Київський національний університет імені Тараса Шевченка
Факультет інформаційних технологій
Кафедра програмних систем і технологій
УДК 004.75;004.042 *На правах рукопису*

ВИПУСКНА КВАЛІФІКАЦІЙНА БАКАЛАВРСЬКА РОБОТА

Тема: “Віртуальний кейс-тренер роботи в інвестиційних додатках ”

Спеціальність – 121 “Інженерія програмного забезпечення”

ПОЯСНЮВАЛЬНА ЗАПИСКА

БР.ІПЗ – 30.00.00.000

Студент
ІПЗ-43 _____ /**Діана ШЕВЧЕНКО**/

Науковий керівник
асист. _____ /**Кирило КАДОМСЬКИЙ**/

Консультант
з питань нормоконтролю
фахівець _____ /**Тамара ЧАПОВСЬКА**/

Допускається до захисту
Завідувач кафедри
д.т.н., проф. _____/**Олексій БИЧКОВ**/

Київ – 2021

Київський національний університет імені Тараса Шевченка

Факультет інформаційних технологій

Кафедра програмних систем і технологій

Освітньо-кваліфікаційний рівень бакалавр

Спеціальність 121 “Інженерія програмного забезпечення”

ЗАТВЕРДЖЕНО

Зав.кафедри програмних систем і технологій

_____(Олексій БИЧКОВ)

(підпис) (прізвище та ініціали)

ЗАВДАННЯ

НА ВИПУСКНУ КВАЛІФІКАЦІЙНУ БАКАЛАВРСЬКУ РОБОТУ СТУДЕНТУ

Шевченко Діані Олександрівні

(прізвище, ім’я, по-батькові)

1. **Тема випускної кваліфікаційної бакалаврської роботи** “Віртуальний кейс-тренер роботи в інвестиційних додатках”, керівник проекту (роботи) Кадомський Кирило Костянтинович, асистент, затверджена наказом вищого навчального закладу від “11” листопада 2021 №
2. **Строк подання студентом роботи** 18 лютого 2021 р.
3. **Вихідні дані до роботи:** публікації, присвячені розгляду фінансових рішень у сфері фінансової самоосвіти, інвестицій. Форма діалогу: мобільний додаток. Перелік використовуваних програмних засобів: ОС Windows 10, інтегроване середовище розробки WebStorm; React Native, Golang, Mongodb, MarketStack.
4. **Зміст пояснювальної записки (перелік питань, що їх належить розробити)**

   1. Огляд існуючих методів та постановка задачі.

2. Аналіз аналогів.

3. Огляд проектованих результатів роботи із застосунком.

4. Проектування системи віртуального кейс-тренера.

5. Програмна реалізація проекту.

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень)

1. Діаграма прецедентів (рис. 2.1.1, ст. 19)

6. Консультанти з роботи із зазначенням розділів роботи, що їх стосуються

Розділ	Консультант	Підпис, дата	
		Завдання видав	Завдання прийняв
Основна частина	асист., Кирило Кадомський		

7. **Дата видачі завдання** 11 листопада 2020 р

Керівник (Кирило КАДОМСЬКИЙ)

_____ _____

(підпис) (розшифровка підпису)

Завдання (Діана ШЕВЧЕНКО)

прийняв до виконання _____ _____

(підпис) (розшифровка підпису)

КАЛЕНДАРНИЙ ПЛАН

№ п/п	Назви етапів бакалаврської роботи	Строк виконання етапів роботи	Примітка
1.	Отримання завдання на випускну кваліфікаційну роботу.	20.10.2020	виконано
2.	Аналіз завдання, пошук і вивчення відповідної літератури.	09.11.2020-30. 11.2020	виконано
3.	Формування постановки задачі.	09.12.2020	виконано
4.	Дослідження й аналіз наявних програмних рішень фінансової самоосвіти.	14.12.2020-04. 01.2021	виконано
5.	Постановка задачі	13.01.2021- 17.01.2021	виконано
6.	Проектування системи.	21.01.2021-01. 02.2021	виконано
7.	Розробка додатку.	02.02.2021-04. 04.2021	виконано
9.	Огляд розробки та створення документації	07.04.2021-05. 04.2021	виконано
10.	Оформлення пояснювальної записки та текстових матеріалів.	09.04.2021-20. 05.2021	виконано
11.	Оформлення презентаційних матеріалів.	21.05.2021-30. 05.2021	виконано

Студент – бакалавр _____ (Діана ШЕВЧЕНКО)

(підпис) (розшифровка підпису)

Керівник роботи _____(Кирило КАДОМСЬКИЙ)

(підпис) (розшифровка підпису)

АНОТАЦІЯ

Випускна кваліфікаційна бакалаврська робота: 52 с., 12 рис., 2 додат., 7 джерел.

Тема: Віртуальний кейс-тренер роботи в інвестиційних додатках.

Об’єкт дослідження: освітні програмні продукти в сфері фінансів.

Мета роботи: дослідження ринку освітніх рішень в сфері фінансової самоосвіти, проектування та реалізація програмного забезпечення типу віртуального кейс тренера інвестування

Предмет дослідження: рішення в сфері фінансової самоосвіти, віртуальні біржі.

Результати: реалізовано додаток віртуальний кейс-тренер із можливістю тренування навичок інвестицій у віртуальній біржі.

Висновок: у результаті спроектовано та реалізовано багатомодульний додаток на тематику фінансової самоосвіти для набуття практичних навичок інвестування на фінансового менеджменту.

ВІРТУАЛЬНА БІРЖА. ІНВЕСТУВАННЯ. ВІРТУАЛЬНИЙ КЕЙС-ТРЕНЕР. MATCHING ENGINE. GOLANG.

ЗМІСТ

ВСТУП

Застосунки, що знайомлять громадян із можливостями та шляхами покращення власного фінансового становища, стають все більш доступними до широкого загалу. Таке програмне забезпечення, будучи успішно спроектованим, може суттєво вплинути на життя користувачів.

Актуальність дослідження - відсутність гарного освітнього базису у сфері особистих фінансів негативно впливає на стан громадян, та зменшує шляхи розвитку як індивідуально, так і в контексті країни в цілому. Наростаючі темпи діджиталізації багатьох сфер життя громадян зумовлюють потребу у проектуванні і реалізації цифрової системи вищої освіти. Відсутність аналогів чи проектів даного спрямування є основною проблемою даної роботи, що визначає її актуальність у соціальному та інформаційному аспектах.

Об’єкт дослідження - спеціалізоване програмне забезпечення у сфері особистої фінансової освіти.

Предмет дослідження - застосування сучасних технологій розробки програмного забезпечення, архітектурних рішень для розробки програмного забезпечення.

Мета - проектування та розробка програмного забезпечення типу кейс-тренер для отримання навичок інвестування.

Практичне значення - спроектоване програмне забезпечення можна використати для покращення навичок менеджменту особистих фінансів, покращити фінансову грамотність громадян.

Додатково, при подальшій підтримці програмного забезпечення та розробці додаткового функціоналу, даний додаток може служити повноцінною навчальною платформою у сфері фінансової освіти.

РОЗДІЛ 1

ОГЛЯД ІСНУЮЧИХ МЕТОДІВ ТА ПОСТАНОВКА ЗАДАЧІ

1.1. Обґрунтування актуальності теми

З квітня 2020 року спостерігається сплеск зацікавленості людей по всьому світу методами заощадження, інвестування своїх грошей. Поштовхом до цього стала коронавірусна криза, що вплинула на мільйони робочих місць.

Україна в свою чергу не стала винятком, проте на відміну від десятків інших країн, фінансова грамотність серед українців є достатньо низькою, а іноді і взагалі невідомим, малопоширеним явищем. Згідно з індексом фінансової грамотності OECP, Україна зайняла останнє 30-те місце в переліку опитуваних країн, В Україні індекс склав 11.6 пунктів з 21[1]. Інфографіку індексу наведено на Рис.1.1.

Щодо низького рівня знань українців у цій сфері, є декілька пояснень:

- у нас не прийнято говорити про гроші чи хотіти навчитись базовим навичкам їх володіння
- фінансовим установам не довіряють
- мати/користуватись кредитним лімітом соромно
- слова “інвестиції” та подібні викликають дискомфорт та асоціюються з пірамідами МММ

Важливо, розуміти, що значною мірою такий менталітет був сформований людьми, які пережили розпад СРСР разом із втратою заощаджень на “книжці”, а потім були свідками революцій, кризи 2008 року, початку війни в 2013 році, націоналізації ПриватБанку в 2016 році. Всі ці події супроводжувались відсутністю матеріалів чи ресурсів по фінансовій грамотності, які могли б покращити ситуацію.

