Univerzita Hradec Králové
Fakulta informatiky a managementu
Katedra informatiky a kvantitativních metod

Webová aplikace pro real-time monitorování burzovních dat s notifikacemi

Diplomová práce

Autor: Marek Šípek
Studijní program: Aplikovaná informatika

Vedoucí práce: prof. RNDr. Petra Poulová, Ph.D.

Hradec Králové Duben 2026

Prohlášení:

Prohlašuji, že jsem diplomovou práci zpracoval samostatně a s použitím uvedené literatury. V případě využití nástrojů umělé inteligence jsem uvedl jejich účel a rozsah použití a ověřil správnost takto získaných informací.

V Hradci Králové dne 21.4.2026 Bc. Marek Šípek

Poděkování:

Děkuji prof. RNDr. Petře Poulové, Ph.D., za odborné vedení mé diplomové práce, cenné rady a trpělivost během celého procesu psaní mé práce.

Abstrakt

Tato diplomová práce se zabývá návrhem a implementací webové aplikace pro real-time monitorování burzovních dat s notifikacemi. Hlavním cílem práce bylo vytvořit systém, který umožní průběžně získávat a zpracovávat tržní data, ukládat je s využitím NoSQL databáze a vyhodnocovat uživatelsky definované podmínky pro generování upozornění. V teoretické části jsou zpracována východiska týkající se zpracování dat v reálném čase, specifik burzovních dat, databází NoSQL, architektury webových aplikací a systémů uživatelských notifikací. Praktická část se zaměřuje na návrh architektury aplikace, datového modelu a implementaci jednotlivých komponent systému. Součástí řešení je webové rozhraní pro správu sledovaných titulů a upozornění, serverová logika pro vyhodnocování podmínek a mechanismus pro doručování upozornění. Výsledkem práce je funkční prototyp aplikace, který ověřuje vhodnost zvoleného technologického řešení pro daný typ úlohy. V závěru práce jsou diskutovány přínosy navrženého řešení, jeho omezení a možnosti dalšího rozšíření.

Abstract

Title: Web application for real-time monitoring of stock exchange data with notifications

This diploma thesis deals with the design and implementation of a web application for real-time monitoring of stock market data with notifications. The main objective of the thesis was to create a system capable of continuously collecting and processing market data, storing them using a NoSQL database, and evaluating user-defined conditions for alert generation. The theoretical part covers the principles of real-time data processing, the specifics of stock market data, NoSQL databases, web application architecture, and user notification systems. The practical part focuses on the design of the application architecture, the data model, and the implementation of the individual system components. The solution includes a web interface for managing tracked instruments and alerts, backend logic for condition evaluation,

and a mechanism for delivering notifications. The result of the thesis is a functional prototype that verifies the suitability of the selected technological solution for this type of task. The conclusion discusses the benefits of the proposed solution, its limitations, and possibilities for further extension.

Klíčová slova:
burzovní data, real-time monitoring, webová aplikace, NoSQL databáze, notifikace

Key words:
stock market data, real-time monitoring, web application, NoSQL database, notifications

Obsah

1 Úvod ... 1

2 Cíl a metodika práce ... 3

3 Rešerše východisek ... 6

3.1 Zpracování dat v reálném čase ... 6

3.2 Charakteristiky a složitost údajů o finančních trzích ... 7

3.2.1 Specifika burzovních transakcí a tržních dat ... 8

3.2.2 Typy alertů používané při monitorování ... 10

3.3 Databáze NoSQL ... 12

3.3.1 Kategorie NoSQL databází ... 12

3.3.2 MongoDB ... 13

3.4 Streamovací rozhraní API a zdroje dat ... 14

3.5 Architektura webových aplikací ... 15

3.5.1 Třívrstvá architektura ... 15

3.5.2 Komunikace klient–server a zpracování událostí v reálném čase ... 17

3.5.3 Mechanismy přijímání dat a upozorňování v reálném čase ... 18

3.5.4 Správa dat, škálovatelnost a nasazení ... 19

3.6 Systémy upozornění uživatelů ... 20

3.7 Komerční aplikace pro sledování burzovních dat ... 22

3.7.1 Open-source aplikace a frameworky ... 23

3.7.2 Shrnutí analýzy existujících řešení ... 24

3.8 Bezpečnost webových aplikací pro finanční monitoring ... 24

4 Návrh aplikace ... 27

4.1 Specifikace požadavků na systém ... 27

4.1.1 Funkční požadavky ... 27

4.1.2 Nefunkční požadavky ... 28

4.1.3 Uživatelské scénáře 29
4.2 Návrh architektury 29
4.2.1 Prezentační vrstva 30
4.2.2 Aplikační vrstva 31
4.2.3 Datová vrstva 31
4.2.4 Komunikační vrstva 31
4.2.5 Notifikační vrstva 32
4.2.6 Bezpečnostní vrstva 32
4.3 Odůvodnění volby technologií 32
4.3.1 Prezentační a aplikační vrstva 33
4.3.2 Datová vrstva 33
4.3.3 Autentizace a správa uživatelů 33
4.3.4 Zpracování asynchronních úloh 34
4.3.5 Zdroj burzovních dat a real-time komunikace 34
4.4 Návrh datového modelu 34
4.5 Návrh zpracování dat v reálném čase 36
4.6 Návrh uživatelského rozhraní 38
5 Implementace aplikace 39
5.1 Implementace aplikační a prezentační vrstvy 39
5.2 Implementace datové vrstvy 41
5.3 Implementace autentizace a správy uživatelů 43
5.4 Implementace správy watchlistu a alertů 44
5.5 Implementace příjmu burzovních dat 45
5.6 Implementace komunikace v reálném čase 46
5.7 Implementace notifikační vrstvy 48
5.8 Implementace asynchronních úloh a umělé inteligence 51

5.9 Nasazení aplikace 52
6 Testování 54
6.1 Cíl testování 54
6.2 Manuální testování hlavních scénářů 54
6.3 Automatizované end-to-end testy pomocí Playwright 55
6.3.1 Test přihlášení a ochrany 55
6.3.2 Test správy watchlistu a alertů 56
6.4 Integrační testy alert logiky 56
6.5 Testování real-time komunikace a notifikací 56
6.6 Vyhodnocení latence 57
6.7 Shrnutí výsledků testování 58
7 Shrnutí a diskuse výsledků 59
7.1 Shrnutí hlavních výsledků 59
7.2 Zhodnocení splnění cílů práce 59
7.3 Přínosy navrženého řešení 60
7.4 Omezení navrženého řešení 60
7.5 Možnosti dalšího rozvoje 61
8 Závěry a doporučení 62
9 Seznam použité literatury 64
10 Přílohy 70

Seznam Obrázků

Obrázek 1: Třívrstvá architektura. Zdroj: vlastní.....17
Obrázek 2: Návrh architektury. Zdroj: vlastní.....30
Obrázek 3: Návrh datového modelu. Zdroj: vlastní.....36
Obrázek 4: Návrh toku dat a vyhodnocení alertu. Zdroj: vlastní.....37
Obrázek 5: Hlavní dashboard aplikace. Zdroj: vlastní.....40
Obrázek 6: Veřejná část aplikace. Zdroj: vlastní.....40
Obrázek 7: Uživatelský watchlist. Zdroj: vlastní.....44
Obrázek 8: Formulář pro vytvoření alertu. Zdroj: vlastní.....45
Obrázek 9: Detail vybraného akciového titulu. Zdroj: vlastní.....46
Obrázek 10: Indikace stavu websocket spojení. Zdroj: vlastní.....48
Obrázek 11: Alert notifikace přes email. Zdroj: vlastní.....52

Seznam Ukázek Kódu

Ukázka kódu 1: Schéma alertu. Zdroj: vlastní. 42
Ukázka kódu 2: Schéma alertEventu. Zdroj: vlastní. 42
Ukázka kódu 3: Middleware. Zdroj: vlastní. 43
Ukázka kódu 4: Inicializace websocket serveru. Zdroj: vlastní. 47
Ukázka kódu 5: Přiřazení uživatele do privátní místnosti. Zdroj: vlastní. 47
Ukázka kódu 6: Pravidla alertu. Zdroj: vlastní. 49
Ukázka kódu 7: Odeslání realtime notifikace po splnění alertu. Zdroj: vlastní. 50
Ukázka kódu 8: Ochrana proti duplicitnímu triggernutí. Zdroj: vlastní. 50

Seznam Tabulek

Tabulka 1: Manuální scénáře. Zdroj: vlastní. 55
Tabulka 2: Měření latence. Zdroj: vlastní. 57

1 Úvod

Exponenciální růst finančních trhů a rostoucí objem dat v reálném čase zásadně změnily způsob, jakým investoři a instituce rozhodují. V současném obchodním prostředí jsou přístup k aktuálním informacím a schopnost pohotově reagovat na cenové výkyvy a tržní události prvořadé pro udržení konkurenceschopnosti. Toto dynamické prostředí vyvolalo poptávku po inteligentních softwarových systémech schopných zpracovávat data v reálném čase, analyzovat je a reagovat na oznámení přizpůsobená konkrétním kritériím definovaným uživatelem.

Rychlý rozvoj webových technologií spolu s vývojem škálovatelných databázových systémů poskytuje nové příležitosti pro vytváření pohotových a efektivních finančních aplikací. Databáze NoSQL jsou navrženy pro zpracování velkých objemů nestrukturovaných a polostrukturovaných dat a nabízejí flexibilitu a škálovatelnost, která tradičním relačním databázovým systémům často chybí. Technologie, jako je MongoDB, se staly zvláště důležitými pro scénáře, kde je vyžadován vysoce výkonný příjem dat a rychlý přístup, zejména v aplikacích pracujících s toky tržních dat v reálném čase [1,2].

Dalším hnacím motorem vývoje těchto systémů je rostoucí zájem o automatizaci a algoritmickou podporu individuálních investorů. V současném finančním prostředí se drobní obchodníci postupně obracejí k sofistikovaným nástrojům, které pečlivě monitorují konkrétní cenné papíry a okamžitě je upozorňují, když jsou splněny předem stanovené technické nebo cenové podmínky. Tento přechod od modelu pasivního pozorování k proaktivním systémům upozorňování založeným na pravidlech je poháněn rostoucí dostupností datových kanálů v reálném čase, efektivních webových rámců a služeb push notifikací [3].

Nedávný pokrok v databázích NoSQL nabízí slibné řešení problému ukládání a dotazování dat v reálném čase. Na rozdíl od běžných relačních databázových systémů (RDBMS) jsou databáze NoSQL, jako je například MongoDB, navrženy tak, aby zvládaly velké objemy polostrukturovaných dat, umožňovaly horizontální škálovatelnost a nabízely přizpůsobitelné datové modely, které se mohou v čase vyvíjet. Díky těmto

vlastnostem jsou obzvláště vhodné pro ukládání finančních dat v časových řadách, která jsou často heterogenní a podléhají změnám schémat [2, 4].

Kromě toho se díky rostoucímu rozšíření webových technologií založených na Javascriptu současných frontendových frameworků, stal vývoj responzivních webových aplikací s možností komunikace v reálném čase životaschopným. V kombinaci s asynchronním zpracováním dat a programováním řízeným událostmi usnadňují tyto technologie vývoj vysoce interaktivních finančních aplikací zaměřených na uživatele.

2 Cíl a metodika práce

Hlavním cílem této diplomové práce je navrhnout, implementovat a vyhodnotit webovou aplikaci pro sledování burzovních dat v reálném čase s možností uživatelských notifikací na základě předem definovaných podmínek. Cílem není pouze vytvořit funkční softwarový nástroj, ale také propojit teoretické poznatky z oblasti databází, webových technologií a zpracování dat s jejich aplikací v konkrétním návrhu a realizaci. Klíčovou součástí řešení je využití NoSQL databázového systému MongoDB, který je vhodný pro flexibilní a škálovatelné ukládání dat s časovou složkou, jakými burzovní data bezesporu jsou.

Na základní cíl navazují dílčí cíle, jejichž splnění přispívá k naplnění záměru práce jako celku. Prvním z těchto dílčích cílů je analyzovat současné přístupy ke zpracování a notifikaci finančních dat v reálném čase a identifikovat možnosti jejich vylepšení v prostředí webových aplikací. Následně je třeba navrhnout systémovou architekturu, která bude modulární, škálovatelná a bude umožňovat oddělení jednotlivých komponent, jako je získávání dat, jejich zpracování, uložení a notifikace. Dalším cílem je vytvořit backendovou službu, která bude zpracovávat příchozí datové toky z vybraného externího burzovního API, vyhodnocovat podmínky definované uživateli a ukládat relevantní informace do NoSQL databáze. Součástí řešení je návrh vhodného datového modelu v systému MongoDB a vývoj frontendového rozhraní, které uživatelům umožní definovat sledované instrumenty a nastavovat podmínky pro upozornění. Finálním cílem je integrovat notifikační mechanismus využívající WebSocket technologii, a provést vyhodnocení výkonnosti systému s důrazem na latenci, správnost vyhodnocení podmínek a rozšiřitelnost.

Zvolený metodologický přístup odpovídá charakteru řešeného problému. Diplomová práce vychází z metodologie design science, která klade důraz na tvorbu a validaci funkčního artefaktu jako prostředku produkce nových poznatků. Tento přístup je vhodný zejména v případech, kdy je cílem vytvořit řešení praktického problému na základě teoretických východisek. Výhodou tohoto přístupu je možnost iterativního vývoje a testování, modularita vývoje jednotlivých komponent a přímé propojení návrhových rozhodnutí s existujícími teoretickými poznatky. Alternativní přístupy, například čistě empirický výzkum trhu či statistická analýza uživatelských platforem,

nebyly zvoleny, neboť by nevedly k vytvoření funkční aplikace a neposkytly by prostor pro hlubší aplikaci technologických znalostí.

Metodika práce dále zohledňuje možnosti a omezení zvoleného přístupu. Hlavní výhodou designově orientovaného přístupu je možnost ověřit funkčnost navrženého systému v praxi, a to formou prototypu, který lze testovat, upravovat a dále rozvíjet. Mezi omezení však patří například absence rozsáhlého kvantitativního uživatelského testování, neboť výkon a škálovatelnost systému není možné ověřit na rozsáhlém vzorku uživatelů v reálném provozu. Dalším limitem je závislost na externích API pro získávání dat, které mohou být omezeny licencí nebo počtem požadavků za časovou jednotku. Notifikační systém je v této fázi omezen na webové rozhraní, bez podpory SMS nebo mobilních notifikací.

Teoretická část práce je koncipována na základě systematické analýzy odborné literatury, technické dokumentace a dostupných vývojářských zdrojů. Jsou využívány jak akademické publikace zaměřené na databázové systémy, streamové zpracování dat a webové technologie, tak i odborné články, dokumentace softwarových nástrojů a open-source příklady architektur. Každý zdroj je posuzován z hlediska jeho relevance, aktuálnosti a přínosu k řešenému problému. Literární rešerše je zaměřena nejen na popis technologií, ale i na porovnání jejich vlastností a analýzu vhodnosti jejich použití v kontextu cílové aplikace.

Charakter výzkumu v této práci je převážně aplikovaný, s převažujícím kvalitativním přístupem. Jádrem práce je návrh a vývoj konkrétního softwarového řešení, které slouží jako výzkumný artefakt. Tímto způsobem jsou ověřovány hypotézy o vhodnosti dané architektury, výkonnosti použitých technologií a správnosti implementace notifikačních logik. Výsledky jsou interpretovány kvalitativně, na základě testovacích scénářů, výstupů aplikace a zpětné vazby z praktického použití.

Během zpracování práce je třeba počítat s možnými komplikacemi, například při integraci různorodých technologií, které mohou mít odlišné požadavky na komunikaci či datové formáty. Dále může být problematické nalezení vhodného API s dostatečně kvalitními a aktuálními daty dostupnými zdarma. Výzvou je také návrh takového systému, který bude zároveň výkonný, přehledný a snadno rozšiřitelný. Tyto a

případné další potíže, které se objeví během implementace, budou reflektovány v diskusní části práce.

V teoretické části práce je využit model ChatGPT-5.4 pro sumarizaci nebo překlad vědeckých článků a publikací z jejich PDF souborů a model Scholar GPT pro práci s akademickými zdroji. Jazykové modely jako ChatGPT nebyly využity jako primární zdroj informací, ale pouze jako pomocný nástroj pro návrh textů nebo sumarizaci obsahu.

3 Rešerše východisek

Tato kapitola se zabývá teoretickými a technologickými základy, na nichž je založen návrh a vývoj navrhované aplikace pro finanční monitorování v reálném čase. Text zkoumá principy zpracování dat v reálném čase, povahu burzovních dat, možnosti databází NoSQL, a to zejména MongoDB a architekturu reaktivních webových aplikací. Dále provádí kritické zhodnocení stávajících platforem a identifikuje nedostatky, které si tato práce klade za cíl odstranit.

3.1 Zpracování dat v reálném čase

Real-time systémy jsou navrhovány tak, aby zpracovávaly data a reagovaly na ně v rámci přesně stanovených časových omezení. V kontextu webových aplikací se termín „real-time“ obvykle používá k označení schopnosti detekovat, zpracovávat a vizualizovat data s minimální latencí, často kratší než jedna sekunda. Taková odezva systému je klíčová všude tam, kde je potřeba bezprostřední zpětná vazba nebo rychlá reakce systému na měnící se vstupy [6].

Real-time systémy lze obecně rozdělit do dvou hlavních kategorií podle nároků na včasnost vykonání. První skupinu tvoří systémy s tvrdými real-time požadavky, které vyžadují okamžité a přesné provedení operací. Jakékoli zpoždění může mít závažné důsledky, a proto nelze překročení časového limitu tolerovat. Příkladem takového systému může být systém aktivace airbagu v automobilu. Druhou skupinu představují soft real-time systémy, kde je sice důležité udržet latenci co nejnižší, avšak občasná zpoždění jsou tolerovatelná. Tyto systémy kladou důraz na plynulý chod a použitelnost, nikoli na absolutní dodržení časových limitů [7, 8].

Webové aplikace, které zpracovávají finanční data například obchodní platformy, burzovní přehledy či upozornění na změny cen spadají obvykle právě do kategorie soft real-time systémů. Nízká latence je v těchto případech nezbytná pro zajištění dobré uživatelské zkušenosti a efektivního rozhodování. Uživatelé musí být včas informováni o významných změnách, jako jsou cenové výkyvy nebo signály k obchodování. Přestože občasné krátké zpoždění nemusí nutně vést k selhání systému, může výrazně snížit jeho praktickou hodnotu [7].

Jedním z moderních přístupů, jak real-time požadavky v distribuovaných systémech řešit, je využití událostmi řízené architektury (Event-Driven Architecture, EDA). V takto navržených systémech spolu jednotlivé komponenty nekomunikují prostřednictvím synchronních volání, ale reagují na události, které jsou asynchronně vysílány a zpracovávány. Tento model přináší výhody v podobě volného provázání komponent, škálovatelnosti a odolnosti vůči chybám [9].

V architektuře EDA jsou události směrovány prostřednictvím front nebo streamovacích nástrojů, jako jsou například Apache Kafka nebo Redis Streams, které slouží jako zprostředkovatelé mezi producenty a konzumenty zpráv. Takový přístup umožňuje efektivní distribuci dat a jejich zpracování v reálném čase napříč různými částmi systému [10].

3.2 Charakteristiky a složitost údajů o finančních trzích

Finanční trhy generují značné objemy dat na globálních burzách. Tržní data lze rozdělit do několika typů. Mezi základní údaje o obchodech patří cena, objem a časové razítko transakce. Dále se srovnávají informace o nabídkových a poptávkových cenách a množství. Kromě těchto primárních dat jsou často využívány také odvozené ukazatele, jako jsou klouzavé průměry, míry volatility, index relativní síly (RSI) a ukazatel konvergence/divergence klouzavých průměrů (MACD) [11].

Bylo prokázáno, že vysokofrekvenční obchodní systémy zpracovávají miliony aktualizací za sekundu, zatímco maloobchodně orientované aplikace sice pracují s nižšími rychlostmi, ale stále vyžadují rychlé vyhodnocení uživatelských podmínek. Data ze současného akciového trhu ukazují, že časové uspořádání se vyznačuje přísnou posloupností událostí. Vysoká variabilita dat je pak důsledkem volatility, která ovlivňuje jak objem, tak frekvenci aktualizací [12].

Z hlediska formátu dat má polostrukturované uspořádání, zejména prostřednictvím rozhraní typu JSON API, praktický význam pro efektivní zpracování. Podle odborné literatury je patrný důraz na modely proudového dotazování, které umožňují průběžné výpočty nad nekonečnými datovými proudy. Pro konstrukci výstražných systémů se v této souvislosti osvědčily techniky, jako jsou klouzavá okna, orientační okna a vyhodnocování založené na spouštěčích [13, 14].

