Вінницький національний технічний університет
Факультет інформаційних технологій та комп'ютерної інженерії
Кафедра обчислювальної техніки

МАГІСТЕРСЬКА КВАЛІФІКАЦІЙНА РОБОТА

на тему:

ВДОСКОНАЛЕНИЙ МЕТОД АУТЕНТИФІКАЦІЇ ДЛЯ ДОСТУПУ ДО ДАНИХ В ХМАРНОМУ СЕРЕДОВИЩІ

Виконав студент 2 курсу, групи 1КІ-23м
спеціальності 123 — «Комп'ютерна інженерія»
_____ Стасюк Є. Є.
Керівник к.т.н., доц. каф. ОТ
_____ Войцеховська О. В.
«__» _____ 2024 р.
Опонент к.т.н., доц. каф. ПЗ
_____ Рейда О. М.
«__» _____ 2024 р.

Допущено до захисту
Завідувач кафедри ОТ
_____ д.т.н., проф. Азаров О.Д.
«__» _____ 2024 р.

ВНТУ 2024

ВІННИЦЬКИЙ НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ

Факультет інформаційних технологій та комп'ютерної інженерії

Кафедра обчислювальної техніки

Освітньо-кваліфікаційний рівень магістр

Спеціальність — комп'ютерна інженерія

Освітня програма – комп'ютерна інженерія

ЗАТВЕРДЖУЮ
Завідувач кафедри ОТ
д.т.н., професор Азаров О.Д.
"18" вересня 2024 року

З А В Д А Н Н Я
НА ДИПЛОМНУ РОБОТУ СТУДЕНТУ

Стасюка Євгенія Євгенійовича

1 Тема роботи: «Вдосконалений метод аутентифікації для доступу до даних в хмарному середовищі», керівник роботи Войцеховська Олена Валеріївна к.т.н., доцент кафедри ОТ, затверджені наказом вищого навчального закладу від "17" вересня 2024 року № 310

2 Строк подання студентом роботи: 17.12.2024

3 Вихідні дані до роботи: сучасні методи аутентифікації, технології передачі даних, сучасні технології розробки серверних застосунків та технології для зберігання даних у хмарному сховищі.

4 Зміст розрахунково-пояснювальної записки: вступ, аналіз сучасних методів аутентифікації користувачів для доступу до даних в хмарних середовищах, вдосконалений метод аутентифікації для доступу до даних в хмарному середовищі, програмна реалізація застосунку для доступу до даних в хмарному середовищі з вдосконаленою аутентифікацією, розгортання та тестування застосунку.

5 Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень): структурна схема вдосконаленого методу аутентифікації, схема реалізації шифрування та розшифрування файлу, схема процесу шифрування маркеру доступу, блок-схема алгоритму вдосконаленої аутентифікації користувача.

6 Консультанти розділів роботи представлено в таблиці 1.

Таблиця 1 — Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		Завдання видав	Завдання прийня
1–4	Войцеховська О.В. к. т. н., доцент каф. ОТ	дата підпис	дата підпис
5	Небава М. І. к. е. н., професор каф. ЕПВМ	дата підпис	дата підпис

7 Дата видачі завдання: 18 вересня 2024р.

8 Календарний план роботи зазначений в таблиці 2.

Таблиця 2 — Календарний план

№ з/п	Назва етапів дипломної роботи	Строк виконання	Примітк
	Постановка задачі роботи	18.09.2024-22.09.2024	Вик.
	Огляд сучасних методів аутентифікації	23.09.2024-25.09.2024	Вик.
	Аналіз сертифікатів для аутентифікації	26.09.2024-27.09.2024	Вик.
	Аналіз та вибір архітектури для клієнт-серверного застосунку	28.09.2024-29.09.2024	Вик.
	Аналіз та вибір програмного забезпечення для документації та тестування	30.09.2024-02.10.2024	Вик.
	Проектування бази даних	03.10.2024-24.10.2024	Вик.
	Програмна реалізація застосунку	25.10.2024-31.10.2024	Вик.
	Перевірка працездатності застосунку	01.11.2024-14.11.2024	Вик.
	Оформлення пояснювальної записки	15.11.2024-17.11.2024	Вик.
	Перевірка якості виконання магістерської роботи	18.11.2024	Вик.

Студент Стасюк Є.

Керівник роботи Войцеховська О.

АНОТАЦІЯ

УДК 004.9

Стасюка Є. Є. Вдосконалений метод аутентифікації для доступу до даних в хмарному середовищі. Магістерська кваліфікаційна робота зі спеціальності 123 — Комп'ютерна Інженерія, Вінниця: ВНТУ, 2024 — 117с.

На укр. мові. Бібліогр.: назв: 27; рис.: 16; табл. 8; ліст.: 3.

У магістерській кваліфікаційній роботі вдосконалено метод аутентифікації користувачів на основі сертифікату шляхом додаткового кодування конфіденційних даних, який використовується для доступу до даних в хмарному середовищі.

Розглянуто сучасні методи аутентифікації, проведено порівняльний аналіз існуючих хмарних систем зберігання інформації, проаналізовано сучасні технології та архітектурні рішення для реалізації серверного застосунку.

Розроблено програмний застосунок для доступу до даних в хмарному середовищі з вдосконаленою аутентифікацією, яка дозволила підняти рівень безпеки застосунку. Проведене тестування застосунку показало адекватність розробленого методу аутентифікації.

Ключові слова: аутентифікація, маркер доступу, шифрування, хмарне сховище, системи зберігання інформації, серверний застосунок.

ABSTRACT

УДК 004.9

Stasiuk Y. Y. Improved authentication method for access to data in a cloud environment. Master's qualification work in the specialty 123 – Computer Engineering, Vinnytsia: VNTU, 2024 — 117p. In ukrainian. Bibliography: titles: 27; drawing: 16; table: 8; listing: 3.

In the master's qualification work was improved the method of user authentication based on a certificate that additional encoding confidential data, which is used for getting access to data in a cloud environment.

Modern authentication methods are considered, a comparative analysis of existing cloud information storage systems is conducted, modern technologies and architectural solutions for the implementation of a server application are analyzed.

A software application for access to data in a cloud environment with improved authentication was developed, which allowed to increase the level of application security. The application testing showed the adequacy of the developed authentication method.

Keywords: authentication, access token, encryption, cloud storage, information storage systems, server application.

ЗМІСТ

ВСТУП 9

1 АНАЛІЗ СУЧАСНИХ МЕТОДІВ АУТЕНТИФІКАЦІЇ КОРИСТУВАЧІВ ДЛЯ ДОСТУПУ ДО ДАНИХ В ХМАРНИХ СЕРЕДОВИЩАХ 12

1.1 Аналітичний огляд методів аутентифікації користувачів 12

1.1.1 Однофакторна аутентифікація 12

1.1.2 Багатофакторна аутентифікація 13

1.1.3 Аутентифікація за допомогою поведінкових характеристик 14

1.1.4 Аутентифікація за сертифікатами 15

1.1.5 Біометрична аутентифікація 16

1.2 Порівняльний аналіз існуючих методів аутентифікації 17

1.3 Порівняльний аналіз існуючих систем з доступом до даних у хмарному середовищі 19

2 ВДОСКОНАЛЕНИЙ МЕТОД АУТЕНТИФІКАЦІЇ ДЛЯ ДОСТУПУ ДО ДАНИХ В ХМАРНОМУ СЕРЕДОВИЩІ 23

2.1 Вдосконалений метод аутентифікації користувача на основі сертифікату 23

2.2 Використання маркеру доступу JWT для вдосконалення аутентифікації користувачів 26

2.3 Використання технології HTTP для передачі даних при аутентифікації користувача 27

2.4 Проектування загальної архітектури застосунку для доступу до даних в хмарному середовищі з вдосконаленою аутентифікацією 31

2.4.1 Побудова кінцевих точок для серверного застосунку з вдосконаленою аутентифікацією 32

2.4.2 Обгрунтування вибору архітектурного стилю 34
2.5 Варіативний аналіз технологій для реалізації застосунку з вдосконаленою аутентифікацією в архітектурному стилі REST API 35
2.5.1 Аналіз платформи Node.js та фреймворку Nest.js 36
2.5.2 Аналіз мови програмування PHP та фреймворку Laravel 37
2.5.3 Аналіз платформи JDK та фреймворку Spring 38
2.6 Обґрунтування вибору стеку технологій для реалізації застосунку в архітектурному стилі REST API 39
2.7 Використання технологій шифрування для вдосконалення методу аутентифікації 41
2.7.1 Використання технології хешування паролю 41
2.7.2 Використання шифрування маркеру доступу 42
2.7.3 Використання шифрування файлів 43
3 ПРОГРАМНА РЕАЛІЗАЦІЯ ЗАСТОСУНКУ ДЛЯ ДОСТУПУ ДО ДАНИХ В ХМАРНОМУ СЕРЕДОВИЩІ З ВДОСКОНАЛЕНОЮ АУТЕНТИФІКАЦІЄЮ 46
3.1 Обґрунтування вибору хмарного середовища для зберігання файлів 46
3.2 Налаштування обробки запитів з аутентифікацією 48
3.3 Розробка логіки для створення та валідації JWT 50
3.4 Програмна реалізація структури застосунку з використанням MVC шаблону 53
3.5 Налаштування маршрутизації між кінцевими точками для HTTP-запитів 56
3.6 Розробка ER-моделі та структури бази даних 58
4 РОЗГОРТАННЯ ТА ТЕСТУВАННЯ ЗАСТОСУНКУ 65
4.1 Створення та налаштування Docker контейнеру 65
4.2 Документування кінцевих точок 66
4.3 Автоматизоване тестування кінцевих точок 69

4.4 Тестування авторизації та аутентифікації 74
5 ЕКОНОМІЧНА ЧАСТИНА 76
5.1 Комерційний та технологічний аудит науково-технічної розробки..... 76
5.2 Прогнозування витрат на виконання науково-дослідної (дослідно-конструкторської) роботи..... 79
5.3 Розрахунок економічної ефективності науково-технічної розробки за її можливої комерціалізації потенційним інвестором 84
ВИСНОВКИ 91
ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ..... 93
ДОДАТОК А Технічне завдання 96
ДОДАТОК Б Структурна схема вдосконаленого методу аутентифікації ... 100
ДОДАТОК В Схема реалізації шифрування та розшифрування файлу..... 101
ДОДАТОК Г Схема процесу шифрування маркеру доступу 103
ДОДАТОК Д Блок-схема алгоритму вдосконаленої аутентифікації користувача..... 104
ДОДАТОК Е Лістинг сервісу JwtService..... 105
ДОДАТОК Ж Лістинг сервісу AESEncryptionService 108
ДОДАТОК И Лістинг сервісу FileService..... 110
ДОДАТОК К ПРОТОКОЛ ПЕРЕВІРКИ КВАЛІФІКАЦІЙНОЇ РОБОТИ НА НАЯВНІСТЬ ТЕКСТОВИХ ЗАПОЗИЧЕНЬ 115

ВСТУП

В сучасному світі обсяги використання хмарних сховищ зростають, що підвищує кількість атак на конфіденційні дані користувачів. Практичні стандартні паролі з багатофакторною аутентифікацією не завжди відповідають сучасним вимогам кібербезпеки, тому що методи атак регулярно вдосконалюються, включаючи фішинговий підбір пароля, викрадення сесії та атаки з використанням викрадених облікових даних. Аутентифікація сприяє кращому захисту даних від втручання третіх сторін. Вона дозволяє не тільки підвищити рівень захисту, але й забезпечити більше комфорту та зручності для кінцевого користувача.

Крім того, системи аутентифікації повинні бути здатними розвиватися відповідно до вимог сучасних правил захисту даних, таких як закон України «Про захист персональних даних» [1], GDPR в Європі [2] або CCPA в Каліфорнії та інших штатах [3].

Хоча методи аутентифікації захищені від зміни та підробки, вони досі несуть в собі закодовану конфіденційну інформацію, яку легко розкодувати. Через це виникає вразливість в першу чергу для даних які ідентифікують користувача і хоча треті особи не зможуть отримати доступ до даних користувача це дає їм уявлення як працює система, що може допомогти з плануванням її зламу. Також сучасні методи аутентифікації мають відкриті методи кодування, що дозволяє перехоплювати ідентифікатор користувача і використовувати його в своїх цілях.

Актуальність магістерської кваліфікаційної роботи полягає в покращенні існуючих методів аутентифікації через їх вразливість до витоку конфіденційної інформації. Розробка вдосконаленого методу аутентифікації має на меті зменшення кількості успішних атак на хмарні платформи в довгостроковій перспективі, що може позитивно обернутися довірою до хмарних сервісів. Така аутентифікація може забезпечити більш ефективний

захист від зловмисників і зменшити грошові втрати від розкриття інформації, а також зменшити навантаження на відділ безпеки. Це робить тему важливою, оскільки вона спрямована на вирішення існуючих проблем щодо прогресу хмарних технологій та кібербезпеки, пропонуючи нові, кращі способи аутентифікації користувачів.

Метою дослідження є вдосконалення методу аутентифікації для забезпечення безпечного доступу до даних у хмарному середовищі шляхом додаткового шифрування маркеру доступу, що дасть можливість підвищити безпеку всієї системи та засекретити конфіденційні дані користувача, що використовуються для його ідентифікації.

Задачі дослідження:

— провести аналіз сучасних методів аутентифікації;

— провести порівняльний аналіз існуючих систем зберігання даних в хмарному середовищі;

— вдосконалити метод аутентифікації користувача шляхом шифрування маркеру доступу;

— виконати проектування архітектури програмного застосунку для доступу до даних в хмарному середовищі з використанням вдосконаленого методу аутентифікації;

— обґрунтувати вибір технологій для програмної реалізації застосунку для доступу до даних в хмарному середовищі;

— реалізувати збереження зашифрованих даних в хмарному середовищі з використанням вдосконаленого методу аутентифікації;

— провести тестування розробленого застосунку з вдосконаленою аутентифікацією;

— розрахувати економічну ефективність розробки програмного застосунку для доступу до даних в хмарному середовищі з використанням вдосконаленого методу аутентифікації.

Об'єктом дослідження є процеси аутентифікації в хмарному середовищі, які використовуються для захисту доступу до даних.

Предметом дослідження є методи передавання інформації в комп’ютерних мережах та методи аутентифікації для доступу до даних у хмарних середовищах.

Новизна одержаних результатів – вдосконалено метод аутентифікації користувачів для забезпечення безпечного доступу до даних у хмарному середовищі, в якому, на відміну від існуючих, реалізовано шифрування сертифікату для збереження конфіденційної інформації користувача, що використовується для ідентифікації та шифрування файлів перед відправкою у хмарне сховище, що дозволило знизити ризики несанкціонованого доступу до інформації та підвищити безпеку даних.

Практичне значення роботи полягає у створенні програмного застосунку, який може бути використаний для підвищення рівня захищеності хмарних сервісів та корпоративних інформаційних систем. Запропонований метод аутентифікації може бути застосований в різних сферах, таких як фінансові установи, охорона здоров’я, державний сектор та інші галузі, де безпека даних є критично важливою. Вдосконалений підхід до аутентифікації дозволить знизити ризики несанкціонованого доступу до інформації.

Апробація результатів роботи здійснена під час виступу на науково-технічній конференції «Молодь в науці: дослідження, проблеми, перспективи (МН-2025)»

Публікація за темою роботи — Стасюк Є. Є., Войцеховська О. В. Вдосконалення аутентифікації шляхом додаткового шифрування маркеру доступу // Матеріали конференції «Молодь в науці: дослідження, проблеми, перспективи (МН-2025)», 2025 р. [Електронний ресурс] — https://conferences.vntu.edu.ua/index.php/mn/mn2025/paper/view/22378 [4].

1 АНАЛІЗ СУЧАСНИХ МЕТОДІВ АУТЕНТИФІКАЦІЇ КОРИСТУВАЧІВ ДЛЯ ДОСТУПУ ДО ДАНИХ В ХМАРНИХ СЕРЕДОВИЩАХ

Аутентифікація — це процес підтвердження того, ким є користувач, який шукає доступ до системи чи ресурсу. Це захід безпеки для встановлення заявленої особи користувача шляхом перевірки облікових даних, наданих ним.

1.1 Аналітичний огляд методів аутентифікації користувачів

Найпоширенішим методом аутентифікації досі вважається введення логіна та паролю, хоча сьогодні активно застосовуються й інші методи, такі як двофакторна аутентифікація (TFA), яка поєднує різні типи підтвердження. Сучасні технології також передбачають використання біометричних даних у вигляді відбитків пальців, розпізнавання обличчя або розпізнавання голосу для більшої надійності аутентифікації. Зокрема, веб-додатки та API здебільшого використовують маркери або сертифікати для певного сеансу для підтвердження користувача після того, як користувач увійшов у систему без повторного введення пароля. У хмарних сховищах та інших системах з важливим вмістом для користувача аутентифікація є критичною в тому сенсі, що вона захищає ресурси від несанкціонованого доступу. Лише аутентифіковані користувачі мають доступ до конфіденційної інформації.

1.1.1 Однофакторна аутентифікація

Однофакторна аутентифікація — це найпростіший метод аутентифікації користувача за логіном та паролем. За логіном ідентифікується який саме користувач намагається отримати доступ до ресурсу, а пароль є секретною інформацією яка підтверджує особу користувача [5].

Цей метод аутентифікації простий для користувача, оскільки не потребує додаткових знань чи обладнання. Також він забезпечує швидкий

доступ до ресурсу, адже включає один єдиний крок, введення логіну та паролю.

Ступінь захисту такого методу залежить від довжини паролю, використання верхнього та нижнього регістру, а також використання спеціальних символів. Гарною практикою вважається спонукати користувача використовувати складний пароль, задля підвищення ступеня захисту.

Хоча цей метод аутентифікації вважається стандартом для доступу до персональних комп'ютерів, він не достатньо надійний для використання його в мережі інтернет, тому сучасні застосунки використовують інші методи.

Перевагами цього методу є простота реалізації, не вимагає великих обчислювальних потужностей та надійність у використанні. Недоліком цього методу є сам користувач, який може створити слабкий пароль, або використати один і той самий пароль для різних систем, зазнати фішингової атаки чи навіть сам повідомити пароль стороннім. Також цей метод аутентифікації вразливий для силових атак, таких як підбір паролю. Водночас, у випадку компрометації пароля інші методи підтвердження особи не передбачені, що суттєво знижує рівень захисту. На відміну від багатофакторної аутентифікації, тут злам одного фактора означає повний доступ до ресурсу. Через ці причини, попри свою доступність і зручність, однофакторна аутентифікація поступово поступається місцем більш надійним підходам у випадках, де рівень безпеки є критично важливим.

1.1.2 Багатофакторна аутентифікація

Багатофакторна аутентифікація — це не просто покращення однофакторної аутентифікації, а принципово інший підхід до захисту, де для входу користувач має підтвердити свою особу за допомогою кількох незалежних факторів. Ці фактори суттєво підвищують надійність системи. Наприклад, на додачу до звичайного паролю може бути код, що приходить на електронну пошту чи в SMS, токен, який можна отримати тільки з певного

смартфону, чи інша секретна інформація, до якої має доступ тільки користувач. Найкращим рішенням буде доступ до фізичного пристрою, наприклад, доступ до телефону, на який приходить код. Зазвичай додаткові фактори є одноразовими. [6]

Перевагою можна назвати високий рівень безпеки системи завдяки кількості та різноманітності факторів, до яких потрібно мати доступ, щоб пройти таку аутентифікацію. Також цей метод має високий ступінь захисту від фішингових атак. Навіть якщо пароль стане відомий стороннім, одноразові маркери на дадуть їм отримати доступ до приватного ресурсу.

Недоліком є незручність у використанні користувачем та залежність від сторонніх пристроїв чи систем. Також цей метод набагато складніший для імплементації (реалізації) та вимагає багато ресурсів як обчислювальних від самого застосунка так і матеріальних для підтримки різних факторів аутентифікації.

1.1.3 Аутентифікація за допомогою поведінкових характеристик

Аутентифікація за допомогою поведінкових характеристик – це метод аутентифікації, який записує та аналізує поведінку користувача та на основі цих даних визначає чи саме користувач отримує доступ до ресурсу. Такий метод є найбільш зручним для користувача, оскільки практично не потребує дій з боку самого користувача, щоб аутентифікувати себе.

Така система потребує часу для створення певного шаблону (патерну) дій користувача, за яким будуть порівнюватися наступні дії. При відхиленні від цього шаблону система запідозрить, що сторонній отримав доступ до акаунту поточного користувача і буде вимагати від нього додаткової аутентифікації. [7]

Хоча такий метод дуже ефективний саме через складність підробки або симуляції дій користувача, тому можна бути певним, що система буде добре

захищати профіль користувача від несанкціонованого доступу до конфіденційних даних.

Поведінка користувача може включати в себе велику кількість різних факторів, таких як кількість і швидкість надсилання запитів, використання методів введення та інших дій користувача.

Однак, при цьому, точність методу може бути низькою залежно від користувача та його стану і це може спричинити помилкові спрацьовування захисних механізмів. Крім того, реалізація такого методу вимагає величезних обчислювальних ресурсів, багато часу та має складність в реалізації. Також для роботи система повинна певний час збирати інформацію про дії користувача, тому вона почне працювати не одразу і не може використовуватись самостійно, а потребує застосування інших методів.

1.1.4 Аутентифікація за сертифікатами

Аутентифікація за сертифікатами — це метод, що базується на використанні цифрових сертифікатів для підтвердження особи користувача. Такі сертифікати мають такі основні ключові фактори як ідентифікатор власника, дата та час видачі, термін дії і підпис. Ідентифікатор власника існує, щоб відрізнити користувача від інших і ним виступає унікальне значення таке як логін, адреса електронної пошти чи просто ідентифікатор у базі даних. Дата та час видачі використовується з терміном дії для визначення тимчасового доступу до приватних ресурсів за цим сертифікатом. Підпис затверджує дійсність сертифікату. Також сертифікати мають можливість бути відкликаними при підозрілих діях користувача. Але це потребує додаткової реалізації, що підвищує складність реалізації системи.

При підключенні користувача система валідує підпис, перевіряє чи ще активний тимчасовий доступ та співставляє ідентифікатор з існуючими користувачами. Всі дані крім підпису сертифікату є відкритими і їх легко прочитати, але при зміні цих даних підпис перестане відповідати цим даним і

доступ до приватного ресурсу буде заблоковано. Завдяки таким заходам безпеки цей метод є одним з найнадійніших. [8]

Перевагами цього методу є простота використання для користувача, в більшості випадків користувач навіть не помічає цього сертифіката та його роботу. Непомітність роботи є не тільки перевагою у використанні для користувача, а також надає додатковий захист від фішингу, соціальної інженерії та витоку сертифікату до інших користувачів. Ще одною перевагою є надійність таких сертифікатів, оскільки їх неможливо підробити без ключа до підпису. Навіть при отриманні такого сертифікату третя сторона мало, що зможе зробити, а потім сертифікат стає недійсним.

Недоліком є складність в розробці та обслуговуванні. Також управління життєвим циклом сертифікатів підвищує складність розробки, вимагає доволі великої кількості обчислювальних ресурсів, а враховуючи, що сертифікат потрібно валідувати при кожному запиті і також періодично оновлювати його, то це підвищує час очікування відповіді від системи.

1.1.5 Біометрична аутентифікація

Біометрична аутентифікація — це ідентифікація користувача на основі деяких унікальних фізичних або поведінкових атрибутів. На відміну від традиційних способів аутентифікації, таких як паролі та PIN-коди, цей спосіб використовує відбитки пальців, сканування обличчя, розпізнавання голосу або аналіз райдужної оболонки ока, які важко підробити чи вкрасти. [9]

Одна з найбільших переваг біометричної аутентифікації полягає в тому, що вона зручна. Користувачеві не потрібно запам'ятовувати будь-які складні паролі або вводити коди. Користувач повинеи просто прикласти палець або подивітися у камеру. Він швидкий та інтуїтивно зрозумілий, що робить його привабливим для смартфонів, комп’ютерів та інших пристроїв.

