Вінницький національний технічний університет

Факультет менеджменту та інформаційної безпеки

Кафедра менеджменту та безпеки інформаційних систем

БАКАЛАВРСЬКА КВАЛІФІКАЦІЙНА РОБОТА

на тему: "Розробка програми захищеної віддаленої роботи користувача у хмарному сховищі з використанням штучного інтелекту"

Виконав: студент 4 курсу, гр.2КІТС-21б
спеціальності 125– Кібербезпека
Освітня програма – Кібербезпека
інформаційних технологій та систем
(шифр і назва напряму підготовки, спеціальності)

Стойківський Ю.В.
(прізвище та ініціали)

Керівник: д.т.н., проф. каф. МБІС
Яремчук Ю.Є.
(прізвище та ініціали)
«4» червня 2025 р.

Рецензент: к.т.н., доц., доцент каф. ОТ
Савицька Л.А.
(прізвище та ініціали)
«4» червня 2025 р.

Допущено до захисту
Голова секції УБ кафедри МБІС
д.т.н., проф., Яремчук Ю.Є.
"4" червня 2025р.

Вінниця ВНТУ – 2025

Вінницький національний технічний університет
Факультет менеджменту та інформаційної безпеки
Кафедра менеджменту та безпеки інформаційних систем

Рівень вищої освіти І-й (бакалаврський)
Галузь знань 12 – Інформаційні технології
Спеціальність 125 – Кібербезпека
Освітньо-професійна програма - Кібербезпека інформаційних технологій та систем

ЗАТВЕРДЖУЮ
Голова секції УБ, кафедра МБІС
д.т.н., проф. Яремчук Ю.Є.
" 20 " березня 2025 р.

З А В Д А Н Н Я
на бакалаврську кваліфікаційну роботу студенту
Стойківський Юрій Володимирович
(прізвище, ім`я, по-батькові)

1. Тема роботи Розробка програми захищеної віддаленої роботи користувача у хмарному сховищі з використанням штучного інтелекту
Керівник роботи д.т.н., проф. Яремчук Ю.Є.
(прізвище, ім`я, по-батькові, науковий ступінь, вчене звання)
затверджені наказом вищого навчального закладу від " 20 " березня 2025року № 97
2. Термін подання студентом роботи 26 травня 2025р
3. Вихідні дані до роботи Електронні джерела, книги, журнали та наукові статті по темі

4. Зміст текстової частини:
Для досягнення мети було поставлено такі задачі: дослідити сучасні підходи до організації захищеного віддаленого доступу в хмарних середовищах; обґрунтувати вибір технологій для розробки системи; спроєктувати архітектуру застосунку; реалізувати функціонал реєстрації, авторизації та керування файлами; інтегрувати елементи штучного інтелекту для виявлення аномальної активності користувача; протестувати систему та підтвердити її працездатність.

5. Перелік ілюстративного матеріалу
Порівняльна таблиця популярних рішень; блок схема алгоритму та представлення ілюстрацій програми користувача.

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Розділ I	Яремчук Ю.Є., д.т.н., проф. кафедри МБІС		
Розділ II	Яремчук Ю.Є., д.т.н., проф. кафедри МБІС		
Розділ III	Яремчук Ю.Є., д.т.н., проф. кафедри МБІС		

7.Дата видачі завдання 20 березня 2025 р.

КАЛЕНДАРНИЙ ПЛАН

№	Назва та зміст етапу	Термін виконання		Примітка
		початок	закінчення	
1	Аналіз предметної області	23.03.2025	30.03.2025	Виконано
2	Аналіз методів розв'язання задачі	25.03.2025	08.04.2025	Виконано
3	Постановка задач розробки програмного модулю	11.04.2025	16.04.2025	Виконано
4	Дослідження інформаційних джерел	16.04.2025	22.04.2025	Виконано
5	Аналіз варіантів роботи модуля	22.04.2025	29.04.2025	Виконано
6	Розробка алгоритму роботи	29.04.2025	05.05.2025	Виконано
7	Реалізація програмного модулю	05.05.2025	12.05.2025	Виконано
8	Тестування програмного модулю	12.05.2025	20.05.2025	Виконано
9	Оформлення матеріалів до захисту БДР	20.05.2025	26.05.2025	Виконано

Студент _____ Стойківський Ю.В.
(підпис) (прізвище та ініціали)

Керівник роботи _____ Яремчук Ю.Є.
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

Бакалаврська кваліфікаційна робота складається з 69 сторінок формату А4, містить 10 рисунків, 0 таблиць, список використаних джерел налічує 25 найменувань.

У дипломній роботі розглянуто процес розробки програмного забезпечення для захищеної віддаленої роботи користувача у хмарному сховищі з використанням штучного інтелекту. Метою проєкту є створення функціональної, безпечної та зручної системи, що дозволяє користувачам здійснювати реєстрацію, авторизацію, завантаження, зберігання та управління файлами в хмарному середовищі, а також контролювати активність з метою виявлення потенційно підозрілих дій.

У роботі проведено аналіз предметної області, визначено сучасні тенденції в галузі хмарних обчислень, систем безпеки та інтеграції елементів штучного інтелекту. Для реалізації програмного забезпечення обрано сучасні вебтехнології: React як засіб створення клієнтської частини та Firebase як хмарну платформу для автентифікації, зберігання файлів і бази даних.

Описано архітектуру та структуру програмної системи, розроблено алгоритм її функціонування. Продемонстровано, як впроваджено логіку взаємодії користувача з інтерфейсом, а також базові функції аналізу поведінки — виявлення аномальної активності через обробку журналів дій.

У результаті реалізовано повнофункціональний веб-застосунок, що відповідає вимогам безпеки, доступності та масштабованості, з можливістю подальшого розширення і впровадження інтелектуальних інструментів аналізу.

Ключові слова: хмарне сховище, віддалений доступ, React, Firebase, авторизація, управління файлами, безпека, аномалії, штучний інтелект.

ANNOTATION

The bachelor's qualification work consists of 69 pages of A4 format, contains 10 figures, 0 tables, the list of references includes 25 items.

The thesis examines the process of developing software for secure remote user work in the cloud storage using artificial intelligence. The goal of the project is to create a functional, secure, and convenient system that allows users to register, authorize, upload, store, and manage files in the cloud environment, as well as monitor activity to identify potentially suspicious activities.

The paper analyzes the subject area, identifies current trends in the field of cloud computing, security systems, and integration of artificial intelligence elements. Modern web technologies were chosen to implement the software: React as a means of creating the client side and Firebase as a cloud platform for authentication, file storage, and database.

The architecture and structure of the software system are described, and an algorithm for its functioning is developed. It is demonstrated how the logic of user interaction with the interface is implemented, as well as the basic functions of behavior analysis - detection of anomalous activity through the processing of action logs.

As a result, a full-featured web application that meets the requirements of security, availability, and scalability has been implemented, with the possibility of further expansion and implementation of intelligent analysis tools.

Keywords: cloud storage, remote access, React, Firebase, authorization, file management, security, anomalies, artificial intelligence.

ЗМІСТ

ВСТУП 7
1. АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ІСНУЮЧИХ РІШЕНЬ 9
1.1. Хмарні технології та віддалений доступ до даних 9
1.2. Проблематика безпеки у хмарних середовищах 15
1.3. Використання штучного інтелекту у системах безпеки 18
1.4. Аналіз існуючих рішень 21
1.5. Висновки до розділу 1 25
2 ПРОЄКТУВАННЯ СИСТЕМИ ЗАХИЩЕНОГО ВІДДАЛЕНОГО ДОСТУПУ ДО ХМАРНОГО СХОВИЩА З ВИКОРИСТАННЯМ ШТУЧНОГО ІНТЕЛЕКТУ . 26
2.1. Загальна характеристика системи 26
2.2. Модель використання системи 28
2.3. Архітектура системи 32
2.4. Алгоритм роботи системи 35
2.5. Висновки до розділу 39
3. РЕАЛІЗАЦІЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ 40
3.1. Загальна характеристика середовища реалізації 40
3.2. Структура програмного забезпечення 43
3.3. Основні компоненти інтерфейсу користувача 45
3.4. Взаємодія з Firebase 48
3.5. Реєстрація, авторизація та управління файлами користувача 51
3.6. Приклади роботи програми 57
3.7 Висновки до розділу 63
ВИСНОВКИ 65
ПЕРЕЛІК ПОСИЛАНЬ 67
Додаток А. Технічне завдання ..... **Помилка! Закладку не визначено.**
Додаток Б. Лістинг програмного коду 75
Додаток В. Ілюстраційний матеріал 81
Додаток Г. Протокол перевірки на антиплагіат 86

ВСТУП

Сучасний світ дедалі активніше переходить до цифрового простору, де ключовими стають безпечний доступ до інформації, ефективне зберігання даних і мобільність користувача. Особливу актуальність у цьому контексті набувають хмарні сховища, які дозволяють здійснювати обмін файлами, їхню обробку та спільну роботу незалежно від фізичного місцезнаходження користувача. Проте, з розширенням функціональних можливостей таких систем зростають і вимоги до безпеки, а також потреба в інтелектуальній підтримці користувача. Це зумовлює необхідність створення програмних рішень, які б поєднували хмарні технології, засоби штучного інтелекту та сучасні методи кіберзахисту.

Тема дипломної роботи — «Розробка програми захищеної віддаленої роботи користувача у хмарному сховищі з використанням штучного інтелекту» — передбачає розробку вебзастосунку, що забезпечує користувачу захищений доступ до локального або хмарного сховища з можливістю реєстрації, авторизації, керування файлами та застосування ШІ для виявлення аномальної активності й інтелектуального пошуку файлів.

Мета роботи. полягає в створенні безпечного середовища для роботи користувача з хмарним сховищем, яке доповнюється функціональністю штучного інтелекту.

Для досягнення поставленої мети було сформульовано такі **завдання**:

провести аналіз існуючих рішень у сфері хмарних технологій і захисту даних;

дослідити засоби інтеграції ШІ у вебзастосунки;

спроєктувати архітектуру системи з урахуванням вимог безпеки та гнучкості;

реалізувати програму з функціями реєстрації, авторизації, керування файлами, інтелектуального пошуку та виявлення підозрілих дій;

протестувати розроблену систему на відповідність функціональним і нефункціональним вимогам.

Об'єктом дослідження. є процес організації безпечного доступу користувача до файлового сховища через вебінтерфейс.

Предметом дослідження. виступають методи і засоби розробки програмного забезпечення з використанням хмарних сервісів, систем авторизації та штучного інтелекту.

Актуальність теми. зумовлена зростанням обсягів даних, що зберігаються у хмарі, та необхідністю забезпечення їх надійного захисту, зручності доступу та інтелектуального аналізу. Використання ШІ в подібних системах відкриває нові можливості для автоматизації рутинних процесів, моніторингу безпеки та швидкого пошуку інформації.

Структура роботи. складається з трьох розділів:

У першому розділі проведено аналіз предметної області, огляд існуючих рішень, досліджено сучасні підходи до побудови захищених хмарних сховищ.

У другому — описано загальну архітектуру програми, наведено алгоритм роботи системи, охарактеризовано засоби реалізації.

У третьому розділі наведено практичну реалізацію системи, представлено результати тестування, здійснено аналіз ефективності реалізованих функцій.

1. АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ІСНУЮЧИХ РІШЕНЬ

1.1. Хмарні технології та віддалений доступ до даних

Хмарні технології стали фундаментальною основою сучасної IT-інфраструктури. Хмарне сховище даних — це модель зберігання інформації, при якій дані розміщуються на віддалених серверах, доступ до яких забезпечується через мережу Інтернет.

Термін “хмара” (cloud) в контексті інформаційних технологій вказує на хмарні обчислення або хмарні сервіси. Це означає використання обчислювальних ресурсів, таких як обчислювальна потужність, зберігання даних та програмне забезпечення, які надаються через Інтернет.

У вигляді простого порівняння, можна уявити “хмару” як велике централізоване місце, де можна отримувати доступ до різних послуг і ресурсів за допомогою Інтернету, подібно тому, як ви можете отримувати доступ до води чи електроенергії з централізованих мереж.

