dáta sa využíva protokol HTTP a rozhranie typu REST API s odpoveďami vo formáte JSON. Server používa oficiálny NPM balíček `@polygon.io/client-js`, ktorý poskytuje JavaScript API klienta s definovaným rozhraním zodpovedajúcim štruktúre REST API služby Polygon.io. Využitie tohto klienta zjednodušuje a sprehľadňuje tvorbu požiadaviek, pričom zároveň zvyšuje bezpečnosť a udržateľnosť kódu — napríklad vďaka vyššej odolnosti voči prípadným budúcim zmenám v štruktúre HTTP endpointov zo strany poskytovateľa, keďže abstrakcia klienta môže zabezpečiť spätnú kompatibilitu.

V prípade služby Yahoo Finance, ktorá je bezplatná a verejne dostupná — no neposkytuje oficiálnu dokumentáciu k svojmu API — sa využíva komunitný open–source klient `yahoo-finance2`. Tento nástroj prináša rovnaké výhody ako klient od služby Polygon.io — ako sú typová bezpečnosť, jednoduchšia práca s požiadavkami, či abstrahovanie HTTP komunikácie — no zároveň prináša aj benefit komunitnej dokumentácie dostupných endpointov a ich komunikačného rozhrania.

Spôsob nasadzovania a prevádzka serveru

Backendová časť projektu je nasadená na serverless platforme Vercel s využitím bezplatného plánu, ktorý je vzhľadom na požiadavky tohto projektu postačujúci. Súčasťou tejto služby je aj vlastná inštancia kompatibilnej PostgreSQL databázy, ktorá slúži ako perzistentné úložisko dát. Napojenie na databázu je vyriešené v prostredí Vercelu, čo zabezpečuje spoľahlivú prevádzku aj v serverless prostredí bez potreby využívania externých databázových riešení.

Platforma Vercel zároveň poskytuje automatickú konfiguráciu procesu kontinuálnej integrácie (*CI pipeline*) pre projekty využívajúce framework Nuxt, čo umožňuje automatické nasadzovanie aplikácie po každom commite v prepojenom GitHub repozitári.

3.2 Dátové modely

Serverová aplikácia pracuje s dvoma typmi dátových modelov — internými modelmi používanými v rámci projektu a modelmi, ktoré reprezentujú dáta získané z externých služieb. Keďže formát odpovedí z týchto služieb nemusí byť vhodný na priame použitie v rámci aplikácie, boli pre ne vytvorené samostatné dátové modely, ktoré zároveň zabezpečujú validáciu a poskytujú zjednodušený prístup k dátam prostredníctvom vlastných getter metód.

Dátové modely pre externé dáta

Nižšie sú popísané jednotlivé modely vytvorené pre spracovanie externých dát. Vo väčšine prípadov modely využívajú len podmnožinu atribútov poskytovaných externými službami, ktoré sú relevantné pre potreby aplikácie.

AggregationPolygonModel Tento model reprezentuje jednotlivý agregačný záznam obchodov konkrétneho aktíva získaný zo služby Polygon.io. Obsahuje nasledovné atribúty [4]:

- `o` – otváracia cena v agregačnom okne,
- `c` – zatváracia cena v agregačnom okne,
- `h` – najvyššia cena v agregačnom okne,
- `l` – najnižšia cena v agregačnom okne,
- `n` – počet transakcií v agregačnom okne,
- `t` – časová značka začiatku agregačného intervalu (v milisekundách),
- `v` – počet zobchodovaných akcií,
- `vw` – vážený priemer ceny na základe objemu.

TickerSummaryPolygonModel Tento model reprezentuje výkon aktíva za posledný obchodný deň. Obsahuje nasledovné atribúty [10]:

- `o` – otváracia cena v daný deň,
- `c` – zatváracia cena v daný deň,

- `l` – najnižšia cena dňa,
- `h` – najvyššia cena dňa.

TickerPolygonModel Tento model poskytuje detailné informácie o konkrétnych aktívach a predstavuje kľúčový prvok pri vytváraní alebo aktualizácií záznamu o aktíve v internej databáze aplikácie. Obsahuje informácie ako [12]:

- `ticker` – symbol aktíva,
- `name` – názov aktíva,
- `description` – stručný popis,
- `market` – typ trhu (v projekte sa uvažuje len `stocks`),
- `type` – typ aktíva (napr. ETF alebo kmeňová akcia),
- `currency_name` – mena, v ktorej sa aktívum obchoduje,
- `primary_exchange` – primárna burza,
- `list_date` – dátum zalistovania na burze.

SearchTickerModel Tento model reprezentuje výsledok vyhľadávania v endpointe kolekcie tickerov služby Polygon.io. Na rozdiel od ostatných modelov pre externé dáta jeho názov nezahŕňa označenie zdrojovej služby. Dôvodom je, že tieto dáta sa neukladajú do databázy a ani nevyužívajú v iných častiach serverovej aplikácie. Model preto slúži ako externý aj interný dátový model zároveň, a tak je pomenovaný v štýle interných modelov — aby sa predišlo zbytočnej duplikácii. Model obsahuje nasledovné atribúty, pričom niektoré z nich sú pri serializácii mapované na názvy atribútov používané v klientskej aplikácii [2]:

- `ticker` (mapuje sa na `symbol`) – symbol aktíva,
- `name` – názov aktíva,
- `market` – typ trhu,
- `type` – typ aktíva,
- `currency_name` (mapuje sa na `currency`) – mena, v ktorej sa aktívum obchoduje,
- `primary_exchange` (mapuje sa na `exchange`) – primárna burza.

TickerYahooModel Tento model predstavuje doplňujúce údaje o aktíve zo služby Yahoo Finance, ktoré nie sú dostupné cez službu Polygon.io:

- `name` – formátovaný plný názov aktíva,
- `fifty_two_week_low` – najnižšia cena za posledný rok,
- `fifty_two_week_high` – najvyššia cena za posledný rok,
- `price_to_earnings` – pomer ceny aktíva k zisku na jednu akciu (P/E),

- `market_cap` – trhová kapitalizácia spoločnosti,
- `beta` – koeficient beta (volatilita aktíva v porovnaní s trhom),
- `earnings_per_share` – zisk na jednu akciu (EPS).

Interné dátové modely

Dátové modely popísané nižšie sú využívané v aplikačnej logike a majú rovnakú štruktúru na strane servera aj v klientskej aplikácii.

AggregationModel Tento model reprezentuje agregačný záznam obchodov pre konkrétne aktívum v definovanom časovom okne. Na strane servera sa rozlišuje medzi čiastočnou a plnou verziou tohto modelu, pričom v klientskej aplikácii sú atribúty plného modelu, ktoré nie sú súčasťou čiastočnej verzie, definované ako voliteľné.

Tento model je zároveň optimalizovaný pre prenos po sieti tak, že pri serializácii sú kľúče jeho atribútov skrátené s cieľom minimalizovať objem prenášaných dát. Tento prístup priniesol zníženie veľkosti odpovedi od servera približne o 21%. Napríklad veľkosť odpovedi pre kolekciu agregácií ceny akcií spoločnosti Apple v časovom rozmedzí päť rokov sa zmenšila z `225,38 KB` na `176,32 KB`.

- `id` (serializované na `i`) – interný identifikátor agregačného záznamu v databáze,
- `tickerId` (serializované na `ti`) – identifikátor aktíva, ku ktorému agregácia patrí,
- `weightedAvgPrice` (serializované na `w`) – vážený priemer ceny na základe obchodovaného objemu v rámci daného agregačného intervalu,
- `timestamp` (serializované na `t`) – časová značka začiatku agregačného intervalu,
- `open` (serializované na `o`) – otváracia cena agregovaného intervalu (na serveri súčasťou len `AggregationFullModel`, na klientovi voliteľný atribút),
- `close` (serializované na `c`) – zatváracia cena agregovaného intervalu (na serveri súčasťou len `AggregationFullModel`, na klientovi voliteľný atribút).

AggregationRangeModel Tento model sa používa na reprezentáciu časových rozsahov, pre ktoré sú agregačné záznamy v databáze servera dostupné. Keďže slúži výlučne na optimalizáciu požiadaviek na externé API, tento model sa v klientskej aplikácií nevyužíva.

- `id` – interný identifikátor záznamu o rozsahu v databáze,
- `tickerId` – identifikátor aktíva, ku ktorému záznam o rozsahu pokrytia patrí,
- `from` – časová značka začiatku pokrytého obdobia,
- `to` – časová značka konca pokrytého obdobia.

TickerModel Tento dátový model reprezentuje aktívum uložené v databáze. Je hlavným modelom, s ktorým aplikácia pracuje.

- `id` – interný identifikátor záznamu v databáze,
- `symbol` – symbol tickera (napr. `AAPL`, `SPY`),
- `name` – názov aktíva,
- `description` – stručný popis aktíva,
- `market` – typ trhu, v ktorom sa aktívum obchoduje (v projekte sa uvažuje len `stocks`),
- `type` – typ aktíva (napr. ETF alebo kmeňová akcia),
- `currency` – mena, v ktorej sa aktívum obchoduje,
- `exchange` – burza, na ktorej je aktívum primárne zalistované,
- `listDate` – dátum prvého zalistovania na burze,
- `price` – uzatváracia cena z posledného obchodného dňa,
- `volume` – objem obchodovania za posledný obchodný deň,
- `volumeWeek` – objem obchodovania za posledných sedem dní,
- `volume3MonthAvg` – priemerný denný objem obchodovania za posledné tri mesiace,
- `lowFiftyTwoWeek` – najnižšia cena za posledný rok,
- `highFiftyTwoWeek` – najvyššia cena za posledný rok,
- `priceToEarnings` – pomer ceny aktíva k zisku na jednu akciu (P/E),
- `marketCap` – trhová kapitalizácia,
- `beta` – koeficient beta (volatilita aktíva v porovnaní s trhom),
- `earningsPerShare` – zisk na jednu akciu (EPS),
- `createdAt` – čas vytvorenia záznamu v databáze,
- `updatedAt` – čas poslednej aktualizácie záznamu.