З іншого боку, біометрія викликає певні проблеми. Оскільки біометричні дані є унікальними та постійними, їхнє компроментування може

завдати шкоди. Звичайні паролі можна замінити, але те саме не можна зробити з відбитками пальців або зображеннями райдужної оболонки ока. Таким чином, системи біометричної аутентифікації вимагають суворого захисту даних разом із дотриманням стандартів безпеки, щоб виключити будь-яку можливість витоку та зловживання.

Ще одна обов'язкова риса — точність. Біометричні системи мають забезпечувати високу точність розпізнавання, щоб виключити помилкові результати. Це важливо, оскільки, наприклад, за різних умов навколишнього середовища, таких як освітлення або фізичні зміни аутентифікація може бути не успішною та точною.

Біометрична аутентифікація сьогодні застосовується в багатьох сферах починаючи від захисту вашого смартфона та банківських послуг і закінчуючи контролем доступу в безпечних зонах. У хмарних середовищах і веб-сервісах біометрія часто реалізується як частина багатофакторної аутентифікації, що підвищує загальний рівень захисту.

1.2 Порівняльний аналіз існуючих методів аутентифікації

Однофакторна аутентифікація має переваги простоти впровадження, використання та економічності. Однак її недолік полягає в низькому рівні безпеки, оскільки цей метод дуже вразливий до атак.

Багатофакторна аутентифікація різко підвищує ступінь захисту завдяки використанню кількох ліній перевірки. Єдиним недоліком є низька зручність для користувачів, оскільки це вимагає додаткових кроків і опори на зовнішні пристрої або служби.

Аутентифікація на основі параметрів поведінки — це аутентифікація в автоматизований спосіб і однозначна параметризація аутентичності будь-якого даного суб'єкта є загальною. Однак цей метод занадто чутливий до змін параметрів поведінки, що може погіршити точність, тоді як складність

реалізацій робить його менш життєздатним рішенням для загальних застосувань.

Біометрична аутентифікація точна та надійна перевірка, що використовує унікальні фізичні атрибути для біометричної аутентифікації, яка є високоточною та надійною. Недоліками є те, що для впровадження потрібне спеціальне обладнання, можливі проблеми з конфіденційністю даних та впровадження є досить дорогим.

Порівняльна характеристика методів аутентифікації показана у таблиці 1.1.

Таблиця 1.1 — Порівняльна характеристика методів аутентифікації

Однофакторна аутентифікація	Простота, доступність	Низька безпека, вразливість до атак
Багатофакторна аутентифікація	Висока безпека, складність зламу	Зниження зручності, залежність від додаткових пристроїв
Аутентифікація за поведінкою	Автоматичність, непряме спостереження	Чутливість до змін у поведінці, складність реалізації
Біометрична аутентифікація	Висока точність, унікальність даних	Потреба у спеціальному обладнанні, питання конфіденційності
Аутентифікація за сертифікатами	Висока безпека, автоматизація, масштабованість, надійність	Потреба в керуванні сертифікатами, початкові витрати

Аутентифікація за сертифікатом є беззаперечним лідером за рівнем безпеки, стійкістю до підробок, автоматизацією процесів, масштабованістю та

мінімальною взаємодією з кінцевим користувачем серед переваг. До недоліків належать необхідність належного керування сертифікатами та початкова вартість впровадження. Аутентифікація за допомогою сертифіката є найкращим методом, оскільки він забезпечує дуже високий рівень безпеки, стійкість до сучасних загроз і зручність використання для систем масштабування. Надійність і довгострокова стабільність також роблять цю технологію рекомендованою для критичного захисту даних.

1.3 Порівняльний аналіз існуючих систем з доступом до даних у хмарному середовищі

Готові для кінцевого користувача хмарні рішення для зберігання, такі як Google Drive, Dropbox і Microsoft OneDrive, призначені для зручного доступу до файлів, зберігання та спільного використання. Вони мають спрощений інтерфейс і не вимагають від користувачів високих спеціальних технічних знань, що ідеально підходить для особистого використання, а також для використання в команді.

Google Drive є одним із широко використовуваних сервісів для зберігання та обміну файлами. Його можна інтегрувати з іншими службами Google, такими як Google Docs, Sheets і Slides, для створення, редагування та безпосереднього збереження документів у хмарі. Диск Google дозволяє зберігати зашифровані файли та має гнучкі налаштування для надання доступу для підвищення безпеки. [10]

Dropbox — популярне рішення для зберігання та обміну файлами. Він ідентифікується своєю очевидною легкістю та чесністю, а також безболісною інтеграцією з такими програмами, як Microsoft Office і Slack. Dropbox пропонує різні індивідуальні та бізнес-плани, які підтримують спільний доступ до файлів, синхронізацію між пристроями та автоматичне резервне копіювання. Він також забезпечує розширену безпеку завдяки таким

функціям, як двофакторна автентифікація та контроль того, хто може отримати доступ до вмісту.[11]

Microsoft OneDrive спеціально призначений для зручного зберігання файлів, фотографій, відео, контактів та інших даних у хмарі. Будучи частиною екосистеми Microsoft, OneDrive автоматично синхронізує всі дані між пристроями. Він має деякі дійсно потужні функції для відновлення даних у випадках, коли файли випадково видаляються; Крім того, обмінюватися файлами та папками надзвичайно легко з ним, і він відповідає найкращим у своєму класі стандартам безпеки за допомогою шифрування даних і двофакторної аутентифікації [12].

Ці готові рішення дозволяють користувачам зберігати файли в хмарі, отримувати до них доступ із будь-якого пристрою та легко й швидко обмінюватися інформацією з іншими людьми.

Порівнюючи такі популярні готові хмарні сервіси, як Google Drive, Dropbox і OneDrive необхідно виділити ряд ключових функцій, які підкреслюють переваги кожного з них.

Google Drive пропонує інтеграцію з екосистемою Google. Він досить простий у використанні та має надійні функції співпраці. Єдина проблема полягає в тому, що захист його даних залежить від загальної безпеки вашого облікового запису Google, яка може бути вразливою, особливо при використанні однофакторної аутентифікації. Google Drive використовує стандартні механізми шифрування та не має спеціалізовані алгоритми для захисту унікальних типів доступу.

З Dropbox дуже легко синхронізувати все на високій швидкості на всіх пристроях. Хоча його функції контролю доступу та безпеки не такі гнучкі, як у розробленому сервісі. Гарна інтеграція з продуктами Microsoft і ціни роблять OneDrive потужним інструментом для бізнес-користувачів. Однак, як і Google Drive, він приділяє більше уваги масовому споживачу і тому не завжди

забезпечує належне врахування деяких специфічних вимог корпоративних клієнтів щодо аутентифікації та налаштування доступу до даних.

Порівняння існуючих систем зображено у таблиці 1.2.

Таблиця 1.2 — Порівняння існуючих систем

Параметр	Google Drive	Microsoft OneDrive	Dropbox
Безпека	Шифрування SSL та інструменти Google безпеки	Інтеграція з Microsoft Security	Стандартне шифрування SSL
Гнучкість налаштувань	Базові параметри доступу та дозволів	Подібні до Google Drive параметри	Обмежені налаштування доступу
Продуктивність та швидкість	Залежить від інфраструктури Google	Залежить від Microsoft Cloud	Залежить від місця розташування серверів
Контроль над даними	Певний контроль, але з обмеженнями політики Google	Частковий контроль, залежно від політик	Більшість контролю в Dropbox
Підтримка масштабування	Масштабованість обмежена інфраструктурою	Хороша масштабованість завдяки Microsoft Cloud	Лімітоване масштабування
Кастомізація інтерфейсу	Обмежена кастомізація	Мінімальна кастомізація	Обмежена кастомізація

Аналіз таблиці 1.1 показав, що Google Drive використовується протокол шифрування SSL з власними інстументами безпеки, Microsoft OneDrive інтегрується з Microsoft Security, а Dropbox використовує тільки стандартне SSL шифрування.

Налаштування у цих популярних сервісах мають багато обмежень щодо керування файлами та доступом до них, зокрема вони сильно обмежують користувача у наданні доступу до своїх файлів іншим користувачам.

Google Drive і Microsoft OneDrive досягають своєї продуктивності та швидкості більшою мірою завдяки інфраструктурам Google і Microsoft відповідно. Сервіс Dropbox робить це залежно від того, де розташовані його сервери.

Політика компанії щодо даних обмежує користувача у використані даних. Існуючі сервіси встановлюють обмеження на збереження тільки певних типів даних та обов’язкове сканування вмісту файлів, що означає повний доступ до вмісту файлів.

Масштабованість деяких популярних сервісів також залежить від підтримки їхньої інфраструктури.

Проведений аналіз показав актуальність та доцільність вдосконалення методу аутентифікації користувачів для доступу до даних в хмарному середовищі та розробки програмного застосунку з використанням запропонованого методу аутентифікації та додаткових методів захисту даних. Запропонований застосунок з вдосконаленою аутентифікацією повинен забезпечити більшу гнучкість, ніж в подібних відомих застосунках, дозволяючи кінцевим користувачам налаштовувати функціональні характеристики відповідно до своїх потреб, використовуючи інтеграцію API та налаштування політик доступу. Також він повинен підвищити продуктивність для конкретних завдань і умов навколишнього середовища та забезпечити високу швидкість і ефективність та дати можливість опрацьовувати великий обсяг даних.

2 ВДОСКОНАЛЕНИЙ МЕТОД АУТЕНТИФІКАЦІЇ ДЛЯ ДОСТУПУ ДО ДАНИХ В ХМАРНОМУ СЕРЕДОВИЩІ

Для забезпечення безпечного доступу до даних у хмарному середовищі пропонується вдосконалити метод аутентифікації користувачів шляхом додаткового шифрування маркеру доступу для підвищення безпеки всієї системи та додатково засекретити конфіденційні дані користувача, що використовуються для його ідентифікації.

2.1 Вдосконалений метод аутентифікації користувача на основі сертифікату

Вдосконалена ідентифікація за допомогою сертифікатів досягається подвійним шифруванням маркера доступу, наприклад JSON Web Token (JWT). Це дозволить передавати інформацію про користувача та сертифікат в одному форматі, більш зручному для взаємодії між клієнтом і сервером. Такий маркер потрібно зашифрувати, щоб приховати конфіденційну інформацію користувача та який саме сертифікат використовується для аутентифікації. Цей підхід передбачає, що маркер JWT, який підписаний закритим ключем, додатково шифрується ключем сервера перед передачею клієнту. Після цього лише сервер може розшифрувати токен і використовувати дані, оскільки він володіє відповідним ключем. Це дозволить запобігти ситуаціям, коли токен перехоплюється та конфіденційна інформація у ньому може бути розкодована. Додатковий рівень шифрування JWT забезпечить більшу безпеку адже жодна третя сторона не зможе розкрити вміст маркера та викрасти всі ці особисті дані. Така реалізація підвищить стійкість до атак на рівні мережі та зупинить витік інформації, оскільки покращено захист від будь-яких спроб отримати доступ до облікових даних або сертифіката, що використовується.

Спочатку потрібно реалізувати аутентифікацію JWT у Spring, що забезпечує отримання токену з HTTP-запиту та перетворює його в об’єкт

аутентифікації, який визначає чи має користувач доступ до ресурсів. Щоб покращити метод аутентифікації на основі JWT потрібно реалізувати ще один рівень шифрування маркерів. Вилучення маркеру із заголовка запиту дасть маркер автентифікації, переданий для створення об’єкта аутентифікації на основі цього маркера. Якщо маркер не знайдено або виникають проблеми з обробкою маркера, створюється виняток. Підхід, який використовує кілька обробників автентифікації, дозволяє вибрати метод на основі контексту запиту. Структурну схему вдосконаленого методу аутентифікації подано в додатку Б.

Реалізація спеціального об’єкта аутентифікації дозволяє зберігати маркер як облікові дані з інформацією про користувача. На початку створення об'єкта прав доступу може не бути. Компонент автентифікації визначає, чи підтримується певний тип автентифікації, а потім перевіряє маркер, щоб отримати ідентифікатор користувача та призначити відповідні права доступу. Об'єкт автентифікації зберігає інформацію про права доступу, що робить автентифікацію завершеною. Це ініціює обробку відмови в доступі, коли користувач намагається отримати доступ до ресурсу, для якого він не має відповідних прав. У таких сценаріях відповідь передається назад із кодом 403 і докладною інформацією про те, чому у доступі до ресурсу було відмовлено. Розташування основних компонентів аутентифікації забезпечує правильну роботу паролів, доступ до інформації користувача та перевірку токенів. Це дає змогу налаштувати фільтри, обробники винятків і менеджери автентифікації, щоб відповідати на запити належним чином. Керування доступом на основі анотацій використовує ієрархію ролей, щоб забезпечити більш гнучкий контроль прав доступу.

Саме конфігурація ролі визначає ролі для незареєстрованих і зареєстрованих користувачів і встановлює ієрархію прав доступу. Таким чином можна спрямувати доступ до ресурсів у програмі, враховуючи деталі, що стосуються ролі. Іншими поширеними компонентами конфігурації безпеки у Spring є фільтр, обробка помилок, автентифікація та правила доступу. Це

гарантує, що лише користувачі з відповідними ролями для певних ресурсів можуть отримати до них доступ. Усе це разом утворює об’єднання, яке включає систему аутентифікації, авторизації та обробки помилок доступу для підвищення безпеки програми шляхом обмеження доступу на основі ролей користувача та анотацій.

Запропоновано використання шифрування файлів для завантаження в хмарне сховище без збереження ключа в системі, яке підвищить безпеку даних користувача. Для цього файл буде зашифровано на сервері перед завантаженням у хмарне сховище. Ключ до шифрування вводить сам користувач та він не зберігається у системі. Жодні сторонні системи, такі як сервери хмарного провайдера, також не мають доступу до ключа. Таким чином доступ до вмісту файлів є тільки у користувача який володіє ключем до шифру. Реалізацію завантаження та вивантаження файлів зображено на рисунках Г.1 та Г.2.

Метод має ряд істотних переваг головною з яких є те, що навіть якщо доступ до файлів здійснюється в хмарі, вони залишаться нечитабельними, оскільки їх можна розшифрувати лише за допомогою унікального ключа. Такий підхід виключає ризики які пов’язані з внутрішніми загрозами, наприклад витік даних через адміністраторів хмарного сховища або компрометація облікового запису. Крім того, користувач має повний контроль над своїми даними і може бути певен, що тільки він та інші користувачі яким він передав ключ мають доступ до вмісту файлу. Це особливо важливо для даних, які мають дуже високий рівень захисту. Це підвищує рівень надійності хмарного сховища для захисту файлів. Навіть постачальник послуг не матиме доступу, щоб відкрити їх без участі користувача. Таким чином сервіс отримує як перевагу високий рівень криптостійкості одночасно із легким доступом до сучасних хмарних сервісів, що підвищує рівень безпеки даних.

2.2 Використання маркеру доступу JWT для вдосконалення аутентифікації користувачів

JSON Web Token (JWT) — це стандарт для створення токенів доступу, що дозволяють безпечно передавати інформацію між клієнтом та сервером. JWT широко використовується для аутентифікації користувачів у веб-додатках та REST API завдяки своїй безпеці, простоті та легкості використання. Токен складається з трьох частин, розділених крапками: заголовка, корисного навантаження та підпису [13].

Заголовок містить тип токена та алгоритм підпису, який буде використовуватися для створення підпису. Зазвичай використовується рядок «HS256» для HMAC SHA-256 алгоритму чи «RS256» для RSA SHA-256 алгоритму.

Корисне навантаження містить дані, які потрібно передати між клієнтом та сервером. Цей блок даних може включати інформацію про користувача або права доступу. Існують зарезервовані ключі, такі як sub (ідентифікатор користувача), iat (час створення), exp (час закінчення дії токена) та інші. Корисне навантаження можна також налаштувати і додавати будь-які інші додаткові дані, необхідні для конкретної системи.

Підпис забезпечує цілісність та захищеність токена. Щоб створити підпис, заголовок і корисне навантаження об’єднуються, шифруються за допомогою обраного алгоритму та секретного ключа (в разі симетричного алгоритму) або приватного ключа (в разі асиметричного RSA-алгоритму). Отриманий підпис додається до токена.

При аутентифікації клієнт надсилає серверу облікові дані користувача, такі як ім'я та пароль. Сервер перевіряє ці дані, і якщо вони коректні, генерує JWT, включаючи в нього ідентифікатор користувача та іншу важливу інформацію. Токен потім повертається клієнту, який зберігає його для подальшого використання.

У запропонованому методі для аутентифікації користувача створюється JWT-токен для ідентифікації користувача, використовуючи такі ключі, як sub для передачі ідентифікатора, iat для запису часу створення та exp для регуляції терміну дії маркеру доступу.

Для доступу до захищених ресурсів клієнт надсилає запити разом із токеном для забезпечення аутентифікації користувача. Сервер отримує запит, перевіряє підпис токена і, якщо він дійсний, дає доступ до ресурсів, на які користувач має права.

JWT має обмежений термін дії, що задається в полі exp у корисному навантажені. Це захищає від використання застарілих токенів. Після закінчення терміну дії токена користувачеві потрібно знову пройти аутентифікацію або оновити токен, щоб отримати новий. Оновлення може виконуватись за допомогою "refresh token" — спеціального токена, який діє довше і служить лише для оновлення основного токена.

2.3 Використання технології HTTP для передачі даних при аутентифікації користувача

HyperText Transfer Protocol (HTTP) — це протокол, який використовується для обміну даними між клієнтом та сервером у мережі. Основна роль цього протоколу — обмін інформацією за принципом «запит-відповідь». Коли клієнт, наприклад веб-браузер, надсилає запит серверу щодо будь-якої інформації, сервер повертає відповідь, надаючи необхідні дані у вигляді тексту, зображень або будь-якого іншого файлу. HTTP складається з трьох різних частин: методів запиту, заголовків і тіла повідомлення [14].

Методи запиту — це механізми, за допомогою яких клієнт передає серверу те, що він хоче, щоб сервер зробив. Метод GET використовується для отримання даних, POST — для надсилання інформації для створення ресурсів, PUT — для повного оновлення, PATCH — для часткового оновлення і DELETE — для видалення. Кожен метод визначає, що саме робитиме сервер.

І хоча розробник серверного застосунку сам реалізує роботу методів рекомендується притримуватись стандартів закладених у HTTP.

Заголовки як у запиті, так і у відповіді пов’язують із ними метадані. Це може бути тип переданих даних, будь-який елемент керування хешуванням, методи аутентифікації, тривалість сеансу або налаштування мови. У взаємодії між клієнтом і сервером заголовки відіграють критичну роль, дозволяючи передавати інформацію про запит або відповідь без зміни тіла самих даних. Структуру HTTP протоколу зображено у вигляді знімку екрану на рисунку 2.3.

Рисунок 2.3 — Структура HTTP протоколу

Тіло повідомлення — це фактичні дані, які пересилаються між клієнтом і сервером. Тіло може передаватися у різних виглядах включаючи форму, бінарний файл та текст у таких форматах як JSON, XML та простий текст. У відповідь сервер може надсилати не тільки перелічені типи, а й навіть цілі HTML сторінки. Тіло повідомлення застосовується лише тоді, коли має бути

передано певний вміст. У запитах з методом GET тіло повідомлення зазвичай не використовується. Натомість інформація передається параметрами в URL.

HTTP можна використовувати в поєднанні з зашифрованими з'єднаннями через HTTPS. Це гарантує, що передані дані захищені від перехоплення та модифікації. У цьому випадку SSL або TLS створює безпечний канал між клієнтом і сервером.

HTTPS функціонує за основним принципом, згідно з яким усі дані, які передаються через нього, зашифровані, а отже, безпечні. Симетричне шифрування застосовується для шифрування інформації в HTTPS, де один ідентичний ключ використовується для шифрування та дешифрування інформації для захисту цієї інформації. Насправді спочатку дані шифруються за допомогою асиметричного шифрування за допомогою ключів, якими обмінюються браузер і сервер, а потім створюється зашифрований канал для передачі симетричного ключа за допомогою асиметричного шифрування.

HTTPS також забезпечує автентифікацію сервера, щоб користувачі могли точно знати, що вони підключаються до надійного сервера, а не до підробленого. Це робиться за допомогою цифрового сертифіката, представленого сервером, який був би виданий відомим довіреним центром сертифікації (CA). Браузер, безпосередньо перед відкриттям будь-якого з'єднання, перевіряє цей сертифікат, щоб переконатися, що сервер дійсно є тим, за що себе видає.

HTTPS забезпечує цілісність переданої інформації, оскільки хеш-функції перевіряють чи була інформація змінена під час передачі від відправника до одержувача.

Сертифікат спочатку перевіряється, а потім відправляється запит на обмін асиметричним ключем, який буде використовуватися для створення симетричного ключа для майбутнього шифрування. Цей повний процес відомий як рукостискання TLS і важливий для безпеки цього конкретного з'єднання.

Завдяки цьому методу інформація захищена під час передачі, щоб зловмисники не могли її перехопити чи підробити, що є важливою особливістю для конфіденційних даних. Але це ніяк не захищає дані на стороні самого клієнту оскільки різні частини запиту можуть бути перехоплені ще до створення запиту.

Основними перевагами HTTP є гнучкість і універсальність. Він підтримує передачу тексту, мультимедійних файлів, запитів API та всіх інших можливих видів даних. Таким чином забезпечується швидкий і ефективний зв'язок між системами в Інтернеті.

В кожному HTTP запиті повинен бути заголовок "Authorization", який є стандартним способом передачі даних аутентифікації клієнта на сервер. При використанні JWT у заголовку передається маркер доступу, який перевіряє сервер та в результаті може дозволити або заборонити доступ до ресурсів.

Даний заголовок містить у собі тип та дані для аутентифікаці. Це дозволяє відокремити аутентифікації від основної інформації у запиті та виконувати аутентифікацію окремо від основного запиту. Для JWT формат заголовка виглядає як Bearer Token, де Bearer — це лише ключове слово, яке вказує, що переданий токен є маркером доступу, а не іншим методом аутентифікації.

У розробленому вдосконаленому методі аутентифікації заголовок "Authorization" використовується для запису у нього Bearer Token з зашифрованим маркером доступу.

Слід врахувати, що оскільки HTTP без HTTPS відображатиме дані у вигляді відкритого тексту, включаючи заголовки, маркер буде доступним для перехоплення. Тому правильне використання заголовка "Authorization" вимагає використання HTTPS для його шифрування.

2.4 Проектування загальної архітектури застосунку для доступу до даних в хмарному середовищі з вдосконаленою аутентифікацією

Правильне проектування архітектури застосунку та вибір технологій для імплементації мають вирішальне значення для забезпечення її безперебійної роботи, масштабованості та відповідності бізнес-вимогам. Якщо структура застосунку спроектована з урахуванням усіх технічних обмежень це дозволить уникнути проблем, які можуть виникнути в майбутньому. Найпоширенішими з таких проблем є проблеми з обслуговуванням, низька продуктивність або неможливість включити нові функції. Недосконала структура призводить до монолітності системи, яку важко налаштувати. Правильно продумана модульна архітектура чітко розподіляє відповідальність між компонентами та спрощує розробку, тестування та обслуговування.

Технології, що обираються для реалізації безпосередньо впливають на час розробки, продуктивність і масштабованість системи. Правильний вибір мови програмування, інфраструктури та бази даних дозволить знайти найкращий баланс між часом, витраченим на розробку системи, вартістю та очікуваною продуктивністю. Довгострокова підтримка технологій є ще однією важливою причиною, яка сприяє правильному вибору. Регулярні оновлення забезпечують безпеку та актуальність застосунку. Якщо програма використовує популярні технології з великою спільнотою та має регулярні оновлення, це підвищує швидкість та ефективність розробки, тоді як вибір рідкісних або застарілих рішень може ускладнити ситуацію, коли справа доходить до майбутнього обслуговування.