В хмарному обчисленні користувачі не повинні прямо володіти або управляти фізичним обладнанням; замість цього, вони можуть використовувати ресурси за потребою та платити лише за фактичне використання. [1]

Хмарні сервіси поділяються на кілька основних категорій в залежності від наданих функцій. Основні категорії хмарних сервісів включають:

1. Інфраструктура як сервіс (IaaS): надання обчислювальної потужності та інфраструктурних ресурсів через Інтернет. Користувачі можуть отримати віртуальні машини, зберігання даних та інші інфраструктурні ресурси за потребою.

2. Платформа як сервіс (PaaS): сервіс надає платформу для розробки, тестування та розгортання додатків. Користувачі можуть концентруватися на розробці програмного забезпечення, оскільки багато інфраструктурних питань приховані від них.

3. Програмне забезпечення як сервіс (SaaS): надання доступу до вже готового програмного забезпечення через Інтернет. Користувачі можуть використовувати програми без необхідності їх установки та оновлення.

4. Хмарні зберігальні системи: ці сервіси дозволяють зберігати та керувати даними в хмарі, надаючи можливість резервного копіювання, синхронізації та спільного використання файлів.

5. Хмарні послуги обробки даних: ці послуги дозволяють виконувати обробку та аналіз даних в хмарі, зазвичай за допомогою спеціалізованих інструментів та фреймворків.

6. Хмарні послуги для розробників (DevOps): ці сервіси дозволяють розробникам та командам управляти, автоматизувати та моніторити різні етапи розробки програмного забезпечення.

Це лише загальні категорії, і в межах кожної з них існують конкретні сервіси від різних постачальників, таких як Amazon Web Services (AWS), Microsoft Azure, Google Cloud Platform (GCP) та інші. [2]

Переваги хмарних технологій

Хмарні технології мають численні переваги, які призводять до їх широкого використання в різних галузях.

Гнучкість та масштабованість: користувачі можуть легко збільшувати або зменшувати використання ресурсів згідно зі змінними потребами. Це забезпечує гнучкість та можливість швидко реагувати на зміни обсягу роботи.

Ефективне використання ресурсів: хмарні постачальники можуть оптимізувати використання обладнання та забезпечити високий рівень ефективності в порівнянні з традиційними моделями власних серверів.

Доступність та надійність: великі хмарні платформи часто працюють на розподілених системах, що забезпечує високий рівень доступності та надійності послуг.

Самообслуговування та автоматизація: користувачі можуть легко самостійно керувати ресурсами, запускати віртуальні машини, налаштовувати сервіси та

автоматизувати багато процесів, що прискорює розгортання та управління інфраструктурою.

Економія витрат: відсутність необхідності власних фізичних серверів та обладнання зменшує капітальні витрати. Крім того, оплата може базуватися на фактичному використанні ресурсів, що знижує витрати в порівнянні з традиційними моделями придбання обладнання.

Глобальний доступ: користувачі можуть отримувати доступ до своїх даних та ресурсів з будь-якого місця, де є Інтернет, що полегшує роботу на віддалених робочих місцях та сприяє спільній роботі.

Безпека: великі хмарні постачальники зазвичай вкладають значні зусилля в забезпечення безпеки даних, включаючи шифрування, автентифікацію та інші заходи.

Швидке впровадження нових функцій та оновлень: хмарні постачальники можуть швидко впроваджувати нові функції та оновлення, забезпечуючи користувачам доступ до останніх технологій без необхідності самостійного оновлення інфраструктури.

Ці переваги роблять хмарні технології дуже привабливими для багатьох компаній та організацій.

Недоліки використання хмарних сервісів

Незважаючи на багато переваг, існують і деякі недоліки використання хмарних сервісів, які можуть вплинути на різні аспекти бізнесу та користувачів.

Залежність від інтернету: використання хмарних сервісів передбачає наявність стабільного Інтернет-з’єднання. В разі відсутності з’єднання може виникнути проблема доступу до даних та сервісів.

Приватність та безпека даних: застосування хмарних сервісів може породжувати питання щодо конфіденційності та безпеки даних. Користувачі можуть бути обурені, якщо їх конфіденційна інформація знаходиться на серверах інших компаній.

Відмова від контролю над інфраструктурою: у випадку хмарних сервісів користувачі втрачають частину контролю над інфраструктурою, так як

обчислювальні ресурси та деякі аспекти безпеки управляються постачальником хмарних послуг.

Можливість виникнення проблем безпеки: хоча постачальники хмарних послуг вкладають значні зусилля в забезпечення безпеки, існують ризики виникнення інцидентів безпеки, таких як витоки даних або атаки на систему.

Вартість: на перший погляд хмарні послуги можуть здаватися економічно вигідними, але в деяких випадках витрати можуть зрости з часом або через несподівані обставини, такі як збільшення використання ресурсів.

Обмежені можливості настроювання: деякі хмарні сервіси можуть обмежувати можливості користувачів налаштовувати інфраструктуру та програмне забезпечення під їхні потреби.

Відсутність контролю над місцезнаходженням даних: користувачі можуть не мати повного контролю над тим, де фізично знаходяться їхні дані, оскільки хмарні послуги можуть використовувати сервери в різних країнах.

Можливість зміни вартості послуг: вартість хмарних послуг може змінюватися, іноді призводячи до неочікуваних витрат для користувачів.

Хоча багато з цих недоліків можливо управляти або зменшити за допомогою правильного планування та вибору постачальника хмарних послуг, вони все ж робляться об’єктом уваги при впровадженні хмарних технологій. [3]

Віддалений доступ до даних — це можливість взаємодії з інформаційними ресурсами, що знаходяться на віддалених серверах або у хмарному середовищі, без необхідності фізичної присутності користувача біля місця зберігання інформації. Такий доступ реалізується завдяки мережевим технологіям, що дозволяють працювати з даними з будь-якого пристрою, підключеного до Інтернету, і є надзвичайно актуальним в умовах стрімкої цифровізації та переходу до віддалених форматів праці.

З технічного погляду, організація віддаленого доступу базується на використанні низки рішень, серед яких найбільш поширеними є віртуальні приватні мережі, веб-інтерфейси хмарних сервісів, спеціалізовані клієнтські додатки та захищені протоколи передачі даних. Віртуальні приватні мережі

дозволяють створювати зашифровані канали зв’язку між користувачем і сервером, забезпечуючи конфіденційність і цілісність переданої інформації. Веб-доступ через браузер є зручним і універсальним способом взаємодії з хмарними ресурсами, адже не потребує встановлення додаткового програмного забезпечення. Окрім цього, багато хмарних платформ пропонують клієнтські додатки, які забезпечують синхронізацію файлів між локальним пристроєм і хмарним сховищем у фоновому режимі. Також поширеним є використання спеціалізованих протоколів, таких як FTP, SFTP, SSH або RDP, які дозволяють працювати з даними або отримувати доступ до віддалених робочих столів та серверів.

Разом із широкими можливостями, які надає віддалений доступ, зростають і ризики, пов’язані з інформаційною безпекою. Однією з головних загроз є несанкціонований доступ, що може бути наслідком викрадення облікових даних або зловмисних дій всередині організації. Також значною проблемою є використання незахищених публічних мереж, де трафік може бути перехоплений. Не менш важливою є проблема недостатньої обізнаності користувачів про потенційні загрози, що часто призводить до фішингових атак або компрометації облікових записів. Ще одним викликом є складність ефективного контролю за поведінкою користувачів у системі, що унеможливлює своєчасне виявлення внутрішніх порушень або загроз.

Для подолання цих викликів дедалі частіше використовуються технології штучного інтелекту, які дозволяють здійснювати інтелектуальний аналіз поведінки користувача, виявляти аномальні дії, реагувати на них у реальному часі, а також прогнозувати можливі інциденти інформаційної безпеки. Використання методів машинного навчання дає змогу переходити від реактивних заходів захисту до проактивних, де система здатна адаптувати свої дії залежно від змін у поведінці користувачів або умов середовища.

Узагальнюючи, можна стверджувати, що ефективна реалізація безпечного віддаленого доступу до даних потребує комплексного підходу, що поєднує технічні засоби захисту, політики управління доступом та інтелектуальні алгоритми аналізу активності. Саме така інтеграція є необхідною умовою побудови сучасних, гнучких

і стійких до загроз інформаційних систем, орієнтованих на віддалену роботу в хмарному середовищі.

Використання віддаленого доступу до даних стало невіддільною частиною сучасних інформаційних систем, зокрема в умовах розвитку хмарних технологій та цифрової трансформації організацій. Основною перевагою такого підходу є зручність, оскільки користувач отримує можливість взаємодіяти з інформацією в будь-який час і з будь-якого місця, що значно підвищує гнучкість і мобільність. Це особливо актуально в умовах дистанційної роботи, коли фізичний доступ до робочого місця є обмеженим або неможливим.

Крім того, віддалений доступ забезпечує економічну ефективність, оскільки дозволяє зменшити витрати на фізичну інфраструктуру, обслуговування серверів і робочих місць, а також оптимізувати витрати на оренду офісних приміщень. Хмарні сервіси, що часто є основою віддаленого доступу, дозволяють масштабувати ресурси відповідно до поточних потреб, що сприяє підвищенню продуктивності праці та ефективному розподілу обчислювальних потужностей. Завдяки централізованому зберіганню даних також спрощується управління інформацією та резервне копіювання, а доступ до актуальних версій файлів може здійснюватися кількома користувачами одночасно.

Однак, разом із перевагами, віддалений доступ до даних має низку недоліків, основним з яких є високі вимоги до безпеки. Відкритий характер з’єднань через Інтернет створює ризик перехоплення даних, несанкціонованого доступу, а також використання шкідливого програмного забезпечення. Зокрема, уразливими є ті системи, де не реалізовано багатофакторну автентифікацію, шифрування або розмежування прав доступу. Втрата облікових даних користувачем або недостатньо сильна політика паролів може призвести до серйозних порушень конфіденційності. Ще одним недоліком є залежність від якості мережевого з’єднання: при нестабільному інтернет-зв’язку знижується продуктивність роботи, виникає затримка в обробці даних, або ж доступ може бути повністю втрачений.

Також слід зазначити, що у випадку з використанням хмарних платформ користувач часто втрачає повний контроль над фізичним розташуванням та

обробкою своїх даних, що може суперечити корпоративній або національній політиці зберігання інформації. Водночас залежність від сторонніх сервісів і провайдерів створює додаткові ризики, пов’язані з надійністю, стабільністю та юридичною відповідальністю у разі інцидентів. [4]

Таким чином, віддалений доступ до даних — це ефективний інструмент для організації сучасного робочого процесу, який значно розширює можливості користувачів і підвищує продуктивність, але вимагає особливої уваги до питань безпеки, контролю доступу, захисту персональних і конфіденційних даних, а також технічної надійності каналів зв’язку.

1.2. Проблематика безпеки у хмарних середовищах

Хмарні середовища дозволяють легко обмінюватися даними, що зберігаються в них. Ці середовища доступні безпосередньо з загальнодоступного Інтернету та включають можливість легко обмінюватися даними з іншими сторонами через прямі запрошення електронною поштою або шляхом спільного використання загальнодоступного посилання на дані.

Простота обміну даними в хмарі – хоча це головний актив і ключ до співпраці в хмарі – викликає серйозні занепокоєння щодо втрати або витоку даних. Насправді 69% організацій вказують на це як на найбільшу проблему безпеки хмарних технологій. Обмін даними за допомогою загальнодоступних посилань або налаштування хмарного сховища на загальнодоступне робить їх доступними для будького, хто знає про посилання, і існують спеціальні інструменти для пошуку в Інтернеті цих незахищених хмарних розгортань.

Конфіденційність і конфіденційність даних є основною проблемою для багатьох організацій. Положення про захист даних, як-от Загальний регламент ЄС щодо захисту даних (GDPR), Закон про мобільність і доступність медичного страхування (HIPAA), Стандарт безпеки даних індустрії платіжних карток (PCI DSS) та багато інших зобов’язують захистити дані клієнтів і накладають суворі штрафи за збої безпеки. Крім того, організації мають велику кількість внутрішніх даних, необхідних для збереження конкурентної переваги.

