Вінницький національний технічний університет

(повне найменування вищого навчального закладу)

Факультет інтелектуальних інформаційних технологій та автоматизації

(повне найменування інституту, назва факультету (відділення))

Кафедра автоматизації та інтелектуальних інформаційних технологій

(повна назва кафедри (предметної, циклової комісії))

МАГІСТЕРСЬКА КВАЛІФІКАЦІЙНА РОБОТА

на тему:

«Розробка програмного забезпечення мікросервісу другого фактору автентифікації»

Виконав: студент 2 курсу, групи 1ІСТ-21м

спеціальності 126 – «Інформаційні системи та технології»

(шифр і назва напряму підготовки, спеціальності)

Омельченко В.О.

(прізвище та ініціали)

Керівник: к.т.н., доцент каф. АІІТ Кулик Я.А.

(прізвище та ініціали)

Опонент: _____

(прізвище та ініціали)

Допущено до захисту

Завідувач кафедри АІІТ

_____ Бісікало О.В.

(прізвище та ініціали)

« ___» _____ 2022 року

Вінниця ВНТУ – 2022 рік

Вінницький національний технічний університет
Факультет інтелектуальних інформаційних технологій та автоматизації
Кафедра автоматизації та інтелектуальних інформаційних технологій
Рівень вищої освіти ІІ-й (магістерський)
Галузь знань – 12 Інформаційні технології
Спеціальність - 126 Інформаційні системи та технології
Освітньо-професійна програма - Інформаційні технології аналізу даних та зображень

ЗАТВЕРДЖУЮ
Завідувач кафедри АІІТ
Бісікало О.В.

« _» вересня 2022 року

З А В Д А Н Н Я
НА МАГІСТЕРСЬКУ КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТУ

_ Омельченко Вікторії Олександрівні___
(прізвище, ім'я, по батькові)

1. Тема роботи «Розробка програмного забезпечення мікросервісу другого фактору автентифікації»
керівник роботи Кулик Ярослав Анатолійович, к. т. н., доцент
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)
затверджені наказом вищого навчального закладу від "___" 2022 року
2. Строк подання студентом роботи 12.12. 2022 року
3. Вхідні дані до роботи: ім'я бота, токен бота, username користувача, chatId користувача.

4. Зміст текстової частини: Вступ; Дослідження способів та видів автентифікації; Аналіз видів двофакторної автентифікації; Інструменти для розробки програмного забезпечення; Розробка програмного забезпечення.

5. Перелік ілюстративного матеріалу (з точним зазначенням обов'язкових креслень) UML-діаграма пакетів для відображення структури програми; UML-діаграма пакетів для відображення детальнішої структури пакету service; UML-діаграма пакетів для відображення детальнішої структури пакету scenario; UML-діаграма класів для відображення взаємозв'язків для модуля Telegram; UML-діаграма класів для відображення взаємозв'язків для процесу отримання повідомлення; UML-діаграма класів для відображення взаємозв'язків для процесу обробки і формування відповіді; Схема комунікації з сервером нотифікацій; UML-діаграма послідовності для процесу автентифікації.

6. Консультанти розділів роботи

Розділ змістової частини роботи	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Розділи 1, 2, 3, 4	доцент кафедри АІІТ, доцент, к. т. н. Кулик Я.А.		
Економічний розділ	професор кафедри ЕПВМ, професор, к.е.н. Лесько О.Й.		

7. Дата видачі завдання "_07__" __09___2022 року

КАЛЕНДАРНИЙ ПЛАН

1	Дослідження способів та видів автентифікації	01.10.2022 р.
2	Аналіз видів двофакторної автентифікації	04.10. 2022 р.
3	Інструменти для розробки програмного забезпечення	20.10. 2022 р.
4	Розробка програмного забезпечення	25.10. 2022 р.
5	Підготовка економічної частини	15.11. 2022 р.
6	Оформлення пояснювальної записки, графічного матеріалу і презентації	30.11. 2022 р.
7	Апробація результатів дослідження	07.11.2022 р.
8	Публікації	10.11.2022 р.
9	Графічні матеріали:	
	UML-діаграма пакетів для відображення структури програми	01.12. 2022 р.
	UML-діаграма пакетів для відображення детальнішої структури пакету service	01.12. 2022 р.
	UML-діаграма пакетів для відображення детальнішої структури пакету scenario	01.12. 2022 р.
	UML-діаграма класів для відображення взаємозв'язків для модуля Telegram	02.12. 2022 р.
	UML-діаграма класів для відображення взаємозв'язків для процесу отримання повідомлення	02.12. 2022 р.
	UML-діаграма класів для відображення взаємозв'язків для процесу обробки і формування відповіді	02.12. 2022 р.
	Схема комунікації з сервером нотифікацій	03.12. 2022 р.
	UML-діаграма послідовності для процесу автентифікації	03.12. 2022 р.
10	Захист МКР	23.12. 2022 р.

Студент _____ _Омельченко В. О.

Керівник роботи _____ _Кулик Я.А.

АНОТАЦІЯ

УДК 004.42

Омельченко В.О. Розробка програмного забезпечення мікросервісу другого фактору автентифікації. Магістерська кваліфікаційна робота зі спеціальності 126 – інформаційні системи та технології, освітня програма – інформаційні технології аналізу даних та зображень. Вінниця: ВНТУ, 2022. 97 с.

На укр. мові. Бібліогр.: 0 назв; рис.: 8; табл. 10.

Магістерська кваліфікаційна робота присвячена розробці програмного забезпечення мікросервісу другого фактору автентифікації з використанням месенджеру Telegram, через можливість компрометації даних. Вона дозволяє зменшити ймовірність витоку даних з додатку Time Tracking.

На основі проведеного дослідження способів та видів автентифікації та аналізу аналогів було обґрунтовано доцільність створюваного продукту. А на основі проведеного аналізу інструментів для розробки було обрано оптимальні рішення та інструменти.

У розділі програмного забезпечення розроблено загальну структуру програми, описано підключення Telegram API, gRPC, розроблено відповідні діаграми пакетів та діаграми класів. Також, описано процес другого фактору автентифікації та розроблено до нього відповідну схему послідовностей. Описано сценарії роботи, їх імплементація та розроблено схему комунікації з іншим мікросервісом.

Графічна частина містить 7 діаграм та 1 схему з ілюстрацією результатів роботи.

Ключові слова: автентифікація, двофакторна автентифікація, Telegram, gRPC, protobuf, мікросервісна архітектура, Bot API.

ABSTRACT

UDC 004.42

Omelchenko V.O. Software development of second factor authentication microservice. Master's thesis on specialty 126 - information systems and technologies, educational program - information technologies of data and image analysis. Vinnytsia: VNTU, 2022. 97 p.

In Ukrainian speech. Bibliography: 0 titles; Fig.: 8; table 10.

The master's thesis is devoted to the development of microservice software for the second authentication factor using the Telegram messenger, due to the possibility of data compromise. It allows you to reduce the probability of data leakage from the Time Tracking application.

On the basis of the conducted study of methods and types of authentication and analysis of analogs, the expediency of the created product was substantiated. And on the basis of the conducted analysis of development tools, optimal solutions and tools were chosen.

In the software section, the general structure of the program is developed, Telegram API, gRPC connections are described, and relevant package diagrams and class diagrams are developed. Also, the process of the second authentication factor is described and the sequence scheme corresponding to it is developed. Work scenarios are described, their implementation, and a communication scheme with another microservice is developed.

The graphic part contains 7 diagrams and 1 scheme illustrating the results of the work.

Keywords: authentication, two-factor authentication, Telegram, gRPC, protobuf, microservice architecture, Bot API.

В І Д Г У К

на магістерську дипломну роботу «Розробка програмного забезпечення мікросервісу другого фактору автентифікації»
студента ФІІТА групи 1ІСТ-21м
Омельченко В.О.

Магістерська кваліфікаційна робота присвячена розробці програмного забезпечення мікросервісу другого фактору автентифікації з використанням месенджеру Telegram, через можливість компрометації даних. Вона дозволяє зменшити ймовірність витоку даних з додатку Time Tracking.

На основі проведеного дослідження способів та видів автентифікації та аналізу аналогів було обґрунтовано доцільність створюваного продукту. А на основі проведеного аналізу інструментів для розробки було обрано оптимальні рішення та інструменти.

При виконанні магістерської дипломної роботи студентка показала достатній рівень інженерної підготовки та вміння з різними програмними системами та інструментами для узгодження інтерфейсів API.

Всі поставлені задачі Омельченко В. вирішувала самостійно. В процесі роботи проявила технічну компетентність, розуміння сучасних програмних підходів та архітектури системи.

Робота виконувалась практично без порушень графіку, поставлені завдання виконувались вчасно та якісно. До магістерської роботи були в основному лише незначні зауваження.

Проте роботі притаманні **деякі недоліки**. Зокрема, в роботі недостатньо висвітлено переваги саме месенджера Telegram для реалізації другого фактору авторизації порівняно з іншими способами. Також, не до кінця висвітлено специфіку мікросервісів розроблюваного проекту. Проте дані недоліки можуть бути спричинені не завершеністю поточного проекту, зокрема технічної документації до нього.

В цілому магістерську дипломну роботу виконано на достатньо високому рівні і вона заслуговує оцінку **“А”**, а Омельченко В.О. – присвоєння ступеня вищої освіти магістр зі спеціальності 126 - «Інформаційні системи та технології», освітня програма «Інформаційні технології аналізу даних та зображень».

Керівник магістерської дипломної роботи

к.т.н., доцент кафедри АІІТ Кулик Я.А.

РЕЦЕНЗІЯ
на магістерську кваліфікаційну роботу
«Розробка програмного забезпечення мікросервісу другого фактору автентифікації»
студента групи 1ІСТ-21м
Омельченко В.О.

Магістерська кваліфікаційна робота, яку подано на рецензію, виконана у повному обсязі у встановлений термін.

Актуальність розробки є достатньою. Необхідність захисту від зламу у світі стоїть надзвичайно гостро. Здобувач пояснює необхідність розробки способу другого фактору авторизації.

Новизна, заявлена автором, що полягає у розробці інформаційної технології двохфакторної автентифікації з використанням месенджеру Telegram, можливо і не є повністю новим підходом, проте відкритих систем, які використовують даний підхід, на даний момент небагато.

Практична цінність є переконливою, оскільки дана розробка є частиною реального проекту, який буде використовуваний багатьма людьми.

У роботі є невеликі помилки по оформленню. Робота відповідає стандартам ДСТУ. Кількість схем у роботі у графічній частині є достатня для графічної частини. Показані схеми відповідають стандартам розробки програмного забезпечення, зокрема UML.

Авторка провела дослідження не тільки теоретичного матеріалу, а також дослідила потреби реальних ІТ-систем.

Зауваження до роботи: невелика кількість літератури і недостатня візуалізація мікросервісів. Дані зауваження не є критичними, враховуючи скорочення вимог до робіт в цьому році.

Магістерська кваліфікаційна робота виконана у відповідності з завданням із дотриманням всіх вимог. Робота заслуговує оцінки «відмінно», а її автор – присвоєння кваліфікації: ступінь вищої освіти магістр, спеціальності 126 - «Інформаційні системи та технології», освітньої програми «Інформаційні технології аналізу даних та зображень».

Рецензент: к.т.н., доц. кафедри САІТ С. О. Жуков

ЗМІСТ

ВСТУП

Актуальність роботи. В наш час люди все більше використовують мобільні додатки, сайти, соціальні мережі та додають туди сотні тисяч гігабайтів своїх даних. Тому, коли справа стосується мережевої безпеки, обережність не буває зайвою. Через це з’являється досить резонна тема для роздумів, а саме: захист своїх персональних даних. Кількість загроз кібербезпеки і рівень їх витонченості зростають швидкими темпами. Malwarebytes повідомляє, що частка кібератак в 2019 році збільшилася на 13%.

Однак є один простий спосіб, який допоможе підвищити культуру кібербезпеки. Це двофакторна автентифікація, або скорочено 2FA. Автентифікація – це процес підтвердження особи користувача з метою отримання доступу до комп'ютерної системи або онлайн-акаунту. Двофакторна автентифікація передбачає додатковий рівень безпеки, крім вашого пароля або PIN-коду. Існує ціла низка способів реалізувати двофакторну автентифікацію: програми для автентифікації, push-сповіщення, програмні маркери, голосова автентифікація і т. д.

Одним з відносно нових методів для створення двофакторної автентифікації є залучення месенджеру Telegram. Але безперечними плюсами використання телеграм є його захищеність та наявність у більш як 500 мільйонів осіб. Тому створення двофакторної автентифікації з залученням месенджеру Telegram виглядає досить перспективним рішенням.

Метою магістерської кваліфікаційної роботи є зменшення ймовірності витоку даних з додатку Time Tracking. Для досягнення поставленої мети було потрібно виконати наступні завдання:

– Проаналізувати наявні методи автентифікації.

– Дослідити існуючі на даний час практичні реалізації двохфакторної автентифікації.

– Проаналізувати і знайти оптимальні методи технічної реалізації.

– Розробити програмне забезпечення з використанням обраних технологій.

– Зробити економічне обґрунтування доцільності даного програмного забезпечення.

Об’єктом дослідження є процес розробки двофакторної автентифікації.

Предметом дослідження є методи та засоби програмної реалізації двофакторної автентифікації з використанням месенджеру Telegram.

Наукова новизна. Розроблено інформаційну технологію двохфакторної автентифікації з використанням месенджеру Telegram, яка відрізняється від існуючих аналогів використанням месенджеру Telegram, мікросервісної архітектури та системи віддаленого виклику процедур gRPC, що дозволяє зменшити відсоток скомпрометованих входів в систему.

Практична цінність одержаних результатів. Створення програмного забезпечення двохфакторної автентифікації з використанням месенджеру Telegram з можливість його подальшого використання та розширення.

Апробація результатів роботи. Результати і основні положення досліджень доповідалися на конференції «Контроль і управління в складних системах» ВНТУ (м. Вінниця, 2022).

Публікації. За тематикою дослідження опублікована робота в репозиторії ВНТУ [1].

1 ДОСЛІДЖЕННЯ СПОСОБІВ ТА ВИДІВ АВТЕНТИФІКАЦІЇ

Автентифікація – це підтвердження того, ким є користувач на вході. Це проходження перевірки автентичності. Термін найчастіше використовують серед інформаційних технологій. Порівняння пароля, введеного користувачем, з паролем, збереженим у базі даних сервера, — один із прикладів автентифікації. Подібна перевірка може бути як односторонньою, так і взаємною — все залежить від способу захисту та безпекової політики сервісу. Автентифікація дозволяє контролювати доступ до сервісів та ресурсів [2].

1.1 Методи автентифікації

Методи автентифікації поділяються залежно від типу ресурсу, структури та тонкощів організації мережі, віддаленості об'єкта та технології, яка використовується у процесі розпізнавання.

На основі ступеня конфіденційності можна виділити декілька рівнів автентифікації:

- доступ до інформації, витік якої не має значних наслідків для користувача та інтернет-ресурсу — у такій ситуації достатньо використовувати багаторазовий пароль;
- доступ до даних, розкриття чи зникнення яких призведе до істотних збитків — необхідна суворіша автентифікація: одноразові паролі, додаткова перевірка при спробі доступу до інших розділів ресурсу;
- доступ до систем конфіденційних даних – передбачає використання взаємної автентифікації та багатофакторних методів перевірки [3].

1.2 Способи автентифікації

Усі способи автентифікації можна розташувати за зростанням їхньої складності.

1.2.1 Базова автентифікація

При використанні цього виду автентифікації логін користувача та пароль входять до складу веб-запиту. Будь-який перехоплювач пакету інформації легко дізнається засекречені дані.

Тому базова автентифікація не рекомендується використовувати навіть у тих ситуаціях, коли засекречені дані не несуть суттєвої інформації ні для користувача, ні для інтернет-ресурсу. Небезпека полягає в тому, що викрадені дані можуть бути використані для доступу до інших систем. Так, за даними Sophos (компанія-виробник засобів інформаційної безпеки), 41% інтернет-користувачів використовують один і той же пароль для реєстрації на різних платформах, чи то банківська сторінка, чи форум, присвячений їх улюбленому хобі [3].

1.2.2 Дайджест-автентифікація

Має на увазі передачу паролів користувача в хешованому стані. На перший погляд може здатися, що рівень захисту в даному випадку трохи відрізняється від базової перевірки. Насправді це не так: до кожного пароля додається довільний рядок, що складається із символів (хеш), який генерується окремо на кожен новий веб-запит. Постійне оновлення хеша не дає зловмиснику можливості розшифрувати пакет даних - кожне нове підключення утворює інше значення пароля.

На основі цього методу автентифікації працює більшість інтернет-браузерів (Mozilla, Google Chrome, Opera) [4].

1.2.3 HTTPS

Цей протокол дає можливість шифрування не тільки логіна та пароля користувача, але й усіх інших даних, що передаються між інтернет-клієнтом та сервером.

Протокол використовується для введення особистої інформації:

- адреса;
- телефон;
- реквізитів кредитної картки;
- інших банківських даних.

У протоколу є один істотний недолік - він сповільнює швидкість з'єднання [5].

1.2.4 Автентифікація із пред'явленням цифрового сертифіката

Такий спосіб передбачає використання протоколів із запитом та відповідною відповіддю на нього.

Сторінка автентифікації направляє до користувача певний набір символів (адреса). Відповіддю є запит сервера, підписаний за допомогою персонального ключа.

Автентифікація за відкритим ключем застосовується як захисний механізм у наступних протоколах:

- SSL;
- Kerberos;
- РАДІУС [6].

1.2.5 Автентифікація з використанням cookies

Cookie — невеликий масив даних, що відправляється інтернет-сервером і зберігається на ПК користувача. Браузер при кожній спробі підключення до цього ресурсу посилає cookies як одну зі складових HTTP-запиту.

Як автентифікації cookie використовуються для систем безпеки чатів, форумів і різних інтернет-ігор. Cookies мають низький ступінь захисту — якщо сесія погано фільтрується, то викрасти їх не важко. Тому додатково застосовується прив'язка за IP-адресою, з якої користувач увійшов до системи [7].

1.2.6 Децентралізована автентифікація

Виділяють кілька основних протоколів, що працюють за принципом децентралізованої автентифікації:

- OpenID. Протокол дозволяє завести один пароль для кількох інтернет-ресурсів. Безпека здійснюється за рахунок цифрового підпису повідомлень на основі алгоритму Діффі-Хеллмана. Недоліками є вразливість перед атаками фішингу та атакою «людина посередині». На основі OpenID зараз працюють Google, Яндекс, BBC, PayPal, Microsoft та інші.
- OpenAuth. Працює за схожим алгоритмом із OpenID. Дозволяє використовувати сервіси AOL та будь-які інші надбудовані поверх них. При цьому у користувача не виникає потреби створювати новий обліковий запис на кожному сайті. Інформація про сесію зберігається над cookies, а самі cookies автентифікації відмежовані специфікованим доменом.
- OAuth. Дає можливість одному веб-ресурсу отримати доступ до даних користувача на іншому веб-ресурсі [3].

1.3 Багатофакторна автентифікація

Багатофакторна автентифікація має на увазі пред'явлення більш ніж одного «доказу» способу автентифікації для доступу до даних.

Такими «доказами» можуть бути:

- певне знання - інформація, якою володіє користувач (пін-код, пароль, кодове слово);
- володіння - предмет, який є у суб'єкта (флеш-пам'ять, електронна перепустка, магнітна банківська картка, токен);
- властивість – якість, властива виключно суб'єкту – сюди відносять дані біометрії та персональні відмінності: форма обличчя, індивідуальні особливості райдужної та сітчастої оболонки ока, відбитків пальців.

Одним із різновидів багатофакторної автентифікації є двофакторна Автентифікація (також називається двоетапною або подвійною). Такий спосіб має на

увазі під собою перевірку даних користувача на підставі двох відмінних один від одного компонентів [3].

1.4 Двофакторна автентифікація

Двофакторна автентифікація – це додатковий рівень захисту облікового запису. Крім введення пароля, потрібно також ввести одноразовий код, який надходить на пошту або телефон, або відбиток пальця. Цим ви підтверджуєте свою особистість. Коли ви активуєте цю опцію, окрім пароля хакеру потрібно також ввести код, щоб зайти у ваш обліковий запис. Ви також отримаєте повідомлення, якщо хтось спробує отримати доступ до вашого обліку.

Одноразовий код діє лише кілька хвилин або годин, після чого він самознищується. Таким чином, завдяки двофакторній автентифікації ваші онлайн-акаунти стають невразливими для кіберзлочинців.

У двофакторній автентифікації користувачі повинні ввести такі дані:

Крок 1: Ім'я користувача та пароль.

Крок 2: Згенерований код із телефону, пошти, іншого програмного забезпечення або біометричні дані – відбиток пальця, скан сітківки, голосова перевірка.

Двофакторна автентифікація за допомогою смартфона або програмного забезпечення – найпоширеніший метод.

1.4.1 Двофакторна автентифікація за SMS

Секретний одноразовий пароль надходить на мобільний телефон користувача у SMS-повідомленні.

Переваги:

- оскільки надходить SMS, кожен може скористатися цією опцією незалежно від наявності інтернету.

Недоліки :

- сигнал мережі – головний фактор, щоб отримати SMS із кодом;
- якщо втратити SIM-картку або телефон, не можливо пройти автентифікацію.

1.4.2 Двофакторна автентифікація по телефону

Користувачі отримують код перевірки за дзвінком після того, як вони правильно ввели пароль та ім'я користувача.

Переваги:

- так як це дзвінок, кожен може скористатися цією опцією незалежно від наявності інтернету.

Недоліки:

- сигнал мережі – головний фактор, щоб отримати дзвінок;
- якщо користувач у роумінгу, це дорого коштуватиме;
- якщо втратити SIM-картку або телефон, не можливо пройти автентифікацію.

1.4.3 Двофакторна автентифікація поштою

Користувачі отримують код перевірки або унікальне посилання на пошту після того, як вони правильно ввели пароль та ім'я користувача.