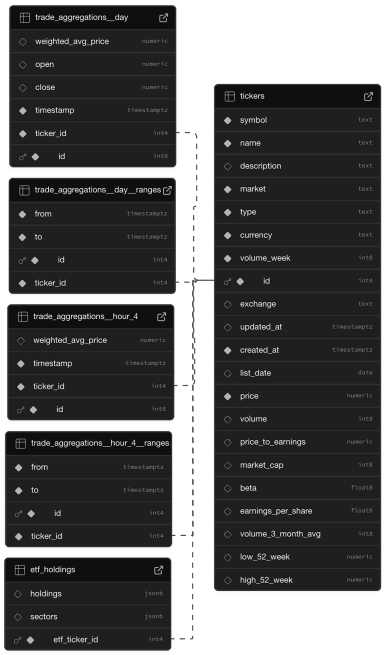
EtfHoldingsModel Tento model obsahuje informácie o zložení konkrétneho ETF fondu. Zloženie je vyjadrené v jednotlivých aktívach alebo v sektoroch. Nie všetky ETF fondy v aplikácii majú tieto informácie dostupné.

- `holdings` – pole objektov vyjadrujúcich percentuálne zastúpenie daných aktív,
- `sectors` – pole objektov vyjadrujúcich percentuálne pokrytie daných sektorov aktívami konkrétneho ETF fondu.

3.3 Databáza serveru

Databáza serveru je postavená na open–source relačnej databáze PostgreSQL. Pre potreby ukladania finančných dát a optimalizáciu prístupu k nim bola navrhnutá databázová schéma, ktorá je zobrazená na obrázku 3.2 a pozostáva z nasledovných tabuliek:

- `tickers` – hlavná tabuľka obsahujúca dáta o jednotlivých aktívach, ako je symbol, názov, typ aktíva, mena, primárna burza, dátum zalistovania, obchodovaný objem za posledný týždeň a uzatváracia cena z predchádzajúceho dňa. Tabuľka obsahuje aj dátumy vytvorenia a poslednej aktualizácie záznamu. Kombinácia `symbol` a `exchange` je definovaná ako unikátna v rámci všetkých záznamov v tejto tabuľke.
- `trade_aggregations__hour_4` – táto tabuľka uchováva agregované obchodné dáta po štyroch hodinách — konkrétne ide o vážený priemer ceny (`weighted_avg_price`) vzhľadom na obchodovaný objem v agregačnom okne, ktorého začiatok je indikovaný časovou značkou `timestamp`. Každý záznam je naviazaný na konkrétne aktívum prostredníctvom cudzieho kľúča.
- `trade_aggregations__day` – táto tabuľka uchováva denné agregované obchodné dáta. Oproti tabuľke pre štvorhodinové agregácie uchováva táto tabuľka aj informácie o otváracej (`open`) a zatváracej (`close`) cene aktíva v danom dni.
- `trade_aggregations__hour_4__ranges` a `trade_aggregations__day__ranges` – pomocné tabuľky uchovávajúce rozsahy časových intervalov, ktoré sú pokryté agregáciami obchodov pre konkrétne aktíva v danom rozlíšení.
- `etf_holdings` – táto tabuľka obsahuje informácie o zložení konkrétnych ETF fondov

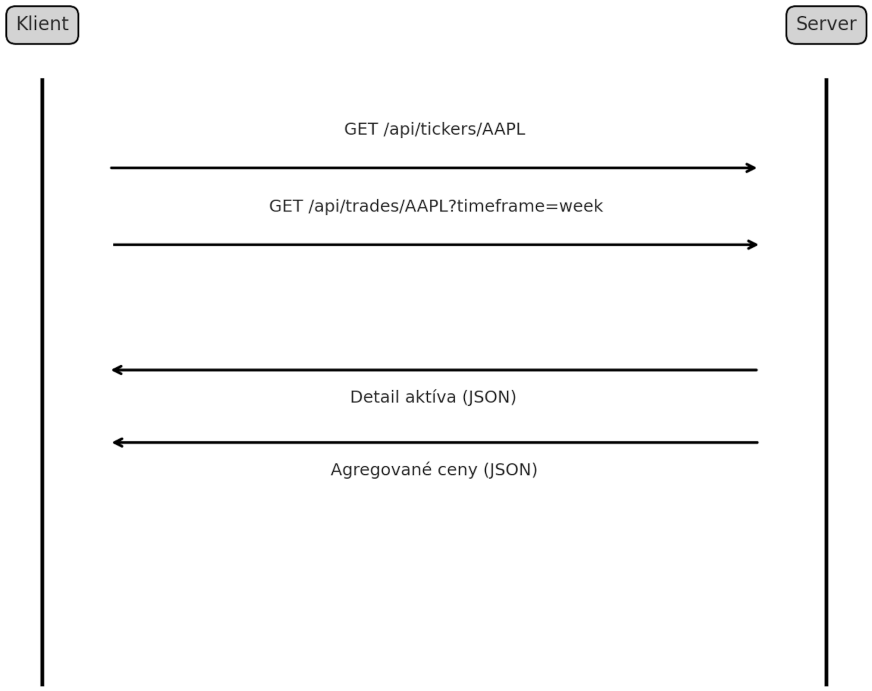


Obr. 3.2: Schéma serverovej databázy projektu

3.4 Komunikačné rozhranie medzi serverom a aplikáciou

Klientská aplikácia pristupuje k backendu cez protokol HTTP s využívaním REST API rozhrania poskytovaného serverom a komunikuje vo formáte JSON. Ten umožňuje klientovi zobraziť detaily o jednotlivých aktívach, získať výpis kolekcie všetkých aktuálne evidovaných aktív, vyhľadať nové aktíva, ktoré ešte v databáze nie sú, získať obchodné agregácie pre určité časové obdobie a zobraziť obsah zloženia ETF fondov.

Aplikácia vykonáva HTTP požiadavky podľa potreby a vždy ide o jednostrannú komunikáciu, kde server len odpovedá na dopyty zo strany klienta bez udržiavania aktívneho spojenia. Aplikácia zároveň komunikuje výhradne s týmto backendom a nevykonáva žiadne priame požiadavky na externé služby. Príklad komunikácie so serverom je znázornený na obrázku 3.3.



Obr. 3.3: Príklad komunikácie klienta so serverom pri načítaní obrazovky detailu aktíva.

Poskytované API endpointy

V rámci REST API server poskytuje nasledovné endpointy, prostredníctvom ktorých klient získava potrebné dáta:

- `GET /api/tickers/:symbol` Poskytuje detailné informácie o konkrétnom aktíve. V prípade, že sa aktívum ešte nenachádza v databáze, server ho dynamicky vytvorí načítaním z externých zdrojov pred odoslaním odpovede.
- `GET /api/tickers` Poskytuje kolekciu aktuálne dostupných aktív v databáze servera, zoradenú podľa objemu obchodov za posledných sedem dní. Endpoint podporuje nasledujúce query parametre:
  - `type` – nepovinný filter podľa typu aktíva (možné hodnoty: `cs`, `etf`),

  - `ids` – zoznam ID aktív (podľa identifikácie v databáze) oddelených čiarkou pre výber konkrétnych záznamov.

- `GET /api/tickers/search` Umožňuje vyhľadávanie aktív v rámci celej množiny aktív dostupnej u externého poskytovateľa dát — teda aj tie, ktoré ešte nie sú uložené v databáze servera. Akceptované sú nasledujúce query parametre:

  - `q` – vyhľadávací reťazec (povinný),
  - `type` – nepovinný filter podľa typu aktíva (možné hodnoty: `cs`, `etf`).

- `GET /api/trades/:symbol` Vracia agregované obchodné dáta pre konkrétne aktívum v špecifikovanom časovom rozmedzí. V prípade, že vo vyžiadanom časovom období na serveri chýba pokrytie dátami z databázy, server ich automaticky získa z externých zdrojov a doplní pred odoslaním odpovede klientovi. Tento endpoint podporuje nasledujúce query parametre:

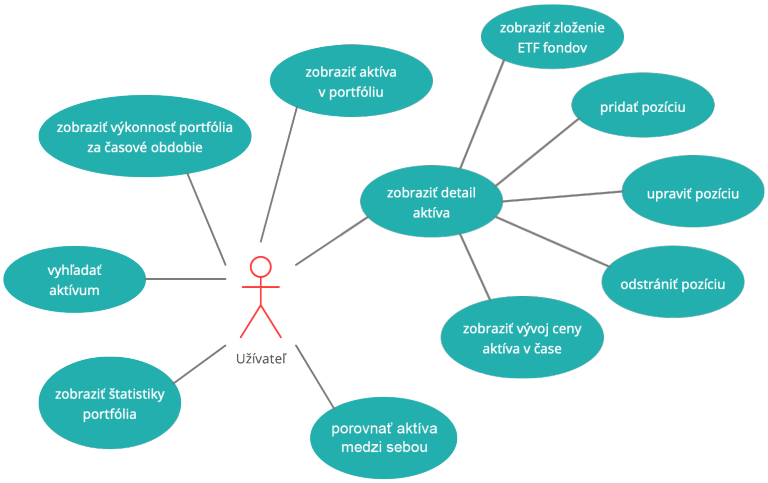
  - `from` – počiatočný časový bod,
  - `to` – koncový časový bod,
  - `timeframe` – preddefinované časové rozmedzie (nemožno kombinovať s `from`/`to`; povolené hodnoty: `week`, `month`, `year`, `all`).

  Pokiaľ nie je špecifikované časové rozmedzie, predpokladá sa najväčší možný interval.

- `GET /api/etf/:symbol/composition` Poskytuje zoznam hlavných prvkov v zložení daného ETF fondu, pokiaľ sú tieto informácie dostupné.