Загалом, чітко розроблена структура додатку та обрана технологія зменшать ризик технічної заборгованості, зроблять систему більш гнучкою до змін та комфортною роботою як для кінцевих користувачів, так і для команди розробників.

Для створення REST API застосунку з вдосконаленим методом аутентифікації найкращим варіантом буде використання мікросервісної архітектури. Мікросервісна архітектура — це підхід до розробки програмного забезпечення, згідно з яким система складається з невеликих незалежних компонентів, які називаються мікросервісами. Кожен із них відповідає за виконання певного набору функцій або бізнес-логіки та може бути розгорнутий, масштабований та оновлений незалежно від інших.

Усі ці частини, що складають систему, є тим, що визначає основну ідею архітектури мікросервісу.

Перевагами мікросервісної архітектури є незалежне розгортання, гнучкість щодо вибору технології для кожного сервісу, висока масштабованість і підвищена надійність системи за рахунок локалізації помилок. Однак ця архітектура має певні труднощі. Управління розподіленою системою набагато складніше, зокрема зв'язок між службами та моніторинг, управління станом системи, вирішення конфліктів даних, захист системи та зменшення затримки мережі. Але ці недоліки повністю перекриваються перевагами цієї архітектури.

2.4.1 Побудова кінцевих точок для серверного застосунку з вдосконаленою аутентифікацією

Для побудови API потрібно розробити структуру кінцевих точок яку буде використовувати користувач для роботи з застосунком. Ці адреси повинні відображати роботу, що вони виконують у своїх назвах та методах, що вони використовують. Структуру кінцевих точок для застосунку з вдосконаленим методом аутентифікації описано на рисунку 2.4.

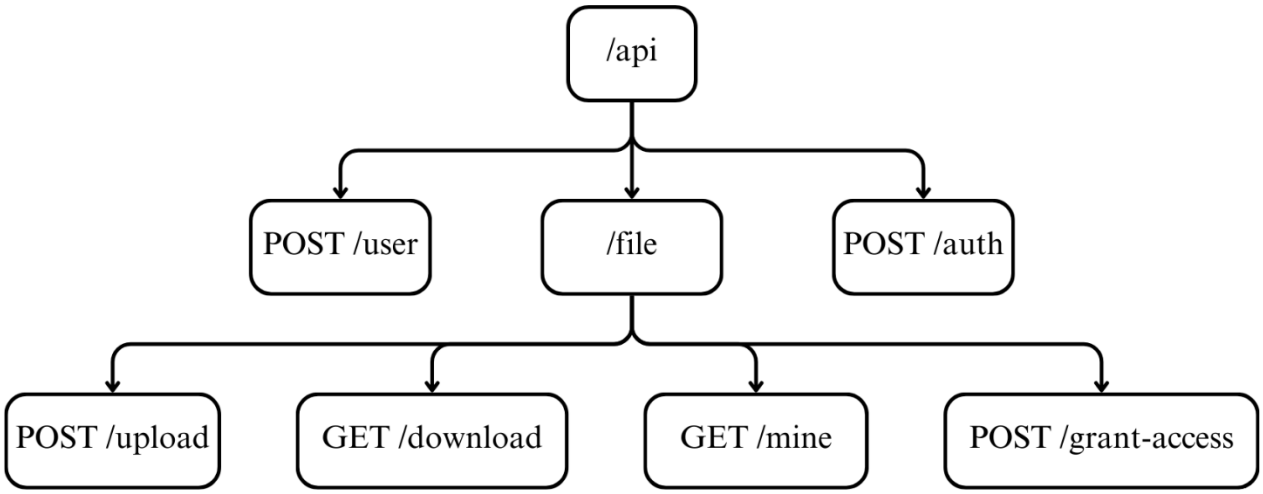

Рисунок 2.4 — Структурна схема кінцевих точок застосунку

Кінцева точка "/api/user" використовується для створення нового користувача. Вона використовується в запиті POST і виконується навіть для неавторизованих користувачів. Основна функція цієї кінцевої точки — створити нового користувача, валідувати надані дані та зберегти їх у базі даних.

Кінцеві точки в "/api/file/*" призначені для роботи з файлами. Кінцева точка POST "/api/file/upload" дозволяє авторизованому користувачеві завантажити файл на сервер. GET "/api/file/download" використовується для завантаження файлу із сервера, якщо користувач має до нього доступ. Кінцева точка GET "/api/file/mine" повертає список усіх файлів які завантажені поточним користувачем. Кінцева точка POST "/api/file/grant-access" дозволяє надати доступ до файлу іншому користувачеві якого вказаному у запиті.

Кінцева точка "/api/auth" відповідальна за генерацію маркерів автентифікації. Це виклик API методу POST, який дозволяє користувачеві після успішного введення облікових даних отримати маркер доступу для подальших операцій.

2.4.2 Обґрунтування вибору архітектурного стилю

Архітектурний стиль Representational State Transfer (REST) — це набір принципів, які визначають підхід до побудови розподілених систем, зокрема веб-сервісів. REST API є інтерфейсом, що дозволяє клієнтам взаємодіяти з сервером через HTTP-запити, щоб отримувати, створювати, змінювати або видаляти ресурси. REST визначає низку принципів, що дозволяють створити легку та ефективну архітектуру, зосереджену на ресурсах і їхньому представленні [15].

Принципи REST формують архітектуру сервісу, орієнтуючись на зручність, незалежність, та масштабованість. Всі запити обробляються незалежно один від одного, що робить кожен з них самодостатнім і дозволяє масштабувати систему. У REST також передбачено хешування для зменшення навантаження на сервер і пришвидшення доступу до ресурсів, а багатошарова система дозволяє розширити можливості, такі як балансування навантаження та безпека, без зміни основної логіки сервісу.

У REST кожен ресурс має унікальний ідентифікатор (URL або URI), і взаємодія відбувається через маніпуляцію з представленнями цих ресурсів. Таке відокремлення ресурсів від їхнього представлення дозволяє передавати дані в зручному для клієнта форматі.

REST API спирається на HTTP-методи, кожен з яких має своє призначення. У REST API використовуються стандартні методи GET, POST, PUT, PATCH та DELETE. Це дозволяє стандартизувати доступ до ресурсів і зменшує потребу в додатковій документації для базових операцій.

REST API є простим у реалізації, масштабованим і гнучким. Завдяки своїм стандартам і незалежності від платформи REST API легко інтегрується з іншими сервісами та системами, що робить його популярним вибором для веб-сервісів. Завдяки чіткому розмежуванню клієнта і сервера, REST API

дозволяє легко впроваджувати нові функції без необхідності змінювати клієнтську сторону.

REST API залишається основним архітектурним стилем для створення веб-сервісів завдяки своїй простоті, адаптивності та ефективності в роботі з розподіленими системами. Він дозволяє зручно обмінюватися даними між клієнтом та сервером через зрозумілий і стандартний інтерфейс, що полегшує взаємодію між різними системами та забезпечує гнучкість при розробці веб-застосунків і мобільних додатків.

2.5 Варіативний аналіз технологій для реалізації застосунку з вдосконаленою аутентифікацією в архітектурному стилі REST API

Вибір технологій для розробки застосунку є надзвичайно важливим, оскільки він визначає вектор розробки, впровадження та підтримки. Можливості, продуктивність, масштабованість, функції безпеки, зручність використання та використання системних ресурсів для застосунку значною мірою залежать від обраних технологій. Вибір неправильних технологій може призвести до необхідності технічних змін через деякий час, великих витрат на підтримку або навіть до необхідності переробити значну частину програми відповідно до змін бізнес-вимог. Основним чинником вибору технології є продуктивність і відповідність технології масштабам і вимогам проекту.

Безпека — ще одна складна функція. Відповідно до ступеню секретності даних, які зберігатимуться або оброблятимуться застосунком, слід обирати технології, які підтримують сучасні функції захисту. Вони повинні включати шифрування, захист від атак та ефективно використовувати ресурси. Це важливо для збереження даних користувача та завоювання його довіри до застосунку.

Рівень підтримки та масштабованість повинні досягатися легко. Розширення можливостей системи має відбуватися без суттєвих змін в її основі. Вибір популярних і перевірених рішень спрощує процес розробки та

обслуговування за рахунок великої кількості навчальних матеріалів та напрацьованих рішень.

Окрім технічних специфікацій вибрані технології мають відповідати бізнес-цілям проекту, а також бюджету та часовим рамкам. Ці технології повинні підходити для реалізації як прототипу, так і довгострокового рішення.

2.5.1 Аналіз платформи Node.js та фреймворку Nest.js

Node.js — це середовище виконання JavaScript, яке відкриває можливість розробки програм на стороні сервера використовуючи JavaScript для запуску коду поза браузером. Його створено за допомогою механізму JavaScript V8 Chrome. Це середовище є дуже продуктивним при використанні для паралельної обробки завдяки не блокуючому входу/виходу та асинхроному підходу. Node.js особливо корисний для роботи в реальному часі, і це пояснює його популярність у системах чату та потокових сервісах, а також в інших системах із високим трафіком [16].

Nest.js — це потужний фреймворк для Node.js, створений за допомогою TypeScript. Він спеціально призначений для забезпечення організованого способу розробки серверних застосунків. Nest.js дозволяє організувати модулі та контролери за допомогою декораторів, що дозволяє зберігти код чистим та модульним. Він використовує деякі бібліотеки такі як Express.js або Fastify, але розширяє їх додатковим рівнем абстракції, що значно полегшує створення масштабованих програм у чистій, підтримуваній структурі. Оскільки бізнес-логіка для кожного архітектурного шаблону зазвичай дуже різна, Nest.js дозволяє досить легко реалізувати бізнес-логіку в окремих модулях [17].

Інтеграція кількох інструментів бази даних, авторизація та тестування – це те, що спонукає Nest.js дозволити розробнику реалізувати логіку та не надто багато стандартних функцій. Заснований на TypeScript, він забезпечує баланс між набором тексту та гнучкістю, таким чином підвищуючи надійність коду. Завдяки цій структурі створення складних API і корпоративних додатків стає

більш контрольованим, і розробникам легше гарантувати, що додаток буде масштабовано.

2.5.2 Аналіз мови програмування PHP та фреймворку Laravel

Мова програмування PHP є широко використовуваною мовою для написання веб-додатків. Основні причини її популярності в тому, що вона має дуже зручний синтаксис і її легко вивчити. Це допомагає PHP реалізувати динамічні веб-застосунки. Величезна кількість бібліотек і фреймворків, що підтримують мову, значно спрощують роботу з базами даних, обробку форм, керування сесіями та інші завдання, покладені на веб-розробника. Мова PHP також є серверною мовою, призначеною для роботи в поєднанні з HTML. Вона підтримується майже всіма звичайними хостами, що може зробити впровадження та тестування програм відносно простим. Сучасна версія PHP є швидкою та безпечною завдяки регулярним оновленням і вдосконаленням [18].

Laravel — це потужний фреймворк PHP. Він був розроблений з метою зробити процес розробки більш організованим і легким. Завдяки елегантній архітектурі та використанню шаблону MVC він відокремлює логіку програми від зовнішнього вигляду. Це покращує не тільки підтримку коду, але й прискорює роботу над великими проектами, де потрібно якнайшвидше вводити зміни та розширювати можливості застосунку. Laravel поставляється з дуже багатими інструментами для роботи з базою даних без підключення додаткових залежностей. Eloquent ORM забезпечує реалізацію активного запису для роботи з базою даних. Крім того, він також постачається з інструментами, необхідними для написання міграцій і виконання транзакцій [19].

З Laravel керування запитами та відповідями значно спрощується оскільки він підтримує розширені механізми маршрутизації. Це забезпечує чітку та гнучку конфігурацію веб-маршрутів. Інші функції, що охоплюють цю

структуру, включають аутентифікацію, кешування, черги тощо, що дозволяє розробникам зосередитися на бізнес-логіці, залишаючи більшість стандартних завдань на виконання Laravel. За допомогою Artisan (інтерфейс командного рядка) виконання різних дій, пов’язаних із програмою, стає легким і швидким — компоненти програми можна створювати миттєво, а міграцію та тестування запускати миттєво. Це справді значно прискорює процес розробки та автоматизації.

2.5.3 Аналіз платформи JDK та фреймворку Spring

Java Development Kit (JDK) — це повний набір інструментів, які використовуються для розробки будь-яких видів програм Java, таких як компілятори, стандартні бібліотеки, засоби налагодження та профілювання. Це встановлений набір програмного забезпечення, який дозволяє програмістам писати код Java, компілювати його, а потім запускати на будь-якій платформі, яка підтримує Java Virtual Machine (JVM). Він базується на парадигмі об’єктно-орієнтованого програмування, яку розробники великих програм приймають через свою чіткість і масштабованість. Набір стандартної бібліотеки JDK включає низку колекцій, які спрощують маніпулювання колекцією, а також багатопотоковість, маніпулювання файлами та мережею без зовнішніх бібліотек. Найважливіше те, що завдяки широкому використанню та сумісності з багатьма операційними системами, JDK є найпоширенішим інструментом для розробки програмного забезпечення Java для бізнес-додатків [20].

Spring — це дуже потужний і гнучкий фреймворк для розробки програм Java. Spring в основному зосереджений на розподілених системах корпоративних додатків. Однак одна величезна перевага використання Spring полягає в тому, що він є модульним і підтримує інверсію керування (IoC) і ін’єкцію залежностей (DI), що дозволяє програмістам легко керувати компонентами програми, а також переконатися, що вони є незалежними.

Spring забезпечує широкі засоби створення веб застосунків і REST API за допомогою таких модулів, як Spring MVC і Spring WebFlux, а також включає засоби обробки транзакцій, безпеки, керування даними та інтеграції інших систем. Така платформа також має потенціал для створення систем, які є розподіленими та масштабованими завдяки таким компонентам, як Spring Boot, які значно спрощують конфігурацію та реалізацію проекту завдяки підходу під назвою «конфігурація за замовчуванням». У Spring архітектура основана на мікросервісах, яка надає розробникам архітектуру на основі кількох окремих і незалежних служб, які взаємодіють між собою [21].

2.6 Обґрунтування вибору стеку технологій для реалізації застосунку в архітектурному стилі REST API

Платформа Node.js набула популярності серед багатьох розробників, зацікавлених у написанні серверного коду, завдяки асинхронній природі мови та високій продуктивності в обслуговуванні багатьох користувачів одночасно. Він найбільше підходить для відносно невеликих, швидких серверів, які мають обробляти багато тисяч запитів. Node.js додатково розширює підтримку створення серверної частини програми структурованим та інтуїтивно зрозумілим способом. Він надає розробникам чудові інструменти для роботи з REST API, підтримує TypeScript і добре інтегрується з іншими технологіями, такими як GraphQL або WebSocket. Однак створення складних і високозахищених рішень, зокрема розширеного методу аутентифікації всередині нього, вимагатиме додаткового налаштування та оптимізації через відсутність такої кількості готових компонентів для роботи з функціями безпеки порівняно з іншими платформами для створення дуже великих і масштабованих програмне забезпечення.

PHP разом із фреймворком Laravel є ще одним варіантом, якому широко віддають перевагу для створення REST API. Laravel забезпечує високу абстракцію, простий синтаксис і підтримку інструментів у широкому діапазоні

речей для роботи разом із базами даних, інтерфейсами користувача та безпекою. Laravel значно полегшує розробку, оскільки він уже має вбудовані функції аутентифікації та авторизації. Він також підтримує JWT для токенів і має готове рішення безпеки API. Але PHP і Laravel найкраще застосовувати в малих і середніх проектах. Вони не дуже продуктивні для великомасштабних рішень, оскільки PHP має низьку продуктивність порівняно з іншими мовами, такими як Java, особливо при обробці багатьох одночасних запитів. Крім того, для роботи зі складнішими схемами автентифікації можуть знадобитися деякі додаткові налаштування.

Java і Spring — це інструменти, які можна використовувати для створення REST API, який є масштабованим, безпечним і продуктивним. Java використовується для підтримки високої продуктивності та багатопоточності. Він давно відомий своєю стабільністю, тому його вибирають для важких і складних систем. Фреймворк Spring використовує набір потужних інструментів для розробки REST API, який включає Spring Boot (для швидкої розробки), Spring Security (для складних механізмів аутентифікації) та Spring Data (для роботи з базами даних). Завдяки підтримці JWT, OAuth2 і багатофакторної автентифікації Spring матиме кращу ефективність у впровадженні передових практик безпеки. Крім того, Spring легко інтегрується з хмарними середовищами, такими як AWS, Google Cloud і Azure, що робить його ідеальним для розробки хмарних сервісів.

Усі ці переваги мають Java та Spring, але є кілька недоліків, таких як складність конфігурації та часом вищі вимоги до ресурсів порівняно з іншими платформами. Проте ці недоліки значною мірою перекриваються такими факторами, як стабільність і здатність ефективно масштабувати додатки навіть на рівні великих підприємств. Це робить їх найоптимальнішим варіантом для створення REST API для хмарного сховища з розширеним методом аутентифікації. Загалом, усі ці платформи мають власні сильні та слабкі сторони, але Java та Spring перевершують з точки зору масштабованості,

безпеки та можливості інтеграції в складні корпоративні рішення. Саме тому вони є найбільш відповідними мовами для створення потужного, надійного REST API для хмарного сховища.

2.7 Використання технологій шифрування для вдосконалення методу аутентифікації

Шифрування — це алгоритм за допомогою якого дані перетворюються у форму, яку майже неможливо прочитати не маючи ключа шифрування. Основна мета шифрування — переконатися, що інформація залишається конфіденційною та безпечною, навіть якщо вона потрапляє до третьої сторони. Шифрування виконується за допомогою алгоритмів, які використовують ключі для перетворення звичайного тексту в зашифрований текст або декодують із зашифрованого тексту у звичайний текст.

Існує два основних типи шифрування симетричне та асиметричне. При симетричному шифруванні використовується один ключ як для шифрування так і для дешифрування. Асиметричне ж використовує пару ключів. Відкритий використовується для шифрування, а закритий для дешифрування. Шифрування дуже широко використовується для захисту даних у різноманітних сферах як зв’язок, фінансові операції, зберігання інформації та аутентифікація [22].

2.7.1 Використання технології хешування паролю

Хешування — це одностороння функція, яка перетворює дані на рядок тексту, який неможливо обернути або декодувати. У контексті кібербезпеки хешування — це спосіб захисту конфіденційної інформації та даних, зокрема паролів, повідомлень і документів. Після перетворення цього вмісту за допомогою алгоритму хешування отримане значення (хеш-код) стає нерозбірливим для людини і його надзвичайно важко розшифрувати.

Хешування пароля призначене для захисту від несанкціонованого доступу до профілів користувачів у разі, якщо база даних зберігання користувацьких даних буде скомпрометована. Саме на цьому етапі пароль проходить через функцію криптографічного хешування, яка створить унікальний односторонній хеш-код пароля. Хешування гарантує, що навіть якщо будь-яким способом зловмисник отримає доступ до бази даних, оригінальні паролі користувачів неможливо дізнатися. Важливо, що функція хешування завжди забезпечує той самий вихід для того самого входу, який, крім того, є повністю незворотним за своєю природою; отже, неможливо відновити пароль із хешу.

Щоб підвищити безпеку хешування паролів, інші методи, які будуть використовуватися, включають застосування того, що відомо як сіль. Сіль — це лише випадкова частина даних, яка додається до пароля перед тим, як буде виконано хешування. Це гарантує, що для двох подібних вхідних даних з них не виходитимуть однакові хеш-коди. Це забороняє атаки за допомогою таблиць відповідностей, які дозволяють швидко зіставити хеш з поширеними паролями. Загалом, хеші для двох користувачів, які можуть мати однакові паролі, відрізняються, оскільки сіль для кожного різна.

Сучасні алгоритми хешування, такі як bcrypt, Argon2 та PBKDF2, були розроблені спеціально для цілей хешування паролів. Вони ускладнюють функцію хешування додатковим алгоритмом, наприклад повторним застосуванням операції хешування. Це унеможлиивлює пошук паролів методом перебору. Правильне проведення процесу хешування з використанням сучасних алгоритмів і практичних методик істотно підвищує загальний рівень кібербезпеки системи.

2.7.2 Використання шифрування маркеру доступу

Маркери доступу представлені у вигляді рядка, тому для їх шифрування потрібен алгоритм, що працює таким типом даних. Серед таких алгоритмів

стандартом у сфері шифрування вважається алгоритм AES.

Алгоритм шифрування Advanced Encryption Standard (AES) — це відносно новий, дуже ефективний симетричний блочний шифр, який використовується в режимі з одним ключем для захисту даних. Він замінює та виправляє всі недоліки які були виявлені в DES. AES є швидким, максимально ефективний, а також вважається безпечним на найвищому рівні.

AES обробляє дані у 128-бітних блоках і шифрує їх за допомогою ключів розміром 128, 192 або 256 біт. Цей алгоритм включає кілька етапів перестановки та заміни даних протягом кількох раундів. Для AES кількість раундів визначатиметься використовуваною довжиною ключа: 10 раундів для 128-бітного ключа, 12 для 192-бітного ключа та 14 для 256-бітного ключа.

AES використовує такі основні операції як SubBytes, ShiftRows, MixColumns та AddRoundKey. SubBytes змінює кожен байт використовуючи спеціальну таблицю спеціальну таблицю, що називається S-box, ShiftRows виконує зсув рядків у блоці, MixColumns зміщує дані в кожному стовпці для додатквої дифузії, а AddRoundKey додає ключ раунду до кожного блоку за допомогою бінарної операції XOR. В останньому раунді операція MixColumns пропускається, після чого процес шифрування завершується.

Алгоритм AES використовується для шифрування маркеру доступу у вигляді рядка. Використання цього методу забезпечує підвищення безпеки цих даних за рахунок чого вдосконалюється метод аутентифікації.

AES вважається дуже безпечним, оскільки він протистоїть багатьом атакам, включаючи грубу силу. Безпека алгоритму досягається за допомогою дуже великої кількості можливих ключів, складності алгоритму та сильної математичної основи.

2.7.3 Використання шифрування файлів

Під час шифрування файлу AES має дуже велику перевагу, оскільки він може обробляти дуже великі дані та має дуже складну структуру алгоритму.

За допомогою AES файли шифруються шляхом розбиття їх на 128-бітні блоки перед тим, як кожен окремий блок шифрується за допомогою секретного ключа. Ключ має бути дуже чутливим при зберіганні та передачі, оскільки він використовується лише для блокування та розблокування. У застосунку збереження ключа перекладається на користувача для підвищення безпеки даних та довіри до застосунку.

У контексті шифрування файлів дуже часто AES використовується в таких режимах роботи, як Cipher Block Chaining (CBC) або Galois/Counter Mode (GCM). Це додає ще один рівень безпеки під час операції XOR для кожного блоку даних із його попереднього зашифрованого блоку, що значно ускладнює аналіз даних, навіть якщо ми знаємо формат файлу. Оскільки алгоритм пропонує вбудоване шифрування, GCM також забезпечує аутентифікацію даних разом із ним, забезпечуючи захист від будь-якого втручання в зашифрований файл.

Практична реалізація AES для шифрування файлів також передбачає використання векторів ініціалізації (IV). Ці вектори забезпечують випадковість у процесі шифрування, навіть якщо і дані, і ключ залишаються незмінними. Будучи константою, IV також зберігається разом із зашифрованим файлом.

Головною перевагою AES із шифруванням файлів є ефективність. Цей алгоритм працює блискавично навіть при обробці величезних обсягів даних; тому його можна реалізувати в системах реального часу. Його широке застосування та стандартизація забезпечують взаємодію різних платформ і програм.