Переваги:

- доступно на комп'ютерах та телефонах.

Недоліки:

- на відміну від SMS або дзвінка, для отримання коду потрібний інтернет;
- лист може потрапити до спаму або не дійти через проблему з сервером;
- якщо хакери зламали поштові облікові записи, вони також можуть отримати доступ до коду.

1.4.4 Двофакторна автентифікація через програмне забезпечення

Це більш розвинений метод, який набирає популярності.

Користувачам потрібно встановити програму на комп'ютер або смартфон, щоб отримати код. Програмне забезпечення динамічно генерує коди для користувача на

короткий період часу. Після успішного входу в обліковий запис користувачу потрібно відкрити програму та ввести код, який згенерувала програму.

Приклади для двофакторної автентифікації - Google Authenticator, Authy, Microsoft Authenticator.

Переваги:

- легко використати;
- код автоматично генерується у додатку автентифікації;
- кросплатформова підтримка - програма працює і на комп'ютері, і на смартфоні. Навіть якщо втратити смартфон, все одно можна згенерувати код на комп'ютері.

Недоліки:

- будь-хто з доступом до вашого телефону або комп'ютера може зламати ваш обліковий запис.

1.4.5 Двофакторна автентифікація через апаратне забезпечення

Цей метод використовується в основному в корпораціях, але його можна використовувати на персональних комп'ютерах. У цьому випадку код автентифікації генерується за допомогою обладнання - брелка або електронного ключа. На устаткуванні є екран, де кожні 30-60 секунд динамічно генерується код.

Переваги:

- легко використати;
- не потрібне інтернет-з'єднання;
- дуже безпечний метод, тому що код генерується за допомогою стороннього обладнання.

Недоліки:

- дорого в установці та підтримці;
- пристрій можна втратити.

1.4.6 Двофакторна автентифікація за біометричними даними

При біометричній перевірці користувач сам стає кодом. Ваші відбитки, сітківка, розпізнавання вашого голосу або особи можуть бути перевірочним ключем під час автентифікації.

Переваги:

- безпечний метод, тому що ніхто не зможе скопіювати відбитки, голос чи обличчя.
- легко використати.
- не потрібне інтернет-з'єднання.

Недоліки :

- зберігання біометричних даних на сторонніх серверах може бути небезпечним.
- потрібні спеціальні пристрої типу сканерів чи камер [8].

1.5 Висновки

У першому розділі представлено опис та аналіз наявних методів автентифікації, де описано їх переваги та недоліки. Тож, після огляду наявних методів автентифікації зроблено висновок, що найоптимальнішим вибором для удосконалення методу автентифікації є – двофакторна Автентифікація через програмне забезпечення. Тому що, цей метод зараз активно розвивається, він має досить незначні мінуси на фоні значущих плюсів. До того ж, він дає можливість безпечніше передавати дані, тим самим збільшуючи захищеність облікового запису користувача.

2 АНАЛІЗ ВИДІВ ДВОФАКТОРНОЇ АУТЕНТИФІКАЦІ

2.1 «Приват24» для бізнесу

ПриватБанк — найбільший за розмірами активів український банк і лідер роздрібного банківського ринку України, зареєстрований 19 березня 1992 року. Ініціатором створення банку, його першим головою правління був український бізнесмен та політик Сергій Тігіпко.

Користуються послугами ПриватБанку майже 18 мільйонів українців, а клієнтів діджитал-банку Приват24 нині 13,5 мільйона. ПриватБанк доводить, що державна компанія може й має бути прибутковою та ефективною. Чистий прибуток ПриватБанку у 2020 році – 24,3 млрд грн. Це 61,2% прибутку всіх українських банків за рік. Загалом після націоналізації банк сплатив до бюджету 55,4 млрд грн.

«Приват24 для бізнесу» – це мобільний додаток для юридичних осіб та підприємців. Завдяки ньому можна отримати цілодобовий доступ до рахунків і платежів у режимі реального часу. Для входу в додаток досить бути клієнтом ПриватБанку [9].

Надаються такі можливості:

- контролювати всі операції за рахунком;
- здійснити будь-який платіж у національній валюті;
- сформувати виписку;
- купити або продати валюту;
- сплатити отримані електронні рахунки-фактури;
- оперативно створити кваліфікований електронний підпис SmartID та багато іншого.

Для входу в «Приват24 для бізнесу» використовується двофакторна Автентифікація. Що надає більшу впевненість в захищеності, як персональних даних так і даних підприємства. Клієнту надається можливість обрати з 3-ох можливих варіантів підтвердження особистості: повідомлення в мобільний додаток «Приват24», та два альтернативних – дзвінок з банку або код, який надсилається SMS-повідомленням. Якщо не виконати ніякої з 3-ох операцій – здійснити вхід неможливо.

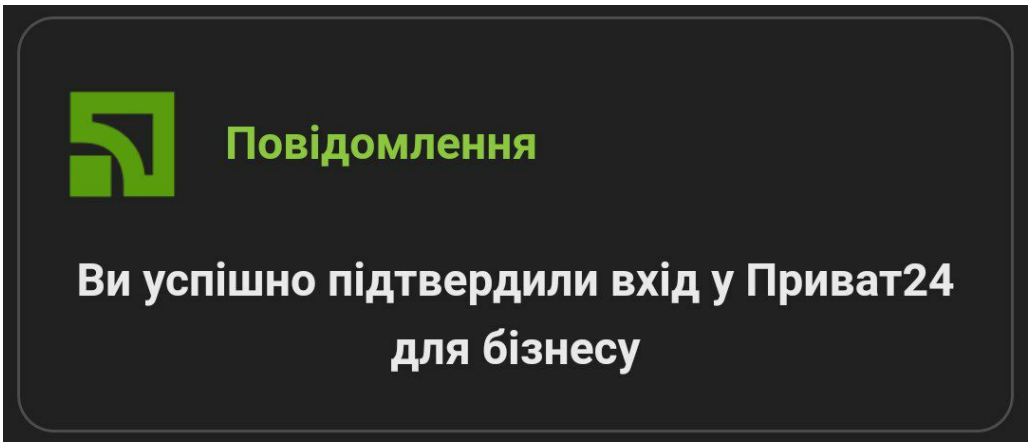

Рисунок 2.1 – Зображення підтвердження другого фактору автентифікації у «Приват24 для бізнесу» з допомогою мобільного додатку.

Стандартний метод підтвердження зображений на рис. 2.1 є досить зручним, але недоліками є те, що потрібно обов'язково мати мобільний телефон та встановлений на ньому додаток «Приват24». Також цей метод дійсний протягом всього кількох хвилин, а після потрібно обрати альтернативний канал комунікації. Ще досить вагомим недоліком є те, що великі компанії для ведення справ підприємства можуть наймати бухгалтерів. І в зв'язку з цим виникає складність, або заводити корпоративний номер телефону для роботи з банком, або налагоджувати зв'язок з керівником підприємства, коли потрібно здійснити вхід.

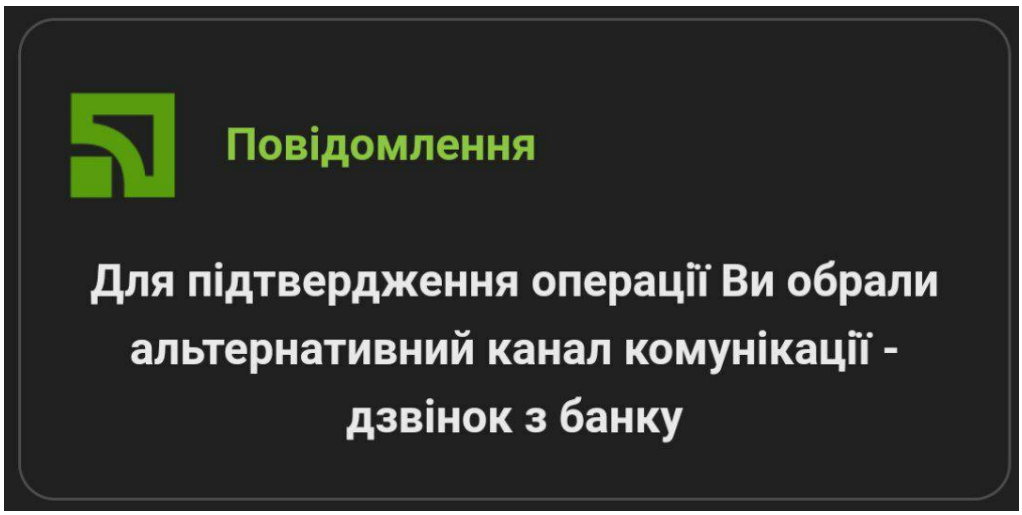

Рисунок 2.2 – Зображення підтвердження другого фактору автентифікації у «Приват24 для бізнесу» з допомогою альтернативного каналу зв'язку – дзвінку з банку.

Дзвінок з банку – є другим альтернативним каналом зв’язку з користувачем зображений на рис. 2.2. Досить зручний, адже має звукове повідомлення, якщо тільки не вимкнути звук телефону. Але прийняти дзвінок не завжди є можливо для користувача. Та й переконатись, що саме цей користувач прийняв дзвінок – не можливо. Ще одним суттєвим мінусом є те – що у багатьох людей на телефонах встановлені блокувальники спаму, а дзвінок від автовідповідача ПриватБанку розпізнається, як спам.

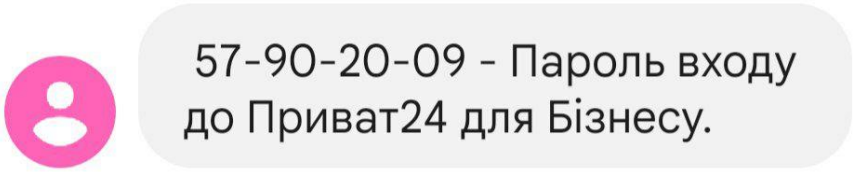

Рисунок 2.3 – Зображення підтвердження другого фактору автентифікації у «Приват 24 для бізнесу» з допомогою альтернативного каналу зв’язку – код, надісланий через SMS-повідомлення.

Код, надісланий через SMS-повідомлення – є третім альтернативним каналом зв’язку з користувачем зображений на рис. 2.3, який є одним з найпопулярніших методів для підтвердження автентифікації. Плюсом є те, що не потрібно встановлювати додаткових додатків. Але це досить незручний і ненадійний метод, тому що, код можна ввести з помилкою, або ж його можуть побачити треті особи і без телефону підтвердити особу є неможливим.

2.2 «Steam»

Steam – онлайн-сервіс цифрового розповсюдження відеоігор та програмного забезпечення від компанії Valve. Вона ж розробила CS:GO та Dota 2 – ігри, які уособлюють сучасний кіберспорт. Офіційний реліз Steam відбувся 2003 року. Зараз це найпопулярніший інтернет-магазин комп'ютерних ігор та передовий майданчик для розповсюдження своїх проектів серед інді-розробників.

Steam виступає засобом техзахисту авторських прав, платформою розрахованою на багатьох користувачів ігор і потокового мовлення, а також соцмережею для геймерів. До того ж, клієнт Steam забезпечує встановлення та регулярне оновлення ігор, їх зберігання на хмарному сервері, текстовий та голосовий зв'язок між гравцями.

Steam приваблює користувачів величезною бібліотекою ігор (більше 30 тисяч), зібраних в одному місці, а також постійними акціями, знижками та різними івентами. У 2020 році Valve також запустила програму очок Steam, що додає функції, які нагороджують користувачів за придбання ігор або активну участь у спільноті.

Для розробників та видавців ігор працює програма Steamworks, що дозволяє стати партнером сервісу та розміщувати на ньому свої ігри. Steamworks – це також набір інструментів та служб, які значно полегшують процес реалізації ігор [10].

Для входу в особистий акаунт зазвичай використовується однофакторна Автентифікація, але є можливість налаштувати двофакторну через мобільний додаток, що зображена на рис. 2.4.

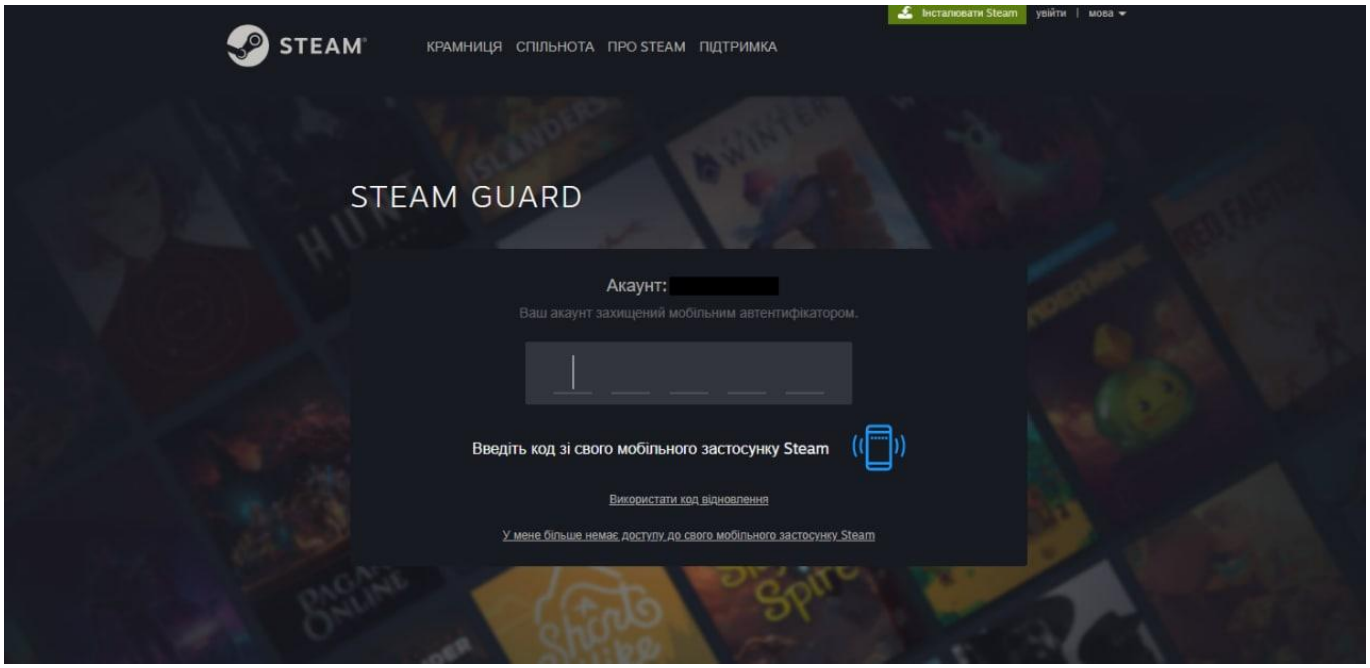

Рисунок 2.4 – Зображення другого фактору автентифікації у Steam.

Для двофакторної автентифікації використовується код, який надсилається в мобільний додаток Steam і відображається знизу, як зображено на рис. 2.5.

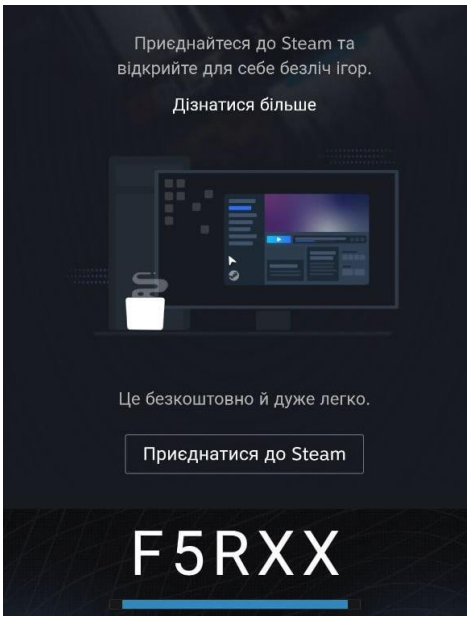

Рисунок 2.5 – Зображення коду підтвердження другого фактору автентифікації у Steam через мобільний додаток.

Значним недоліком є встановлення мобільного додатку, можливість неправильно ввести код та необхідність мобільного телефону з доступом до інтернету. Також як зображено на рис. 2.4 є можливість використання коду відновлення, який дається при налаштуванні двофакторної автентифікації, але це не є другим фактором, скоріше допоміжним механізмом при втраті мобільного телефону чи доступу до додатку.

2.3 Google акаунт

Обліковий запис Google дозволяє користуватися всіма послугами Google. Обліковий запис допомагає робити більше, персоналізуючи роботу з Google і пропонуючи легкий доступ до найважливішої інформації з будь-якого місця.

При входженні в обліковий запис, усі служби Google, якими використовуються, безперебійно працюють разом, допомагаючи виконувати повсякденні завдання, як-от синхронізувати Gmail із календарем Google і Картами Google, щоб завжди бути в курсі розкладу. Обліковий запис Google захищено передовою системою безпеки, яка автоматично допомагає виявляти та блокувати загрози [11].

Обліковий запис Google наполегливо рекомендує додати другий фактор авторизації, адже зберігає досить багато приватної інформації про користувача, аж до реквізитів банківських карток. Тому витік такої інформації до рук шахраїв може призвести до жахливих наслідків, від компрометуючих даних до втрати всіх коштів.

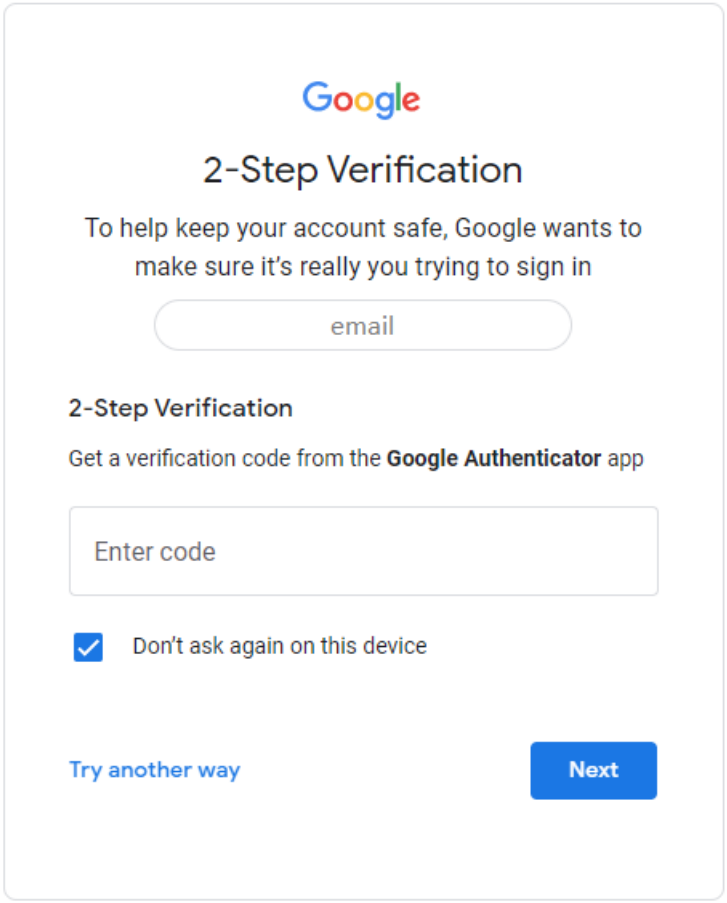

Рисунок 2.6 – Зображення другого фактору автентифікації у акаунті Google.

Досить незручним є встановлення мобільного додатку, хоча можливість використання його для інших ресурсів виправдовує встановлення. Також зберігається можливість неправильно ввести код, тому що час існування коду обмежений 30-ма секундами, хоча це забезпечує меншу можливість компрометації коду. Також є необхідність наявності мобільного телефону з доступом до інтернету. Навести зображення даного функціоналу не є можливим, адже Google Authenticator не дозволяє зробити захоплення екрану через приватність.

2.4 Висновки

В даному розділі було розглянуто програми, які використовують двофакторну автентифікацію. Було розглянуто їх структуру, переваги та недоліки кожного виду автентифікації. Проаналізувавши отримані дані, можна сказати, що удосконалення методу автентифікації для додатку Time Tracking на основі засобів месенджера Telegram, це досить нова ідея – яка не має прямих аналогів, але має загальні принципи двофакторної автентифікації через програмне забезпечення. Основним недоліком використання програмного забезпечення є його встановлення, але у випадку Telegram, він і так встановлений вже в багатьох користувачів. При тому, його можна використовувати не тільки для двофакторної автентифікації в даному додатку, а й в інших та за його прямим призначенням – як месенджер. Також месенджер є досить захищеним, так що хвилюватись за компрометацію даних не доводиться. Головним плюсом є те, що дана реалізація може бути розширена, залучати також інші месенджери та створювати додатковий функціонал, такий як – комунікація з користувачем.

3 ІНСТРУМЕНТИ ДЛЯ РОЗРОБКИ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

3.1 Вибір архітектури

Підхід до вибору архітектури проекту один з найважливіших етапів. Нижче буде розглянуто два ключових типи і наведено їх переваги та недоліки.

Монолітний застосунок — це єдиний загальний модуль, тоді як мікросервісна архітектура - це набір невеликих служб, що розгортаються незалежно.

У 2009 році компанія Netflix зіштовхнулася з проблемами зростання. Її інфраструктура не справлялася з попитом на послуги Netflix, що стрімко розвиваються. Компанія вирішила перенести IT-інфраструктуру з приватних центрів обробки даних до публічної хмари, а також перейти від монолітної архітектури до мікросервісної.

Компанія Netflix стала однією з перших великих організацій, які успішно перейшли з монолітної архітектури на хмарну архітектуру мікросервісів. Вона отримала спеціальний приз журі JAX у 2015 році — завдяки новій інфраструктурі, в яку вдалося впровадити методологію DevOps. Сьогодні компанія Netflix має понад тисячу мікросервісів. З їх допомогою здійснюється управління окремими частинами платформи та їх підтримка, тоді як розробники регулярно розгортають код (кількість розгортань може досягати кількох тисяч разів на день).

