Архитектурен дизајн

Апликација за следење и анализа на берзата во Македонија

Јована Анѓелковска 221040
Загорка Аневска 221196
Асија Зукорлиќ 221210

1. ВОВЕД

1.1 Цел

Целта на овој документ е да се прикажат различните видови на архитектури кои ќе се користат при креирање на апликација за следење и анализа на берзата во Македонија. Овие архитектури вклучуваат: концептуална, извршна и имплементациона архитектура. Во овој документ, исто така, ќе се прикаже како подархитектурните компоненти, како што се: цевки, филтри, слоевита веб архитектура, дистрибуирана архитектура со микросервиси и контејнеризација, ќе придонесат за дизајнирање и изградба на оптимален систем.

1.1.2 Главни цели:

а) да се дефинираат структурата и компонентите на апликацијата и како тие ќе влијаат на главните функционалности;
б) опис на технологиите, компонентите и имплементациските стратегии за дизајнирање на стабилна и функционална апликација;
в) исполнување на барањата, како што се: real-time известувања и ажурирања, систем за безбедност и целосна функционалност.

1.2 Функционалности

Апликацијата е креирана со цел следење и анализа на македонската берза. Ќе биде лесно достапна за користење како на веб, така и на мобилните платформи. Преку употреба на различни архитектури и подархитектурни компоненти, апликацијата ќе им обезбеди на сите корисници безбеден и лесен пристап до системот и сите негови функционалности.

1.2.1 Главни функционалности:

а) real-time известувања за цените на акциите, нивниот пораст и/или пад;
б) пристап кон историските податоци и можност за нивно користење во анализи;
в) новитети и дневни информации за берзата во Македонија;
г) лесна и брза селекција и пребарување на одредени акции по име или шифра;
д) достапност на апликацијата како на веб, така и на мобилните платформи.

1.3 Примена

1.3.1 Скалабилна и лесно одржлива хибридна архитектура:

Хибридната архитектура овозможува полесно собирање и користење на големо количество на real-time податоци и информации. Овој пристап исто така дозволува

системот полесно да ги разделува податоците во подгрупи, што е од клучно значење за одржувањето на апликацијата и за можностите за идни надградби.

1.3.2 Голема побарувачка и толеранција на грешки:

Со оглед на големата побарувачка на овој пазар, но и можноста за системски грешки која е последица на оваа побарувачка, апликацијата треба да биде високо достапна и да ги минимизира шансите за пад на системот. Повеќе сервери, како и back-up сервери кои ќе работат истовремено и ќе помагаат да не дојде до оптоварување на само еден сервер. За толеранција на грешки, системот ќе примени кеширање и автоматско одржување, односно доколку дојде до несакана грешка, системот автоматски одговара на таа грешка со повторно извршување на наредбата која претходно не успеала да се изврши.

1.3.3 Независно распоредување на основните функционалности со помош на микроуслуги:

Основните функции кои ќе бидат лесно достапни за сите корисници ќе се извршуваат и користат како независни услуги. Оваа архитектурна стратегија овозможува поедноставување на одржувањето и развојот на одредени делови од системот, како што се: real-time информации, новитети и дневни информации за берзата, пребарување на одредени акции.

2. ПРЕГЛЕД НА СИСТЕМОТ

2.1 Основни компоненти

Во овој дел ќе се прикажат основните компоненти на системот кои ќе го формираат целокупниот процес на работа. Овие компоненти се дизајнирани за да обезбедат ефикасна комуникација помеѓу корисниците и системот, при тоа овозвожувајќи безбедни и сигурни real-time функционалности.

1. Frontend:
Целта на frontend компонентата е да обезбеди интерактивен интерфејс преку кој корисниците може да комуницираат со системот. Апликацијата е изработена со React, што овозможува интеграција, како со веб, така и со мобилни платформи. Овој пристап гарантира лесно и флексибилно користење на апликацијата од страна на сите корисници.

2. Backend:
Backend компонентата е изградена на Spring Boot, со примена на микроуслуги, што овозможува системот да биде скалабилен и модуларен. Овие микроуслуги ќе ги процесираат сите бизнис-логики, анализираат real-time податоци и обезбедуваат оперативна стабилност.

3. Складирање на податоци:
Складирањето на податоци ќе се обезбеди преку PostgreSQL за структурирани податоци, додека за неструктурирани податоци ќе се користи MongoDB. Овој комбиниран пристап овозможува поефикасно управување со различни типови податоци, што е клучно за динамичниот карактер на апликацијата.