Алгоритм використовується для шифрування файлів перед завантаженням у хмарне сховище, що дозволяє додатково захистити дані, а використання ключа, що ввів користувач, при шифруванні унеможливлює доступ до даних всіх крім самого користувача. Функції цілісності та конфіденційності даних добре працюють для шифрування файлів AES під час

зберігання файлів у хмарі. Навіть якщо є компрометація сховища, дані залишаються в безпеці, оскільки без ключа неможливо буде розшифрувати дані.

3 ПРОГРАМНА РЕАЛІЗАЦІЯ ЗАСТОСУНКУ ДЛЯ ДОСТУПУ ДО ДАНИХ В ХМАРНОМУ СЕРЕДОВИЩІ З ВДОСКОНАЛЕНОЮ АУТЕНТИФІКАЦІЄЮ

Для перевірки практичної цінності та адекватної роботи вдосконаленого методу аутентифікації потрібно розробити програмну реалізацію застосунку з використанням цього методу.

3.1 Обґрунтування вибору хмарного середовища для зберігання файлів

Google Cloud Storage — це сервіс, який забезпечує зберігання різних типів даних, зокрема документів, зображень, відео, резервних копій та інших типів файлів. Висока масштабованість, що забезпечується для цього, дозволяє зберігати дуже великі обсяги даних, не потребуючи фізичного сервера для керування сховищем. Це дозволяє користувачам зберігати та обробляти дані без будь-яких обмежень щодо розміру. Google Cloud Storage має різні класи сховища, включаючи стандартне сховище для даних, до яких часто звертаються, і архівне сховище для рідкодоступних даних, щоб зменшити витрати. Він призначений для високопродуктивного доступу до даних, часто у випадку величезних аналітичних завдань, а також використовується для зберігання резервних копій даних або архівів. Доступ до файлів здійснюється через RESTful API, що дозволяє інтегрувати його з будь-якими програмами чи сервісами. Інший спосіб — через консоль Google Cloud або за допомогою інтерфейсів командного рядка для керування файлами та доступу [23].

DigitalOcean Spaces — це сховище об’єктів, розроблене компанією DigitalOcean. Сервіс дозволяє зберігати будь-який обсяг і розмір файлу без необхідності керувати основними фізичними серверами чи інфраструктурою. Це доволі простий та зручний сервіс для хмарного зберігання, який забезпечує простоту використання та необхідну масштабованість [24].

Ключовим фактором DigitalOcean Spaces є простота та інтеграція з іншими сервісами, що надаються в екосистемі DigitalOcean. Все це створює чудову можливість зберігати файли та об’єкти без прив’язки до налаштувань інфраструктури, що робить сервіс дуже зручним у налаштуванні та використанні.

Крім того, сховище в DigitalOcean Spaces доступне через Інтернет, що робить його зручним для користувача в управлінні файлами та доступом, створенні сегментів і аудиті активності. DigitalOcean Spaces забезпечує як масштабування, так і оптимізацію витрат. Це дозволяє користувачам мати план тарифів на зберігання та обсяг трафіку, це фактично дає можливість тримати витрати під контролем із зростанням обсягу даних.

Amazon S3 — це сервіс AWS, який пропонує сховище об’єктів за допомогою Amazon Web Services. Цей сервіс сам керує процесом зберігання та керування файлами, що спрощує роботу з ними. Серед усіх інших причин S3 є одним із найкращих рішень для хмарних сховищ, тому що він має високу масштабованість, надійність, безпеку та економічність [25].

Найбільша перевага Amazon S3 полягає в тому, що користувачі можуть зберігати будь-які обсяги даних, не турбуючись про фізичні сервери, їх обслуговування чи масштабування інфраструктури. Автоматично S3 підтримує масштабоване навантаження і підтримку великої кількості запитів.

Кожен об’єкт на Amazon S3 зберігається як файл разом із метаданими, що описують цей файл. Ці об’єкти зберігаються в контейнерах, відомих як відра (buckets). Доступ до кожного об’єкта всередині відра можна отримати на основі унікальної назви відра, в якому він знаходиться. Amazon автоматично копіює дані в різних регіонах, щоб гарантувати доступність і стійкість. Amazon надає високі гарантії доступності даних протягом з різних точок світу.

Amazon S3 має багато переваг перед DigitalOcean Spaces і Google Cloud Storage. S3 пропонує найкращу масштабованість, надійність і гнучкість. Він підтримує службу високої доступності даних із гарантіями надійності,

підтримує кілька класів зберігання для оптимізації ресурсів, має потужні інструменти безпеки та контролю доступу. Ключовою перевагою S3 є потужна підтримка в екосистемі AWS, яка, безсумнівно, дає змогу легко та зручно інтегрувати інші сервіси. Велика кількість можливостей, таких як шифрування під час передачі, можливість визначати політику життєвого циклу та багато API, дозволяють S3 робить його найкращим за своїми перевагами серед конкурентів.

3.2 Налаштування обробки запитів з аутентифікацією

На рисунку 3.1 представлено UML-діаграму класів сервісів створення та шифрування JWT токенів. Класи AESEncryptionService та JwtService реалізують шифрування та генерування токенів JWT.

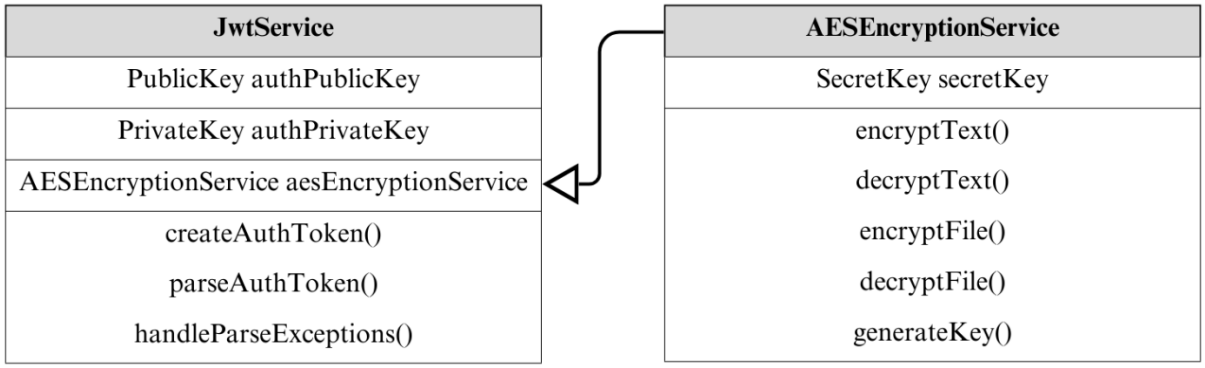

Рисунок 3.1 — UML-діаграма класів JwtService та AESEncryptionService

Клас AESEncryptionService реалізує шифрування та дешифрування даних за допомогою алгоритму AES. Секретний ключ отримується через конструктор, декодуючи його з формату Base64 для ініціалізації об’єкта SecretKey, який і буде використовуватися для шифрування та дешифрування даних.

Щоб виконати шифрування, викликається метод encryptText() який подано у лістингу 3.1 та у який передається рядок для шифрування. Цей метод повертає зашифрований алгоритмом AES результат.

Лістинг 3.1 — Метод encryptText()

```
public String encryptText(String plainText) throws Exception {
    Cipher cipher = Cipher.getInstance(AES_ALGORITHM);
    cipher.init(Cipher.ENCRYPT_MODE, secretKey);
    byte[] encryptedBytes = cipher.doFinal(plainText.getBytes());
    return Base64.getEncoder().encodeToString(encryptedBytes);
}
```

Для дешифрування викликається метод decryptText(), що наведено у лістингу 3.2. Він розшифрує зашифрований текст і повертає початковий текст після його декодування за допомогою алгоритму AES.

Лістинг 3.2 — Метод decryptText()

```
public String decryptText(String encryptedText) throws Exception {
    Cipher cipher = Cipher.getInstance(AES_ALGORITHM);
    cipher.init(Cipher.DECRYPT_MODE, secretKey);
    byte[] decodedBytes = Base64.getDecoder().decode(encryptedText);
    byte[] decryptedBytes = cipher.doFinal(decodedBytes);
    return new String(decryptedBytes);
}
```

Схему шифрування маркеру доступу подано на рисунку В.1. У ній продемонстровано як JWT який складається з заголовку, корисного навантаження та підпису зашифровується алгоритмом AES у стрічку яку не можна декодувати без відповідного ключа.

Клас JwtService відповідає за створення та перевірку JWT. Цей сервіс потребує приватний і публічний ключ для підпису та перевірки токенів. Він створює маркер аутентифікації за допомогою методу createAuthToken(). У токен записується алгоритм кодування, ідентифікатор, дата створення та термін дії маркера. Також цей метод підписує маркер закритим ключем. Після створення цього маркера сервіс шифрує його за допомогою AES за запитом інтегрованого сервісу шифрування, який описано вище. Метод parseAuthToken() призначений для забезпечення його аутентичності шляхом перевірки підпису використовуючи відкритий ключ. Також усередині цього методу використовується AESEncryptionService для розшифрування маркеру перед валідацією. При будь-яких винятках при аналізу маркерів викликається функція handleParseExceptions(), яка видає відповідні помилки для будь-яких проблем з маркерами.

Загалом ці сервіси реалізують вдосконалений метод аутентифікації за меркером доступу використовуючи можливості та зручність JWT і переваги безпеки від шифрування AES.

3.3 Розробка логіки для створення та валідації JWT

Для роботи з маркером доступу у Spring спочатку потрібно реалізувати обробку маркеру перед виконанням запитів. Це робиться шляхом відділенням маркера від інших даних у HTTP, перевірки цього маркера, а потім перетворення його в об’єкт аутентифікації, який дозволить чи заборонить користувачеві доступ до ресурсів.

Блок схему алгоритму вдосконаленої аутентифікації користувача зображено на Б.1.

Для реалізації аутентифікації у Spring використовується Spring Security.

Spring Security — це дуже ефективний та гнучкий фреймворк, для забезпечення безпеки у програмах на Java. Він є частиною екосистеми Spring і реалізує функції для аутентифікації, авторизації, захисту від CSRF, XSS та

інших можливих атак. Цей фремворк тісно пов’язаний з іншими компонентами Spring, для легкого та швидкого створення безпечних додатків з високим рівнем захисту.

Основою у Spring Security є використання фільтрів, які отримують HTTP-запити, обробляють їх, перевіряють, які права потрібні під час визначення доступу, обмеження на основі ролей чи інші методи. Авторизація, охоплює визначення того, які ресурси або дії може виконувати за певних наданих повноважень. Spring Security забезпечує просту ієрархію ролей, яку можна визначити в конфігурації. Крім того, фреймворк має механізми обробки помилок доступу, надання спеціальних сторінок входу або виходу з системи та захист від кількох поширених веб-загроз. Основну логіку Spring Security описано в класах, що зображено на UML-діаграмі класів на рисунку 3.4.

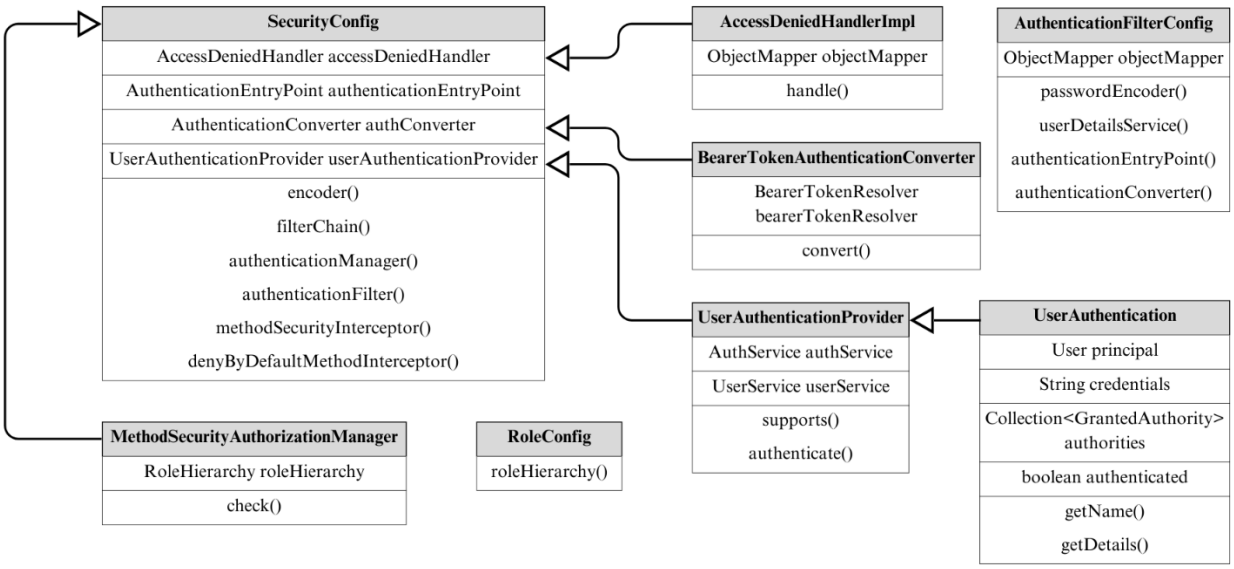

Рисунок 3.4 — UML-діаграма класів, що реалізують логіку Spring Security

Клас BearerTokenAuthenticationConverter призначений для отримання маркера Bearer із запиту. Він робить це за допомогою об’єкту класу BearerTokenResolver для вилучення маркера із заголовка запиту. Якщо маркер знайдено, він передається в метод tokenToAuthentication, де на його основі створюється об’єкт аутентифікації типу UserAuthentication. Якщо з якоїсь

причини токен не вдалося знайти або виникли проблеми з його обробкою, буде створено виключення AuthException.

Клас UserAuthentication реалізує інтерфейс аутентифікації та визначає користувача в контексті Spring Security. Він зберігає маркер як облікові дані, а також містить інформацію про користувача, якщо такий є, і статус аутентифікації. Коли використовується JWT, цей клас створюється на основі маркера зберігаючи у собі об’єкт користувача для зберігання його у контексті.

UserAuthenticationProvider — це компонент у якому реалізовано ідентифікацію користувача та його ролей для використання при запиті. Тут він перевіряє, чи підтримується ним переданий тип аутентифікації, а саме UserAuthentication. Після цього, викликається метод сервісів AuthService та UserService. Метод getUserIdFromToken() валідує JWT та повертає ідентифікатор з поля sub. Метод findById () виконує пошук користувача за ідентифікатором у базі даних. Після того, як користувача буде знайдено, для нього буде створено об’єкт UserAuthentication із призначеними правами доступу та аутентифікація цього користувача буде позначена як успішна шляхом встановлення значення true.

Клас AccessDeniedHandlerImpl реалізує обробку відмов у доступі. Якщо в запиті користувач намагається отримати доступ до будь-якого ресурсу, для якого йому не призначено необхідні дозволи, цей клас сформує й поверне відповідь з кодом 403 і надішле вміст JSON із повідомленням про заборону доступу.

Клас AuthenticationFilterConfig налаштовує основні компоненти, пов’язані з аутентифікацією. У ньому конфігуруються кодувальник паролів, сервіси обробки даних користувача та обробники аутентифікації та авторизації.

MethodSecurityAuthorizationManager перевіряє права доступу до методів на основі анотацій на рівні контролеру. Також він встановлює доступні ролі

для користувача, якщо авторизацію можна виконати. Він порівнює права доступу з ролями за допомогою ієрархії ролей.

Саме у конфігураторі RoleConfig відбувається налаштування ролей та їх ієрархія. Цей клас визначає ролі для анонімних користувачів та користувачів, які ввійшли в систему.

Клас SecurityConfig налаштовує загальну безпеку застосунку. Це стосується аспектів захисту фільтрів, безпеки обробки винятків, а також аутентифікації та авторизації менеджером аутентифікації. Він також визначає правила доступу до певних ресурсів, які можуть мати лише користувачі з певними ролями.

Таким чином, ці два класи охоплюють аутентифікацію, авторизацію та обробку помилок доступу для застосунку Spring. Вони значною мірою забезпечують безпеку, обмежуючи доступ до деяких ресурсів на основі анотацій для ролей користувачів.

3.4 Програмна реалізація структури застосунку з використанням MVC шаблону

Архітектурний шаблон Spring MVC передбачає розділення програми на різні компоненти. Це встановлює рівні модульності, сприяє масштабованості та простоті підтримки. Також надаються інструменти для створення цих застосунків із чітким розділенням логіки на різні аспекти.

Контролер у Spring MVC є центральним компонентом, який отримує запити на основі протоколу HTTP, які надсилає користувач. Контролер обробляє запити та повертає відповіді клієнту. Контролер використовує сервіси для виконання бізнес-логіки при обробці запитів.

Моделі є стандартами для будь-якої бізнес-логіки або операцій з базою даних. Модель складатиметься із сервісів, які виконують бізнес логіку, та репозиторіїв, які забезпечують транзакції БД. Цього можна досягти у Spring шляхом використання анотацій для створення сервісу (@Service) і репозиторія

(@Repository). Модель реалізує всю логіку для отримання, оновлення, зберігання та видалення даних.

Представлення відповідає за відображення результату користувачеві. Для REST API зазвичай використовуються об'єкти у форматі JSON. Він отримує дані, підготовлені моделлю, а потім повертає їх користувачеві у належному форматі.

Головним контролером у веб-фреймворку Spring є DispatcherServlet. Це точка входу в цикл відправки для Spring MVC. Усі HTTP запити направляються через цей компонент. Він відповідає за обробку запитів і відправлення їх у відповідний контролер. DispatcherServlet також інтегрується з іншими Spring API для обробки валідації та створення контейнерів для сервісів.

На рисунку 3.5 зображено модульну структуру застосунку у архітектурі MVC із використанням різних рівнів обробки даних і служб для авторизації, керування користувачами та файлами.

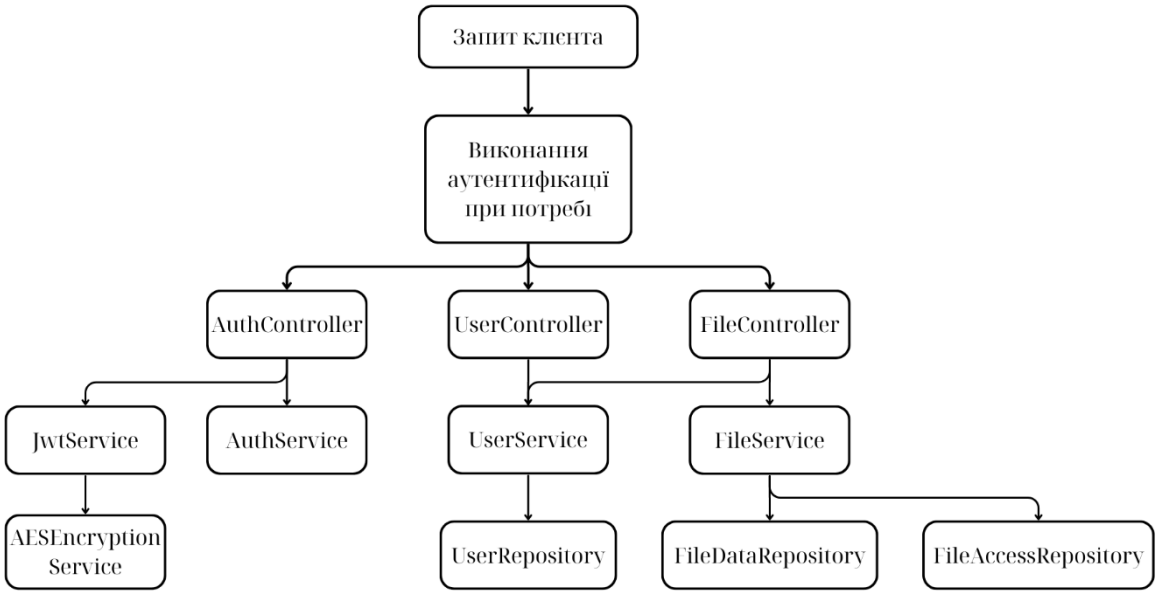

Рисунок 3.5 — Структурна схема модулів застосунку

Запит клієнта надходить до серверу і за потреби аутентифікує користувача. Після цього пересилає запит на вказаний контролер — AuthController, UserController або FileController. До обов’язків контролерів входить обробка запитів і взаємодія з відповідними сервісами.

AuthController відповідає за аутентифікацію та авторизацію. Це робиться шляхом виклику функцій сервісу AuthService, які виконують генерацію маркерів доступу та їх перевірку за допомогою JwtService. JwtService використовує AESEncryptionService для шифрування та розшифрування рядків, що використовується до JWT та шифрування то розшифрування файлів, що використовується перед збереженням та вивантаженням файлів з хмарного сховища. UserController делегує керування інформацією користувача UserService. Сервіс виконує бізнес-логіку пов'язану з користувачами та звертається до UserRepository для доступу до бази даних. FileController керує файловими операціями, а саме завантаженням файлів, збереженням файлів і керуванням доступом до файлів. Він делегує виконання бізнес-логіки керування правами доступу та шифрування файлів сервісу FileService. FileAccessRepository керує запитами файлових служб щодо інформації про те чи дозволено запиту доступ до певного файлу та хто може отримати доступ до файлів. FileDataRepository відповідає за збереження інформації про файли збережені у хмарному сховищі.

Ця архітектура розділяє програмну логіку між контролерами, сервісами та репозиторіями. Це робить систему масштабованою та спрощує тестування, обслуговування та розробку. Взаємодія між компонентами побудована безпечно та зручно для використання. Логіка доступу до файлів і даних забезпечує найвищий ступінь контролю користувачів над своїми ресурсами.

3.5 Налаштування маршрутизації між кінцевими точками для HTTP-запитів

У Spring розробка кінцевих точок виконуються у класах контролерах, які оброблятимуть HTTP-запити та далі делегуватимуть їх певній службі для виконання цієї конкретної операції. Контролери, як правило, реалізовані з деякими спеціальними анотаціями, наприклад, @RestController, який виконує серіалізацію об’єкта у формат JSON і передає його у відповідь. Кінцеві точки визначаються за допомогою анотацій, таких як @GetMapping, @PostMapping тощо, де вони визначають методи HTTP та шлях для кожної з них. Для забезпечення безпеки також можна використовувати анотації, такі як @Secured, які додатково обмежують доступність кінцевої точки на основі ролей користувача. Приклад використання анотацій наведено у лістингу 3.3, що зображає кінцеву точку отримання маркеру доступу.

Лістинг 3.3 — Кінцева точка отримання маркеру доступу

```
@PostMapping
@Secured(ApiRoles.ANONYMOUS)
public ResponseEntity<AuthResponse> createAuthToken(@RequestBody CreateAuthTokenRequest request) {
    User user = userService.findByUsername(request.username());
    if (passwordEncoder.matches(request.password(), user.getPassword())) {
        String token = jwtService.createAuthToken(user);
        return new ResponseEntity<>(new AuthResponse(token), CREATED);
    }
    throw new AuthException(new Error("Wrong password"));
}
```

Для повернені відповідей з контролерів у Spring використовується клас ResponseEntity, що дозволяє тонко налаштовувати відповіді HTTP. Він дає

змогу вказати тіло відповіді, код статусу та заголовки надаючи детальний контроль над вмістом HTTP відповіді, яка надсилається від сервера до клієнта. Головна перевага ResponseEntity полягає в тому, що він гнучкий. Цей клас дозволяє динамічно налаштовувати відповідь залежно від обробки запиту. Завичай інформація яку повертає сервер записується у тіло. Вона містить результат роботи запиту або повідомлення про помилку під час його виконання. Цей клас також інтегрується з анотацією @ResponseBody для безпроблемної роботи, перетворюючи об’єкти Java у відповідний формат за типом. Також за допомогою цього класу можна самому визначати тип тіла HTTP за допомогою встановлення відповідного типу в заголовок “Content Type”.