3.5 Návrh aplikácie

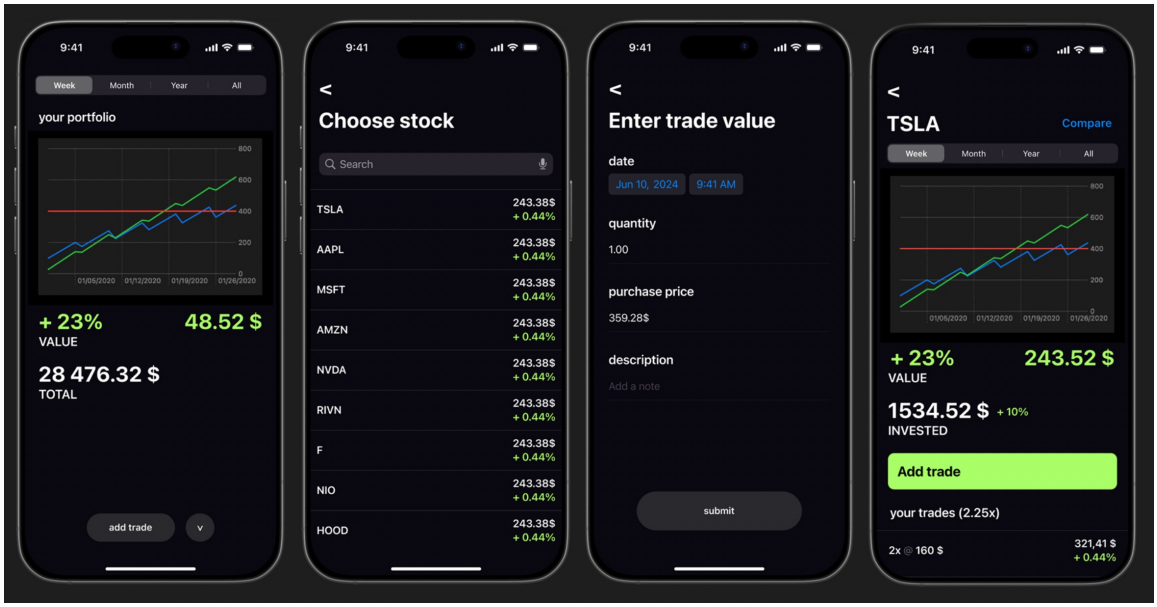
Návrh mobilnej aplikácie vychádzal z požiadaviek na jednoduchosť, intuitívnosť a dostupnosť informácií pre používateľa bez veľkých skúseností v oblasti investícií. Hlavné akcie, ktoré môže používateľ v aplikácii vykonávať, sú zobrazené v diagrame prípadov použitia na obrázku 3.4 nižšie.



Obr. 3.4: Diagram prípadov použitia aplikácie.

Používateľské rozhranie aplikácie bolo pred implementáciou navrhnuté v online nástroji **Figma**, pričom jeho časť je zobrazená na obrázku 3.5. Pri návrhu bol použitý oficiálny balíček komponentov *iOS 18 & iPadOS 18 UI Kit*, ktorý je súčasťou oficiálnych dizajnových usmernení *Apple Design Resources* pre vývoj aplikácií určených pre platformy spoločnosti Apple. Tento balíček obsahuje množstvo špecializovaných komponentov, ktoré zjednodušujú tvorbu návrhov používateľského rozhrania a zabezpečujú vizuálnu konzistenciu. V prípadoch, kde neboli dostupné vhodné komponenty, boli použité zástupné obrázky alebo jednoduché textové symboly.

Pre ikonografiu bola použitá knižnica *SF Symbols*, čo je balíček viac než 6 000 symbolov navrhnutých a poskytovaných spoločnosťou Apple, ktoré sú vizuálne konzistentné so systémovým písmom *San Francisco* a dizajnovým jazykom operačných systémov Apple [3].



Obr. 3.5: Ukážka niektorých obrazoviek z Figma návrhu.

3.6 Lokálna databáza aplikácie

Používateľské dáta o portfóliu — záznamy o investíciách do jednotlivých aktív — sú v aplikácii uložené lokálne pomocou databázového systému **Core Data**, ktorý je súčasťou platformy iOS. V celej schéme sa využíva jedna entita s názvom `Trade`, ktorá obsahuje informácie o počte nakúpených jednotiek aktíva (`amount`), nákupnej cene (`price`), internom identifikátore aktíva (`tickerId`), čase vykonania transakcie (`timestamp`) a voliteľný popis záznamu (`descriptionText`).

Táto štruktúra poskytuje všetky potrebné údaje pre výpočet hodnoty portfólia a zobrazenie histórie obchodov priamo na zariadení používateľa. Zároveň vďaka tomuto prístupu nie je potrebná autentifikácia ani ukladanie dát používateľov na strane servera.

Kapitola 4

Implementácia serverovej časti

V tejto kapitole sú popísané kľúčové aspekty implementácie serverovej logiky, ktorá zabezpečuje získavanie, spracovanie a poskytovanie dát pre klientskú aplikáciu. Rozoberá sa tu spôsob validácie vstupných dát, štruktúra výstupných dát, prístup k dátam z databázy a z endpointov externých služieb, automatická aktualizácia a správa dát, organizácia kódu, ošetrenie chybových stavov a ďalšie implementačné detaily.

4.1 Validácia dát

Nevyhnutnou súčasťou akejkoľvek produkčnej aplikácie je dôkladná validácia všetkých dát, ktoré nie sú pod kontrolou systému. Jej cieľom je čo najskôr identifikovať a eliminovať potenciálne zdroje nekonzistencie či bezpečnostných rizík, ktoré by neskôr mohli viesť k chybám alebo narušeniu integrity aplikácie. Vzhľadom na to, že serverová aplikácia pracuje vo veľkej miere s dátami získanými z externých služieb, validácia zohráva kľúčovú úlohu a je priamo zakomponovaná do implementácie dátových modelov.

Na definovanie validačných pravidiel sa využíva knižnica `Zod`, ktorá umožňuje opísať očakávanú štruktúru a typ dát pomocou deklaratívnej definície dátovej schémy, a ktorá zároveň slúži na generovanie TypeScript typov dátových modelov. Knižnica tiež podporuje transformáciu vstupu už počas validácie, čo zjednodušuje ďalšie spracovanie dát v aplikácii. Príklad tejto schémy je uvedený nižšie vo výpise 4.1. Ide o schému validujúcu základné údaje o aktíve zo služby Polygon.io, kde sú atribúty s hodnotamy výčtových typov Polygonu transformované počas validácie na interné hodnoty používané aplikáciou.

```
export const partialTickerSchema = z.object({
    ticker: z.string(),
    name: z.string(),
    market: zEnum(PolygonMarket).transform(v => convertPolygonMarket(v)),
    type: zEnum(PolygonTickerType).transform(v => convertPolygonTickerType(v)),
    currency_name: zEnum(PolygonCurrency).transform(v => convertPolygonCurrency(v)),
    primary_exchange: zEnum(PolygonExchange).nullable().optional().transform(v =>
        convertPolygonExchange(v)),
})
```

Výpis 4.1: Validačná schéma s použitím knižnice Zod. Poznámka: `zEnum` je vlastná pomocná funkcia postavená na `z.enum`, ktorá zjednodušuje prevod JavaScript objektu na Zod enumeráciu.

4.2 Implementácia dátových modelov

Každý dátový model na serveri dedí z abstraktnej triedy `BaseModel`, ktorá vykonáva validáciu vstupných dát priamo pri vytváraní inštancie. V prípade zlyhania validácie sa vyvolá výnimka určená na spracovanie vo volajúcom kóde, čím sa predchádza ďalšiemu spracovaniu chybných údajov. Táto trieda je generická a využíva typ odvodený z poskytnutej Zod schémy pre dáta, čo umožňuje zaistiť typovú kontrolu bez duplicitnej definície typov a validačnej logiky.

Model taktiež obsahuje `private` metódu `toJSON()`, ktorá sa podľa ECMAScript špecifikácie volá automaticky pri serializácii objektu pomocou funkcie `JSON.stringify()` [7]. Pre možnosť úpravy výsledného serializovaného objektu bola kvôli typovej kompatibilite zavedená metóda `serialize()`, ktorá je určená na preťaženie v modeloch, ktoré vyžadujú úpravu výstupných dát v API odpovedi. Tento mechanizmus je využívaný napríklad pri serializácii agregačných záznamov, kde dochádza k nahradeniu dlhých názvov atribútov ich skrátenými verziami pre zníženie objemu dát prenášaných po sieti, ako bolo podrobnejšie popísané v kapitole 3.2. Vo výpise 4.2 nižšie je uvedený príklad implementácie základu všetkých dátových modelov serverovej aplikácie.

```
export abstract class BaseModel<TSchema extends ZodSchema> {
    private readonly data: output<TSchema>

    constructor(schema: TSchema, data: unknown) {
        const parsed = schema.safeParse(data)

        if (!parsed.success) {
            throw new ModelDataError(parsed.error)
        }

        this.data = parsed.data
    }

    protected _getAttribute<T extends keyof output<TSchema>>(name: T):
        output<TSchema>[T] {
        return this.data[name]
    }

    protected _setAttribute<T extends keyof output<TSchema>>(name: T, value:
        output<TSchema>[T]): void {
        this.data[name] = value
    }

    protected serialize(): Record<string, any> {
        return this.data
    }

    private toJSON() {
        return this.serialize()
    }
}
```

Výpis 4.2: Abstraktná trieda `BaseModel`, z ktorej vychádzajú všetky dátové modely.

Konkrétne dátové modely

Odvodené dátové modely predávajú do konštruktora nadradenej triedy (`BaseModel`) príslušnú validačnú Zod schému spolu so zdrojovými dátami. Taktiež definujú vlastné `getter` metódy, ktoré slúžia na prístup k jednotlivým atribútom modelu. Konkrétny príklad je zobrazený vo výpise 4.3. Tento prístup poskytuje vyššiu kontrolu nad tým, ako sa k atribútom pristupuje, čo zaisťuje väčšiu flexibilitu implementácie pre prípadné budúce úpravy interného spracovania dát v modeloch.

```
export class SearchTickerModel extends BaseModel<typeof partialTickerSchema> {
    constructor(data: unknown) {
        super(partialTickerSchema, data)
    }

    get symbol() {
        return this._getAttribute("ticker")
    }

    get name() {
        return this._getAttribute("name")
    }

    // ...
```

Výpis 4.3: Hlavná časť implementácie dátového modelu `SearchTickerModel`

4.3 Proces získavania a spracovania dát

Za účelom lepšej separácie aplikačnej logiky od prístupu k dátam a väčšej modularity je na strane servera použitý návrhový vzor *repository*. V tejto podkapitole je popísaný spôsob implementácie dátových repozitárov, ich úloha v prístupe k dátam a využívanie databázových transakcií pre zaistenie integrity databázy.