3.1.1 Монолітна архітектура

Монолітна архітектура - це традиційна модель програмного забезпечення, яка є єдиним модулем, що працює автономно і незалежно від інших додатків. Монолітом часто називають щось велике і неповоротке, і ці два слова добре описують монолітну архітектуру для проектування ПЗ. Монолітна архітектура — це окрема велика обчислювальна мережа з єдиною базою коду, де об'єднані всі бізнес-завдання. Щоб внести зміни в таку програму, необхідно оновити весь стек через базу коду, а також створити та розгорнути оновлену версію інтерфейсу, що знаходиться на стороні служби. Це обмежує роботу з оновленнями та потребує багато часу.

Моноліти зручно використовувати на початкових етапах проектів, щоб полегшити розгортання та не витрачати надто багато розумових зусиль при керуванні кодом. Це дозволяє одразу випускати все, що є в монолітному додатку.

Рисунок 3.1 – Зображення монолітної архітектури.

Організації можуть отримати вигоду як з монолітної архітектури, так і з мікросервісної в залежності від різних факторів. При використанні монолітної архітектури зручно створювати програми на основі однієї бази коду, тому її основна перевага полягає у швидкості розробки.

До переваг монолітної архітектури можна віднести такі особливості:

- просте розгортання (використання одного файлу або каталогу, що виконується, спрощує розгортання);
- розробка (програму легше розробляти, коли вона створена з використанням однієї бази коду);

- продуктивність (централізованій базі коду та репозиторії один інтерфейс API часто може виконувати ту функцію, яку під час роботи з мікросервісами виконують численні API);
- спрощене тестування (монолітний додаток є єдиним централізованим модулем, тому наскрізне тестування можна проводити швидше, ніж при використанні розподіленої програми);
- зручне налагодження (весь код знаходиться в одному місці, завдяки чому стає легше виконувати запити та знаходити проблеми).

Як і у випадку з Netflix, монолітні програми працюють досить ефективно доти, доки вони не стають занадто великими і не викликають проблем із масштабуванням. Щоб внести невелику зміну в одну функцію, необхідно виконати компіляцію та тестування всієї платформи, що суперечить agile-підходу, якому віддають перевагу сучасні розробники.

До недоліків монолітної архітектури можна віднести такі особливості:

- зниження швидкості розробки (великий монолітний додаток ускладнює та уповільнює розробку);
- масштабованість (неможливо масштабувати окремі компоненти);
- надійність (помилка в одному модулі може вплинути на доступність програми);
- перешкоди для впровадження технологій (будь-які зміни в інфраструктурі чи мові розробки впливають на додаток цілком, що часто призводить до збільшення вартості та тимчасових витрат);
- недостатня гнучкість (можливості монолітних додатків обмежені технологіями, що використовуються);
- розгортання (при внесенні невеликої зміни буде потрібно повторне розгортання всього монолітного додатку) [12].

3.1.2 Мікросервісна архітектура

Мікросервісна архітектура (або просто «мікросервіси») являє собою метод організації архітектури, заснований на ряді служб, що незалежно розгортаються. Ці служби мають власну бізнес-логіку та базу даних з конкретною метою. Оновлення, тестування, розгортання та масштабування виконуються всередині кожної служби.

Мікросервіси розбивають великі завдання, характерні для конкретного бізнесу, на кілька незалежних баз коду. Мікросервіси не знижують складність, але вони роблять будь-яку складність видимою і більш керованою, поділяючи завдання на дрібніші процеси, які функціонують незалежно один від одного і роблять внесок у загальне ціле.

Впровадження мікросервісів часто тісно пов'язане з DevOps, оскільки вони лежать в основі методики безперервного постачання, яка дозволяє командам швидко адаптуватися до вимог користувачів.

Рисунок 3.2 – Зображення мікросервісної архітектури.

Мікросервіси вирішують низку проблем, з якими стикаються компанії, що ростуть, при розвитку ПЗ. Оскільки архітектура мікросервісів складається з незалежно працюючих модулів, кожну службу можна розробляти, оновлювати, розгортати та масштабувати окремо від інших. Оновлення можна виконувати частіше, підвищуючи надійність, час безперебійної роботи та продуктивність програмного забезпечення.

Крім того, мікросервіси спрощують для команд оновлення коду та прискорюють цикли релізу завдяки безперервній інтеграції та безперервній поставці (CI/CD) . Команди можуть поекспериментувати з кодом і повернутись до попередньої версії, якщо щось піде не так.

До переваг мікросервісної архітектури можна віднести такі особливості:

- гнучкість (можливість використання гнучких методів роботи серед невеликих команд, які регулярно розгортаються);
- гнучке масштабування (коли мікросервіс досягає граничного навантаження, можна швидко виконати розгортання нових екземплярів цієї служби у супутньому кластері та знизити навантаження);
- безперервне розгортання (можливість випускання оновленнь не лише раз на тиждень, а й двічі-тричі на день);
- легкість обслуговування та тестування (команди можуть експериментувати з новими функціями та повертатися до попередньої версії, якщо щось не працює. Це спрощує оновлення коду та прискорює випуск нових функцій на ринок. Крім того, в окремих службах легко знаходити та виправляти помилки);
- незалежне розгортання (мікросервіси є окремими модулями, тому з ними можна легко і швидко виконувати незалежне розгортання окремих функцій);
- гнучкість технологій (у разі використання архітектури мікросервісів команди можуть вибирати інструменти з урахуванням своїх переваг);
- висока надійність (розгортаючи зміни для конкретної служби, можна не боятися, що програма вийде з ладу повністю);
- задоволені команди (команди, що працюють з мікросервісами, набагато краще відгукуються про свою роботу завдяки автономності та можливості самостійно створювати та розгортати програми, не чекаючи схвалення запиту pull протягом кількох тижнів).

Перехід до нової архітектури спочатку може здатись складним. Мікросервіси можуть зробити процес розробки складнішим і призвести до його розростання — швидкого та некерованого зростання. Іноді буває складно визначити, як різні компоненти пов'язані один з одним, хто володіє конкретним програмним компонентом або як уникнути втручання у роботу залежних компонентів.

До недоліків мікросервісів можна віднести такі особливості:

- розростання процесу розробки (мікросервіси ускладнюють роботу порівняно з монолітною архітектурою, оскільки у різних місцях виникає дедалі більше служб, створених кількома командами. Якщо розростання не контролюється належним чином, воно призводить до уповільнення розробки та зниження операційної ефективності);
- експонентне зростання витрат на інфраструктуру (кожен новий мікросервіс може мати свою вартість комплекту тестів, інструкцій з розгортання, інфраструктури хостингу, інструментів моніторингу тощо);
- додаткові організаційні витрати (командам потрібний додатковий рівень комунікації та співробітництва, щоб координувати роботу над оновленнями та інтерфейсами);
- проблеми при налагодженні (кожен мікросервіс має свій набір журналів, що ускладнює налагодження. Крім того, додаткові труднощі можуть виникати у тому випадку, коли один бізнес-процес виконується на кількох машинах);
- відсутність стандартизації (без загальної платформи може виникнути ситуація, де розширюється список мов, стандартів ведення журналів та засобів моніторингу);
- відсутність ясності у питаннях володіння (у міру появи нових служб збільшується і кількість команд, що працюють над ними. Згодом стає складніше визначити, які служби команда може використовувати і до кого слід звертатись за підтримкою) [13].

Відповідно описаній інформації про типи архітектури, було прийнято рішення обрати мікросервісну архітектуру. Адже, завдяки цьому програма буде відокремленою від всієї загальної логіки та зможе використовуватись для різних проектів. Окрім того, мікросервісна архітектура дозволяє незалежну мову програмування та технології для розробки. Також, обрання цього типу є логічним, але комунікація з телеграмом є окремим сервісом, і відділення її допоможе незалежно вносити зміни (наприклад, оновлення API для телеграму) та розгортати нові версії, коли це буде потрібно.

3.2 Вибір інтерфейсу передачі даних між мікросервісами

Інтерфейси прикладного програмування дозволяють програмам отримувати доступ до даних або функцій служби, операційної системи чи інших програм. gRPC і REST — це дві загальні специфікації API, які використовуються для визначення дизайну цих інтерфейсів.

3.2.1 REST API

REST — це абревіатура від Representational State Transfer. Це реалізація сучасних архітектурних стилів, які спираються на набір обмежень для забезпечення обміну даними в гіпермедійних системах. Системи, які підкоряються обмеженням REST, називаються RESTful.

Веб-API RESTful використовує параметри, закодовані в URL-адресі, для доступу до ресурсів за допомогою методів HTTP. Фреймворк REST API широко використовується в сучасній веб-розробці для створення масштабованих і надійних API без збереження стану.

API REST використовують дієслова HTTP: GET, POST, PUT і DELETE для виконання операцій створення ресурсу, читання, оновлення та видалення (CRUD). Сервер API надсилає дані клієнту та визначає формат передачі повідомлення, що використовується в тілі відповіді. Тіло відповіді також містить код відповіді, який сповіщає клієнта про успішний статус операції API [14].

Для того, щоб API був «RESTful», він повинен притримуватись 4-ох правил:

1) Уніфікований інтерфейс: це дивовижний спосіб опису відокремлених відносин між клієнтами та серверами. У той час як клієнти спілкуються з серверами через модель запит/відповідь, їх можна розробляти незалежно, не вимагаючи змін на сервері.

2) Без стану: RESTful API не має стану, тобто коли клієнт викликає сервер, він отримує всю інформацію у корисному навантаженні відповіді. Для цієї відповіді не потрібен додатковий контекст, і клієнт, що викликає, має всю необхідну інформацію з корисного навантаження відповіді.

3) Кешування: загальна функція RESTful API полягає в тому, що кінцеві точки кешуються веб-браузерами.

4) Багаторівневі системи: RESTful API можуть обмінюватися даними через кілька мережевих переходів, включаючи проксі та балансувальники навантаження. Це не впливає на зв'язок запит/відповідь [15].

3.2.2 gRPC

gRPC — це універсальна високопродуктивна платформа Remote Procedure Call (RPC) із відкритим кодом, яка забезпечує масштабованість і продуктивність архітектури мікросервісу. gRPC використовує виклики функцій для забезпечення зв’язку клієнт-сервер у мікросервісах, створених на різних мовах програмування. gRPC покладається на мову визначення інтерфейсу (IDL) для встановлення контракту/консенсусу щодо форматів даних і функцій, які потрібно викликати. API gRPC реалізовано за допомогою моделі RPC і HTTP 2.0 як транспортного протоколу.

API gRPC покладаються на буфери протоколів (protobufs), потокове передавання та протокол HTTP/2 для передачі повідомлень. Protobuf — це протокол серіалізації, який дозволяє автоматично генерувати клієнтські бібліотеки та просте визначення мікросервісів. Розробники API визначають служби та повідомлення між клієнтами та серверами в протокольних файлах. Файли завантажуються компілятором protoc, який генерує клієнтський і серверний код для обміну повідомленнями з віддаленими службами. Повідомлення, закодовані за допомогою буферів протоколу, набагато менші, ніж представлення XML або JSON, що робить аналіз менш інтенсивним для ЦП.

gRPC також використовує HTTP/2, який представляє численні оновлення архітектури RPC. Протокол представляє бінарний кадровий рівень, який розділяє пакети на менші повідомлення, оформлені у двійковому форматі, що робить їх компактними та легко переносимими. HTTP/2 дозволяє надсилати кілька паралельних запитів із моделлю двонаправленого зв’язку, що дозволяє реалізувати декілька викликів в одному каналі.

У той час як транспортний протокол HTTP/2 допускає кілька одночасних потоків, gRPC розширює цю можливість за допомогою каналів. Кожен канал підтримує кілька одночасних потоків через різні одночасні з'єднання. Канали забезпечують просте підключення до сервера API за певним портом і адресою. Канали також використовуються для створення заглушки клієнта.

Проектні обмеження, які керують розробкою API gRPC, включають:

1) Дизайн, орієнтований на ресурси та сервіси. gRPC сприяє розробці архітектури мікросервісів і філософії для слабозв'язаного обміну повідомленнями між системами для забезпечення ефективного доступу до розподілених об'єктів. API gRPC моделюються як ієрархія ресурсів, де хости поділяються на прості ресурси або ресурси колекції. Ресурсно-орієнтований дизайн наголошує на моделі даних над функціональністю, реалізованою на ресурсах; таким чином, API надає численні ресурси з невеликою кількістю методів HTTP.

2) З відкритим вихідним кодом. Під час розробки gRPC API команди розробників повинні зробити всі основні функції безкоштовними для загального використання. Усі артефакти та компоненти API мають бути випущені з відкритим кодом із використанням умов ліцензування, які сприяють глобальному прийняттю, а не перешкоджають йому.

3) Рівнева архітектура. gRPC реалізує багаторівневу архітектуру, базовим рівнем якої є базовий рівень gRPC. Інші рівні виконують абстракцію, тому розробникам gRPC API не потрібно турбуватися про базові деталі реалізації RPC. gRPC реалізує низькорівневі деталі зв'язку за допомогою коду, згенерованого Protobuf, і створює високорівневі абстракції для роботи клієнтів.

4) Мікросервіси , що не залежать від корисного навантаження, використовують різні кодування та типи повідомлень, зокрема JSON, XML і буфери протоколів. Усі реалізації API повинні дозволяти будь-якому мікросервісу використовувати будь-який формат повідомлень, щоб полегшити обмін повідомленнями та стиснення корисного навантаження. Протокол gRPC і його реалізації також повинні підтримувати підключаються механізми стиснення для забезпечення продуктивного обміну повідомленнями в широкому класі варіантів використання.

5) Контроль потоку. Хоча протокол HTTP/2 забезпечує швидкий обмін повідомленнями між кількома каналами, відсутність контролю потоку може призвести до вузьких місць трафіку. Реалізація керування потоком на рівні потоку та на рівні з'єднання дає змогу детально керувати пам'яттю, яка використовується для буферизації повідомлень у дорозі. gRPC використовує елемент керування кадром WINDOW_UPDATE, де і сервер, і клієнт повідомляють про кількість пакетів, які вони можуть отримати незалежно. Одноранговий вузол повинен поважати вказане значення октету, гарантуючи, що одержувач завжди має буферну ємність для вхідних повідомлень.

6) Обмін метаданими. Функції керування мікросервісом, такі як трасування та автентифікація, покладаються на обмін даними, які не оголошені як частина інтерфейсу служби. API повинні надавати ці дані, щоб забезпечити безперебійне виконання служб, оскільки декларативний інтерфейс служби змінюється з різною швидкістю, ніж дані, оброблені цими службами.

7) Розширення як API. Команди розробників програмного забезпечення, які хочуть реалізувати слабозв'язану взаємодію між службами, повинні розглянути можливість використання API замість розширень протоколу. Ці API можна використовувати для створення розширень для таких функціональних можливостей, як самоаналіз служби, балансування навантаження, моніторинг навантаження та перевірка працездатності, серед іншого.

8) Стандартні коди стану. Клієнти мають певний набір способів реагування на помилки API. Щоб спростити рішення щодо обробки помилок, коди помилок і статусів повинні бути обмежені та стандартизовані. У великих розгортаннях розробники можуть використовувати платформу обміну метаданими, щоб надати простір імен для стандартизованих кодів стану [14].

3.2.3 gRPC проти REST API — чим вони відрізняються

Інтерфейси API gRPC і REST побудовані з різними архітектурними стилями для обслуговування різних варіантів використання. API gRPC і REST знаходять широке застосування в сучасних розгортаннях додатків на основі мікросервісів із слабким зв'язком.

REST API створено для підключення розгортань мікросервісів, які запускають як внутрішні, так і ресурси з відкритим кодом, оскільки вони пропонують численні публічні інтеграції. API REST найкраще підходять для програм, які вимагають високошвидкісної ітерації передачі повідомлень HTTP. Оскільки API REST використовують виклики без збереження стану, вони ідеально підходять для хмарних програм, які можуть адаптувати гнучкі зміни робочого навантаження.

У більшості сторонніх інструментів сьогодні відсутні вроджені функції інтеграції gRPC API, що робить gRPC ідеальним для створення внутрішніх систем. gRPC можна використовувати для легких підключень до мікросервісів, оскільки повідомлення Protobuf дуже переносимі. Здатність gRPC обробляти мультиплексування та двосторонній зв’язок робить його придатним для потокової передачі в реальному часі в з’єднаннях із низьким енергоспоживанням і низькою пропускною здатністю.

Так як, одною з головних цілей розробки є портативність і швидкодія, було обрано технологію gRPC, яка дозволяє швидше здійснювати комунікацію з іншими мікросервісами.

3.3 Вибір API для Telegram

Для простої автентифікації можна було б використати віджет входу в Telegram, представлений в 2018 році. Віджет входу в Telegram – це простий спосіб авторизувати користувачів на веб-сайті. Щоб використовувати віджет входу, знадобиться бот Telegram. При першому вході в систему Telegram, віджет запитує номер телефону та надсилає повідомлення з підтвердженням через Telegram, щоб авторизувати браузер. Але недоліками цієї розробки є:

- неможливість змінити вигляд віджета (адаптувати під себе);
- неможливість створити мікросервіс, адже автентифікація буде додана з боку веб-сайту та буде керуватись особисто Telegram, тим самим додавання альтернативних месенджерів є неможливим;

- мала кількість використань та прикладів, що ставить під сумнів підтримуваність цього продукту.

Тому більшу перевагу надано саме використанню API для реалізації цієї задачі.

Існує два типи API для розробників. API Telegram і TDLib дозволяють створювати власні клієнти Telegram. Bot API дозволяє легко створювати програми, які використовують повідомлення Telegram як інтерфейс. Обидва цих API є безкоштовними.

API Telegram дозволяє створювати власні налаштовані клієнти Telegram. Він відкритий для всіх розробників, які хочуть створювати програми Telegram. Ознайомитись з цим API дозволяє відкритий вихідний код існуючих програм Telegram, в ньому можна знайти необхідні приклади.

TDLib – це бібліотека баз даних Telegram, інструмент для сторонніх розробників, який полегшує створення швидких, безпечних і багатофункціональних програм Telegram. В TDLib вже враховані всі деталі реалізації мережі, шифрування та локального зберігання даних, щоб розробники могли сконцентруватися на дизайні, адаптивних інтерфейсах та складній анімації.

TDLib підтримує всі функції Telegram і робить розробку програм Telegram простою на будь-якій платформі. Його можна використовувати на Android, iOS, Windows, macOS, Linux і практично в будь-якій іншій системі. Бібліотека має відкритий код і сумісна практично з будь-якою мовою програмування.

Bot API дозволяє підключати ботів до нашої системи. Боти Telegram — це спеціальні облікові записи, для налаштування яких не потрібен додатковий номер телефону. Ці облікові записи служать інтерфейсом для коду, що виконується на сервері.

Bot API є спрощеною версією Telegram API. Спілкування з цим сервером відбувається через HTTPS-інтерфейс. Це дозволяє не поглиблюватись у роботу протоколу шифрування MTProto — проміжний сервер оброблятиме все шифрування та зв’язок із Telegram API. Також є можливість використовувати Payments API, щоб підключити платіжну систему до боту [15].

Для створення комунікації з користувачем та другого фактору авторизації буде достатньо використати Bot API. Він є спрощеною версією, що дозволить зекономити

час та ресурси на розробку мікросервісу. Також згідно документації цей вид API містить всі плановані інструменти для розробки.

3.4 Висновки

В даному розділі було розглянуто та ключові інструменти для розробки. Аналоги було порівняно, а вибір аргументовано. Було обрано тип архітектури – мікросервісна. Мікросервісна архітектура є більш доцільною, так як головною перевагою є автономність і можливість використання у різних проектах. Також обрано інтерфейс для передачі даних – gRPC, через швидкість та портативність останнього. Та обрано інструмент для зв’язку з месенджером Telegram – Bot Api, який є спрощеною, але доречною в даному видку версією API.

4 РОЗРОБКА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

4.1 Розробка загальної структури

Головною частиною розробки проекту є створення зручної і зрозумілої архітектури проекту. Середовище програмування для Java – IntelliJ Idea, дозволяє створити загально-зручну структуру, яка потім доповнюється розробниками по мірі розробки.

Відповідно до додатку Б (рис. Б.1), ключовими для розробки пакетами є – scr та target. В пакеті scr знаходиться код, що виконується, а в target – згенеровані після збірки проекту файли. Сам scr складається з пакетів main та test, в яких відповідно знаходяться код програми та тести для налагодження.

Головним пакетом в main є – java, який містить вже реалізацію коду. Додатковими є згенерований пакет – resouces, який містить статичні ресурси для проекту, та доданий власноруч пакет – proto. Resouces – містить .yml файли, які зберігають ключову інформацію, таку як: паролі, ідентифікаційні номери, назву хостингу і т.д., для кожного з середовищ розгортання. Пакет proto складається з .proto файлів, які описують сценарії виклику віддалених процедур для gRPC.

В пакеті java міститься створена власноруч структура проекту. Для даної реалізації було обрано такі ключові елементи структури:

- service;
- scenario;
- grpc;
- storage;
- conf;
- exceptions.

Відповідно до додатку Б (рис. Б.2) пакет service містить загальний інтерфейс для імплементацій – BotService, в якому описані ключові функції, які потрібні для нової реалізації та пакет telegram з імплементацією цих вимог. Якщо виникне потреба

додати ще кілька засобів комунікації, це буде досить легко зробити. Потрібно лише додати імплементувати описаний вище інтерфейс та визначити відповідні методи.

Відповідно до додатку Б (рис. Б.3) пакет Scenario складається з загального інтерфейсу для імплементації сценаріїв та двох вже наявних реалізації. Окрім того, ще містяться пакети з DTO(Data Transfer Object), обробниками (handlers) та маперами, які допомагають перетворити один об’єкт на інший.

Conf – містить конфігураційні файли, а exceptions – типи помилок та їх обробники.

4.2 Підключення Bot API для Telegram

Для підключення Bot API було додано відповідні залежності для цієї бібліотеки у pom.xml. Та створено через FatherBot нового бота – спеціальний акаунт для якого не потрібен номер телефону і який існує для комунікації з користувачем. Отриманий токен та нове ім’я було додано в .yml файл.