Розміщення цих даних у хмарі має свої переваги, але також створює серйозні проблеми з безпекою для 66% організацій. Багато організацій запровадили хмарні обчислення, але їм не вистачає знань, щоб переконатися, що вони та їхні співробітники використовують їх безпечно. У результаті конфіденційні дані знаходяться під загрозою розголошення, про що свідчить величезна кількість порушень хмарних даних. [5]

Фішери зазвичай використовують хмарні програми та середовища як привід для своїх фішингових атак. Зі зростанням використання хмарної електронної пошти (GSuite, Microsoft 365 тощо) і служб обміну документами (Google Drive, Dropbox, OneDrive) співробітники звикли отримувати електронні листи з посиланнями, які можуть попросити їх підтвердити обліковий запис. облікові дані, перш ніж отримати доступ до певного документа або веб-сайту. Це дозволяє кіберзлочинцям легко дізнатися облікові дані співробітника для хмарних сервісів. У результаті випадкове відкриття хмарних облікових даних викликає серйозне занепокоєння для 44% організацій, оскільки це потенційно ставить під загрозу конфіденційність і безпеку їхніх хмарних даних та інших ресурсів.

Багато організацій мають стратегії реагування на внутрішні інциденти кібербезпеки. Оскільки організація володіє всією внутрішньою мережевою інфраструктурою, а персонал служби безпеки працює на місці, можна заблокувати інцидент. Крім того, це право власності на їх інфраструктуру означає, що компанія, ймовірно, має видимість, необхідну для визначення масштабу інциденту та виконання відповідних дій з усунення.

Завдяки хмарній інфраструктурі компанія має лише часткову видимість і право власності на свою інфраструктуру, що робить традиційні процеси та інструменти безпеки неефективними. У результаті 44% компаній стурбовані своєю здатністю ефективно реагувати на інциденти в хмарі.

Правила захисту даних, такі як PCI DSS і HIPAA, вимагають від організацій продемонструвати, що вони обмежують доступ до захищеної інформації (даних кредитних карток, медичних записів пацієнтів тощо). Це може вимагати створення

фізично або логічно ізольованої частини мережі організації, яка буде доступна лише для працівників, які мають законну потребу в доступі до цих даних. [6]

Під час переміщення даних, захищених цими та подібними правилами, у хмару досягти та продемонструвати відповідність нормативним вимогам може бути складніше. Завдяки хмарному розгортанню організації мають можливість переглядати та контролювати лише деякі рівні своєї інфраструктури. Як наслідок, 42% організацій вважають відповідність законодавству та нормативним вимогам основною проблемою безпеки хмари та потребують спеціалізованих рішень відповідності хмарі.

Більшість хмарних провайдерів мають кілька територіально розподілених центрів обробки даних. Це допомагає підвищити доступність і продуктивність хмарних ресурсів і полегшує для постачальників послуг гарантування того, що вони здатні підтримувати угоди про рівень обслуговування в умовах руйнівних подій, таких як стихійні лиха, відключення електроенергії тощо.

Організації, які зберігають свої дані в хмарі, часто не знають, де насправді зберігаються їхні дані в масиві центрів обробки даних CSP. Це викликає серйозні занепокоєння щодо суверенітету даних, місця розміщення та контролю для 37% організацій. З нормативними актами щодо захисту даних, такими як GDPR, які обмежують, куди можна надсилати дані громадян ЄС, використання хмарної платформи з центрами обробки даних за межами затверджених зон може привести організацію до стану невідповідності нормативним вимогам. Крім того, різні юрисдикції мають різні закони щодо доступу до даних для правоохоронних органів і національної безпеки, що може вплинути на конфіденційність даних і безпеку клієнтів організації.

Хмара надає організаціям ряд переваг; однак вона також має свої власні загрози безпеці та проблеми. Хмарна інфраструктура дуже відрізняється від локального центру обробки даних, і традиційні інструменти та стратегії безпеки не завжди здатні ефективно захистити її. [7]

1.3. Використання штучного інтелекту у системах безпеки

Інформаційна безпека у сучасному цифровому середовищі є одним з найважливіших аспектів функціонування IT-інфраструктури, особливо в умовах віддаленої роботи та зростання обсягів обробки персональних і корпоративних даних у хмарних середовищах. Зі зростанням складності загроз традиційні системи кіберзахисту, що базуються на статичних правилах і фіксованих політиках, поступово поступаються місцем інтелектуальним рішенням, які використовують алгоритми штучного інтелекту (ШІ) та машинного навчання.

Штучний інтелект у сфері кібербезпеки виконує насамперед аналітичну, прогностичну й адаптивну функції. Завдяки здатності до обробки великих обсягів даних у режимі реального часу, ШІ-системи можуть аналізувати поведінкові шаблони користувачів, оцінювати аномалії у мережевому трафіку, виявляти потенційно шкідливі дії, які не потрапляють під класичні визначення атак, та автоматично реагувати на загрози. Це дозволяє значно знизити час реагування на інциденти безпеки та підвищити ефективність захисту інформаційних систем. [8]

Однією з ключових переваг використання ШІ є можливість побудови адаптивних систем, які постійно навчаються на основі нових даних і досвіду, що дозволяє виявляти нові типи атак, зокрема нульового дня (zero-day), які не можуть бути виявлені антивірусними засобами або фаєрволами зі статичною сигнатурою. Сучасні системи з використанням машинного навчання здатні формувати профілі нормальної поведінки користувача або системи, що дає змогу оперативно виявляти будь-які відхилення, які можуть вказувати на спробу несанкціонованого доступу, витік даних або внутрішні загрози.

Також активно розвивається використання ШІ для управління ідентифікацією та доступом (Identity and Access Management, IAM). Алгоритми ШІ дозволяють динамічно оцінювати ризики при спробі входу в систему, зважаючи на час, геолокацію, тип пристрою та інші фактори, що допомагає виявляти підозрілу активність навіть при використанні правильних облікових даних. Наприклад, якщо користувач намагається увійти до системи з незвичної локації або пристрою,

система може автоматично запитати додаткову автентифікацію або обмежити доступ. [9]

У хмарних інфраструктурах ШІ використовується для моніторингу цілісності даних, аналізу журналів подій, управління правами доступу до ресурсів, а також в автоматичних системах реагування на інциденти (SOAR — Security Orchestration, Automation, and Response). Такі системи дозволяють автоматизувати велику кількість рутинних дій фахівців із кібербезпеки, включаючи обробку інцидентів, генерацію звітів та впровадження політик захисту.

Не менш важливим є застосування ШІ у боротьбі з фішингом та соціально-інженерними атаками. Завдяки обробці текстів повідомлень електронної пошти або чатів за допомогою технологій обробки природної мови (NLP), системи можуть виявляти потенційно небезпечні листи ще до того, як користувач їх відкриє. Деякі сучасні рішення також здатні автоматично позначати такі повідомлення як небезпечні або блокувати їх доставку.

Водночас слід зазначити, що використання ШІ у системах безпеки не є позбавленим викликів. Надмірна автоматизація може призводити до помилкових спрацьовувань або блокування легітимної активності користувачів. Крім того, моделі ШІ потребують постійного оновлення та навчання, що вимагає високої якості та актуальності вхідних даних. Якщо система навчена на неповних або некоректних даних, вона може демонструвати упередженість або некоректно інтерпретувати поведінку.

Ще одним важливим аспектом є етичні питання, пов’язані з використанням штучного інтелекту у сфері безпеки. Йдеться насамперед про прозорість прийняття рішень системою, можливість аудитування її дій, а також уникнення надмірного контролю за діями користувачів, що може порушувати їхнє право на конфіденційність.

Підсумовуючи, можна стверджувати, що штучний інтелект є потужним інструментом у забезпеченні безпеки сучасних інформаційних систем. Його використання дозволяє здійснювати проактивний захист, зменшити навантаження на фахівців, прискорити виявлення загроз та забезпечити гнучку адаптацію до

динамічних умов цифрового середовища. Проте ефективне впровадження таких рішень потребує не лише технічної реалізації, а й ретельного управління ризиками, безперервного навчання моделей і дотримання етичних принципів.

Використання штучного інтелекту в системах інформаційної безпеки має цілу низку переваг, які зумовлюють його стрімке впровадження у сучасні IT-інфраструктури. Насамперед, це можливість оперативної обробки великих обсягів даних у режимі реального часу. Завдяки цьому системи на базі ШІ здатні ефективно виявляти аномальні шаблони поведінки, що можуть свідчити про спроби несанкціонованого доступу або інші кіберзагрози. Така здатність до аналізу подій у динаміці значно перевершує традиційні методи моніторингу, які часто ґрунтуються на статичних правилах.

Ще однією суттєвою перевагою є здатність алгоритмів машинного навчання адаптуватися до нових типів загроз без необхідності ручного оновлення сигнатур або конфігурацій. Це особливо важливо у боротьбі з атаками нульового дня, які не мають відомих сигнатур і можуть залишатися непоміченими традиційними засобами захисту. Крім того, ШІ дозволяє автоматизувати виявлення, класифікацію й обробку інцидентів безпеки, знижуючи тим самим навантаження на спеціалістів з інформаційної безпеки та підвищуючи загальну ефективність реагування.

Штучний інтелект також сприяє побудові більш гнучких та динамічних систем управління доступом, де рішення приймаються на основі контексту — геолокації, типу пристрою, часу доби тощо. Це дозволяє значно знизити ризики компрометації облікових даних. Окремо варто відзначити можливості використання методів обробки природної мови (NLP) для виявлення фішингових повідомлень, що значно підвищує захищеність користувачів від соціальної інженерії.

Проте, попри численні переваги, застосування ШІ в безпеці супроводжується низкою викликів і обмежень. Однією з головних проблем є можливість помилкових спрацьовувань. Інтелектуальні системи, особливо на початкових етапах впровадження, можуть некоректно інтерпретувати поведінку користувача або

мережеву активність, внаслідок чого легітимні дії будуть позначені як загрозливі. Це може призвести до втрат доступу або перерв у роботі.

Ще одним суттєвим недоліком є потреба в якісних, репрезентативних та актуальних даних для навчання моделей. У разі недостатнього або викривленого набору даних штучний інтелект може демонструвати упередженість або неправильну класифікацію, що прямо впливає на рівень захисту. Крім того, моделі потребують постійного оновлення, що потребує значних обчислювальних ресурсів і участі фахівців.

Слід також звернути увагу на питання прозорості прийняття рішень. Більшість алгоритмів глибокого навчання є "чорними скриньками", тобто не дають можливості пояснити, чому саме було прийняте певне рішення. У сфері безпеки, де важливо мати змогу обґрунтувати дії системи, це створює додаткові складнощі в аудиті й аналізі інцидентів.

Крім того, надмірна довіра до автоматизованих систем без належного контролю з боку людини може стати причиною серйозних вразливостей, особливо якщо зловмисники навчаться маніпулювати алгоритмами ШІ або використовувати їх для створення фальшивих атак, що відволікають увагу від реальних загроз. [10]

Отже, інтеграція штучного інтелекту у сферу безпеки відкриває нові горизонти для захисту даних та інфраструктур, але водночас потребує виваженого підходу, технічної грамотності, етичної відповідальності та постійного удосконалення моделей. Ефективність таких рішень безпосередньо залежить від балансу між автоматизацією, людським контролем та якістю даних, на яких базується інтелектуальна система.

1.4. Аналіз існуючих рішень

У сучасному цифровому середовищі питання безпечного доступу до хмарних сервісів набуває особливої актуальності. Широке впровадження віддаленої роботи, зростання обсягів даних та ускладнення кіберзагроз стимулюють розвиток інноваційних рішень, здатних забезпечити як зручність, так і високий рівень захищеності. У цьому контексті важливо розглянути вже наявні програмні

продукти та технології, які реалізують захищений віддалений доступ до хмарних середовищ з використанням елементів штучного інтелекту. [11]

Одним із популярних корпоративних рішень є Microsoft Azure Virtual Desktop (AVD). Ця платформа надає можливість створення віртуальних робочих столів з централізованим керуванням політиками доступу, підтримкою багатофакторної автентифікації (MFA) та інтеграцією з Azure AI. Завдяки машинному навчанню, AVD може відстежувати підозрілу активність та автоматично обмежувати доступ при виявленні аномалій. Водночас, цей інструмент орієнтований переважно на великі підприємства і вимагає значних ресурсів на налаштування та обслуговування.