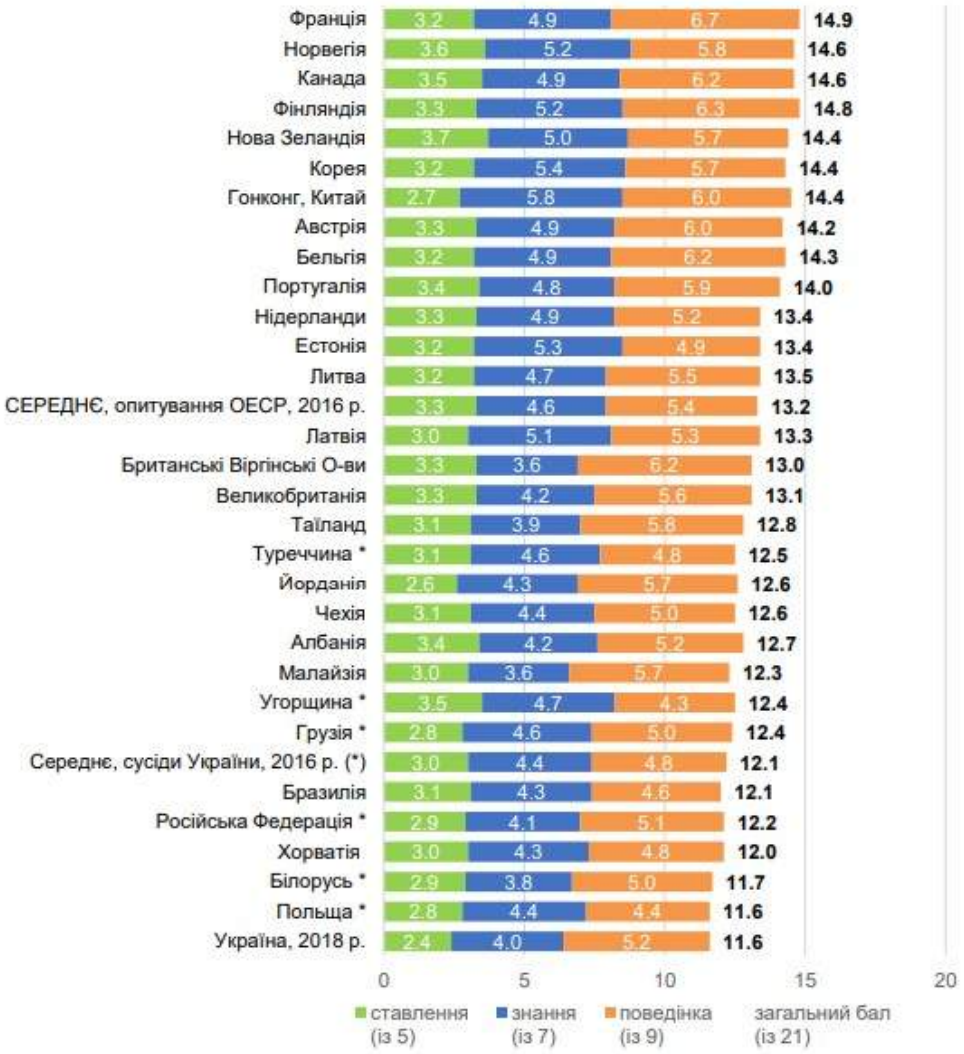


Рис.1.1 Індекс фінансової грамотності в різних країнах / Інфографіка USAID[1]

За результатами загальнонаціонального опитування наприкінці 2018 року[2]:

- близько 20% громадян узагалі не роблять заощаджень;
- з тих, хто заощаджує, кожен другий тримає заощадження вдома;
- В результаті чого, українці з низькими доходами та низьким рівнем фінансової освіти демонструють значно нижчі результати та проявляли значно менший інтерес до фінансової грамотності

[1] USAID (United States Agency for International Development) - Агентство США з міжнародного розвитку

У більшості українців досить пасивне ставлення до створення власного добробуту, підкріплене філософією патерналізму та поглядами на світ, сформованими крізь призму соціалістичних ідей. Немає почуття повної відповідальності за свій фінансовий стан, натомість є ціла купа претензій та вимог до державних і фінансових інститутів[2].

Користуючись статистикою із Google Trends, можна відмітити, що зацікавленість українців в тематиці особистих фінансів росте дуже повільно. Це можна прослідкувати за допомогою лінії тренду чотирьох запитів в Google, що наведені на скріншоті на Рис.1.2, а саме - “инвестирование”, “инвестиции”, “інвестиції”, “финансовая грамотность”.

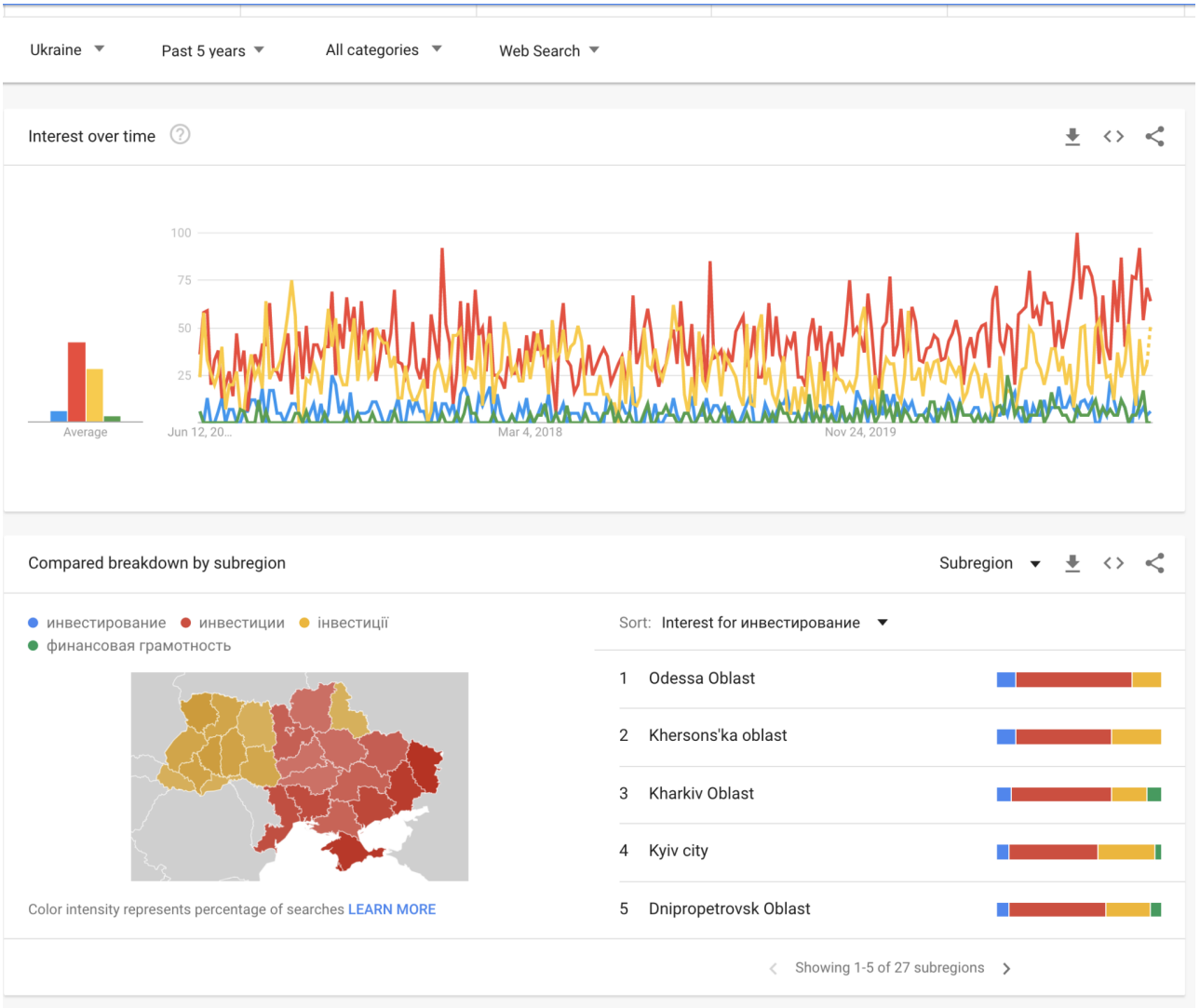


Рис.1.2. Лінії тренду чотирьох запитів в Google

Волатильність ринку в 2020, а також колосальний приріст акцій індексу S&P 500 у розмірі 15.76% (при інвестування дивідендів приріст складає 17.88%), спровокували ріст кількості інвесторів по всьому світу, які, на жаль, доволі часто приймали гарячкові рішення через необізнаність.

Враховуючи, все вищесказане, якісні ресурси самоосвіти у фінансовій сфері необхідні для покращення добробуту пересічного громадянина, який згодом зможе:

- ефективно розпоряджатись своїми фінансами
- займатись плануванням своїх доходів, витрат
- ставити чіткі та реалістичні довго- або короткострокові цілі, та використовувати різні інструменти для їх досягнення

1.2. Світові аналоги

Наявність матеріалів та програмного забезпечення для набуття навичок інвестування пропорційна залученню таких інструментів державою. Наприклад, в США кошти пенсійних фондів, заощадження громадян поміщаються в інвестиційні акаунти(напр. 401k, Roth IRA[2] та ін.) та вкладаються в акції, інші фонди та ін. Тому обізнаність громадян США щодо інвестування значно більша, адже це явно впливає на їхнє фінансове становище.

Прикладом аналогового програмного забезпечення є веб-сайт WallStreetSurvivor, де можна завести аккаунт і навчитись вкладати віртуальні несправжні кошти, що отримує користувач при реєстрації. На мою думку, це гарний спосіб навчитись на практиці ризик-менеджменту, якщо є страх втрати кошти.

---

[2] 401k, Roth IRA - основні найбільш популярні пенсійні плани пенсійної системи в США.

Wall Street Survivor спочатку був запущений як гра для інвестування акцій, що дозволяє користувачам інвестувати в акції, використовуючи віртуальні гроші. Поточна версія була запущена як доповнення до сайту в 2012 році і була представлена на виставці Finovate в Сан-Франциско, штат Каліфорнія того ж року. Згодом, веб-сайт було гейміфіковано за допомогою програмного забезпечення на платформі Bunchball Nitro.

Окрім такого практичного застосунку, є багато курсів на освітніх ресурсах на кшталт Coursera, edx, Udemy та інші. Знову ж таки, ці ресурси надають контент здебільшого англійською мовою, зрідка - російською.

1.3. Складність реалізації системи

Даний проект націлений на створення повноцінної навчальної платформи, що містила б наступний перелік функціональних частин:

- теоретичний матеріал для освоєння користувачем. Такий контент має бути гарно поясненим та оптимізованим для різних користувачів, бажано із визначенням рівня знань та рекомендацією відповідних матеріалів для ефективного навчання користувача
- функціонал для торгування акціями в навчальній формі. Повинен містити кількість типів ордерів відповідну навчальному матеріалу
- ігрова механіка у вигляді динамічних опитувань, міні ігор у стилі розгалуження історій, заснованих на рішеннях користувача
- блог із оглядом тенденцій ринку в світі та Україні

Найбільш вимогливою технічно складовою такої платформи є торгування акціями. Це зумовлено high-load природою таких застосунків та використанням зовнішнього API для отримання поточного курсу акцій.