Repozitáre pre prístup k dátam

V projekte sú implementované repozitáre, ktoré poskytujú statické metódy na získavanie dát z databázy a z externých API služieb, ako aj na ukladanie dát do databázy. Podľa internej konvencie sú metódy pracujúce s API tretích strán označené prefixom `fetch` a metódy, ktoré pracujú s databázou prefixami `get` alebo `store`.

Každá metóda repozitára transformuje získané surové dáta do zodpovedajúcich inštancií dátových modelov. Aplikačná logika tak pracuje výhradne s validovanými, typovo bezpečnými objektami. Zároveň to znamená, že ak vstupné dáta nie sú validné, počas vytvárania inštancií modelov môže dôjsť k vyvolaniu výnimky, ktorú aplikácia musí korektne spracovať. Všetky repozitáre očakávajú, že ich volajúci kód zabezpečí ošetrenie týchto chybových stavov, ako aj potenciálnych chýb databázových operácií, sieťových chýb a chýb vzniknutých pri komunikácií s externými API službami.

Konkrétny repozitár

Vo výpise 4.4 nižšie je uvedený príklad jednoduchej metódy `getTickerDetails()` repozitára pre aktíva, ktorá na základe symbolu alebo interného identifikátora záznamu o aktíve z

databázy vráti inštanciu dátového modelu `TickerModel`. V prípade, že sa v databáze dané aktívum nenachádza, metóda vyvolá výnimku typu `ResourceError`.

```
export class TickersRepository {

    static async getTickerDetails(ticker: string | number) {
        const db = useDb()

        const [details] = await db.select()
            .from(tableTickers)
            .where(typeof ticker === "string" ? eq(tableTickers.symbol, ticker) :
                eq(tableTickers.id, ticker))

        if (!details) {
            throw new ResourceError("resource:not-found")
        }

        return new TickerModel(details)
    }

    // ...
```

Výpis 4.4: Metóda v repozitári pre aktíva, ktorá získa detaily aktíva z databázy.

Transakčné spracovanie databázových operácií

V projekte existujú prípady, kedy je potrebné manipulovať s dátami vo viacerých databázových tabuľkách atomicky. Hlavným príkladom je vkladanie obchodných agregácií do databázy, pri ktorom je treba okrem samotných záznamov o obchodoch súčasne uložiť aj záznamy o časových intervaloch, ktoré tieto dáta pokrývajú. Zároveň môže dôjsť k prekrývaniu intervalov alebo doplneniu chýbajúcej časti dát medzi dvoma intervalmi, čo vyžaduje zlúčenie daných časových rozsahov. Všetky tieto operácie musia prebehnúť spoločne pre zachovanie integrity databázy a pokiaľ nejaká z nich zlyhá, je nutné všetky vykonané operácie vrátiť. O túto funkcionalitu sa stará ORM nástroj Drizzle, ktorý poskytuje jednoduché rozhranie na prácu s databázovými transakciami. Tie sú definované pomocou metódy `transaction()`, ktorá prijíma callback funkciu s kontextom transakcie ako prvým argumentom, ktorý poskytuje rozhranie pre vykonávanie databázových operácií v rámci jednej transakcie. Príklad využitia transakcií je uvedený nižšie vo výpise 4.5.

```
await db.transaction(async (tx) => {
    // ...
    // create a new range
    await tx.insert(tableTradeAggregationsRanges)
        .values({
            ticker_id: tickerId,
            from: data.range.from,
            to: data.range.to,
        })
    // ...
    // insert the new entries
    await tx.insert(tableTradeAggregations)
        .values(
            // ...
        )
})
```

Výpis 4.5: Zjednodušený príklad metódy `storeAggregations()`.

4.4 Implementácia REST API

HTTP REST API tvorí základnú komunikačnú vrstvu medzi backendom a klientskou aplikáciou. Nasledujúce sekcie popisujú najdôležitejšie technické informácie o jej implementácií. Popis štruktúry jednotlivých endpointov a ich poskytovaného rozhrania sa nachádza v kapitole 3.4.

Formát odpovedí a chýb

Všetky endpointy v aplikácii využívajú identickú štruktúru JSON odpovedi bez ohľadu na typ požiadavku (kolekcia alebo detail modelu). Pre vyššiu flexibilitu a možnosť prípadného rozširovania o dodatočné metadáta sa nachádza hlavný obsah odpovede vždy pod kľúčom `data`, ktorý obsahuje pole v prípade kolekcie alebo objekt v prípade požiadavky na detail nejakej entity. Táto jednotná štruktúra odpovedí zjednodušuje spracovanie na strane klienta, pretože je možné vždy predpokladať konzistentnú formu výsledného JSON objektu.

Server využíva štandardný prístup k spracovaniu chýb pomocou konštrukcií `try/catch`, ktoré sú súčasťou jazyka JavaScript. Tento spôsob umožňuje oddeliť zodpovednosti spracovania konkrétnych chýb do príslušných častí kódu. V rámci implementácie je definovaná vlastná trieda výnimky s názvom `_CustomError`, ktorá poskytuje metódu `toH3Error()`. Táto metóda prevádza internú výnimku na formát kompatibilný so serverovým prostredím Nitro, čím je zabezpečené správne zobrazenie chyby v HTTP odpovedi.

Proces spracovania HTTP požiadavky

Spracovanie požiadaviek na serveri je postavené na funkcii `defineEventHandler()` poskytovanej frameworkom `h3`, ktorá slúži na definovanie jednotlivých endpointov. Každý endpoint je definovaný ako samostatná asynchrónna funkcia, v ktorej sa vykonávajú všetky kroky potrebné k zostaveniu odpovede pre klienta. Prvým krokom je spravidla validácia vstupných URL query parametrov. Na tento účel sa používa knižnica `Zod`, rovnako ako pri validácii dát v dátových modeloch. Ak parametre nevyhovujú požadovanej schéme — či už typom, formátom alebo logickou kombináciou — je ešte pred vykonaním akejkoľvek ďalšej logiky požiadavka ukončená so stavovým HTTP kódom `400 Bad Request`.

Po úspešnej validácii parametrov sa realizuje spracovanie požiadavky pomocou prístupu k dátam cez repozitáre. Výsledné dáta sú následne zabalené do objektu triedy `ApiResponse`, ktorá udržiava konzistentnú štruktúru JSON výstupu vo všetkých endpointoch.

Niektoré endpointy okrem validácie a čítania dát z databázy obsahujú aj komplexnejšiu logiku, ktorá zabezpečuje dynamické doplnenie chýbajúcich dát v prípade, že pre vyžiadaný časový úsek nie sú v databáze dostupné. Tieto endpointy môžu registrovať neblokujúce asynchrónne operácie pomocou funkcie `event.waitUntil()`, ktoré sa vykonajú bez zdržania odpovede pre klienta — napríklad uloženie aktualizovaných informácií do databázy po ich získaní.

Kapitola 5

Implementácia klientskej aplikácie

Táto kapitola popisuje hlavné časti implementácie klientskej aplikácie, vrátane architektúry vlastného API klienta na komunikáciu s backendom, práce s dátami v aplikácii a implementácie používateľského rozhrania. Riešenie kladie dôraz na efektivitu, výkonnosť, dobrú organizáciu a čitateľnosť kódu. Niektoré časti implementácie vychádzajú z verejne dostupných zdrojov alebo boli nimi inšpirované. Ich bližší popis sa nachádza v prílohe A.

5.1 API klient

Pre komunikáciu s backendovým REST API sa v aplikácii využíva vlastná implementácia API klienta postavená na triede `API.Service`. Celá implementácia je združená pod menným priestorom `API`, čím sa dosahuje vyššia prehľadnosť a jasné oddelenie od zvyšku aplikačnej logiky. Trieda `API.Service` poskytuje jednotné rozhranie pre všetky požiadavky smerujúce na server. Zabezpečuje vytváranie správne naformátovaných URL adries, vykonávanie požiadaviek prostredníctvom `URLSession`, dekódovanie odpovedí na očakávané dátové modely a spracovanie chybových stavov API.

API.URLBuilder

Základom pre vykonanie HTTP požiadavky je vytvorenie URL, čo zabezpečuje trieda `API.URLBuilder`. Táto trieda implementuje protokol `API.URLBuildable` (zobrazený vo výpise 5.1) a umožňuje nastavenie základnej API URL adresy, postupné dosadzovanie ciest pre endpointy a nastavovanie query parametrov.

Tento protokol implementuje aj trieda `API.Service`, ktorá takýmto spôsobom sprístupňuje metódy objektu `API.URLBuilder`, ktorý je v `API.Service` využívaný interne. Zvolený prístup umožňuje vytvárať API požiadavky pomocou návrhového vzoru *builder pattern*. Toto rozhranie je inšpirované ORM nástrojmi, ako napríklad Drizzle, ktorý je využívaný na strane servera.

```
protocol URLBuildable {
    /// Add a new route parameter to the URL. Example: /path/param
    func addRouteParam(_ value: String?) -> Self
    /// Set a custom URL query parameter. Example: ?key=value
    func setQueryParam(_ key: String, _ value: Encodable?) -> Self
    /// A shorthand for setting the LAST route parameter
    func forId<T: CustomStringConvertible>(_ id: T) -> Self
}
```

Výpis 5.1: Príklad protokolu `API.URLBuildable`.