Ще одним прикладом є Amazon WorkSpaces, який дозволяє запускати віртуальні робочі середовища з доступом до Amazon Web Services (AWS). Ця система підтримує розширене шифрування, контроль мережевих параметрів і використання інтелектуальних інструментів безпеки, таких як Amazon GuardDuty. Хоча сервіс і забезпечує високу надійність, він не завжди є оптимальним у плані витрат для невеликих компаній або індивідуальних користувачів.

З-поміж безкоштовних або умовно безкоштовних рішень варто згадати AnyDesk і TeamViewer, які широко використовуються для віддаленого доступу. Вони забезпечують базовий рівень шифрування даних, автентифікацію користувача та підтримку підключення до хмарного середовища. Проте, їхня архітектура менш гнучка у плані інтеграції з власними системами безпеки та алгоритмами штучного інтелекту. До того ж, у безкоштовних версіях доступні обмежені функції моніторингу й автоматизації безпеки.

На ринку також з’являються хмарні рішення нового покоління, які використовують ШІ для глибокого аналізу поведінкових шаблонів користувачів. Наприклад, Citrix Analytics for Security на основі аналізу активності в режимі реального часу формує ризикові профілі користувачів і приймає автоматичні рішення про надання або обмеження доступу. Такий підхід забезпечує високий рівень персоналізації й реагування на потенційні інциденти.

Крім того, активно розвиваються Zero Trust моделі, в основі яких лежить принцип постійної перевірки кожної дії користувача незалежно від його попереднього статусу. Компанії на кшталт Google і Cisco інтегрують моделі Zero Trust у свої хмарні платформи, застосовуючи алгоритми штучного інтелекту для оцінки контексту кожного запиту до системи. [12]

Попри велику кількість існуючих рішень, жодне з них не є універсальним. Деякі орієнтовані на великі організації з достатнім бюджетом, інші — на швидкий і простий доступ без глибокої кастомізації. Існує потреба у створенні гібридного програмного забезпечення, яке поєднуватиме гнучкість, адаптивність та вбудовані інструменти штучного інтелекту, здатні ефективно реагувати на загрози в реальному часі. Саме така концепція лягла в основу розробки власного рішення, яке розглядатиметься у наступних розділах цієї роботи.

Сучасні хмарні рішення

Microsoft Azure Virtual Desktop (AVD) є однією з найбільш розвинених платформ, яка дозволяє організаціям запускати віртуальні робочі столи, забезпечуючи централізоване управління ресурсами, доступом і безпекою. Завдяки сервісу Azure AI система може здійснювати аналіз поведінки користувачів, виявляти аномалії та автоматично запускати механізми реагування. Суттєвою перевагою є інтеграція з іншими сервісами Microsoft, як-от OneDrive, SharePoint і Defender for Cloud.

Amazon WorkSpaces, натомість, пропонує більш гнучке ціноутворення та ширший вибір конфігурацій. Захист даних забезпечується багаторівневим шифруванням, системами багатофакторної автентифікації, а також інструментами машинного навчання для виявлення підозрілої поведінки через Amazon Macie та GuardDuty. Це рішення є привабливим для середнього бізнесу завдяки гнучкості налаштування середовищ.

Google Cloud Workstations пропонує високопродуктивні середовища з потужною інтеграцією з Google AI та автоматизованими системами моніторингу. Серед переваг – автоматичне масштабування, підтримка Zero Trust принципів через

BeyondCorp та глибока інтеграція з Google Workspace. Слабким місцем є складність в адаптації для новачків через високу складність конфігурацій.

Локальні програми для віддаленого доступу

TeamViewer і AnyDesk — це кросплатформенні рішення, що дозволяють здійснювати швидкий віддалений доступ до ПК. Обидва продукти підтримують TLS-1.2 шифрування, авторизацію через токени та контроль сесій. Проте відсутність глибокої інтеграції з системами штучного інтелекту обмежує їхню здатність до проактивного реагування на загрози. До того ж, повний функціонал доступний лише у комерційних версіях.

Новітні моделі на базі Zero Trust

Окрему увагу слід звернути на Zero Trust Architecture, яка вважається передовою концепцією кіберзахисту. Наприклад, рішення Cisco Duo і Google BeyondCorp реалізують принцип нульової довіри, де кожен запит перевіряється на основі багатьох факторів (місце розташування, пристрій, поведінка користувача тощо). У таких системах ШІ застосовується для адаптивного контролю доступу, що знижує ризик компрометації навіть у разі викрадення облікових даних. [13]

Характеристика	Microsoft AVD	Amazon WorkSpaces	Google Cloud Workstations	TeamViewer	AnyDesk	Cisco Duo / BeyondCorp
Тип рішення	Хмарне	Хмарне	Хмарне	Локальне/ хмарне	Локальне/ хмарне	Zero Trust / Cloud
Інтеграція ШІ	Так (Azure AI)	Так (Macie, GuardDuty)	Так (Google AI)	Ні	Ні	Так
Безпека (MFA, шифрування)	Висока	Висока	Висока	Середня	Середня	Висока
Підтримка Zero Trust	Часткова	Часткова	Повна	Відсутня	Відсутня	Повна
Проста інтеграція	Середня	Висока	Низька	Висока	Висока	Середня
Придатність для малого бізнесу	Середня	Висока	Низька	Висока	Висока	Середня
Вартість	Висока	Середня	Середня/висока	Безкоштовна/ Платна	Безкоштовна/ Платна	Платна (для підприємств)

Рисунок 1 – Порівняльна таблиця популярних рішень

Аналіз існуючих рішень у сфері захищеного віддаленого доступу до хмарних середовищ показав, що сучасні платформи активно впроваджують елементи штучного інтелекту та концепцію Zero Trust для підвищення рівня безпеки. Однак

жодне з наявних рішень не є універсальним і не повною мірою задовольняє потреби малих та середніх користувачів в умовах обмежених ресурсів. Це створює актуальну потребу у розробці гнучкого, масштабованого та інтелектуального програмного продукту, здатного адаптуватися до конкретних умов експлуатації та забезпечити ефективний захист даних у хмарному середовищі. [14]

1.5. Висновки до розділу 1

У першому розділі було проведено ґрунтовний аналіз теоретичних засад, що стосуються захищеної віддаленої роботи користувача у хмарному середовищі з використанням штучного інтелекту. Розглянуто основні поняття, пов’язані з організацією віддаленого доступу до даних, проаналізовано сучасні підходи до побудови безпечних інформаційних систем та роль штучного інтелекту у забезпеченні кібербезпеки.

Особливу увагу приділено аналізу сучасних рішень, які реалізують віддалений доступ до хмарних сервісів із вбудованими засобами захисту та інтелектуального моніторингу. Встановлено, що хоча на ринку існує низка потужних платформ, таких як Microsoft AVD, Amazon WorkSpaces чи Google Cloud Workstations, більшість із них орієнтовані на корпоративний сегмент, є фінансово затратними або обмеженими у можливостях адаптації під конкретні вимоги.

Таким чином, на основі виявлених тенденцій, переваг та недоліків існуючих рішень, сформовано вимоги до майбутньої розробки — створення програмного забезпечення, яке забезпечить безпечний, адаптивний та інтелектуально підтримуваний віддалений доступ до хмарних ресурсів для користувачів різних категорій.

2 ПРОЄКТУВАННЯ СИСТЕМИ ЗАХИЩЕНОГО ВІДДАЛЕНОГО ДОСТУПУ ДО ХМАРНОГО СХОВИЩА З ВИКОРИСТАННЯМ ШТУЧНОГО ІНТЕЛЕКТУ

Зважаючи на результати теоретичного аналізу, проведеного у попередньому розділі, у цьому розділі розглянуто проєктування власної системи, що забезпечує безпечний віддалений доступ до хмарного сховища з використанням інструментів штучного інтелекту. Особливу увагу приділено постановці задачі, визначенню вимог до системи, вибору відповідної архітектури, інструментів реалізації, а також моделюванню загроз і методам їх виявлення. Метою даного етапу є створення функціонального та безпечного рішення, здатного адаптуватися до змін у поведінці користувача та забезпечувати надійний захист конфіденційних даних у хмарному середовищі.

2.1. Загальна характеристика системи

Із зростанням обсягів цифрових даних та потребою у доступі до них з різних пристроїв з’являється необхідність у надійних і функціональних хмарних системах керування файлами. Запропонована система розроблена як безпечне хмарне сховище, яке дозволяє користувачам здійснювати повний спектр операцій із файлами (створення, редагування, перегляд, видалення, збереження), а також забезпечує захист даних за допомогою автентифікації, авторизації та інтелектуального аналізу активності.

Основним призначенням розроблюваної системи є забезпечення віддаленої роботи користувача з особистими файлами у хмарному середовищі, незалежно від типу пристрою чи платформи. Такий підхід дозволяє обробляти дані онлайн без потреби встановлення спеціалізованого програмного забезпечення, що підвищує гнучкість і зручність користування системою.

Застосунок реалізовано як SPA (Single Page Application) за допомогою бібліотеки React, що дозволяє забезпечити швидку реакцію інтерфейсу,

мінімізацію запитів до сервера та безперервну взаємодію користувача із системою без оновлення сторінки. Компонентна структура React забезпечує масштабованість, модульність і повторне використання коду. [15]

На стороні бекенду використовується платформа Firebase, яка надає набір хмарних сервісів, зокрема:

Firebase Authentication — для реєстрації та входу користувача з підтримкою email-пароля, OAuth та інших механізмів;

Cloud Firestore — для зберігання структурованих даних, таких як метадані файлів, структури папок, журнал дій користувача;

Firebase Storage — для фізичного зберігання файлів (зображення, документи, текстові файли тощо).

Користувач після автентифікації потрапляє у власне віртуальне середовище, де має змогу:

створювати нові папки та вкладені підкаталоги;

завантажувати файли з локального пристрою до хмари;

переглядати або редагувати файли у візуальному редакторі;

видаляти непотрібні елементи.

Дані користувачів ізольовано: кожен обліковий запис має власне файлове середовище. Доступ до даних можливий лише за наявності відповідної авторизації. Таким чином реалізовано концепцію zero trust — "жодної довіри" до зовнішнього доступу без перевірки.

Ключовою інноваційною особливістю системи є інтеграція модуля штучного інтелекту, що виконує аналіз поведінкової активності користувача. Зокрема, реєструються всі ключові дії (завантаження, видалення, створення файлів/папок), які записуються до спеціальної колекції Firestore. Використовуючи просту евристичну модель, система аналізує інтенсивність дій у часовому інтервалі (наприклад, кількість дій за останні 20 секунд), і у разі перевищення порогу активності — ідентифікує поведінку як потенційно аномальну.

Цей механізм дозволяє:

виявляти автоматизовані або масові несанкціоновані операції з файлами;

сигналізувати про можливий злам акаунту;

підвищити загальний рівень інформаційної безпеки.

Система також передбачає можливість масштабування функціоналу штучного інтелекту шляхом інтеграції класифікаторів, машинного навчання або зовнішніх API для семантичного аналізу вмісту файлів, створення розумного пошуку, виявлення шкідливих вкладів тощо.

Таким чином, запропонована система поєднує:

технологічні рішення сучасної frontend-розробки (React);

надійну та масштабовану хмарну платформу (Firebase);

інтелектуальні алгоритми аналізу поведінки користувача;

захищену багаторівневу архітектуру.

Це забезпечує як функціональність, так і надійність, роблячи систему зручною для кінцевого користувача й перспективною для подальшого розвитку.