Зрозуміло, що на початку проекту при малій кількості користувачів можна обійтись без оптимізації коду, проте все ж варто проектувати систему із розрахунком на high-load використання надалі. Більше того, вибір технологій,

чи структур даних пропорційно впливає на якість matching engine[3], що доволі часто визначає популярність брокерів по всьому світу, адже не тільки швидкість важлива кінцевому користувачу, але і стабільність роботи, відсутність помилок при обрахунках.

Щодо API, вони як правило мають обмеження на сумарну кількість запитів чи на одиницю акції, або мають постачають дані з певною затримкою. В такому випадку, доведеться обмежити вибір акцій до попередньо заданого списку, зорієнтувавшись на основні тренди чи зацікавленість користувачів. В майбутньому такий список можна буде розширити, або надати можливість користувачам купити підписку в додатку, щоб можна було розширити ліміт використання API.

1.4. Постановка задачі

В силу складності та обширності такого проекту, було прийнято рішення обмежити реалізацію кожного модуля та відкласти розробку і ведення блогу цілком.

Таким чином, в межах MVP[4] буде реалізовано наступний функціонал:

1. Ведення профілю користувача - реєстрація, збереження прогресу користувача, його стану. Застосування реферальної програми для отримання бонусів чи додаткових віртуальних коштів. Таким чином

---

[3] Matching engine - це електронна система, яка відповідає заказам на купівлю та продаж на фондовому ринку, товарному ринку чи іншій фінансовій біржі. Система узгодження замовлень є ядром усіх електронних бірж і використовується для виконання замовлень від учасників обміну.

[4] MVP — продукт з мінімальним функціоналом, який можна дати користувачам для використання. Зазвичай реалізується шляхом вдалого планування релізів, що дає можливість користувачам уже почати працювати, не очікуючи остаточної версії.

можна буде заохотити та поширити проект серед більшої кількості населення

2. Біржа для торгування акціями
    a. Інтеграція зовнішнього API з урахуванням можливих обмежень
    b. Підтримка створення ордерів типу Buy/Sell[5]
    c. Збереження ціни акцій в локальну базу для зменшення кількості реквестів до зовнішнього API
3. Ігрова механіка - декілька міні-ігор із розгалуженими історіями. За допомогою таких ігор, буде реалізована міні ефект метелика, що допоможе користувачу оцінювати мікро та макро наслідки його рішень, або ситуацій на ринку світу чи своєї країни
4. Статичний теоретичний матеріал. Такий матеріал буде подано в окремому модулі в додатку, із можливістю збереження прогресу його освоєння. Сам контент буде сформовано базуючись на статтях, книгах такої тематики.

Така реалізація передбачає розробку high-load сервера та клієнтського рішення. На даному етапі - найголовніше реалізувати ефективну та оптимізовану систему узгодження ордерів, адже це є серцем даного проекту.

Варто зазначити, що оскільки на початку система матиме мало користувачів, необхідно передбачити можливість запуску робота market-maker[6] з метою створення ліквідності на біржі. Такий бот буде створювати рівномірний потік ордерів по кожному типу замовлення, акції і так далі. Варто відмітити, що

---

[5] Buy/Sell - базові типи ордерів купівлі або продажу акцій без використання маржі

[6] Market maker - автоматизована інвестиційна стратегія, яка використовується для забезпечення ліквідності, заповнюючи книгу замовлень замовленнями на купівлю та продаж, щоб інші учасники ринку, як покупці, так і продавці, могли виконувати свої замовлення, коли їм потрібно.

для коректної роботи системи необхідно заборонити здійснення замовлення із самим собою.

Важливим нюансом кожної біржі є списання комісії за кожен здійснене виставлене замовлення. На початку, така інформація може відлякувати користувачів, тому є сенс відкласти реалізацію списування комісій до наступної ітерації проекту.

Висновки за розділом

В даному розділі було розглянути необхідність такого проекту для українського ринку, висвітлено актуальність тематики. Також було описано складність реалізації такої системи та різні нюанси, що могли вплинути на роботу системи в цілому, які важливо врахувати при постановці задачі і проектуванні системи в цілому в наступних розділах.

РОЗДІЛ 2

ПРОЕКТУВАННЯ СИСТЕМИ

При проектуванні системи важливо проаналізувати методи тренування навичок роботи в додатку. Оскільки дана тематика доволі обширна, на даному етапі виділяється перелік навичок та вмінь, що користувач отримає в ході роботи з додатком.

На даному етапі розвитку проекту планується розглянути набуття користувачем наступного переліку навичок та знань:

- навички ризик менеджменту

  Користувач вміє оцінювати якість прийнятих рішень, може побудувати інвест-стратегію та діяти відповідно до неї; адаптовує стратегію відповідно до стану ринку, вміє керувати своїми емоціями
- базові знання користування інструментами

  Вміє користуватись різними інструментами на біржі, знає їх переваги, недоліки, а також найбільш підходящі сценарії використання
- аналіз інформацію з відкритих джерел

  Слідкує та аналізує інформацію від різних джерел для прийняття виважених рішень, слідкує за тенденціями ринку

Даний перелік, незважаючи на малий розмір, є гарним фундаментом навчання та підлягає розширенню згодом. Варто зауважити, що такі навички вимагають часу та усвідомлених зусиль на їх досягнення, в той час як проект надає можливість пришвидшити цей процес за допомогою підібраних методів навчання, а також структурованого матеріалу.

Щодо методів навчання, на даному етапі розглядається два основних:

- симуляція роботи на біржі

Оскільки планується використання реальних даних стану ринку в проекті, користувачі зможуть практично інвестувати віртуальні кошти й

відслідковувати зміни свого портфоліо. Цей метод охоплює всі перераховані вище навички певною мірою

- тренування на практичних кейсах

  За допомогою різних наявних історичних кейсів, користувач зможе спостерігати так званий “ефект метелика” в інвестуванні

2.1 Аналіз вимог

Для реалізації описаної технології необхідно розробити додаток - віртуальний тренер. Такий додаток повинен відповідати функціональним та нефункціональним вимогам, розглянутим нижче, а також вимогам до інтерфейсу.

2.1.1 Функціональні вимоги

1. Користувач може зареєструватись в додатку та при реєстрації отримати віртуальні кошти на баланс, попередньо задані адміном додатку
2. Користувач може використати реферальний код при реєстрації для отримання віртуальних коштів на свій баланс в системі
3. Користувач може здійснювати замовлення на віртуальній біржі, а також змінювати їх статус/скасовувати.
4. Користувач може використовувати тип замовлень Market/Limit Buy та Market/Limit Sell.
5. Користувач має доступ до теоретичного матеріалу в додатку.
6. Користувач може зберегти свій прогрес в проходженні теоретичного матеріалу
7. Користувач має доступ до опитувань та тренування на практичних кейсах.
8. Коли користувач здійснює замовлення, це повинно відображатись на стані його рахунку відповідно до замовлення

9. Система повинна містити market-maker для забезпечення ліквідності на біржі.
10. Market-maker повинен конфігуруватись адміном додатку, включаючи такі характеристики інструменту, як частота замовлень, об’єм замовлень, здатність задовольняти власні замовлення.
11. Користувач має змогу змінювати пароль від свого профілю.
12. Адмін додатку має змогу створювати нові сторінки/розділи навчання теоретичного матеріалу
13. Адмін додатку може поповнювати перелік практичних кейсів у відповідному розділі додатку
14. Додаток повинен відображати реальний стан ринку відповідно до інформації отриманої через інтегроване API.
15. Користувач має змогу переглядати історію змін стану його портфоліо.
16. Користувач має змогу повідомити адмінів чи підтримку додатку в разі виникнення проблем з системою.

2.1.2 Нефункціональні вимоги

1. Коли користувач переходить на вкладку зі своїм портфоліо, оновлена інформація повинна розрахуватися протягом 2 секунд.
2. Система віртуальної біржі повинна витримувати навантаження у 1000 замовлень в секунду.
3. Система віртуальної біржі повинна коректно порахувати баланс користувача при здійсненні замовлень
4. Розроблений додаток повинен коректно працювати на платформах iOS 10.0+ та Android 4.1+.
5. Тексти програм повинні бути максимально простими для сприйняття і розуміння людиною.
6. Програма повинна передбачати можливість для змін і доповнень.

7. Кожна програма повинна супроводжуватися інструкціями щодо її використання, і ці інструкції повинні бути доступними і зрозумілими. В самій програмі повинні використовуватися коментарі, які пояснюють суть самої програми та основних її елементів.

2.1.3 Вимоги до інтерфейсу

1. Інтерфейс повинен забезпечувати простоту переходу від виконання однієї функції до іншої.
2. Реакція системи на всі типи запитів також повинна бути однозначною і зрозумілою і, по можливості, простою.
3. Інтерфейс не повинен бути перевантажений деталями щодо представлення розв’язку поставленої задачі
4. Процес взаємодії користувача з системою не повинен представляти ніяких труднощів.
5. Інтерфейс повинен надавати користувачу можливість відчувати себе повноправним керівником ситуації при розв’язанні всіх типів задач, тобто, забезпечувати його всією необхідною інформацією; користувач повинен бути впевненим, що він сам розв’язує поставлену задачу.
6. Інтерфейс повинен забезпечувати користувача на високому рівні вказівками стосовно його можливих дій, а також генерувати належний зворотний зв’язок на його запити.

2.2 Діаграма прецедентів

На Рис. 2.2.1 зображено діаграму прецедентів, на якій зображено відношення між акторами і прецедентами в системі.