2.2 Клучни функционалности

1. Автентикација и управување со корисници:
Корисниците ќе можат да се регистрираат безбедно преку користење на повеќефакторска автентикација. Дополнително, системот ќе ги групира корисниците во две главни категории: обични корисници и администратори, што ќе овозможи различни нивоа на пристап.

2. Анализа на податоците:
Системот ќе обезбеди анализа на податоци која ќе овозможи корисниците да генерираат извештаи засновани на личните податоци и сметки, како и да ги проценат ризиците и можните добивки или загуби.

3. Real-time известувања преку WebSocket:
Real-time известувањата ќе обезбедат важни информации за промените во корисничките сметки и настаните на берзата, што е особено важно за корисниците кои се активни во тргување со акции.

4. Скалабилни процесни цевки за податоци:
За да се обезбеди непречено функционирање на системот дури и во периоди на висока побарувачка, ќе се користат скалабилни процесни цевки за податоци. Овие процеси ќе се управуваат преку Kubernetes и ќе бидат оптимизирани за обработка на големи количини податоци во real-time услови.

2.3 Ограничувања:

1. Фиксен технолошки стек:
Технолошкиот стек е избран со цел да обезбеди стабилност и континуитет на системот. React, Spring Boot и PostgreSQL се докажани технологии кои овозможуваат висока достапност и ефикасност во овој дел за дизајн и изградба на системи и апликации.

2. Интеграција со Kubernetes:
Интеграцијата со Kubernetes ќе обезбеди автоматско распределување на услугите, што е клучно за постигнување висока скалабилност и управување со големите количини на податоци во системот.

3. Real-time перформанси:
Со оглед на тоа што системот работи со real-time податоци, мора да се внимава и размислува за минималните доцнења. Механизмите за кеширање и оптимизација на базата на податоци ќе бидат клучни за обезбедување на константни перформанси.

4. Регулаторна усогласеност:
Сите функционалности на системот ќе бидат во согласност со регулативите за финансиско работење во Македонија, што обезбедува безбедност и правна сигурност за корисниците.

3. АРХИТЕКТУРА НА СИСТЕМОТ

Во овој дел ќе покажеме како различните видови архитектури влијаат на организацијата на системот и како функционираат сите негови компоненти во real-time.

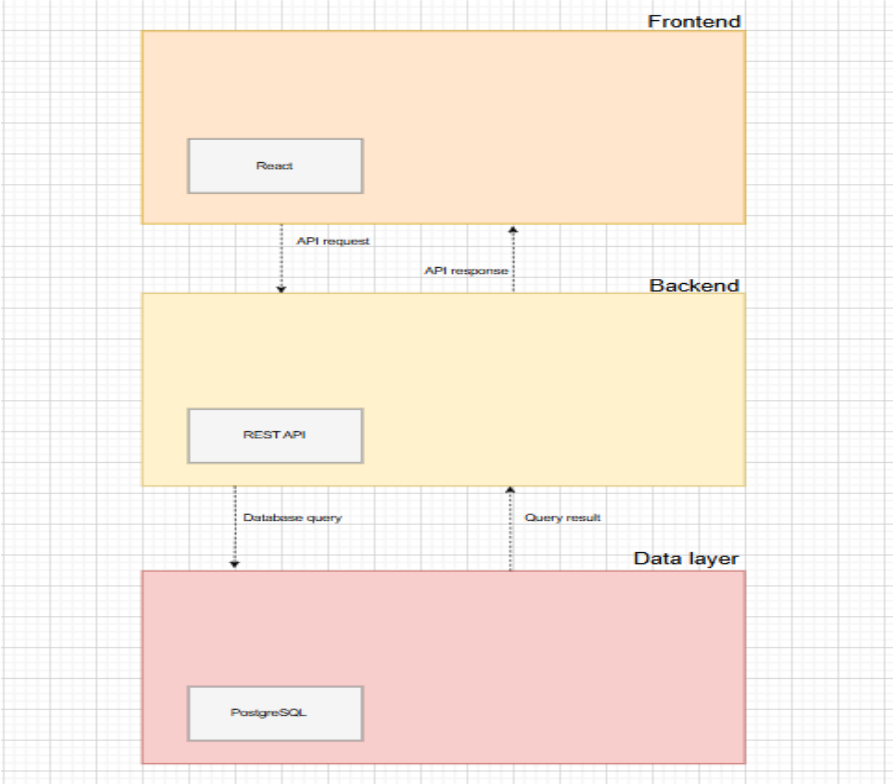
3.1 Концептуална архитектура

Концептуалната архитектура на системот се состои од три основни слоеви: презентациски слој, слој за деловна логика и податочен слој. Секој од овие слоеви има своја улога во правилното функционирање на апликацијата.