3.2.1 Specifika burzovních transakcí a tržních dat

Pro návrh webové aplikace zaměřené na monitorování burzovních dat je nezbytné porozumět samotné podstatě tržních transakcí. Aplikace sledující vývoj cen finančních instrumentů a generující upozornění při splnění definovaných podmínek musí vycházet nejen z technických možností zpracování dat v reálném čase, ale také z toho, jak jsou tržní data vytvářena a jaký mají význam. [11]

Burzovní transakce představuje realizovaný obchod s finančním instrumentem na organizovaném trhu. Finančním instrumentem může být například akcie, dluhopis, fondový podíl nebo jiný obchodovatelný cenný papír. Každá realizovaná transakce je výsledkem spárování nákupního a prodejního příkazu, přičemž obchodovaná cena odráží aktuální tržní konsenzus mezi kupujícími a prodávajícími. Cena na burze není statická, ale dynamická, takzvaně se mění na základě poptávky a nabídky. Jelikož hodnota instrumentu může kolísat v řádech milisekund. Pro spolehlivou logiku alertů je nutné zohlednit několik klíčových specifik těchto dat. [12]

Při práci s burzovní daty je důležité rozlišovat několik typů cen. Nejčastěji se pracuje s poslední realizovanou cenou *last price*. Vedle ní se používají také ceny *bid* a *ask*, tedy nejlepší aktuální nabídková a poptávková cena. Rozdíl mezi těmito hodnotami se označuje jako spread a představuje důležitý ukazatel likvidity trhu i okamžitých nákladů obchodování. Zatímco pro sofistikovanější obchodní systémy jsou použity tyto ceny, pro základní cenové alerty je last price dostatečný [15].

Vedle aktuální ceny mají význam také další cenové ukazatele, které se běžně používají pro interpretaci vývoje instrumentu v čase. Patří mezi ně zejména otevírací cena, nejvyšší a nejnižší cena dosažená v určitém období a zavírací cena. Tyto hodnoty bývají často uváděny ve formátu OHLC (*open, high, low, close*). Tyto údaje pomáhají uživateli zasadit aktuální výkyv do širšího kontextu obchodního dne a tvoří základ pro zobrazování historického vývoje kurzu i pro konstrukci grafů. Tyto údaje jsou užitečné zejména při zobrazování kontextu aktuální ceny, například při porovnání současné hodnoty s denním minimem nebo maximem. Umožňují uživateli lépe interpretovat význam konkrétního pohybu ceny [16].

Významným ukazatelem je také objem obchodů, obvykle označovaný jako *volume*. Tento údaj vyjadřuje množství zobchodovaných kusů za určité období. Samotný cenový pohyb bez znalosti objemu může být zavádějící. Pokud je však změna ceny doprovázena vysokým objemem obchodů, signalizuje to silnější tržní sentiment. V některých aplikacích proto může být objem využit i jako doplňková podmínka pro vyhodnocování upozornění. V rámci navrhovaného řešení může být objem v budoucnu využit jako rozšíření logiky alertů, například pro upozornění na neobvykle vysokou aktivitu spojenou s konkrétním titulem [15].

Další důležitou vlastností burzovních dat je jejich časová povaha. Finanční trhy generují data kontinuálně v průběhu obchodních hodin a hodnota sledovaných instrumentů se může měnit v řádu sekund, milisekund. Pro přesné fungování notifikační logiky jsou ideální data v reálném čase tzv. tická data. Bezplatná či veřejně dostupná API však často pracují s umělým zpožděním nebo s limity počtu dotazů. V této souvislosti je důležité rozlišovat mezi daty v reálném čase a zpožděnými daty. Reálná data umožňují aplikaci reagovat bezprostředně na pohyb trhu a jsou proto zásadní pro přesné fungování notifikační logiky. Naopak zpožděná data mohou být pro některé účely dostačující, například pro orientační zobrazení historického vývoje. Při návrhu aplikace je proto nutné pracovat s tím, že spolehlivost a užitečnost alertů přímo souvisí s kvalitou, aktuálností a frekvencí příchozích dat. Uživatel by měl být zároveň informován o tom, zda systém pracuje s daty v reálném čase, nebo zda může docházet k určitému zpoždění [17].

Specifickým rysem burzovních dat je také jejich volatilita. Volatilita vyjadřuje míru kolísání ceny v čase a je přirozenou součástí finančních trhů. U některých titulů mohou být změny ceny během krátkého intervalu minimální, zatímco u jiných může docházet k výrazným pohybům i v horizontu několika sekund. Tato vlastnost má přímý dopad na návrh logiky upozornění. Pokud je například alert nastaven přesně na konkrétní cenovou hranici, může při vysoké volatilitě docházet k častému opakovanému překračování této hodnoty. Systém proto musí být navržen tak, aby takové situace uměl vhodně ošetřit. Třeba deaktivace již splněného alertu nebo nastavením časového intervalu mezi upozorněními. Pokud by tyto mechanism aplikace neměla, mohla by generovat duplicitní nebo nadměrné množství notifikací. [11, 18]

Při interpretaci burzovních dat je třeba zohlednit i obchodní hodiny jednotlivých trhů. Akcie obchodované na amerických burzách mají jiné obchodní hodiny než evropské trhy a některé instrumenty mohou být obchodovány i mimo hlavní obchodní seanci. Z pohledu aplikace to znamená, že frekvence změn dat, aktivita trhu i relevance alertů se může výrazně lišit podle času a podle konkrétního trhu. Systém by proto měl být navržen s vědomím, že ne ve všech obdobích lze očekávat stejnou intenzitu aktualizací [15, 19].

3.2.2 Typy alertů používané při monitorování

Základním nástrojem, jak převést nepřetržitý tok tržních dat na užitečné informace, je automatizovaný systém upozornění, který odpovídá předem definovaným podmínkám. Upozornění neboli alerty tím zvyšují uživatelský komfort, podporují rychlejší reakci na vývoj trhu a zároveň představují hlavní funkční prvek aplikací zaměřených na real-time monitoring finanční. [18, 20]

Nejjednodušší a současně nejčastěji používanou formou upozornění je cenový alert. Tento typ alertu je založen na porovnání aktuální ceny s předem definovanou mezní hodnotou. Aktivuje se v momentě, kdy aktuální cena překročí definovanou horní hranici, nebo klesne pod hranici dolní. Výhodou cenového alertu je jeho jednoduchost, srozumitelnost a přímá vazba na rozhodovací proces uživatele. Zároveň jde o typ podmínky, který lze relativně snadno implementovat i vyhodnocovat v reálném čase. Právě proto bývá cenový alert základním stavebním prvkem monitorovacích systémů zaměřených na burzovní data [20].

Vedle absolutní cenové hranice lze využívat také alerty založené na relativní změně ceny. V tomto případě není podmínka definována jednou konkrétní hodnotou, ale procentní změnou vůči předchozí ceně, otevírací ceně daného dne nebo jiné referenční hodnotě. Uživatel tak může být upozorněn například na růst nebo pokles ceny o určité procento. Tento přístup lépe zachycuje dynamiku trhu a je univerzálnější, jelikož odstiňuje rozdíly mezi nominálními hodnotami různých akciových titulů. [21]

Další možnost představují alerty založené na objemu obchodů. Tyto alerty se aktivují při překročení určité úrovně obchodní aktivity a mohou signalizovat zvýšený zájem

trhu o daný instrument. Objem sám o sobě neříká, zda cena poroste nebo klesne, ale poskytuje důležitý kontext k interpretaci pohybu ceny. Slouží jako vhodný doplňkový ukazatel k cenovým pohybům a v základních aplikacích se často využívají jako nadstavbová funkce. Ve složitějších systémech se využívá pokročilejší analytické scénáře [22].

V pokročilejších systémech lze využívat také kombinované alerty, které spojují více podmínek současně. Upozornění se tak nevygeneruje pouze na základě jediné změny ceny, ale při současném překročení cenové hranice a zvýšeném objemu obchodů, případně při kombinaci několika ukazatelů v určitém časovém intervalu. Výhodou tohoto přístupu je vyšší přesnost a menší riziko vzniku falešně významných upozornění, ale zároveň to klade vysoké nároky na backendové zpracování, ukládání historického stavu v databázi a nároky na pochopitelnost rozhraní pro koncového uživatele [23].

Specifickou kategorií jsou alerty založené na časovém nebo opakovaném výskytu určité situace. Systém může například upozornit uživatele, pokud cena setrvává nad definovanou hranicí po určitou dobu, pokud se v daném intervalu opakovaně přiblíží ke stejné hodnotě nebo pokud během krátkého časového období dojde k sérii výrazných změn. Tyto mechanismy mohou být užitečné při snaze odlišit krátkodobé výkyvy od stabilnějších tržních pohybů [24].

Zásadním problémem při návrhu alertů je riziko vzniku „notification fatigue“ (únavy z notifikací). Pokud cena osciluje těsně kolem nastavené hranice, může systém vygenerovat desítky upozornění během pár minut. Z technického hlediska je proto nutné rozlišovat mezi *jednorázovým alertem* a *opakovatelným alertem*. U opakovatelných alertů musí systém implementovat ochranné mechanismy, například časovou prodlevu (cooldown/debouncing) nebo zaznamenání posledního interního stavu. Přesnost celého tohoto mechanismu je pak přímo závislá na latenci a kvalitě vstupních datových toků. Uživatel pak může začít systém ignorovat, přestože jeho primárním účelem bylo usnadnit sledování relevantních událostí. Při návrhu logiky alertů je proto nutné hledat rovnováhu mezi citlivostí systému a jeho praktickou použitelností. Alert by měl zachytit významnou situaci, ale současně by neměl uživatele zahlcovat. [18]

Dalším důležitým aspektem je vazba alertu na kvalitu a povahu vstupních dat. Přesnost vyhodnocení upozornění je přímo závislá na tom, jak aktuální a spolehlivá data systém přijímá. Pokud aplikace pracuje se zpožděnými daty nebo s omezenou frekvencí aktualizací, může dojít k tomu, že alert bude vyhodnocen později, než by uživatel očekával. To je zvláště důležité u aplikací, které mají působit jako real-time nástroj. Z tohoto důvodu je vhodné v aplikacích zdůraznit, že přesnost a užitečnost alertů není dána pouze návrhem samotné logiky, ale i vlastnostmi datového zdroje, způsobem přenosu dat a latencí celého systému [25].

3.3 Databáze NoSQL

V současném kontextu aplikace založené na datech často fungují v rámci provozních omezení, která přesahují možnosti běžných relačních databází (RDBMS). Nutnost spravovat značné množství nestrukturovaných nebo polostrukturovaných dat, udržovat vysokou propustnost zápisu, horizontálně se rozšiřovat a reagovat v reálném čase vedla k vývoji a přijetí databází NoSQL. Tato část se zabývá klasifikací databází NoSQL, jejich základními principy a specifickou vhodností databáze MongoDB pro aplikace zahrnující monitorování a upozorňování na burze v reálném čase [26].

Databáze NoSQL uvolňují omezení relační konzistence ve prospěch dostupnosti a tolerance rozdělení, jak je uvedeno v teorému CAP. Dokumentově orientované databáze, jako je MongoDB, umožňují ukládat flexibilní dokumentové formáty bez schémat typicky BSON nebo JSON, které jsou obzvláště vhodné pro reprezentaci různorodých a nepravidelných dat z burzy [26, 27].

3.3.1 Kategorie NoSQL databází

NoSQL je zastřešující termín pro různorodou sadu nerelačních databázových technologií, z nichž každá je optimalizována pro specifické datové modely a scénáře použití. Čtyři hlavní kategorie databází NoSQL jsou následující:

- Úložiště klíčových hodnot (např. Redis, Riak) poskytují extrémně rychlé ukládání a načítání hodnot indexovaných pomocí jedinečných klíčů, takže jsou vhodné pro ukládání do mezipaměti a správu relací [28].

- Sloupcová úložiště (např. Apache Cassandra) organizují data ve sloupcovém formátu a jsou optimalizována pro rozsáhlé a řídké datové soubory, které se často používají v analytických úlohách [28].

- Dokumentová úložiště (např. MongoDB a CouchDB jsou významnými příklady dokumentových databází, které spravují samopopisné, schematicky flexibilní dokumenty obvykle formátované ve formátu JSON nebo BSON, což je činí ideálními pro polostrukturovaná data. Grafové databáze, jejichž příkladem je Neo4j, reprezentují data jako uzly a hrany, čímž umožňují efektivní dotazování na složité vztahy, které jsou běžné v sociálních sítích a doporučovacích systémech [28].

V rámci těchto kategorií nabízejí dokumentově orientované databáze optimální rovnováhu mezi strukturou, flexibilitou a možností expresivního dotazování. To je činí obzvláště vhodnými pro správu dat časových řad doprovázených hierarchickými nebo vnořenými metadaty, což jsou převládající rysy záznamů o akciových trzích. Úložiště dokumentů, jako je MongoDB, usnadňují efektivní ukládání dat o finančních událostech flexibilním způsobem podle schématu a zároveň podporují indexování, agregaci a dotazování v reálném čase, což jsou schopnosti, které jsou nezbytné pro vývoj systémů pro sledování akcií s rychlou odezvou [27, 28].

3.3.2 MongoDB

MongoDB je široce rozšířená databáze NoSQL s otevřeným zdrojovým kódem, orientovaná na dokumenty, určená k ukládání dat ve formátu BSON (Binary JSON). Od svého prvního vydání v roce 2009 si získala značnou oblibu díky svému designu zaměřenému na vývojáře, rozsáhlému ekosystému a silné podpoře přijímání dat s vysokou propustností. Díky těmto vlastnostem je MongoDB obzvláště vhodná pro moderní aplikace založené na datech, včetně systémů finančního monitorování v reálném čase [29].

Vhodnost MongoDB pro takové případy použití je podpořena několika klíčovými architektonickými vlastnostmi. Flexibilita schématu umožňuje, aby se dokumenty v rámci jedné kolekce lišily svou strukturou. To umožňuje aplikacím dynamicky vyvíjet své datové modely bez nutnosti odstávek nebo složitých migrací. MongoDB nativně podporuje vnořené dokumenty a pole, které dobře ladí s hierarchickými datovými

strukturami, jako jsou finanční ukazatele vložené do obchodních záznamů. Zatřetí nabízí propracovaný a expresivní dotazovací jazyk, který zahrnuje podporu pokročilého filtrování, agregačních potrubí, geoprostorového indexování a fulltextového vyhledávání [28, 29].

Z hlediska škálovatelnosti a dostupnosti podporuje MongoDB horizontální škálování prostřednictvím shardingu, což je proces, který umožňuje rozdělit velké datové sady na více uzlů. Bylo prokázáno, že využití sad replik zvyšuje odolnost systému prostřednictvím automatického převzetí služeb při selhání a redundance dat. Za zmínku stojí, že od verze 5.0 obsahuje MongoDB nativní kolekce časových řad, které optimalizují ukládání a indexování časových dat, čímž výrazně snižují využití disku a zvyšují výkon čtení/zápisu u pracovních zátěží zahrnujících vysokofrekvenční aktualizace [29].

Tyto architektonické vlastnosti jsou v těsném souladu s požadavky finančního monitorování v reálném čase. V takových systémech jsou příchozí data velmi proměnlivá jak ve struktuře, tak ve frekvenci, a přesto je třeba se na ně dotazovat a zpracovávat je s nízkou latencí a vysokou spolehlivostí, to jsou cíle, které návrh MongoDB účinně podporuje [29].

Srovnávací studie opakovaně prokázaly, že MongoDB překonává relační databáze ve scénářích, které se vyznačují náročnými operacemi zápisu a flexibilitou schématu. To je zvláště relevantní v pipelinech pro příjem finančních dat, kde velké objemy dat a jejich variabilita vyžadují výkonný a přizpůsobivý backend úložiště. Koncept polyglotní persistence navíc podporuje použití více typů databází v rámci jedné systémové architektury. Například PostgreSQL lze použít k ukládání relačních uživatelských profilů, zatímco MongoDB se dobře hodí ke správě velkých objemů polostrukturovaných finančních dat s časovými řadami [30].

3.4 Streamovací rozhraní API a zdroje dat

Zavedení monitorovacího systému v reálném čase vyžaduje spolehlivý přístup k aktuálním údajům o finančním trhu. Několik veřejných poskytovatelů dat nabízí k tomuto účelu vhodné rozhraní API. Alpha Vantage poskytuje bezplatná aplikační programová rozhraní pokrývající široké spektrum nástrojů, včetně akcií, burzovně

obchodovaných fondů (ETF) a deviz (Forex). Finnhub umožňuje přístup k tržním datům v reálném čase i historickým datům a využívá protokol WebSocket pro nepřetržitý datový tok. Podobně služby jako Polygon.io a Twelve Data podporují přístup k datům s vysokým rozlišením, včetně minutové a tickové granularity [31, 32].

Tato rozhraní API obvykle podporují dotazy založené na symbolech a nabízejí řadu časových rozlišení, včetně intervalů tick-by-tick, 1 minuta, 5 minut a den. Data jsou běžně vracena ve formátu JSON (JavaScript Object Notation), který je vhodný pro integraci s moderními webovými aplikacemi [31, 32].

Při využívání těchto zdrojů dat však vyvstává několik problémů. Patří mezi ně limity rychlosti API, které omezují četnost požadavků, nekonzistence ve formátování dat u různých poskytovatelů a problémy související se synchronizací času. Efektivní zpracování streamovaných finančních dat často vyžaduje strategie vyrovnávací paměti a dávkování spolu s časovým sladěním na základě jednotných časových značek. V kontextu této práce je kladen důraz na využití veřejně přístupných rozhraní API k simulaci datových toků v reálném čase, čímž se snižuje závislost na proprietárních nebo komerčních poskytovatelích dat [21, 33].

3.5 Architektura webových aplikací

Architektura současných webových aplikací se značně vyvinula, aby splňovala požadavky na odezvu v reálném čase, distribuované datové toky a modulární vývoj. Za tímto vývojem stojí pokrok ve frontendových frameworcích, asynchronních komunikačních protokolech, backendových procesorech a databázových technologiích. V této části jsou nastíněny dominantní architektonické principy a vzory relevantní pro webové systémy reálného času, se zvláštním zaměřením na systémy používané pro monitorování, zpracování proudových dat a upozorňování uživatelů [34].

3.5.1 Třívrstvá architektura

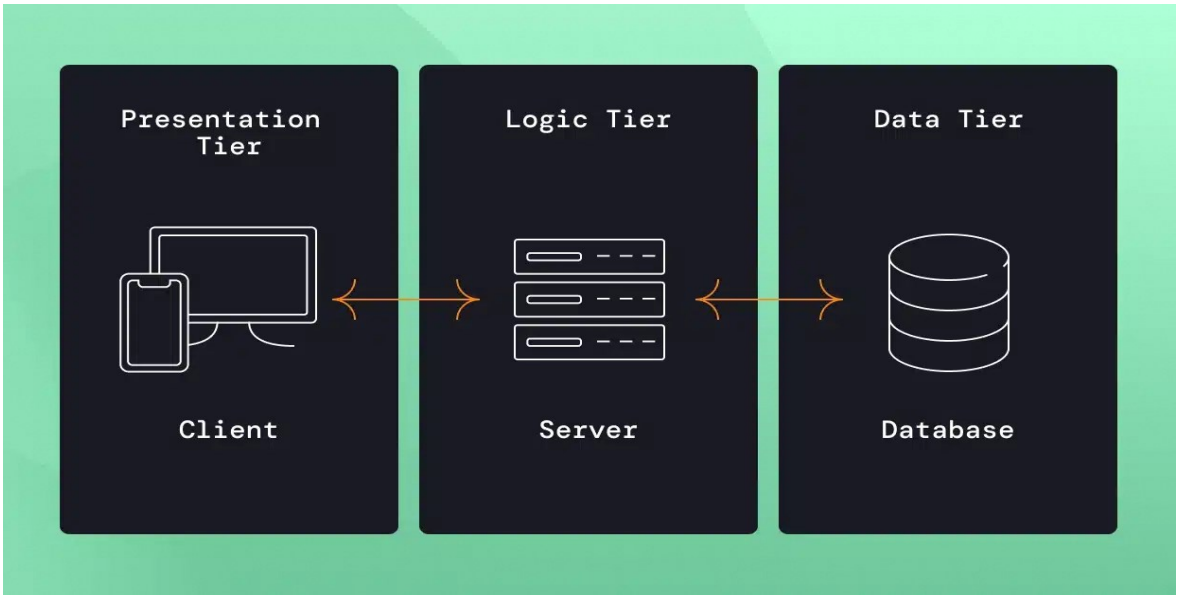
Většina webových systémů je strukturována podle třívrstvé architektury, která rozděluje systém do různých, ale vzájemně propojených vrstev, přičemž každá vrstva je zodpovědná za určitý soubor problémů. Ukázalo se, že tento modulární přístup

zvyšuje škálovatelnost, udržovatelnost a bezpečnost tím, že umožňuje nezávislý vývoj, nasazení a škálování každé vrstvy [34, 35].

Prezentační vrstva (Frontend) je zodpovědná za usnadnění interakce s uživateli prostřednictvím grafických uživatelských rozhraní (GUI). Implementace takových systémů obvykle probíhá prostřednictvím využití současných frameworků Javascriptu, mimo jiné React, Angular a Vue.js. Tato vrstva je zodpovědná za správu směrování, vykreslování dynamického obsahu a usnadnění komunikace s backendovými službami prostřednictvím rozhraní REST API nebo WebSockets [35].

Vrstva aplikační logiky (označovaná jako „backend“) zapouzdřuje základní obchodní logiku systému. Systém je zodpovědný za zpracování uživatelských požadavků, vynucování ověřování a autorizace, interakci s externími rozhraními API a provádění vyhodnocování podmínek nebo provádění pravidel. Mezi oblíbené backendové frameworky patří Spring Boot (Java), Node.js (JavaScript) a Django (Python), které podporují modulární vývoj služeb a integraci s databázemi a systémy pro zasílání zpráv [35, 36].