У контролері аутентифікації створено кінцеву точку для створення маркерів доступу. Цей метод збирається перевірити облікові дані за допомогою сервісу користувачів та після підтвердження правильності створити маркер через JWT сервіс. Якщо виникла помилка, у відповідь контролер повертає відповідний код стану та повідомлення про помилку з поясненням проблеми.

Контролер керування файлами має кінцеві точки для завантаження файлів, переліку файлів поточного користувача і надання доступу до файлу. Кінцева точка завантаження файлів дозволяє аутентифікованим користувачам завантажувати файли для зберігання за допомогою сервісу, що інтегрований з AWS S3 для зберігання файлів. Інша кінцева точка дозволяє користувачам завантажувати власні файли з хмари, якщо користувач має права доступу до цих файлів. Також імплементована кінцева точка, яка повертає список файлів до яких є доступ у поточного користувача. Остання точка надає користувачеві доступ до файлу, зберігаючи відповідні права доступу в системі.

Контролер користувачів містить кінцеву точку для створення нових користувачів. Він приймає облікові дані, передає їх службі користувача, яка створює новий об’єкт користувача та зберігає його в базі даних. Після

успішної обробки даних контролер повертає відповідь із новоствореним користувачем або обробляє відповідні винятки.

3.6 Розробка ER-моделі та структури бази даних

В основному існує два типи баз даних —– Structured Query Language (SQL) і Not Only SQL (NoSQL). Кожен тип з яких має власний набір характеристик, переваги, недоліки та способи використання.

Бази даних SQL є моделлю структурованих даних. Вони продовжують маніпулювання даними через SQL, маючи команди для запиту, вставки, оновлення та видалення даних. Реляційні бази даних — це ті бази даних, які зберігають свої дані у рядках і стовпцях, які також мають суворі схеми та типи даних для додаткового захисту даних. Бази даних SQL зазвичай використовуються для виконання більш складних транзакцій, що має велике значення для підтримки точності та узгодженості даних. Принципи ACID підтримуються у SQL, де стверджується, що всі операції мають бути завершеними, узгодженими та ізольованими, з гарантованою сталістю навіть після збоїв.

Бази даних NoSQL більш гнучкі і не потребують схем. Вони можуть підтримувати документи, графи, стовпці або ключ-значення для підтримки підвищеної гнучкості відповідно до вимог застосунку. У звичайному концептуальному випадку бази даних NoSQL застосовуються до великих обсягів для спрощеного масштабування, де дані також можуть швидко змінюватися або бути неструктурованими. Зазвичай вони працюють за принципом BASE, що дозволяє пожертвувати деякими аспектами узгодженості надаючи високу доступность та швидкість.

Бази даних SQL найкраще підходять для програм, які вимагають високої цілісності даних і складних транзакцій, а NoSQL використовуються в середовищах, де швидка масштабованість і обробка великих обсягів даних мають перевагу.

Liquibase — це програмний інструмент, який допомагає керувати змінами у базах даних і спрощує роботу зі структурою будь-якого програмного проекту. Liquibase допомагає створювати, оновлювати та підтримувати структуру бази даних із програмними проектами. Зміни схеми бази даних автоматично застосовуються Liquibase для забезпечення кращого рівня узгодженості та відстеження в різних середовищах.

Основні поняття Liquibase обертаються навколо файлів змін (change logs), які описують зміни, внесені до схеми бази даних у різних форматах, таких як XML, YAML, JSON або SQL. Ці файли діють як свого роду журнал, де зберігається вся історія внесених змін, зокрема створення таблиць, додавання стовпців, створення індексів, зміна типів даних тощо. Операції, описані в цих файлах, можуть виконуватися інструментом автоматично в будь-якому бажану послідовність, незалежно від середовища чи версії бази даних.

Liquibase можна використовувати з CLI, Maven, Gradle або іншими інструментами, завдяки чому він може легко вписуватися в процес розробки. Таким чином, він може бути частиною конвеєрів CI/CD, виконуючи базові зміни кожного разу, коли новий код надсилається. Інформація про застосовані зміни зберігається Liquibase у спеціальній таблиці системи керування базами даних для забезпечення однорідності. Цей процес допомагає уникнути дублікатів операцій.

Однією з головних переваг Liquibase є те, що він підтримує більшість баз даних, включаючи MySQL, PostgreSQL, Oracle, SQL Server та інші. Це дозволяє йому бути універсальним рішенням для проектів, що реалізуються в умовах дуже різних технологічних стеків. Liquibase також підтримує оборотність змін. Тобто він може визначати сценарії «відкату», що дозволяє повернутися до попередньої версії базової схеми при необхідності. Це зменшує ймовірність помилок на будь-якому етапі оновлення бази даних, а також прискорює та забезпечує безпечне розгортання для проектів будь-якого

розміру та складності. За допомогою Liquibase розробники можуть зосередитися на реалізації логіки та перестати турбуватися про ручне відстеження сценаріїв, які використовувалися чи не використовувалися для оновлення конкретної бази даних.

Для розробки структури бази даних потрібно створити та описати ER-модель, що зображено на рисунку 3.6. У ній присутні три сутності: users, file_data та file_access.

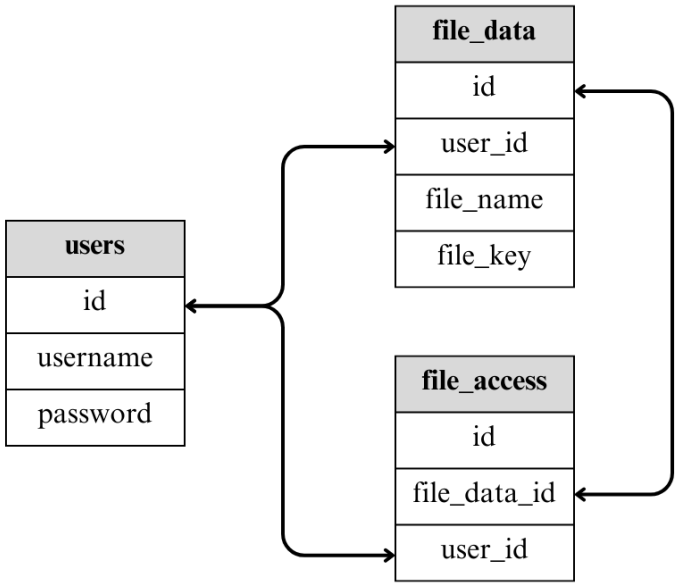

Рисунок 3.6 — ER-модель бази даних користувачів

Сутність users зберігає у собі інформацію про користувачів та складається з таких полів, як id — унікальний ідентифікатор користувача (первинний ключ), представлено у форматі bigint; username — ім’я користувача, представлено у форматі varchar; password — пароль користувача (захешований), представлено у форматі varchar.

Сутність file_data зберігає у собі інформацію про файли та складається з таких полів, як id — унікальний ідентифікатор даних файлу (первинний ключ), представлено у форматі bigint; user_id — ідентифікатор користувача, зовнішній ключ пов’язаний з полем id з таблиці users, представлено у форматі

bigint; file_name — оригінальне ім’я файлу, представлено у форматі varchar; file_key — ключ файлу збереженого в хмарному сховищі, представлено у форматі varchar.

Сутність file_access зберігає у собі інформацію про доступ користувачів до файлів та складається з таких полів, як id — унікальний ідентифікатор доступу (первинний ключ), представлено у форматі bigint; file_data_id — ідентифікатор даних файлу, зовнішній ключ пов’язаний з полем id з таблиці file_data, представлено у форматі bigint; user_id — ідентифікатор користувача, зовнішній ключ пов’язаний з полем id з таблиці users, представлено у форматі bigint.

ER-модель представляє чітку структуру взаємозв'язків між користувачами, файлами та їх доступом до них. Крім структури сутностей у ній також зображено зв’язки між таблицями.

Таблиці users та file_data пов’язані полями id та user_id відповідно. Ці таблиці пов’язані зв’язком один до багатьох, оскільки один користувач може бути власником багатьох файлів.

Таблиця users також пов’язана з таблицею file_access полями id та user_id. Ці таблиці пов’язані зв’язком багато до багатьох, оскільки багато користувачів можуть мати права доступу до багатьох файлів.

Таблиця file_data пов’язана з file_access полями id та file_data_id. Ці таблиця пов’язані зв’язком один до багатьох, оскілки один файл може мати багато прав доступу.

Ці таблиці використовуються у бізнес логіці та застосунок використовує запити для роботи безпосередньо з даними зі цих таблиць.

Крім того для роботи бібліотеки Liquibase потрібні сутності databasechangelog та databasechangeloglock, зображені на рисунку 3.7.

Як видно з графічного зображення сутностей між сутностями немає ніяких зв’язків, адже всю логіку виконує бібліотека Liquibase, що

використовує обрану базу даних та файл з прописаними змінами у форматі SQL запитів.

databasechangeloglock
id
locked
lockgranted
lockedby

databasechangelog	
id	description
author	comments
filename	tag
dateexecuted	liquibase
orderexecuted	contexts
exectype	labels
md5sum	deployment_id

Рисунок 3.7 — Сутності для роботи Liquibase

Сутність databasechangeloglock зберігає у собі інформацію про блокування змін у базі даних та складається з таких полів, як id — унікальний ідентифікатор доступу (первинний ключ), представлено у форматі bigint; locked — прапор, що визначає чи зміни заблоковані, представлено у форматі boolean; lockgranted — дата блокування даних, представлено у форматі timestamp; lockedby — рядок з ім’ям хто заблокував зміни, представлено у форматі varchar.

Таблиця databasechangeloglock використовується для забезпечення блокування, коли справа доходить до виконання міграції бази даних у Liquibase. Вона запобігає одночасному внесенню змін до бази даних декількома процесами або екземплярами Liquibase. Таким чином, це стає абсолютно необхідним у розподілених системах або навіть під час процесу CI/CD, щоб не викликати конфліктів і не пошкодити структуру бази даних.

Сутність databasechangelog зберігає у собі інформацію виконання змін у базі даних та має такі основні поля, як id — унікальний ідентифікатор доступу (первинний ключ), представлено у форматі varchar; author — рядок з іменем

автора змін, представлено у форматі varchar; filename — рядок з іменем файлу, що містить зміни, представлено у форматі varchar; dateexecuted — дата виконання змін, представлено у форматі timestamp; md5sum — контрольна сума змін, представлено у форматі varchar; description — рядок з описом змін, представлено у форматі varchar; contexts — набір контекстів у яких застосовується ця зміна, представлено у форматі varchar; deployment_id — унікальний ідентифікатор розгортання, що вказує, під час якого розгортання ця зміна була застосована, представлено у форматі varchar;

Liquibase використовує таблицю databasechangelog для відстеження всіх змін, застосованих до бази даних. Ця інформація про зміни бази даних забезпечує шлях для відстеження та повернення цих змін до попередньої версії. У цій таблиці представлено історію змін, що дає змогу дізнатися, які зміни було внесено до конкретного екземпляра бази даних.

Ці сутності не пов’язані з бізнес логікою і не використовуються в застосунку напряму та мають лише сервісне призначення.

Для створення й використання SQL запитів до цієї бази даних використовується Hibernate.

Hibernate — одна з найпопулярніших бібліотек об’єктно-реляційного відображення (ORM) у Java, яку можна використовувати для відображення таблиць з реляційної бази даних у вигляді об’єктів Java. Ця бібліотека забезпечує автоматизацію багатьох завдань, пов’язаних із базами даних, таких як збереження, отримання та оновлення даних. Основна мета Hibernate — спростити взаємодію з базами даних і за потреби перетворити об’єктно-орієнтований код у реляційну модель даних і навпаки. Використовуючи Hibernate, користувачі можуть працювати з базами даних за допомогою об’єктів Java, а не через запити SQL, що спрощує код і підвищує продуктивність розробки.

Hibernate має кілька методів запитів, таких як Hibernate Query Language (HQL) і Criteria API. HQL дає змогу створювати запити на основі об’єктів Java

замість таблиць бази даних, тому код залишається об’єктно-орієнтованим. З іншого боку, Criteria API дозволяє створювати запити через об’єкти динамічним способом, забезпечуючи таким чином більшу гнучкість.

Однак однією з найцінніших функцій, які пропонує Hibernate, є його здатність кешувати, що призводить до скорочення викликів доступу до бази даних, тим самим підвищуючи продуктивність програми. Hibernate додатково підтримує керування транзакціями, яке підтримує узгодженість даних під час складних операцій. Використання Hibernate сильно знижує складність роботи з базою, значно спрощує код та ефективно працює з даними.

Для імплементації сутностей бази даних використовуються класи з анотацією @Entity які визначають поля та зв’язуються з відповідною таблицею у базі даних. Також у таких класах за допомогою використання відповідної анотації визначаються зв’язки між таблицями для подальшого використання у бізнес логоці.

Для зв’язування моделі з відповідною таблицею у Hibernate використовуються інтерфейси репозиторії. Для визначення такого інтерфейсу його потрібно наслідувати від JpaRepository в якому визначені стандартні методи доступу до даних з таблиць. Також у цих інтерфейсах можна визначити новий метод для вибірки даних на основі полів у моделі.

4 РОЗГОРТАННЯ ТА ТЕСТУВАННЯ СИСТЕМИ

Для підтвердження адекватності запропонованого вдосконаленого методу авторизації для доступу до даних в хмарному середовищі потрібно розгорнути розроблений застосунок та виконати його тестування. Також для опису функціональних можливостей застосунку потрібно задокументувати кінцеві точки.

4.1 Створення та налаштування Docker контейнеру

Для розгортання застосунку в реальних умовах зазвичай використовується Doker. Docker — це платформа, яка використовується для керування автоматизацією розгортання, інтеграції та керування додатками у окремих контейнерах. Це дає розробникам можливість упаковувати свої програми разом із усіма залежностями в ізольовані контейнери з можливістю запускати їх на будь-якій машині з підтримкою Docker, переміщувати контейнери без необхідності вручну налаштовувати середовище, що значно спрощує розгортання та мінімізує потенційні проблеми від розбіжностей у конфігураціях середовища. [26]

Контейнер Docker — це практично легке віртуальне середовище, яке робить можливим ізоляцію додатків від основної ОС та інших додатків. Кожний контейнер має свій файловий простір для запису. Docker має власні бібліотеки, налаштування та все програмне забезпечення, яке потрібне для роботи програми. Це робить його сумісним і стабільним у різних середовищах. Його використання робить додатки дуже портативними, оскільки контейнер може працювати на будь-якому сервері чи хмарі, де запущено Docker. Такі властивості особливо корисні для розробки та тестування, оскільки за його допомогою можна переконатися, що застосунок працюватиме нормально в будь-якому середовищі.

Крім того, Docker надає набори інструментів для керування контейнерами — для їх створення, тестування, запуску та масштабування. Це стає особливо важливим для великих і розподілених систем. Він підтримує менеджмент контейнерів, тому ви можете автоматизувати та відслідковувати його роботу за допомогою таких інструментів як Docker Compose та Kubernetes. Усі ці можливості зробили Docker одним із основних інструментів у сучасному процесі розробки програмного забезпечення.

Щоб створити Docker контейнер для початку ініціалізується застосунок Spring Boot, компілюється проект і витягається JAR файл з цього проекту. Далі створюється Dockerfile. Dockerfile схожий на рецепт для створення образу, який визначає, що таке базовий образ, які всі типи залежностей потрібно встановити та які команди слід запускати для запуску програми. Цей файл являє базовий образ, залежності та команди, необхідні для виконання програми. Як правило, одна з широко застосовуваних стратегій полягає в тому, щоб вибрати базовий образ, наприклад OpenJDK, а потім перемістити файл JAR туди.

Створивши свій Dockerfile потрібно продовжити створення образу Docker, який міститиме застосунок. Щоб запустити цей образ, використовується команда docker run, визначаючи відповідні порти та змінні середовища, якщо це необхідно під час виконання.

Для оптимізації можна налаштувати багатоетапну збірку. Проект будується в контейнері, потім готовий файл JAR копіюється в кінцевий легкий образ. Таким чином розмір остаточного зображення Docker зменшується.

4.2 Документування для кінцевих точок

Документування кінцевих точок REST API є обов’язковою умовою для ефективного використання застосунку. Документація допомагає розробникам швидко зрозуміти, як працює API, які існують кінцеві точки, які дані надсилати для запитів і яких відповідей слід очікувати. Це зменшить час

інтеграції та допомагає уникнути помилок, які виникають через неправильне використання API.

Кінцеві точки мають бути задокументовані для зручності обслуговування та оновлення API. Це зручний список всіх кінцевих точок з їх повним описом. Документація також стосується тестування та налагодження. Тут також описуються можливі помилки, коди стану та способи їх використання, щоб зрозуміти, що має робити кожна кінцева точка.

Також для зовнішніх клієнтів ця документація є основним ресурсом, який демонструє можливості служби. Це впливає на сприйняття API як надійного та добре організованого продукту. На рисунку 4.1 зображено графічне представлення кінцевих точок за допомогою Swagger.

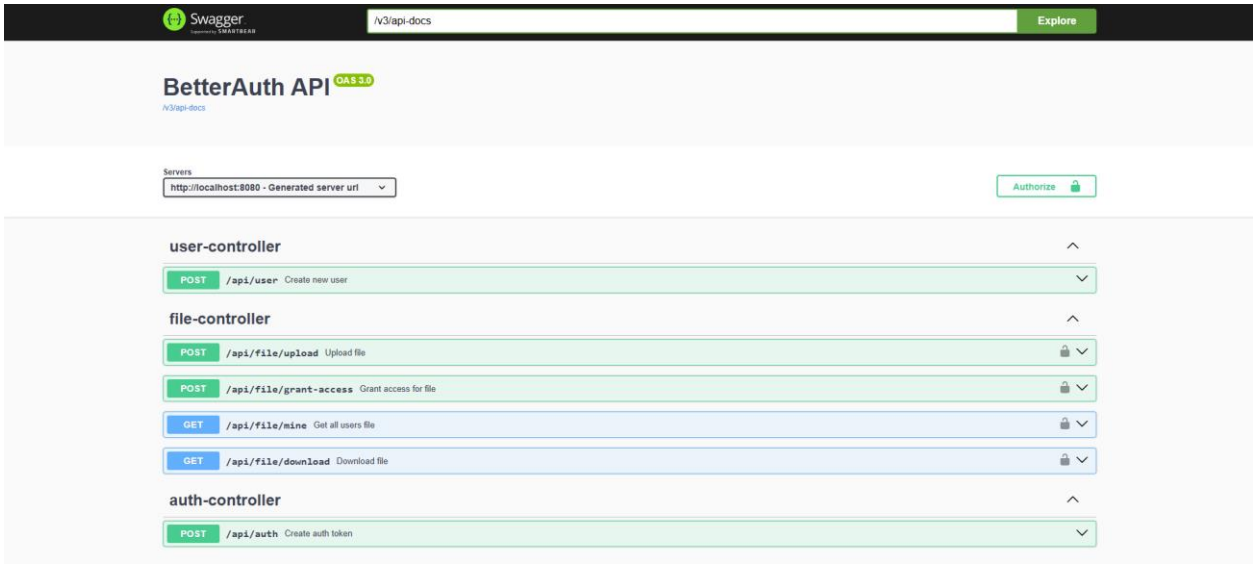

Рисунок 4.1 — Графічне представлення кінцевих точок за допомогою Swagger

Swagger — це інструмент, який допомагає створювати, документувати та тестувати REST API. Практичні реалізації дозволяють автоматизувати процес генерації такої інтерактивної документації API, щоб розробникам і навіть тестувальникам було справді легко їх використовувати. Сьогодні більшість програм Spring Boot використовують бібліотеку OpenAPI, яка

включає Swagger. За допомогою Swagger можна описати кінцеві точки свого API, доступні методи HTTP, параметри, тіла запитів і відповіді в одному місці. Інформація виводиться у форматі JSON або YAML і може бути перетворена у зручний для читання набір інтерактивної документації API. Документація для одного з методів застосунку зображено на рисунку 4.2

Основна перевага Swagger полягає в тому, що він надає інтерактивну документацію, яка дозволяє споживачам API не тільки переглядати можливості, надані API, але й фактично використовувати їх безпосередньо через веб-додаток. Це може стати в нагоді під час тестування кінцевих точок або демонстрації деяких функцій.

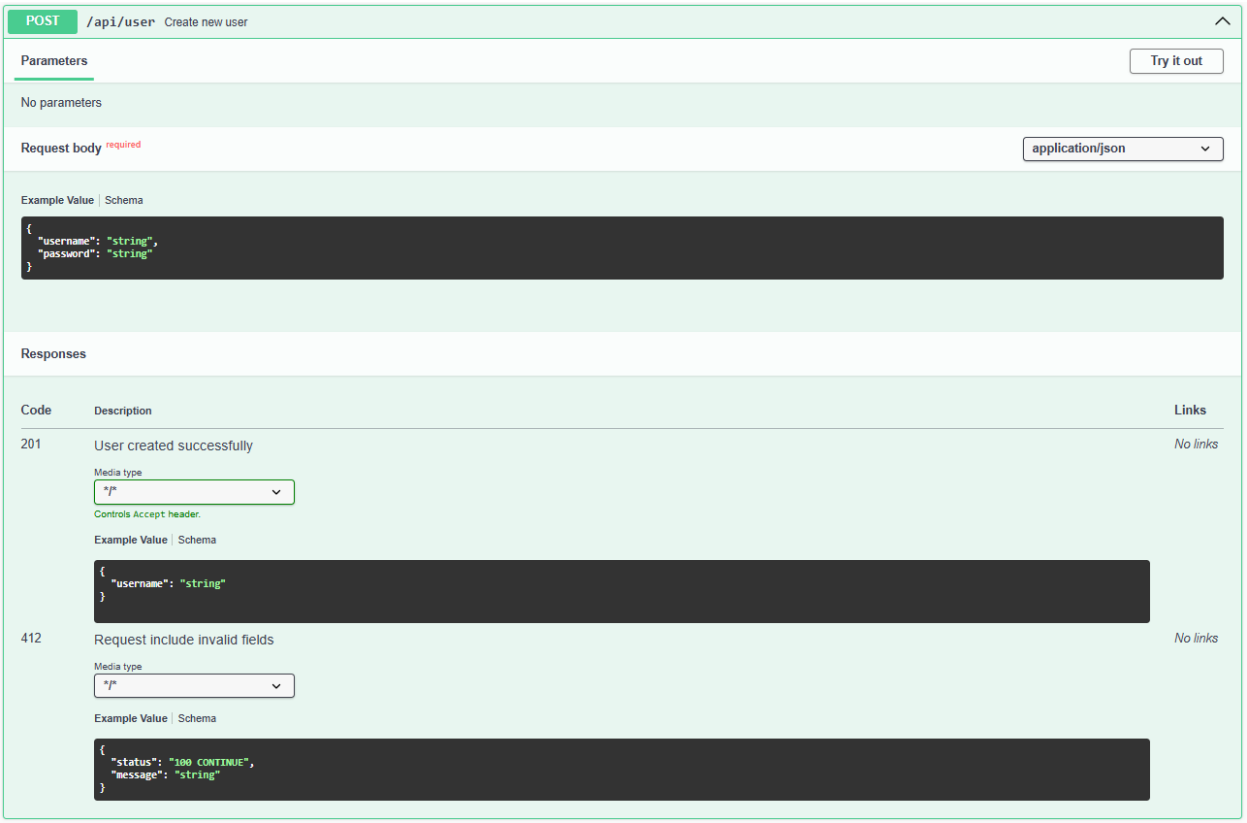

Рисунок 4.2 – Опис кінцевої точки в Swagger для створення користувача

Spring Boot інтегрує Swagger через залежності Springdoc OpenAPI. Встановлення цієї бібліотеки дозволяє автоматично генерувати документацію, додаючи у код анотації для опису методів або структур даних. Керування

версіями API та інші функції, такі як аспекти безпеки також додаються за допомогою Swagger. Цей інструмент дійсно важливий для сучасної розробки REST API, оскільки він значно спрощує спілкування між командами розробників і знижує ймовірність помилок інтеграції.