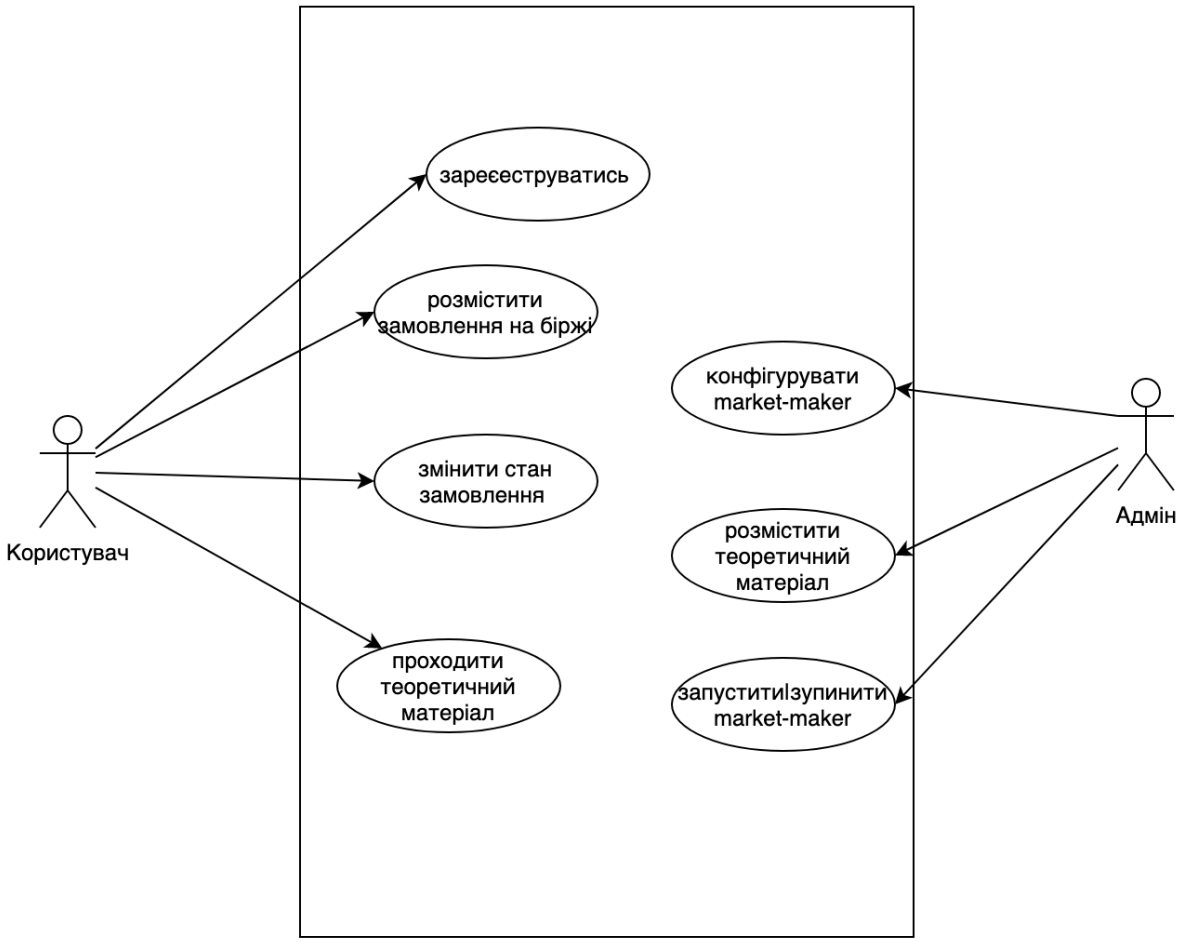


Рис. 2.2.1

Варто зауважити, що в системі відсутність тісно зв’язані дії користувачів та адмінів, отож система повинна працювати без нагляду з боку адмінів, та бути автономною.

2.3 Огляд проектованої архітектури додатку

На Рис. 2.3.1 наведено взаємодія програмних компонентів, та розбиття функцій серверної частини на сервіси. На даному етапі, таких проектується три основних:

- OrdersService

  Відповідатиме за створення та менеджмент ордерів. Містить в собі реалізацію matching engine та market maker. Тісно пов'язаний з роботою з базою даних.

- AuthService

  Відповідатиме за авторизацію, реєстрацію користувача, а також за ін’єкцію даних про користувача в контекст при використанні таких ендпоінтів, що вимагають авторизації від користувача

- StockRatesService

  На даному етапі, цей сервіс передбачає інтеграцію з MarketStack. Оскільки вибір інтеграції може змінитись з часом, було прийнято рішення реалізувати окремий сервіс, щоб запобігти зміні великої кількості коду. StockRatesService передаватиме стан ринку, ціни акцій і тд у зручному форматі, незалежно від інтеграції

Решта функціоналу по своїй суті не вимагає створення додаткових сервісів, а тому на даному етапі не передбачається їх проектування.

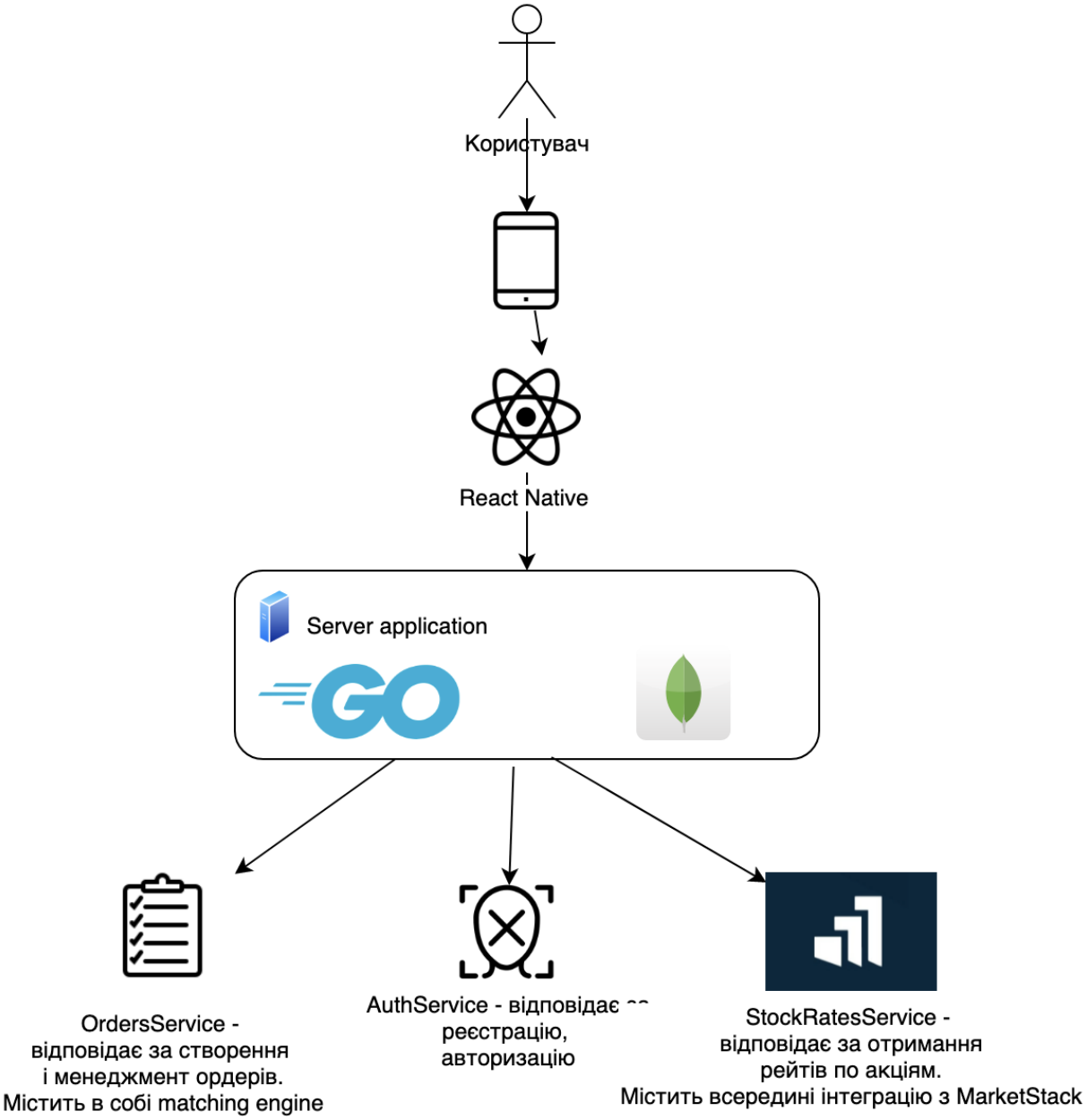


Рис. 2.3.1

Висновки за розділом

У даному розділі розглянуто проектування системи додатку. Описано, які саме результати має отримати користувач від взаємодії з додатком(навички та знання), розглянуто конкретні методи та інструменти, що використовуватимуться для досягнення таких результатів. Наведено перелік

функціональних, нефункціональних вимог, а також вимог до інтерфейсу додатку.

РОЗДІЛ 3

РЕАЛІЗАЦІЯ ВІРТУАЛЬНОГО КЕЙС ТРЕНЕРА

Розробка даного проекту відбувалась у кілька етапів. Перший з них полягав у визначенні підходящих інструментів, так як при збільшенні кількості користувачів, значно зросте навантаження на сервер. А тому було необхідно спроектувати застосунок так, щоб він міг витримати велике навантаження. Даний підхід орієнтований на перспективу, адже набагато важче масштабувати систему, що не є для того розрахованою.

3.1 Вибір інструментарію

3.1.1 Пререквізити

Початок розробки передбачав собою розгляд кількох питань стосовно вибору відповідних задачі інструментів, а саме:

- мова програмування, на якій здійснюватиметься розробка серверної частини
- база даних, що буде використовуватись
- вибір технологій для розробки клієнтського рішення

Незважаючи на те, що даний проект в ході цієї дипломної роботи являє собою MVP, при подальшій підтримці та розробці він має абсолютний потенціал стати корисним освітнім інструментом для багатьох громадян не лише нашої країни, а і поза нею, якщо перекласти матеріал та інтерфейс застосунку. Саме тому, відбір інструментарію відбувався із врахуванням наступних речей:

1. Оскільки в подальшому підтримка та розробка цього проекту може вестись не одноосібно, мова програмування має мати невеликий поріг входження для забезпечення максимальної кількості людей, що можуть долучитись до проекту.