Після цього можливе використання бібліотеки у проекті. Відповідно до додатку Б (рис. Б.4) для реалізації комунікації з Telegram було створено сервіс та унаслідувано TelegramLongPollingCommandBot. LongPolling - технологія дозволяє не створювати відкритих точок доступу до мікросервісу, як вебхук, а реалізується на технології опитування використовуваних елементів через якийсь сталий час. Тобто, бот-сервер раз в секунду, наприклад, запитує сервер Telegram чи не було змін у роботі з ботом. Така технологія є більш захищеною, але несе певне навантаження, та для полегшення розробки, це досить ефективний варіант.

Після, було імплементувано потрібні для комунікації з сервером Telegram методи, такі як:

- getBotUsername() (повертає ім’я боту);
- getBotToken() (повертає токен, виданий боту при реєстрації);
- processNonCommandUpdate(Update update) (отримує об’єкт в якому містяться дані про будь-які зміни).

Для опрацювання змін було використано патерн програмування Chain of Responsibility, щоб уникнути накопичення структури if-else, або switch case.

Ланцюжок обов’язків (Chain of Responsibility) — це поведінковий патерн проектування, що дає змогу передавати запити послідовно ланцюжком обробників. Кожен наступний обробник вирішує, чи може він обробити запит сам і чи варто передавати запит далі ланцюжком [17].

Тож завдяки йому, можливо створити окремі обробники на кожен тип оновлень і реалізувати їх окремими класами. Для цього було створено інтерфейс UpdateHandler, який імплементують його реалізації для конкретних функції. Реалізацію зображено в додатку Б (рис. Б.4).

Окрім цього, для імплементації взаємодії з Telegram можливо використовувати команди, які починаються з «/». Прикладом є реалізація команди «/start» при додаванні боту певним користувачем. Завдяки цьому було отримано інформацію про користувача, таку як chatId, name, username і т.д. Потім цю інформацію було використано для ініціювання взаємодії з користувачем.

4.3 Підключення gRPC

Для підключення gRPC потрібно додати залежності для цієї бібліотеки у pom.xml. Також потрібно додати плагін, який під час компіляції коду згенерує ptorobuf файли, які реалізують обмін інформацією між мікросервісами. Після цього потрібно обрати порт для сервера та зазначити хости і порти клієнтів (інших мікросервісів) у .yml файлі.

Після того, для запуску серверу потрібно імплементувати самий сервер та запускати його при збірці проекті завдяки анотації @PostConstruct. Також варто створити метод, який буде коректно завершувати всі процеси серверу і анотувати його @PreDestroy.

Також, було створено клас GrpcSender, як обгортку для імплементації внутрішньої логіки. Цей патерн програмування називається – «Фасад» і має на меті приховати складну імплементацію під досить простим інтерфейсом. GrpcSender

містить у собі компонент GrpcNotificationClientImpl, який саме і виконує створення і відправку повідомлення на інший мікросервіс, тобто віддалено викликає метод отримання даних на мікросервісі notification.

І заключною частиною для створення повноцінної комунікації було створено клас GrpcReceiverImpl. Який приймає запит на запуск певного сценарію та запускає його. В ньому знаходиться також, TypeEvaluator, який дозволяє обрати тип сценарію (автентифікація чи нагадування) та тип месенджеру. Додавання нового месенджеру не змусить змінювати поточну логіку комунікації між мікросервісами. Реалізацію зображено в додатку Б (рис. Б.5 та рис. Б.6).

4.4 Опис реалізації другого фактору автентифікації через Telegram

В даній реалізації існує два сценарії для мікросервісної комунікації. Одним із них і найголовнішим є MfaScenario, або сценарій другого фактору автентифікації.

Відповідно до додатку В можна прослідкувати, що початок сценарію ініціюється поза даною розробкою. Клієнт проходить перший фактор автентифікації, потім клієнт (браузер) робить запит на сервіс автентифікації, який генерує код та відсилає його сервісу нотифікацій, який в свою чергу пересилає його на бот сервіс. Після цього на сервіс приходить запит, на виконання певного сценарію, наразі це – автентифікація. Далі, як було описано раніше TypeEvaluator визначає сценарій та передає запит класу MfaScenario, який є імплементацією інтерфейсу Scenario. MfaScenario в свою чергу деактивує минуле повідомлення, якщо воно було, створює нове, яке зображено на рис. 4.1 та записує ідентифікатор користувача в сховище.

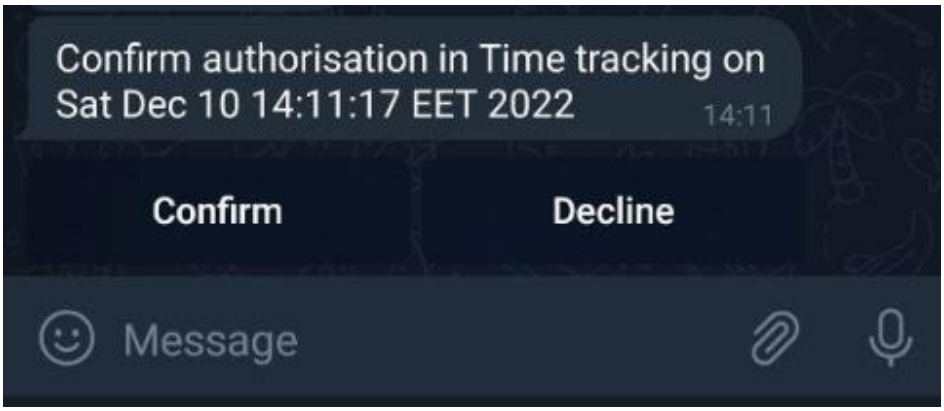

Рисунок 4.1 – Зображення повідомлення для підтвердження автентифікації.

Для запису даних було вирішено не використовувати віддалену базу даних, щоб не навантажувати сервіс. Збереження відбувається в пам’яті сервісу, що дає перевагу в швидкості отримання даних. Імплементація збереження даних описана в MfaScenarioRequestRepositoryImpl класі. Запис даних у сховище дає можливість відслідковувати кому були відправлені повідомлення та валідувати їх час життя. TimeOutHandler відповідає за це, він має асинхронну функцію, яка викликається раз на 10 секунд та перевіряє в сховищі чи дані ще валідні. Якщо у повідомлення вийшов час життя, користувач вже не може відповісти на запит, тому що приховуються кнопки і з’являється повідомлення, що час відповіді на запит вийшов. Схема комунікації користувача з ботом зображена у додатку Б (рис. Б.7).

Після того, як користувач підтвердить автентифікацію в клас TelegramBot приходить оновлення, яке в свою чергу опрацьовується відповідним обробником - ConfirmedAnswerHandler. ConfirmedAnswerHandler перевіряє на наявність потрібної відповіді та дістає з сховища код для автентифікації. Після цього формує відповідь для сервісу нотифікацій, яка містить код та відсилає її. Також, для користувача ініціалізується приховування кнопок та надсилається його результат його вибору який зображено на рис. 4.2.

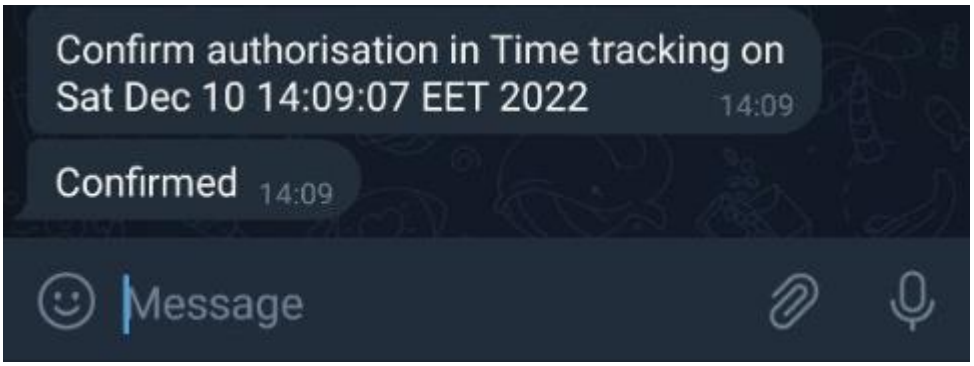

Рисунок 4.2 – Зображення повідомлення підтвердженої автентифікації.

Відповідно до цього, коли користувач не підтверджує автентифікацію, DeclinedAnswerHandler – формує іншу відповідь, яка не містить коду, відсилає її та приховує кнопки. Приклад зображено на рис 4.3.

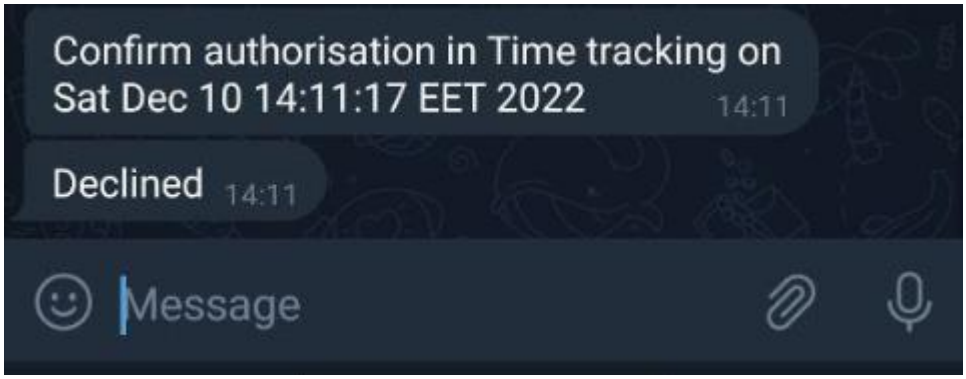

Рисунок 4.3 – Зображення повідомлення не підтвердженої автентифікації.

Після відповідно до наданої відповіді, дані передаються сервісу нотифікацій, а після сервісу автентифікації, які опрацьовують результат і надають або ні, доступ до даних. Діаграму послідовності зображено в додатку Б (рис. Б.8).

4.5 Опис реалізації відправлень нагадувань через Telegram

Другим можливим сценарієм відповідно до додатку В є – відправка повідомлень користувачеві. В даному проекті вона використовується для створення нагадувань. Цей сценарій ініціюється сервісом нотифікацій, в якого, в свою чергу є асинхронна функція для створення щоденних запланованих повідомлень.

Коли вона ініціалізується, запит проходить описаний вище шлях і потрапляє до класу NotificationScenario, де створюється повідомлення і надсилається користувачеві в чат. Приклад зображено на рис. 4.4.

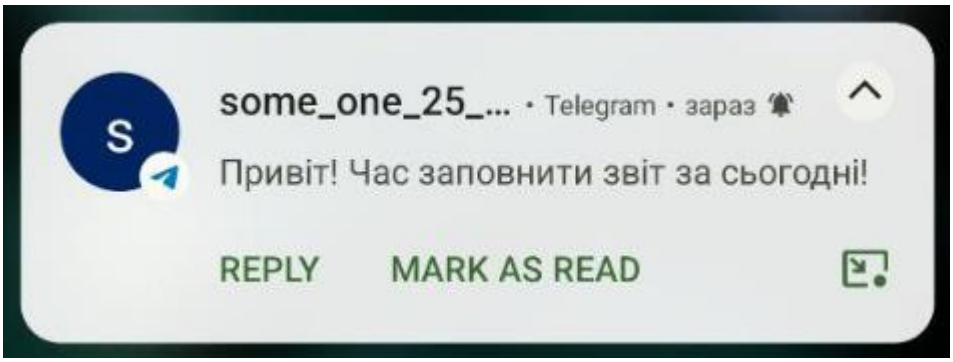

Рисунок 4.4 – Зображення нагадування у верхньому меню телефону.

4.6 Висновки

В даному розділі було описано етапи розробки програмного забезпечення другого фактору автентифікації. Детально описана структура проекту, підключення до Bot API для комунікації з Telegram, підключення засобу для мікросервісної комунікації gRPC. Також було створено три діаграми пакетів, одна з яких загальна, а ще дві уточнюючі та діаграму класів. Було описано реалізацію другого фактору автентифікації та відправлення повідомлень. До них було створено діаграму прецедентів.

5 ЕКОНОМІЧНА ЧАСТИНА

Науково-технічна розробка має право на існування та впровадження, якщо вона відповідає вимогам часу, як в напрямку науково-технічного прогресу та і в плані економіки. Тому для науково-дослідної роботи необхідно оцінювати економічну ефективність результатів виконаної роботи.

5.1 Проведення комерційного та технологічного аудиту науково-технічної розробки

Метою проведення комерційного і технологічного аудиту дослідження за темою «Розробка програмного забезпечення мікросервісу другого фактору автентифікації» є оцінювання науково-технічного рівня та рівня комерційного потенціалу розробки, створеної в результаті науково-технічної діяльності.

Оцінювання науково-технічного рівня розробки та її комерційного потенціалу рекомендується здійснювати із застосуванням 5-ти бальної системи оцінювання за 12-ма критеріями [18].

Таблиця 5.1 – Результати оцінювання науково-технічного рівня і комерційного потенціалу розробки експертами

Критерії	Експерт (ПІБ, посада)		
	1	2	3
	Бали:		
1. Технічна здійсненність концепції	5	5	5
2. Ринкові переваги (наявність аналогів)	1	2	2
3. Ринкові переваги (ціна продукту)	1	1	1
4. Ринкові переваги (технічні властивості)	1	2	2
5. Ринкові переваги (експлуатаційні витрати)	2	2	2
6. Ринкові перспективи (розмір ринку)	3	4	3
7. Ринкові перспективи (конкуренція)	3	2	3

8. Практична здійсненність (наявність фахівців)	5	5	5
9. Практична здійсненність (наявність фінансів)	3	4	3
10. Практична здійсненність (необхідність нових матеріалів)	5	5	5
11. Практична здійсненність (термін реалізації)	5	4	4
12. Практична здійсненність (розробка документів)	4	5	4
Сума балів	38	41	39
Середньоарифметична сума балів $СБ_с$	39,3		

За результатами розрахунків, наведених в таблиці 5.1, зробимо висновок щодо науково-технічного рівня і рівня комерційного потенціалу розробки. При цьому використаємо рекомендації, наведені в [18].

Загалом в роботі здійснено створення мікросервісу, що проводить другий фактор автентифікації через «Телеграм». Можливі розширення з використанням інших месенджерів (Slack, наприклад) та розширення функціоналу для комунікації з користувачем, створення нагадувань, сповіщень, додавання інформації про користувача до системи.

Відмінні ознаки розробленого мікросервісу: масштабуємість (можна дублювати мікросервіс для витримування більших навантажень), розширюваність (можна підключати інші месенджери, для проведення другого фактору), автономність (можливе використання для кількох проектів), захищеність (через власну імплементацію, можливе використання різних схем шифрування та збереження даних), зручність (немає необхідності встановлювати додаткові застосунки для користувача).

Згідно проведених досліджень рівень комерційного потенціалу розробки за темою «Розробка програмного забезпечення мікросервісу другого фактору автентифікації» становить 39,3 бала, що, відповідно до [18], свідчить про комерційну важливість проведення даних досліджень (рівень комерційного потенціалу розробки вище середнього).

5.2 Розрахунок узагальненого коефіцієнта якості розробки

Узагальнений коефіцієнт якості ($B_н$) для нового технічного рішення розрахуємо за формулою [19]**:**

$$B_н = \sum_{i=1}^{k} \alpha_i \cdot \beta_i \quad , \tag{5.1}$$

де k – кількість найбільш важливих технічних показників, які впливають на якість нового технічного рішення;

α_i – коефіцієнт, який враховує питому вагу і-го технічного показника в загальній якості розробки. Коефіцієнт α_i визначається експертним шляхом і при цьому має виконуватись умова $\sum_{i=1}^{k} \alpha_i = 1$;

β_i – відносне значення *і-го* технічного показника якості нової розробки.

Результати порівняння зведемо до таблиці 5.2.

Таблиця 5.2 – Порівняння основних параметрів розробки та аналога.

Показники (параметри)	Одиниця вимірю-вання	Аналог	Проектований продукт	Відношення параметрів нової розробки до аналога	Питома вага показника
Швидкість відправлення повідомлення про підтвердження	с	0,7	0,3	2,33	0,2
Швидкість комунікації з іншими мікросервісами	с	0,1	0,03	3,33	0,25
Завантаження процесора	%	50	20	2,5	0,2
Використовуваний об’єм оперативної пам’яті	МБ	290	120	2,42	0,1
Масштабованість	екземпляр	1	5	5	0,25

Узагальнений коефіцієнт якості ($B_н$) для нового технічного рішення складе:

$$B_н = \sum_{i=1}^{k} \alpha_i \cdot \beta_i = 2{,}33 \cdot 0{,}2 + 3{,}33 \cdot 0{,}25 + 2{,}5 \cdot 0{,}2 + 2{,}42 \cdot 0{,}1 + 5 \cdot 0{,}25 = 3{,}29.$$

Отже за технічними параметрами, згідно узагальненого коефіцієнту якості розробки, науково-технічна розробка переважає існуючі аналоги приблизно в 3,29 рази.

5.3 Розрахунок витрат на проведення науково-дослідної роботи

Витрати, пов'язані з проведенням науково-дослідної роботи на тему «Розробка програмного забезпечення мікросервісу другого фактору автентифікації», під час планування, обліку і калькулювання собівартості науково-дослідної роботи групуємо за відповідними статтями.

5.3.1 Витрати на оплату праці

Основна заробітна плата дослідників

Витрати на основну заробітну плату дослідників ($З_о$) розраховуємо у відповідності до посадових окладів працівників, за формулою [18]:

$$З_о = \sum_{i=1}^{k} \frac{M_{ni} \cdot t_i}{T_p}, \qquad (5.2)$$

де k – кількість посад дослідників залучених до процесу досліджень;

M_{ni} – місячний посадовий оклад конкретного дослідника, грн;

t_i – число днів роботи конкретного дослідника, дн.;

T_p – середнє число робочих днів в місяці, T_p=21 дні.

$$З_о = 17300{,}00 \cdot 35 / 21 = 28833{,}33 \text{ } грн.$$

Проведені розрахунки зведемо до таблиці.

Таблиця 5.3 – Витрати на заробітну плату дослідників

Найменування посади	Місячний посадовий оклад, грн	Оплата за робочий день, грн	Число днів роботи	Витрати на заробітну плату, грн
Керівник проекту	17300,00	823,81	35	28833,33
Інженер-розробник програмного забезпечення 1-ї категорії	16850,00	802,38	35	28083,33
Консультант (дослідник систем забезпечення безпеки автентифікації)	15500,00	738,10	7	5166,67
Технік 1-ї категорії	7550,00	359,52	21	7550,00
Всього				69633,33

Основна заробітна плата робітників

Витрати на основну заробітну плату робітників ($З_р$) за відповідними найменуваннями робіт НДР на тему «Розробка програмного забезпечення мікросервісу другого фактору автентифікації» розраховуємо за формулою:

$$З_р = \sum_{i=1}^{n} C_i \cdot t_i , \qquad (5.3)$$

де C_i – погодинна тарифна ставка робітника відповідного розряду, за виконану відповідну роботу, грн/год;

t_i – час роботи робітника при виконанні визначеної роботи, год.

Погодинну тарифну ставку робітника відповідного розряду C_i можна визначити за формулою:

$$C_i = \frac{M_M \cdot K_i \cdot K_c}{T_p \cdot t_{зм}} , \qquad (5.4)$$

де M_M – розмір мінімальної місячної заробітної плати, приймемо *Мм*=6700,00 грн;

K_i – коефіцієнт міжкваліфікаційного співвідношення (табл. Б.2, додаток Б) [18];

K_c – мінімальний коефіцієнт співвідношень місячних тарифних ставок;

T_p – середнє число робочих днів в місяці, приблизно T_p = 21 дн;

$t_{зм}$ – тривалість зміни, год.

$C_1 = 6700{,}00 \cdot 1{,}10 \cdot 1{,}65 / (21 \cdot 8) = 72{,}38$ *грн.*

$З_{р1}$ = 72,38 · 8,30 = 600,79 *грн.*

Таблиця 5.4 – Величина витрат на основну заробітну плату робітників

Найменування робіт	Тривалість роботи, год	Розряд роботи	Тарифний коефіцієнт	Погодинна тарифна ставка, грн	Величина оплати на робітника грн
Підготовка робочого місця розробника програмного забезпечення	8,30	2	1,10	72,38	600,79
Інсталяція програмного забезпечення середовища розробки і моделювання	6,20	3	1,35	88,83	550,78
Компіляція програмних блоків	7,50	5	1,70	111,87	839,00
Всього					1990,56

Додаткова заробітна плата дослідників та робітників

Додаткову заробітну плату розраховуємо як 10 … 12% від суми основної заробітної плати дослідників та робітників за формулою:

$$З_{дод} = (З_о + З_р) \cdot \frac{Н_{дод}}{100\%}, \quad (5.5)$$

де $Н_{дод}$ – норма нарахування додаткової заробітної плати. Приймемо 10%.

$З_{дод}$ = (69633,33 + 1990,56) · 10 / 100% = 7162,39 грн.

5.3.2 Відрахування на соціальні заходи

Нарахування на заробітну плату дослідників та робітників розраховуємо як 22% від суми основної та додаткової заробітної плати дослідників і робітників за формулою:

$$З_н = (З_о + З_р + З_{дод}) \cdot \frac{Н_{зп}}{100\%} \quad (5.6)$$

де $Н_{зп}$ – норма нарахування на заробітну плату. Приймаємо 22%.

Зн = (69633,33 + 1990,56 + 7162,39) · 22 / 100% = 17332,98 грн.

5.3.3 Сировина та матеріали

Витрати на матеріали (*М*), у вартісному вираженні розраховуються окремо по кожному виду матеріалів за формулою:

$$M = \sum_{j=1}^{n} H_j \cdot Ц_j \cdot K_j - \sum_{j=1}^{n} B_j \cdot Ц_{вj}\ , \qquad (5.7)$$

де H_j – норма витрат матеріалу *j*-го найменування, кг;

n – кількість видів матеріалів;

$Ц_j$ – вартість матеріалу *j*-го найменування, грн/кг;

K_j – коефіцієнт транспортних витрат, ($K_j = 1,1 \ldots 1,15$);

B_j – маса відходів *j*-го найменування, кг;

$Ц_{вj}$ – вартість відходів *j*-го найменування, грн/кг.