Datová vrstva je zodpovědná za správu uchovávání a vyhledávání strukturovaných i nestrukturovaných dat. Zahrnuje jak konvenční relační databáze, jako jsou PostgreSQL a MySQL, tak systémy NoSQL, včetně MongoDB a Cassandra, jejichž výběr závisí na povaze dat a způsobech přístupu k nim [35, 36].



Obrázek 1: Třívrstvá architektura. Zdroj: vlastní.

Jasným vymezením odpovědností mezi těmito třemi vrstvami podporuje architektura rozšiřitelnost systému a izolaci chyb a umožňuje týmům optimalizovat každou vrstvu podle specifických požadavků na výkon, škálovatelnost a dostupnost [36].

3.5.2 Komunikace klient–server a zpracování událostí v reálném čase

Moderní webové aplikace stále častěji využívají paradigma jednostránkové aplikace (SPA), kdy prohlížeč načte jediný dokument HTML a dynamicky aktualizuje obsah pomocí Javascriptu. Model SPA zahrnuje načtení jediného dokumentu HTML prohlížečem a následnou dynamickou aktualizaci obsahu pomocí Javascriptu. Ukázalo se, že tento model zvyšuje odezvu a uživatelský zážitek tím, že eliminuje potřebu načítání celé stránky. Komunikace mezi klientem a serverem obvykle probíhá podle dvou vzájemně se doplňujících vzorců. Rozhraní API RESTful poskytuje bezstavové koncové body pro obsluhu operací CRUD a načítání počátečních dat během načítání stránky. Pro usnadnění doručování průběžných nebo událostmi řízených aktualizací klientovi se používají komunikační kanály v reálném čase, jako jsou WebSockets, události zasílané serverem (SSE) nebo odběry GraphQL. Z těchto kanálů jsou v systémech reálného času obzvláště efektivní WebSockets, protože nabízejí plně duplexní komunikaci a umožňují serveru vysílat aktualizace okamžitě, aniž by vyžadovaly dotazování ze strany klienta [36, 37, 38].

Backend webového systému musí být schopen zpracovávat asynchronní události, periodické úlohy a vyhodnocování podmínek definovaných uživatelem. Tento proces

zahrnuje několik klíčových funkcí, včetně asynchronního zpracování událostí pro správu souběžných aktualizací a interakcí s uživateli bez blokování vláken, plánování úloh pro periodické vyhodnocování dat, logiku vyhodnocování pravidel pro vyhodnocování živých dat podle uživatelem definovaných prahových hodnot nebo podmínek a rozesílání oznámení pro upozornění uživatelů, když jsou splněny podmínky. Plnění těchto povinností se často dosahuje implementací architektur založených na mikroslužbách, návrhových vzorů řízených událostmi a front zpráv (např. RabbitMQ, Kafka), které usnadňují oddělení fází příjmu, zpracování a doručení dat. Moderní backendové frameworky, včetně Spring Boot, Node.js a Django, nabízejí podporu neblokujících I/O, vstřikování závislostí a bezproblémovou integraci s databázemi a rozhraními API třetích stran, což je činí vhodnými pro vývoj škálovatelných a citlivých systémů v reálném čase [39, 40, 41].

3.5.3 Mechanismy přijímání dat a upozorňování v reálném čase

Efektivní příjem dat je základním aspektem monitorovacích systémů v reálném čase, zejména v oblastech, jako je analýza akciového trhu nebo analýza internetu věcí, kde jsou externí data přijímána v nepřetržitých tocích. Mezi běžné strategie příjmu dat patří API založené na principu pull, kdy jsou data periodicky stahována z externích zdrojů pomocí koncových bodů REST API založené na principu push nebo WebSockets, kdy se systém přihlašuje k odběru živých datových toků a hybridní modely, které kombinují dotazování s ukládáním do mezipaměti a deduplikací, aby simulovaly streamování. Po načtení jsou data uložena v backendových databázích pro účely analýzy, vizualizace nebo dalšího vyhodnocení. V kontextu dat časových řad se ukázalo, že databáze NoSQL, jako je MongoDB, nabízejí významné výhody díky své flexibilitě schématu, podpoře indexovaných rozsahů dotazů a efektivním operacím založeným na oknech. Tyto vlastnosti jsou zvláště užitečné pro zachycení a dotazování časově uspořádaných údajů o cenách akcií [1, 42].

Webové systémy jsou stále více závislé na oznamovacích mechanismech, které informují uživatele o kritických událostech, výstrahách nebo anomáliích. Šíření oznámení může být umožněno různými kanály, které zahrnují rozhraní v prohlížeči (např. modální okna, toastové zprávy), rozhraní API webových oznámení (které usnadňuje doručování výstrah, i když je aplikace ve stavu nečinnosti), toky WebSocket

nebo Pub/Sub pro poskytování trvalých aktualizací a externí kanály, jako je e-mail, SMS nebo mobilní push oznámení. Logika spouštění těchto oznámení je obvykle založena na porovnávání příchozích dat v reálném čase s předem definovanými podmínkami uživatele. Ty mohou zahrnovat pravidla založená na prahových hodnotách, časových omezeních nebo rozpoznávání vzorů, například „pokud se cena zvýší o více než 5 % během 10 minut“ [43, 44].

3.5.4 Správa dat, škálovatelnost a nasazení

Architektura backendu současných aplikací prošla výrazným přechodem k přijetí polyglotní perzistence, což je princip, který zahrnuje využití různých typů databází v souladu s charakteristikami dat a vzorci přístupu k nim. Relační databáze, jako jsou PostgreSQL a MySQL, jsou ideální pro správu strukturovaných dat, včetně uživatelských účtů, oprávnění a ověřovacích záznamů. Naopak dokumentově orientované databáze, jako je MongoDB, jsou vhodnější pro zpracování vysokofrekvenčních, částečně strukturovaných dat, mimo jiné včetně cen akcií, protokolů a výsledků analýz. Aby databáze mohla fungovat jako monitorovací systém v reálném čase, musí být schopna podporovat rychlé načítání, efektivní dotazy na rozsah a rychlé vyhodnocování stavu nad aktuálními daty. Nativní kolekce časových řad databáze MongoDB, zavedené ve verzi 5.0, ve spojení s agregačními pipeline a horizontálním shardingem poskytují základ pro takové případy použití, který je efektivní a účinný [1, 29]

Při navrhování robustních webových aplikací je nejdůležitější škálovatelnost a zabezpečení. Horizontální škálovatelnost takových systémů se obvykle dosahuje pomocí nástrojů pro kontejnerizaci (např. Docker), orchestrační platformy (např. Kubernetes) a cloudových hostingových prostředí. Autentizace a autorizace jsou vynucovány pomocí mechanismů, jako jsou OAuth 2.0, JSON Web Tokens (JWT) a řízení přístupu na základě rolí (RBAC), které chrání uživatelská data a omezují přístup k citlivým operacím. Efektivitu a spolehlivost systému zvyšují strategie vyrovnávání zátěže a ukládání do mezipaměti s využitím nástrojů, jako je NGINX pro směrování provozu a Redis pro ukládání dat v paměti. Vyvinuté modely nasazení se pohybují mezi monolitickými architekturami, které jsou jednodušší na správu, ale méně flexibilní, a architekturami mikroslužeb, které umožňují nezávislé nasazení a škálování volně provázaných služeb, jako je příjem dat nebo rozesílání oznámení. Pro zajištění

optimálního zabezpečení se doporučuje implementovat ověřování vstupů, šifrovanou komunikaci prostřednictvím protokolu TLS, bezpečnou správu tokenů a rutinní posuzování zranitelností v souladu s doporučeními OWASP [41, 45].

3.6 Systémy upozornění uživatelů

Je zřejmé, že systémy oznamování uživatelům se staly klíčovou součástí současných webových aplikací, protože usnadňují šíření relevantních informací uživatelům způsobem, který je pohotový a vhodný k akci. V kontextu finančních aplikací v reálném čase jsou požadavky na tyto systémy obzvláště přísné. Výstrahy musí být doručovány s minimálním zpožděním, vykazovat vysokou spolehlivost a podporovat uživatelem definované podmínky pro spuštění. Účelem této části je prozkoumat účel, modely doručování, podporu platforem, mechanismy vyhodnocování pravidel a architektonické vzory, které jsou základem webových systémů upozornění v reálném čase [37, 29].

Úkolem oznámení je sloužit jako rozhraní mezi daty řízenými procesy na pozadí a interakcemi směrem k uživateli. Účinná oznámení v interaktivních systémech musí najít rovnováhu mezi včasností, relevancí a nerušivostí, zejména pokud se uživatelé současně věnují jiným úkolům. Tato rovnováha je zvláště důležitá v systémech finančního monitorování, kde musí být upozornění dostatečně včasná, aby podpořila reakce trhu, dostatečně přesná, aby se zabránilo falešně pozitivním nebo nadbytečným signálům, a dostatečně diskrétní, aby si zachovala důvěru uživatelů a celkovou použitelnost. V takových systémech fungují upozornění jako reakce v reálném čase na backendovou logiku, která neustále vyhodnocuje příchozí datové toky podle pravidel zadaných uživatelem [29].

Doručování oznámení lze rozdělit podle základního komunikačního modelu. Oznámení na bázi push, kdy server proaktivně předává klientovi upozornění při detekci události, jsou vhodná zejména pro aplikace v reálném čase. Toto paradigma podporují technologie jako WebSockets, Firebase Cloud Messaging a Web Notifications API. Naopak oznámení založená na tahu závisí na tom, že klient pravidelně dotazuje server, aby získal nová data. Zatímco implementace systémů založených na tahu je méně složitá, tyto systémy vykazují vyšší latenci a vyšší zatížení serveru. Hybridní přístupy kombinují obě techniky, často využívají techniku push pro výstrahy s vysokou

prioritou a techniku pull pro periodické nebo dávkové aktualizace. V oblasti finančních systémů jsou obvykle upřednostňovány architektury založené na technice push vzhledem k jejich efektivitě a bezprostřednosti. Web Notifications API dále zvyšuje použitelnost tím, že umožňuje zobrazování výstrah na úrovni prohlížeče, i když je aplikace neaktivní nebo minimalizovaná [28, 46].

Moderní prohlížeče a platformy nabízejí řadu mechanismů pro doručování oznámení. Patří mezi ně rozhraní API webových oznámení, které využívá pracovníky služeb k podpoře upozornění mimo aktivní kartu prohlížeče, oznámení v uživatelském rozhraní aplikace, jako jsou modální okna, toasty nebo bannery vložené přímo do frontendu, a mobilní push oznámení, která využívají služby specifické pro danou platformu, jako je Firebase nebo Apple Push Notification Service. Většina těchto rozhraní API vyžaduje výslovné povolení uživatele a nabízí volitelné funkce, jako jsou ikony, akce a zvukové signály pro zvýšení viditelnosti upozornění [47, 48].

Architektura systémů pro oznamování v reálném čase se často řídí modelem publish-subscribe (pub/sub). V tomto provedení jsou vydavatelé definováni jako backendové komponenty odpovědné za generování událostí oznámení, zatímco odběratelé jsou označeni jako klienti, obvykle ve formě relací prohlížeče, a jsou registrováni pro příjem aktualizací, které odpovídají jejich předem stanoveným podmínkám. Toto oddělení zvyšuje škálovatelnost, a umožňuje tak systému podporovat tisíce klientů současně, aniž by došlo k přetížení serveru. Technologie podporující tento model zahrnují WebSockets pro obousměrnou komunikaci, Server-Sent Events (SSE) pro jednosměrný streaming přes HTTP a message brokery, jako jsou Redis Pub/Sub, RabbitMQ nebo Kafka pro asynchronní výměnu událostí mezi službami. Backend je zodpovědný za udržování mapování mezi uživateli a jejich podmínkami, vyhodnocování každého příchozího datového bodu a vysílání oznámení, pokud jsou nalezeny shody. Uznává se, že bezstavové služby mají schopnost uchovávat tato mapování v paměti nebo je ukládat do databází klíč-hodnota za účelem usnadnění rychlého přístupu [31, 37].

Pro vytvoření účinného systému oznamování je nutné zvážit několik aspektů návrhu a osvědčených postupů. Základním úkolem je zvládnout kompromis mezi latencí a přesností systém by měl reagovat rychle, aniž by generoval falešné nebo předčasné výstrahy. V takových případech se jako účinná metoda minimalizace šumu v obdobích

volatility ukázalo použití technik, jako je debouncing nebo prahové vyhlazování. Další významnou otázkou je škálovatelnost, což je problém, který vyžaduje distribuované vyhodnocovací mechanismy a horizontálně škálovatelné dispečery oznámení. Zásadní význam má definice záruk doručení, zahrnující přesný počet výstrah, které mají být zaslány (např. přesně jednou, nejméně jednou nebo jako služba s nejlepší snahou). Důvodem je skutečnost, že tyto záruky ovlivňují logiku opakování a očekávání uživatele. Systémy by dále měly uživatelům usnadnit konfiguraci preferencí upozornění, zahrnující frekvenci doručování, výběr kanálu (např. v aplikaci, e-mailem, push) a možnost ztlumit nebo odhlásit se z odběru konkrétních upozornění. Uchovávání historie výstrah, často prostřednictvím protokolování v sekundární databázové kolekci, je rovněž cenné pro transparentnost uživatelů, ladění a auditní stopy [29, 49].

3.7 Komerční aplikace pro sledování burzovních dat

Mezi nejznámější komerční platformy pro sledování finančních trhů patří TradingView, MetaTrader 5 a Yahoo Finance. Tato řešení pokrývají různé skupiny uživatelů, od běžných investorů až po aktivní tradery, a liší se především rozsahem analytických nástrojů, možnostmi alertů a mírou otevřenosti. Pro účely této práce jsou důležité zejména proto, že ukazují, jaké funkce jsou v oblasti monitorování burzovních dat považovány za standard, a současně odhalují prostor pro návrh vlastního specializovaného řešení [50].

TradingView představuje rozsáhlou a velmi populární webovou platformu zaměřenou na tvorbu grafů, technickou analýzu a nastavování alertů. Uživatelé mohou vytvářet upozornění nad datovými řadami, indikátory i kreslícími objekty, a to s odezvou v reálném čase. Platforma podporuje i serverově běžící alerty, které nevyžadují aktivní přihlášení uživatele. Pro potřeby této práce je TradingView relevantní jako příklad robustního webového systému s velmi silnou podporou uživatelských upozornění a práce s tržními daty [50].

MetaTrader 5 je multi-asset platfroma, která je určena pro obchodování na trzích Forex, akcií a futures. Kromě monitorování trhu nabízí pokročilou technickou analýzu a podporu algoritmického obchodování. Zatímco MetaTrader představuje vysoce

komplexní obchodní nástroj, jeho orientace na aktivní trading jej činí zbytečně složitým pro uživatele, kteří vyžadují pouze lehký webový monitoring a základní notifikační logiku [51].

Yahoo Finance slouží jako příklad jednoduššího a uživatelsky velmi přístupného řešení. Nabízí přehled tržních dat, možnost tvorby watchlistů portfolií a nastavování základních cenových alertů prostřednictvím push notifikací. Ve srovnání s TradingView nebo MetaTrader 5 jde o méně komplexní, ale uživatelsky přístupné řešení, které dobře reprezentuje kategorii aplikací určených spíše pro orientační sledování trhu než pro detailní analytickou práci [52].

Komerční platformy tedy představují dva extrémy, buď nabízejí vysoce komplexní analytické a obchodní funkce, nebo poskytují jen základní přehled. Navrhovaná aplikace se inspiruje kombinací webové dostupnosti (Yahoo Finance) a robustní logiky alertů (TradingView), přičemž se zaměřuje čistě na uživatelsky přívětivé sledování titulů bez ambice zasahovat do reálného tradingu.

3.7.1 Open-source aplikace a frameworky

Vedle komerčních systémů existují také otevřená řešení, která umožňují větší míru přizpůsobení a jsou vhodná zejména pro vývojáře, výzkumníky a uživatele se zájmem o algoritmické obchodování. Do této skupiny lze zařadit QuantConnect LEAN, Backtrader a Freqtrade. Tato řešení nejsou přímými alternativami ke komerčním platformám pro běžného uživatele, ale poskytují cenný pohled na to, jak lze navrhovat modulární systémy pro práci s tržními daty, strategiemi a automatizovaným vyhodnocováním podmínek [53].

QuantConnect LEAN je open-source algoritmický engine určený pro výzkum, backtesting a live trading. Je to otevřený engine podporující Python i C#, který lze provozovat lokálně i v cloudu a který je navržen jako modulární a rozšiřitelný. Z hlediska této práce je LEAN relevantní především tím, že ukazuje architekturu systému schopného pracovat s více datovými zdroji, více třídami aktiv a různými strategiemi. Současně je však orientován spíše na algoritmické obchodování a vývoj strategií než na jednoduchou webovou aplikaci pro monitorování cenových alertů koncovým uživatelem [53].

Backtrader je populární Python framework primárně určený pro vývoj, testování a backtesting obchodních strategií. Umožňuje vytvářet opakovaně použitelné strategie, indikátory a analyzátory bez nutnosti budovat vlastní infrastrukturu od základu. Pro účely této práce sice demonstruje efektivní práci s obchodní logikou, ale zcela postrádá vrstvu moderní webové aplikace (správa uživatelů, UI, real-time notifikace) [54].

Freqtrade je otevřený trading bot, který se od předchozích liší tím, že podporuje vzdálené řízení přes aplikaci Telegram nebo vlastní webové rozhraní. Obsahuje backtesting, vizualizaci, money management a optimalizaci strategií. Přestože je Freqtrade zaměřen především na kryptoměnové trhy, ukazuje zajímavý přístup k propojení otevřeného kódu, automatizovaného vyhodnocování strategií a webově přístupného ovládání [55].

Open-source řešení tedy přinášejí vysokou flexibilitu a možnost přizpůsobení, avšak často cílí spíše na technicky pokročilé uživatele nebo na oblast algoritmického tradingu. Na rozdíl od komerčních aplikací nejsou obvykle optimalizována jako jednoduché koncové produkty pro běžného investora, ale spíše jako frameworky nebo platformy pro další vývoj [53].

3.7.2 Shrnutí analýzy existujících řešení

Z porovnání komerčních a open-source řešení vyplývá, že současný trh je polarizovaný. Existující komerční aplikace nabízejí velmi široké funkce za cenu vysoké komplexity (a často i uzavřenosti ekosystému), zatímco open-source alternativy poskytují otevřené vývojářské frameworky, které však nejsou navrženy jako jednoduché webové nástroje pro běžné uživatele. Tento stav tak vytváří ideální prostor pro návrh a vývoj vlastní webové aplikace, která vyplní mezeru na trhu a propojí přehledné uživatelské rozhraní, real-time zpracování burzovních dat a flexibilní, spolehlivý alertovací mechanismus bez zbytečné složitosti spojené s reálným tradingem.

3.8 Bezpečnost webových aplikací pro finanční monitoring

Bezpečnost u webových aplikací představuje jednu z nejdůležitějších vlastností systém, převážně v aplikacích určené pro finanční trh a jeho sledování. Pro tyto systémy je tato vlastnost ještě větší, protože pracují s uživatelskými účty,

individuálním nastavením sledovaných instrumentů a pravidly pro generování upozornění. I když tyto aplikace samy nerealizuji obchodní operace, jejich výstupy mohou ovlivňovat rozhodování uživatele. Bezpečnost proto nelze brát pouze jako ochranu před neoprávněným přístupem, ale také jako důvěryhodnost v celý systém. Zabezpečení webových aplikaci obecně vychází z OWASP Top 10, detailnější jsou pak definovaným standardem OWASP ASVS [56].

Základní vrstvu zabezpečení tvoří autentizace a autorizace uživatelů, autentizace slouží k ověření uživatele, kde autorizace určuje, k jakým datům a operacím má daný uživatel přístup. Pro webové aplikace je klíčové nejen bezpečné přihlášeni, ale také správná správa relaci a zajištění nepřerušené autentizované relace uživatele po dobu práce se systémem. NIST SP 800-63B uvádí normativní požadavky pro práci s autentizatory a session management. NIST tedy tvoří vhodný základ pro návrh bezpečného přístupu pro uživatele do aplikace. Ve finanční monitoringu je navíc důležité zajistit, aby uživatel přistupoval pouze ke svým vlastním datům [57].

Dalším důležitým aspektem je ochrana komunikace a bezpečné zacházeni s citlivými údaji uživatele. Přenos dat mezi klientem, serverem a externími služby by měl probíhat prostřednictvím zabezpečených kanálu. NIST explicitně uvádí protokol TLS jako protokol poskytující důvěrnost a integritu komunikace. Je podstatný jak pro přihlašovaní uživatele, tak i při komunikaci s externími API. V produkci je rovněž důležité zabránit, aby přístupové tokeny nebo API klíče byly dostupné na klientské straně aplikace a v přímo ve zdrojovém kódu. Vhodným řešením je jejich správa na serverové straně aplikace, pomoci jejich proměnných prostředí na oddělení veřejné a neveřejné konfigurace. Tato opatření se vztahují k širší ochraně přístupových konfiguraci a bezpečné konfigurace aplikace [57].