2. Система повинна працювати швидко та стабільно в умовах великого навантаження. Тобто із збільшенням використання даної системи та із розширенням функціоналу, що в свою чергу теж впливатиме та кількість запитів користувачів, система може швидко стати непридатною для великих навантажень.
3. Аналогічно до пункту 1., база даних має бути простою у використанні під час розробки. Оскільки не передбачається необхідність саме в реляційних базах, нереляційні також постають при розгляді.
4. Інструментарій для користувацького застосунку повинен охоплювати достатню кількість платформ, проте і бути умовно легким у підтримці, веденні проекту та розширенню функціоналу

3.1.2 Мова програмування та фреймворки

В результаті основною мовою програмування для розробки програмного забезпечення було обрано Golang.

Golang (Go) — це мова програмування народжена фахівцями всередині Google, дизайн якої, базується на принципі KISS. За рахунок цього код більш подібний до C, а нові конструкції та типи зробили мову ефективною та сучасною[3].

Як зазначили автори мови, Go поєднує в собі швидкість розробки, характерну для динамічних мов, таких, як Python, із продуктивністю й безпекою компільованих, таких, як C й C++.

Також стало відомо, що Go підтримує багатопроцесорність і призначена у першу чергу для системного програмування. З її допомогою, наприклад, можна написати сервер, що обслуговує одночасно тисячі з'єднань[4].

Оскільки більша частина застосунку являє собою REST API, для пришвидшення та спрощення певних аспектів реалізації цього застосунку, потрібно було обрати HTTP веб-фреймворк на мові Go. Зрештою, вибір був між

фреймворками Gin[5] та Macaron[6]. Вони мають певні схожі риси, оскільки було засновані на іншому фреймворку Martini[7], що більше не підтримується. Проте в силу зменшення підтримки Gin, було обрано Macaron.

3.1.3 База даних

Для основного функціоналу проекту було обрано MongoDb. Ця база даних цілком задовольняє потреби програми, та легка в користуванні. Аналогічно до мови програмування, як було вказано вище, оперування базою даних не повинно вимагати великого досвіду у розробника чи навичок, що не можна здобути за короткий проміжок часу. MongoDb є однією з найпоширеніших NoSQL баз даних, а отже має багато ресурсів та більшість проблем у початківців мають рішення “з коробки”.

Також варто зазначити, що драйвер MongoDb для Golang є широко підтримуваний спільнотою розробників та є надзвичайно зручним у користуванні.

3.1.4 Вибір технологій для розробки клієнтського рішення

Оскільки переважна частина населення, що є цільовою аудиторією даного проекту, має смартфон - було прийнято рішення на початковому етапі вести розробку мобільного додатку в якості клієнтського рішення. На пізніших етапах можна розширитись на веб-застосунки чи планшети.

Оскільки на меті є досягнення великого числа користувачів, було прийнято обрати за основну технологію React Native. Це спричинено тим, що за винятком врахування деталей розробки для платформ Android та iOS, це значно пришвидшить розробку.

React Native - це багатоплатформний фреймворк з відкритим вихідним кодом для розробки нативних мобільних і настільних додатків на JavaScript і TypeScript, створений Facebook, Inc. [3] React Native підтримує такі платформи як Android, iOS, macOS, дозволяючи розробникам використовувати можливості

бібліотеки React поза браузера для створення нативних додатків, що мають повний доступ до системних API платформ.

Основні принципи роботи React Native практично ідентичні принципам роботи React, за винятком того, що React Native керує не браузерної DOM, а платформеними інтерфейсними компонентами. JavaScript-код, написаний розробником, виконується в фоновому потоці, і взаємодіє з платформеними API через асинхронну систему обміну даними, звану Bridge. У 2021 році очікується заміна Bridge на більш продуктивну синхронну модель обміну даними, підтримує парадигму zero-copy.

Хоча система стилів (спосіб конфігурації візуальних властивостей елементів інтерфейсу) React Native має синтаксис, схожий на CSS, фреймворк не використовує технології HTML або CSS як такі. Замість цього для кожної з підтримуваних фреймворком операційних систем реалізовані програмні адаптери, які застосовують заданий розробником стиль до платформенному интерфейсному елементу.

Також, необхідно враховувати, що користувач згодом вивчатиме фінансові інструменти, найбільш поширені та базові з яких - графіки(напр. графік японських свічок[7]).

Тому, дуже важливо, щоб обрані технологій підтримували такі інструменти та містили наявні бібліотеки, щоб не сповільнювати розробку додатку. В процесі дослідження цього питання було знайдено достатньо різних фреймворків чи бібліотек для React Native.

[7] Японські свічки - вид інтервального графіка і технічний індикатор, що застосовується переважно для показу змін біржових котирувань акцій

3.2 Реалізація основних функцій

3.2.1 Імплементація matching engine

Оскільки здійснення ордерів базується на швидкому сортуванні ордерів зі протилежною стороною, тобто ордер покупки зіставляється зі всіма можливими ордерами продажу, відфільтровані по тикеру[8] та відсортовані по ціні. В такому випадку, на ефективну роботу впливає вибір структур даних для реалізації швидкої вставки, видалення та сортування.

Проаналізувавши різні імплементації систем узгодження замовлень, було прийнято рішення використовувати червоно-чорні дерева як основну структура даних.

Червоно-чорні дерева — різновид збалансованих дерев, в яких за допомогою спеціальних трансформацій гарантується, що висота дерева h не буде перевищувати O(log n). Зважаючи на те, що час виконання основних операцій на бінарних деревах (пошук, видалення, додавання елементу) є O(h), ці структури даних на практиці є набагато ефективнішими, аніж звичайні бінарні дерева пошуку.

Також було реалізовано інструмент market-maker. На даному етапі цей інструмент є необхідним для забезпечення ліквідності на ринку, та конфігуруються адміном додатку.

Єдиною складністю такого інструменту, яка буде розглянута в наступнних ітераціях проекту - є конфігурація інструменту відповідно до так званого “біржового стакану” - це таблиця заявок на замовлення, що дозволяє оцінити попит та пропозицію на ринку. Таким чином, можна уникнути ситуацій, коли наприклад на реальному ринку всі продаються акції з тікером X, а в системі

---

[8] Тикер — коротка назва котируваних інструментів в біржовій інформації. Є унікальним ідентифікатором в межах однієї біржі або інформаційної системи. Напр. AAPL - Apple Inc

проекту market maker продовжує створювати попит і закривати замовлення, чого не сталося б на реальній біржі. Це доволі важко реалізувати, а тому відкладено до наступних ітерацій.

3.2.2 Керування доступом на основі ролей

При розробці першим питанням постало визначення способу розподілення та надання доступів користувачам застосунку. Проблема полягала в тому, що застосунок містить велику кількість функціоналу, що залежить від типу користувача. Наприклад, передбачається, що система матиме такі види користувачів:

- адмін біржі/додатку
- звичайний користувач
- контент-мейкер блогу/теоретичного матеріалу

У наступних ітераціях також передбачається введення підтримки - тобто користувача, що допомагатиме вирішити проблеми звичайних користувачів.

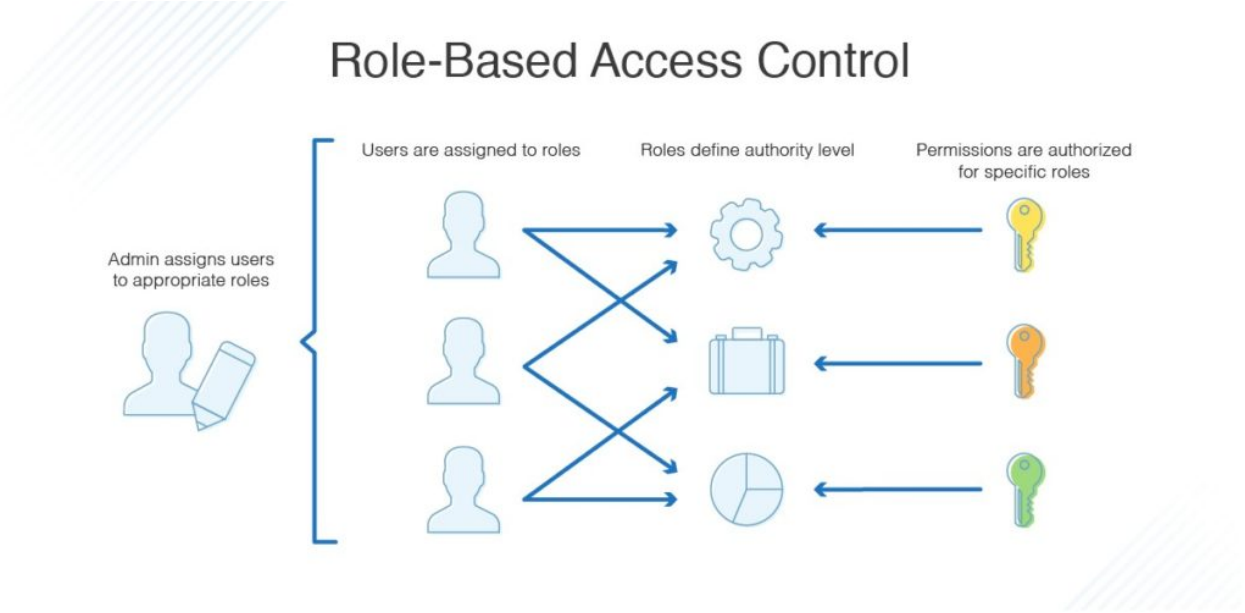
Оптимальним рішенням даної проблеми є керування доступом на основі ролей — розвиток політики вибіркового керування доступом, при якому права доступу суб'єктів системи на об'єкти групуються з урахуванням специфіки їх застосування, утворюючи ролі.