4.3 Автоматизоване тестування кінцевих точок

Автоматизоване тестування — це перевірка працездатності програмного забезпечення шляхом запуску спеціальних інструментів, сценаріїв або фреймворків, які виконують сценарії тестування без втручання людини. Таким чином забезпечується виявлення несправностей з найменшими зусиллями, перевірка коректності та збереження стабільності системи [27].

Основна концепція автоматизованого тестування полягає у створенні набору тестів, які можна запускати знову і знову в автоматизованому режимі. Вони можуть бути розроблені з опорою на специфікацію програми, бізнес-логіку або вимоги до застосунку. Тести можуть перевіряти як окремі елементи коду (юніт-тести), так і цілі модулі пов'язані між собою (інтеграційні або функціональні тести), а також роботу всього застосунку (наскрізні тести).

Це економить час і ресурси, оскільки не вимагає ручного втручання для запуску кожного тесту. Його можна інтегрувати в системи CI/CD, щоб тести запускалися з кожною збіркою, випуском або виправленням. У результаті людські помилки також будуть зменшені, оскільки сама автоматизація уникає людського втручання у виконання більшу частину часу.

Автоматизоване тестування є життєво важливим для розробки проектів. Завдяки їм можна швидко перевірити чи нові функції сумісні з існуючими. Це також має тенденцію до зменшення ймовірності виникнення нових помилок у вже перевірених і стабільних частинах системи. Таким чином, автоматизоване тестування є однією з найбільш невід’ємних частин розробки сучасного програмного забезпечення, що робить його ефективним, точним і швидким.

Інфраструктура Spring Boot є досить надійною, тому вона дозволяє легко тестувати кожен рівень застосунку. Фреймворк містить інструменти та методології для тестування окремих компонентів та їх взаємодії для забезпечення загальної функціональності системи. Завдяки анотаціям і попередньо налаштованим механізмам Spring Boot дозволяє легко тестувати навіть складні програми.

Модульні тести у Spring Boot означають тестування окремих класів, наприклад служб або контролерів. Це робиться за допомогою бібліотек JUnit або TestNG, а також інструментів імітації залежностей, таких як Mockito. За допомогою цих інструментів можна запускати тести на рівні компоненту з використанням макетних об’єктів, щоб перевірити логіку тестованого компонента та уникнути застосування тесту до бази даних чи іншої системи.

Інтеграційні тести в Spring Boot перевіряють, як компоненти взаємодіють один з одним. Під час виконання тесту платформа Spring забезпечує контекст програми, який можна використовувати для перевірки поведінки будь-якого компонента на основі реального сценарію. Оскільки анотація @SpringBootTest створює повний контекст програми, звідси ми можемо запускати тести, що обробляють взаємодії з базою даних, черги повідомлень або інші зовнішні служби. Вужчі тести використовують анотації @WebMvcTest, @DataJpaTest або @MockBean, коли потрібно виділити певний рівень програми.

Функціональне тестування або наскрізні тести тестують програму в цілому. Spring Boot використовується для емуляції HTTP-запитів до контролерів, одночасно отримуючи перевірку відповідей. Це стосується правильності обробки запитів, маршрутів і результатів повернення.

Щоб перевірити бази даних Spring Boot містить засоби для налаштування бази даних у пам’яті. Тестова база даних @DataJpaTest у пам’яті надасть репозиторій або компоненти JPA для перевірки для запитів до бази даних.

Тестування контролерів у Spring Boot — це важливий процес для перевірки роботи логіки обробки HTTP-запитів. Таке тестування перевіряє чи працює контролер належним чином під час обробки вхідних даних і взаємодії із залежностями, щоб отримати правильну відповідь. Зазвичай при тестуванні контролерів використовується модульне тестування у якому контролер відокремлений від інших компонентів для більшого контролю над процесом тестування.

Одним із ключових інструментів у тестуванні Spring Boot є MockMvc, який дозволяє симулювати HTTP-запити до контролера без потреби у запуску справжнього веб-серверу. Особливо корисний у використані під час виконання модульного тестування коли потрібно перевірити як контролер поводиться з різними вхідними даними.

Під час тестування основний акцент робиться на перевірці статусів відповідей, структури та вмісту даних, які повертає контролер, а також його поведінки щодо неправильних запитів.

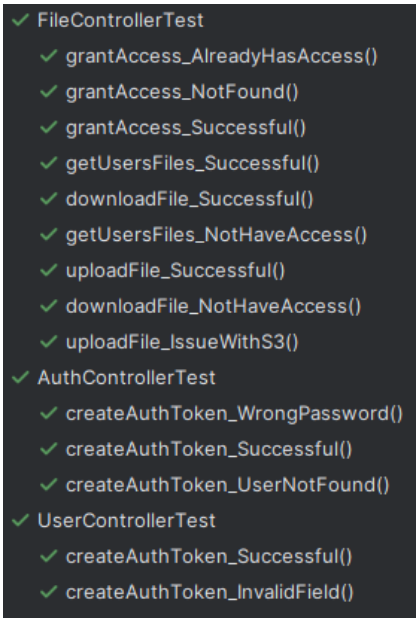

Рисунок 4.3 — Результат запуску тестування контролерів

Результати тестування зображено на рисунку 4.3. Як видно з рисунку

було запущено тестування контролерів FileController, AuthController та UserController у різних випадках для повноцінного тестування контролерів. Це дозволило перевірити правильність формату даних, що повертає застосунок користувачу. Також було протестовано реакцію застосунку на виникнення помилок при роботі та коректність повідомлень про помилку та статусів які повертаються користувачу при виникненні цих помилок.

Тестування сервісів у Spring Boot зосереджено на тестуванні бізнес-логіки, що реалізована в класах сервісах. Таким чином перевіряється чи логіка правильно працює, незалежно від інших компонентів застосунку. Зазвичай сервіси тестуються модуль за модулем у відокремлений спосіб зовнішніх залежностей. Для цього використовуються об’єкти mock — імітації залежностей, які дозволяють перевірити лише логіку сервісу. Spring Boot надає для цього великий вибір інструментів, включаючи такі речі, як бібліотеки Mockito та Spring Test.

Тестування має на меті перевірити, чи працює бізнес-логіка правильно чи ні, разом із обробкою помилок, правильністю очікуваних результатів і як вона взаємодіє з API. Під час тестування слід використовувати як успішні, так і помилкові варіанти використання, перевіряючи, чи служба належним чином реагує на будь-які умови, навіть на помилки чи недійсні записи.

Результат тестування сервісів застосунку зображено на рисунку 4.4. Проведене тестування усіх сервісів застосунку з різними вхідними даними для функцій. Тестування продемонструвало адекватну роботу реалізованого вдосконаленого методу аутентифікації для доступу до даних у хмарному сховищі. Було протестовано роботу методу при різних обставинах та як застосунок реагує на помилки під час роботи. Крім методу також тестувалась бізнес логіка застосунку.

Тестування репозиторію Spring Boot зосереджено навколо перевірки належної взаємодії з базою даних. Використовуючи JPA або інші технології доступу до даних, репозиторії є нічим іншим, як інтерфейсами, які

взаємодіють з операціями бази даних і вимагають модульного тестування для правильного функціонування операцій збереження, оновлення, видалення, читання тощо.

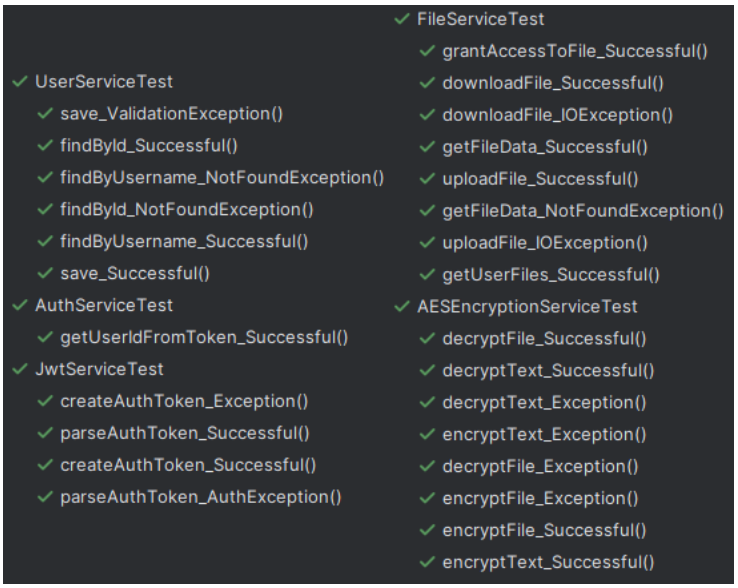

Рисунок 4.4 — Результат запуску тестування сервісів

Основними аспектами тестування є перевірка правильності зберігання даних, коректність операції спеціального запиту та виконання стандартних методів JPA, таких як findById або deleteById. Також може бути корисно перевірити, що станеться, якщо дії будуть здійснені на основі неправильних або неіснуючих даних.

Під час написання тестів для репозиторіїв можна використовувати фіктивні дані або база даних може бути попередньо заповнена тестовими записами, щоб переконатися, що всі запити виконуються правильно. Тестування репозиторіїв має вирішальне значення для забезпечення стабільності програми, оскільки репозиторії є кінцевими точками для зберігання та отримання даних. Правильна конфігурація тестів для них запобігає виникненню критичних помилок, які можуть зробити всю систему непридатною для використання.

На рисунку 4.5 зображено результат тестування репозиторіїв з різними варіантами використання.

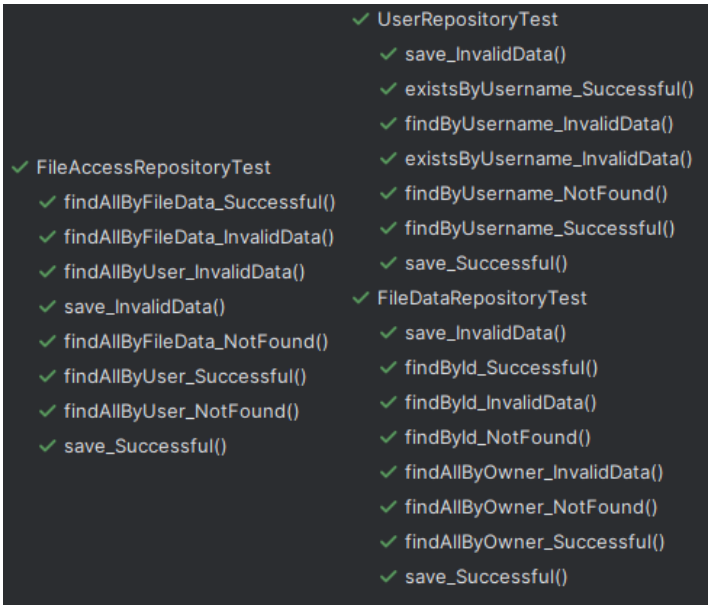

Рисунок 4.5 — Результат запуску тестування репозиторіїв

4.4 Тестування авторизації та аутентифікації

Для тестування авторизації користувача в розробленому програному застосунку відправимо запит на авторизацію до застосунку через програму Postman. Як видно зі знімку екрану, зображеному на рисунку 4.6, запит повертає маркер доступу у зашифрованому вигляді з закритим алгоритмом шифрування на відмінну від звичайного JWT токену.

Рисунок 4.5 — Вигляд маркеру доступу як результат запиту на авторизацію користувача

Цей маркер використовується користувачем для отримання доступу до інших кінцевих точок.

При збереженні файлу, що надсилається користувачем у хмарне сховище, останній шифрується. В консолі S3 переглянемо наявність завантаженого файлу (рисунок 4.6).

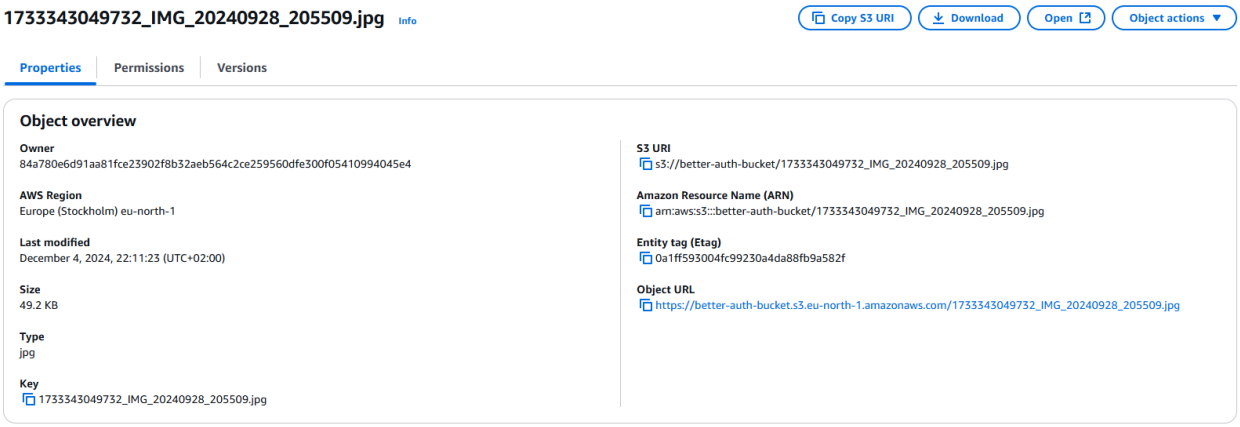

Рисунок 4.6 — Знімок екрану інформаційної панелі сервісу AWS S3

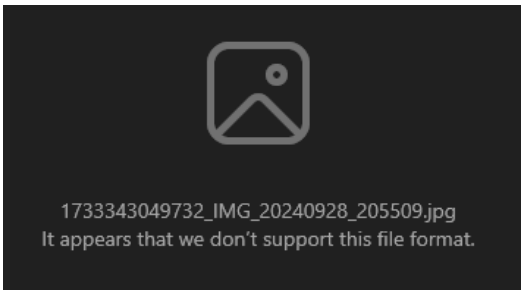

Рисунок 4.7 — Вигляд вікна з помилкою

З рисунку 4.6 видно в ідентифікаторі “Key” наявність файлу 1733343049732_IMG_20240928_205509.jpg, а при спробі відкрити цей файл отримується помилка «File appears that we don't support this file format» (рисунок 4.7), що означає неможливість розкодувати зашифрований файл без ключа, наявного тільки у користувача.

5 ЕКОНОМІЧНА ЧАСТИНА

5.1 Комерційний та технологічний аудит науково-технічної розробки

Метою даного розділу є проведення технологічного аудиту, в даному випадку нового хмарного сховища з покращеною системою аутентифікації. Метою є підвищення безпеки збережених даних які зберігаються у сховищі. Особливістю розробки є вдосконаленні методу аутентифікації за допомогою додаткового шифрування даних для аутентифікації.

Аналогами розробки можуть бути: Google One – 1000 грн/місяць, Microsoft OneDrive – 2600 грн/рік, Dropbox – 10-24 доларів/місяць.

Для проведення комерційного та технологічного аудиту залучають не менше 3-х незалежних експертів. Оцінювання науково-технічного рівня розробки та її комерційного потенціалу рекомендується здійснювати із застосуванням п’ятибальної системи оцінювання за 12-ма критеріями, у відповідності із табл. 5.1.

Таблиця 5.1 — Рекомендовані критерії оцінювання комерційного потенціалу розробки та їх можлива бальна оцінка

Бали (за 5-ти бальною шкалою)					
Критерій	0	1	2	3	4
Технічна здійсненність концепції					
1	Достовірність концепції не підтверджена	Концепція підтверджена експертними висновками	Концепція підтверджена розрахунками	Концепція перевірена на практиці	Перевірено роботоздатність продукту в реальних умовах
2	Багато аналогів на малому ринку	Мало аналогів на малому ринку	Кілька аналогів на великому ринку	Один аналог на великому ринку	Продукт не має аналогів на великому ринку
3	Ціна продукту значно вища за ціни аналогів	Ціна продукту дещо вища за ціни аналогів	Ціна продукту приблизно дорівнює цінам аналогів	Ціна продукту дещо нижче за ціни аналогів	Ціна продукту значно нижче за ціни аналогів

Продовження табл. 5.1

Ринкові переваги					
4	Технічні та споживчі властивості продукту значно гірші, ніж в аналогів	Технічні та споживчі властивості продукту трохи гірші, ніж в аналогів	Технічні та споживчі властивості продукту на рівні аналогів	Технічні та споживчі властивості продукту трохи кращі, ніж в аналогів	Технічні та споживчі властивості продукту значно кращі, ніж в аналогів
5	Експлуатаційні витрати значно вищі, ніж в аналогів	Експлуатаційні витрати дещо вищі, ніж в аналогів	Експлуатаційні витрати на рівні експлуатаційних витрат аналогів	Експлуатаційні витрати трохи нижчі, ніж в аналогів	Експлуатаційні витрати значно нижчі, ніж в аналогів
Ринкові перспективи					
6	Ринок малий і не має позитивної динаміки	Ринок малий, але має позитивну динаміку	Середній ринок з позитивною динамікою	Великий стабільний ринок	Великий ринок з позитивною динамікою
7	Активна конкуренція великих компаній на ринку	Активна конкуренція	Помірна конкуренція	Незначна конкуренція	Конкурентів немає
Практик на здійсненність					
8	Відсутні фахівці як з технічної, так і з комерційної реалізації ідеї	Необхідно наймати фахівців або витрачати значні кошти та час на навчання наявних фахівців	Необхідне незначне навчання фахівців та збільшення їх штату	Необхідне незначне навчання фахівців	Є фахівці з питань як з технічної, так і з комерційної реалізації ідеї
9	Потрібні значні фінансові ресурси, які відсутні. Джерела фінансування ідеї відсутні	Потрібні незначні фінансові ресурси. Джерела фінансування відсутні	Потрібні значні фінансові ресурси. Джерела фінансування є	Потрібні незначні фінансові ресурси. Джерела фінансування є	Не потребує додаткового фінансування
10	Необхідна розробка нових матеріалів	Потрібні матеріали, що використовуються у військово-промисловому комплексі	Потрібні дорогі матеріали	Потрібні досяжні та дешеві матеріали	Всі матеріали для реалізації ідеї відомі та давно використовуються у виробництві
11	Термін реалізації ідеї більший за 10 років	Термін реалізації ідеї більший за 5 років. Термін окупності інвестицій більше 10-ти	Термін реалізації ідеї від 3-х до 5-ти років. Термін окупності	Термін реалізації ідеї менше 3-х років. Термін окупності інвестицій	Термін реалізації ідеї менше 3-х років. Термін окупності інвестицій менше 3-х

Продовження табл. 5.1

12	Необхідна розробка регламентних документів та отримання великої кількості дозвільних документів на виробництво та реалізацію продукту	Необхідно отримання великої кількості дозвільних документів на виробництво та реалізацію продукту, що вимагає значних коштів та часу	Процедура отримання дозвільних документів для виробництва та реалізації продукту вимагає незначних коштів та часу	Необхідно тільки повідомлення відповідним органам про виробництво та реалізацію продукту	Відсутні будь-які регламентні обмеження на виробництво та реалізацію продукту

Усі дані по кожному параметру занесено в таблиці 5.2

Таблиця 5.2 — Результати оцінювання комерційного потенціалу розробки

Критерії оцінювання	ПІБ експертів		
	Експерт 1	Експерт 2	Експерт 3
	Бали		
Технічна здійсненність концепції	3	4	4
Наявність аналогів на ринку	3	4	3
Цінова політика	4	4	4
Технічні та споживчі властивості виробу	4	4	3
Експлуатаційні витрати	4	3	4
Ринок збуту	4	4	3
Конкурентоспроможність	3	3	4
Фахівці з технічної і комерційної реалізації	4	4	3
Фінансування	4	3	4
Матеріально-технічна база	3	3	3
Термін реалізації ідеї	4	4	4
Супровідна документація	4	4	4
Сума	44	44	43
Середньоарифметична сума балів	(44+44+43) / 3 = 43,67		

За даними таблиці 5.2 можна зробити висновок щодо рівня комерційного потенціалу даної розробки. Для цього доцільно скористатись рекомендаціями, наведеними в таблиці 5.3.

Як видно з таблиці, рівень комерційного потенціалу розроблюваного нового програмного продукту є високим, що досягається за рахунок

вдосконаленні методу аутентифікації за допомогою додаткового шифрування даних для аутентифікації.

Таблиця 5.3 — Рівні комерційного потенціалу розробки

Середньоарифметична сума балів, розрахована на основі висновків експертів	Рівень комерційного потенціалу розробки
0-10	Низький
11-20	Нижче середнього
21-30	Середній
31-40	Вище середнього
41-48	Високий

Як видно з таблиці, рівень комерційного потенціалу розроблюваного нового програмного продукту є високим, що досягається за рахунок вдосконаленні методу аутентифікації за допомогою додаткового шифрування даних для аутентифікації.

5.2 Прогнозування витрат на виконання науково-дослідної (дослідно-конструкторської) роботи

5.2.1 Основна заробітна плата розробників, яка розраховується за формулою:

$$З_о = \frac{M}{T_р} \cdot t \,, \tag{5.1}$$

де М — місячний посадовий оклад конкретного розробника (дослідника), грн.;

$T_р$ — число робочих днів за місяць, 23 днів;

t — число днів роботи розробника (дослідника).

Результати розрахунків зведемо до таблиці 5.4.

Так як в даному випадку розробляється програмний продукт, то розробник виступає одночасно і основним робітником, і тестувальником розроблюваного програмного продукту.

Таблиця 5.4 — Основна заробітна плата розробників

Найменування посади	Місячний посадовий оклад, грн.	Оплата за робочий день, грн.	Число днів роботи	Витрати на заробітну плату, грн.
Керівник проекту	30000	1304,35	30	39130,435
Програміст	28500	1239,13	30	37173,913
Всього				76304,35

5.2.2 Додаткова заробітна плата розробників, які брали участь в розробці застосунку.

Додаткову заробітну плату прийнято розраховувати як 14 % від основної заробітної плати розробників та робітників:

$$З_д = З_о \cdot 14\ \% / 100\ \% \quad (5.2)$$

$$З_д = (76304,35 \cdot 14\ \% / 100\ \%) = 10682,61 \ \text{(грн.)}$$

5.2.3 Нарахування на заробітну плату розробників.

Згідно діючого законодавства нарахування на заробітну плату складають 22 % від суми основної та додаткової заробітної плати.

$$Н_з = (З_о + З_д) \cdot 22\ \% / 100\% \quad (5.3)$$

$$Н_з = (76304,35 + 10682,61) \cdot 22\ \% / 100\ \% = 19137,13 \ \text{(грн.)}$$

5.2.4. Оскільки для розроблювального пристрою не потрібно витрачати матеріали та комплектуючи, то витрати на матеріали і комплектуючі дорівнюють нулю.

5.2.5 Амортизація обладнання, яке використовувалось для проведення розробки.

Амортизація обладнання, що використовувалось для розробки в

спрощеному вигляді розраховується за формулою:

$$A = \frac{Ц}{Тв} \cdot \frac{t_{вик}}{12} \text{ [грн.].} \quad (5.4)$$

де Ц — балансова вартість обладнання, грн.;

Т — термін корисного використання обладнання згідно податкового законодавства, років;

$t_{вик}$ — термін використання під час розробки, місяців.