Významnou kapitolu tvoří validace vstupů a prevence běžných zranitelností webových aplikací. Ať už jde o přihlašovací údaje, vyhledávací dotazy nebo parametry alertů, data se musí ověřovat primárně na serveru. Klientská validace zlepšuje uživatelské rozhraní a zážitek uživatele, ale z hlediska bezpečnosti ji lze snadno obejít. Z tohoto důvodu je důležité ověřovat datové proudy na serveru. Tento přístup reflektuje doporučení metodiky OWASP Top 10 a detailnější požadavky standardu OWASP ASVS [56].

Specifickou oblasti pro bezpečnost finančních aplikací je komunikace v reálném čase. Tato oblast vyžaduje specificky přístup i protokol WebSocket (RFC 6455). Tento protokol je navržen pro obousměrnou komunikaci mezi klientem a serverem a využívá úvodní handshake následovaný přenosem zpráv nad jedním perzistentním spojem. Krom jeho výhod v podobě nízké latence a okamžitého doručování aktualizací však přináší i specifické bezpečnostní požadavky. Jak upozorňuje dokumentace MDN, komunikace by měla probíhat výhradně přes zabezpečený protokol (WSS) a nesmí docházet k míchání zabezpečeného a nezabezpečeného obsahu. Server navíc musí neustále ověřovat, zda aktivní relace odpovídá datům, která se klientovi odesílají. V aplikacích pro finanční monitoring je proto důležité nejen správně navrhnout samotný přenos dat, ale také zajistit, aby uživatel přijímal pouze informace odpovídající jeho oprávněním a aktivní relaci [37].

Spolehlivost systému nakonec podtrhuje auditovatelnost. Podle OWASP ASVS nemá bezpečnost sloužit jen jako prevence zranitelností, ale má budovat důvěru v systém. U aplikace spravující finanční upozornění to znamená pečlivé logování klíčových událostí od vytvoření alertu přes jeho vyhodnocení až po selhání komunikace s API. Tyto záznamy jsou klíčové pro diagnostiku problémů a usnadňují následné testování [58].

4 Návrh aplikace

Tato kapitola se zaměřuje na návrh aplikace pro real-time monitorování burzovních dat s uživatelskými notifikacemi. Popisuje specifikaci požadavků, návrh architektury systému, datového modelu a hlavních aplikačních mechanismů, které tvoří základ výsledného řešení. Cílem této části je systematicky vymezit, jak má být aplikace navržena z pohledu softwarového inženýrství, tak aby splňovala požadavky na modularitu, rozšiřitelnost, nízkou latenci a spolehlivé vyhodnocování personalizovaných upozornění. Navržené řešení současně vychází z teoretických východisek uvedených v předchozí kapitole a vytváří podklad pro následnou implementaci.

4.1 Specifikace požadavků na systém

Před samotným zahájením vývoje bylo nezbytné stanovit sadu základních požadavků, které musí první verze systému splňovat. Cílem navrhovaného řešení je vytvořit minimální životaschopný produkt (MVP), který umožní ověřit primární funkčnost a technickou proveditelnost (zejména práci s real-time daty a notifikacemi). Rozšiřování systému o další komplexní funkce je plánováno iterativně na základě analytických dat a zpětné vazby z reálného provozu.

4.1.1 Funkční požadavky

Funkční požadavky definují, co přesně musí systém umět a jaké operace musí uživateli umožnit:

- Správa uživatelského účtu: Uživatel se musí moci do systému bezpečně přihlásit a pracovat v izolovaném prostředí výhradně se svými vlastními daty a nastaveními.
- Vyhledávání instrumentů: Systém musí umožnit vyhledání konkrétního finančního titulu (např. akcie) a načtení jeho základních informací, včetně aktuální tržní ceny.
- Správa watchlistu: Uživatel musí mít možnost přidávat vyhledané instrumenty do svého osobního seznamu sledovaných titulů (watchlistu) a případně je z něj odebírat.

- Správa alertů (upozornění): Jádro celého systému. Uživatel musí mít možnost nad vybraným instrumentem vytvořit podmínku (např. překročení horní nebo dolní cenové hranice). Systém by měl být připraven na budoucí rozšíření o další typy podmínek.
- Vyhodnocování dat: Systém musí na pozadí kontinuálně porovnávat příchozí tržní data z externího API s aktivními alerty všech uživatelů a při shodě vygenerovat notifikační událost.
- Zobrazení notifikací: V případě splnění podmínky alertu musí být uživatel informován prostřednictvím notifikace v uživatelském rozhraní webové aplikace.
- Historie a auditovatelnost: Systém musí ukládat záznamy o vyhodnocených alertech a zobrazených notifikacích, aby měl uživatel zpětný přehled o spuštěných událostech a aby bylo možné zpětně diagnostikovat chování systému.

4.1.2 Nefunkční požadavky

Nefunkční požadavky definují, jaké kvalitativní vlastnosti musí systém splňovat během svého běhu:

- Nízká latence: Odezva mezi přijetím nových dat z API, vyhodnocením podmínek v databázi a doručením notifikace na klienta musí být co nejkratší, aby měla informace pro uživatele praktickou hodnotu.
- Responsivní design – uživatelské rozhraní je optimalizováno I pro mobilní zařízení a různé velikosti displejů.
- Použitelnost a UI/UX: Uživatelské rozhraní musí být intuitivní a přehledné. Vytváření alertů a správa watchlistu nesmí uživatele zahltit zbytečnou složitostí.
- Modularita a rozšiřitelnost: Architektura musí být rozdělena do logických celků (příjem dat, vyhodnocovací služba, API rozhraní, frontend), aby bylo možné v budoucnu snadno přidat nové datové zdroje nebo notifikační kanály.
- Bezpečnost: Aplikace musí chránit citlivá data (hesla, API klíče) a zajišťovat validaci všech vstupů na straně serveru podle doporučení OWASP.

- Odolnost vůči provozním limitům: Aplikace musí umět adekvátně reagovat na výpadky externího API, zpoždění dat nebo na limity počtu dotazů (rate limiting), a to bez pádů celého systému.

4.1.3 Uživatelské scénáře

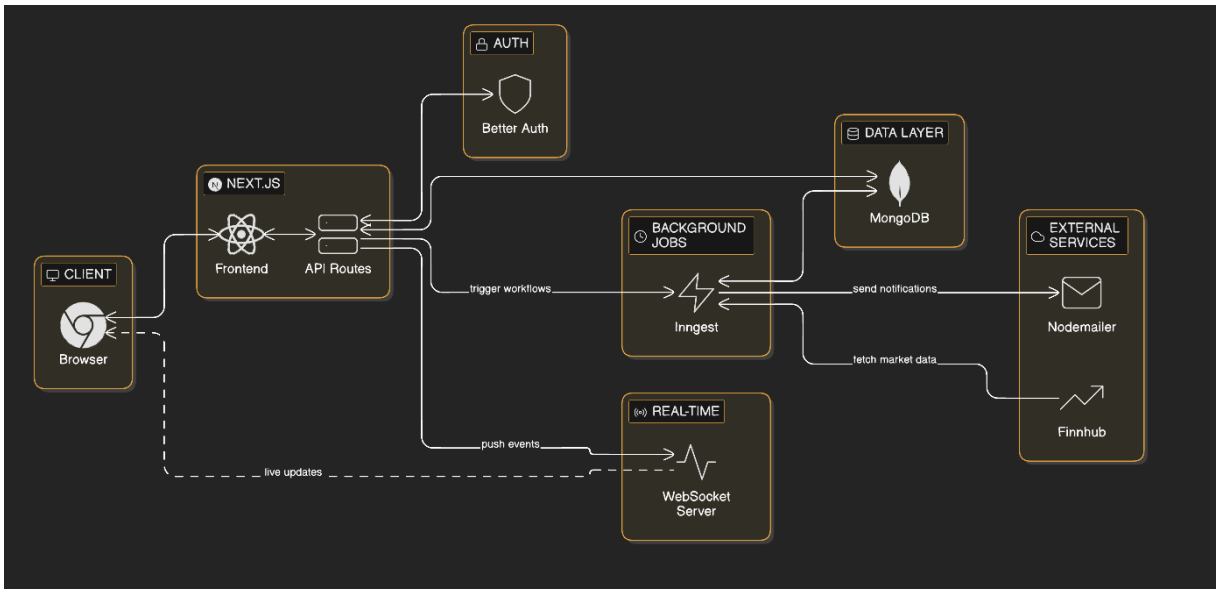
Pro lepší konkretizaci požadavků a následný návrh logiky aplikace jsou definovány tyto základní scénáře použití:

- UC-01: Správa watchlistu. Uživatel se přihlásí, vyhledá požadovaný akciový titul (např. AAPL) a přidá si jej do svého sledovaného seznamu pro rychlý přehled.
- UC-02: Vytvoření alertu. Uživatel vybere instrument ze svého watchlistu a nastaví podmínku upozornění (např. "Upozorni mě, pokud cena klesne pod 150 USD"). Systém tento alert uloží jako aktivní.
- UC-03: Automatické vyhodnocení. Aplikace na pozadí přijme nová tržní data. Vyhodnocovací služba zjistí, že cena AAPL klesla na 149 USD, porovná to s alertem z UC-02 a vytvoří událost k odeslání.
- UC-04: Notifikace uživatele. Uživateli, který má otevřenou webovou aplikaci, vyskočí upozornění o dosažení ceny. Událost se následně uloží do databáze historie pro pozdější kontrolu.

4.2 Návrh architektury

Na základě definovaných požadavků byla architektura aplikace navržena jako vícevrstvé webové řešení. Hlavním cílem tohoto návrhu je zajistit přehledné oddělení odpovědností (Separation of Concerns), snadnou rozšiřitelnost a efektivní vývoj prototypu. Vzhledem k tomu, že by aplikace měla sloužit ke sledování burzovních dat, správě watchlistů, vyhodnocování alertů a doručování notifikací, je architektura logicky rozdělena do několika klíčových vrstev.

Systém je koncipován jako full-stack webová aplikace postavená na frameworku Next.js. Datová vrstva je v návrhu svěřena NoSQL databázi MongoDB Atlas, pro autentizaci se počítá s nasazením knihovny Better Auth a asynchronní procesy má podle návrhu orchestrovat platforma Inngest.



Obrázek 2: Návrh architektury. Zdroj: vlastní.

4.2.1 Prezentační vrstva

Prezentační vrstva představuje část systému, která zajišťuje kontakt uživatele s aplikací. Jejím hlavním úkolem je zobrazování dat, přijímání uživatelských vstupů a zprostředkování přehledné interakce se sledovanými funkcemi systému. Tato vrstva neobsahuje žádnou složitou obchodní logiku, čímž je dosaženo čistého oddělení odpovědností (Separation of Concerns).

Při návrhu prezentační vrstvy byl kladen důraz především na přehlednost, jednoduchost ovládání a rychlou orientaci v datech. Vzhledem k tomu, že aplikace pracuje s průběžně se měnícími finančními údaji, musí uživatelské rozhraní umožnit rychlé rozlišení důležitých informací od doplňkového obsahu. Současně musí být schopno reagovat na průběžné aktualizace bez toho, aby docházelo ke ztrátě srozumitelnosti nebo k zahlcení uživatele. Vrstva je proto koncipována tak, aby plynule reagovala na asynchronní datové aktualizace, aniž by překreslováním celé obrazovky narušila uživatelskou zkušenost nebo způsobila ztrátu kontextu.

4.2.2 Aplikační vrstva

Aplikační vrstva tvoří hlavní logické jádro systému. Jejím úkolem je zpracovávat požadavky přicházející z uživatelského rozhraní a koordinovat komunikaci mezi ostatními vrstvami. Taktéž provádět klíčové operace související s obchodní logikou aplikace. Aplikační vrstva jednoduše přijímá abstraktní požadavky od uživatele, překládá je do konkrétních výpočetních úkonů a orchestruje datovou, komunikační i notifikační vrstvu k jejich vykonání. Důležitou vlastností aplikační vrstvy je, že odděluje vlastní rozhodovací logiku systému od způsobu prezentace dat i od jejich fyzického uložení. Tím je dosaženo vyšší modularity a lepší rozšiřitelnosti systému.

4.2.3 Datová vrstva

Datová vrstva zajišťuje ukládání, načítání a správu všech informací, se kterými navrhovaná aplikace pracuje. Patří sem zejména údaje o uživatelích, seznamy sledovaných instrumentů, definované alerty a záznamy o notifikačních událostech. Vzhledem k charakteru řešeného problému je tato vrstva navržena s důrazem na flexibilitu datového modelu, možnost snadného rozšiřování a efektivní práci s polostrukturovanými daty.

Volba databáze typu NoSQL odpovídá jak povaze ukládaných dat, tak i formálním požadavkům zadání práce. Dokumentově orientovaný přístup umožňuje jednoduše pracovat s entitami, jejichž struktura se může v čase vyvíjet, aniž by bylo nutné provádět složité migrační zásahy do schématu. Tato vlastnost je důležitá zejména u alertů a souvisejících událostí, kde lze do budoucna očekávat rozšiřování logiky a doplňování dalších atributů.

4.2.4 Komunikační vrstva

Komunikační vrstva zajišťuje přenos dat mezi jednotlivými částmi systému i mezi systémem a externím prostředím. V navrhované aplikaci plní dvojí roli. Na jedné straně slouží ke komunikaci mezi uživatelským rozhraním a aplikační vrstvou při běžných operacích. Na druhé straně zajišťuje přenos průběžně aktualizovaných dat a notifikačních událostí v reálném čase.

Z návrhového hlediska je vhodné rozlišovat mezi standardní synchronní komunikací a komunikací průběžnou. Běžné operace, které nevyžadují okamžitý přenos změn,

mohou být řešeny klasickým REST API modelem. Naproti tomu aktualizace cenových dat a upozornění na splněné podmínky je účelné zajišťovat pomocí perzistentního komunikačního kanálu, který umožní bezprostřední přenos změn směrem ke klientovi.

4.2.5 Notifikační vrstva

Notifikační vrstva je odpovědná za převod vyhodnocené systémové události na formu upozornění, kterou lze předat uživateli. Jejím účelem je zajistit, aby uživatel obdržel relevantní informaci v okamžiku, kdy dojde ke splnění některé z jím definovaných podmínek. V navrhovaném řešení tato vrstva navazuje na aplikační logiku a současně využívá komunikační mechanismy systému.

Z návrhového hlediska je důležité, aby notifikační vrstva nepůsobila izolovaně, ale byla úzce propojena s logikou alertů, historií událostí a uživatelskými preferencemi. Systém musí být schopen rozlišit, kdy má být upozornění vytvořeno, jakou formou má být uživateli předáno a jak zabránit nadměrnému nebo duplicitnímu doručování při opakovaném splnění stejné podmínky.

4.2.6 Bezpečnostní vrstva

Bezpečnostní vrstva představuje soubor principů a mechanismů, jejichž cílem je chránit uživatelská data, řídit přístup k jednotlivým částem systému a zajistit důvěryhodnost aplikace. V navrhovaném řešení je bezpečnost chápána jako průřezová vlastnost, která se dotýká všech ostatních vrstev systému, nikoli pouze jedné samostatné technické části.

V první řadě jde o bezpečné ověření identity uživatele a zajištění toho, aby měl každý uživatel přístup pouze ke svým vlastním datům, alertům a notifikační historii. Vedle autentizace je tedy důležitá i autorizace, která určuje rozsah povolených operací. Dalším důležitým aspektem je ochrana komunikace mezi klientem, aplikační vrstvou a externími službami, stejně jako bezpečné nakládání s konfiguračními údaji a přístupovými klíči.

4.3 Odůvodnění volby technologií

Při návrhu aplikace bylo nutné zvolit takový technologický stack, který bude odpovídat charakteru řešeného problému. Důležitými kritérii při výběru byly zejména rychlost vývoje, možnost zpracování dat v reálném čase, rozšiřitelnost systému, vhodnost pro

práci s databází NoSQL a snadné nasazení do cloudového prostředí. Zvolený stack byl navržen s cílem vytvořit funkční a přehledně strukturovaný prototyp, který bude možné dále rozšiřovat bez nutnosti zásadních změn architektury.

4.3.1 Prezentační a aplikační vrstva

Pro návrh uživatelského rozhraní a současně i hlavní aplikační vrstvy byl zvolen framework Next.js postavený nad knihovnou React. Hlavním důvodem této volby je možnost sjednocení klientské a serverové části do jednoho vývojového prostředí. Tento přístup je vhodný zejména pro prototypové a menší až středně rozsáhlé aplikace. Výhodou zvoleného řešení je také široká podpora moderního webového vývoje. Next.js zároveň umožňuje kombinovat klasické API rozhraní s interaktivním frontendem v rámci jedné aplikace [59, 60].

4.3.2 Datová vrstva

Pro datovou vrstvu byla zvolena databáze MongoDB Atlas doplněná o knihovnu Mongoose. Toto rozhodnutí vychází jak z charakteru ukládaných dat, tak z formálního požadavku práce na využití databáze NoSQL. MongoDB je vhodná zejména pro ukládání polostrukturovaných a proměnlivých dat, což odpovídá povaze uživatelských alertů, notifikačních událostí a dalších záznamů spojených s monitorováním finančních dat. Výhodou MongoDB je také snadná rozšiřitelnost datového modelu. V průběhu vývoje aplikace lze relativně jednoduše upravovat strukturu ukládaných dokumentů bez nutnosti složitých migrací schématu. Knihovna Mongoose pak přináší vyšší úroveň kontroly nad strukturou dat, validací a vazbou mezi aplikační logikou a databázovou vrstvou.

4.3.3 Autentizace a správa uživatelů

Pro správu identit a autorizaci uživatelů bylo do architektury začleněno řešení Better Auth. Hlavním motivem byla bezpodmínečná nutnost izolovat uživatelská data a bezpečně spravovat přístup k alertům a zajistit, aby každý klient interagoval výhradně se svými osobními záznamy. V aplikaci finančního zaměření je totiž stěžejní nejen ochrana přihlašovacích údajů, ale především konzistentní správa relačních tokenů (session management) a oprávnění [61].

4.3.4 Zpracování asynchronních úloh

Pro zpracování asynchronních a plánovaných úloh byl do architektury zařazen nástroj Inngest. Důvodem této volby je potřeba oddělit úlohy, které není vhodné provádět přímo v rámci běžného request-response cyklu webové aplikace. Typicky jde o činnosti, jako je odesílání e-mailových notifikací, pravidelné generování souhrnů nebo jiné pomocné procesy, které mají probíhat samostatně a nemají zpomalovat hlavní interakci uživatele s aplikací [62].

Zařazení této technologie odpovídá návrhovému principu oddělení odpovědností mezi hlavní aplikační logiku a podpůrné procesy. Z architektonického hlediska tak dochází ke zvýšení přehlednosti systému a k lepší připravenosti na budoucí rozšíření.

4.3.5 Zdroj burzovních dat a real-time komunikace

Důležitou součástí návrhu je volba externího zdroje burzovních dat. Jelikož cílem aplikace není budování vlastního datového feedu, ale návrh a implementace monitorovací aplikace nad existujícími tržními daty, bylo jako vhodné řešení zvoleno využití veřejně dostupného externího API.

Zároveň bylo při návrhu počítáno s využitím WebSocket komunikace pro přenos vybraných aktualizací v reálném čase. WebSocket byl zvolen proto, že oproti klasickému periodickému dotazování umožňuje efektivnější a bezprostřednější přenos dat mezi serverovou a klientskou částí systému. To je pro aplikaci zaměřenou na nízkolatenční upozorňování na změny cen přirozená volba. Pro oba typy komunikací byl zvolen Finnhub [63, 64].