API.Service

Táto trieda obsahuje základnú logiku na odosielanie HTTP požiadaviek a spracovanie odpovedí zo servera. Interne trieda využíva pomocný objekt `API.URLBuilder`, ktorý sa stará o vytvorenie finálnej URL štruktúry. Táto URL je následne použitá na vytvorenie štruktúry `URLRequest`, ktorá je odoslaná pomocou `URLSession` vo vlastnej metóde `fetch`, zobrazenej vo výpise 5.2. Všetky tieto sieťové štruktúry a trieda vykonávajúca HTTP požiadavky sú súčasťou knižnice jazyka Swift. Po obdržaní odpovede sa pomocou triedy `API.Response` dekódujú JSON dáta a prevedú sa na modely.

```
private func fetch(request: URLRequest) async throws -> API.Response<Model> {
    let (data, response) = try await URLSession.shared.data(for: request)

    guard let httpResponse = response as? HTTPURLResponse else {
        throw URLError(.badServerResponse)
    }

    if httpResponse.statusCode != 200 {
        throw API.ResponseError(data: data)
    }

    return try API.Response<Model>(data: data, response: httpResponse)
}
```

Výpis 5.2: Asynchrónna metóda `fetch()`, ktorá zabezpečuje odoslanie požiadavky.

```
let tradeAggregationsResponse = try await API.getTradesApiService()
    .forId("AAPL")
    .setQueryParam("timeframe", "week")
    .get()

tradeAggregations = tradeAggregationsResponse.getItems()
```

Výpis 5.3: Príklad vykonania HTTP GET požiadavky pomocou `API.Service`.

API.Response

Táto trieda reprezentuje odpoveď servera vo forme, s ktorou pracuje klientská aplikácia. Pri konštrukcii jej inštancie sa vykonáva dekódovanie JSON dát pomocou objektu triedy `JSONDecoder` z knižnice jazyka Swift, pričom sa využíva pomocná štruktúra `ApiResponseData`, ktorá reprezentuje formát odpovede zo servera a je schopná spracovať odpovede rôznych typov. Pre toto spracovanie sa využíva pomocný výčtový typ `SingleValueOrArray`, ktorý automaticky rozpozná, či ide o odpoveď so samostatným modelom alebo s kolekciou. Tento enum umožňuje jednotné spracovanie odpovedí bez potreby rozlišovať ich typ vo volajúcom kóde.

Dekódované dáta sú následne dostupné pomocou dvoch metód, podľa typu odpovede:

- `getItem()` – vráti objekt modelu v prípade, že odpoveď obsahovala detail entity, inak vracia `nil`,
- `getItems()` – vráti pole objektov modelov v prípade, že odpoveď obsahovala kolekciu, inak vracia prázdne pole

Táto abstrakcia výrazne zjednodušuje prácu s odpoveďami na strane klienta, poskytuje odolnosť voči zmenám v štruktúre API odpovedí a zvyšuje čitateľnosť kódu.

5.2 Manipulácia s dátami v aplikácii

Aplikácia pracuje s dvoma hlavnými typmi dátových zdrojov — lokálne používateľské dáta a externé dáta získavané z backendového API. Lokálne dáta sú reprezentované entitou `Trade` v rámci databázového systému CoreData. Záznamy tejto entity slúžia pre výpočet hodnoty portfólia, štatistík a zobrazenie histórie obchodov. Externé dáta, ako sú napríklad informácie o aktívach, sú získavané prostredníctvom API klienta, v ktorom sú zároveň aj dekódované do zodpovedajúcich modelov. Tie aplikácia následne využíva vo výpočtoch a na zobrazovanie informácií.

Pri práci s dátami sa využívajú view modely vytvorené pomocou makra `@Observable`, ktoré umožňuje reaktívne prepojenie používateľského rozhrania s konkrétnymi premennými, ktoré UI využíva. Výsledkom tohto prístupu je efektívnejšie spracovanie prekreslení používateľského rozhrania, čo zvyšuje výkonnosť aplikácie.

Centralizovaný view model s názvom `AppViewModel` spravuje prístup k lokálnym dátam prostredníctvom objektu `TradesManager` a taktiež zabezpečuje načítanie dát potrebných na výpočet hodnoty portfólia z backendu. Tieto získané dáta sa udržiavajú v pamäti po celú dobu behu aplikácie za účelom zvýšenia používateľského komfortu a zníženia záťaže na backendový server. Tento prístup sa využíva pre viaceré dáta v rôznych view modeloch a umožňuje napríklad efektívne prepínanie medzi rôznymi časovými rámcami bez potreby opätovného načítania rovnakých údajov.

5.3 Používateľské rozhranie aplikácie

V tejto časti je popísaný spôsob implementácie používateľského rozhrania aplikácie, ktoré bolo navrhnuté tak, aby poskytovalo intuitívny, prehľadný, vizuálne konzistentný a zaujímavý spôsob sledovania portfólia a získavania nových poznatkov o základoch investovania. Pri implementácií boli využité moderné dizajnové princípy s technológiami frameworku SwiftUI. Na obrázku 5.1 sú zobrazené hlavné obrazovky aplikácie.

HomeView

Domovská obrazovka aplikácie zobrazuje graf hodnoty portfólia za vybraný časový úsek (týždeň, mesiac, rok, 5 rokov) spolu s ďalšími údajmi o používateľovom portfóliu. Po výbere časového intervalu sa vo view modeli aplikácie spustí asynchrónny blok kódu pomocou Swift konštrukcie `Task`, ktorý získa požadované dáta pre graf z backendu. Tieto dáta sú potom priradené do premennej `portfolioChartEntries`, čím sa graf automaticky aktualizuje. Nové dáta sa zároveň uložia do pamäti aby sa predišlo opätovnému sťahovaniu rovnakých dát počas doby behu aplikácie.

Na tejto obrazovke sú tiež priamo zobrazené rôzne štatistiky o aktuálnom portfóliu, ako napríklad jeho aktuálna hodnota, celková investovaná suma, počet aktív v portfóliu, dátum posledného obchodu a iné. Tieto štatistiky je možné zobraziť aj v podrobnejšej forme v prehľade celého portfólia prostredníctvom pohľadu `PortfolioMetricsSheetView`, ktorý je doplnený aj o grafy zobrazujúce napríklad mesačné zisky, počet vykonaných transakcií za mesiac, percentuálne zloženie portfólia a podobne. Výpočet údajov pre tento prehľad je náročnejšia operácia, ktorá by pri vykonávaní na hlavnom vlákne mohla spôsobiť dočasné zamrznutie používateľského rozhrania. Z tohto dôvodu view model pre `HomeView` vykonáva túto operáciu asynchrónne vo vlákne na pozadí pomocou Swift konštrukcie `Task.detached`.

Celý pohľad `HomeView` je rozdelený na dve časti a využíva sa tu špeciálne gesto na prechod medzi detailom portfólia a obrazovkou s výpisom aktív používateľovho portfólia. Potiahnutím nahor na obrazovke detailu sa obsah plynulo rozmazáva a stmavuje. Zároveň sa postupne zvýrazňuje vznášajúce sa tlačidlo so šípkou nadol. Po dosiahnutí určitého prahu posunutia obrazovky sa tlačidlo zarovná do stredu obrazovky, čo značí splnenie podmienok pre vykonanie navigácie. Uvoľnením prsta sa tento prechod medzi obrazovkami vykoná. Kliknutím na tlačidlo je možné vykonať rovnakú akciu bez tohto gesta. Prechod naspäť je obdobný, vyžaduje však potiahnutie smerom dolu. Definované SwiftUI animácie zabezpečujú plynulý prechod medzi obrazovkami. Pre implementáciu tohto gesta bol využitý view modifikátor `.onScrollGeometryChange`, ktorý sleduje zmenu posunutia obrazovky a vlastné gesto `ReleaseGesture`, implementujúce protokol `UIGestureRecognizerRepresentable`, ktoré umožňuje detekciu uvoľnenia dotyku po posunutí obrazovky.

Z domovskej obrazovky je možné prejsť na obrazovku vyhľadávania aktív (`AssetListView`) po výbere typu aktíva v hárku s `BrowseAssetsTypeSelectionView`. Zo spodnej časti domovskej obrazovky, kde sa nachádza výpis aktív v portfóliu, je následne možné prejsť na obrazovku detailu konkrétneho aktíva (`AssetDetailView`).

Obr. 5.1: Hlavné obrazovky aplikácie.

AssetListView

Táto obrazovka slúži na vyhľadávanie jednotlivých aktív na základe používateľského vstupu. Prvotne sa zobrazuje prednačítaný zoznam aktív, ktoré sú uložené v databáze servera. Po zadaní vstupu do textového poľa na vrchu obrazovky sa vo view modeli špecifickom pre túto obrazovku spustí vyhľadávanie s využitím knižnice `Combine`. Táto knižnica poskytuje tzv. *Publishery*, čo sú objekty, ktoré vysielajú hodnoty v čase a umožňujú ich reaktívne spracovanie prostredníctvom naviazaných *Subscriberov*. Pre prípad vyhľadávania sa využíva modifikátor `.debounce`, ktorý redukuje počet vykonaných HTTP požiadaviek na endpoint s výsledkami vyhľadávania pri písaní textu. View modely vytvorené pomocou makra `@Observable` nespolupracujú s knižnicou `Combine` priamo, a preto sú prvky používateľského rozhrania naviazané na premenné s explicitne definovanými `get` a `set` metódami, pričom v `set` metódach dochádza k odosielaniu nových hodnôt cez príslušných Publisherov.

AssetDetailView

Obrazovka `AssetDetailView` zobrazuje informácie o konkrétnom aktíve vrátane historického vývoja jeho hodnoty vo forme grafu. Načítavanie dát a ich cachovanie pre jednotlivé časové intervaly prebieha podobne ako na domovskej obrazovke, avšak prostredníctvom vlastného view modelu špecifického pre túto obrazovku. V prípade, že ide o zobrazenie detailu ETF fondu, je tu navyše možné prezerať jeho zloženie podľa aktív alebo sektorov. Z tejto obrazovky je taktiež možné prejsť na obrazovku porovnania dvoch aktív s názvom `AssetComparisonView`.

Používateľ tu má možnosť vytvárať nové záznamy o nákupoch daného aktíva a v prípade, že sa aktívum už nachádza v jeho portfóilu, sú zobrazené aj súvisiace investičné štatistiky a zoznam transakcií, ktoré je možné individuálne upravovať. Obrazovka taktiež obsahuje aj edukačné prvky vo forme hárkov (*sheets*) s vysvetľujúcim textom k investičným metrikám a grafickým znázornením hodnôt, ktoré uľahčuje porovnanie a pochopenie kontextu. Podobne ako na domovskej obrazovke, aj tu je možné zobraziť podrobné štatistiky o transakciách pre dané aktívum v portfóliu cez hárok, ktorý zobrazuje pohľad `AssetDetailTradeMetrics-View`.