$M_1 = 2,0 \cdot 283,00 \cdot 1,1 - 0 \cdot 0 = 622,60$ *грн.*

Проведені розрахунки зведемо до таблиці.

Таблиця 5.5 – Витрати на матеріали

Найменування матеріалу, марка, тип, сорт	Ціна за 1 кг, грн	Норма витрат, кг	Величина відходів, кг	Ціна відходів, грн/кг	Вартість витраченого матеріалу, грн
Офісний папір А4 500	283,00	2,0	-	-	622,60
Папір для записів А5 250	154,00	3,0	-	-	508,20
Органайзер офісний	200,00	3,0	-	-	660,00
Набір канцелярський офісний	210,00	3,0	-	-	693,00
Картридж для принтера	980,00	1,0	-	-	1078,00
Диск оптичний CD-RW	23,00	2,0	-	-	50,60
Flesh-пам'ять 32 GB	410,00	1,0	-	-	451,00
Всього					4063,40

5.3.4 Розрахунок витрат на комплектуючі

Витрати на комплектуючі *($К_в$)*, які використовують при проведенні НДР на тему «Розробка програмного забезпечення мікросервісу другого фактору автентифікації», розраховуємо, згідно з їхньою номенклатурою, за формулою:

$$К_в = \sum_{j=1}^{n} Н_j \cdot Ц_j \cdot К_j \quad (5.8)$$

де $Н_j$ – кількість комплектуючих *j*-го виду, шт.;

$Ц_j$ – покупна ціна комплектуючих *j*-го виду, грн;

$К_j$ – коефіцієнт транспортних витрат, ($К_j$ *= 1,1 ... 1,15*).

$К_в = 1 \cdot 120,00 \cdot 1,1 = 132,00$ *грн.*

Проведені розрахунки зведемо до таблиці.

Таблиця 5.6 – Витрати на комплектуючі

Найменування комплектуючих	Кількість, шт.	Ціна за штуку, грн	Сума, грн
Дата кабель OTG USB 3.0 AF to Type-C 0.2m Cablexpert (A-OTG-CMAF3-01)	1	120,00	132,00
Дата кабель USB 2.0 AM to Type-C 2.0m black ColorWay (CW-CBUC008-BK)	1	160,00	176,00
Всього			308,00

5.3.5 Спецустаткування для наукових (експериментальних) робіт

Балансову вартість спецустаткування розраховуємо за формулою:

$$В_{спец} = \sum_{i=1}^{k} Ц_i \cdot С_{пр.i} \cdot К_i \ , \quad (5.9)$$

де $Ц_i$ – ціна придбання одиниці спецустаткування даного виду, марки, грн;

$С_{пр.i}$ –кількість одиниць устаткування відповідного найменування, які придбані для проведення досліджень, шт.;

$К_i$ – коефіцієнт, що враховує доставку, монтаж, налагодження устаткування тощо, ($К_i$ *= 1,10…1,12);*

k – кількість найменувань устаткування.

$B_{спец} = 8799,00 \cdot 1 \cdot 1,1 = 9678,90$ *грн.*

Отримані результати зведемо до таблиці:

Таблиця 5.7 – Витрати на придбання спецустаткування по кожному виду

Найменування устаткування	Кількість, шт	Ціна за одиницю, грн	Вартість, грн
Мобільний телефон Xiaomi Redmi Note 11 4/128GB Graphite Gray	1	8799,00	9678,90
Всього			9678,90

5.3.6 Програмне забезпечення для наукових (експериментальних) робіт

Балансову вартість програмного забезпечення розраховуємо за формулою:

$$B_{прг} = \sum_{i=1}^{k} Ц_{iпрг} \cdot C_{прг.i} \cdot K_i \ , \qquad (5.10)$$

де $Ц_{iпрг}$ – ціна придбання одиниці програмного засобу даного виду, грн;

$C_{прг.i}$ – кількість одиниць програмного забезпечення відповідного найменування, які придбані для проведення досліджень, шт.;

K_i – коефіцієнт, що враховує інсталяцію, налагодження програмного засобу тощо, ($K_i = 1,10…1,12$);

k – кількість найменувань програмних засобів.

$B_{прг} = 7860,00 \cdot 1 \cdot 1,11 = 8724,60$ *грн.*

Отримані результати зведемо до таблиці:

Таблиця 5.8 – Витрати на придбання програмних засобів по кожному виду

Найменування програмного засобу	Кількість, шт	Ціна за одиницю, грн	Вартість, грн
Прикладне програмне забезпечення розробки та моделювання	1	7860,00	8724,60
Всього			8724,60

5.3.7 Амортизація обладнання, програмних засобів та приміщень

В спрощеному вигляді амортизаційні відрахування по кожному виду обладнання, приміщень та програмному забезпеченню тощо, розраховуємо з використанням прямолінійного методу амортизації за формулою:

$$A_{обл} = \frac{Ц_б}{T_в} \cdot \frac{t_{вик}}{12}, \quad (5.11)$$

де $Ц_б$ – балансова вартість обладнання, програмних засобів, приміщень тощо, які використовувались для проведення досліджень, грн;

$t_{вик}$ – термін використання обладнання, програмних засобів, приміщень під час досліджень, місяців;

$T_в$ – строк корисного використання обладнання, програмних засобів, приміщень тощо, років.

$A_{обл} = (24650{,}00 \cdot 2) / (2 \cdot 12) = 2054{,}17$ *грн.*

Проведені розрахунки зведемо до таблиці.

Таблиця 5.9 – Амортизаційні відрахування по кожному виду обладнання

Найменування обладнання	Балансова вартість, грн	Строк корисного використання, років	Термін використання обладнання, місяців	Амортизаційні відрахування, грн
Персональний комп'ютер проведення розробки та моделювання	24650,00	2	2	2054,17
Робоче місце інженера-розробника програмного забезпечення	9200,00	5	2	306,67
Пристрої передачі даних	7510,00	4	2	312,92
Пристрій виводу інформації	6740,00	5	2	224,67
Оргтехніка	6750,00	4	2	281,25
Приміщення лабораторії	620000,00	25	2	4133,33
ОС Windows 11	8570,00	2	2	714,17

Прикладний пакет Microsoft Office 2019	7825,00	2	2	652,08
Всього				8679,25

5.3.8 Паливо та енергія для науково-виробничих цілей

Витрати на силову електроенергію (B_e) розраховуємо за формулою:

$$B_e = \sum_{i=1}^{n} \frac{W_{yi} \cdot t_i \cdot Ц_e \cdot K_{вni}}{\eta_i}, \quad (5.12)$$

де W_{yi} – встановлена потужність обладнання на визначеному етапі розробки, кВт;

t_i – тривалість роботи обладнання на етапі дослідження, год;

$Ц_e$ – вартість 1 кВт-години електроенергії, грн; приймемо Ц$_e$ = 6,20 грн;

$K_{вni}$ – коефіцієнт, що враховує використання потужності, $K_{вni} < 1$;

η_i – коефіцієнт корисної дії обладнання, $\eta_i < 1$.

$B_e = 0,25 \cdot 320,0 \cdot 6,20 \cdot 0,95 / 0,97 = 496,00$ *грн.*

Проведені розрахунки зведемо до таблиці.

Таблиця 5.10 – Витрати на електроенергію

Найменування обладнання	Встановлена потужність, кВт	Тривалість роботи, год	Сума, грн
Персональний комп'ютер проведення розробки та моделювання	0,25	320,0	496,00
Робоче місце інженера-розробника програмного забезпечення	0,12	300,0	223,20
Пристрої передачі даних	0,01	250,0	15,50
Пристрій виводу інформації	0,42	15,0	39,06
Оргтехніка	0,50	5,0	15,50
Всього			789,26

5.3.9 Службові відрядження

Витрати за статтею «Службові відрядження» розраховуємо як 20…25% від суми основної заробітної плати дослідників та робітників за формулою:

$$B_{св} = (З_о + З_р) \cdot \frac{Н_{св}}{100\%}, \quad (5.13)$$

де $Н_{св}$ – норма нарахування за статтею «Службові відрядження», приймемо $Н_{св}$ = 20%.

$В_{св}$ = (69633,33 + 1990,56) · 20 / 100% = 14324,78 грн.

5.3.10 Витрати на роботи, які виконують сторонні підприємства, установи і організації

Витрати на роботи, які виконують сторонні підприємства, установи і організації відсутні.

5.3.11 Інші витрати

Витрати за статтею «Інші витрати» розраховуємо як 50…100% від суми основної заробітної плати дослідників та робітників за формулою:

$$І_в = (З_о + З_р) \cdot \frac{Н_{ів}}{100\%}, \quad (5.14)$$

де $Н_{ів}$ – норма нарахування за статтею «Інші витрати», приймемо $Н_{ів}$ = 50%.

$І_в$ = (69633,33 + 1990,56) · 50 / 100% = 35811,95 грн.

5.3.12 Накладні (загальновиробничі) витрати

Витрати за статтею «Накладні (загальновиробничі) витрати» розраховуємо як 100…150% від суми основної заробітної плати дослідників та робітників за формулою:

$$В_{нзв} = (З_о + З_р) \cdot \frac{Н_{нзв}}{100\%}, \quad (5.15)$$

де $Н_{нзв}$ – норма нарахування за статтею «Накладні (загальновиробничі) витрати», приймемо $Н_{нзв}$ = 100%.

$В_{нзв}$ = (69633,33 + 1990,56) · 100 / 100% = 71623,89 грн.

Витрати на проведення науково-дослідної роботи на тему «Розробка програмного забезпечення мікросервісу другого фактору автентифікації» розраховуємо як суму всіх попередніх статей витрат за формулою:

$$B_{заг} = З_о + З_р + З_{дод} + З_н + М + К_в + В_{спец} + В_{прг} + А_{обл} + В_е + В_{св} + В_{сп} + І_в + В_{нзв}. \quad (5.16)$$

$B_{заг}$= 69633,33 +1990,56 +7162,39 +17332,98171 +4063,40 +308,00 +9678,90 + 8724,60 + 8679,25 +789,26 +14324,78 +0,00 +35811,95 +71623,89 = 250123,29 грн.

Загальні витрати *ЗВ* на завершення науково-дослідної (науково-технічної) роботи та оформлення її результатів розраховується за формулою:

$$ЗВ = \frac{B_{заг}}{\eta}, \quad (5.17)$$

де η - коефіцієнт, який характеризує етап (стадію) виконання науково-дослідної роботи, приймемо η=0,95.

ЗВ = 250123,29 / 0,95 = 263287,67 грн.

5.4 Розрахунок економічної ефективності науково-технічної розробки при її можливій комерціалізації потенційним інвестором

Результати дослідження проведені за темою «Розробка програмного забезпечення мікросервісу другого фактору автентифікації» передбачають комерціалізацію протягом 4-х років реалізації на ринку і відповідають випадку - розробка чи суттєве вдосконалення програмного засобу (програмного забезпечення, програмного продукту) для використання масовим споживачем.

В цьому випадку майбутній економічний ефект буде формуватися на основі таких даних:

ΔN – збільшення кількості споживачів продукту, у періоди часу, що аналізуються, від покращення його певних характеристик;

Показник	1-й рік	2-й рік	3-й рік	4-й рік
Збільшення кількості споживачів, осіб	3000	9000	15000	8500

N – кількість споживачів які використовували аналогічний продукт у році до впровадження результатів нової науково-технічної розробки, приймемо 35000 осіб;

$Ц_б$ – вартість програмного продукту у році до впровадження результатів розробки, приймемо 1020,00 грн;

$\pm\Delta Ц_о$ – зміна вартості програмного продукту від впровадження результатів науково-технічної розробки, приймемо зростання на 136,31 грн.

Можливе збільшення чистого прибутку у потенційного інвестора $\Delta П_i$ для кожного із 4-х років, протягом яких очікується отримання позитивних результатів від можливого впровадження та комерціалізації науково-технічної розробки, розраховуємо за формулою [18]:

$$\Delta П_i = (\pm\Delta Ц_о \cdot N + Ц_о \cdot \Delta N)_i \cdot \lambda \cdot \rho \cdot (1 - \frac{\vartheta}{100}), \quad (5.18)$$

де λ – коефіцієнт, який враховує сплату потенційним інвестором податку на додану вартість. У 2022 році ставка податку на додану вартість складає 20%, а коефіцієнт $\lambda = 0,8333$;

ρ – коефіцієнт, який враховує рентабельність інноваційного продукту). Приймемо ρ =35%;

ϑ – ставка податку на прибуток, який має сплачувати потенційний інвестор, у 2022 році $\vartheta = 18\%$;

Збільшення чистого прибутку 1-го року:

$\Delta П_1$ =(136,31·35000,00+1156,31·3000)·0,83·0,35·(1-0,18/100%)=1962797,99 грн.

Збільшення чистого прибутку 2-го року:

$\Delta П_2$ =(136,31·35000,00+1156,31·12000)·0,83·0,35·(1-0,18/100%)=4441799,44 грн.

Збільшення чистого прибутку 3-го року:

$\Delta П_3$ =(136,31·35000,00+1156,31·27000)·0,83·0,35·(1-0,18/100%)=8573468,52 грн.

Збільшення чистого прибутку 4-го року:

$\Delta П_4 = (136{,}31 \cdot 35000{,}00 + 1156{,}31 \cdot 35500) \cdot 0{,}83 \cdot 0{,}35 \cdot (1\text{-}0{,}18/100\%) = 10914747{,}66$ грн.

Приведена вартість збільшення всіх чистих прибутків *ПП*, що їх може отримати потенційний інвестор від можливого впровадження та комерціалізації науково-технічної розробки:

$$ПП = \sum_{i=1}^{T} \frac{\Delta П_i}{(1+\tau)^t}, \quad (5.19)$$

де $\Delta П_i$ – збільшення чистого прибутку у кожному з років, протягом яких виявляються результати впровадження науково-технічної розробки, грн;

T – період часу, протягом якого очікується отримання позитивних результатів від впровадження та комерціалізації науково-технічної розробки, роки;

τ – ставка дисконтування, за яку можна взяти щорічний прогнозований рівень інфляції в країні, τ =0,15;

t – період часу (в роках) від моменту початку впровадження науково-технічної розробки до моменту отримання потенційним інвестором додаткових чистих прибутків у цьому році.

$$ПП = 1962797{,}99/(1+0{,}15)^1 + 4441799{,}44/(1+0{,}15)^2 + 8573468{,}52/(1+0{,}15)^3 +$$
$$+ 10914747{,}66/(1+0{,}15)^4 = 1706780{,}86 + 3358638{,}52 + 5637194{,}72 + 6240542{,}40 =$$
$$= 16943156{,}50 \text{ грн.}$$

Величина початкових інвестицій PV, які потенційний інвестор має вкласти для впровадження і комерціалізації науково-технічної розробки:

$$PV = k_{інв} \cdot ЗВ, \quad (5.20)$$

де $k_{інв}$ – коефіцієнт, що враховує витрати інвестора на впровадження науково-технічної розробки та її комерціалізацію, приймаємо $k_{інв}$=2,1;

$ЗВ$ – загальні витрати на проведення науково-технічної розробки та оформлення її результатів, приймаємо 263287,67 грн.

$PV = k_{інв} \cdot ЗВ = 2{,}1 \cdot 263287{,}67 = 552904{,}11$ грн.

Абсолютний економічний ефект $E_{абс}$ для потенційного інвестора від можливого впровадження та комерціалізації науково-технічної розробки становитиме:

$$E_{абс} = ПП - PV \quad (5.21)$$

де $ПП$ – приведена вартість зростання всіх чистих прибутків від можливого впровадження та комерціалізації науково-технічної розробки, 16943156,50 грн;

PV – теперішня вартість початкових інвестицій, 552904,11 грн.

$E_{абс} = ПП - PV$ = 16943156,50 - 552904,11 = 16390252,39 грн.

Внутрішня економічна дохідність інвестицій $E_в$, які можуть бути вкладені потенційним інвестором у впровадження та комерціалізацію науково-технічної розробки:

$$E_в = \sqrt[Тж]{1 + \frac{E_{абс}}{PV}} - 1, \quad (5.22)$$

де $E_{абс}$ – абсолютний економічний ефект вкладених інвестицій, 16390252,39 грн;

PV – теперішня вартість початкових інвестицій, 552904,11 грн;

$T_ж$ – життєвий цикл науково-технічної розробки, тобто час від початку її розробки до закінчення отримування позитивних результатів від її впровадження, 4 роки.

$$E_в = \sqrt[Тж]{1 + \frac{E_{абс}}{PV}} - 1 = (1+16390252,39/552904,11)^{1/4} = 1,35.$$

Мінімальна внутрішня економічна дохідність вкладених інвестицій $\tau_{мін}$:

$$\tau_{мін} = d + f, \quad (5.23)$$

де d – середньозважена ставка за депозитними операціями в комерційних банках; в 2022 році в Україні d =0,11;

f – показник, що характеризує ризикованість вкладення інвестицій, приймемо 0,3.

$\tau_{мін}$ = 0,11+0,3 = 0,41 < 1,35 свідчить про те, що внутрішня економічна дохідність інвестицій $E_в$, вища мінімальної внутрішньої дохідності. Тобто інвестувати в науково-дослідну роботу за темою «Розробка програмного забезпечення мікросервісу другого фактору автентифікації» доцільно.

Період окупності інвестицій $T_{ок}$ які можуть бути вкладені потенційним інвестором у впровадження та комерціалізацію науково-технічної розробки:

$$T_{ок} = \frac{1}{E_в}, \tag{5.24}$$

де $E_в$ – внутрішня економічна дохідність вкладених інвестицій.

$T_{ок}$ = 1 / 1,35 = 0,74 р.

$T_{ок}$ < 3-х років, що свідчить про комерційну привабливість науково-технічної розробки і може спонукати потенційного інвестора профінансувати впровадження даної розробки та виведення її на ринок.

5.5 Висновки

Згідно проведених досліджень рівень комерційного потенціалу розробки за темою «Розробка програмного забезпечення мікросервісу другого фактору автентифікації» становить 39,3 бала, що, свідчить про комерційну важливість проведення даних досліджень (рівень комерційного потенціалу розробки вище середнього).

При оцінюванні за технічними параметрами, згідно узагальненого коефіцієнту якості розробки, науково-технічна розробка переважає існуючі аналоги приблизно в 3,29 рази.

Також термін окупності становить 0,74 р., що менше 3-х років, що свідчить про комерційну привабливість науково-технічної розробки і може спонукати потенційного інвестора профінансувати впровадження даної розробки та виведення її на ринок.

Отже можна зробити висновок про доцільність проведення науково-дослідної роботи за темою «Розробка програмного забезпечення мікросервісу другого фактору автентифікації».

ВИСНОВКИ

В процесі виконання магістерської кваліфікаційної роботи отримані такі результати.

У першому розділі представлено опис та аналіз наявних методів автентифікації, де описано їх переваги та недоліки. Тож, після огляду наявних методів автентифікації зроблено висновок, що найоптимальнішим вибором для удосконалення методу автентифікації є – двофакторна Автентифікація через програмне забезпечення. Тому що, цей метод зараз активно розвивається, він має досить незначні мінуси на фоні значущих плюсів. До того ж, він дає можливість безпечніше передавати дані, тим самим збільшуючи захищеність облікового запису користувача.

У другому розділі було розглянуто програми, які використовують двофакторну автентифікацію. Було розглянуто їх структуру, переваги та недоліки кожного виду автентифікації. Проаналізувавши отримані дані, можна сказати, що удосконалення методу автентифікації на основі засобів месенджера Telegram, це досить нова ідея – яка не має прямих аналогів, але має загальні принципи двофакторної автентифікації через програмне забезпечення. Основним недоліком використання програмного забезпечення є його встановлення, але у випадку Telegram, він і так встановлений вже в багатьох користувачів. При тому, його можна використовувати не тільки для двофакторної автентифікації в даному додатку, а й в інших та за його прямим призначенням – як месенджер. Також месенджер є досить захищеним, так що хвилюватись за компрометацію даних не доводиться. Головним плюсом є те, що дана реалізація може бути розширена, залучати також інші месенджери та створювати додатковий функціонал, такий як – комунікація з користувачем.

У третьому розділі було розглянуто та ключові інструменти для розробки. Аналоги було порівняно, а вибір аргументовано. Було обрано тип архітектури – мікросервісна. Мікросервісна архітектура є більш доцільною, так як головною перевагою є автономність і можливість використання у різних проектах. Також обрано інтерфейс для передачі даних – gRPC, через швидкість та портативність останнього. Та обрано інструмент для зв’язку з месенджером Telegram – Bot Api, який є спрощеною, але доречною в даному видку версією API.

У четвертому розділі було описано етапи розробки програмного забезпечення другого фактору автентифікації. Детально описана структура проекту, підключення до Bot API для комунікації з Telegram, підключення засобу для мікросервісної комунікації gRPC. Також було створено три діаграми пакетів, одна з яких загальна, а ще дві уточнюючі та діаграму класів. Було описано реалізацію другого фактору автентифікації та відправлення повідомлень. До них було створено діаграму прецедентів.

У економічному розділі було розраховано доцільність проведення проведення науково-дослідної роботи. Згідно проведених досліджень рівень комерційного потенціалу розробки за темою «Розробка програмного забезпечення мікросервісу другого фактору автентифікації» становить 39,3 бала, що, свідчить про комерційну важливість проведення даних досліджень (рівень комерційного потенціалу розробки вище середнього).

При оцінюванні за технічними параметрами, згідно узагальненого коефіцієнту якості розробки, науково-технічна розробка переважає існуючі аналоги приблизно в 3,29 рази.

Також термін окупності становить 0,74 р., що менше 3-х років, що свідчить про комерційну привабливість науково-технічної розробки і може спонукати потенційного інвестора профінансувати впровадження даної розробки та виведення її на ринок.