Розрахуємо, для прикладу, амортизаційні витрати на комп'ютер балансова вартість якого становить 20000 грн., термін його корисного використання згідно податкового законодавства – 2 роки, а термін його фактичного використання – 1,30 міс.

$$А_{обл} = \frac{20000}{2} \times \frac{1,3}{12} = 1086,957 грн.$$

Аналогічно визначаємо амортизаційні витрати на інше обладнання та приміщення. Розрахунки заносимо до таблиці 5.5. Так як вартість ліцензійної ОС та спеціалізованих ліцензійних нематеріальних ресурсів менше 20000 грн, то даний нематеріальний актив не амортизується, а його вартість включається у вартість розробки повністю, а також вартість використання IntelliJ IDEA Ultimate – 600 доларів/рік : 600*40*1,3/12=2600 грн., $B_{нем.ак.}$ = 13600 грн.

5.2.6 Тарифи на електроенергію для непобутових споживачів (промислових підприємств) відрізняються від тарифів на електроенергію для населення. При цьому тарифи на розподіл електроенергії у різних постачальників (енергорозподільних компаній), будуть різними.

Таблиця 5.5 — Амортизаційні відрахування на матеріальні та нематеріальні ресурси для розробників

Найменування обладнання	Балансова вартість, грн.	Строк корисного використання, років	Термін використання обладнання, місяців	Амортиза-ційні відрахуван-ня, грн.
Комп'ютер та комп'ютерна периферія	20000	2	1,30	1086,957
Офісне обладнання (меблі)	25000	4	1,30	679,348
Приміщення	1500000	20	1,30	8152,174
Всього				9918,48

Крім того, розмір тарифу залежить від класу напруги (1-й або 2-й клас). Тарифи на розподіл електроенергії для всіх енергорозподільних компаній встановлює Національна комісія з регулювання енергетики і комунальних послуг (НКРЕКП). Витрати на силову електроенергію розраховуються за формулою:

$$В_е = В \cdot П \cdot Ф \cdot К_п, \quad (5.5)$$

де В — вартість 1 кВт-години електроенергії для 1 класу підприємства з ПДВ в 2024 році для Вінницької області за даними Енера-Вінниця, В = 10,42 грн./кВт;

П — встановлена потужність обладнання, кВт. П = 0,35 кВт;

Ф — фактична кількість годин роботи обладнання, годин.

$К_п$ — коефіцієнт використання потужності, $К_п$ = 0,9.

$$В_е = 0,9 \cdot 0,35 \cdot 8 \cdot 30 \cdot 10,42 = 787,752 \text{ (грн.)}$$

5.2.7 Інші витрати та загальновиробничі витрати.

До статті «Інші витрати» належать витрати, які не знайшли відображення у зазначених статтях витрат і можуть бути віднесені безпосередньо на собівартість досліджень за прямими ознаками. Витрати за

статтею «Інші витрати» розраховуються як 50…100% від суми основної заробітної плати дослідників:

$$I_в = (З_о + З_р) \cdot \frac{Н_{ів}}{100\%}, \quad (4.6)$$

де $Н_{ів}$ — норма нарахування за статтею «Інші витрати».

$$I_в = 76304,35 * 60\% / 100\% = 45782,61 \text{ (грн.)}$$

До статті «Накладні (загальновиробничі) витрати» належать: витрати, пов'язані з управлінням організацією; витрати на винахідництво та раціоналізацію; витрати на підготовку (перепідготовку) та навчання кадрів; витрати, пов'язані з набором робочої сили; витрати на оплату послуг банків; витрати, пов'язані з освоєнням виробництва продукції; витрати на науково-технічну інформацію та рекламу та ін. Витрати за статтею «Накладні (загальновиробничі) витрати» розраховуються як 100…150% від суми основної заробітної плати дослідників:

$$Н_{нзв} = (З_о + З_р) \cdot \frac{Н_{нзв}}{100\%}, \quad (4.7)$$

де $Н_{нзв}$ — норма нарахування за статтею «Накладні (загальновиробничі) витрати».

$$Н_{нзв} = 76304,35 * 130\ \% / 100\ \% = 99196 \text{ (грн.)}$$

5.2.9 Витрати на проведення науково-дослідної роботи.

Сума всіх попередніх статей витрат дає загальні витрати на проведення науково-дослідної роботи:

Взаг = 76304,35+10682,61+19137,13+9918,48+13600+787,75+45782,61+

+99196 = 275408,58 грн.

5.2.11 Розрахунок загальних витрат на науково-дослідну (науково-технічну) роботу та оформлення її результатів.

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

$$ЗВ = \frac{В_{заг}}{\eta} \text{ (грн)}, \quad (4.8)$$

де η — коефіцієнт, який характеризує етап (стадію) виконання науково-дослідної роботи.

Так, якщо науково-технічна розробка знаходиться на стадії: науково-дослідних робіт, то η=0,1; технічного проектування, то η=0,2; розробки конструкторської документації, то η=0,3; розробки технологій, то η=0,4; розробки дослідного зразка, то η=0,5; розробки промислового зразка, то η=0,7; впровадження, то η=0,9. Оберемо η = 0,5, так як розробка, на даний момент, знаходиться на стадії дослідного зразка:

$ЗВ$ = 275408,58 / 0,5 = 550817 грн.

5.3 Розрахунок економічної ефективності науково-технічної розробки за її можливої комерціалізації потенційним інвестором

В ринкових умовах узагальнювальним позитивним результатом, що його може отримати потенційний інвестор від можливого впровадження результатів тієї чи іншої науково-технічної розробки, є збільшення у потенційного інвестора величини чистого прибутку. Саме зростання чистого прибутку забезпечить потенційному інвестору надходження додаткових коштів, дозволить покращити фінансові результати його діяльності,

підвищить конкурентоспроможність та може позитивно вплинути на ухвалення рішення щодо комерціалізації цієї розробки.

Для того, щоб розрахувати можливе зростання чистого прибутку у потенційного інвестора від можливого впровадження науково-технічної розробки необхідно:

— вказати, з якого часу можуть бути впроваджені результати науково-технічної розробки;

— зазначити, протягом скількох років після впровадження цієї науково-технічної розробки очікуються основні позитивні результати для потенційного інвестора (наприклад, протягом 3-х років після її впровадження);

— кількісно оцінити величину існуючого та майбутнього попиту на цю або аналогічні чи подібні науково-технічні розробки та назвати основних суб’єктів (зацікавлених осіб) цього попиту;

— визначити ціну реалізації на ринку науково-технічних розробок з аналогічними чи подібними функціями.

При розрахунку економічної ефективності потрібно обов’язково враховувати зміну вартості грошей у часі, оскільки від вкладення інвестицій до отримання прибутку минає чимало часу. При оцінюванні ефективності інноваційних проектів передбачається розрахунок таких важливих показників:

— абсолютного економічного ефекту (чистого дисконтованого доходу);

— внутрішньої економічної дохідності (внутрішньої норми дохідності);

— терміну окупності (дисконтованого терміну окупності).

Аналізуючи напрямки проведення науково-технічних розробок, розрахунок економічної ефективності науково-технічної розробки за її можливої комерціалізації потенційним інвестором можна об’єднати, враховуючи визначені ситуації з відповідними умовами.

5.3.1 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:

$$\Delta П_i = (\pm \Delta Ц_0 \cdot N + Ц_0 \cdot \Delta N)_i \cdot \lambda \cdot \rho \cdot (1 - \frac{\vartheta}{100}), \quad (5.9)$$

де $\pm\Delta Ц_о$ — зміна вартості програмного продукту (зростання чи зниження) від впровадження результатів науково-технічної розробки в аналізовані періоди часу;

N — кількість споживачів які використовували аналогічний продукт у році до впровадження результатів нової науково-технічної розробки;

$Ц_о$ — основний оціночний показник, який визначає діяльність підприємства у даному році після впровадження результатів наукової розробки, $Ц_о = Ц_б \pm \Delta Ц_о$;

Цб — вартість програмного продукту у році до впровадження результатів розробки;

ΔN — збільшення кількості споживачів продукту, в аналізовані періоди часу, від покращення його певних характеристик;

λ — коефіцієнт, який враховує сплату податку на додану вартість. Ставка податку на додану вартість дорівнює 20%, а коефіцієнт λ = 0,8333.

р — коефіцієнт, який враховує рентабельність продукту;

ϑ — ставка податку на прибуток, у 2024 році ϑ =18%.

Припустимо, що при прогнозованій ціні 200 грн. за одиницю виробу, термін збільшення прибутку складе 3 роки. Після завершення розробки і її вдосконалення, можна буде підняти її ціну на *50* грн. Кількість одиниць реалізованої продукції також збільшиться: протягом першого року – на 30000

шт., протягом другого року – на 35000 шт., протягом третього року на 40000 шт. До моменту впровадження результатів наукової розробки реалізації продукту не було:

$$\Delta П_1 = (0*50 + (200 + 50)*30000)* 0{,}8333* 0{,}35) * (1 - 0{,}18) = 1434999{,}943 \text{ грн.}$$

$$\Delta П_2 = (0*50 + (200 + 50)*(30000+35000)* 0{,}8333* 0{,}35) * (1 - 0{,}18) = 3886458{,}178 \text{ грн.}$$

$$\Delta П_3 = (0*50 + (200 + 50)*(30000+35000+40000)* 0{,}8333* 0{,}35) * (1 - 0{,}18) = 6278124{,}749 \text{ грн.}$$

Отже, комерційний ефект від реалізації результатів розробки за три роки складе 11599582,87 грн.

5.3.2 Розрахунок ефективності вкладених інвестицій та періоду їх окупності.

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

$$ПП = \sum_{1}^{T} \frac{\Delta П_i}{(1+\tau)^t}, \qquad (5.10)$$

де $\Delta П_i$ — збільшення чистого прибутку у кожному із років, протягом яких виявляються результати виконаної та впровадженої науково-дослідної (науково-технічної) роботи, грн;

Т — період часу, протягом якою виявляються результати впровадженої науково-дослідної (науково-технічної) роботи, роки;

τ — ставка дисконтування, за яку можна взяти щорічний прогнозований рівень інфляції в країні, $\tau = 0{,}05…0{,}15;$

t — період часу (в роках).

Збільшення прибутку ми отримаємо, починаючи з першого року:

$$ПП = (1434999,943/(1+0,1)^1)+(3886458,178/(1+0,1)^2)+(6278124,749/$$
$$/(1+0,1)^3) = 1304545,40 + 3211948,907 + 4716848,046 = 9233342,355 \text{ грн.}$$

Далі розраховують величину початкових інвестицій *PV*, які потенційний інвестор має вкласти для впровадження і комерціалізації науково-технічної розробки. Для цього можна використати формулу:

$$PV = k_{інв} * ЗВ, \quad (5.11)$$

де $k_{інв}$ — коефіцієнт, що враховує витрати інвестора на впровадження науково-технічної розробки та її комерціалізацію. Це можуть бути витрати на підготовку приміщень, розробку технологій, навчання персоналу, маркетингові заходи тощо; зазвичай $k_{інв}=2...5$, але може бути і більшим;

ЗВ — загальні витрати на проведення науково-технічної розробки та оформлення її результатів, грн.

$$PV = 2 * 550817 = 1101634,31 \text{ грн.}$$

Тоді абсолютний економічний ефект $E_{абс}$ або чистий приведений дохід (NPV, Net Present Value) для потенційного інвестора від можливого впровадження та комерціалізації науково-технічної розробки становитиме:

$$E_{абс} = ПП – PV, \quad (5.12)$$

$$E_{абс} = 9233342,355 – 1101634,31 = 8131708,04 \text{ грн.}$$

Оскільки $E_{абс} > 0$ то вкладання коштів на виконання та впровадження результатів даної науково-дослідної (науково-технічної) роботи може бути доцільним.

Для остаточного прийняття рішення з цього питання необхідно розрахувати внутрішню економічну дохідність або показник внутрішньої но-рми дохідності (IRR, Internal Rate of Return) вкладених інвестицій та порівняти її з так званою бар'єрною ставкою дисконтування, яка визначає ту мінімальну внутрішню економічну дохідність, нижче якої інвестиції в будь-яку науково-технічну розробку вкладати буде економічно недоцільно.

Розрахуємо відносну (щорічну) ефективність вкладених в наукову розробку інвестицій $E_в$. Для цього використаємо формулу:

$$E_в = \sqrt[T_ж]{1 + \frac{E_{абс}}{PV}} - 1, \tag{4.13}$$

де $T_ж$ — життєвий цикл наукової розробки, роки;

$$E_в = \sqrt[3]{(1 + 8131708{,}04/1101634{,}31} - 1 = 1{,}031.$$

Визначимо мінімальну ставку дисконтування, яка у загальному вигляді визначається за формулою:

$$\tau = d + f, \tag{5.14}$$

де d — середньозважена ставка за депозитними операціями в комерційних банках; в 2024 році в Україні d = (0,09…0,14);

F — показник, що характеризує ризикованість вкладень; зазвичай, величина f = (0,05...0,5).

$$\tau_{\min} = 0{,}14 + 0{,}05 = 0{,}19.$$

Так як $E_в > \tau_{min}$, то інвестор може бути зацікавлений у фінансуванні даної наукової розробки.

Розрахуємо термін окупності вкладених у реалізацію наукового проекту інвестицій за формулою:

$$T_{ок} = \frac{1}{E_в}, \quad (5.15)$$

де $T_{ок}$ = 1 / 1,031 = 0,97 р.

Оскільки $T_{ок}$ < 3-х років, а саме термін окупності рівний 0,97 роки, то фінансування даної наукової розробки є доцільним.

Висновки до розділу: економічна частина даної роботи містить розрахунок витрат на розробку нового програмного продукту, сума яких складає 550817 гривень. Було спрогнозовано орієнтовану величину витрат по кожній з статей витрат. Також розраховано чистий прибуток, який може отримати виробник від реалізації нового технічного рішення, розраховано період окупності витрат для інвестора та економічний ефект при використанні даної розробки. В результаті аналізу розрахунків можна зробити висновок, що розроблений програмний продукт за ціною дешевший за аналог і є високо конкурентоспроможним. Період окупності складе близько 0,97 роки.

ВИСНОВКИ

Магістерська кваліфікаційна роботи присвячена розробці вдосконаленого методу аутентифікації користувачів на основі сертифікату для доступу до даних в хмарному середовищі.

В першому розділі було проведено аналіз сучасних методів аутентифікації користувачів, який показав переваги та недоліки цих методів. Також проведено порівняльний аналіз існуючих систем зберігання даних в хмарному середовищі, що дало можливість розробити концепцію застосунку з використанням запропонованого методу аутентифікації.

В другому розділі вдосконалено методу аутентифікації користувачів для безпечного доступу до даних у хмарному середовищі шляхом додаткового шифрування маркеру доступу, що дало можливість підвищити безпеку всієї системи та засекретити конфіденційні дані користувача, що використовуються для його ідентифікації. Розроблено архітектуру програмного застосунку з використанням запропонованого методу аутентифікації та додаткових методів захисту даних, а також обґрунтовано вибір стеку технологій для його реалізації та технологій для додаткового шифрування файлів, що завантажуються в хмарне середовище.

В третьому розділі розроблено програмний застосунок з вдосконаленим методом аутентифікації користувачів для безпечного доступу до даних у хмарному середовищі шляхом додаткового шифрування маркеру доступу, в якому реалізовано запропонований вдосконалений метод аутентифікції, що дало змогу перевірити його практичну цінність. Розроблено бізнес логіку застосунку, у якій використовується вдосконалений метод, реалізовано структуру бази даних та виконано конфігурування кінцевих точок.

В четвертому розділі протестовано розроблений застосунок, що надало змогу підтвердити адекватність вдосконаленого методу. Протестовано

застосунок на різних рівнях сервісів для перевірки працездатності методу та застосунку при різних успішних та помилкових випадках.

У п’ятому розділі було представлено економічний розрахунок доцільності розробки та можливість виведення її на ринок. Було розраховано витрати на розробку, визначено чистий прибуток. Період окупності складає менше одного року.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Про захист персональних даних [Електронний ресурс]. – 2024. – Режим доступу до ресурсу: https://zakon.rada.gov.ua/laws/show/2297-17#Text.

2. Загальний регламент про захист даних (GDPR) [Електронний ресурс] – Режим доступу до ресурсу: https://gdpr-text.com/uk/.

3. California Consumer Privacy Act [Електронний ресурс]. – 2024. – Режим доступу до ресурсу: https://oag.ca.gov/privacy/ccpa.

4. Стасюк Є. Є., Войцеховська О. В. «Вдосконалення аутентифікації шляхом додаткового шифрування маркеру доступу». Матеріали конференції «Молодь в науці: дослідження, проблеми, перспективи (МН-2025)», 2025 р. [Електронний ресурс] — https://conferences.vntu.edu.ua/index.php/mn/mn2025/paper/view/22378.

5. Що таке однофакторна автентифікація? Плюси та мінуси SFA [Електронний ресурс] // Hideez. – 2021. – Режим доступу до ресурсу: https://hideez.com/uk-ua/blogs/news/single-factor-authentication.

6. Що таке багатофакторна автентифікація (MFA)? [Електронний ресурс] // Smart IT. – 2023. – Режим доступу до ресурсу: https://cloud.smart-it.com/news-post/what-is-mfa/.

7. Майбутнє аутентифікації без паролів у кібербезпеці [Електронний ресурс] // iIT Distribution. – 2024. – Режим доступу до ресурсу: https://iitd.ua/majbutnye-autentifikacziyi-bez-paroliv-u-kiberbezpeczi/.

8. Методи аутентифікації без пароля [Електронний ресурс] // Wise It – Режим доступу до ресурсу: https://wiseit.com.ua/metody-autentyfikacziyi-bez-parolya-vid-fudo/.

9. Біометрична автентифікація – переваги та недоліки використання [Електронний ресурс] // ESET. – 2022. – Режим доступу до ресурсу: https://www.eset.com/ua/about/newsroom/blog/data-protection/mogut-li-khakery-pokhitit-vash-otpechatok-paltsa-nedostatki-biometricheskoy-

autentifikatsii/?srsltid=AfmBOooE2VwMH6VhlWAxmfrWJSKbEcuXtOFa2-lKZI0jB5HMleMokIju.

10. Google Drive [Електронний ресурс] – Режим доступу до ресурсу: https://workspace.google.com/products/drive/.

11. Dropbox [Електронний ресурс] – Режим доступу до ресурсу: https://www.dropbox.com/official-teams-page?_tk=paid_sem_goog_biz_b&_camp=21205014608&_kw=dropbox|e&_ad=697020683404||c&gad_source=1&gclid=Cj0KCQiAsOq6BhDuARIsAGQ4-zhIQiFXYNcvpGva1Qmijs0UMydzL2mgMaoCTpo6baTenanZrYKke8EaAuG9EALw_wcB.

12. Microsoft OneDrive [Електронний ресурс] – Режим доступу до ресурсу: https://www.microsoft.com/uk-ua/microsoft-365/onedrive/online-cloud-storage.

13. The Anatomy of a JSON Web Token [Електронний ресурс] // DigitalOcean – Режим доступу до ресурсу: https://www.digitalocean.com/community/tutorials/the-anatomy-of-a-json-web-token.

14. Ludin S. Learning HTTP/2: A Practical Guide for Beginners / S. Ludin, J. Garza., 2017. – 154 с.

15. Richardson L. RESTful Web APIs: Services for a Changing World / L. Richardson, M. Amundsen, S. Ruby., 2013. – 406 с.

16. Powers S. Learning Node / Shelley Powers., 2016. – 392 с.

17. Magolan G. Nest.js: A Progressive Node.js Framework / G. Magolan, D. Guijarro, J. Bell., 2019. – 317 с.

18. Lockhart J. Modern PHP: New Features and Good Practices / Josh Lockhart., 2015. – 268 с.

19. Stauffer M. Laravel: Up & Running: A Framework for Building Modern PHP Apps / Matt Stauffer., 2019. – 552 с.

20. Sierra K. Head First Java / K. Sierra, B. Bates, T. Gee., 2022.

21. Turnquist G. Learning Spring Boot 3.0: Simplify the development of production-grade applications using Java and Spring / Greg Turnquist., 2022. – 270 с.

22. Hellerman H. Digital Computer System Principles / Herbert Hellerman. – New York, 1967. – 466 с.

23. Google Cloud Storage [Електронний ресурс] – Режим доступу до ресурсу: https://cloud.google.com/storage.

24. DigitalOcean Spaces [Електронний ресурс] – Режим доступу до ресурсу: https://www.digitalocean.com/products/spaces.

25. Amazon S3 [Електронний ресурс] – Режим доступу до ресурсу: https://aws.amazon.com/s3/.

26. Mouat A. Using Docker: Developing and Deploying Software with Containers / Adrian Mouat., 2016. – 358 с.

27. Mohan G. Full Stack Testing. A Practical Guide for Delivering High Quality Software / Gayathri Mohan., 2022. – 406 с.

ДОДАТОК А

Технічне завдання

Міністерство освіти і науки України
Вінницький національний технічний університет
Факультет інформаційних технологій та комп'ютерної інженерії

ЗАТВЕРДЖУЮ
Завідувач кафедри ОТ
д.т.н., проф. О.Д. Азаров
"18" вересня 2024 р.

ТЕХНІЧНЕ ЗАВДАННЯ

на виконання магістерської кваліфікаційної роботи

«ВДОСКОНАЛЕНИЙ МЕТОД АУТЕНТИФІКАЦІЇ ДЛЯ ДОСТУПУ ДО ДАНИХ В ХМАРНОМУ СЕРЕДОВИЩІ»
08–54.МКР.019.00.000 ПЗ

Науковий керівник к.т.н., доц., каф. ОТ
_____Войцеховська О. В.
Магістрант групи 1КІ–23м
Стасюк Є. Є.

Вінниця 2024

1 Підстава для виконання магістерської кваліфікаційної роботи (МКР)

Підставою для розробки даної магістерської кваліфікаційної роботи є наказ ВНТУ №_310_ від «17» 09 2024 року та рішення засідання кафедри обчислювальної техніки (протокол №__ від «_____»__ 2024 року).

2 Мета і призначення МКР

2.1 Мета роботи — вдосконалення методу аутентифікації для забезпечення безпечного доступу до даних у хмарному середовищі шляхом додаткового шифрування маркеру доступу, що дасть можливість підвищити безпеку всієї системи та засекретити конфіденційні дані користувача, що використовуються для його ідентифікації.

2.2 Призначення розробки — підвищити рівень безпеки конфіденційних даних користувача при авторизації та аунтентифікації при отримані доступу до хмарного сховища.

3 Вихідні дані для виконання МКР

3.1 Проведення аналізу існуючих методів аутентифікації, а також огляд сучасних застосунків з доступом до хмарного сховища з метою визначення їх сильних та слабких сторін і шляхів для вдосконалення.

3.2 Розробка вдосконаленого методу аутентифікації та вибір основних технологій для його реалізації з використанням шифрування для забезпечення безпеки даних.

3.3 Програмна реалізація застосунку для доступу до даних у хмарному сховиі з вдосконаленою аутентифікацією та створення бізнес логіки до неї.

3.4 Проведення тестування застосунку, включаючи обробку реальних помилок які можуть виникнути при експлуатації.

4 Вимоги до виконання МКР

Головна вимога — вдосконалити існуючий метод аутентифікації для підняття рівня безпеки даних користувача шляхом шифрування сертифікату, що використовується при аутентифікації. Застосунок повинний забезпечувати ефективну обробку маркеру доступу попри додаткове завантаження через шифрування та не втрачати своєї ефективності при використанні.

Розробка має відповідати вимогам безпеки, стабільності роботи та забезпечувати взаємодію між собою у застосунку.

5. Етапи МКР та очікувані результати

Етапи роботи та очікувані результати приведено в таблиці А.1.

6 Матеріали, що подаються до захисту МКР

До захисту подаються: пояснювальна записка МКР, графічні і ілюстративні матеріали, протокол попереднього захисту МКР на кафедрі, відгук наукового керівника, відгук опонента, протоколи складання державних екзаменів, анотації до МКР українською та іноземною мовами, нормоконтроль про відповідність оформлення МКР діючим вимогам.