1. Презентациски слој:
Овој слој е развиен со React и обезбедува динамичен и интерактивен кориснички интерфејс, преку кој корисниците можат да пристапат до сите потребни податоци.

2. Слој за деловна логика:
Овој слој претставува збир на сите микросервиси и секој микросервис е одговорен за специфична микроуслуга. Комуникацијата помеѓу овие услуги се изведува преку REST API, RabbitMQ или Kafka.

3. Податочен слој:
Во овој слој се чуваат базите на податоци за различните структурни податоци. Такви бази се: PostgreSQL и MongoDB.

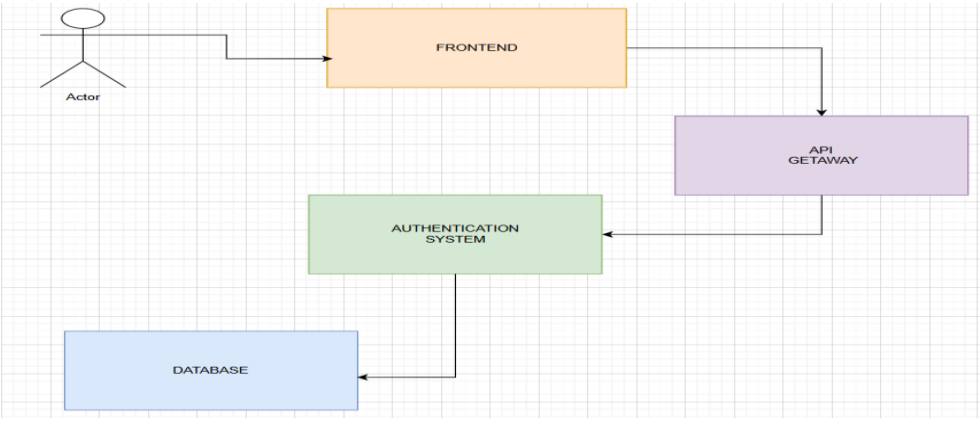


3.2 Извршна архитектура

Извршната архитектура објаснува како различните компоненти на системот функционираат во реално време. Следните пример сценарија ќе илустрираат како изгледаат овие процеси.

Пример сценарио 1: корисникот се најавува во системот

Кога корисникот ги внесува своите податоци за најава, корисничкиот интерфејс (frontend) испраќа HTTP барање преку API Gateway до автентикацискиот систем. Податоците за најава се проверуваат во PostgreSQL базата, а при успешна автентикација се генерира JWT токен кој му овозможува на корисникот пристап до неговата контролна табла.



Пример сценарио 2: real-time известувања
Сервисот за известувања воспоставува WebSocket врска со корисничкиот интерфејс. При додавање нов производ, сервисот за производи испраќа порака преку RabbitMQ до сервисот за известувања. Овој сервис ја испраќа пораката до сите поврзани клиенти преку WebSocket.

3.3 Имплементациона архитектура

Имплементационата архитектура ги опишува технологиите кои се користат за развој, контенеризација и одржување на системот.

1. Frontend:
Се користи React за развој на корисничкиот интерфејс и Axios за воспоставување на API повици.

2. Микросервиси:
Се користат Spring Boot, Hibernate и RabbitMQ за развој на backend логиката, за интеракции помеѓу различните бази на податоци и за меѓусервисна комуникација, соодветно.

3. Бази на податоци:
Структурираните податоци, како што се податоците за корисниците и трансакциите, се складираат во PostgreSQL базата на податоци, а додека пак неструктурираните податоци, како што се известувањата, аналитичките информации и корисничките сметки, се складираат во MongoDB базата.

4. Контенеризација и распределба:
Docker овозможува создавање на контејнери за секоја компонента, а Kubernetes се грижи за распределба и скалабилност на истите.

5. Мониторинг и регистрирање:
Prometheus собира метрики за мониторинг, додека Grafana се користи за визуализација и поставување на аларми кои се вклучуваат при грешки и проблеми.