Отже можна зробити висновок про доцільність проведення науково-дослідної роботи за темою «Розробка програмного забезпечення мікросервісу другого фактору автентифікації».

Отримані в результаті виконання магістерської роботи результати дозволяють зменшити ймовірність витоку даних з додатку Time Tracking завдяки створенню програмного забезпечення другого фактору авторизації.

СПИСОК ЛІТЕРАТУРИ

1. ДОСЛІДЖЕННЯ РОЗРОБКИ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ДРУГОГО ФАКТОРУ АВТЕНТИФІКАЦІЇ З ВИКОРИСТАННЯМ МЕСЕНДЖЕРУ TELEGRAM Матеріал із статті. URL: https://conferences.vntu.edu.ua/index.php/mccs/mccs2022/paper/view/16486 (дата звернення: 10.11.2022).

2. Різниця між аутентифікацією та авторизацією. Матеріал із статті. URL: https://uk.gadget-info.com/difference-between-authentication (дата звернення: 01.10.2022).

3. Автентифікація. Матеріал із статті. URL: https://www.unisender.com/ (дата звернення: 01.10.2022).

4. Звичайна та дайджест-автентифікація. Матеріал із статті. URL: https://learn.microsoft.com/kk-kz/dotnet/framework/network-programming/basic-and-digest-authentication (дата звернення: 01.10.2022).

5. HTTP автентифікація. Матеріал із статті. URL: https://developer.mozilla.org/docs/Web/HTTP/Authentication (дата звернення: 01.10.2022).

6. Отримання цифрового сертифіката та створення цифрового підпису. Матеріал із статті. URL: https://support.microsoft.com (дата звернення: 01.10.2022).

7. Автентифікація у cookies. Матеріал із статті. URL: https://studfile.net/preview/9429366/page:7/ (дата звернення: 01.10.2022).

8. Двофакторна автентифікація для безпеки облікового запису - що це, її види та як використовувати. Матеріал із статті. URL: https://ssl.com.ua/blog/what-is-2fa/ (дата звернення: 01.10.2022).

9. ПриватБанк. Матеріал із статті. URL: https://privatbank.ua/about (дата звернення: 02.10.2022).

10. Що таке Steam та як ним користуватися. Матеріал із статті. URL: https://tsn.ua/cybersport/chto-takoe-steam-i-kak-im-polzovatsya-1719481.html (дата звернення: 02.10.2022).

11. All of Google, working for you. Матеріал із статті. URL: https://www.google.com/intl/en-GB/account/about/#better (дата звернення: 02.10.2022).

12. Pattern: Monolithic Architecture. Матеріал із статті. URL: https://microservices.io/patterns/monolithic.html (дата звернення: 10.11.2022).

13. Microservices architecture. Матеріал із статті. URL: https://www.atlassian.com/microservices/microservices-architecture (дата звернення: 10.11.2022).

14. gRPC vs REST APIs — Key Differences. Матеріал із статті. URL: https://blog.getambassador.io/grpc-vs-rest-apis-key-differences-800ecc66d173 (дата звернення: 12.11.2022).

15. Event Driven vs REST in Microservice Architecture. Матеріал із статті. URL: https://www.stackchief.com/blog/Event%20Driven%20vs%20REST%20in%20Microservice%20Architecture (дата звернення: 12.11.2022).

16. Telegram APIs. Матеріал із статті. URL: https://core.telegram.org/#getting-started (дата звернення: 13.11.2022).

17. Ланцюжок обов'язків. Матеріал із статті. URL: https://refactoring.guru/uk/design-patterns/chain-of-responsibility (дата звернення: 20.11.2022).

18. Методичні вказівки до виконання економічної частини магістерських кваліфікаційних робіт / Уклад. : В. О. Козловський, О. Й. Лесько, В. В. Кавецький. – Вінниця : ВНТУ, 2021. – 42 с.

19. Кавецький В. В. Економічне обґрунтування інноваційних рішень: практикум / В. В. Кавецький, В. О. Козловський, І. В. Причепа – Вінниця : ВНТУ, 2016. – 113 с.

ДОДАТКИ

Додаток А (обов'язковий)

Технічне завдання

ЗАТВЕРДЖЕНО

Завідувач кафедри АІІТ

_____

(прізвище та ініціали)

«____» _____2022

ТЕХНІЧНЕ ЗАВДАННЯ

на магістерську кваліфікаційну роботу

«Розробка програмного забезпечення мікросервісу другого фактору автентифікації»

08-02.МКР.000.07.000 ТЗ

Виконав: студент 2 курсу, групи 1ІСТ-21м спеціальності 126 – Інформаційні системи та технології

(шифр і назва спеціальності)

Керівник: к.т.н., доцент кафедри АІІТ Кулик Я. А.

(прізвище та ініціали)

Вінниця ВНТУ 2022

1. Назва та галузь застосування

Магістерська кваліфікаційна робота: «Розробка програмного забезпечення мікросервісу другого фактору автентифікації». Галузь застосування – інформаційні технології.

2. Підстава для проведення розробки

Тема магістерської кваліфікаційна роботи затверджена наказом по ВНТУ №___ від «__» _____ 2022 р.

3. Мета та призначення розробки

Метою магістерської кваліфікаційної роботи є зменшення ймовірності витоку даних з додатку Time Tracking.

4. Джерела розробки

- ISO/IEC 2382-1:1993, Information technology – Vocabulary – Part 1: Fundamental terms.
- ISO/IEC 9126-1:2001, Software engineering – Product quality – Part 1: Quality model.
- ISO/IEC TR 9126-2:2003, Software engineering – Product quality – Part 2: External metrics.
- ISO/IEC TR 9126-3:2003, Software engineering – Product quality – Part 3: Internal metrics.

5. Показники призначення

Необхідні вхідні дані для коректної роботи програми:

- стабільний зв'язок з мережею Інтернет;
- встановлений додаток «Telegram» чи його веб-версія;
- наявність запасу пам’яті на девайсі;
- посилання на чат-бот.

Результати роботи програми:

- отримання клієнтом повідомлень;
- здійснення другого фактору автентифікації завдяки месенджеру Telegram;

– здійснення розсилки на всіх користувачів.

6. Економічні показники

– Прогнозовані витрати на розробку – не більше 20 тис. грн;

– Абсолютна ефективність розробки – не менше 60 тис. грн;

– Термін окупності витрат для виробника – не більше 3 років.

7. Стадії розробки

1. Розділ 1 « Дослідження способів та видів автентифікації» має бути виконаний до 10.10.2022.

2. Розділ 2 «Аналіз видів двофакторної автентифікації» має бути виконаний до 20.10.2022.

3. Розділ 3 «Інструменти для розробки програмного забезпечення» має бути виконаний до 20.11.2022.

4. Розділ 4 «Розробка програмного забезпечення» має бути виконаний до 30.11.2022.

5. Економічний розділ має бути виконаний до 03.12.2022.

8. Порядок контролю та приймання

1. Рубіжний контроль. Провести до 05.12.2022.

2. Попередній захист магістерської кваліфікаційної роботи. Провести до 12.12.2020.

3. Захист магістерської кваліфікаційної роботи. Провести в період з 19.12.2022 до 23.12.2022.

Додаток Б (обов’язковий)

ІЛЮСТРАТИВНА ЧАСТИНА

РОЗРОБКА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ МІКРОСЕРВІСУ ДРУГОГО ФАКТОРУ АВТЕНТИФІКАЦІЇ

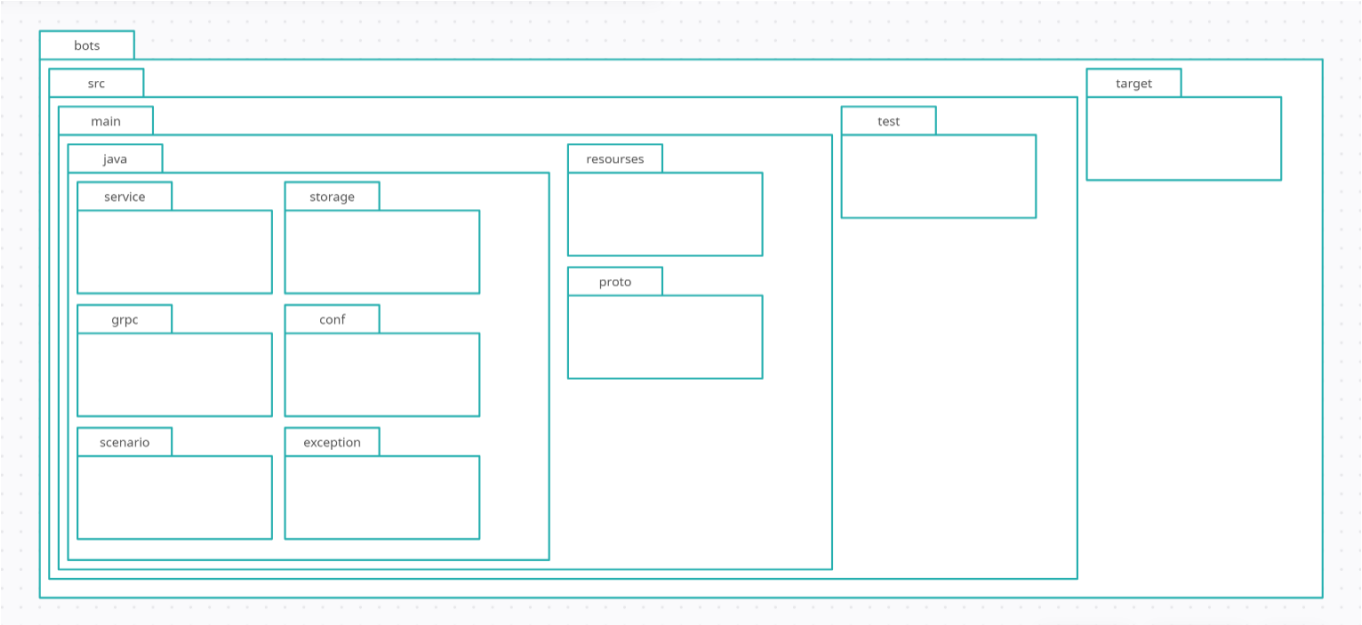

Рисунок Б.1 – Діаграма пакетів для відображення структури програми.

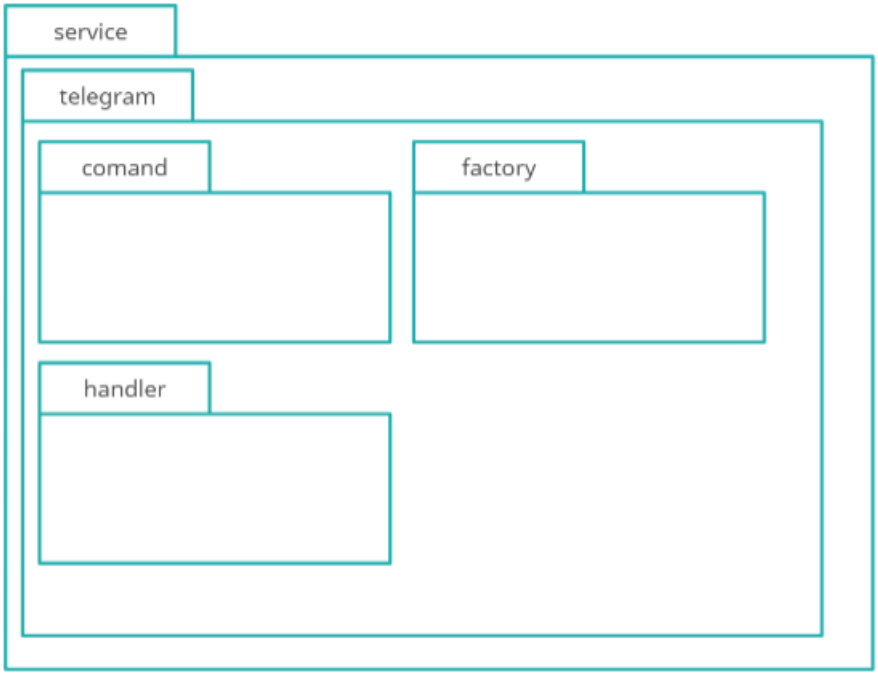

Рисунок Б.2 – Діаграма пакетів для відображення детальнішої структури пакету service.

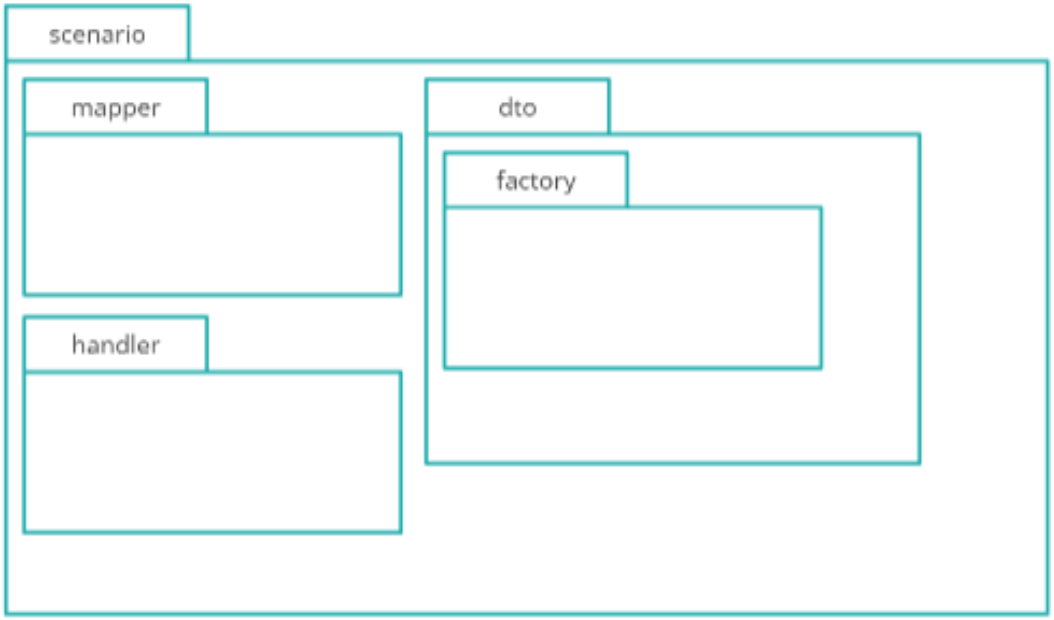

Рисунок Б.3 – Діаграма пакетів для відображення детальнішої структури пакету scenario.

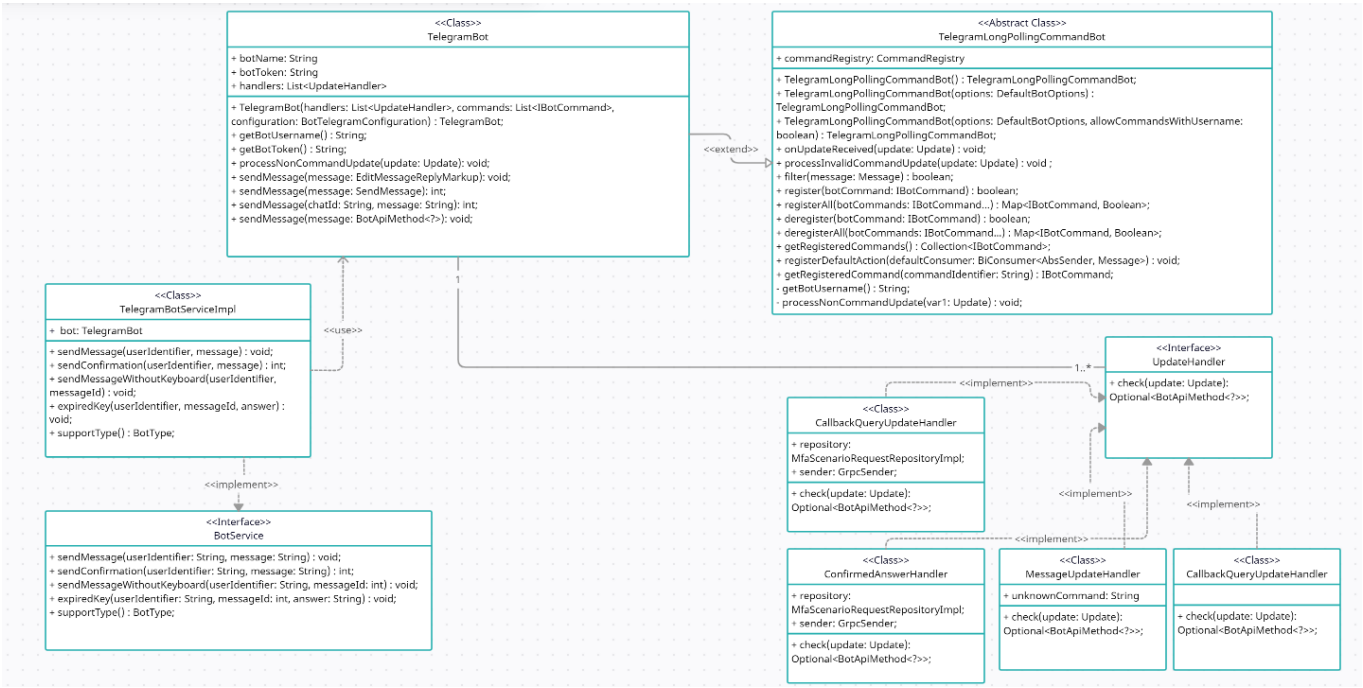

Рисунок Б.4 – Діаграма класів для відображення взаємозв’язків для модуля Telegram.

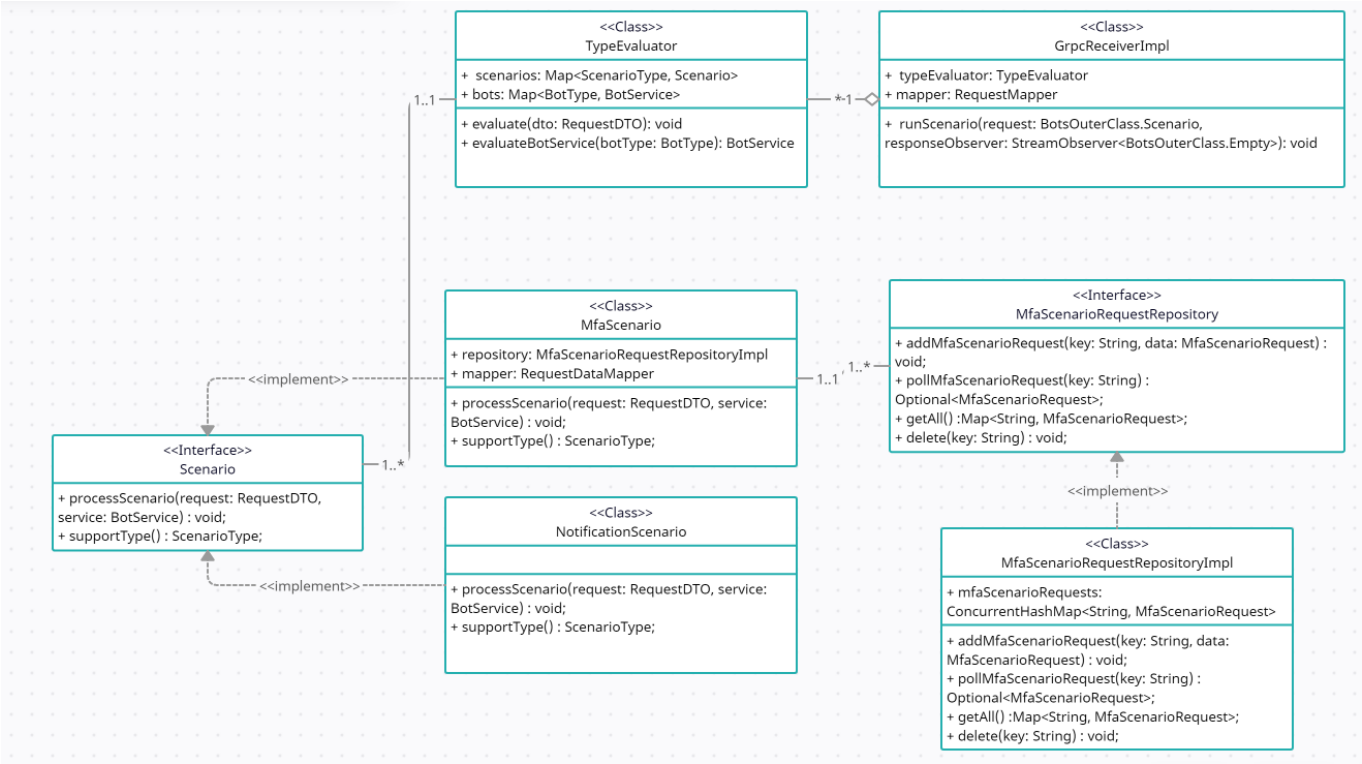

Рисунок Б.5 – Діаграма класів для відображення взаємозв’язків для процесу отримання повідомлення.