2.2. Модель використання системи

Модель використання програмної системи — це структурований опис можливих сценаріїв взаємодії користувача із програмним забезпеченням. Основне призначення цієї моделі — формалізувати основні дії користувача, визначити логіку обробки запитів та дати уявлення про функціональні можливості системи з точки зору кінцевого користувача. [16]

У межах даної дипломної роботи розроблено хмарну систему управління файлами, яка дозволяє зареєстрованому користувачу здійснювати базові та розширені операції з файлами й папками, зберігаючи при цьому дані у Firebase — сучасній хмарній платформі з високим рівнем захисту. Крім того, система має інтегрований інтелектуальний модуль, що здатен аналізувати поведінку користувача у режимі реального часу та сигналізувати про аномалії.

Модель використання охоплює повний життєвий цикл роботи користувача в системі — від моменту реєстрації до завершення сеансу. Опис базується на аналізі функціоналу, реалізованого у системі, а також на потребах цільової аудиторії — користувачів, які потребують безпечного онлайн-доступу до персональних даних і файлів.

Основні ролі:

Користувач: особа, яка взаємодіє із системою через вебінтерфейс. Має право створювати/видаляти/редагувати файли та теки у власному середовищі.

Система: клієнт-серверна частина, що реагує на дії користувача, відображає дані, обробляє запити, передає інформацію до Firebase та, за потреби, активує ШІ-модуль.

Типові сценарії використання:

1. Реєстрація нового користувача

Користувач відкриває вебінтерфейс системи, переходить до форми реєстрації та вводить електронну пошту й пароль. Валідація здійснюється на клієнтському рівні, після чого запит передається до Firebase Authentication. У разі успішної реєстрації створюється унікальний обліковий запис з UID, який надалі використовується для ідентифікації та персоналізації доступу.

2. Авторизація у системі

Після введення логіна і пароля відбувається автентифікація користувача. У разі успіху користувач перенаправляється до особистого файлового середовища, яке завантажується динамічно через запити до Firestore (виведення структури папок і файлів).

3. Перегляд вмісту файлового сховища

Після входу користувач бачить інтерфейс, що візуально нагадує класичний файловий менеджер. Інтерфейс дозволяє переглядати список наявних файлів та тек, переходити по вкладених каталогах. Структура завантажується з Firestore на основі UID користувача.

4. Створення файлу або теки

Користувач натискає кнопку "Create File" або "New Folder", вводить назву нового елементу. Система перевіряє унікальність назви в поточному каталозі. У разі успіху створюється новий об’єкт, який додається до Firestore (у вигляді документа з полями `name`, `type`, `parent`, `uid`, `path`, `timestamp`). Якщо це текстовий файл — автоматично відкривається редактор (CodeMirror).

5. Завантаження файлів

Користувач має змогу завантажувати файли зі свого пристрою. При цьому вміст передається до Firebase Storage, а метадані про файл зберігаються у Firestore. Після завершення завантаження інтерфейс оновлюється, щоб відобразити новий файл.

6. Редагування вмісту файлу

Для створених файлів підтримується редагування. У текстовому редакторі користувач може змінити вміст і зберегти. Система автоматично оновлює поле `data` або `content` у Firestore, і при наступному відкритті користувач бачить оновлений вміст.

7. Видалення файлів або папок

Користувач має можливість видалити будь-який файл або теку, що йому належать. Видалення відбувається на двох рівнях: запис у Firestore видаляється, а сам файл із Storage очищується. Також оновлюється дерево папок у фронтенді.

8. Пошук за назвою файлу

У системі реалізовано базовий пошук за назвою. Пошуковий запит фільтрує масив доступних файлів у поточному каталозі, що дозволяє користувачеві швидко знаходити потрібні документи.

9. Логування дій користувача

Кожна дія користувача (створення, редагування, завантаження, видалення) фіксується у Firestore в колекції `user_activity_logs`, із зазначенням часу, UID користувача, типу дії та об’єкта. Це створює повний журнал активності.

10. Виявлення аномальної активності

Інтегрований інтелектуальний модуль (React-хук `useAnomalyDetector`) постійно аналізує останні 10 дій користувача. Якщо протягом 20 секунд виявлено

≥5 активних дій, система виводить попередження про потенційну аномалію. Це дозволяє запобігти спробам зламу, автоматизованим атакам або масовому знищенню даних.

11. Завершення сеансу

Користувач може натиснути "Log out", після чого відбувається вихід із системи, і доступ до особистого середовища блокується.

Модель використання системи є фундаментальним елементом процесу розробки програмного забезпечення, оскільки вона не лише описує передбачувані сценарії взаємодії користувача з програмою, але й слугує основою для формалізації вимог та подальшого тестування. На практиці це означає, що перед впровадженням будь-якого функціонального модуля розробник має чітке уявлення про очікувану поведінку користувача, про послідовність кроків, які користувач виконує, та про те, яким чином система має на них реагувати.

Окрім того, така модель дає змогу зменшити кількість помилок у майбутньому, бо вона виявляє потенційні неузгодженості ще до початку активної фази розробки. Саме тому в сучасних підходах до створення програмного забезпечення (зокрема, в методологіях Agile, Scrum) моделі використання займають важливе місце вже на етапі проєктування. Ретельно пропрацьовані сценарії дають змогу покрити практично всі шляхи взаємодії користувача з інтерфейсом, що своєю чергою підвищує надійність та ергономіку системи в цілому.

У моїй розробці модель використання дозволяє не лише ідентифікувати ключові точки взаємодії, а й закласти логіку реакції інтелектуальних підсистем — зокрема механізму виявлення підозрілої активності. З огляду на те, що сучасні інформаційні системи дедалі частіше стають мішенню для атак або внутрішніх зловживань, важливо не лише фіксувати дії користувача, а й своєчасно аналізувати їх. Саме тому сценарії використання системи поєднують у собі як класичні функції файлового менеджера (створення, перегляд, редагування, видалення), так і новітні функції аналітики та адаптивної реакції.

Таким чином, модель використання системи, запропонована у цьому підрозділі, є не лише описом логіки взаємодії, а й концептуальною основою, яка забезпечує інтеграцію функціоналу, безпеки та інтелектуального реагування на поведінку користувача.

2.3. Архітектура системи

Архітектура системи — це сукупність взаємопов’язаних структурних компонентів, які реалізують визначені функціональні можливості, забезпечуючи цілісну роботу програмного забезпечення. У межах даної роботи система побудована як односторінковий вебзастосунок (SPA), що працює за моделлю клієнт-серверної архітектури із широким використанням хмарних технологій. [17]

Загальний огляд архітектури

Система умовно поділяється на три основні рівні:

Клієнтський рівень (Frontend):

Реалізований з використанням бібліотеки React.

Забезпечує інтерфейс користувача, обробку подій, валідацію форм.

Відображає файлову структуру, редактор текстових файлів, сповіщення про підозрілу активність.

Використовує React Router для маршрутизації між сторінками (реєстрація, логін, dashboard).

Хмарний рівень (Backend-as-a-Service, BaaS):

Побудований на платформі Firebase.

Складається з:

Firebase Authentication — автентифікація користувачів.

Cloud Firestore — база даних для збереження метаданих, структури файлів, логів дій.

Firebase Storage — фізичне зберігання файлів користувача.

Всі сервіси взаємодіють через API з клієнтською частиною.

Модуль штучного інтелекту:

Представлений у вигляді клієнтського хука useAnomalyDetector.

Аналізує останні дії користувача на основі журналу активності (колекція user_activity_logs у Firestore).

Реагує на аномальну поведінку (наприклад, >5 дій за 20 секунд), формуючи відповідне повідомлення у UI.

Компоненти клієнтської частини

App.jsx — головний контейнер застосунку.

Auth компоненти — Login, Register, Logout.

Dashboard — головна сторінка після входу, відображає структуру файлів.

Компоненти управління файлами — CreateFile, UploadFile, FileList.

Інтелектуальний компонент — SuspiciousActivityAlert, який візуалізує виявлену аномалію.

Redux — використовується для централізованого керування станом (auth, files, folders).

Структура збереження даних у Firestore

FireStore зберігає кілька основних колекцій:

users — інформація про зареєстрованих користувачів.

files — метадані завантажених та створених файлів (назва, тип, шлях, власник, дата).

folders — структура папок користувача.

user_activity_logs — журнал активності користувача, що фіксує кожну дію з міткою часу.

Інтеграція з Firebase

React-проєкт використовує модульну бібліотеку firebase для ініціалізації та взаємодії з бекендом:

js

КопіюватиРедагувати

```
import { getFirestore } from "firebase/firestore";
import { getAuth } from "firebase/auth";
import { getStorage } from "firebase/storage";
```

У файлі конфігурації (firebase.js) створюється підключення на основі .env-змінних, а потім експортуються відповідні сервіси.

Інтелектуальна підсистема виявлення підозрілої активності

Штучний інтелект представлений у вигляді простого евристичного аналізатора, реалізованого через React-хук:

Виконується запит до user_activity_logs, отримується останні 10 дій.

Обчислюється кількість дій за останні 20 секунд.

Якщо їх ≥5, система вважає активність потенційно небезпечною.

Цей підхід дозволяє вбудувати просту модель поведінкової аналітики без необхідності в ML-сервері або зовнішньому API, що робить його оптимальним для навчальних та MVP-проєктів.

Побудова архітектури програмної системи є одним із ключових етапів у процесі її проєктування. Від правильно обраної архітектурної моделі залежить не лише стабільність функціонування, а й масштабованість, зручність супроводу, безпечність та загальна ефективність майбутнього застосунку. Особливої актуальності ці аспекти набувають у разі створення вебзастосунку, що має працювати у хмарному середовищі та взаємодіяти з великою кількістю користувачів.

В умовах сучасного IT-середовища популярність набуває саме клієнт-серверна архітектура, у межах якої обробка даних та логіка взаємодії розподілені між двома сторонами: фронтендом (клієнтом) і бекендом (сервером). У контексті даної роботи роль сервера виконує хмарна платформа Firebase, що надає набір готових сервісів, зокрема автентифікацію, базу даних у реальному часі та файлове сховище. Такий підхід дає змогу уникнути розробки складної серверної логіки та зосередити зусилля на створенні функціонального та зручного інтерфейсу користувача. [18]

Застосування React як основної бібліотеки для побудови інтерфейсу забезпечує високий рівень інтерактивності, підтримку компонентного підходу та ефективну роботу з віртуальним DOM. Це дозволяє значно підвищити

продуктивність, а також зменшити навантаження на мережу, оскільки дані завантажуються частинами, відповідно до активності користувача. [19]

Інтеграція модуля штучного інтелекту у вигляді поведінкового аналізатора є важливим етапом побудови сучасної безпечної інформаційної системи. Незважаючи на простоту реалізації (на основі евристики), цей підхід ілюструє можливість вбудованого моніторингу активності користувача, що є ефективним методом виявлення потенційних атак або зловживань обліковими записами. Крім того, ця підсистема повністю автономна, не потребує підключення до зовнішніх моделей машинного навчання чи складних нейронних мереж, що спрощує її реалізацію в рамках навчального проєкту.

У результаті, побудована архітектура відзначається збалансованістю: поєднання сучасних вебтехнологій, хмарних сервісів та інтелектуального аналізу дозволяє створити стабільну, адаптивну та безпечну систему, яка може масштабуватись і бути основою для подальших удосконалень. Такий підхід повністю відповідає сучасним вимогам до створення захищених вебзастосунків у сфері управління інформацією.

2.4. Алгоритм роботи системи

Алгоритм роботи системи — це формалізований покроковий опис дій, що виконуються програмним забезпеченням під час взаємодії з користувачем. Він дає змогу побудувати чітке розуміння структури процесів, що відбуваються в системі, визначити основні етапи життєвого циклу сесії користувача, а також забезпечити єдину логіку обробки дій та подій у межах усіх модулів застосунку.[20]

У запропонованій системі реалізовано типову для вебзастосунків модель “клієнт — хмара”, де клієнтська частина (React-застосунок) відповідає за інтерфейс користувача, а хмарна інфраструктура Firebase — за зберігання даних, автентифікацію та управління файлами. Така архітектура дозволяє створити адаптивну, масштабовану та безпечну систему з мінімальним навантаженням на локальні ресурси користувача.