Формування ролей покликане визначити чіткі і зрозумілі для користувачів комп'ютерної системи правила розмежування доступу. Рольове розмежування доступу дозволяє реалізувати гнучкі та динамічно змінні в процесі функціонування комп'ютерної системи правила розмежування доступу.

Так як привілеї не призначаються користувачам безпосередньо і отримуються ними тільки через свою роль (або ролі), управління індивідуальними правами користувача по суті зводиться до призначення йому ролей. Це спрощує такі операції, як додавання користувача або зміна підрозділу користувачем.

Діаграма зображена на Рис. 3.1 пояснює схему роботи керування доступом на основі ролей. В Додатку А наведена реалізація контексту, що огортає macaron.Context додатковими властивостями та методами

Рис. 3.1



У застосунку визначено ряд ролей, що наведені в Лістингу 3.1:

```
const (
    UserNumber RoleNumber = iota + 1
    ContentMakerNumber
    AdminNumber
    SupportNumber
)
```

Лістинг 3.1

Відповідно, кожна з цих ролей має попередньо заданий набір дозволів, що були реалізовані у вигляді мапи ключ-значення. Ключем у даній мапі була операція, а значенням визначений для неї дозвіл.

Дозволи мають дефакто три стани - відсутність дозволу, дозвіл на read, дозвіл на write. В коді це визначено трьома константами, що наведено на Лістингу 3.2.

`PermissionRead` - дозволяє зчитувати інформацію, тобто менш абстрактно дозволяє здійснювати запити із методом типу **GET.** Інші запити, що змінюють стан ресурсів в системі, отримають відповідь “Unauthorized”.

`PermissionWrite` - дозволяє здійснювати запити, що змінюють стан ресурсів в системі. До таких запитів можна віднести основні три типи запитів із методами **PUT**, **POST**, **DELETE**.

`PermissionNone` - аналогічна відсутності дозволу на операцію.

```
type Permission int
const (
    PermissionNone Permission = iota
    PermissionRead
    PermissionWrite
)
```

Лістинг 3.2

Операцій - це попередньо визначений набір констант, кожна з яких представляє окремий функціонал. Наприклад, є певний сет операцій, що наведені в Лістингу 3.3:

```
const (
    OperationBlog Operation = iota + 1
    OperationExchange
    OperationSupport
    ...
)
```

Лістинг 3.3

Тоді, кожна роль буде мати строго визначені дозволи для кожної операції. Відсутність операції у мапі прирівнюється до відсутності дозволу будь-якого типу для даної операції. Це можна продемонструвати на прикладі визначення операцій та їх дозволів для ролі викладача факультету.

Створення та конфігурування ролі викладача факультету наведено в Лістингу 3.4.

```
RoleSuperAdmin = Role{
    Name:   "super_admin",
    Number: AdminNumber,
    Operations: map[Operation]Permission{
        OperationBlog: PermissionWrite,
        OperationExchange: PermissionWrite,
        OperationSupport: PermissionWrite,
        },
    }
```

Лістинг 3.4

Таким чином, RBAC було використано у проміжному програмному коді, що наведено в Додатку Б.

3.2.3 Написання кастомного логера

На початку реалізації програми з нуля, як правило, багато часу відходить на налаштування мінімальної працездатності різних сервісів, систем та проміжного програмного забезпечення(middleware). Одним із таких middleware є логгер подій, що був кастомно налаштований для максимізації отримання корисної інформації із логів. Додатково, він має різні конфігурації залежно від середовища виконання(напр. dev/production).

Його реалізація була заснована на аналогічному кастомному логері опен-сорс проекту Grafana. На Рис. 2.2 наведено приклад різних рівнів логування:

- INFO(інформативне логування)
- WARN(логування попередження)
- EROR(логування помилки)
- CRIT(логування критичної помилки)

```
INFO[05-12|20:36:30] HTTP Server Listen                    logger=http.server address=localhost port=3000
WARN[05-12|20:36:30] HTTP Server Listen                    logger=http.server address=localhost port=3000
EROR[05-12|20:37:09] HTTP Server Listen                    logger=http.server address=localhost port=3000
CRIT[05-12|20:38:03] HTTP Server Listen                    logger=http.server address=localhost port=3000
```

Рис. 2.2

Логер налаштовано таким чином, щоб логи зберігались протягом певного часу в файл, наприклад з ротацією в місяць.

3.3 Огляд розробки

На Рис. 3.3.1 зображено скрін-шот одного з екранів симулятора девайсу iPhone 12 Pro. На рисунку представлено екран портфоліо користувача, де він може оглянути стан покупок, розподіл акцій по секторах.

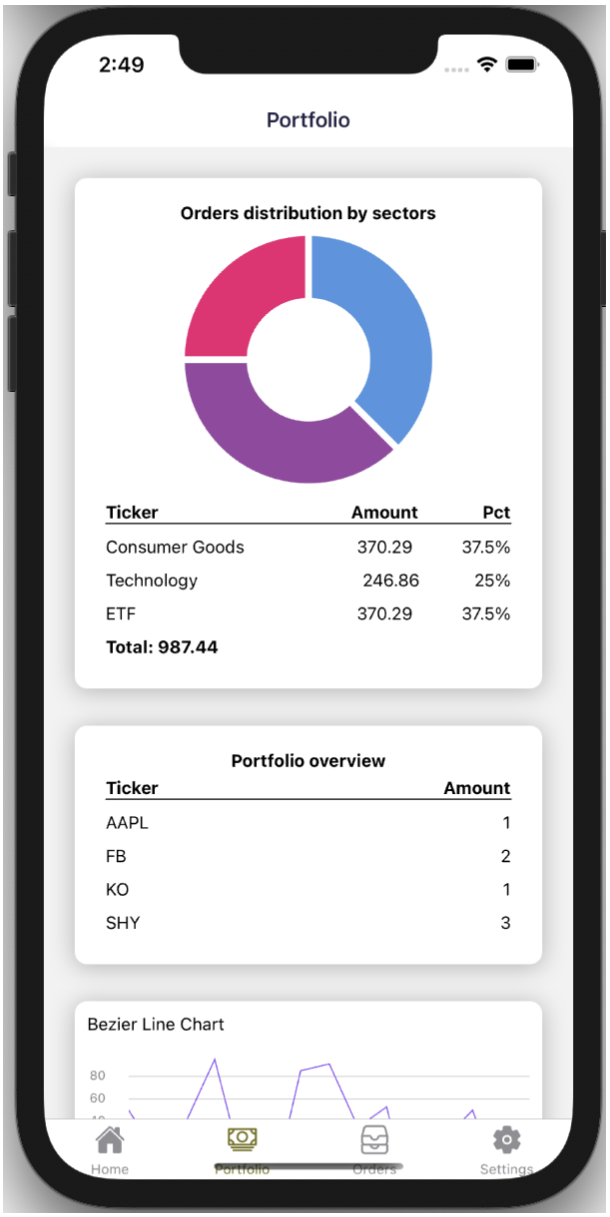


Рис. 3.3.1. Екран огляду портфоліо користувача

В третьому віджеті зображено графік зміни вартості портфоліо від часу. Оскільки графік досить легко конфігурувати, можна переглянути графік на різних проміжках часу - за один день, тиждень, місяць чи рік.

Такий зручний аналіз руху цін є однією з причин, чому було прийнято рішення зберігати у локальну базу результати запитів до зовнішнього API щодо поточної ціни кожної акції.

В наступних ітераціях буде додано можливість простого аналізу портфоліо користувача:

- для допомоги з диверсифікацією, щоб допомогти навчитись основам менеджменту ризиків і тд
- для надання рекомендацій з останніми новинами, тенденціями ринку щодо секторів, в яких користувач виявив найбільше зацікавленості

Також варто зазначити, що саме на цьому екрані відображаються наслідки рішень користувача, відповідно потрібно зробити інтерфейс максимально зрозумілим та ефективним з точки зору UI/UX[9].

На наступному Рис. 3.3.2 зображено аналогічний скріншот екрану створення ордера. Це доволі проста форма, яка в наступних ітерація буде змінена, щоб містити інші типи ордерів. Окрім того, важливо додати більше інформації по обраному тикеру для користувача. Для цього можна залучити ще одне зовнішнє API, наприклад з сайту finviz.com.

[9] UX / UI дизайн - це проектування будь-яких призначених для користувача інтерфейсів в яких зручність використання так само важливо як і зовнішній вигляд.