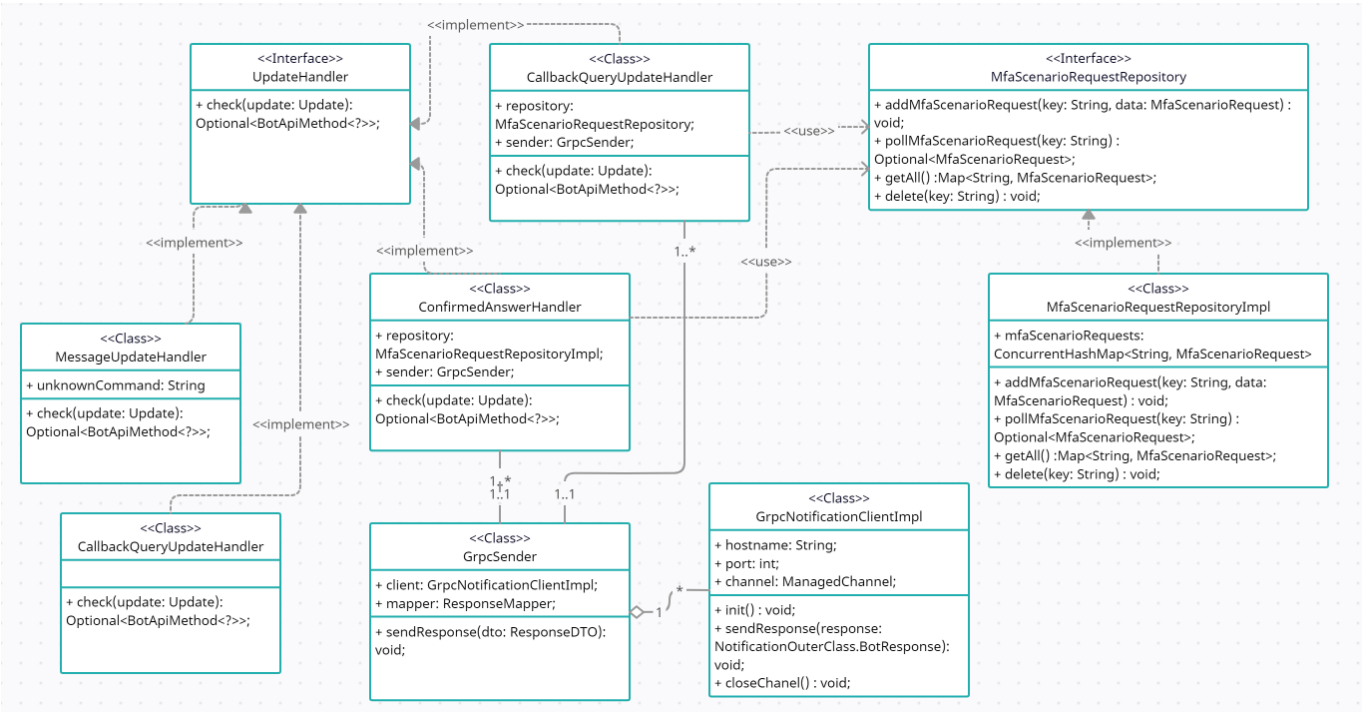

Рисунок Б.6 – Діаграма класів для відображення взаємозв’язків для процесу обробки і формування відповіді.

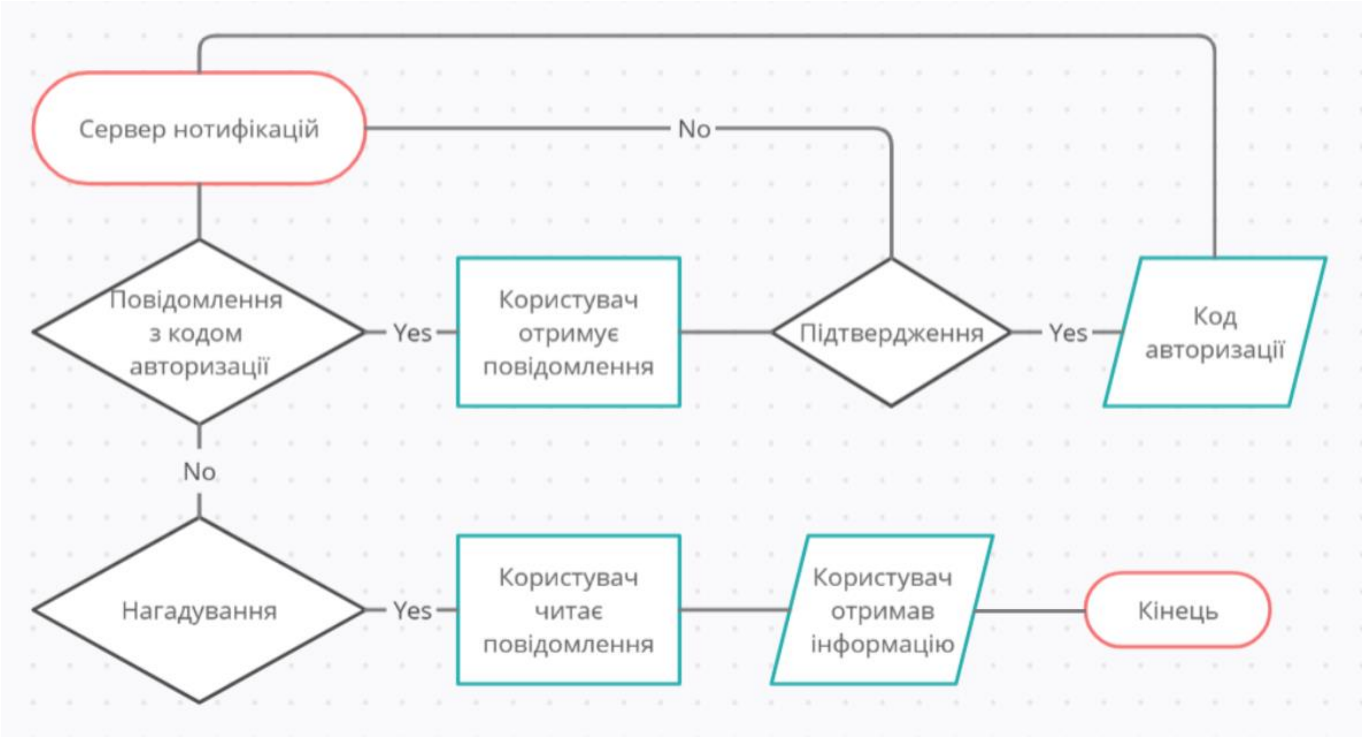

Рисунок Б.7 – Схема комунікації з сервером нотифікацій.

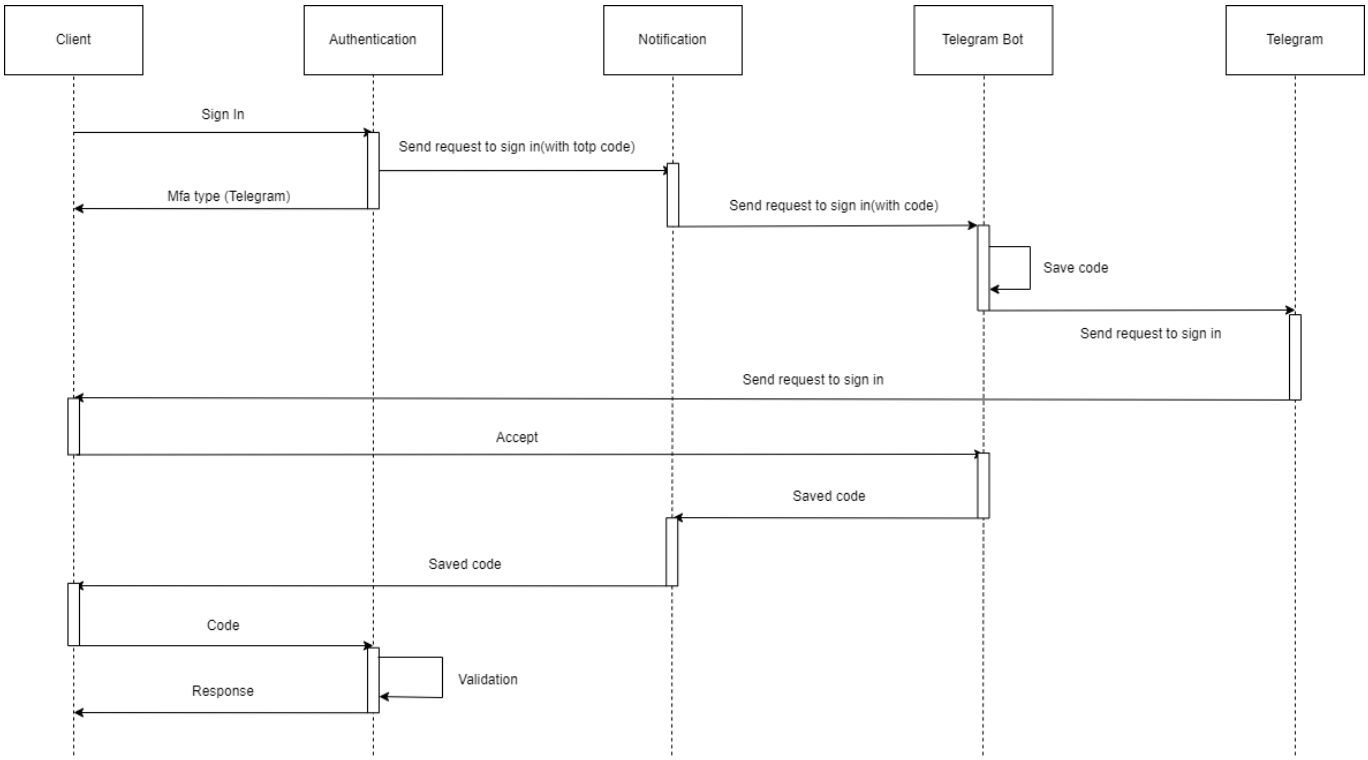

Рисунок Б.8 – UML-діаграма послідовності для процесу автентифікації.

Додаток В (обов'язковий)