Vizuálnu stránku obrazovky dopĺňajú animované prechody numerických textových hodnôt založené na SwiftUI animáciách, ako napr. `.contentTransition(.numericText())`.

Kapitola 6

Testovanie

Táto kapitola sa zameriava na overenie správnej funkčnosti a výkonnosti vypracovaného riešenia. Kapitola popisuje prístup záťažového testovania serverovej časti, ako aj testovanie aplikácie v simulátore počas vývoja a finálne testovanie s reálnymi používateľmi pri využití modelového portfólia. Cieľom testovania bolo identifikovať nedostatky, overiť funkčnosť a stabilitu a zabezpečiť, že aplikácia spĺňa očakávania používateľov.

6.1 Testovanie servera

Serverová časť projektu bola testovaná kombináciou jednotkových testov implementovaných pomocou testovacieho frameworku `Vitest` a záťažového testovania vykonaného pomocou nástroja `wrk`, ktorý je určený na testovanie výkonnosti HTTP rozhraní. Výsledky týchto testov sú uvedené vo výpisoch 6.1, 6.2, 6.3 a 6.4.

Jednotkové testy overujú najmä správnosť validácie vstupných dát a korektnosť implementácie dátových modelov. Testuje sa vytváranie inštancií a očakávané vyvolávanie výnimiek aby sa zamedzilo vytvoreniu modelov s nesprávnymi dátami, spracovanie voliteľných atribútov modelov a správna serializácia výstupov.

Záťažové testovanie bolo zamerané na analýzu výpočtovej kapacity servera pri spracovaní veľkého množstva požiadaviek a na posúdenie vhodnosti aktuálnej implementácie pre produkčné prostredie. Keďže projekt využíva bezplatný plán služby Vercel s obmedzeným výkonom a počtom požiadaviek, server bol pre účely tohto testovania spustený v produkčnom režime lokálne. Týmto sa predišlo obmedzeniam zo strany Vercelu a umožnilo to získať realistickejší obraz o výkone servera. Pripojenie na produkčnú PostgreSQL databázu nasadenú na Verceli však zostalo zachované pre konzistentnosť konfigurácie s reálnym použitím. Tento prístup však mohol viesť k zvýšenej latencii pri komunikácií s databázou, keďže pri reálnom nasadení prebieha komunikácia cez optimalizovanú internú sieť Vercelu.

Testovanie prebiehalo na zariadení Apple MacBook Pro s čipom M2 Max (12 jadier CPU, 38 jadier GPU) a 96 GB pamäte v prostredí Node.js vo verzii 22.14.0. Vzhľadom na to, že optimalizácia výkonnosti servera nebola primárnym cieľom projektu, sú dosiahnuté výsledky uspokojivé. Využitie cachovania na strane klienta zároveň výrazne znižuje počet požiadaviek na server, čo ďalej prispieva k jeho škálovateľnosti.

Výsledky záťažových testov

```
> ./wrk -t10 -c100 -d10s http://localhost:3000/api/tickers/AAPL
Running 10s test @ http://localhost:3000/api/tickers/AAPL
  10 threads and 100 connections
  Thread Stats Avg Stdev Max +/- Stdev
    Latency 642.70ms 283.29ms 1.94s 90.75%
    Req/Sec 21.93 15.57 70.00 65.93%
  1578 requests in 10.08s, 2.10MB read
  Socket errors: connect 0, read 0, write 0, timeout 1
Requests/sec: 156.54
Transfer/sec: 213.11KB
```

Výpis 6.1: Výkonnostný test API endpointu pre detail aktíva, ktorý simuloval 100 súbežných spojení počas 10 sekúnd a potvrdil stabilnú obsluhu požiadaviek s minimom chýb. Endpoint zvláda v priemere 156 požiadaviek za sekundu.

```
> ./wrk -t10 -c100 -d10s http://localhost:3000/api/tickers?type=cs
Running 10s test @ http://localhost:3000/api/tickers?type=cs
  10 threads and 100 connections
  Thread Stats Avg Stdev Max +/- Stdev
    Latency 620.72ms 100.39ms 797.42ms 88.16%
    Req/Sec 19.96 14.43 90.00 78.01%
  1571 requests in 10.09s, 15.85MB read
Requests/sec: 155.66
Transfer/sec: 1.57MB
```

Výpis 6.2: Test endpointu pre zoznam aktív, ktorý simuloval 100 súbežných spojení počas 10 sekúnd a zvláda priemerne 155 požiadaviek za sekundu.

```
> ./wrk -t10 -c100 -d10s http://localhost:3000/api/tickers/search\?q\=appl
Running 10s test @ http://localhost:3000/api/tickers/search?q=appl
  10 threads and 100 connections
  Thread Stats Avg Stdev Max +/- Stdev
    Latency 229.44ms 179.76ms 1.16s 92.46%
    Req/Sec 55.20 26.78 101.00 67.13%
  5002 requests in 10.07s, 7.51MB read
Requests/sec: 496.87
Transfer/sec: 763.74KB
```

Výpis 6.3: Test endpointu pre vyhľadávanie aktív podľa vstupu používateľa, ktorý dosiahol približne 496 požiadaviek za sekundu s priemernou latenciou pod 230 ms, čo je uspokojivý výsledok nakoľko tu prebieha komunikácia s externou službou.

```
> ./wrk -t10 -c10 -d10s http://localhost:3000/api/trades/AAPL\?timeframe\=week
Running 10s test @ http://localhost:3000/api/trades/AAPL?timeframe=week
  10 threads and 10 connections
  Thread Stats Avg Stdev Max +/- Stdev
    Latency 303.20ms 37.19ms 546.93ms 81.65%
    Req/Sec 3.00 0.59 5.00 77.37%
  327 requests in 10.09s, 562.67KB read
Requests/sec: 32.40
Transfer/sec: 55.76KB
```

Výpis 6.4: Test endpointu pre agregované obchodné dáta s týždenným časovým rámcom. Test s 10 súbežnými spojeniami ukazuje stabilnú latenciu približne 303 ms a spoľahlivú obsluhu s 32 požiadavkami za sekundu. Toto je oblasť, na ktorú by sa prípadná optimalizácia mala zamerať ako prvé.

6.2 Testovanie aplikácie v simulátore

Počas vývoja aplikácie prebiehalo priebežné testovanie v simulátore systému iOS (verzia 18.3.1) na zariadení iPhone 16 Pro dostupnom vo vývojovom prostredí `XCode`. Cieľom bolo overiť správnu funkčnosť aplikácie, navigáciu medzi obrazovkami a reakcie používateľského rozhrania na interakcie v aplikácii. V simulátore boli testované scenáre bežného používania, ako aj hraničné prípady – napríklad neplatné vstupy, chybné odpovede serveru a podobne. Testovanie sa tiež zameriavalo na kontrolu výkonnosti aplikácie a ladenie chýb.

6.3 Testovanie používateľmi s využitím modelového portfólia

Pre účely testovania reálneho používania aplikácie bolo vypracované modelové portfólio simulujúce správanie začínajúceho investora. Zvolená bola investičná stratégia *Dollar Cost Averaging*, v rámci ktorej boli do portfólia pravidelne každý mesiac, vždy v 8. deň, počas obdobia jedného roka (od 8.5.2024 do 8.5.2025) investované nasledovné fixné čiastky do troch vybraných aktív:

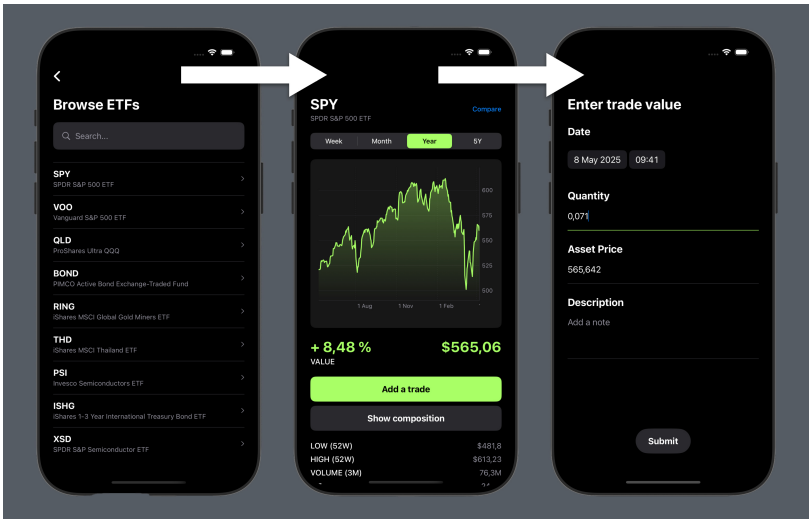
- **S&P 500 ETF** – **40$**, ako stabilná voľba pre zaistenie diverzifikácie,
- **Johnson & Johnson** – **30$**, ako defenzívna akcia,
- **Tesla, Inc.** – **30$**, ako dynamická rastová akcia.

Cieľom tohto testovania bolo overiť praktickú použiteľnosť aplikácie pri analýze a správe portfólia, vrátane pridávania a upravovania transakcií, sledovania výkonnosti investícií, správneho výpočtu analytických metrík a celkovej zrozumiteľnosti prezentovaných informácií.

Do testovania boli zapojení dvaja používatelia – študent vysokej školy s predchádzajúcimi základnými skúsenosťami v oblasti investovania, ktorý si vedie vlastné investičné portfólio u viacerých brokerov, a študentka posledného ročníka strednej školy bez predchádzajúcich skúseností s investovaním. Prvý používateľ dostal prázdnu aplikáciu a na základe poskytnutého výpisu transakcií mal sám vytvoriť celé modelové portfólio v aplikácii. Druhá používateľka dostala aplikáciu s už vyplneným portfóliom a bez ďalších inštrukcií o používaní aplikácie alebo vysvetlenia základných pojmov z teórie investovania dostala konkrétne úlohy, ktoré mala v aplikácii vykonať. Testovanie prebiehalo na reálnom zariadení iPhone 16 Pro Max s verziou systému iOS 18.4.1.