4.4 Návrh datového modelu

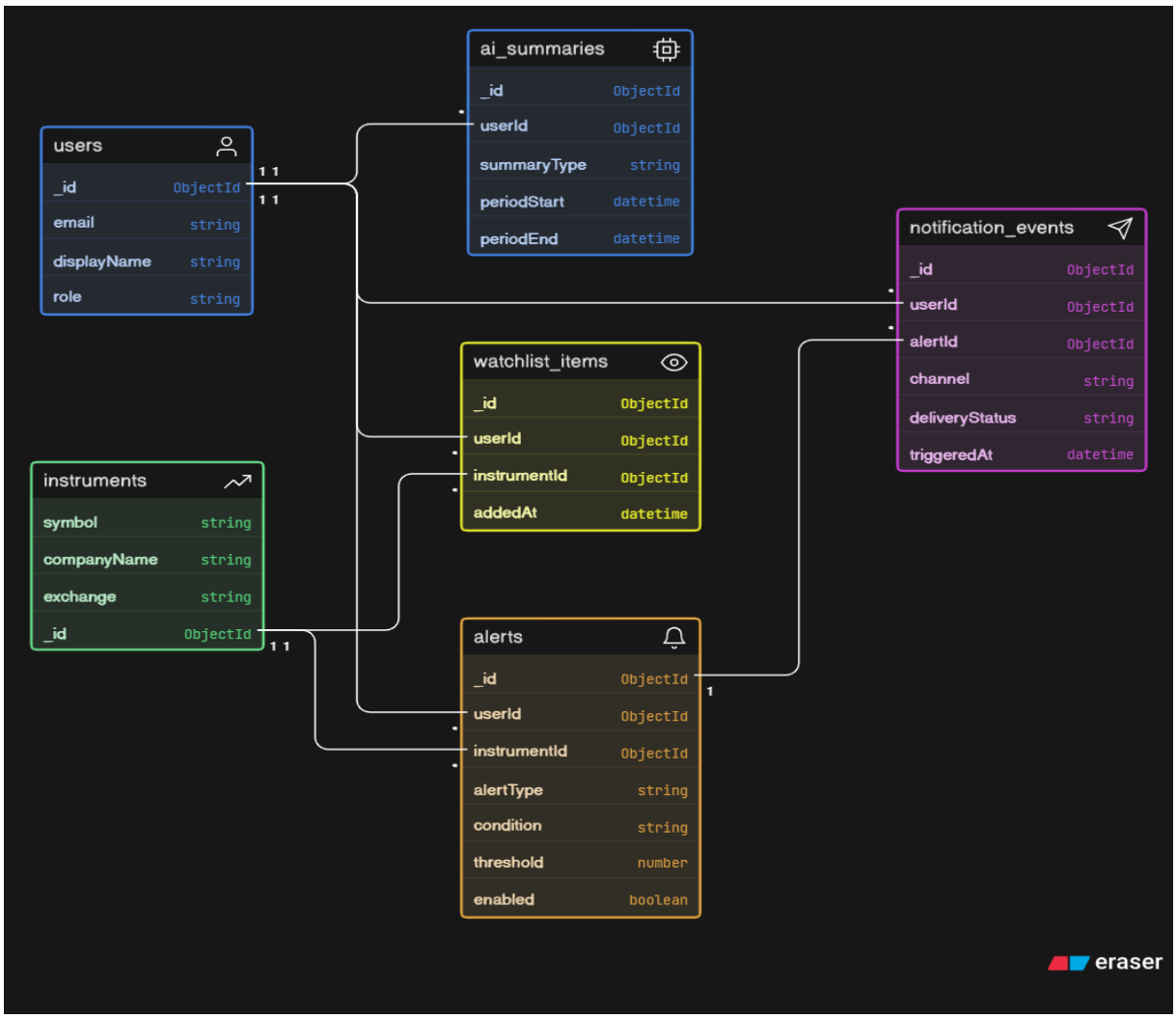
Návrh datového modelu vychází z požadavků na ukládání uživatelských dat, definovaných alertů, sledovaných instrumentů a záznamů o vyhodnocených událostech. Jelikož aplikace pracuje s daty, jejichž struktura se může v čase dále rozšiřovat, byl datový model navržen s důrazem na flexibilitu, přehlednost a možnost postupného rozvoje bez nutnosti zásadních změn celé databázové vrstvy.

Základní entitou systému je uživatel. Na uživatele jsou navázány další části systému, zejména jeho seznam sledovaných titulů, definované alerty a historie vzniklých

notifikačních událostí. Uživatel je tak v datovém modelu chápán jako vlastník většiny provozních dat aplikace.

Další entitou je sledovaný instrument. Návrh datového modelu počítá s tím, že jeden uživatel může sledovat více instrumentů současně a nad každým z nich definovat více různých pravidel. Z toho důvodu je vhodné oddělit samotný identifikátor instrumentu od konkrétních alertů, které jsou nad ním vytvářeny.

Klíčovou roli v datovém modelu mají entity reprezentující alerty. Každý alert musí nést informaci o tom, ke kterému uživateli a ke kterému instrumentu se vztahuje, jaký typ podmínky sleduje a za jakých okolností má být považován za splněný. Návrh počítá s tím, že alert nebude pouze statickým záznamem prahové hodnoty, ale bude obsahovat i stavové informace potřebné pro jeho správné vyhodnocování. To je důležité zejména kvůli tomu, aby bylo možné zabránit opakovanému spouštění stejného upozornění při kolísání ceny kolem hraniční hodnoty.



Obrázek 3: Návrh datového modelu. Zdroj: vlastní.

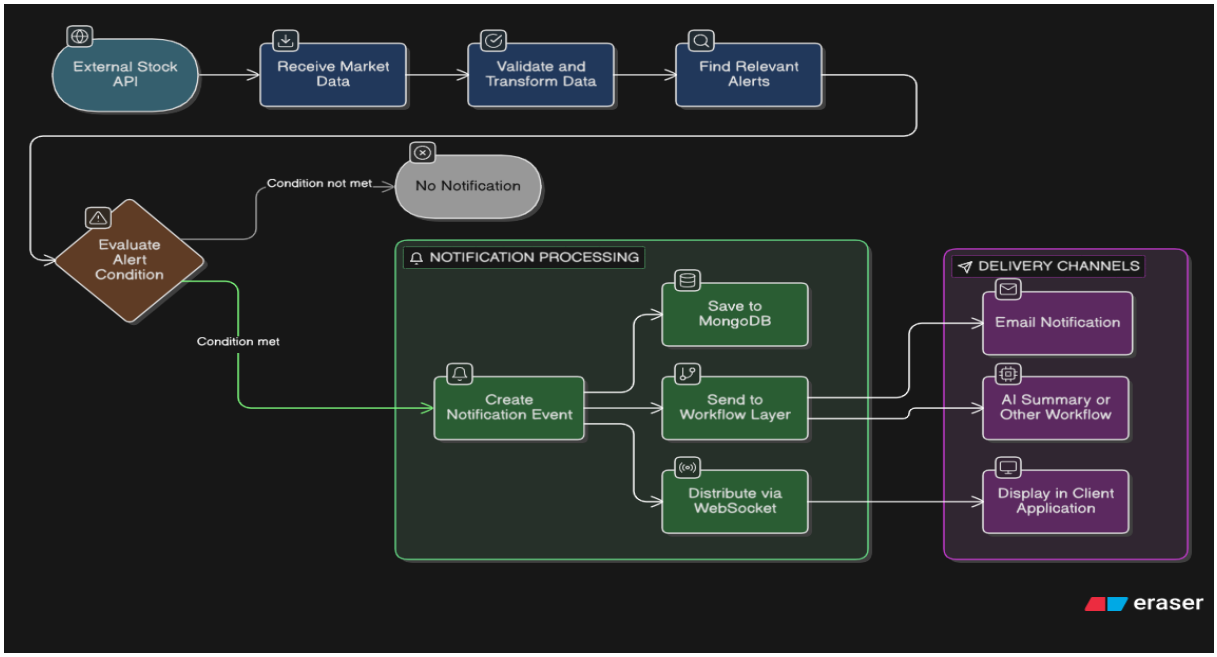
4.5 Návrh zpracování dat v reálném čase

Schopnost reagovat na tržní výkyvy s co nejnižší latencí je fundamentálním pilířem celého systému. Zpracování dat v reálném čase neslouží k účelům vysokofrekvenčního obchodování, ale k zajištění spolehlivého a včasného upozornění uživatele. Jeho cílem je zajistit, aby příchozí tržní data byla průběžně přijímána, vyhodnocována vzhledem k uživatelsky definovaným podmínkám a v případě splnění pravidla vedla ke vzniku odpovídající notifikační události.

Z návrhového hlediska lze tok dat systémem rozdělit do několika navazujících kroků. Nejprve dochází k přijetí dat z externího zdroje, který poskytuje informace o vývoji ceny a dalších relevantních údajích o sledovaných instrumentech. Tato data je následně nutné normalizovat do podoby, se kterou bude systém dále pracovat jednotným

způsobem. Po přijetí a normalizaci dat následuje identifikace těch alertů, které se vztahují ke konkrétnímu instrumentu. Návrh systému předpokládá, že není účelné při každé změně vyhodnocovat všechny alerty v systému, ale pouze ty, které odpovídají právě aktualizovanému titulu. Tím je snížena zbytečná výpočetní zátěž a zvyšuje se efektivita systému. Následně dochází k samotnému vyhodnocení podmínky, tedy porovnání nově přijaté hodnoty s pravidlem definovaným uživatelem.

Pokud je podmínka splněna, systém nevytváří pouze okamžitou vizuální reakci v uživatelském rozhraní, ale generuje samostatnou notifikační událost. Tato událost představuje mezikrok mezi vyhodnocením alertu a jeho doručením uživateli. Součástí návrhu musí být také ošetření situací, kdy cena osciluje těsně kolem nastavené hranice. V takovém případě by bez dalších mechanismů mohlo docházet k opakovanému generování totožných upozornění v krátkých časových intervalech. Návrh proto počítá s tím, že alert nebude po splnění vyhodnocován zcela bezstavově, ale že systém zohlední jeho předchozí stav nebo uplatní časovou prodlevu mezi jednotlivými upozorněními. Tím se snižuje riziko vzniku nadměrného množství notifikací a zvyšuje se praktická použitelnost aplikace.



Obrázek 4: Návrh toku dat a vyhodnocení alertu. Zdroj: vlastní.

4.6 Návrh uživatelského rozhraní

Návrh uživatelského rozhraní vychází z potřeby vytvořit aplikaci, která bude srozumitelná, rychle ovladatelná a přehledná. V případě monitorování burzovních informací je uživatelské rozhraní důležitou součástí celkové použitelnosti systému, protože právě jeho prostřednictvím uživatel sleduje změny cen, spravuje alerty a reaguje na vzniklé notifikace. Návrh rozhraní proto klade důraz především na jednoduchost orientace, minimalizaci zbytečných kroků a jasné oddělení hlavních funkčních oblastí aplikace.

Z pohledu uživatele lze rozhraní rozdělit do několika základních částí. První z nich představuje přístupová část systému, která zahrnuje přihlášení a práci s uživatelským účtem. Druhou oblast tvoří přehled sledovaných instrumentů, který uživateli umožňuje mít na jednom místě soustředěné základní informace o vybraných titulech. Třetí oblastí je detail konkrétního instrument. Čtvrtou důležitou částí je přehled upozornění a historie vzniklých událostí.

Při návrhu rozhraní je důležité, aby uživatel mohl snadno rozlišit mezi běžnými informacemi a událostmi, které vyžadují jeho pozornost. Rozhraní proto musí vizuálně podporovat hierarchii informací, kdy nejvýznamnější změny nebo upozornění vystupují do popředí, zatímco méně důležitý obsah zůstává doplňkový. Tato zásada je zvlášť důležitá u aplikace, která má fungovat jako monitorovací nástroj, nikoli jako rozsáhlá analytická platforma s vysokou informační hustotou.

Návrh uživatelského rozhraní zároveň vychází z požadavku na jednoduchou správu alertů. Vytvoření upozornění by mělo být pro uživatele co nejpřímější a nemělo by vyžadovat složitou konfiguraci. Rozhraní tedy musí podporovat jasné zadání sledovaného instrument. Důležitá je také přehledná správa již vytvořených alertů, aby bylo možné je bez obtíží upravit, deaktivovat nebo odstranit.

5 Implementace aplikace

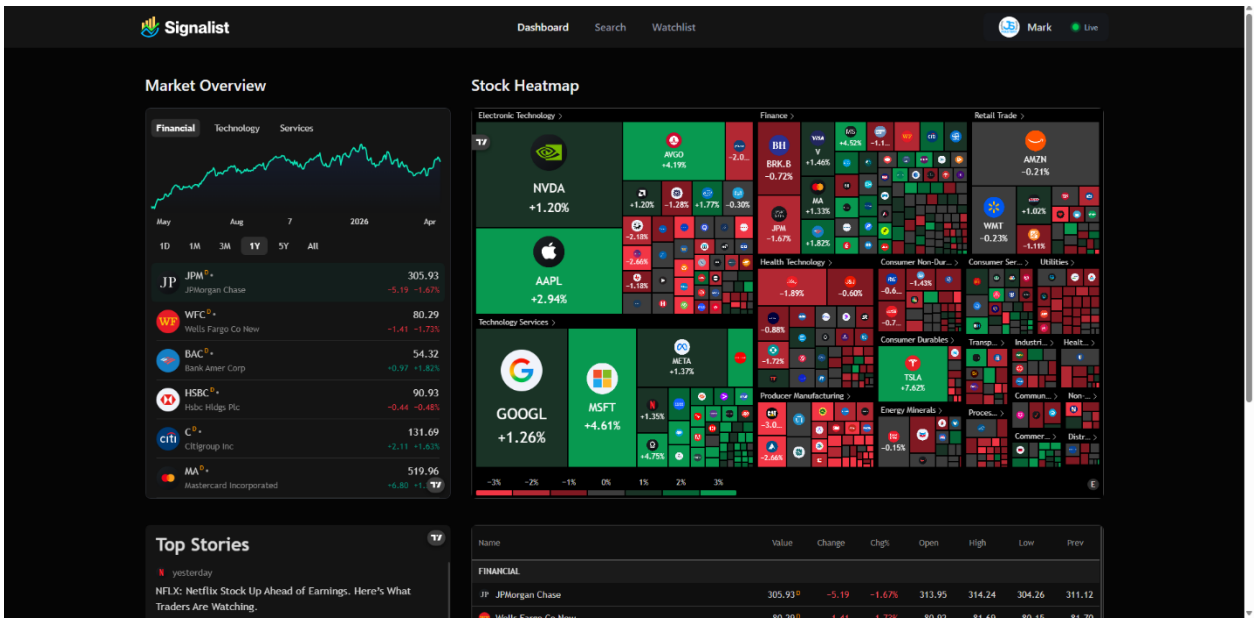
Tato kapitola detailně popisuje technickou realizaci navrženého systému. Zabývá se strukturou zdrojového kódu, způsobem implementace databázových modelů, zajištěním bezpečnosti a programovým řešením klíčových funkcionalit, primárně příjmu burzovních dat až po vyhodnocování alertů.

5.1 Implementace aplikační a prezentační vrstvy

Aplikace je implementována jako full-stack webové řešení postavené na frameworku Next.js. Toto řešení umožňuje realizovat prezentační vrstvu i významnou část aplikační logiky v jednom technologickém celku, což zjednodušuje vývoj, správu i následné nasazení systému. Struktura projektu využívá moderní přístup App Router a je rozdělena do několika logických celků.

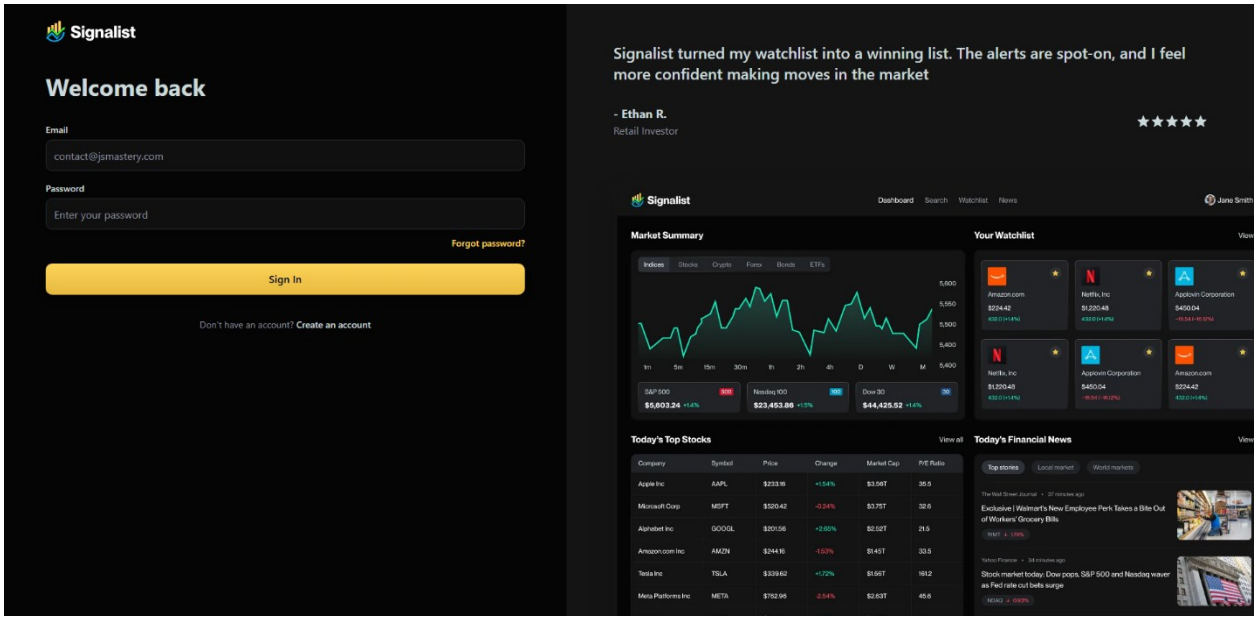
- app/ – obsahuje definici rout a samotných stránek. Adresář je logicky rozdělen na veřejnou autentizační část ((auth)) a na chráněnou aplikační část ((root)), která obsahuje hlavní dashboard, správu watchlistu a dynamické routy pro detail jednotlivých titulů.
- components/ – slouží k uchování znovupoužitelných prvků uživatelského rozhraní, jako jsou tlačítka, dialogová okna nebo grafy.
- database/ – obsahuje modely a konfigurační logiku pro spojení s databází.
- lib/ a middleware/ – sdružují pomocnou aplikační logiku, serverové akce a bezpečnostní filtry.

Prezentační vrstva je realizována pomocí React komponent. Pro stylování uživatelského rozhraní byl použit framework Tailwind CSS v kombinaci s komponentovým přístupem založeným na Shadcn UI. Tato kombinace umožňuje vytvářet responzivní, vizuálně konzistentní a dobře znovupoužitelné prvky rozhraní. Vzhledem k charakteru aplikace byl při návrhu rozhraní kladen důraz především na přehlednost, rychlou orientaci v datech a jednoduchou správu uživatelských alertů.



Obrázek 5: Hlavní dashboard aplikace. Zdroj: vlastní.

Z hlediska funkčnosti prezentační vrstva zahrnuje zejména hlavní dashboard, vyhledávání akciových titulů, detail konkrétní akcie, správu watchlistu, formuláře pro vytváření alertů a rozhraní pro zobrazení notifikačních událostí. Veřejná část aplikace obsahuje stránky pro přihlášení, registraci a obnovu hesla, zatímco hlavní aplikační část je přístupná pouze ověřeným uživatelům. Tím je dosaženo jasného oddělení veřejných a neveřejných částí systému.



Obrázek 6: Veřejná část aplikace. Zdroj: vlastní.

Aplikační vrstva zajišťuje zpracování požadavků přicházejících z uživatelského rozhraní, komunikaci s databází, externími službami i vyhodnocovací logikou systému. V rámci implementace tak tvoří prostředníka mezi uživatelem, uloženými daty a vstupními burzovními informacemi.

5.2 Implementace datové vrstvy

Datová vrstva aplikace je realizována pomocí databáze MongoDB s využitím knihovny Mongoose. Připojení k databázi je řešeno centrálně a je navrženo tak, aby při opakovaném použití nedocházelo ke zbytečnému vytváření nových spojení. Tento způsob je vhodný zejména v prostředí serverově renderovaných aplikací, kde je důležité zachovat stabilní a efektivní databázovou komunikaci [65].

Datový model byl implementován tak, aby odpovídal hlavním funkčním oblastem systému. Samostatně jsou evidovány položky watchlistu, alerty a historie vyhodnocených alertových událostí. Takové rozdělení podporuje přehlednost systému a umožňuje lépe oddělit aktivní pravidla od historických záznamů o jejich triggernutí. Jejich struktura je rozdělena do několika samostatných modelů podle hlavních funkčních oblastí systému:

- Watchlist model: Realizuje vazbu mezi uživatelem a sledovaným symbolem. Schéma obsahuje složený unikátní index (userId + symbol), který databázově zabraňuje vzniku duplicitních záznamů u jednoho uživatele.
- Alert model: Uchovává definici pravidla pro upozornění. Kromě samotné podmínky a prahové hodnoty model ukládá i stavové údaje potřebné pro vyhodnocování (cooldownMinutes, lastTriggeredAt, lastSeenValue). Nad kolekcí jsou definovány indexy pro rychlé vyhledávání aktivních pravidel podle uživatele a instrumentu. Konkrétní implementace databázového schématu pro entitu Alert je uvedena v ukázce kódu 1. Z ukázky je patrné začlenění stavových proměnných (např. lastTriggeredAt), které tvoří nezbytný základ pro mechanismus ochrany proti vícenásobnému spuštění (cooldown).

```
const AlertSchema = new Schema<IAlert>(
    {
        userId: { type: String, required: true, index: true },
        symbol: { type: String, required: true, uppercase: true, trim: true, index: true },
        company: { type: String, required: true },

        alertType: { type: String, enum: ["price", "percent", "volume"], required: true },
        threshold: { type: Number, required: true, min: 0 },
        condition: { type: String, enum: ["upper", "lower"], required: true },

        enabled: { type: Boolean, default: true, index: true },

        cooldownMinutes: { type: Number, default: 15, min: 0 },
        lastTriggeredAt: { type: Date, default: null },
        lastSeenValue: { type: Number, default: null },
    },
    { timestamps: true }
);

```

Ukázka kódu 1: Schéma alertu. Zdroj: vlastní.