Загальний принцип роботи системи

Основна логіка базується на взаємодії користувача з візуальним інтерфейсом, у межах якого реалізовано всі дії: реєстрація, робота з файлами, завантаження, редагування, видалення тощо. Усі дані зберігаються у хмарі, а передача інформації відбувається за допомогою API-запитів до Firestore та Firebase Storage.

Особливістю реалізації є також вбудований модуль аналізу дій користувача, який умовно можна віднести до підсистеми штучного інтелекту. Цей модуль здійснює базовий аналіз активності та сигналізує про потенційно аномальну поведінку, підвищуючи тим самим загальний рівень захищеності системи.

Покроковий алгоритм роботи системи

Крок 1. Ініціалізація системи

Після запуску користувач відкриває сторінку застосунку.

Ініціалізуються основні сервіси Firebase: автентифікація (Auth), база даних (Firestore), хмарне сховище (Storage).

Компонент App.jsx визначає, чи користувач уже авторизований.

Крок 2. Авторизація/реєстрація

Якщо користувач неавторизований — він потрапляє на сторінку входу або створення облікового запису.

Усі дані перевіряються на клієнтському рівні, після чого передаються до Firebase Authentication.

У разі успішного входу — користувач перенаправляється на основну сторінку системи (Dashboard).

Крок 3. Завантаження персонального середовища

Система отримує унікальний UID користувача.

На основі UID виконується запит до Firestore, де зберігаються метадані файлів та структуровані дані тек.

Створюється динамічний інтерфейс файлової системи, який відображає доступний вміст.

Крок 4. Робота з файлами

Користувач може:

створити текстовий файл;

створити теку;

завантажити файл з локального комп’ютера;

переглянути, відредагувати або видалити об’єкти.

Залежно від типу дії, інформація записується до Firestore (структура, текст, метадані) або до Storage (бінарні файли).

Крок 5. Логування дій користувача

Після кожної дії (створення, редагування, завантаження, видалення) система виконує додаткову дію: запис у колекцію user_activity_logs.

Запис містить: UID користувача, тип дії, об’єкт дії, час.

Це формує повноцінний журнал подій — важливий компонент як для безпеки, так і для аналітики.

Крок 6. Аналіз активності користувача (ШІ-модуль)

Паралельно з роботою інтерфейсу виконується React-хук useAnomalyDetector.

Він періодично виконує запит до Firestore, щоб отримати останні 10 дій користувача.

Якщо ≥5 дій відбулися протягом 20 секунд — система вважає це аномальною поведінкою.

У такому разі на інтерфейсі з'являється попередження для користувача.

Примітка: хоча реалізація побудована на простій евристиці, модуль аналізу активності демонструє принципи адаптивної поведінкової безпеки без використання складних ML-алгоритмів.

Крок 7. Оновлення інтерфейсу в реальному часі

Усі зміни у файловій структурі автоматично оновлюються завдяки onSnapshot() від Firestore.

Користувач бачить зміни без перезавантаження сторінки.

Це забезпечує плавну та сучасну взаємодію з системою.

Крок 8. Вихід із системи

Після завершення роботи користувач натискає «Вийти».

Система викликає функцію signOut() з Firebase Auth.

Сесія очищується, доступ до системи закривається, і користувач перенаправляється на форму входу.

Алгоритм роботи системи охоплює повний цикл взаємодії з користувачем — від початкового входу до логування дій та реагування на потенційні ризики. Завдяки покроковій логіці та інтеграції з хмарними технологіями досягнуто ефективну, динамічну й адаптивну систему, яка відповідає вимогам сучасних підходів до розробки інформаційних вебзастосунків.

Рисунок 2.1 – Блок-схема алгоритму роботи

Представлена блок-схема ілюструє основні етапи роботи користувача з хмарною системою управління файлами — від моменту входу в систему до завершення сесії. Візуалізований алгоритм дозволяє наочно показати послідовність

дій, логіку обробки запитів, інтеграцію з Firebase та активацію модуля виявлення аномалій. Завдяки такій структурі забезпечується зручна взаємодія, швидке реагування інтерфейсу та базовий рівень поведінкової безпеки. Блок-схема відображає цілісність архітектури та демонструє збалансований підхід до реалізації функціональності й захисту системи.

2.5. Висновки до розділу

У даному розділі було здійснено проєктування програмної системи, яка забезпечує захищену віддалену роботу користувача з хмарним сховищем. Розглянуто загальну характеристику програмного забезпечення, визначено ключові компоненти, модулі та логіку взаємодії системи з користувачем. Особливу увагу приділено архітектурному підходу до побудови системи, що базується на використанні бібліотеки React на клієнтському рівні та хмарної платформи Firebase як backend-рішення.

Ретельно проаналізовано модель використання системи, у якій описано типові сценарії взаємодії користувача з інтерфейсом: від реєстрації та входу до створення, редагування, збереження та видалення файлів. Структурована модель забезпечує зручність, швидкодію та прозорість у роботі з даними. Вбудована підтримка логування активності дозволяє здійснювати контроль за діями користувача, а модуль виявлення аномалій — підвищити безпеку системи.

Алгоритм функціонування системи охоплює основні етапи життєвого циклу взаємодії користувача із застосунком. Він демонструє, яким чином реалізується безперервна обробка подій, динамічне оновлення інтерфейсу та перевірка поведінкової активності. Представлена блок-схема чітко ілюструє ключові кроки функціонування — від входу користувача до фіксації його дій у системі.

Таким чином, розроблений проєкт є логічно завершеною та структурованою основою для подальшої реалізації програмного продукту, що відповідає сучасним вимогам до безпеки, масштабованості та інтерактивності.

3. РЕАЛІЗАЦІЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

На етапі реалізації програмного забезпечення всі теоретичні напрацювання, проєктні рішення та алгоритмічні моделі, розроблені у попередніх розділах, набувають практичного втілення. Саме тут формується безпосередньо програмний код, конфігуруються зовнішні сервіси, створюються інтерфейси користувача та впроваджуються допоміжні функції, необхідні для повноцінного функціонування системи. Реалізація є критично важливою фазою, яка поєднує концепцію, архітектуру та функціональність в єдину діючу структуру.

Зважаючи на актуальні технологічні вимоги, система була реалізована як вебзастосунок, що працює в середовищі браузера та не потребує встановлення на комп’ютер користувача. Це забезпечує гнучкість у використанні, мультиплатформеність, а також спрощує оновлення й масштабування системи.

3.1. Загальна характеристика середовища реалізації

Реалізація програмного забезпечення є одним із найвідповідальніших етапів у життєвому циклі створення інформаційної системи. Саме на цьому етапі усі теоретичні напрацювання, структурні та функціональні моделі, що були сформовані під час аналізу та проєктування, знаходять своє втілення у вигляді реального програмного продукту. Ефективна реалізація значною мірою залежить від правильно обраного середовища розробки, технологічного стека, мов програмування, бібліотек і платформ. У межах цієї дипломної роботи було прийнято рішення використати сучасний, популярний та широко підтримуваний стек технологій, зокрема JavaScript (JS), React, Firebase та хмарні сервіси Google, у поєднанні з гнучким середовищем розробки Visual Studio Code.

Основною мовою програмування застосунку є JavaScript — високорівнева мова, що підтримує об’єктно-орієнтоване, імперативне та функціональне програмування. У контексті розробки вебзастосунків JavaScript займає провідне місце, адже вона є нативною для браузерів, має величезну екосистему бібліотек і фреймворків, постійно оновлюється та вдосконалюється. Сучасна версія мови —

ECMAScript 6 (ES6) — надає зручний синтаксис, підтримку модулів, асинхронну обробку подій, роботу з класами та багато іншого, що дозволяє писати чистий, читабельний і масштабований код.[21]

Для побудови інтерфейсу користувача було обрано React — відкриту JavaScript-бібліотеку, розроблену компанією Meta (раніше Facebook). React дозволяє створювати динамічні та інтерактивні вебінтерфейси з використанням компонентоорієнтованого підходу. У межах цієї парадигми вся система розбивається на окремі компоненти, кожен з яких відповідає за конкретну частину інтерфейсу або логіки. Така структура не лише покращує підтримку коду, а й забезпечує високу гнучкість при подальшій модифікації або масштабуванні застосунку. React також забезпечує ефективне оновлення інтерфейсу завдяки використанню віртуального DOM (Virtual DOM), що дозволяє значно зменшити кількість непотрібних перерендерів і, відповідно, підвищити продуктивність програми. [22]

Середовище розробки, яке використовувалося у проєкті, — це Visual Studio Code (VS Code) — один із найпопулярніших сучасних редакторів коду з відкритим кодом, який поєднує в собі легкість, багатофункціональність та підтримку практично всіх мов програмування. Visual Studio Code надає інтегровану підтримку Git, можливість встановлення тисяч розширень, таких як ESLint, Prettier, Firebase Tools, React Snippets, Tailwind CSS IntelliSense та інші, що значно полегшують роботу розробника. Завдяки підтримці автоматичного доповнення коду (IntelliSense), підсвічуванню синтаксису, інтегрованому терміналу та зручній навігації VS Code сприяє підвищенню продуктивності програміста та мінімізації помилок на етапі написання коду.[23]

Для реалізації серверної частини та зберігання даних застосунку не використовувались окремі сервери або фізичні бази даних, натомість було обрано хмарну платформу Firebase, яка надає повний спектр послуг для розробників: автентифікація, база даних, сховище файлів, хостинг тощо. Firebase — це набір інструментів від компанії Google, орієнтований на швидку розробку веб- і

мобільних застосунків. Його головними перевагами є легкість інтеграції, масштабованість, безпека та оновлення даних у реальному часі.

У даному застосунку були використані такі сервіси Firebase:

Firebase Authentication — для створення, аутентифікації та управління обліковими записами користувачів. Забезпечує безпечний вхід у систему, зберігання токенів, а також підтримку багатьох методів автентифікації.

Cloud Firestore — документоорієнтована база даних, яка дозволяє зберігати структуровану інформацію про користувачів, файли, дії та структуру каталогів. Firestore має гнучку модель запитів, автоматичну синхронізацію даних між клієнтами та високу швидкодію.

Firebase Storage — хмарне сховище для зберігання бінарних даних, зокрема файлів, які завантажуються користувачами. Платформа гарантує захист доступу до даних, використання правил безпеки (Security Rules) і можливість масштабування з урахуванням навантаження.

Проєкт також використовує менеджер пакетів pnpm, який забезпечує ефективне керування залежностями, оптимізоване зберігання модулів та пришвидшене встановлення. Його перевага — економія дискового простору та уникнення дублювання пакетів у проєктах.

Крім основного функціоналу, до системи був інтегрований простий модуль виявлення аномальної активності користувача, який умовно можна віднести до елементів штучного інтелекту. Цей модуль побудований на основі простих евристичних правил: система аналізує журнал дій користувача у Firestore і, у випадку виявлення частих дій за короткий час, ідентифікує таку поведінку як потенційно небезпечну. Хоча модуль не використовує складних алгоритмів машинного навчання, він ілюструє базові принципи поведінкової безпеки та адаптивного реагування системи на підозрілі дії.

Таким чином, обране середовище розробки — це гармонійне поєднання сучасних вебтехнологій, хмарних сервісів та гнучких засобів програмування. Використання React, Firebase, JavaScript і Visual Studio Code забезпечило реалізацію усіх функціональних вимог, швидкість розробки, адаптивність,

зручність користувача та базовий рівень безпеки, що відповідає тематиці дипломної роботи.

3.2. Структура програмного забезпечення

Структура програмного забезпечення є фундаментальним аспектом усього застосунку, адже саме вона визначає, яким чином буде організовано код, де і як будуть зберігатись компоненти, логіка взаємодії, обробка подій та робота з базою даних. Чітко спроєктована структура значно спрощує процес розробки, дозволяє розподіляти обов'язки між модулями, підвищує читабельність коду та полегшує майбутню підтримку та масштабування системи.