Лістинг коду

```
/**
 * Creates channel for remote invoking methods of Notification`s service
 */
@Slf4j
@Component
public class GrpcNotificationClientImpl {

    @Value("${grpc.service.notification.host}")
    private String hostname;
    @Value("${grpc.service.notification.port}")
    private int port;

    private ManagedChannel channel;

    @PostConstruct
    void init() {
        channel = ManagedChannelBuilder.forAddress(hostname, port)
                .usePlaintext()
                .build();
    }

    public void sendResponse(NotificationOuterClass.BotResponse response) {
        NotificationGrpc.NotificationBlockingStub stub
                = NotificationGrpc.newBlockingStub(channel);

        log.info("Notification service response: {}", stub.botsReply(response).toString());
    }

    @PreDestroy
    void closeChanel() {
        channel.shutdownNow();
        log.info("Chanel shutdown: {}", channel.isShutdown());
    }
}


/**
 * runScenario receives request from Notification service,
 * converts it into POJO, and passes it into ScenarioEngine
 */
@Slf4j
@Component
@RequiredArgsConstructor
public class GrpcReceiverImpl extends BotsGrpc.BotsImplBase {

    private final TypeEvaluator typeEvaluator;
    private final RequestMapper mapper;

    @Override
    public void runScenario(BotsOuterClass.Scenario request, StreamObserver<BotsOuterClass.Empty> responseObserver) {
        RequestDTO dto = mapper.toDTO(request);
        log.info("Received data from notification service: {}", dto);

        typeEvaluator.evaluate(dto);

        responseObserver.onNext(createEmptyResponse());
```

```
        responseObserver.onCompleted();
    }

    private BotsOuterClass.Empty createEmptyResponse() {
        return BotsOuterClass.Empty.newBuilder().build();
    }
}


@Slf4j
@Component
@RequiredArgsConstructor
public class GrpcSender {

    private final GrpcNotificationClientImpl client;
    private final ResponseMapper mapper;

    public void sendResponse(ResponseDTO dto) {
        log.info("Bot response to Notification service: {}", dto.toString());

        client.sendResponse(mapper.toEntity(dto));
    }
}


/**
 * Creates server for ability Notification`s service to remote invoking methods from GrpcReceiverImpl
 */
@Slf4j
@Component
@RequiredArgsConstructor
public class GrpcServer {

    private final Server server;

    @PostConstruct
    private void startServer() {
        try {
            server.start();
            log.info("Server for connection to Notification service starts");
        } catch (IOException e) {
            e.printStackTrace();
        }
    }

    @PreDestroy
    private void stopServer(){
        server.shutdown();
        log.info("Server for connection to Notification service stopped");
    }
}


public interface Scenario {

    void processScenario(RequestDTO request, BotService service);

    ScenarioType supportType();
}


@Slf4j
@Component
```

```
@RequiredArgsConstructor
public class MfaScenario implements Scenario {

    private final MfaScenarioRequestRepositoryImpl repository;
    private final RequestDataMapper mapper;

    @Override
    public void processScenario(RequestDTO dto, BotService service) {
        String userIdentifier = dto.getUserIdentifier();

        deleteOldMessageIfPresent(service, userIdentifier);

        MfaScenarioRequest mfaScenarioRequest = mapper.toMfaScenarioRequest(dto);

        int messageId = service.sendConfirmation(userIdentifier, dto.getMessage());
        mfaScenarioRequest.setMessageId(messageId);

        repository.addMfaScenarioRequest(userIdentifier, mfaScenarioRequest);
        log.info("{} added in storage: {}", userIdentifier, mfaScenarioRequest);
    }

    /**
     * Checks if in repository exists active message with the came userIdentifier
     * If exists, delete it and hide the buttons
     */
    private void deleteOldMessageIfPresent(BotService service, String userIdentifier) {
        repository.pollMfaScenarioRequest(userIdentifier).ifPresent(requestData ->
                service.sendMessageWithoutKeyboard(userIdentifier, requestData.getMessageId()));
    }

    @Override
    public ScenarioType supportType() {
        return ScenarioType.MFA;
    }
}


@Slf4j
@Component
@RequiredArgsConstructor
public class NotificationScenario implements Scenario {

    @Override
    public void processScenario(RequestDTO dto, BotService service) {
        String userIdentifier = dto.getUserIdentifier();

        service.sendMessage(userIdentifier, dto.getMessage());
    }

    @Override
    public ScenarioType supportType() {
        return ScenarioType.NOTIFICATION;
    }
}


/**
 * Chooses a scenario and bot for executing
 */
@Component
@RequiredArgsConstructor
public class TypeEvaluator {

    private final Map<ScenarioType, Scenario> scenarios;
```

```
    private final Map<BotType, BotService> bots;

    public void evaluate(RequestDTO dto) {
        BotService bot = getBotService(dto);
        Scenario scenario = getScenario(dto);
        scenario.processScenario(dto, bot);
    }

    public BotService evaluateBotService(BotType botType){
        return getBotService(botType);
    }

    private Scenario getScenario(RequestDTO request) {
        return Optional.ofNullable(scenarios.get(request.getType()))
                .orElseThrow(() -> new UnsupportedOperationException("Scenario type \"" + request.getType() + "\" is not
supported"));
    }

    private BotService getBotService(RequestDTO request) {
        return getBotService(request.getBotType());
    }

    private BotService getBotService(BotType botType) {
        return Optional.ofNullable(bots.get(botType))
                .orElseThrow(() -> new UnsupportedOperationException("Bot type \"" + botType + "\" is not supported"));
    }
}


@Component
@RequiredArgsConstructor
public class RequestMapper {

    private final ScenarioDataMapper scenarioDataMapper;
    private final BotTypeMapper botTypeMapper;
    private final ScenarioTypeMapper scenarioTypeMapper;

    public RequestDTO toDTO(BotsOuterClass.Scenario request) {
        RequestDTO dto = new RequestDTO();
        dto.setId(request.getId());
        dto.setType(scenarioTypeMapper.toScenarioType(request.getType()));
        dto.setBotType(botTypeMapper.toBotType(request.getBotType()));
        dto.setUserIdentifier(request.getUserIdentifier());
        dto.setData(scenarioDataMapper.toDTO(request.getData()));
        return dto;
    }
}


@Component
@RequiredArgsConstructor
public class ResponseMapper {
    private final BotTypeMapper botTypeMapper;

    public NotificationOuterClass.BotResponse toEntity(ResponseDTO dto) {
        return NotificationOuterClass.BotResponse.newBuilder()
                .setId(dto.getId())
                .setUserIdentifier(dto.getUserIdentifier())
                .setBotType(botTypeMapper.toBotTypeGrpc(dto.getBotType()))
                .putAllParameters(dto.getParameters())
                .build();
    }
}
```

```
@Component
public class ScenarioDataMapper {

    public ScenarioData toDTO(BotsOuterClass.ScenarioData data) {
        ScenarioData dto = new ScenarioData();
        dto.setMessage(data.getMessage());
        dto.setParameters(data.getParametersMap());
        return dto;
    }
}
```

```
@Slf4j
@Component
public class ExpiredKeyVerifier {

    @Value("${storage.time-to-live-seconds}")
    private Duration timeToLiveSeconds;

    public boolean isExpired(String key, MfaScenarioRequest mfaScenarioRequest) {
        if (mfaScenarioRequest.isAfter(timeToLiveSeconds)) {
            log.warn("Time out of key {} with data {}", key, mfaScenarioRequest);
            return true;
        }
        return false;
    }
}
```

```
@Slf4j
@Component
@RequiredArgsConstructor
public class TimeOutHandler {

    private final MfaScenarioRequestRepository repository;
    private final ExpiredKeyVerifier verifier;
    private final TypeEvaluator typeEvaluator;

    @Value("${answer.command.no-action}")
    private String answer;

    @Scheduled(cron = "*/10 * * * * *") //every 10 sec
    public void evictExpiredConfirmation() {
        repository.getAll().forEach((key, mfaScenarioRequest) -> {
            if (verifier.isExpired(key, mfaScenarioRequest)) {
                BotService service = typeEvaluator.evaluateBotService(mfaScenarioRequest.getBotType());
                service.expiredKey(key, mfaScenarioRequest.getMessageId(), answer);

                repository.delete(key);
            }
        });
    }
}
```

```
@Data
@AllArgsConstructor
@NoArgsConstructor
@Builder
public class RequestDTO {
    private String id;
```

```
    private ScenarioType type;
    private BotType botType;
    private ScenarioData data;
    private String userIdentifier;

    public String getMessage() {
        return data.getMessage();
    }
}


@Data
@AllArgsConstructor
@NoArgsConstructor
@Builder
public class ResponseDTO {
    private String id;
    private String userIdentifier;
    private BotType botType;
    @Builder.Default
    private Map<String, String> parameters = new HashMap<>();
}


@Data
@AllArgsConstructor
@NoArgsConstructor
@Builder
public class ScenarioData {
    private String message;
    private Map<String, String> parameters;
}


@AllArgsConstructor(access = AccessLevel.PRIVATE)
public class ResponseDTOFactory {

    public static ResponseDTO createTelegram(String id, String userIdentifier) {
        return ResponseDTO.builder()
                .id(id)
                .userIdentifier(userIdentifier)
                .botType(BotType.TELEGRAM)
                .build();
    }

    public static ResponseDTO createConfirmedTelegram(String id, String userIdentifier, String otp) {
        return ResponseDTO.builder()
                .id(id)
                .botType(BotType.TELEGRAM)
                .userIdentifier(userIdentifier)
                .parameters(Map.of(RequestParams.OTP.getValue(), otp, RequestParams.IS_CANCELED.getValue(), "false"))
                .build();
    }

    public static ResponseDTO createDeclinedTelegram(String id, String userIdentifier) {
        return ResponseDTO.builder()
                .id(id)
                .botType(BotType.TELEGRAM)
                .userIdentifier(userIdentifier)
                .parameters(Map.of(RequestParams.IS_CANCELED.getValue(), "true"))
                .build();
    }
}
```

```
/**
 * Interface for determining main functions of different kinds of bots needed for execution scenarios
 */
public interface BotService {

    void sendMessage(String userIdentifier, String message);

    int sendConfirmation(String userIdentifier, String message);

    void sendMessageWithoutKeyboard(String userIdentifier, int messageId);

    void expiredKey(String userIdentifier, int messageId, String answer);

    BotType supportType();
}


/**
 * Overrides needed method for Telegram's work
 * Receives responses of user and chooses further interaction scenario
 */
@Slf4j
@Component
public class TelegramBot extends TelegramLongPollingCommandBot {

    private final String botName;
    private final String botToken;

    private final List<UpdateHandler> handlers;

    public TelegramBot(List<UpdateHandler> handlers, List<IBotCommand> commands, BotTelegramConfiguration
configuration) {
        this.botName = configuration.getTelegramName();
        this.botToken = configuration.getTelegramToken();
        this.handlers = handlers;
        commands.forEach(this::register);
    }

    @Override
    public String getBotUsername() {
        return botName;
    }

    @Override
    public String getBotToken() {
        return botToken;
    }

    @Override
    public void processNonCommandUpdate(Update update) {
        handlers.forEach(handler -> handler.check(update).ifPresent(this::sendMessage));
    }

    public void sendMessage(EditMessageReplyMarkup message) {
        try {
            execute(message);
        } catch (TelegramApiException e) {
            log.error("Telegram error in sending message in chat \"{}\":", message.getChatId(), e);
            throw new SendMessageToBotException(e);
        }
    }
```

```
    public int sendMessage(SendMessage message) {
        try {
            return execute(message).getMessageId();
        } catch (TelegramApiException e) {
            log.error("Telegram error in sending message \"{}\" in chat \"{}\":", message.getText(), message.getChatId(), e);
            throw new SendMessageToBotException(e);
        }
    }

    public int sendMessage(String chatId, String message) {
        SendMessage answer = TelegramMessageFactory.createMessage(chatId, message);
        return sendMessage(answer);
    }

    public void sendMessage(BotApiMethod<?> message) {
        try {
            execute(message);
        } catch (TelegramApiException e) {
            log.error("Telegram error in sending message \"{}\"", message, e);
            throw new SendMessageToBotException(e);
        }
    }
}


/**
 * Determines main functions of Telegram bot
 */
@Slf4j
@Component
@RequiredArgsConstructor
public class TelegramBotServiceImpl implements BotService {

    private final TelegramBot bot;

    @Override
    public void sendMessage(String userIdentifier, String message) {
        bot.sendMessage(userIdentifier, message);
    }

    @Override
    public BotType supportType() {
        return BotType.TELEGRAM;
    }

    @Override
    public int sendConfirmation(String userIdentifier, String message) {
        SendMessage confirmation = TelegramMessageFactory.createMessage(userIdentifier, message);
        confirmation.setReplyMarkup(KeyboardFactory.confirmationKeyBoard());

        return bot.sendMessage(confirmation);
    }

    @Override
    public void expiredKey(String userIdentifier, int messageId, String answer) {
        sendMessageWithoutKeyboard(userIdentifier, messageId);
        sendMessage(userIdentifier, answer);
    }

    @Override
    public void sendMessageWithoutKeyboard(String userIdentifier, int messageId) {
        EditMessageReplyMarkup message = KeyboardFactory.deleteKeyBoard(userIdentifier, messageId);
        bot.sendMessage(message);
    }
```

```
}

/**
 * Created for hiding part of handle exception in sending message
 */
@Slf4j
abstract class ServiceCommand extends BotCommand {

    ServiceCommand(String identifier, String description) {
        super(identifier, description);
    }

    void sendAnswer(AbsSender absSender, Long chatId, String commandName, String userName, String text) {
        SendMessage message = TelegramMessageFactory.createMessage(chatId, text);
        try {
            absSender.execute(message);
        } catch (TelegramApiException e) {
            log.error("Telegram error in processing {} command for user \"{}\":", commandName, userName, e);
            throw new SendMessageToBotException(e);
        }
    }
}


/**
 * Created for sending messages for "/start" command
 * It receives needed data, process it and invoke sendAnswer() in parent class ServiceCommand
 */
@Slf4j
@Component
public class StartCommand extends ServiceCommand {

    private final String answer;
    private final GrpcSender sender;

    public StartCommand(CommandStartConfiguration config, GrpcSender sender) {
        super(config.getUserIdentifier(), config.getDescription());
        this.answer = config.getAnswer();
        this.sender = sender;
    }

    @Override
    public void execute(AbsSender absSender, User user, Chat chat, String[] strings) {
        if (strings.length != 0) {
            String userName = getUserName(user);
            log.info("User \"{}\" started bot with chatId: {}", userName, chat.getId());

            sendAnswer(absSender, chat.getId(), this.getCommandIdentifier(), userName,
                    String.format(answer, userName));

            sender.sendResponse(ResponseDTOFactory.createTelegram(strings[0], chat.getId().toString()));
        }
    }

    /**
     * Telegram allows to leave the "username" field empty
     * In this case, firstName and lastName are combined
     *
     * @param user is Telegram user
     * @return String which is username or firstName with lastName together
     */
    private String getUserName(User user) {
        String userName = user.getUserName();
        return (userName != null) ? userName : String.format("%s %s", user.getLastName(), user.getFirstName());
```

```
    }
}
@AllArgsConstructor(access = AccessLevel.PRIVATE)
public class KeyboardFactory {

    /**
     * Added in exists message replyMarkup = null
     */
    public static EditMessageReplyMarkup deleteKeyBoard(String chatId, int messageId) {
        return EditMessageReplyMarkup.builder()
                .chatId(chatId)
                .messageId(messageId)
                .replyMarkup(null)
                .build();
    }

    public static InlineKeyboardMarkup confirmationKeyBoard() {
        List<InlineKeyboardButton> buttons = createButtons();
        return createKeyboard(buttons);
    }

    private static InlineKeyboardMarkup createKeyboard(List<InlineKeyboardButton> buttons) {
        InlineKeyboardMarkup inlineKeyboardMarkup = new InlineKeyboardMarkup();
        inlineKeyboardMarkup.setKeyboard(Collections.singletonList(buttons));
        return inlineKeyboardMarkup;
    }

    private static List<InlineKeyboardButton> createButtons() {
        List<InlineKeyboardButton> buttons = new ArrayList<>();
        addButton(buttons, ButtonId.CONFIRM.getValue(), ButtonId.CONFIRMED.getValue());
        addButton(buttons, ButtonId.DECLINE.getValue(), ButtonId.DECLINED.getValue());
        return buttons;
    }

    private static void addButton(List<InlineKeyboardButton> buttons, String text, String callbackData) {
        buttons.add(InlineKeyboardButton.builder()
                .text(text)
                .callbackData(callbackData)
                .build()
        );
    }
}


@AllArgsConstructor(access = AccessLevel.PRIVATE)
public class TelegramMessageFactory {

    public static SendMessage createMessage(Long chatId, String text) {
        return createMessage(chatId.toString(), text);
    }

    public static SendMessage createMessage(String chatId, String text) {
        return SendMessage.builder()
                .chatId(chatId)
                .text(text)
                .build();
    }
}


public interface UpdateHandler {

    Optional<BotApiMethod<?>> check(Update update);
}
```

```
@Slf4j
@Component
@RequiredArgsConstructor
@Order(Ordered.HIGHEST_PRECEDENCE)
public class CallbackQueryUpdateHandler implements UpdateHandler {

    @Override
    public Optional<BotApiMethod<?>> check(Update update) {
        if (update.hasCallbackQuery()) {
            CallbackQuery callback = update.getCallbackQuery();
            log.info("Answer for confirmation: {}", callback.getData());

            return Optional.ofNullable(createEditMessageReplyMarkup(callback));
        }
        return Optional.empty();
    }

    private EditMessageReplyMarkup createEditMessageReplyMarkup(CallbackQuery callback) {
        return EditMessageReplyMarkup.builder()
                .chatId(callback.getMessage().getChatId().toString())
                .messageId(callback.getMessage().getMessageId())
                .replyMarkup(null)
                .build();
    }
}


@Slf4j
@Component
@RequiredArgsConstructor
public class ConfirmedAnswerHandler implements UpdateHandler {

    private final MfaScenarioRequestRepositoryImpl repository;
    private final GrpcSender sender;

    @Override
    public Optional<BotApiMethod<?>> check(Update update) {
        if (update.hasCallbackQuery()) {
            CallbackQuery callback = update.getCallbackQuery();
            String chatId = callback.getMessage().getChatId().toString();

            if (callback.getData().equals(ButtonId.CONFIRMED.getValue())) {
                MfaScenarioRequest data = repository.pollMfaScenarioRequest(chatId)
                        .orElseThrow(() -> new StorageException("Data for chatId: " + chatId + " don`t exist"));

                ResponseDTO responseDTO = ResponseDTOFactory.createConfirmedTelegram(data.getRequestId(), chatId,
data.getOtp());
                sender.sendResponse(responseDTO);

                return Optional.ofNullable(TelegramMessageFactory.createMessage(chatId, ButtonId.CONFIRMED.getValue()));
            }
        }
        return Optional.empty();
    }
}


@Slf4j
@Component
@RequiredArgsConstructor
public class DeclinedAnswerHandler implements UpdateHandler {
```

```
    private final MfaScenarioRequestRepositoryImpl repository;
    private final GrpcSender sender;

    @Override
    public Optional<BotApiMethod<?>> check(Update update) {
        if (update.hasCallbackQuery()) {
            CallbackQuery callback = update.getCallbackQuery();
            String chatId = callback.getMessage().getChatId().toString();

            if (callback.getData().equals(ButtonId.DECLINED.getValue())) {
                MfaScenarioRequest data = repository.pollMfaScenarioRequest(chatId)
                        .orElseThrow(() -> new StorageException("Data for chatId: " + chatId + " don`t exist"));

                ResponseDTO responseDTO = ResponseDTOFactory.createDeclinedTelegram(data.getRequestId(), chatId);
                sender.sendResponse(responseDTO);

                return Optional.ofNullable(TelegramMessageFactory.createMessage(chatId, ButtonId.DECLINED.getValue()));
            }
        }
        return Optional.empty();
    }
}


@Slf4j
@Component
public class MessageUpdateHandler implements UpdateHandler {

    @Value("${answer.command.unknown}")
    private String unknownCommand;

    @Override
    public Optional<BotApiMethod<?>> check(Update update) {
        if (update.hasMessage()) {
            Message message = update.getMessage();
            log.warn("User from chat \"{}\" send message: {}", message.getChatId(), message.getText());

            return Optional.ofNullable(TelegramMessageFactory.createMessage(message.getChatId().toString(),
unknownCommand));
        }
        return Optional.empty();
    }
}


/**
 * Created for saving request data when user`s answer is expecting
 */
@Data
@AllArgsConstructor
@NoArgsConstructor
@Builder
public class MfaScenarioRequest {
    private String requestId;
    private int messageId;
    private String otp;
    private BotType botType;
    private final Instant time = Instant.now();

    public boolean isAfter(Duration duration) {
        return Instant.now().isAfter(time.plus(duration));
    }
}
```

```
@AllArgsConstructor(access = AccessLevel.PRIVATE)
public class MfaScenarioRequestFactory {

    public static MfaScenarioRequest create(String requestId, int messageId, String opt, BotType botType){
        return MfaScenarioRequest.builder()
                .requestId(requestId)
                .messageId(messageId)
                .botType(botType)
                .otp(opt)
                .build();
    }

    public static MfaScenarioRequest create(int messageId, BotType botType){
        return create(null, messageId, null, botType);
    }

    public static MfaScenarioRequest create(String requestId, String opt, BotType botType){
        return create(requestId, 0, opt, botType);
    }
}


public interface MfaScenarioRequestRepository {

    Map<String, MfaScenarioRequest> getAll();

    void delete(String key);
}


@Component
@RequiredArgsConstructor
public class MfaScenarioRequestRepositoryImpl implements MfaScenarioRequestRepository {

    private final ConcurrentHashMap<String, MfaScenarioRequest> mfaScenarioRequests = new ConcurrentHashMap<>();

    public void addMfaScenarioRequest(String key, MfaScenarioRequest data) {
        mfaScenarioRequests.put(key, data);
    }

    public Optional<MfaScenarioRequest> pollMfaScenarioRequest(String key) {
        return Optional.ofNullable(mfaScenarioRequests.remove(key));
    }

    @Override
    public Map<String, MfaScenarioRequest> getAll() {
        return mfaScenarioRequests;
    }

    @Override
    public void delete(String key) {
        mfaScenarioRequests.remove(key);
    }
}


@Component
public class RequestDataMapper {

    public MfaScenarioRequest toMfaScenarioRequest(RequestDTO request){
        MfaScenarioRequest data = new MfaScenarioRequest();
        data.setRequestId(request.getId());
```

```
        String otp = getOtp(request);
        data.setOtp(otp);
        data.setBotType(request.getBotType());
        return data;
    }

    private String getOtp(RequestDTO request) {
        return Optional.ofNullable(request.getData().getParameters().get(RequestParams.OTP.getValue()))
                .orElseThrow(() -> new ValidationException("Parameter \"otp\" is required"));
    }
}


@Component
public class GrpcExceptionHandler implements ServerInterceptor {

    @Override
    public <ReqT, RespT> ServerCall.Listener<ReqT> interceptCall(ServerCall<ReqT, RespT> serverCall, Metadata metadata,
                                                 ServerCallHandler<ReqT, RespT> serverCallHandler) {
        ServerCall.Listener<ReqT> listener = serverCallHandler.startCall(serverCall, metadata);
        return new ExceptionHandlingServerCallListener<>(listener, serverCall, metadata);
    }

    private static class ExceptionHandlingServerCallListener<ReqT, RespT>
            extends ForwardingServerCallListener.SimpleForwardingServerCallListener<ReqT> {
        private final ServerCall<ReqT, RespT> serverCall;
        private final Metadata metadata;

        ExceptionHandlingServerCallListener(ServerCall.Listener<ReqT> listener, ServerCall<ReqT, RespT> serverCall,
                                            Metadata metadata) {
            super(listener);
            this.serverCall = serverCall;
            this.metadata = metadata;
        }

        @Override
        public void onHalfClose() {
            try {
                super.onHalfClose();
            } catch (RuntimeException ex) {
                handleException(ex, serverCall, metadata);
                throw ex;
            }
        }

        @Override
        public void onReady() {
            try {
                super.onReady();
            } catch (RuntimeException ex) {
                handleException(ex, serverCall, metadata);
                throw ex;
            }
        }

        private void handleException(RuntimeException exception, ServerCall<ReqT, RespT> serverCall, Metadata metadata) {
            if (exception instanceof SendMessageToBotException) {
                serverCall.close(Status.INTERNAL.withDescription(exception.getMessage()), metadata);
            } else if ((exception instanceof UnsupportedOperationException) || (exception instanceof ValidationException)) {
                serverCall.close(Status.INVALID_ARGUMENT.withDescription(exception.getMessage()), metadata);
            } else {
                serverCall.close(Status.UNKNOWN, metadata);
            }
        }
```

```
    }
}
public class InjectPropertiesException extends RuntimeException{
    public InjectPropertiesException(String message) {
        super(message);
    }
}


public class SendMessageToBotException extends RuntimeException {
    public SendMessageToBotException(TelegramApiException e) {
        super(e);
    }
}


public class StorageException extends RuntimeException{
    public StorageException(String message) {
        super(message);
    }
}


public class ValidationException extends RuntimeException{
    public ValidationException(String message) {
        super(message);
    }
}


@Configuration
public class GrpcServerConfig {

    @Value("${grpc.server.port}")
    private int serverPort;

    @Bean
    public Server serverForGRPCCommunication(GrpcReceiverImpl botsService, GrpcExceptionHandler handler) {
        return ServerBuilder
                .forPort(serverPort)
                .addService(botsService)
                .intercept(handler)
                .build();
    }

}


@Configuration
@EnableScheduling
@PropertySource(value = "classpath:application-strings.yml", factory = YamlPropertySourceFactory.class)
public class JavaConfig {

    @Bean
    public Map<ScenarioType, Scenario> scenarios(List<Scenario> scenariosList) {
        return scenariosList.stream().collect(Collectors.toMap(Scenario::supportType, o -> o));
    }

    @Bean
    public Map<BotType, BotService> bots(List<BotService> botServicesList) {
        return botServicesList.stream().collect(Collectors.toMap(BotService::supportType, o -> o));
    }
}
```

```
public class YamlPropertySourceFactory implements PropertySourceFactory {

    @Override
    public PropertySource<?> createPropertySource(String name, EncodedResource encodedResource){
        YamlPropertiesFactoryBean factory = new YamlPropertiesFactoryBean();
        factory.setResources(encodedResource.getResource());

        Properties properties = factory.getObject();

        return new PropertiesPropertySource(encodedResource.getResource().getFilename(), properties);
    }
}


@Data
@Component
@ConfigurationProperties(prefix = "bot")
public class BotTelegramConfiguration {

    private Telegram telegram;


    public String getTelegramName() {
        if (telegram.getName() != null) {
            return telegram.getName();
        }
        throw new InjectPropertiesException("Telegram bot name can`t be empty");
    }

    public String getTelegramToken() {
        if (telegram.getToken() != null) {
            return telegram.getToken();
        }
        throw new InjectPropertiesException("Telegram bot token can`t be empty");
    }

    @Data
    public static class Telegram {
        private String name;
        private String token;
    }
}


@Data
@Component
@ConfigurationProperties(prefix = "command")
public class CommandStartConfiguration {

    private Start start;

    public String getUserIdentifier() {
        if (start.getUserIdentifier() != null) {
            return start.getUserIdentifier();
        }
        throw new InjectPropertiesException("Start command identifier can`t be empty");
    }

    public String getDescription() {
        if (start.getDescription() != null) {
            return start.getDescription();
        }
```

```
        throw new InjectPropertiesException("Start command description can`t be empty");
    }

    public String getAnswer() {
        if (start.getAnswer() != null) {
            return start.getAnswer();
        }
        throw new InjectPropertiesException("Start command answer can`t be empty");
    }

    @Data
    public static class Start {
        private String userIdentifier;
        private String description;
        private String answer;
    }
}


@Slf4j
@Configuration
public class TelegramConfig {

    @Bean
    public TelegramBotsApi telegramBotsApi(TelegramBot telegramBot) {
        try {
            TelegramBotsApi botsApi = new TelegramBotsApi(DefaultBotSession.class);
            botsApi.registerBot(telegramBot);
            return botsApi;
        } catch (TelegramApiException e) {
            log.error("Telegram error in register bot: ", e);
            throw new SendMessageToBotException(e);
        }
    }
}


/**
 * Accords to BotsOuterClass.BotType
 */
public enum BotType {
    TELEGRAM, SLACK
}


@Component
public class BotTypeMapper {

    private static final Map<BotsOuterClass.BotType, BotType> enumBotType = Map.of(
            BotsOuterClass.BotType.TelegramBot, BotType.TELEGRAM,
            BotsOuterClass.BotType.SlackBot, BotType.SLACK);

    private static final Map<BotType, NotificationOuterClass.BotType> enumBotTypeGrpc = Map.of(
            BotType.TELEGRAM, NotificationOuterClass.BotType.TelegramBot,
            BotType.SLACK, NotificationOuterClass.BotType.SlackBot);

    public BotType toBotType(BotsOuterClass.BotType type) {
        return Optional.ofNullable(enumBotType.get(type))
                .orElseThrow(() -> new UnsupportedOperationException("Bot type \"" + type + "\" is not supported"));
    }

    public NotificationOuterClass.BotType toBotTypeGrpc(BotType type) {
        if (type == null){
            return null;
```

```
        }
        return Optional.ofNullable(enumBotTypeGrpc.get(type))
                .orElseThrow(() -> new UnsupportedOperationException("Bot type \"" + type + "\" is not supported"));
    }
}


public enum ButtonId {

    CONFIRMED("Confirmed"),
    DECLINED("Declined"),
    CONFIRM("Confirm"),
    DECLINE("Decline");

    private final String value;

    ButtonId(String value) {
        this.value = value;
    }

    public String getValue() {
        return value;
    }
}


public enum RequestParams {

    OTP("otp"),
    IS_CANCELED("isCanceled");

    private final String value;

    RequestParams(String value) {
        this.value = value;
    }

    public String getValue() {
        return value;
    }
}


/**
 * Accords to BotsOuterClass.ScenarioType
 */
public enum ScenarioType {
    MFA, NOTIFICATION, TACK_DIALOG
}


@Component
public class ScenarioTypeMapper {

    private static final Map<BotsOuterClass.ScenarioType, ScenarioType> enumScenarioType = Map.of(
            BotsOuterClass.ScenarioType.MfaScenario, ScenarioType.MFA,
            BotsOuterClass.ScenarioType.NotificationScenario, ScenarioType.NOTIFICATION,
            BotsOuterClass.ScenarioType.TaskDialogScenario, ScenarioType.TACK_DIALOG);

    private static final Map<ScenarioType, BotsOuterClass.ScenarioType> enumScenarioTypeGrpc = Map.of(
            ScenarioType.MFA, BotsOuterClass.ScenarioType.MfaScenario,
            ScenarioType.NOTIFICATION, BotsOuterClass.ScenarioType.NotificationScenario,
            ScenarioType.TACK_DIALOG, BotsOuterClass.ScenarioType.TaskDialogScenario);
```

```
    public ScenarioType toScenarioType(BotsOuterClass.ScenarioType type) {
        return Optional.ofNullable(enumScenarioType.get(type))
                .orElseThrow(() -> new UnsupportedOperationException("Scenario type \"" + type + "\" is not supported"));
    }

    public BotsOuterClass.ScenarioType toScenarioTypeGrpc(ScenarioType type) {
        return Optional.ofNullable(enumScenarioTypeGrpc.get(type))
                .orElseThrow(() -> new UnsupportedOperationException("Scenario type \"" + type + "\" is not supported"));
    }

}


syntax = "proto3";

package skysoft.enterprise.backend.bots;
option csharp_namespace = "SkySoft.Enterprise.Notifications.Grpc.Protos.Bots";

service Bots{
  rpc RunScenario(Scenario) returns (Empty);
}

message Scenario{
  string Id = 1;
  string UserIdentifier = 2;
  ScenarioType Type = 3;
  ScenarioData Data = 4;
  BotType BotType = 5;
}

message ScenarioData{
  string Message = 1;
  map<string, string> Parameters = 2;
}

message Empty{

}

enum ScenarioType{
  MfaScenario = 0;
  NotificationScenario = 1;
  TaskDialogScenario = 2;
}

enum BotType{
  TelegramBot = 0;
  SlackBot = 1;
}


syntax = "proto3";

package skysoft.enterprise.backend.notification;

option csharp_namespace = "SkySoft.Enterprise.Notifications.Grpc.Protos.Notifications";

service Notification{
  rpc SendNotification(NotificationRequest) returns (NotificationReply);

  rpc BotsReply(BotResponse) returns (Empty);
}

message NotificationRequest{
```

```
  Recipient Recipient = 1;
  EventType MessageType = 2;
  ChannelType ChannelType = 3;
  string Message = 4;
  VariablesKeyValue Variables = 5;

  message VariablesKeyValue{
    map<string, string> Values = 1;
  }
}

message BotResponse{
  string Id = 1;
  string UserIdentifier = 2;
  BotType BotType = 3;
  map<string, string> Parameters = 4;
}

message Empty {

}

enum EventType {
  ProjectAssignment = 0;
  RegisterInvite = 1;
  Mfa = 2;
}

enum ChannelType {
  None = 0;
  Email = 1;
  Telegram = 2;
  Push = 3;
  WebPush = 4;
}

enum BotType{
  TelegramBot = 0;
  SlackBot = 1;
}

message NotificationReply{
  bool SendingStatus = 1;
}

message Recipient{
  string UserName = 1;
}
```

Додаток Г (обов'язковий)

ПРОТОКОЛ ПЕРЕВІРКИ НАВЧАЛЬНОЇ (КВАЛІФІКАЦІЙНОЇ) РОБОТИ

Назва роботи: *«Розробка програмного забезпечення мікросервісу другого фактору автентифікації»*.

Тип роботи: кваліфікаційна робота

(кваліфікаційна робота, курсовий проект (робота), реферат, аналітичний огляд, інше – зазначити)

Підрозділ: кафедра АІІТ, ФІІТА, 1ІСТ-21м.

(кафедра, факультет, навчальна група)

Науковий керівник: Кулик Я.А., доц. каф. АІІТ

(прізвище, ініціали, посада)

Показники звіту подібності

Plagiat.pl (StrikePlagiarism)		*Unicheck*	
КП1	-	Оригінальність	89,5
КП2	-		
Тривога/Білі знаки	/	Схожість	10,5

Аналіз звіту подібності (відмітити потрібне)

X Запозичення, виявлені у роботі, оформлені коректно і не містять ознак плагіату.

Заявляю, що ознайомлений (-на) з повним звітом подібності, який був згенерований Системою щодо роботи (додається)

Автор _____ Омельченко В.О.
(підпис) (прізвище, ініціали)

Опис прийнятого рішення:

Допустити до захисту

Особа, відповідальна за перевірку _____ Маслій Р.В.
(підпис) (прізвище, ініціали)