- AlertEvent model: Slouží k uchování historie vyhodnocených alertů. Zaznamenává se zde reálná cena v okamžiku splnění podmínky a přesný čas spuštění, což umožňuje zpětné zobrazení notifikační historie. Struktura záznamu historické události je definována v ukázce kódu 2.

```
const AlertEventSchema = new Schema<IAlertEvent>(
    {
        ruleId: { type: Schema.Types.ObjectId, ref: "Alert", required: true, index: true },
        userId: { type: String, required: true, index: true },

        symbol: { type: String, required: true, uppercase: true, trim: true, index: true },
        condition: { type: String, required: true, enum: ["upper", "lower"] },
        threshold: { type: Number, required: true },

        triggerPrice: { type: Number, required: true },
        triggeredAt: { type: Date, required: true, index: true },

        reason: { type: String, default: null },
    },
    { timestamps: true }
);
```

Ukázka kódu 2: Schéma alertEventu. Zdroj: vlastní.

Datová vrstva je doplněna vhodnou indexací nad nejčastěji používanými atributy, zejména nad identifikací uživatele, symbolem a časem vzniku události. Tím je dosaženo efektivnějšího filtrování a načítání dat v běžných scénářích použití.

5.3 Implementace autentizace a správy uživatelů

Autentizace a správa uživatelských relací je realizována pomocí knihovny Better Auth. Toto řešení zajišťuje přihlášení uživatelů, správu session a ochranu neveřejných částí aplikace. Veřejná část systému obsahuje samostatné stránky pro registraci, přihlášení a obnovu hesla, zatímco hlavní aplikační část je dostupná pouze po úspěšném ověření identity uživatele.

Pro ochranu neveřejných částí systému je implementován middleware mechanismus, který kontroluje přítomnost aktivní session. Pokud uživatel není přihlášen, je přesměrován na veřejnou část aplikace. Programová realizace této ochrany směrování je znázorněna v ukázce kódu 3. Ukázka ukazuje využití metody getSessionCookie, která bezpečně ověřuje stav přihlášení. Konfigurace pole matcher zajišťuje, že z této kontroly jsou záměrně vyjmuty veřejné API cesty. Tím je zajištěno, že k funkcím, jako je správa watchlistu, práce s alerty nebo přístup k notifikační historii, mají přístup pouze ověření uživatelé.

```
import { NextRequest, NextResponse } from "next/server";
import { getSessionCookie } from "better-auth/cookies";

export async function middleware(request: NextRequest) {
    const sessionCookie = getSessionCookie(request);

    if (!sessionCookie) {
        return NextResponse.redirect(new URL("/", request.url));
    }

    return NextResponse.next();
}

export const config = {
    matcher: [
        '/((?!api|_next/static|_next/image|favicon.ico|sign-in|sign-up|assets).*)',
    ],
};

```

Ukázka kódu 3: Middleware. Zdroj: vlastní.

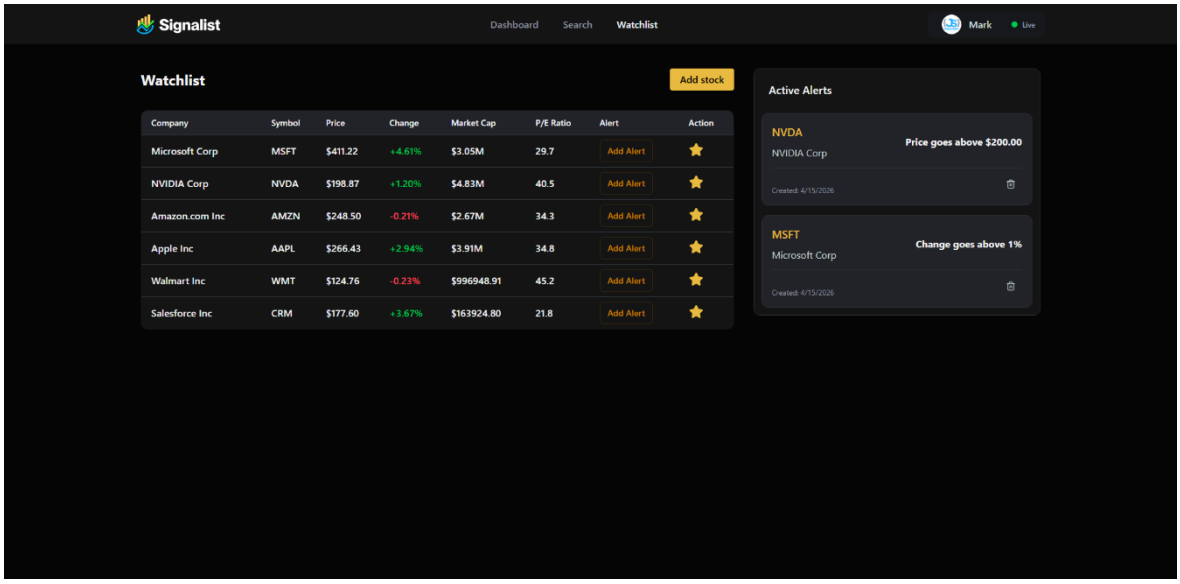
Správa uživatelských dat je v celé aplikaci postavena na vazbě provozních záznamů ke konkrétnímu uživateli. Každý watchlist, každý alert i každá alertová událost jsou přiřazeny identifikaci uživatele. Tento princip je důležitý nejen z hlediska bezpečnosti,

ale i z hlediska správné funkčnosti systému, protože každý uživatel pracuje se svým vlastním souborem sledovaných titulů a pravidel.

Autentizační vrstva tak v implementaci neplní pouze roli přihlášení, ale představuje průřezový mechanismus, který ovlivňuje řízení přístupu, práci s daty i notifikační logiku celé aplikace.

5.4 Implementace správy watchlistu a alertů

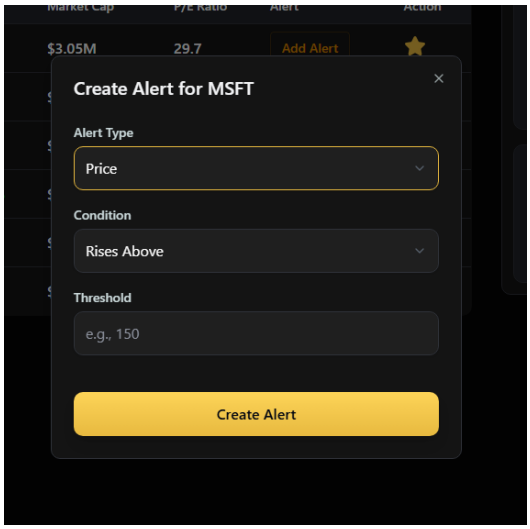
Správa watchlistu a alertů tvoří hlavní funkční jádro aplikace. Uživatel má možnost vybrané akciové tituly přidávat do vlastního watchlistu, zobrazovat je a případně je z něj odebírat. Watchlist tak představuje personalizovaný seznam sledovaných instrumentů, nad nimiž může uživatel dále vytvářet vlastní pravidla upozornění.
Každá položka watchlistu je vázána ke konkrétnímu uživateli. Implementace současně zabraňuje tomu, aby si uživatel přidal stejný titul opakovaně. Tím je zachována konzistence uložených dat i přehlednost uživatelského rozhraní.



Obrázek 7: Uživatelský watchlist. Zdroj: vlastní.

Nad sledovanými tituly může uživatel vytvářet alerty. Implementace podporuje více typů podmínek, konkrétně cenové alerty, procentní změny a alerty navázané na objem obchodů. Alert současně rozlišuje, zda má být pravidlo aktivováno při překročení horní hranice nebo při poklesu pod hranici dolní. Tento přístup umožňuje pokrýt základní

scénáře monitorování trhu bez nutnosti komplikovaného nastavování ze strany uživatele.



Obrázek 8: Formulář pro vytvoření alertu. Zdroj: vlastní.

Implementace alertů zároveň zohledňuje i problém opakovaného triggernutí při oscilaci hodnoty kolem prahové meze. Proto jsou součástí modelu stavové atributy a cooldown mechanismus, které pomáhají omezovat nadměrné nebo duplicitní notifikace. Tento prvek je důležitý zejména u aplikace pracující s daty v reálném čase, kde by jinak mohlo docházet k zahlcení uživatele.

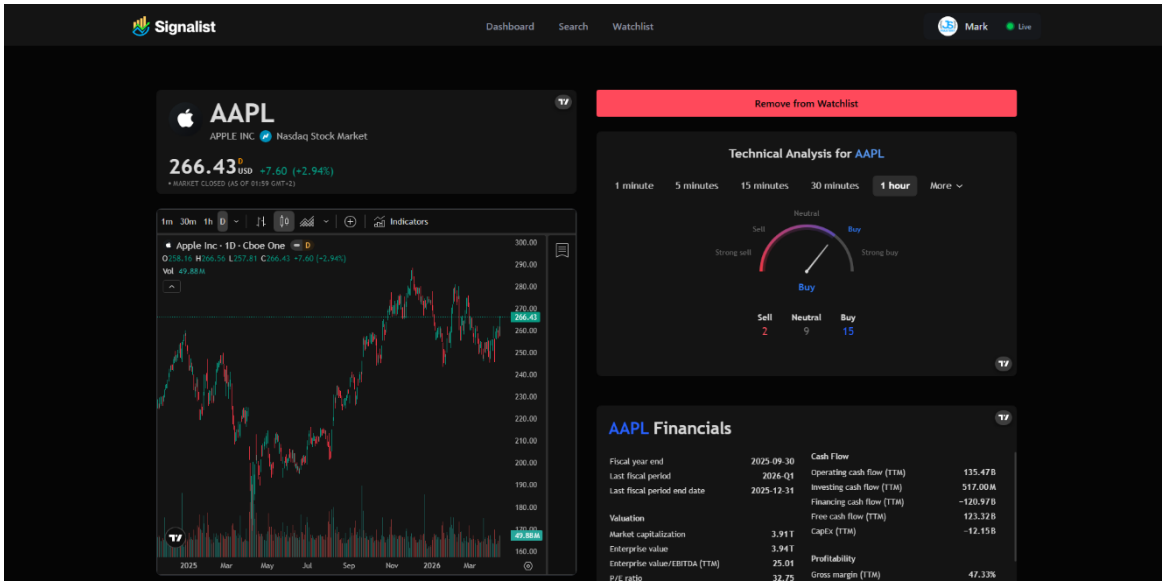
5.5 Implementace příjmu burzovních dat

Příjem burzovních dat je realizován prostřednictvím externího API Finnhub. Toto API slouží jako hlavní zdroj tržních dat využívaných v aplikaci jak pro zobrazování na dashboardu a v detailu instrumentu, tak pro následné vyhodnocování alertů. Systém tak nepracuje s lokálně generovanými nebo statickými daty, ale s reálnými datovými vstupy z externího poskytovatele.

Integrace s externím API je navržena tak, aby byla oddělena od prezentační vrstvy. Přijatá data jsou nejprve načtena a transformována do podoby, kterou dále využívá aplikační logika. Tento přístup zjednodušuje práci s daty v dalších částech systému a současně usnadňuje případné budoucí nahrazení zdroje dat jiným providerem.
Načítaná data slouží dvěma hlavním účelům. Prvním je vizualizace trhu v uživatelském rozhraní, zejména na hlavním dashboardu a ve stránce detailu konkrétní akcie. Druhým účelem je vstup pro alertovací logiku, která porovnává aktuální hodnoty s

podmínkami definovanými uživatelem. Příjem dat je tedy přímo propojen s hlavním účelem aplikace.

Při implementaci této části bylo nutné zohlednit i praktická omezení externích API, například limity počtu požadavků nebo možnou nedostupnost služby. Z tohoto důvodu je aplikace navržena tak, aby případné problémy s externím zdrojem nevedly k pádu celého systému, ale pouze k dočasnému omezení aktualizace dat.



Obrázek 9: Detail vybraného akciového titulu. Zdroj: vlastní.

5.6 Implementace komunikace v reálném čase

Pro zajištění okamžitého doručování upozornění a aktualizací bez nutnosti manuálního obnovování stránky byla do aplikace integrována technologie WebSocket. Architektura tohoto řešení je z důvodu odlišných nároků na infrastrukturu rozdělena na samostatný serverový proces a klientskou část.

Serverová část (Realtime Server) Logika WebSocket serveru je implementována v samostatném modulu realtime-server, jehož vstupním bodem je soubor src/index.ts. Toto oddělení od hlavní aplikace v Next.js bylo zvoleno záměrně, jelikož běžné serverless prostředí není optimalizováno pro udržování dlouhodobě otevřených (persistentních) spojení. Samostatný Node.js server přijímá události z hlavní aplikační vrstvy (například informaci o splnění alertu) a okamžitě je distribuuje připojeným klientům. Základní inicializace tohoto nezávislého WebSocket serveru, včetně nezbytné konfigurace CORS politik pro křížové dotazování z domény Vercelu, je uvedena v ukázce kódu 4 [64].

```
const server = http.createServer();
const io = new Server(server, {
    path: "/socket.io",
    cors: {
        origin: process.env.NEXT_PUBLIC_APP_URL ? [process.env.NEXT_PUBLIC_APP_URL, "http://localhost:3000"] : "*",
        methods: ["GET", "POST"],
    },
});
```

Ukázka kódu 4: Inicializace websocket serveru. Zdroj: vlastní.

Klientská část a správa spojení Na straně frontendu je navázání a správa připojení řešena v souboru lib/websocket/socket-client.ts. Tento modul zajišťuje inicializaci spojení s realtime serverem a definuje metody pro naslouchání příchozím událostem. Zapouzdření této logiky do jednoho souboru usnadňuje údržbu a umožňuje snadné volání WebSocket služeb z jakékoliv části uživatelského rozhraní. Adresné doručování zpráv konkrétnímu klientovi zajišťuje směrovací logika zobrazená v ukázce kódu 5.

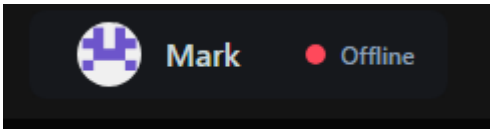
```
io.on("connection", (socket) => {
    const verifiedUserId = socket.data.userId as string | undefined;

    if (verifiedUserId) {
        socket.join(`user:${verifiedUserId}`);
        console.log(`👤 User ${verifiedUserId} connected to his private room`);
    }

    socket.on("identify", () => {
        const currentUserId = socket.data.userId as string | undefined;
        if (!currentUserId) return;

        for (const room of socket.rooms) {
            if (room.startsWith("user:") && room !== `user:${currentUserId}`) {
                socket.leave(room);
            }
        }

        socket.data.userId = currentUserId;
        socket.join(`user:${currentUserId}`);

        console.log(`👤 User ${currentUserId} identified`);
    });
});
```

Ukázka kódu 5: Přiřazení uživatele do privátní místnosti. Zdroj: vlastní.

Integrace do uživatelského rozhraní Pro vizuální reprezentaci real-time funkcí byly vytvořeny dvě hlavní React komponenty:

- WsStatusIndicator.tsx – Jednoduchá vizuální komponenta, která uživateli poskytuje neustálou zpětnou vazbu o stavu připojení k WebSocket serveru (např. indikace úspěšného připojení nebo výpadku sítě).
- RealtimeAlerts.tsx – Klíčová komponenta naslouchající na události ze serveru. Pokud vyhodnocovací logika na backendu zaznamená splnění podmínky alertu a odešle zprávu přes WebSocket, tato komponenta ji zachytí a bezprostředně zobrazí uživateli formou vyskakovací notifikace (toast) přímo v prohlížeči.

Tato implementace zajišťuje, že notifikační vrstva aplikace reaguje s minimální latencí, což je u softwaru pro monitorování burzovních dat kritickým požadavkem.



Obrázek 10: Indikace stavu websocket spojení. Zdroj: vlastní.

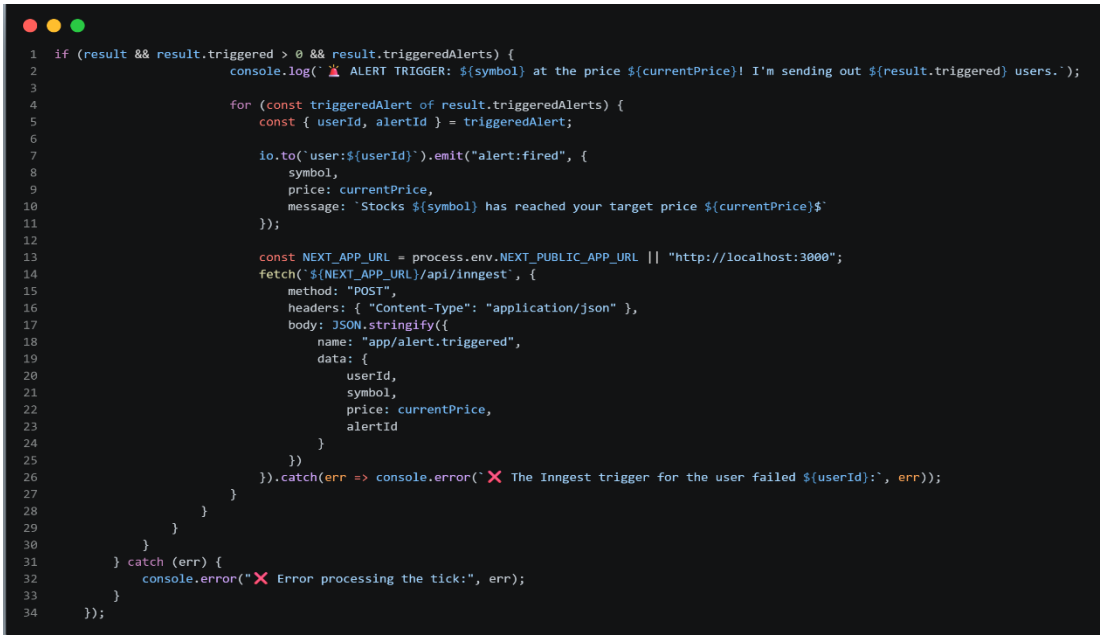
5.7 Implementace notifikační vrstvy

Notifikační vrstva je v aplikaci realizována jako kombinace okamžitých upozornění v uživatelském rozhraní a perzistence alertových událostí do databáze. Základním cílem této části systému je zajistit, aby po splnění definované podmínky nevznikla pouze jednorázová reakce v rozhraní, ale i dohledatelný záznam o tom, že daný alert byl skutečně aktivován. V implementaci tomu odpovídá samostatný model AlertEvent, který uchovává vazbu na pravidlo, uživatele, symbol, cenu v okamžiku triggernutí a čas vzniku události. Součástí modelu je i nepovinné pole reason, které slouží pro deduplikaci a ladění systému [66].

```
const rules = await Alert.find({
        symbol,
        "alertType": params.alertType,
        "enabled": true,
    }).lean();

    let triggeredCount = 0;

    const triggeredAlerts: Array<{ alertId: string; userId: string }> = [];

    for (const rule of rules) {
        const prev = rule.lastSeenValue ?? null;
        let crossed = false;

        if (rule.alertType === "percent") {
            crossed = didCrossPercentThreshold({
                condition: rule.condition,
                initial: prev,
                current: params.currentValue,
                threshold: rule.threshold,
            });
        } else {
            crossed = didCrossThreshold({
                condition: rule.condition,
                prev,
                current: params.currentValue,
                threshold: rule.threshold,
            });
        }

        const cooldownOk = isCooldownOver({
            lastTriggeredAt: rule.lastTriggeredAt ?? null,
            cooldownMinutes: rule.cooldownMinutes ?? 15,
            now,
        });
```

Ukázka kódu 6: Pravidla alertu. Zdroj: vlastní.

Při aktivním používání aplikace je uživatel na splnění alertu upozorněn prostřednictvím websocketové události alert:fired. Na klientské straně je tato logika implementována v komponentě RealtimeAlerts.tsx, která po přijetí události zobrazí toast notifikaci s popisem vzniklé události. Tím je zajištěna okamžitá odezva systému bez nutnosti ručního obnovení stránky. Současně je klientské websocket spojení vázáno na konkrétního uživatele prostřednictvím události identify, takže jsou doručovány pouze relevantní alerty. Postup vyhodnocení alertového pravidla je zachycen v ukázce kódu 6.



```
if (result && result.triggered > 0 && result.triggeredAlerts) {
                    console.log(`🚨 ALERT TRIGGER: ${symbol} at the price ${currentPrice}! I'm sending out ${result.triggered} users.`);

                    for (const triggeredAlert of result.triggeredAlerts) {
                        const { userId, alertId } = triggeredAlert;

                        io.to(`user:${userId}`).emit("alert:fired", {
                            symbol,
                            price: currentPrice,
                            message: `Stocks ${symbol} has reached your target price ${currentPrice}$`
                        });

                        const NEXT_APP_URL = process.env.NEXT_PUBLIC_APP_URL || "http://localhost:3000";
                        fetch(`${NEXT_APP_URL}/api/inngest`, {
                            method: "POST",
                            headers: { "Content-Type": "application/json" },
                            body: JSON.stringify({
                                name: "app/alert.triggered",
                                data: {
                                    userId,
                                    symbol,
                                    price: currentPrice,
                                    alertId
                                }
                            })
                        }).catch(err => console.error(`❌ The Inngest trigger for the user failed ${userId}:`, err));
                    }
                }
            }
        } catch (err) {
            console.error("❌ Error processing the tick:", err);
        }
    });
```

Ukázka kódu 7: Odeslání realtime notifikace po splnění alertu. Zdroj: vlastní.