Таблиця А.1 — Етапи МКР

№ етапу	Назва етапу	Термін виконання		Очікувані результати
		початок	кінець	
1	2	3	4	5
1	Огляд і аналіз джерел інформації	23.09.2024	25.09.2024	Розділ 1
2	Теоретичні дослідження технологій для реалізації вдосконаленої аутентифікації	26.09.2024	02.10.2024	Розділ 2

3	Розробка застосунку з реалізованим вдосконаленим методом	03.10.2024	31.10.2024	Розділ 3
4	Тестування застосунку	01.11.2024	14.11.2024	Розділ 4
5	Економічна частина	15.11.2024	17.11.2024	Розділ 5

7 Порядок контролю виконання та захисту МКР

Виконання етапів графічної та розрахункової документації МКР контролюється науковим керівником згідно зі встановленими термінами. Захист МКР відбувається на засіданні Державної екзаменаційної комісії, затвердженою наказом ректора

8. Вимоги до оформлювання та порядок виконання МКР

8.1 При оформлювання МКР використовуються:

— ДСТУ 3008: 2015 «Звіти в сфері науки і техніки. Структура та правила оформлювання»;

— ДСТУ 8302: 2015 «Бібліографічні посилання. Загальні положення та правила складання»;

— міждержавний ГОСТ 2.104—2006 «Єдина система конструкторської документації. Основні написи»;

— Методичні вказівки до виконання бакалаврських кваліфікаційних робіт зі спеціальності 123 «Комп'ютерна інженерія» (освітня програма «Комп'ютерна інженерія»). Кафедра обчислювальної техніки ВНТУ 2022;

— документами на які посилаються у вище вказаних.

8.2 Порядок виконання МКР викладено в «Положення про кваліфікаційні роботи на другому (магістерському) рівні вищої освіти СУЯ ВНТУ—03.02.02— П.001.01:21».

ДОДАТОК Б

Структурна схема вдосконаленого методу аутентифікації

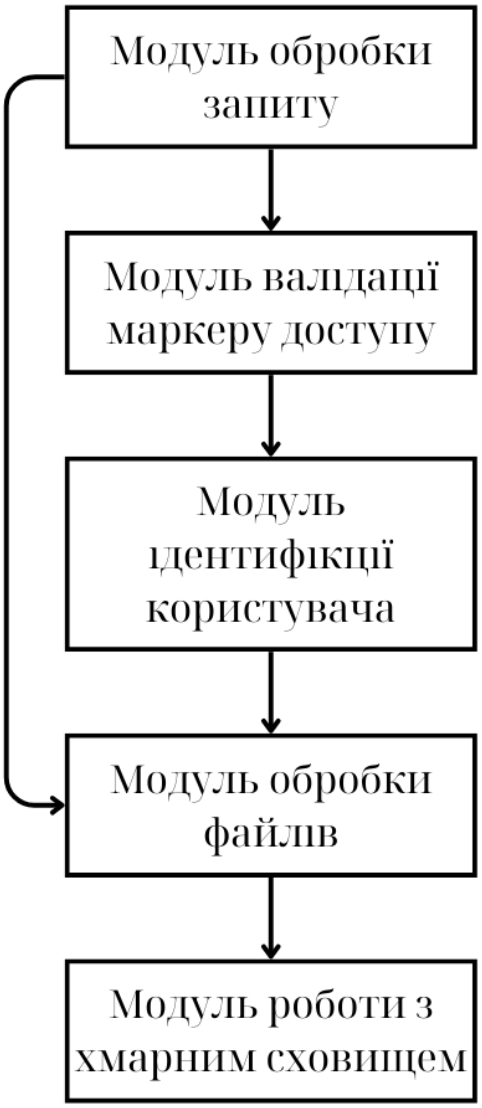

Рисунок Б.1 — Структурна схема вдосконаленого методу аутентифікації

ДОДАТОК В

Схема реалізації шифрування та розшифрування файлу

Рисунок В.1 — Схема реалізації шифрування файлу перед завантаженням у хмарне сховище

Рисунок В.2 — Схема реалізації вивантаження зашифрованого файлу з хмарного сховища

ДОДАТОК Г

Схема процесу шифрування маркеру доступу

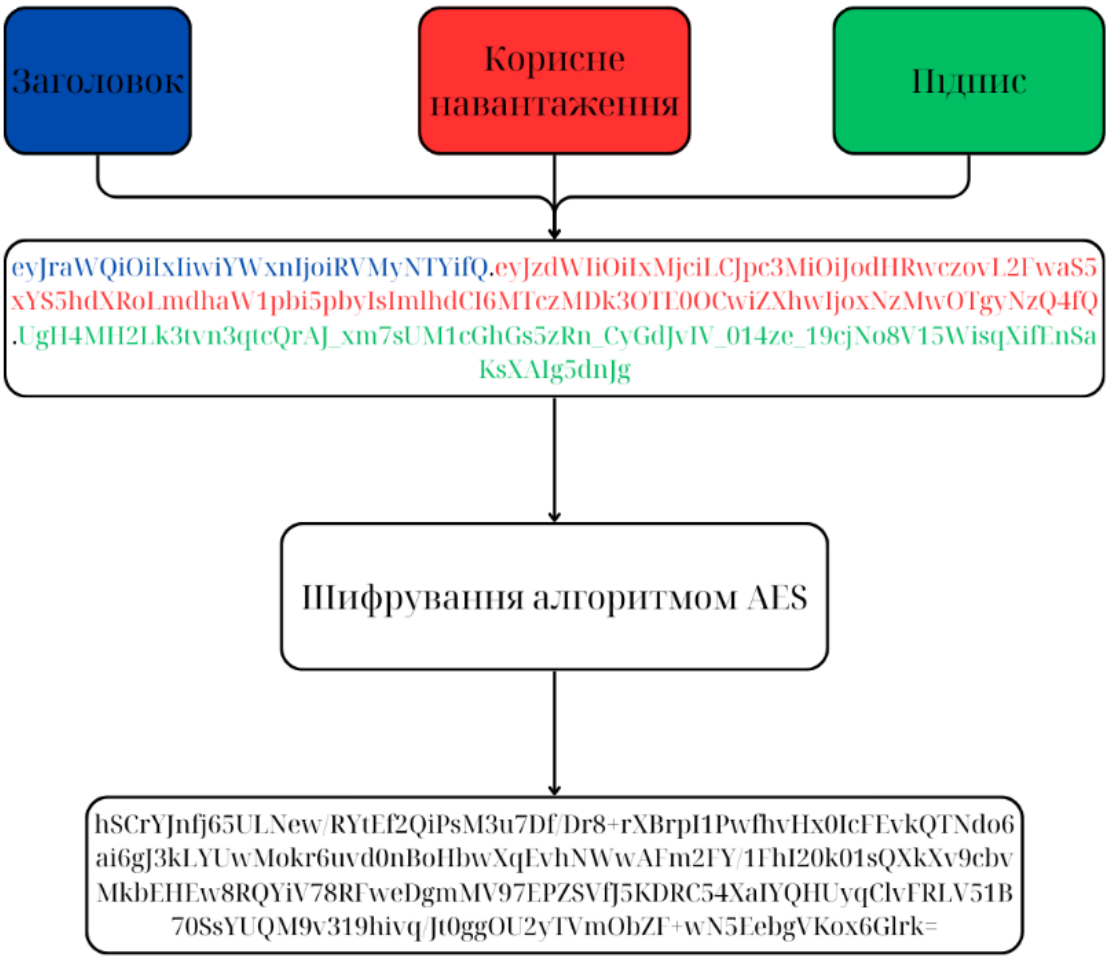

Рисунок Г.1 — Схема процесу шифрування маркеру доступу

ДОДАТОК Д

Блок-схема алгоритму вдосконаленої аутентифікації користувача

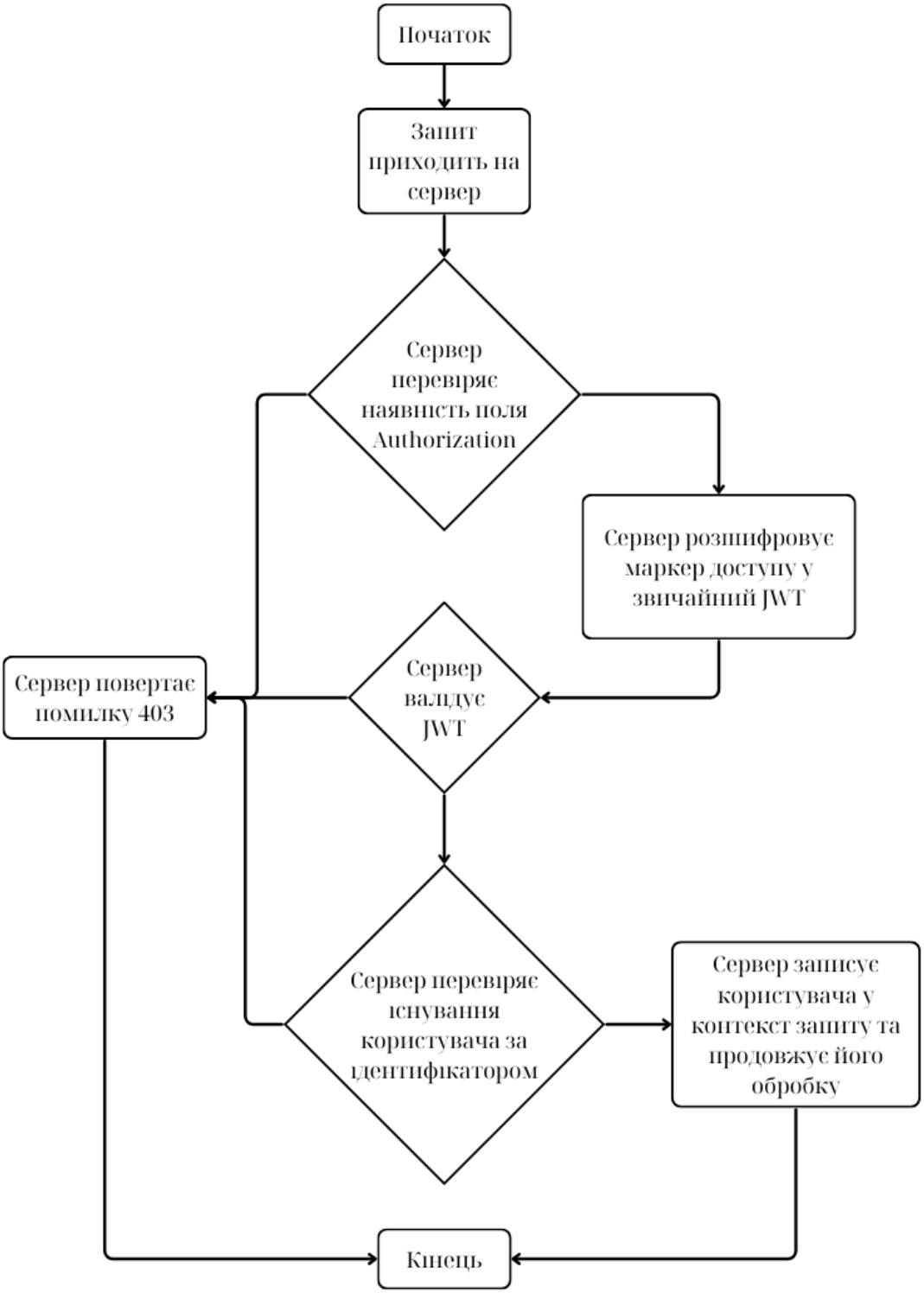

Рисунок Д.1 — Блок-схема алгоритму вдосконаленої аутентифікації користувача

ДОДАТОК Е

Лістинг сервісу JwtService

```
package com.example.betterauth.service;
import com.example.betterauth.domain.db.User;
import com.example.betterauth.exception.AuthException;
import com.example.betterauth.exception.InvalidArgumentException;
import com.example.betterauth.exception.UnknownException;
import com.example.betterauth.utils.KeyUtils;
import io.jsonwebtoken.*;
import lombok.AccessLevel;
import lombok.experimental.FieldDefaults;
import org.springframework.beans.factory.annotation.Value;
import org.springframework.stereotype.Service;
import java.security.PrivateKey;
import java.security.PublicKey;
import java.time.Duration;
import java.time.Instant;
import java.time.temporal.TemporalAmount;
import java.util.Date;
import java.util.function.Supplier;
@Service
@FieldDefaults(level = AccessLevel.PRIVATE, makeFinal = true)
public class JwtService {
    static TemporalAmount AUTH_TOKEN_DURATION = Duration.ofHours(6);
    PublicKey authPublicKey;
    PrivateKey authPrivateKey;
    AESEncryptionService aesEncryptionService;
    public JwtService(@Value("${jwt.local-key.auth.public}") String
```

```
authPublicKeyFile,
                @Value("${jwt.local-key.auth.private}") String authPrivateKeyFile,
                AESEncryptionService aesEncryptionService) {
        authPublicKey = KeyUtils.getPublicKey(authPublicKeyFile);
        authPrivateKey = KeyUtils.getPrivateKey(authPrivateKeyFile);
        this.aesEncryptionService = aesEncryptionService;
    }
    public String createAuthToken(User user) {
        String jwt = Jwts.builder()
                .setSubject(user.getId().toString())
                .setIssuedAt(new Date(Instant.now().toEpochMilli()))
                .setExpiration(new
Date(Instant.now().plus(AUTH_TOKEN_DURATION).toEpochMilli()))
                .signWith(SignatureAlgorithm.ES256, authPrivateKey)
                .compact();
        try {
            return aesEncryptionService.encryptText(jwt);
        } catch (Exception e) {
            throw new RuntimeException(e);
        }
    }
    public Claims parseAuthToken(String token) {
        try { String jwt = aesEncryptionService.decryptText(token);
            return handleParseExceptions(() ->
Jwts.parser().setSigningKey(authPublicKey).build().parseClaimsJws(jwt).getBody
());
        } catch (Exception e) {
            throw new AuthException(new Error("Cannot decode token"));}
}
```

```
    private Claims handleParseExceptions(Supplier<Claims> parser) {
        try {
            return parser.get();
        } catch (IllegalArgumentException | UnsupportedJwtException |
MalformedJwtException e) {
            throw new InvalidArgumentException("Invalid Token Argument");
        } catch (Exception e) {
            throw new UnknownException("Invalid Token");
        }
    }
}
```

ДОДАТОК Ж

Лістинг сервісу AESEncryptionService

```
package com.example.betterauth.service;
import lombok.AccessLevel;
import lombok.experimental.FieldDefaults;
import org.springframework.beans.factory.annotation.Value;
import org.springframework.stereotype.Service;
import javax.crypto.Cipher;
import javax.crypto.SecretKey;
import javax.crypto.spec.SecretKeySpec;
import java.util.Base64;
@Service
@FieldDefaults(level = AccessLevel.PRIVATE, makeFinal = true)
public class AESEncryptionService {
   static String AES_ALGORITHM = "AES";
   SecretKey secretKey;
   public AESEncryptionService(@Value("${jwt.encryption.aesKey}") String
base64Key) {
      byte[] decodedKey = Base64.getDecoder().decode(base64Key);
      this.secretKey = new SecretKeySpec(decodedKey, 0, decodedKey.length,
AES_ALGORITHM);
   }
   public String encryptText(String plainText) throws Exception {
      Cipher cipher = Cipher.getInstance(AES_ALGORITHM);
      cipher.init(Cipher.ENCRYPT_MODE, secretKey);
      byte[] encryptedBytes = cipher.doFinal(plainText.getBytes());
      return Base64.getEncoder().encodeToString(encryptedBytes);
   }
```

```
    public String decryptText(String encryptedText) throws Exception {
        Cipher cipher = Cipher.getInstance(AES_ALGORITHM);
        cipher.init(Cipher.DECRYPT_MODE, secretKey);
        byte[] decodedBytes = Base64.getDecoder().decode(encryptedText);
        byte[] decryptedBytes = cipher.doFinal(decodedBytes);
        return new String(decryptedBytes);
    }
    public byte[] encryptFile(byte[] file, String key) throws Exception {
        Cipher cipher = Cipher.getInstance(AES_ALGORITHM);
        cipher.init(Cipher.ENCRYPT_MODE, generateKey(key));
        return cipher.doFinal(file);
    }
    public byte[] decryptFile(byte[] file, String key) throws Exception {
        Cipher cipher = Cipher.getInstance(AES_ALGORITHM);
        cipher.init(Cipher.DECRYPT_MODE, generateKey(key));
        return cipher.doFinal(file);
    }
    private SecretKey generateKey(String key) {
        return new SecretKeySpec(key.getBytes(), AES_ALGORITHM);
    }
}
```

ДОДАТОК И

Лістинг сервісу FileService

```
package com.example.betterauth.service;
import com.example.betterauth.domain.db.FileAccess;
import com.example.betterauth.domain.db.FileData;
import com.example.betterauth.domain.db.User;
import com.example.betterauth.exception.AccessNotGrantedException;
import com.example.betterauth.exception.NotFoundException;
import com.example.betterauth.repository.FileAccessRepository;
import com.example.betterauth.repository.FileDataRepository;
import com.example.betterauth.utils.AuthenticationUtils;
import lombok.AccessLevel;
import lombok.experimental.FieldDefaults;
import org.springframework.beans.factory.annotation.Value;
import org.springframework.stereotype.Service;
import software.amazon.awssdk.auth.credentials.AwsBasicCredentials;
import software.amazon.awssdk.auth.credentials.StaticCredentialsProvider;
import software.amazon.awssdk.core.ResponseInputStream;
import software.amazon.awssdk.core.sync.RequestBody;
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.s3.S3Client;
import software.amazon.awssdk.services.s3.model.GetObjectRequest;
import software.amazon.awssdk.services.s3.model.GetObjectResponse;
import software.amazon.awssdk.services.s3.model.PutObjectRequest;
import java.io.ByteArrayOutputStream;
import java.io.IOException;
import java.util.Date;
import java.util.List;
```

```
import java.util.stream.Stream;
@Service
@FieldDefaults(level = AccessLevel.PRIVATE, makeFinal = true)
public class FileService {
   String bucketName;
   S3Client s3Client;
   FileDataRepository fileDataRepository;
   FileAccessRepository fileAccessRepository;
   public FileService(@Value("${aws.s3.bucketName}") String bucketName,
                      @Value("${aws.s3.accessKey}") String accessKeyId,
                      @Value("${aws.s3.secretKey}") String secretAccessKey,
                      @Value("${aws.s3.region}") String region,
                      FileDataRepository fileDataRepository, FileAccessRepository
fileAccessRepository) {
      this.bucketName = bucketName;
      this.fileDataRepository = fileDataRepository;
      this.fileAccessRepository = fileAccessRepository;
      this.s3Client = S3Client.builder()

.credentialsProvider(StaticCredentialsProvider.create(AwsBasicCredentials.create
(accessKeyId, secretAccessKey)))
            .region(Region.of(region))
            .build();
   }
   public FileData uploadFile(byte[] file, String fileName, User owner) throws
IOException {
      String keyOfSavedFile = uploadFileToS3(file, fileName);
      FileData fileData = FileData.builder()
            .fileName(fileName)
```

```
                .fileKey(keyOfSavedFile)
                .owner(owner)
                .build();
        return fileDataRepository.save(fileData);
    }
    public byte[] downloadFile(Long id) throws IOException {
        FileData fileData = fileDataRepository.findById(id)
                .orElseThrow(() -> new NotFoundException("Can't find file data with
this id"));
        requireUserHasAccessToFile(fileData);
        return downloadFileFromS3(fileData);
    }
    public List<FileData> getUserFiles(User user) {
        List<FileData> fileDataUserOwns =
fileDataRepository.findAllByOwner(user);
        List<FileData> fileDataUserHasAccess =
fileAccessRepository.findAllByUser(user).stream()
                .map(FileAccess::getFileData)
                .toList();
        return Stream.concat(fileDataUserOwns.stream(),
fileDataUserHasAccess.stream()).toList();
    }
    public FileData getFileData(Long id) {
        return fileDataRepository.findById(id)
                .orElseThrow(() -> new NotFoundException("Can't find file data with
this id"));
    }
    public FileAccess grantAccessToFile(FileData fileData, User user) {
        requireUserHasNotAccessToFile(fileData, user);
```

```
        FileAccess fileAccess = new FileAccess(null, fileData, user);
        return fileAccessRepository.save(fileAccess);
    }
    private String uploadFileToS3(byte[] file, String fileName) throws IOException
{
        String key = new Date().getTime() + "_" + fileName;
        PutObjectRequest putObjectRequest = PutObjectRequest.builder()
                .bucket(bucketName)
                .key(key)
                .build();
        s3Client.putObject(putObjectRequest, RequestBody.fromBytes(file));
        return key;
    }
    private byte[] downloadFileFromS3(FileData fileData) throws IOException {
        GetObjectRequest getObjectRequest = GetObjectRequest.builder()
                .bucket(bucketName)
                .key(fileData.getFileKey())
                .build();
        ResponseInputStream<GetObjectResponse> responseInputStream =
s3Client.getObject(getObjectRequest);
        ByteArrayOutputStream outputStream = new ByteArrayOutputStream();
        byte[] buffer = new byte[1024];
        int bytesRead;
        while ((bytesRead = responseInputStream.read(buffer)) != -1) {
            outputStream.write(buffer, 0, bytesRead);
        }
        return outputStream.toByteArray();
    }
    private void requireUserHasAccessToFile(FileData fileData) {
```

```
        User currentUser = AuthenticationUtils.getAuthenticatedUser();
        boolean isCurrentUserNotHasAccess =
fileAccessRepository.findAllByFileData(fileData).stream()
                .noneMatch(fileAccess -> fileAccess.getUser().equals(currentUser));

        if (!fileData.getOwner().equals(currentUser) &&
isCurrentUserNotHasAccess) {
            throw new AccessNotGrantedException("User does not has access to file");
        }
    }
    private void requireUserHasNotAccessToFile(FileData fileData, User user) {
        boolean isCurrentUserHasAccess =
fileAccessRepository.findAllByFileData(fileData).stream()
                .anyMatch(fileAccess -> fileAccess.getUser().equals(user));
        if (fileData.getOwner().equals(user) || isCurrentUserHasAccess) {
            throw new AccessNotGrantedException("User already has access to file");
        }
    }
}
```

ДОДАТОК К

ПРОТОКОЛ ПЕРЕВІРКИ НАВЧАЛЬНОЇ (КВАЛІФІКАЦІЙНОЇ) РОБОТИ

Назва роботи:_____ Вдосконалений метод аутентифікації для доступу до даних в хмарному середовищі

_____

Тип роботи:_____Магістерська кваліфікаційна робота_____

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

Підрозділ _____ кафедра обчислювальної техніки_____

(кафедра, факультет (інститут), навчальна група)

Керівник_____Войцеховська О.В., доц. каф. ОТ_____

(прізвище, ініціали, посада)

Показники звіту подібності

Turnitin	
Оригінальність	98%
Загальна схожість	2%

Аналіз звіту подібності (відмітити потрібне)

- ✓ Запозичення, виявлені у роботі, оформлені коректно і не містять ознак плагіату.
- ☐ Виявлені у роботі запозичення не мають ознак плагіату, але їх надмірна кількість викликає сумніви щодо цінності роботи і відсутності самостійності її автора. Роботу направити на доопрацювання.
- ☐ Виявлені у роботі запозичення є недобросовісними і мають ознаки плагіату та/або в ній містяться навмисні спотворення тексту, що вказують на спроби приховування недобросовісних запозичень.

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

Автор _____ _____ Стасюк Є. Є._____

(підпис) (прізвище, ініціали)

Опис прийнятого рішення

_____

_____

_____

_____

Особа, відповідальна за перевірку_____ _Захарченко С.М__

(підпис) (прізвище, ініціали)

Експерт _____ _____

(за потреби) (підпис) (прізвище, ініціали, посада)