У межах реалізації захищеного хмарного сховища було побудовано вебзастосунок, організований за компонентним принципом. Це означає, що система розділена на окремі логічні блоки, кожен з яких відповідає за свою частину інтерфейсу або функціональності. Такий підхід є типовим для React-додатків і забезпечує модульність, незалежність і повторне використання коду.

Після ініціалізації проєкту структура директорій виглядає наступним чином:

```
react-firebase-file-management-system/
├── public/
│   └── index.html
├── src/
│   ├── components/
│   │   ├── FileList.jsx
│   │   ├── FileUploader.jsx
│   │   ├── FileViewer.jsx
│   │   └── AnomalyWarning.jsx
│   ├── pages/
│   │   ├── Dashboard.jsx
│   │   ├── Login.jsx
│   │   └── Register.jsx
│   ├── hooks/
│   │   └── useAnomalyDetector.js
│   ├── firebase/
│   │   └── config.js
│   ├── App.jsx
│   └── index.jsx
├── .env
├── package.json
└── tailwind.config.js
```

Основні елементи структури:

Папка src/ — основна директорія, в якій зберігається увесь вихідний код програми. Саме тут розміщено компоненти, сторінки, конфігурації Firebase, хуки та стилі.

components/ — містить багаторазові функціональні компоненти, які відповідають за окремі частини інтерфейсу. Наприклад, FileUploader.jsx реалізує логіку завантаження файлів у Firebase Storage, а FileList.jsx — відображення списку доступних файлів. Усі ці компоненти використовуються на головній сторінці (Dashboard) та взаємодіють між собою.

pages/ — включає в себе сторінки, які відповідають за маршрути користувача в застосунку. Зокрема, Login.jsx і Register.jsx реалізують функції автентифікації, а Dashboard.jsx — головний робочий інтерфейс користувача після входу.

hooks/ — директорія для розміщення власних React-хуків. Один із них — useAnomalyDetector.js — реалізує базовий механізм аналізу поведінки користувача для виявлення аномальної активності.

firebase/ — окрема директорія, у якій зберігаються файли для ініціалізації та конфігурації Firebase. Зокрема, config.js містить SDK-ключі, об'єкти налаштувань та функції підключення до бази даних, авторизації й сховища.

App.jsx — головний компонент React, що реалізує логіку маршрутизації, визначає структуру інтерфейсу й відображає основні сторінки залежно від шляху доступу.

index.jsx — точка входу у застосунок. Саме тут відбувається ініціалізація React-додатку, підключення Tailwind CSS, Firebase та рендер у DOM через компонент <App />.

package.json — файл керування залежностями, у якому вказані всі бібліотеки, що використовуються у проєкті. Також містить скрипти для запуску, компіляції та деплою застосунку.

.env — файл середовища, де зберігаються конфіденційні дані, такі як ключі доступу до Firebase. Він не включається до публічних репозиторіїв для збереження безпеки.

Архітектурна організація

Такий підхід до структурування коду дозволяє легко масштабувати проєкт, додаючи нові сторінки або компоненти. Наприклад, при додаванні нової функції, наприклад пошуку файлів за вмістом, достатньо створити новий компонент у components/ і підключити його у відповідному місці в Dashboard.jsx. Це робить систему гнучкою й розширюваною.

Структура підтримує односторінкову архітектуру, у межах якої перехід між сторінками не призводить до повного перезавантаження сторінки, а забезпечується швидке перемальовування компонентів через React Router.

Також завдяки використанню хуків та Context API можливо централізовано зберігати й обробляти стан застосунку, що зменшує дублювання коду та підвищує зручність у підтримці проєкту.

Таким чином, структура програмного забезпечення, розроблена у цьому дипломному проєкті, є логічно організованою, чіткою та повністю відповідає стандартам сучасної веброзробки. Вона базується на модульному підході, що дозволяє не лише реалізувати необхідну функціональність, але й забезпечити легкість у масштабуванні, повторному використанні компонентів і внесенні змін у логіку програми без значних затрат часу та ресурсів.

3.3. Основні компоненти інтерфейсу користувача

Інтерфейс користувача (UI — User Interface) є одним із ключових елементів будь-якого програмного забезпечення, особливо коли йдеться про вебзастосунки, що взаємодіють із кінцевим користувачем. Саме інтерфейс забезпечує зручний доступ до функціональних можливостей програми, створює перше враження про систему та формує загальну якість взаємодії між користувачем і програмним забезпеченням. У рамках реалізованого вебзастосунку для захищеної роботи в хмарному сховищі інтерфейс був реалізований на основі бібліотеки React, що дозволило забезпечити сучасний, швидкий та інтуїтивно зрозумілий користувацький досвід. [24]

Уся структура інтерфейсу базується на модульному підході: окремі компоненти реалізують різні частини функціональності, що спрощує розробку, тестування й масштабування. Нижче розглянемо основні з них.

1. Сторінка автентифікації (Login.jsx)

Цей компонент забезпечує вхід користувача до системи. Візуально це форма, яка містить поля для введення email та паролю, а також кнопки для входу та переходу до сторінки реєстрації.

З технічної точки зору, Login.jsx реалізує обробку подій введення та надсилання даних до Firebase Authentication. У разі правильного введення даних користувач отримує доступ до сторінки з особистим файловим сховищем. У випадку помилки (наприклад, невірний пароль чи відсутній обліковий запис) користувач отримує інформативне повідомлення.

2. Сторінка реєстрації (Register.jsx)

Компонент `Register.jsx` реалізує логіку створення нового облікового запису. Структура сторінки схожа на Login: поля для введення email, паролю та підтвердження паролю, а також кнопка для реєстрації.

Цей компонент використовує Firebase Authentication для створення нового користувача в базі. Після успішної реєстрації користувач автоматично авторизується та перенаправляється на сторінку Dashboard.

3. Головна сторінка користувача (Dashboard.jsx)

`Dashboard.jsx` — це центральна сторінка вебзастосунку, яка відкривається після авторизації. Саме тут зосереджено основний функціонал:

Завантаження файлів у хмарне сховище через компонент `FileUploader`.

Відображення списку файлів, збережених користувачем, через `FileList`.

Можливість перегляду або завантаження файлів.

Інтерфейс для виявлення підозрілої активності**:** при фіксації частих дій протягом короткого часу система виводить повідомлення про потенційну аномалію через `AnomalyWarning`.

Інтерфейс компонента Dashboard організовано у вигляді зручного, адаптивного макету з використанням CSS-бібліотеки Tailwind, що дозволяє створювати чистий і приємний візуальний стиль без зайвої верстки.

4. Завантаження файлів (FileUploader.jsx)

Компонент `FileUploader` відповідає за прийом файлів від користувача, відображення прогресу завантаження та передачу даних до Firebase Storage. Система обробляє як випадки успішного завантаження, так і помилки (перевищено розмір файлу, відсутній файл тощо). Завантажений файл записується до бази метаданих (Firestore) для подальшого відображення в інтерфейсі.

5. Перелік файлів (FileList.jsx)

`FileList` відповідає за виведення списку усіх файлів, що належать поточному користувачу. Інформація про файли береться з Firestore, а дані регулярно оновлюються в реальному часі через підписку (onSnapshot). Кожен файл має можливість перегляду або завантаження.

6. Попередження про аномалію (AnomalyWarning.jsx)

Цей компонент відображає просте візуальне повідомлення, якщо система за допомогою `useAnomalyDetector` фіксує надмірну активність (наприклад, багато завантажень за короткий час). Це елемент базового аналізу поведінки, що виконує роль індикатора потенційної загрози.

7. Навігація та маршрутизація

Завдяки використанню `React Router` застосунок підтримує маршрутизацію між сторінками:

`/login` — вхід до системи

`/register` — реєстрація нового користувача

`/dashboard` — головна сторінка роботи з файлами

Навігація відбувається без перезавантаження сторінки, що забезпечує плавний користувацький досвід.

Загальний вигляд і UX

Інтерфейс користувача розроблено з урахуванням принципів доступності, простоти та зручності використання. Використання Tailwind CSS дозволило

дотримуватись єдиного стилю в усіх компонентах, а адаптивна верстка — забезпечити коректне відображення програми на різних розмірах екранів.

Кольорова гама інтерфейсу нейтральна, кнопки мають чіткі позначення, повідомлення про помилки відображаються у зрозумілому вигляді. Інтерфейс витримано у мінімалістичному стилі — без зайвих деталей, що відволікають, що відповідає вимогам до вебзастосунків для безпечної роботи з файлами.

3.4. Взаємодія з Firebase

Firebase — це потужна хмарна платформа від компанії Google, яка надає широкий спектр інструментів для розробки веб- та мобільних застосунків. У межах дипломного проєкту Firebase виконує ключову роль як бекенд-рішення, що забезпечує автентифікацію користувачів, зберігання даних і файлів, моніторинг активності та базові можливості для реалізації системи штучного інтелекту, зокрема виявлення аномальної поведінки користувачів. [25]

Завдяки інтеграції з Firebase вдалося реалізувати безсерверну архітектуру, де обробка даних, управління користувачами та доступ до файлів відбувається без розгортання власного серверного оточення. Це значно спрощує підтримку застосунку, зменшує витрати на інфраструктуру та пришвидшує процес розробки.

Основні сервіси Firebase, використані у проєкті

1. Firebase Authentication

Firebase Authentication відповідає за автентифікацію користувачів за допомогою email і паролю. Цей сервіс спрощує створення, перевірку та збереження облікових записів користувачів. Користувачі можуть зареєструватися, увійти у свій акаунт і безпечно працювати зі своїм файловим середовищем.

Інтеграція у React-застосунок реалізується за допомогою Firebase SDK:

```
import { getAuth, createUserWithEmailAndPassword, signInWithEmailAndPassword } from "firebase/auth";
```

Усі дії супроводжуються перевіркою облікових даних, обробкою помилок та збереженням сесії.

2. Firebase Cloud Firestore

Firestore — це NoSQL база даних у реальному часі, яка використовується для зберігання метаданих про файли користувача, інформацію про його дії, ієрархію тек та інші структуровані дані. Зокрема, сюди записуються:

Ім’я файлу

Шлях у файловій структурі

UID користувача

Дата і час завантаження

Тип дії (upload, download тощо)

Firestore забезпечує автоматичну синхронізацію між базою даних і клієнтом, що дозволяє виводити змінені дані в інтерфейсі користувача в режимі реального часу:

```
onSnapshot(query, (snapshot) => {
  snapshot.docs.forEach(doc => {
    // обробка документа
  });
});
```

Ця база даних є ключовим елементом для роботи з системою виявлення аномальної активності: в неї записується кожна дія користувача, а React-хук `useAnomalyDetector` вивчає останні записи для оцінки підозрілої поведінки.

3. Firebase Storage

Це хмарне сховище для зберігання файлів користувача — документів, зображень, архівів тощо. У рамках проєкту файли завантажуються у Firebase Storage через компонент `FileUploader`, а метадані записуються у Firestore. Завантаження відбувається асинхронно з підтримкою прогресу та обробкою помилок:

```
import { getStorage, ref, uploadBytesResumable } from "firebase/storage";
```

Кожен файл зберігається у структурі, що відповідає UID користувача, що унеможливлює доступ до чужих даних.

4. Firebase Security Rules

Щоб забезпечити конфіденційність та захист файлів і даних, у проєкті реалізовані **правила безпеки** (Security Rules), які визначають, хто і які дії може виконувати. Наприклад, правила для Firestore:

```
match /user_activity_logs/{docId} {
  allow read, write: if request.auth.uid == resource.data.uid;
}
```

А для Firebase Storage:

```
match /{userId}/{allPaths=**} {
  allow read, write: if request.auth.uid == userId;
}
```

Такі правила не дозволяють неавторизованим користувачам переглядати чи змінювати чужі дані.