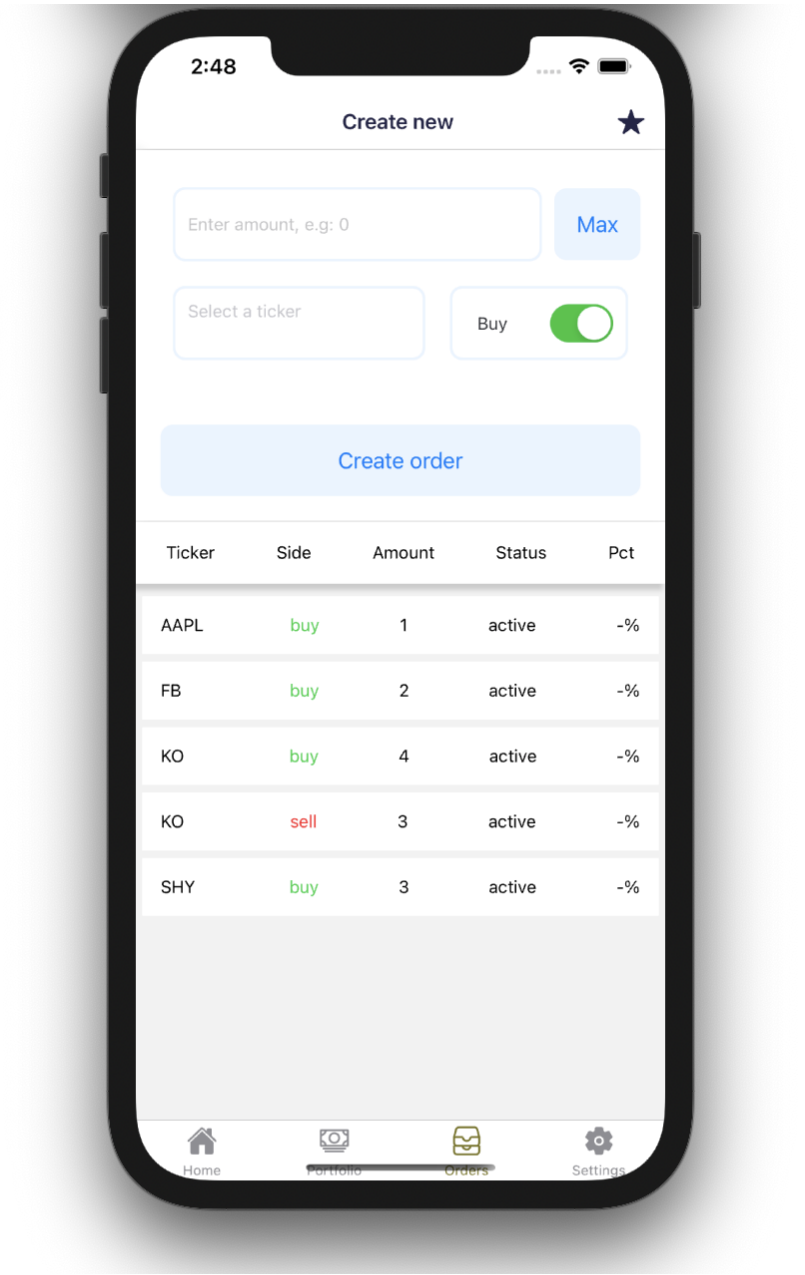


Рис. 3.3.2. Екран створення ордера

Для кінцевого користувача функціонал є доволі легким та не перевантаженим, попри те, що реалізація додатку є умовно складною. Це можна пояснити складністю матеріалу для навчання та часо-затратністю такої самоосвіти.

Також, для прикладу на Рис. 3.3.3 зображено одне з питань в модулі питань після теоретичного матеріалу, що відображає простоту у взаємодії із інтерфейсом, проте змушує користувача замислитись над питанням.

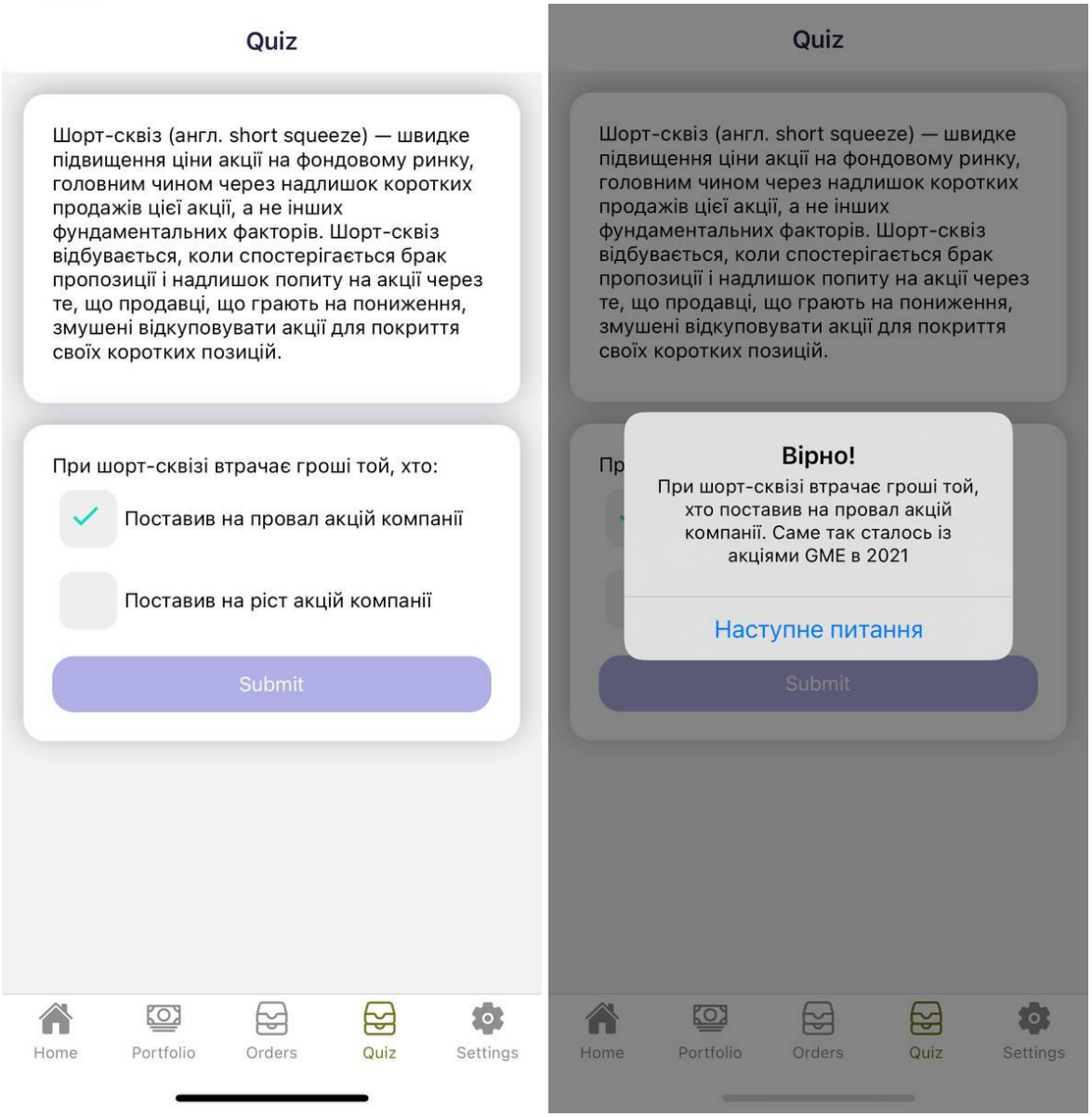


Рис. 3.3.3. Скріншот одного з питань по матеріалу

3.4 Створення документації

Незважаючи на те, що основна частина даного застосунку велась абсолютно одноосібно, подальша підтримка проекту на створення функціоналу спираючись на існуючу кодову базу буде значно ускладнено без належної документації.

Golang має вбудований інструмент генерації документації, проте він не надає необхідної гнучкості документації. Також в ньому відсутня підтримка відправлення запитів на сервер із документації. Тому було прийнято рішення використати застосунок OpenApi(в минулому Swagger) для автогенерації документації.

Swagger - це фреймворк і специфікація для визначення REST APIs в форматі, дружньому до користувача і комп'ютера (в нашому випадку JSON або YAML). Як правило, це працює таким чином - у кожного сервісу в певній папці лежить файл зі Swagger описом і зберігається це все прямо в git-репозиторії. Описи можуть бути як згенеровані за допомогою Swagger generator, так і записані туди вручну.

Тому було прийнято рішення створювати коментарі у підходящому для Swagger генератора форматі. Такі коментарі розташовуються біля кожного роута, а також моделей та деяких змінних. Пізніше за допомогою генератора вся документація збирається в один файл, що пізніше використовується для інтерактивної документації, що розташовується за адресою /docs.

3.5 Тестування та дебаг

На тестування даного додатку було відведено значну частину часу. Оскільки розробка основного функціоналу велась одноосібно, було важливо перевірити весь функціонал дуже прискіпливо та уважно, так як залучення інших людських ресурсів до цієї задачі було неможливим.

Основна частина тестування здійснювалась із використанням вбудованого в мову Golang пакету тестування, що значно полегшує проведення unit тестування. Даний пакет - go testing - входить в стандартну бібліотеку Golang.

Для того, щоб написати unit тест, необхідно:

1. Створити файл із назвою <$NAME$>_test.go, де $NAME$ - назва файлу, що містить функціонал, що буде покритий тестом або безпосередньо назва функціоналу. Важливо, щоб цей файл знаходився, у тій же директорії, де знаходиться файл із необхідним функціоналом.
2. Створити функцію з назвою Test$NAME$, де $NAME$ - назва функції, відносно якої проводиться тестування або назва довільна назва тесту.

Приклад тестування функції, що парсить таймстемп типу у рядковому представлені в формат типу `time.Time` рядок зображено на Рис. 3.5.1.

```
var date = time.Date( year: 2020,  month: 01,  day: 01,  hour: 12,
     min: 00,  sec: 0,  nsec: 0, time.Local)

var timestampStr = "1577872800"

func TestTimestampFromStringSuccess(t *testing.T) {
    testDate, err := TimestampFromString(timestampStr)
    if err != nil {
        t.Error(err.Error())
        t.FailNow()
    }

    if !testDate.Equal(date) {
        t.Error( args...: "test date not equal")
        t.FailNow()
    }
}
```

Рис. 3.5.1

Таким чином, було створено достатній об’єм unit тестів, що покривають основний функціонал.

Проте, в ході використання даного функціоналу неодноразово виникала проблема ініціалізації окремих сервісів. Суть задачі в тому, що за допомогою використання під час розробки пакету *github.com/facebookgo/inject,* ін’єкція залежностей(dependency injection) не становила труднощів в реалізації, але під час тестування потрібно було передбачити про ініціалізувати всі залежності сервісу, що використовувався при тестуванні.

Додатково за допомогою вбудованого пакету тестування, можна легко оцінити ступінь покриття тестами за допомогою вбудованої команди:

```
go test -cover
```
(2.1)

```
go test -cover -coverprofile=c.out
```
(2.2)

Наступна команда генерує звіт у форматі веб-сторінки та зберігає його у `html` розширенні.

```
go tool cover -html=c.out -o coverage.html
```
(2.3)

На Рис. 3.5.2 представлено скріншот сторінки, що була згенерована команд 2.2 та 2.3 у директорії, що в якій знаходиться всього один файл `timestamp.go`. Цей файл містив функції, що використовувались для парсингу часу у формат `Timestamp`.