Implementace alertů zároveň počítá s omezením duplicitních upozornění. Model Alert obsahuje stavové atributy cooldownMinutes, lastTriggeredAt a lastSeenValue, které slouží k omezení opakovaného triggernutí v situacích, kdy sledovaná hodnota krátkodobě osciluje kolem prahové hranice. Tento mechanismus je důležitý pro zachování použitelnosti systému, protože zabraňuje nadměrnému počtu opakovaných notifikací.

```
if (crossed && cooldownOk) {
            const cooldownMs = (rule.cooldownMinutes ?? 15) * 60 * 1000;
            const latestAllowed = new Date(now.getTime() - cooldownMs);

            // Atomic guard prevents duplicate triggers
            const updated = await Alert.findOneAndUpdate(
                {
                    _id: rule._id,
                    enabled: true,
                    lastSeenValue: rule.lastSeenValue ?? null,
                    $or: [{ lastTriggeredAt: null }, { lastTriggeredAt: { $lte: latestAllowed } }],
                },
                {
                    $set: {
                        lastTriggeredAt: now,
                        lastSeenValue: params.currentValue,
                    },
                },
                { new: true }
            ).lean();

            if (updated) {
                await AlertEvent.create({
                    ruleId: updated._id,
                    userId: updated.userId,
                    symbol: updated.symbol,
                    alertType: updated.alertType,
                    condition: updated.condition,
                    threshold: updated.threshold,
                    triggerValue: params.currentValue,
                    triggeredAt: now,
                    reason: "threshold_crossing",
                });
```

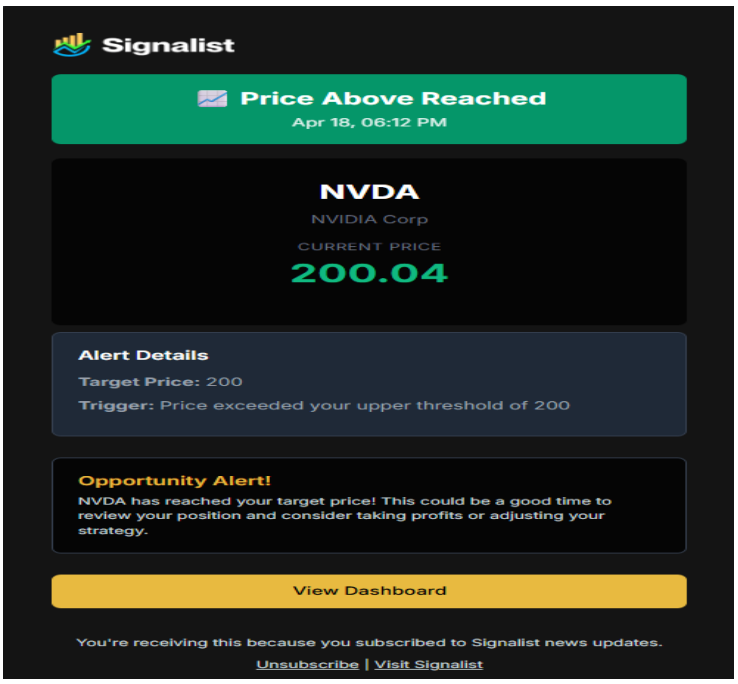
Ukázka kódu 8: Ochrana proti duplicitnímu triggernutí. Zdroj: vlastní.

5.8 Implementace asynchronních úloh a umělé inteligence

Pro zajištění plynulého chodu aplikace a zamezení blokování hlavního uživatelského vlákna při časově náročných operacích byl do systému integrován nástroj Inngest. Tento nástroj slouží k orchestraci procesů na pozadí a ke spolehlivému vykonávání asynchronních úloh (background jobs).

Konfigurace a orchestrace procesů Infrastruktura pro zpracování asynchronních úloh je logicky sdružena v adresáři lib/inngest. Samotná inicializace Inngest klienta probíhá v souboru client.ts, zatímco definice jednotlivých úloh (funkcí) je umístěna v souboru functions.ts. Propojení tohoto nástroje s frameworkem Next.js je realizováno prostřednictvím dedikovaného API endpointu v app/api/inngest/route.ts. Díky tomuto rozhraní může platforma Inngest bezpečně a spolehlivě spouštět definované úlohy na základě vzniklých aplikačních událostí (eventů) [62].

E-mailové notifikace na pozadí Jednou z hlavních asynchronních agend systému je odesílání e-mailových upozornění. Vzhledem k tomu, že komunikace s poštovním SMTP serverem může trvat i několik vteřin, je požadavek delegován do Inngest fronty, aby nebyla zdržována interakce uživatele v prohlížeči. K fyzickému doručení zpráv je využita knihovna Nodemailer, jejíž konfigurace se nachází v lib/nodemailer/index.ts. Vizuální podoba a struktura odesílaných e-mailů jsou odděleny do znovupoužitelných šablon v souboru templates.ts. Zapojení nástroje Inngest zde navíc automaticky zajišťuje mechanismus opakování (retry) v případě dočasného výpadku poštovních služeb [66].



Obrázek 11: Alert notifikace přes email. Zdroj: vlastní.

Generování doplňkových AI souhrnů Inovativním prvkem implementace je zapojení velkého jazykového modelu pro tvorbu textových analytických souhrnů nad burzovními daty. Volání externího AI rozhraní (např. modelu Gemini) se vyznačuje vysokou časovou latencí, a proto je i tato logika realizována výhradně jako úloha na pozadí.

V systému je implementován soubor lib/inngest/prompts.ts, ve kterém jsou uloženy předpřipravené a strukturované instrukce (prompty) pro jazykový model. Tím je zajištěno, že umělá inteligence dostává data vždy v jednotném formátu a dokáže ze surových tržních tiků nebo z historie uživatelských alertů vygenerovat formátovaný, srozumitelný report bez nutnosti jakéhokoliv manuálního zásahu uživatele.

5.9 Nasazení aplikace

Proces nasazení (deployment) plně reflektuje architektonické rozdělení systému do nezávislých vrstev. Místo monolitického nasazení na jeden virtuální server byla zvolena moderní cloudová infrastruktura, která každé části systému poskytuje technologicky nejvhodnější běhové prostředí.

Nasazení aplikační a prezentační vrstvy (Vercel) Pro hosting hlavní části aplikace, postavené na frameworku Next.js, byla zvolena cloudová platforma Vercel. Tato

platforma poskytuje nativní podporu pro Next.js a umožňuje bezproblémové nasazení typu serverless. Proces nasazení je plně automatizován pomocí CI/CD (Continuous Integration / Continuous Deployment) pipeliny propojené s repozitářem zdrojového kódu. Jakmile dojde ke schválení změn do produkční větve, Vercel automaticky spustí proces sestavení (build) a novou verzi aplikace distribuuje na svou globální síť [67].

Nasazení datové vrstvy (MongoDB Atlas) Databázová vrstva je hostována jako plně spravovaná cloudová služba (DBaaS) prostřednictvím platformy MongoDB Atlas. Tento přístup zbavuje vývojáře nutnosti konfigurovat a udržovat vlastní databázový server. MongoDB Atlas automaticky řeší zálohování dat, bezpečnostní aktualizace a umožňuje plynulé škálování výkonu v závislosti na rostoucím objemu uložených alertů a historických notifikačních událostí.

Nasazení a integrace WebSocket serveru (Railway) Z provozního hlediska není WebSocket server nasazen uvnitř stejného běhového prostředí jako hlavní webová aplikace. Serverless architektura platformy Vercel má totiž omezenou dobu běhu jednotlivých funkcí a není určena pro dlouhodobě otevřená TCP spojení. Proto je real-time server hostován jako samostatná služba na platformě Railway, která je optimalizována pro běh nepřetržitých Node.js procesů a správu souběžných perzistentních spojení [68].

Správa konfigurace a proměnné prostředí Bezpečnost a propojení těchto tří izolovaných platforem jsou plně řízeny prostřednictvím proměnných prostředí (Environment Variables). Tato konfigurace zahrnuje připojovací řetězce k databázi v MongoDB Atlas (URI), tajné klíče pro autentizační vrstvu, přístupové tokeny k externímu burzovnímu API a adresu WebSocket serveru na platformě Railway. Tento přístup důsledně dodržuje moderní cloudovou metodiku *Twelve-Factor App*. Citlivé údaje jsou izolovány v administrátorských rozhraních jednotlivých platforem, čímž je zamezeno jejich úniku do verzovacího systému zdrojových kódů.

Výhody zvoleného modelu nasazení Hlavním benefitem takto distribuovaného nasazení (Vercel, MongoDB Atlas, Railway) je vysoká modularita a odolnost. Hlavní webová aplikace může být v případě zvýšené návštěvnosti škálována zcela nezávisle

na real-time vrstvě a databázi. Toto rozdělení plně koresponduje s teoretickým návrhem softwarové architektury a zásadně zvyšuje provozní stabilitu celého systému.

6 Testování

6.1 Cíl testování

Cílem testovací fáze bylo ověřit, zda implementovaná aplikace spolehlivě splňuje hlavní funkční a technické požadavky definované v návrhové části. Testování bylo primárně zaměřeno na ověření správy uživatelských portfolií (watchlistů), korektního vytváření a vyhodnocování alertů, funkčnosti real-time distribuční vrstvy a základní výkonnostní odezvy systému.

Vzhledem k prototypovému charakteru řešení nebylo primárním cílem realizovat rozsáhlé zátěžové testy (stress-testy) infrastruktury, ale prokázat stabilitu a korektní chování systému v reprezentativních scénářích reálného použití. Zvolená strategie proto kombinuje vícevrstvý přístup: manuální testování uživatelského rozhraní (UI), automatizované end-to-end (E2E) testy ve frameworku Playwright, integrační testy aplikační logiky a orientační měření systémové latence.

6.2 Manuální testování hlavních scénářů

Manuální testování probíhalo formou průchodu předem definovanými uživatelskými scénáři. Účelem bylo potvrdit srozumitelnost uživatelského rozhraní, plynulou návaznost jednotlivých obrazovek a celkovou ergonomii aplikace (UX).

Pro manuální testování byla využita nasazená produkční verze aplikace. Během procesu bylo ověřeno, že klientské toky (flows) probíhají bez výskytu chybových stavů. Výsledky klíčových testovacích scénářů jsou shrnuty v Tabulce 1.

ID	Testovaný scénář	Vstupní podmínky	Očekávaný výsledek	Skutečný stav
MT1	Přihlášení uživatele	Existující účet	Přesměrování do chráněné části	Splněno

MT2	Přidání akcie do watchlistu	Přihlášený uživatel	Akcie se objeví v přehledu watchlistu	Splněno
MT3	Vytvoření alertu	Přihlášený uživatel, vybrán titul	Alert je trvale uložen a zobrazen v seznamu	Splněno
MT4	Deaktivace alertu	Existující aktivní alert	Alert se přepne do pasivního stavu a nevyhodnocuje se	Splněno
MT5	Real-time notifikace	Aktivní WS spojení, splněná podmínka	Zobrazí se "toast" notifikace na obrazovce uživatele	Splněno

Tabulka 1: Manuální scénáře. Zdroj: vlastní.

Během testování byly rovněž identifikovány přirozené limity prototypového řešení, zejména závislost plynulosti aplikace na latenci externího poskytovatele dat a nezbytnost aktivního okna prohlížeče pro doručení in-app notifikace.

6.3 Automatizované end-to-end testy pomocí Playwright

Pro zajištění opakovatelnosti testování a zachycení případných regresních chyb byly do projektu začleněny automatizované end-to-end (E2E) testy. K tomuto účelu byl nasazen moderní testovací framework Playwright. Tento nástroj simuluje reálné chování uživatele přímo v prostředí prohlížeče. Automatizovaně navštěvuje URL adresy, vyplňuje formuláře, kliká na ovládací prvky a ověřuje výsledný stav aplikace. Zásadní výhodou tohoto nástroje je vestavěná podpora pro asynchronní čekání na stav sítě [69].

6.3.1 Test přihlášení a ochrany

Testovací skript ověřuje funkčnost autentizačního middlewaru. Nejprve nasimuluje pokus o neoprávněný vstup na chráněnou stránku /watchlist (ověření HTTP přesměrování). Následně provede automatizované vyplnění přihlašovacího formuláře a asertuje úspěšný přístup do neveřejné části aplikace.

6.3.2 Test správy watchlistu a alertů

Sada testů ověřuje celistvost formulářových operací. Robotizovaný klient vyhledá konkrétní akciový titul, klikne na tlačítko pro přidání do watchlistu a ověří přítomnost odpovídajícího UI elementu v seznamech. Obdobným způsobem je automatizovaně proklikán formulář pro vytvoření alertu (zadání typu podmínky a mezní hodnoty) včetně aserce zobrazení úspěšné potvrzovací zprávy.

6.4 Integrační testy alert logiky

Samotné jádro systému – vyhodnocovací engine pro alerty – vyžadovalo hlubší prověření nad rámec uživatelského rozhraní. Pomocí integračních testů byla ověřována přesnost a spolehlivost aplikační logiky zpracovávající surové cenové tiky. Testy simulovaly vstup dat a ověřovaly změny stavu v databázových modelech (MongoDB).

Ověřovány byly specifické krajní případy (edge-cases):

- Zpracování podmínek typu překročení maxima (upper) a poklesu pod minimum (lower).
- Ignorování alertů, které uživatel explicitně deaktivoval (enabled: false).
- Zátěž a oscilace (Cooldown mechanismus): Byla testována situace, kdy jsou v rychlém sledu zaslány tři cenové tiky splňující podmínku. Test asertoval, že systém úspěšně zaznamená pouze první tik, vytvoří přesně jeden dokument v kolekci AlertEvent a zbylé dva tiky bezpečně zahodí, dokud neuplyne definovaná časová prodleva (cooldown).

Tímto testováním byla potvrzena schopnost systému úspěšně odolávat nežádoucímu duplicitnímu notifikování.

6.5 Testování real-time komunikace a notifikací

Samostatná pozornost byla věnována distribuční vrstvě. Cílem nebylo pouze ověřit uložení dat v databázi, ale prozkoumat rychlost a stabilitu doručení informace z backendu na obrazovku klienta.

Testovací procedury prověřily úspěšnost navázání asynchronního (WebSocket) spojení po přihlášení uživatele a schopnost komponenty WsStatusIndicator korektně reflektovat stav připojení. Hlavní testovaný scénář spočíval v umělém splnění podmínky alertu (triggernutí) a následném měření schopnosti samostatného Node.js WebSocket serveru spolehlivě adresovat a doručit událost alert:fired správnému klientovi bez jakéhokoliv dotazování (polling) ze strany klientského prohlížeče. Výsledky potvrdily plnou funkčnost nezávislé real-time infrastruktury.

6.6 Vyhodnocení latence

Pro objektivní posouzení vhodnosti řešení pro monitorování burzovních dat bylo provedeno orientační výkonnostní měření vnitřní latence systému. Pro tyto účely bylo na serverové straně využito nativního Node.js API perf_hooks (metoda performance.now()), které poskytuje vysoce přesné měření časových úseků. Měřen byl interval mezi přijetím datového tiku do aplikace a fyzickým odesláním události do real-time sítě [70].

Získaná referenční data demonstruje Tabulce 2. Výsledky nezahrnují vnější síťovou latenci internetu (ping na zařízení koncového uživatele) ani zpoždění externího poskytovatele dat.

Krok procesu	**Min [ms]**	**Průměr [ms]**	**Max [ms]**	**Zajišťující modul**
Vyhodnocení podmínky	2	4	12	Hlavní vyhodnocovací logika
Zápis události (AlertEvent)	45	78	134	Komunikace s MongoDB Atlas
Předání do WebSocket sítě	8	15	31	Server-to-Server komunikace
Celkový vnitřní čas zpracování	**55**	**97**	**177**	**Back-end infrastruktura**

Tabulka 2: Měření latence. Zdroj: vlastní.

Z naměřených průměrných hodnot jasně vyplývá, že drtivou většinu výpočetního času zabírá zápis historické události do cloudové databáze (I/O operace). Nicméně i tak je celková odezva zlomkem vteřiny plně dostačující pro požadavky navrhované webové aplikace.

6.7 Shrnutí výsledků testování

Realizovaná testovací fáze přesvědčivě prokázala funkčnost a stabilitu vyvinutého systému. Manuální testování potvrdilo vysokou použitelnost koncového rozhraní. Automatizované E2E skripty zajistily integritu aplikačních toků a integrační testy doložily bezpečnost a korektnost zpracování složitých stavových operací u burzovních alertů.

Měření latence a analýza WebSocket komunikace prokázaly, že architektonické rozdělení systému na bezstavový REST API backend a nezávislý perzistentní distribuční server plní svůj účel a dokáže poskytovat informace v téměř reálném čase. Ačkoliv prototyp nebyl podroben masivnímu zátěžovému stress-testingu v řádu tisíců souběžných uživatelů, ověřené mechanismy tvoří stabilní základ pro případné další škálování projektu.

7 Shrnutí a diskuse výsledků

7.1 Shrnutí hlavních výsledků

Cílem této diplomové práce bylo navrhnout, implementovat a vyhodnotit webovou aplikaci pro monitorování burzovních dat v reálném čase. Na základě provedené rešerše, architektonického návrhu a následného vývoje byl vytvořen plně funkční prototyp systému. Tento systém úspěšně propojuje moderní klientské rozhraní, dokumentově orientovanou NoSQL databázi, externí poskytovatele tržních dat, vyhodnocovací logiku alertů a asynchronní real-time komunikační vrstvu.

Ve výsledné podobě aplikace umožňuje uživateli vyhledávat akciové tituly, vytvářet personalizovaná portfolia, definovat vlastní limitní podmínky (alerty) nad sledovanými hodnotami a přijímat okamžitá upozornění při jejich splnění. Nad rámec samotného vyhodnocování podmínek bylo implementováno také perzistentní ukládání historie spuštěných událostí. Tento krok zásadně zvyšuje auditovatelnost a transparentnost celého řešení. Integrace samostatného WebSocket serveru navíc zajistila, že jsou tyto události doručovány aktivním klientům s minimální latencí přímo do prohlížeče.

Nejvýznamnějším výsledkem práce je skutečnost, že navržený koncept nezůstal pouze v teoretické rovině, ale byl přetaven v nasazený, provozuschopný webový systém. Funkčnost aplikace byla ověřena komplexním testováním, zahrnujícím automatizované end-to-end testy a měření vnitřní latence.

7.2 Zhodnocení splnění cílů práce

Hlavní cíl diplomové práce lze považovat za úspěšně splněný. Byl vytvořen funkční prototyp webové aplikace, který efektivně využívá NoSQL databázi, asimiluje data z externího streamovacího API a proaktivně upozorňuje uživatele na tržní výkyvy. Dosažené výsledky byly navíc podrobeny objektivní evaluaci.

Za splněné lze považovat rovněž všechny dílčí cíle. V úvodních fázích byla zpracována detailní rešerše zaměřená na real-time data, NoSQL databáze, streamovací architektury a bezpečnost. Z těchto poznatků plynule vzešel konceptuální návrh systému a datového modelu. Následná implementační fáze úspěšně pokryla vývoj

uživatelského rozhraní, robustní autentizace, správy watchlistů a asynchronní notifikační logiky. Součástí realizace byla i doplňková integrace generativní umělé inteligence pro tvorbu textových souhrnů.

Dílčí cíl týkající se provozního ověření systému byl splněn v rozsahu odpovídajícím prototypovému charakteru práce. Aplikace byla funkčně i integračně otestována, nicméně nebyly prováděny extrémní zátěžové testy simulující masivní komerční provoz s desetitisíci souběžnými připojeními. Systém je proto nutné vnímat jako stabilní technologický základ , který je plně připraven pro další iterativní rozvoj.

7.3 Přínosy navrženého řešení

Zásadním přínosem práce je úspěšná integrace několika technologických domén, které bývají často řešeny izolovaně. Vytvořená aplikace kombinuje reaktivní webové rozhraní, relačně nezávislou databázovou vrstvu, integraci na finanční trhy a multi-kanálové doručování zpráv. Systém tak neslouží pouze jako pasivní analytický dashboard, ale funguje jako proaktivní asistent uživatele.

Z inženýrského hlediska je velkým přínosem samotná zvolená architektura. Oddělení bezstavové webové aplikace od stavového WebSocket serveru prokazuje pochopení odlišných nároků na síťový provoz a garantuje lepší budoucí škálovatelnost. Vysoce přínosná se ukázala také volba databáze MongoDB, jejíž flexibilita umožnila přirozené mapování dynamických tržních entit a uživatelských logů.

Z uživatelského pohledu spočívá hlavní hodnota v automatizaci monitoringu. Uživatel je zbaven nutnosti neustále sledovat grafy, systém přebírá kognitivní zátěž a informuje ho pouze v momentech, které on sám definoval jako relevantní. Přítomnost ochranných mechanismů (cooldown) navíc garantuje, že uživatel nebude při vysoké volatilitě trhu zahlcen redundantními zprávami. Za metodický přínos lze rovněž považovat nasazení moderního frameworku Playwright pro automatizované testování, což dodává výslednému softwarovému dílu vysokou technickou kredibilitu.