Vytváranie portfólia v aplikácii

Používateľ začal s aplikáciou v počiatočnom stave bez akýchkoľvek dát a podľa poskytnutého výpisu transakcií postupne vytvoril v aplikácii celé portfólio. Na úvodnej obrazovke správne identifikoval potrebu kliknúť na tlačidlo `Browse`, pomocou ktorého prešiel na obrazovku vyhľadávania aktív. Po výbere konkrétneho aktíva cez tlačidlo `Add a trade`, ktoré sa nachádza na obrazovke detailu aktíva, prešiel na formulár pre pridanie novej transakcie. Tento proces prechodu z výpisu aktív až po formulár pre pridanie transakcie je znázornený na obrázku 6.1. Postupne takto zadal všetky transakcie pre jednotlivé aktíva, čím vytvoril v aplikácii kompletné modelové portfólio zodpovedajúce zadaniu.



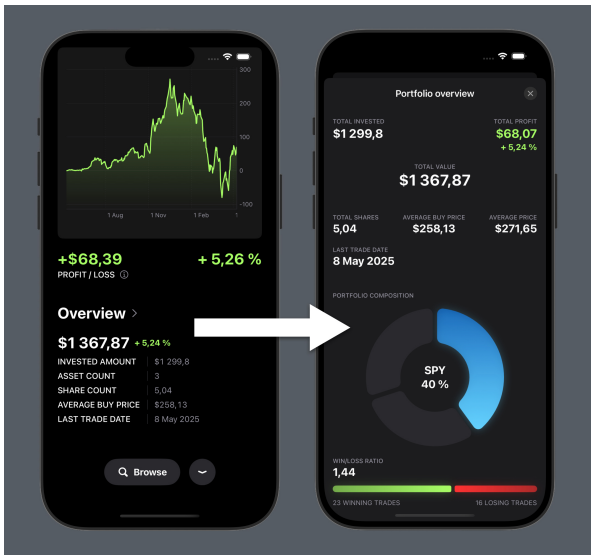
Obr. 6.1: Proces pridávania transakcie pre S&P 500 ETF.

Interakcia používateľa bez predchádzajúcich skúseností

Toto testovanie prebiehalo formou postupného zadávania úloh, ktoré mala používateľka samostatne vyriešiť v aplikácii. Testy boli zamerané na overenie zrozumiteľnosť používateľského rozhrania a prístupnosti dôležitých informácií pre niekoho, kto nemá žiadne skúsenosti s investovaním.

1. *Zisti aktuálnu hodnotu portfólia a celkový zisk v porovnaní s investovanou sumou.*

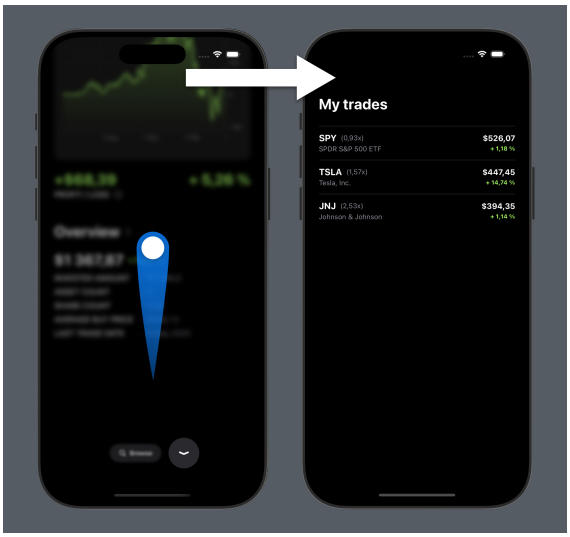
   Pri tejto úlohe používateľka okamžite rozpoznala, že veľký text na domovskej obrazovke zobrazuje aktuálnu hodnotu portfólia. Pre potvrdenie svojej domnienky klikla na tlačidlo `Overview`, čím sa jej zobrazil prehľad s ďalšími metrikami, ktoré obsahovali aj presnú hodnotu celkového zisku. Túto obrazovku vnímala ako zrozumiteľnú a prehľadne spracovanú, pričom obzvlášť ocenila jej farebné spracovanie, ako je možné vidieť na obrázku 6.2.



Obr. 6.2: Zobrazenie prehľadu portfólia.

2. *Ktoré aktívum v portfóliu dosahuje najvyššiu výnosnosť (zisk v %) oproti nákupnej cene a ktoré aktívum tvorí najväčší podiel na celkovej hodnote portfólia?*

   S druhou úlohou nemala používateľka žiadne problémy a okamžite prešla pomocou špeciálneho gesta, znázorneného na obrázku 6.3, na zoznam aktív v portfóliu. Spôsob prechodu medzi obrazovkami jej nebol dopredu vysvetlený a považovala ho za prirodzený a intuitívny. Výpis aktív hodnotila ako prehľadný, s dostatočným množstvom potrebných informácií, a bola schopná bez problémov identifikovať požadované aktíva.



Obr. 6.3: Prechod na prehľad všetkých aktív v portfóliu.

3. *V ktorom mesiaci za posledný rok zaznamenalo portfólio najvyšší zisk?*

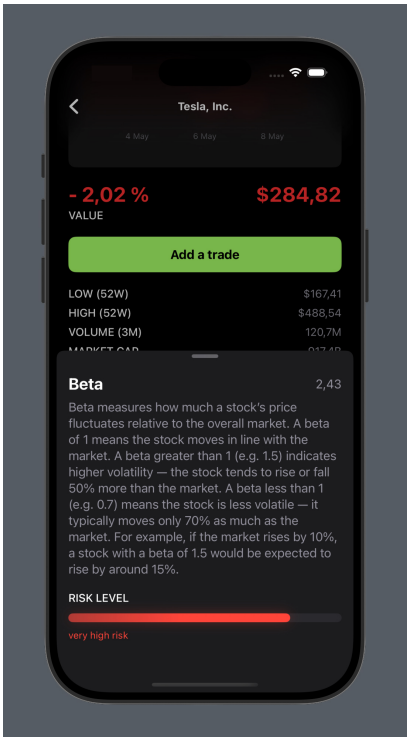
   Aj túto úlohu vyriešila veľmi rýchlo, keďže obrazovku s prehľadom portfólia si zobrazila už pri riešení prvej úlohy. Potrebný graf, zobrazený v spodnej časti obrázku 6.4, ihneď identifikovala a bez problémov z neho vyčítala potrebnú hodnotu.

Obr. 6.4: Zobrazenie prehľadu portfólia.

4. *Ktoré aktívum v portfóliu má najväčšie cenové výkyvy (volatilitu)? A ktoré naopak kolíše najmenej?*

   Táto úloha jej spôsobila menšie ťažkosti, pretože sa spočiatku pokúsila vyčítať požadovanú hodnotu z grafov jednotlivých aktív. Následne si všimla sekciu metrík zobrazených pod tlačidlom `Add a trade`, kde postupne prechádzala jednotlivé hodnoty, až nakoniec objavila koeficient *beta*, ktorý mala aplikáciou vysvetlený, ako je znázornené na obrázku 6.5. Vďaka tomu dokázala identifikovať požadované aktíva.

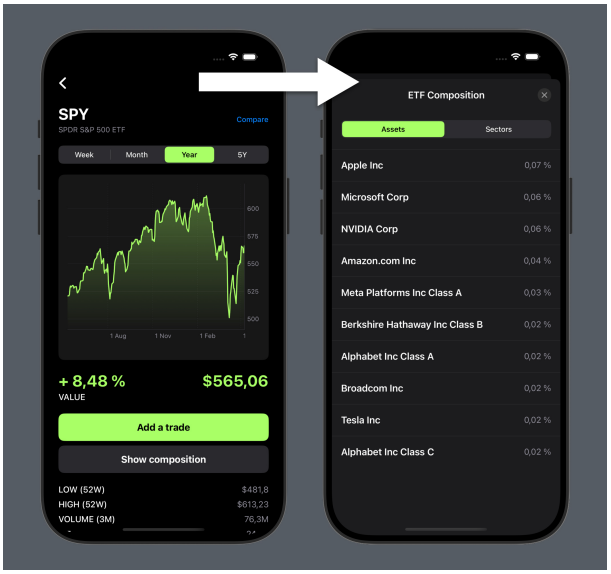
   Pri tejto úlohe bol odhalený nedostatok v používateľskom rozhraní – pri aktívach sa zobrazovali iba tie metriky, ktoré boli pre dané aktívum dostupné. To znamená, že v prípadoch, keď server nepoznal hodnotu určitej metriky (napríklad koeficient *beta* pri niektorých ETF fondoch), príslušný riadok sa vôbec nezobrazil. Tento prístup mohol u menej skúsených používateľov spôsobiť nejasnosti. Preto bolo na základe tohto zistenia používateľské rozhranie upravené tak, aby v prípade nedostupnej hodnoty zobrazovalo reťazec „n/a“.



Obr. 6.5: Zobrazenie nápovedy pre koeficient *beta* reprezentujúci volatilitu aktíva.

5. *Ktorá spoločnosť má najväčšie zastúpenie v ETF fonde nachádzajúcom sa v portfóliu?*

   Posledná úloha bola ešte o niečo náročnejšia, pretože používateľka nemala žiadne znalosti o ETF fondoch. Istý čas prechádzala aplikáciou, až si nakoniec všimla rozdiel, že pri klasických akciách sa nenachádza tlačidlo `Show composition`. Po tomto zistení si dokázala zobraziť hlavné zloženie ETF fondu v portfóliu, ktoré je možné vidieť na obrázku 6.6, a správne identifikovať spoločnosť s najväčším zastúpením. Na základe pozorovania tohto testu boli do aplikácie neskôr pridané ďalšie nápovedy vysvetľujúce typy podporovaných aktív.



Obr. 6.6: Zobrazenie zloženia ETF fondu.

Vyhodnotenie testovania

Testovanie aplikácie s reálnymi používateľmi s rôznou úrovňou skúseností potvrdilo, že navrhnuté používateľské rozhranie je intuitívne a vhodné aj pre úplných začiatočníkov. Osoby pri testovaní dokázali pracovať samostatne a bez výraznejšej pomoci dosiahnuť požadované ciele.