Звісно, такі веб сторінки рідко використовуються для наглядності, проте числові дані, як наприклад 58.8% покриття тестами, дозволяють розробнику не тільки зрозуміти, скільки вже було зроблено, а й скільки, можливо, ще тестів потрібно буде написати.

Проте, в ході розробки та дебагу виникало багато нюансів, які неможливо передбачити, чи перевірити за допомогою таких тестів. Прикладом такої проблеми стало написання агрегації в базу даних, результат якої не тільки не відповідав очікуванню, а й став причиною доволі довгого розслідування некоректної поведінки агрегації.

```
import (
        "encoding/json"
        "github.com/globalsign/mgo/bson"
        "strconv"
        "time"
)

type Timestamp struct {
        time.Time
}

func (t Timestamp) MarshalJSON() ([]byte, error) {
        return json.Marshal(t.Unix())
}
func (t *Timestamp) UnmarshalJSON(v []byte) error {
        i, err := strconv.ParseInt(string(v), 10, 64)
        if err != nil {
                return err
        }
        t.Time = time.Unix(i, 0)
        return nil

}
func (t Timestamp) GetBSON() (interface{}, error) {
        return t.Time, nil
}
func (t *Timestamp) SetBSON(raw bson.Raw) error {
        var decoded time.Time

        bsonErr := raw.Unmarshal(&decoded)

        if bsonErr == nil {
                t.Time = decoded
                return nil
        } else {
                return bsonErr
        }
}

func TimestampFromString(s string) (Timestamp, error) {
        i, err := strconv.ParseInt(s, 10, 64)
        if err != nil {
                return Timestamp{}, err
        }
        return Timestamp{time.Unix(i, 0)}, nil
}
```

Рис. 3.5.2

Також варто зазначити, що основна взаємодія даних MongoDb та структур Golang здійснюється за допомогою тегів типу bson, що

проставляються навпроти кожного поля структури даних, що використовується для вставки, оновлення, агрегації та інших операцій з базою даних. Відсутність такого тегу або його некоректність впливає на правильність даних та/або їх присутність.

На Рис. 3.5.3 представлено скріншот документу бази даних, що був створений із використання структури, що містила правильні теги, проте була оновлена за допомогою іншої, що містила невідповідні теги. Результатом проведення таких операція став об'єкт, що містить в собі старі(`firstNname, lastNname, middleNname`) та нові дані(`firstname, lastname, middlename`), проте про наявність останніх система, а точніше сервер, не матиме жодних відомостей.

password	$2a$08$Sem9e5YJBogfpvAT9YbP8O5XRAUyk91/vdKiVf/1l7LyStmJli9s2
email	
accessToken	
roleNumber	2
status	1
notifications	{ 3 fields }
firstName	Test
lastName	User
middleName	Zero
firstname	Updated
lastname	zero
middlename	User

Рис. 3.5.3

3.6 Розширення функціоналу в майбутніх ітераціях

Як вже було вказано раніше, цей проект був реалізований як MVP для забезпечення мінімального поширення застосунку та використання користувачами. Проте в ході роботи виникало достатньо ідей та пропозицій до розвитку, що на це було виділено окремий розділ.

Далі буде перераховано функціонал, що міг би позитивно вплинути на використання застосунку, а також охоплення користувачами.

1. Розширення навчального матеріалу рекомендаціями, порадами, туторіалами

   При ознайомлені з новою сферою стають в нагоді списки корисних ресурсів, що як правило позитивно впливає на розвиток та навчання. Важливо зазначити, що сучасних ресурсів українською чи російською мовами досить мало, тому чим більш такі рекомендації пристосовані для кінцевого користувача, тим краще.

2. Розширення охоплених інструментів

   Як правило, інвестори не обмежуються одним інструментом - після “традиційного” інвестування також можна оглянути SPAC[10] та IPO[11]

Висновки за розділом

У даному розділі було безпосередньо розглянуто реалізацію додатку - вибір інструментів та технологій для проекту, а також обґрунтування вибору. Огляд розробки супроводжується скріншотами додатку, а також описано процеси розробки як створення документації, тестування та дебаг. Висунуто пропозиції до розвитку проекту.

[10] SPAC - компанія, створена спеціально для злиття з іншою приватною компанією, яка бажає вийти на біржу, минаючи процедуру IPO

[11] IPO - перший публічний продаж акцій приватної компанії

ВИСНОВКИ

В рамках даної дипломної роботи було спроектовано та розроблено програмне забезпечення теоретичного і практичного застосування для набуття навичок фінансової грамотності та інвестування. В силу обширності такої теми потрібно продовжити подальше обговорення креативного підходу, проте не можна заперечувати, що таке програмне забезпечення, будучи успішно спроектованим, здатне покращити фінансовий стан багатьох громадян нашої країни. Саме тому, ця робота має неабияке значення на розвиток та зміну особистої фінансової освіти в Україні.

В першому розділі цієї роботи було розглянуто різні причини та потреби громадян, що призвели до розгляду даної тематики. Аспекти реалізації та проектування системи з різноманітними нюансами розробки також було розглянуто.

В другому розділі дипломної роботи було описано процес вибору інструментарію та реалізації програмного забезпечення, що було раніше спроектовано. Варто зазначити, що багато рішень було прийнято із думкою про подальший розвиток цього застосунку. Через це було проаналізовано і різні мови програмування, бази даних та інші інструменти з усією можливою прискіпливістю.

Третій розділ роботи описує велику кількість ідей та функціоналу, що виходили за межі реалізації даного програмного забезпечення в ході дипломної роботи, але, незважаючи на це, є важливим джерелом ідей для впровадження змін та покращення програмного забезпечення.

Варто зазначити, що поставлені цілі було досягнуто у різних аспектах - від отримання нових знань та навичок до реалізації застосунку, що може змінити життя громадян у різних аспектах.

На основі всього вищевикладеного цілком можна стверджувати, що дана тема залишається актуальною, а тому потребує подальшого всебічного та глибокого дослідження.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Зятюк С. Індекс фінансової грамотності: яке місце посіла Україна – інфографіка [Електронний ресурс] / Сергій Зятюк. – 2021. – Режим доступу до ресурсу: https://education.24tv.ua/indeks-finansovoyi-gramotnosti-yake-mistse-posila-ukrayina-novini_n1537778.

2. Фінансова грамотність українців: бути чи не бути [Електронний ресурс] // Погляди. – 2021. – Режим доступу до ресурсу: https://thepage.ua/ua/experts/finansova-gramotnist-ukrayinciv-buti-chi-ne-buti.

3. Google придумав нову мову програмування [Електронний ресурс] – Режим доступу до ресурсу: https://ukranews.com/ua/news/14801-google-prydumav-novu-movu-programuvannya.

4. Советы сеньоров: как прокачать знания junior Go [Електронний ресурс] – Режим доступу до ресурсу: https://dou.ua/lenta/articles/senior-go/.

5. Gin Web Framework [Електронний ресурс] – Режим доступу до ресурсу: https://github.com/gin-gonic/gin.

6. Package macaron [Електронний ресурс] – Режим доступу до ресурсу: https://go-macaron.com/.

7. Martini [Електронний ресурс] – Режим доступу до ресурсу: https://github.com/go-martini/martini.

Додаток А - Реалізація контексту із використанням RBAC

```
func (hs *HTTPServer) InjectUserByAuth(ctx *m.StudentReqContext) {
	logger := hs.log.New("InjectUserByAuth")
	token := ctx.Req.Header.Get("Authorization")
	accToken, err := hs.AuthService.ValidateToken(token)
	if err != nil {
		logger.Error(err.Error())
		ctx.JSON(http.StatusForbidden, "not authorized")
		return
	}
	user, err := hs.UsersRepo.FindOne(
m.NewFindByAccessTokenQuery(accToken))
	if err != nil {
		logger.Error(err.Error())
		ctx.JSON(http.StatusForbidden, "not authorized")
		return
	}
	ctx.Map(user)
	ctx.Next()
}
func (hs *HTTPServer) InitStudentContext() macaron.Handler {
	return func(ctx *macaron.Context) {
		studentCtx := &m.StudentReqContext{
			GenericReqContext: m.GenericReqContext{
				Context:    ctx,
				IsSignedIn: false,
				Logger:     log.New("client_context"),
```

```
        },
    }
    if ctx.Req.Header.Get("Authorization") != "" {
        accessToken, err := hs.AuthService.ValidateToken(tok)
        if err == nil {
            user, err := hs.UsersRepo.FindOne(
            m.NewFindByAccessTokenQuery(accessToken))
            if err == nil {
                studentCtx.Role = user.Role
                studentCtx.UserId = user.Id
                studentCtx.IsSignedIn = true
            }
        }
    }
    ctx.Map(studentCtx)   }}
```

```
func Auth(operation models.Operation) macaron.Handler {

	return func(c *models.StudentReqContext) {

		if !c.IsSignedIn {

			c.JSON(http.StatusForbidden,
map[string]interface{}{"message": "not authorized"})

			return

		}

		if perm, ok := c.Role.Operations[operation]; !ok {

			c.JSON(http.StatusForbidden,
map[string]interface{}{"message": "Permission denied"})

			return

		} else {

			if c.Req.Method != http.MethodGet&& !perm.CanWrite() {

				c.JSON(http.StatusForbidden,
map[string]interface{}{"message": "Permission denied"})

				return

			}}

		c.Next()

	}}

func RequireAuth() macaron.Handler {

	return func(c *models.StudentReqContext) {

		if !c.IsSignedIn {

			c.JSON(http.StatusForbidden,
map[string]interface{}{"message": "not authorized"})

			return

		}

		c.Next()

	}}
```