7.4 Omezení navrženého řešení

Pro objektivní zhodnocení je nutné reflektovat i existující limity navrženého prototypu. Prvním a nejvýraznějším omezením je absolutní závislost na externím poskytovateli

tržních dat (Finnhub API). Přesnost, granularita a spolehlivost alertovacího mechanismu jsou přímo úměrné kvalitě a limitům tohoto poskytovatele. V případě výpadku služby třetí strany ztrácí aplikace dočasně svou monitorovací schopnost.

Další limitace pramení z prototypové povahy systému. Ačkoliv byla architektura navržena s ohledem na budoucí růst, systém nebyl optimalizován pro nasazení v prostředí High Frequency Trading (HFT), kde se latence měří v mikrosekundách. Aplikace rovněž postrádá robustní nástroje pro telemetrii a monitoring samotné infrastruktury, které by byly nezbytné pro komerční SaaS řešení.

Určitý prostor pro zlepšení vykazuje také notifikační vrstva. Přestože bezchybně doručuje e-maily a in-app upozornění, nepodporuje nativní mobilní push notifikace, které by doručitelnost zpráv mimo aktivní okno prohlížeče posunuly na úroveň nativních mobilních aplikací.

7.5 Možnosti dalšího rozvoje

Díky modulární architektuře poskytuje vytvořená aplikace široký prostor pro budoucí expanzi. Jedním z nejpřirozenějších kroků by bylo rozšíření vyhodnocovací logiky o komplexní složená pravidla. Portfolio podporovaných instrumentů by mohlo být doplněno o kryptoměny či komodity.

V oblasti notifikací by systém významně profitoval z integrace mobilních push notifikací nebo webhooků pro přímé napojení do komunikačních platforem typu Discord, Telegram či Slack. Zvýšila by se tím jistota, že zpráva zastihne uživatele i mimo e-mailovou schránku a webový prohlížeč.

Po stránce infrastruktury (DevOps) se jako další logický krok nabízí provedení podrobných zátěžových testů pro určení maximální propustnosti WebSocket serveru a případné nasazení technologie typu Redis pro efektivnější distribuci pub/sub zpráv mezi vícero uzly serveru. Analytický potenciál aplikace by pak mohl být dále umocněn širším nasazením nástrojů umělé inteligence pro prediktivní analýzu a generování vysoce personalizovaných tržních doporučení na základě historického chování uživatele.

8 Závěry a doporučení

Cílem této diplomové práce bylo navrhnout, implementovat a vyhodnotit webovou aplikaci pro monitorování burzovních dat. Tento cíl byl úspěšně naplněn vytvořením funkčního prototypu systému.

V teoretické části práce byla provedena rešerše zaměřená na problematiku zpracování dat v reálném čase, specifika finančních trhů, dokumentově orientované databáze, streamovací API, moderní architektury webových aplikací a bezpečnost. Tato východiska byla následně transformována do konceptuálního a architektonického návrhu aplikace.

Na základě tohoto návrhu byla naimplementována a do produkčního cloudového prostředí nasazena plně funkční aplikace. Systém uživatelům umožňuje vyhledávat akciové tituly, spravovat personalizovaná portfolia (watchlisty), definovat limitní podmínky nad sledovanými hodnotami a přijímat okamžitá upozornění při jejich splnění. Významnou součástí řešení je perzistentní logování historie spuštěných událostí a nasazení nezávislého WebSocket serveru pro doručování notifikací s minimální latencí přímo do prohlížeče. Výsledné dílo tak nepředstavuje pouze teoretický koncept, ale reálně provozovatelný softwarový prototyp.

Funkčnost a stabilita výsledného řešení byla exaktně ověřena v rámci praktické části. Testovací fáze pokryla hlavní uživatelské scénáře, přesnost vyhodnocovací logiky, spolehlivost real-time komunikace a základní výkonnostní odezvu. Kombinace manuálního testování, automatizovaných end-to-end testů ve frameworku Playwright, integračních testů a orientačního měření vnitřní latence potvrdila, že navržený systém plní svůj zamýšlený účel a zvolená architektura je pro danou doménu vysoce vhodná.

Vývoj prototypu rovněž prokázal, že při stavbě monitorovacích nástrojů nelze alerty redukovat na pouhé bezstavové podmínky. Pro zachování uživatelského komfortu bylo nezbytné implementovat ochranné časové mechanismy (cooldown) zabraňující notifikačnímu zahlcení při běžné oscilaci trhu. Jako klíčové architektonické rozhodnutí se ukázalo také striktní oddělení bezstavové webové aplikace (REST API) od stavové real-time vrstvy, což zásadně přispívá ke stabilitě a budoucí škálovatelnosti systému.

Navzdory prokázané funkčnosti vykazuje vytvořený prototyp i svá objektivní omezení. Mezi nejvýznamnější patří absolutní závislost na externím poskytovateli tržních dat a absence extrémních zátěžových testů simulujících masivní produkční provoz. Tyto skutečnosti však nijak nesnižují přínos vytvořeného řešení.

Za stěžejní přínos práce lze považovat vytvoření komplexního, prakticky využitelného softwarového díla, které úspěšně propojuje moderní webové technologie, NoSQL databáze a notifikační mechanismy do kohezního celku. Výsledná aplikace tvoří robustní technologický základ plně připravený pro budoucí expanzi.

9 Seznam použité literatury

1. *P. J. Sadalage a M. Fowler, NoSQL Distilled: A Brief Guide to the Emerging World of Polyglot Persistence. Boston: Addison-Wesley, 2012.*

2. *I. Holubová, J. Kosek, K. Minařík a D. Novák, Big Data a NoSQL databáze. Praha: Grada, 2015.*

3. *X. Li, Y. Zhang a H. Wang, „Optimizing Event Processing for Real-Time User Notification Systems," Proceedings of the VLDB Endowment, 2020.*

4. *M. Kleppmann, Designing Data-Intensive Applications. Sebastopol: O'Reilly Media, 2017.*

5. *J. Lee a S. Kim, „Time-series Data Management for Financial Market Analytics," Journal of Data Engineering, 2019.*

6. *M. Stonebraker, D. J. Abadi, A. Batkin, X. Chen, M. Cherniack, M. Ferreira, E. Lau, A. Lin, S. Madden, E. J. O'Neil, P. E. O'Neil, A. Rasin, N. Tran a S. B. Zdonik, „C-Store: A Column-oriented DBMS," in Very Large Data Bases Conference, 2005.*

7. *Y. Ma, J. Zhou, T. Chantem, R. P. Dick, S. Wang a X. S. Hu, „Improving Reliability of Soft Real-Time Embedded Systems on Integrated CPU and GPU Platforms," IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, roč. 39, s. 2218–2229, 2020.*

8. *A. Guasque, H. Tohidi, P. Balbastre, J. M. Aceituno, J. Simó a A. Crespo, „Integer Programming Techniques for Static Scheduling of Hard Real-Time Systems," IEEE Access, roč. 8, s. 170389–170403, 2020.*

9. *H. Cao, X. Yang a R. Deng, „Ontology-Based Holonic Event-Driven Architecture for Autonomous Networked Manufacturing Systems," IEEE Transactions on Automation Science and Engineering, roč. 18, s. 205–215, 2021.*

10. *A. D. Rocha, N. Freitas, D. Alemão, M. Guedes, R. Martins a J. Barata, „Event-Driven Interoperable Manufacturing Ecosystem for Energy Consumption Monitoring," Energies, 2021.*

11. *S. Raubitzek a T. Neubauer, „An Exploratory Study on the Complexity and Machine Learning Predictability of Stock Market Data," Entropy, roč. 24, č. 3, čl. 332, 2022, doi: 10.3390/e24030332.*

12. *M. Baldauf a J. Mollner, „High-Frequency Trading and Market Performance," Journal of Finance, roč. 75, s. 1495–1526, 2020.*

13. *X. Gou, Y. Zhang, Z. Hu, L. He, K. Wang, X. Liu, T. Yang, Y. Wang a B. Cui, „A Sketch Framework for Approximate Data Stream Processing in Sliding Windows," IEEE Transactions on Knowledge and Data Engineering, roč. 35, s. 4411–4424, 2023.*

14. *M. Muñoz a C. Riveros, „Streaming enumeration on nested documents," ACM Transactions on Database Systems, roč. 49, č. 4, čl. 15, s. 1–39, 2024.*

15. *A. Madhavan, „Market microstructure: A survey," Journal of Financial Markets, roč. 3, č. 3, s. 205–258, 2000, doi: 10.1016/S1386-4181(00)00007-0.*

16. *D. Ardia, E. Guidotti a T. A. Kroencke, „Efficient estimation of bid–ask spreads from open, high, low, and close prices," Journal of Financial Economics, 2024, doi: 10.1016/j.jfineco.2024.103916.*

17. *G. Dionne, P. Duchesne a M. Pacurar, „Intraday Value at Risk (IVaR) using tick-by-tick data with application to the Toronto Stock Exchange," Journal of Empirical Finance, 2009, doi: 10.1016/j.jempfin.2009.05.005.*

18. *D. S. McCrickard, C. M. Chewar, J. Somervell a A. Ndiwalana, „A model for notification systems evaluation—assessing user goals for multitasking activity," ACM Transactions on Computer-Human Interaction, roč. 10, č. 4, s. 312–338, 2003, doi: 10.1145/966930.966933.*

19. *M. T. Vo, „Limit orders and the intraday behavior of market liquidity: Evidence from the Toronto stock exchange," Global Finance Journal, 2007, doi: 10.1016/j.gfj.2006.06.012.*

20. *G. Cugola a A. Margara, „Processing flows of information: From data stream to complex event processing," ACM Computing Surveys, roč. 44, čl. 15:1–15:62, 2012.*

21. *J. Guo, J. Peng, H. Yuan et al., „HXPY: A High-Performance Data Processing Package for Financial Time-Series Data," Journal of Computer Science and Technology, roč. 38, s. 3–24, 2023, doi: 10.1007/s11390-023-2879-5.*

22. *D. Easley, M. L. de Prado a M. O'Hara, „Discerning information from trade data," Journal of Financial Economics, 2016, doi: 10.1016/j.jfineco.2016.01.018.*

23. *G. Cugola a A. Margara, „Processing Flows of Information: From Data Stream to Complex Event Processing," ACM Computing Surveys, roč. 44, č. 3, čl. 15, 2012, doi: 10.1145/2110486.2110493.*

24. *L. Golab, „Querying sliding windows over online data streams," in Advances in Database Technology – EDBT 2004, 2004, doi: 10.1007/978-3-540-30192-9_1.*

25. *A. Almeida, S. Brás, S. Sargento a F. C. Pinto, „Time series big data: a survey on data stream frameworks, analysis and algorithms," Journal of Big Data, 2023, doi: 10.1186/s40537-023-00760-1.*

26. *S. D. Dhasade, B. K. Birla a C. Kalyan, „NoSQL Database," Big Data, 2021.*

27. *J. Chen a W. Lee, „An Introduction of NoSQL Databases Based on Their Categories and Application Industries," Algorithms, roč. 12, čl. 106, 2019.*

28. *P. J. Sadalage a M. Fowler, NoSQL Distilled: A Brief Guide to the Emerging World of Polyglot Persistence. Boston: Addison-Wesley, 2012. ISBN 9780133036121.*

29. *MongoDB, Inc., „What is MongoDB? – Database Manual." Viděno: 20. duben 2026. [Online]. Dostupné z: https://www.mongodb.com/docs/manual/introduction/*

30. *A. Makris, K. Tserpes, G. Spiliopoulos et al., „MongoDB Vs PostgreSQL: A comparative study on performance aspects," Geoinformatica, roč. 25, s. 243–268, 2021, doi: 10.1007/s10707-020-00407-w.*

31. *J. Grennan a R. Michaely, „FinTechs and the Market for Financial Analysis," Journal of Financial and Quantitative Analysis, roč. 56, s. 1877–1907, 2020.*

32. *Z. Hu, Y. Zhao a M. Khushi, „A Survey of Forex and Stock Price Prediction Using Deep Learning," Applied System Innovation, roč. 4, č. 1, čl. 9, 2021, doi: 10.3390/asi4010009.*

33. *C. Antal, T. Cioara, I. Anghel, M. Antal a I. Salomie, „Distributed Ledger Technology Review and Decentralized Applications Development Guidelines," Future Internet, roč. 13, č. 3, čl. 62, 2021, doi: 10.3390/fi13030062.*

34. *J. Herzog, „Software Architecture in Practice Third Edition Written by Len Bass, Paul Clements, Rick Kazman," ACM SIGSOFT Software Engineering Notes, roč. 40, s. 51–52, 2015.*

35. *G. Q. Huang a K. Mak, „Issues in the development and implementation of web applications for product design and manufacture," International Journal of Computer Integrated Manufacturing, roč. 14, s. 125–135, 2001.*

36. *V. Gavrila, L. Bajenaru a C. Dobre, „Modern Single Page Application Architecture: A Case Study," Studies in Informatics and Control, 2019.*

37. *I. Fette a A. Melnikov, The WebSocket Protocol, RFC 6455, 2011. Viděno: 13. červenec 2025. [Online]. Dostupné z: https://doi.org/10.17487/rfc6455*

38. *M. Grinberg, Flask Web Development. Sebastopol: O'Reilly Media, 2014.*

39. *S. A. Tien, J. S. DeArmon, H. Bateman, D. Freer a P. Somersall, „Developing a real-time monitoring and alerting capability for traffic flow management," in 2016 IEEE/AIAA 35th Digital Avionics Systems Conference (DASC), s. 1–10, 2016.*

40. A. Reddy Kommera, „The Power of Event-Driven Architecture: Enabling RealTime Systems and Scalable Solutions," Turkish Journal of Computer and Mathematics Education (TURCOMAT), 2020.

41. S. Newman, Building Microservices. Beijing: O'Reilly, 2015.

42. L. Li, W. Chou, W. Zhou a M. Luo, „Design Patterns and Extensibility of REST API for Networking Applications," IEEE Transactions on Network and Service Management, roč. 13, s. 154–167, 2016.

43. D. A. Douglass, W. A. Chisholm, G. A. Davidson, I. S. Grant, K. E. Lindsey, M. Lancaster, D. Lawry, T. McCarthy, C. D. Nascimento, M. S. Pasha, J. Reding, T. Seppa, J. Tóth a P. Waltz, „Real-Time Overhead Transmission-Line Monitoring for Dynamic Rating," IEEE Transactions on Power Delivery, roč. 31, s. 921–927, 2016.

44. L. Burgueño, J. Boubeta-Puig a A. Vallecillo, „Formalizing Complex Event Processing Systems in Maude," IEEE Access, roč. 6, s. 23222–23241, 2018.

45. S. Patil, M. Rao, L. Misal, D. Phaldesai a K. Shivsharan, „A Review of the OWASP Top 10 Web Application Security Risks and Best Practices for Mitigating These Risks," in 2023 7th International Conference on Computing, Communication, Control and Automation (ICCUBEA), s. 1–8, 2023.

46. A. Rahmatulloh, A. N. Rachman a F. Anwar, „Implementasi Web Push Notification pada Sistem Informasi Manajemen Arsip Menggunakan PUSHJS," Jurnal Teknologi Informasi dan Ilmu Komputer, 2019.

47. Google, „Firebase Cloud Messaging documentation." Viděno: 13. červenec 2025. [Online]. Dostupné z: https://firebase.google.com/docs/cloud-messaging

48. Apple, „Apple Push Notification Service (APNs)." Viděno: 13. červenec 2025. [Online]. Dostupné z: https://developer.apple.com/notifications/

49. J. Kreps, „Kafka: A Distributed Messaging System for Log Processing," 2011.

50. TradingView, „Pine Script® documentation." Viděno: 13. červenec 2025. [Online]. Dostupné z: https://www.tradingview.com/pine-script-docs/welcome/

51. MetaQuotes, „MetaTrader 5 trading platform." Viděno: 23. březen 2026. [Online]. Dostupné z: https://www.metatrader5.com/en/trading-platform

52. Yahoo, „Yahoo Finance." Viděno: 23. březen 2026. [Online]. Dostupné z: https://finance.yahoo.com/

53. QuantConnect, „QuantConnect documentation v2." Viděno: 23. březen 2026. [Online]. Dostupné z: https://www.quantconnect.com/docs/v2/

54. *Backtrader, „Backtrader documentation." Viděno: 23. březen 2026. [Online]. Dostupné z: https://www.backtrader.com/docu/*

55. *Freqtrade, „Freqtrade." GitHub repozitář. Viděno: 23. březen 2026. [Online]. Dostupné z: https://github.com/freqtrade/freqtrade*

56. *OWASP Foundation, „OWASP Top 10:2025." Viděno: 23. březen 2026. [Online]. Dostupné z: https://owasp.org/Top10/2025/*

57. *National Institute of Standards and Technology, Digital Identity Guidelines: Authentication and Lifecycle Management, NIST SP 800-63B. Viděno: 23. březen 2026. [Online]. Dostupné z: https://pages.nist.gov/800-63-4/sp800-63b.html*

58. *OWASP Foundation, „OWASP Application Security Verification Standard (ASVS)." Viděno: 23. březen 2026. [Online]. Dostupné z: https://owasp.org/www-project-application-security-verification-standard/*

59. *Vercel, „Next.js Documentation." Viděno: 20. duben 2026. [Online]. Dostupné z: https://nextjs.org/docs*

60. *Meta Platforms, Inc., „React Documentation." Viděno: 20. duben 2026. [Online]. Dostupné z: https://react.dev/*

61. *Better Auth, „Better Auth Documentation." Viděno: 20. duben 2026. [Online]. Dostupné z: https://www.better-auth.com/docs*

62. *Inngest, „Inngest Documentation." Viděno: 20. duben 2026. [Online]. Dostupné z: https://www.inngest.com/docs*

63. *Finnhub, „API Documentation." Viděno: 20. duben 2026. [Online]. Dostupné z: https://finnhub.io/docs/api*

64. *Socket.IO, „Socket.IO Documentation." Viděno: 20. duben 2026. [Online]. Dostupné z: https://socket.io/docs/v4/*

65. *Mongoose, „Mongoose Documentation." Viděno: 20. duben 2026. [Online]. Dostupné z: https://mongoosejs.com/docs/*

66. *Nodemailer, „Nodemailer Documentation." Viděno: 20. duben 2026. [Online]. Dostupné z: https://nodemailer.com/*

67. *Vercel, „Vercel Documentation." Viděno: 20. duben 2026. [Online]. Dostupné z: https://vercel.com/docs*

68. *Railway, „Railway Documentation – Deployments." Viděno: 20. duben 2026. [Online]. Dostupné z: https://docs.railway.com/deployments*

69. *Microsoft, „Playwright Documentation." Viděno: 20. duben 2026. [Online]. Dostupné z: https://playwright.dev/docs/intro*

70. Node.js, „Performance measurement APIs." Viděno: 20. duben 2026. [Online]. Dostupné z: https://nodejs.org/api/perf_hooks.html

10 Přílohy

GitHub repozitář:

- https://github.com/Sipisak/stock-tracker-app

Nasazená aplikace:

- https://stock-tracker-app-snowy.vercel.app/

UNIVERZITA HRADEC KRÁLOVÉ

Fakulta informatiky a managementu

Akademický rok: 2024/2025

ZADÁNÍ DIPLOMOVÉ PRÁCE

(projektu, uměleckého díla, uměleckého výkonu)

Jméno a příjmení:	**Bc. Marek Šípek**
Osobní číslo:	**I2300700**
Studijní program:	**N0613A140042 Aplikovaná informatika**
Téma práce:	**Webová aplikace pro real-time monitorování burzovních dat s notifikacemi**
Zadávající katedra:	**Katedra informatiky a kvantitativních metod**

Zásady pro vypracování

Cílem práce je návrh a implementace webové aplikace s využitím NoSQL databáze. Aplikace bude sbírat a zpracovávat burzovní data, na jejichž základě při splněných předem definovaných podmínek budou notifikováni vhodní uživatelé.
Struktura práce:

- Úvod
- Cíl a metodika práce
- Rešerše východisek
- Návrh aplikace
- Implementace aplikace
- Diskuse výsledků

Rozsah pracovní zprávy:	**50-60 stran**
Rozsah grafických prací:	
Forma zpracování diplomové práce:	**tištěná/elektronická**

Seznam doporučené literatury:

HOLUBOVÁ, Irena, KOSEK, Jiří, MINAŘÍK, Karel, NOVÁK, David. *Big Data a NoSQL databáze*. Grada, 2015. ISBN 978-80-247-5.
Connoly, Thomas M., Begg Carolyn E. *Database systems. A Practical Approach to Design, Implementation and Management*. Addison Wesley, 2010.
SADALAGE, Pramod J. a FOWLER, Martin. *NoSQL distilled: a brief guide to the emerging world of polyglot persistence*. Addison-Wesley, 2013. ISBN 978-0321826626.

SADALAGE, Pramod J. a FOWLER, Martin. *NoSQL distilled: a brief guide to the emerging world of polyglot persistence*. Addison-Wesley, 2013. ISBN 978-0321826626.

Vedoucí diplomové práce:	**prof. RNDr. Petra Poulová, Ph.D.** Katedra informatiky a kvantitativních metod

Datum zadání diplomové práce: **3. října 2024**

L.S.

prof. Ing. Mgr. Petra Marešová, Ph.D., MBA v.r.
děkanka

Mgr. Jana Medková, Ph.D. v.r.
vedoucí katedry

V Hradci Králové dne 3. října 2024