Používateľ so základnými investičnými skúsenosťami ocenil najmä prehľadný graf zobrazujúci vývoj výnosov portfólia v čase, spracovanie investičných metrík a porovnávanie jednotlivých aktív, zatiaľ čo používateľka bez predchádzajúcich znalostí považovala za veľmi prínosné grafické spracovanie aplikácie, jednoduché a intuitívne ovládanie a pomocné nápovedy s vizualizáciami. Zároveň bola schopná vykonať vlastné zhodnotenie poskytnutého portfólia a poukázať na výhody pravidelného investovania.

Vďaka testovaniu sa tiež podarilo identifikovať niekoľko nedostatkov, ktoré boli neskôr vyriešené úpravou používateľského rozhrania.

V rámci ukážky na pripravenom modelovom portfóliu sa potvrdila schopnosť stratégie *Dollar Cost Averaging* znižovať vplyv krátkodobých výkyvov trhu. Pravidelné investovanie fixných čiastok umožnilo minimalizovať riziko nesprávneho načasovania obchodov. Táto metodika sa ukázala ako vhodná pre začínajúcich investorov pretože im poskytuje jednoduchý a systematický prístup k budovaniu svojho portfólia — čo potvrdili aj samotní používatelia počas testovania aplikácie.

Testovanie možno celkovo považovať za úspešné, pretože sa potvrdila schopnosť aplikácie efektívne vizualizovať vývoj hodnoty portfólia v čase, sledovať užitočné metriky portfólia, prezerať aktíva na akciových trhoch, porovnať výkonnosť jednotlivých investícií. Aplikácia sa tiež ukázala ako vhodná pre cieľovú skupinu začiatočníkov v oblasti investovania.

Kapitola 7

Záver

Cieľom tejto bakalárskej práce bolo navrhnúť a implementovať mobilnú aplikáciu, ktorá umožní začínajúcim investorom zobrazovať informácie potrebné na analýzu investičných stratégií pre ich portfólio zrozumiteľným a prehľadným spôsobom. Zároveň im má pomôcť získať potrebné znalosti na zostavenie portfólia podľa konzervatívnejších a bezpečnejších stratégií vhodných pre začiatočníkov. Výsledné riešenie reaguje na nedostatky existujúcich riešení a spája ich silné stránky do jednej aplikácie, ktorá sa zameriava na jednoduchosť použitia, zrozumiteľnosť a vizuálnu konzistenciu so systémom iOS.

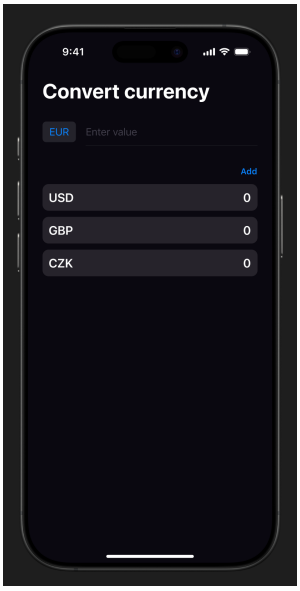
V práci bola implementovaná kompletná architektúra systému vrátane serverovej časti, databázovej schémy a klientskej aplikácie pre platformu iOS. Server slúži ako jednotné rozhranie pre poskytovanie dát klientovi, pričom kombinuje viacero externých poskytovateľov dát a umožňuje ich jednoduchú výmenu v prípade potreby. Zároveň prináša zvýšenie efektivity vďaka rôznym mechanizmom cachovania. Mobilná aplikácia umožňuje používateľom sledovať aktuálny stav svojho investičného portfólia, porovnávať výkonnosť jednotlivých aktív a analyzovať svoje investície na základe historických dát. Taktiež obsahuje aj edukačné prvky, ktoré umožňujú začínajúcim investorom lepšie porozumieť investičnej terminológii.

Server bol otestovaný pomocou jednotkových testov a výkonnostných testov. Funkčnosť a použiteľnosť aplikácie bola overená testovaním s reálnymi používateľmi, ktoré potvrdilo jej vhodnosť pre cieľovú skupinu začiatočníkov.

7.1 Možné budúce rozšírenia

Mobilná aplikácia spolu s navrhnutou backendovou architektúrou poskytuje dobrý základ pre ďalšie rozšírenia, ktoré by mohli obohatiť existujúce funkcie, rozšíriť podporu pre rôzne typy aktív alebo pridať nové analytické nástroje. Takéto vylepšenia by mohli pomôcť osloviť širšie spektrum investorov. Zároveň je však dôležité zachovať jednoduché, prehľadné a zrozumiteľné používateľské rozhranie pre začiatočníkov. Pri budúcich rozšíreniach by preto aplikácia mala podporovať dynamické prispôsobenie úrovne zložitosti zobrazenia na základe znalostí používateľa a zároveň pokračovať v pridávaní edukačných prvkov.

Jedným z rozšírení by mohla byť napríklad implementácia podpory devízového trhu (FOREX), ktorá by zahŕňala sledovanie vývoja menových kurzov, prepočty hodnoty portfólia do rôznych základných mien, prevodník mien podľa aktuálnych kurzov (zobrazený na obrázku 7.1) a analytické nástroje podobné tým, ktoré sú v súčasnosti implementované v aplikácii pre akciové trhy.



Obr. 7.1: Návrh prevodníka mien.

Ďalším rozšírením by mohla byť podpora pre vytváranie simulácií investičných stratégií priamo v aplikácii. Tieto stratégie by bolo možné testovať na historických dátach a ich výsledky vizualizovať v porovnaní s portfóliami či inými aktívami. Táto funkcionalita by zároveň mala edukačný účel a podporovala by rozvoj investičného myslenia používateľov.

V produkčnom využití by aplikácia mohla podporovať aj automatický import portfólia z rôznych brokerských platforiem. Táto funkcionalita nebola v rámci tejto práce implementovaná pretože všetky preskúmané platformy umožňujú export dát o portfóliu len z reálnych účtov a dokumentácia k formátu týchto exportov nebola verejne dostupná. Získanie potrebných dát by si preto vyžadovalo uskutočnenie reálnych obchodov u daných brokerov. V budúcnosti by však táto integrácia veľmi prispela k zvýšeniu používateľského komfortu.

Literatúra

[1] *Aké sú 4 rôzne typy akcií?* online. XTB S.A. Dostupné z: https://www.xtb.com/sk/vzdelavanie/ake-su-4-rozne-typy-akcii. [cit. 2025-04-14].

[2] *All Tickers | Stocks API - Polygon* online. Dostupné z: https://polygon.io/docs/rest/stocks/tickers/all-tickers. [cit. 2025-04-20].

[3] *Apple Design Resources - Apple Developer* online. Dostupné z: https://developer.apple.com/design/resources/. [cit. 2025-04-21].

[4] *Custom Bars (OHLC) | Stocks API - Polygon* online. Dostupné z: https://polygon.io/docs/rest/stocks/aggregates/custom-bars. [cit. 2025-04-20].

[5] *Čo je investovanie?* online. Ministerstvo financií Slovenskej republiky. Dostupné z: https://www.fininfo.sk/fininfo/financne-produkty/investovanie/co-je-investovanie/co-je-investovanie.html. [cit. 2025-04-13].

[6] *Čo sú to ETF fondy?* online. Inštitút bankového vzdelávania NBS, n.o. Dostupné z: https://5penazi.sk/vzdelavaci-obsah/co-su-to-etf-fondy/. [cit. 2025-04-15].

[7] *JSON.stringify() - JavaScript | MDN* online. 03. apríla 2025. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/JSON/stringify#tojson_behavior. [cit. 2025-04-22].

[8] *Návod pre začiatočníkov, ako investovať do akcií* online. Inštitút bankového vzdelávania NBS, n.o. Dostupné z: https://5penazi.sk/vzdelavaci-obsah/navod-pre-zaciatocnikov-ako-investovat-do-akcii/. [cit. 2025-04-14].

[9] *Polygon.io - Stock Market API* online. Dostupné z: https://polygon.io. [cit. 2025-04-15].

[10] *Previous Day Bar (OHLC) | Stocks API - Polygon* online. Dostupné z: https://polygon.io/docs/rest/stocks/aggregates/previous-day-bar. [cit. 2025-04-20].

[11] *Server vs. serverless architektura* online. Think Easy s.r.o., 8. augusta 2023. Dostupné z: https://thinkeasy.cz/server-vs-serverless-architektura/. [cit. 2025-04-15].

[12] *Ticker Overview | Stocks API - Polygon* online. Dostupné z: https://polygon.io/docs/rest/stocks/tickers/ticker-overview. [cit. 2025-04-20].

Príloha A

Zoznam inšpirácií a prevzatých častí kódu

V tejto prílohe sú uvedené časti kódu, ktoré boli prevzaté z verejne dostupných zdrojov alebo vytvorené na základe inšpirácie z open-source projektov. Každá z týchto častí je v zdrojovom kóde projektu jednoznačne označená príslušným komentárom s odkazom.

- **Zachovanie gesta na návrat späť pri skrytom natívnom tlačidle na navigáciu v SwiftUI** – Pri použití modifikátora `.navigationBarBackButtonHidden()` v SwiftUI prestáva fungovať štandardné gesto návratu späť potiahnutím doprava. Na obnovenie tejto funkcionality bolo použité riešenie zo StackOverflow[1].
- **Pomocné funkcie pre prácu s URL reťazcami** – Pre jednoduchšiu manipuláciu s URL reťazcami boli v projekte vytvorené pomocné funkcie inšpirované JavaScript knižnicou `ufo`[2].
- **Efekt progresívneho rozmazania obsahu** – Moderný vizuálny štýl aplikácií od Apple často využíva efekt plynulého rozmazania obsahu, pre ktorý zatiaľ neexistuje jednoduché natívne API. Preto sa v projekte využíva verejne dostupné riešenie z diskusie na Reddite[3].

[1] Príspevok od Nazar Lisovyi na fóre Stack Overflow – https://stackoverflow.com/a/79383873/19906191

[2] unjs: ufo – https://github.com/unjs/ufo/blob/main/src/utils.ts

[3] Komentár používateľa iospeterdev na Reddite – https://www.reddit.com/r/SwiftUI/comments/1hyuhvk/comment/m6l8ylm/