Реалізація підключення Firebase у проєкті

Файл `src/firebase/config.js` містить конфігурацію Firebase, отриману з консолі проєкту. Він ініціалізує застосунок і експортує об’єкти `auth`, `db`, `storage` для подальшого використання в компонентах:

```
import { initializeApp } from "firebase/app";
import { getAuth } from "firebase/auth";
import { getFirestore } from "firebase/firestore";
import { getStorage } from "firebase/storage";

const firebaseConfig = {
  apiKey: process.env.REACT_APP_FIREBASE_API_KEY,
  authDomain: "...",
  projectId: "...",
  storageBucket: "...",
  messagingSenderId: "...",
  appId: "..."
};

const app = initializeApp(firebaseConfig);

export const auth = getAuth(app);
export const db = getFirestore(app);
export const storage = getStorage(app);
```

Завдяки використанню змінних середовища (`.env`) критичні дані не зберігаються у відкритому коді.

Використання Firebase дозволило реалізувати надійну, масштабовану та захищену хмарну архітектуру без потреби в окремому сервері. Платформа забезпечила не лише просту автентифікацію та зберігання даних, а й можливість реалізації базового аналізу поведінки користувача. Інтеграція Firebase з React дозволила досягти високої швидкості розробки та зручності підтримки коду, що позитивно впливає на загальну якість проєкту.

3.5. Реєстрація, авторизація та управління файлами користувача

Однією з основних функціональних складових розробленої системи є механізм автентифікації користувача, а також подальша робота з файлами — завантаження, перегляд, створення папок. Саме ці процеси забезпечують персоналізовану, безпечну й ефективну взаємодію користувача із системою.

Для реалізації автентифікації та управління даними було використано Firebase Authentication, Firebase Firestore та Firebase Storage. Комбінація цих інструментів забезпечує не лише надійне зберігання та доступ до файлів, а й можливість побудови ієрархії, логування дій користувача та розширення функціональності в майбутньому.

Реєстрація нового користувача

Реєстрація реалізована в окремому компоненті `Register.jsx`. Користувач заповнює форму з полями email та password, після чого натискає кнопку для реєстрації. Після обробки запиту у Firebase створюється новий обліковий запис.

```
import { createUserWithEmailAndPassword } from "firebase/auth";
import { auth } from "../firebase/config";

const handleRegister = async (e) => {
  e.preventDefault();
  try {
    await createUserWithEmailAndPassword(auth, email, password);
    navigate("/dashboard");
  } catch (error) {
    alert("Помилка реєстрації: " + error.message);
```

```
  }
};
```

У разі успішної реєстрації користувача автоматично переадресовують на головну сторінку — Dashboard.

Авторизація (вхід до системи)

Компонент `Login.jsx` відповідає за вхід користувача через введення email та пароля. Якщо дані вірні, користувач потрапляє до інтерфейсу файлового сховища.

```
import { signInWithEmailAndPassword } from "firebase/auth";

const handleLogin = async (e) => {
  e.preventDefault();
  try {
    await signInWithEmailAndPassword(auth, email, password);
    navigate("/dashboard");
  } catch (error) {
    alert("Помилка входу: " + error.message);
  }
};
```

Firebase автоматично зберігає сесію користувача, тому після оновлення сторінки він залишається авторизованим.

Завантаження файлів у хмарне сховище

Користувач може завантажувати файли у свою власну область у Firebase Storage. Це реалізовано через компонент `FileUploader.jsx`.

```
import { getStorage, ref, uploadBytesResumable, getDownloadURL } from "firebase/storage";
import { addDoc, serverTimestamp } from "firebase/firestore";

const handleUpload = async (file) => {
  const storageRef = ref(storage, `${user.uid}/${file.name}`);
  const uploadTask = uploadBytesResumable(storageRef, file);

  uploadTask.on("state_changed", null, console.error, async () => {
    const url = await getDownloadURL(uploadTask.snapshot.ref);
    await addDoc(collection(db, "files"), {
      name: file.name,
      url,
      uid: user.uid,
      timestamp: serverTimestamp()
    });
  });
};
```

Після завантаження файл потрапляє до Firebase Storage, а у Firestore створюється запис із метаданими (назва, посилання, uid, час).

Створення папок

Firebase не має вбудованої підтримки папок, однак ієрархію можна створити логічно, додаючи у Firestore записи типу "папка" і використовуючи ієрархічний шлях у Storage:

```
await addDoc(collection(db, "folders"), {
  name: folderName,
  path: `${user.uid}/${folderName}`,
  uid: user.uid,
  createdAt: serverTimestamp()
});
```

Відображення списку файлів

Компонент `FileList.jsx` підключається до Firestore й отримує всі файли користувача через підписку:

```
const q = query(
  collection(db, "files"),
  where("uid", "==", user.uid),
  orderBy("timestamp", "desc")
);
```

Файли автоматично оновлюються у реальному часі.

Видалення файлів

Користувач може видалити файл, і тоді він зникає як із бази даних, так і з Firebase Storage:

```
await deleteObject(ref(storage, `${user.uid}/${file.name}`));
await deleteDoc(doc(db, "files", fileId));
```

Інтеграція ШІ (виявлення підозрілої активності)

Система виявлення підозрілої активності реалізована як кастомний хук `useAnomalyDetector`, що аналізує останні дії користувача у Firestore:

```
const q = query(
  collection(db, "user_activity_logs"),
  where("uid", "==", user.uid),
  orderBy("timestamp", "desc"),
  limit(10)
```

```
);
```

Якщо протягом короткого часу було багато дій (наприклад, 5 за 20 секунд), програма вважає це потенційною аномалією.

Обробка помилок та безпека

Для захисту від несанкціонованого доступу використовуються правила безпеки Firebase:

```
match /files/{fileId} {
  allow read, write: if request.auth.uid == resource.data.uid;
}
```

Це гарантує, що лише власник може взаємодіяти зі своїми файлами.

Реалізація інтерфейсу користувача

Інтерфейс користувача реалізований за допомогою бібліотеки React. Структура інтерфейсу побудована компонентно, що забезпечує простоту масштабування та повторне використання коду. Кожен компонент виконує окрему функцію: реєстрація, авторизація, завантаження файлів, створення папок, вивід списку файлів тощо.

Основна структура інтерфейсу (`App.jsx`)

```
import React from "react";
import { BrowserRouter as Router, Routes, Route } from "react-router-dom";
import Login from "./pages/Login";
import Register from "./pages/Register";
import Dashboard from "./pages/Dashboard";

const App = () => {
  return (
    <Router>
      <Routes>
        <Route path="/" element={<Login />} />
        <Route path="/register" element={<Register />} />
        <Route path="/dashboard" element={<Dashboard />} />
      </Routes>
    </Router>
  );
};

export default App;
```

Цей компонент визначає маршрутизацію в межах програми.

Компонент Dashboard.jsx — головна сторінка користувача

```
import React from "react";
import FileUploader from "../components/FileUploader";
import FileList from "../components/FileList";
import FolderCreator from "../components/FolderCreator";

const Dashboard = () => {
 return (
  <div className="dashboard-container">
   <h2>Хмарне сховище</h2>
   <FolderCreator />
   <FileUploader />
   <FileList />
  </div>
 );
};

export default Dashboard;
```

Цей компонент об’єднує три головні частини: створення папок, завантаження файлів і список файлів.

Компонент FolderCreator.jsx — створення нової папки

```
import React, { useState } from "react";
import { db } from "../firebase/config";
import { addDoc, collection, serverTimestamp } from "firebase/firestore";
import { getAuth } from "firebase/auth";

const FolderCreator = () => {
 const [folderName, setFolderName] = useState("");
 const user = getAuth().currentUser;

 const handleCreateFolder = async () => {
  if (!folderName.trim()) return;
  await addDoc(collection(db, "folders"), {
   name: folderName,
   path: `${user.uid}/${folderName}`,
   uid: user.uid,
   createdAt: serverTimestamp(),
  });
  setFolderName("");
 };

 return (
  <div>
```

```
      <input type="text" value={folderName} onChange={(e) => setFolderName(e.target.value)}
placeholder="Назва папки" />
      <button onClick={handleCreateFolder}>Створити папку</button>
    </div>
  );
};

export default FolderCreator;
```

Цей код дозволяє створювати нові папки в системі з логічною ієрархією у Firebase.

Компонент FileUploader.jsx — завантаження файлів

```
import React from "react";
import { storage, db } from "../firebase/config";
import { ref, uploadBytesResumable, getDownloadURL } from "firebase/storage";
import { addDoc, collection, serverTimestamp } from "firebase/firestore";
import { getAuth } from "firebase/auth";

const FileUploader = () => {
  const user = getAuth().currentUser;

  const handleUpload = async (file) => {
    const storageRef = ref(storage, `${user.uid}/${file.name}`);
    const uploadTask = uploadBytesResumable(storageRef, file);

    uploadTask.on("state_changed", null, console.error, async () => {
      const downloadURL = await getDownloadURL(uploadTask.snapshot.ref);
      await addDoc(collection(db, "files"), {
        name: file.name,
        url: downloadURL,
        uid: user.uid,
        timestamp: serverTimestamp(),
      });
    });
  };

  return (
    <input type="file" onChange={(e) => handleUpload(e.target.files[0])} />
  );
};

export default FileUploader;
```

Компонент FileList.jsx — список і управління файлами

```
import React, { useEffect, useState } from "react";
import { db, storage } from "../firebase/config";
```

```
import { collection, query, where, onSnapshot, orderBy } from "firebase/firestore";
import { getAuth } from "firebase/auth";
import { deleteObject, ref } from "firebase/storage";
import { deleteDoc, doc } from "firebase/firestore";

const FileList = () => {
  const [files, setFiles] = useState([]);
  const user = getAuth().currentUser;

  useEffect(() => {
    const q = query(
      collection(db, "files"),
      where("uid", "==", user.uid),
      orderBy("timestamp", "desc")
    );

    const unsubscribe = onSnapshot(q, (snapshot) => {
      setFiles(snapshot.docs.map((doc) => ({ id: doc.id, ...doc.data() })));
    });

    return () => unsubscribe();
  }, [user.uid]);

  const handleDelete = async (file) => {
    await deleteObject(ref(storage, `${user.uid}/${file.name}`));
    await deleteDoc(doc(db, "files", file.id));
  };


export default FileList;
```

Цей компонент дозволяє переглядати, відкривати і видаляти файли без перезавантаження сторінки.

Таким чином, реалізований комплекс функцій — від реєстрації та авторизації до повного управління файлами, гнучкого інтерфейсу та виявлення підозрілої активності — дозволяє створити зручне, безпечне середовище для роботи з даними користувача у хмарному сховищі.

3.6. Приклади роботи програми

Щоб проілюструвати роботу розробленої системи, у цьому підрозділі наведено приклади взаємодії користувача з інтерфейсом програми. Описано

ключові дії, які може виконувати користувач після авторизації, а також поведінку окремих компонентів під час роботи з хмарним середовищем.

Розроблений застосунок має простий та інтуїтивно зрозумілий інтерфейс, реалізований за допомогою бібліотеки React, що забезпечує швидке оновлення інтерфейсу, гнучкість та високу реактивність. Завдяки поєднанню з Firebase користувачі можуть взаємодіяти з хмарним сховищем у режимі реального часу.\

На зображенні представлено розділ **Authentication** у Firebase Console, який відображає зареєстрованих користувачів системи. Тут видно облікові записи користувачів, що успішно створені через інтерфейс програми. Для кожного користувача вказано:

Email (Identifier) — унікальний ідентифікатор (адреса електронної пошти);

Provider — спосіб автентифікації (в даному випадку — email+password);

Дата створення облікового запису;

Останній вхід;

User UID — унікальний ідентифікатор користувача, що використовується для зв’язку з іншими сервісами (Firestore, Storage).

Рисунок 3.1 – Вікно **Authentication** у Firebase Console

Ця панель є важливим інструментом адміністратора, який дозволяє перевіряти активність, створювати нових користувачів вручну або, за потреби, блокувати доступ. У межах даної системи вся робота з файлами, папками та аномаліями поведінки прив’язується саме до UID, що гарантує персоналізацію та безпеку зберігання даних.

Реєстрація нового користувача

Користувач переходить на сторінку `/register`, де заповнює поля email і password. Після натискання кнопки «Зареєструватися» система створює новий акаунт і автоматично перенаправляє користувача на Dashboard.

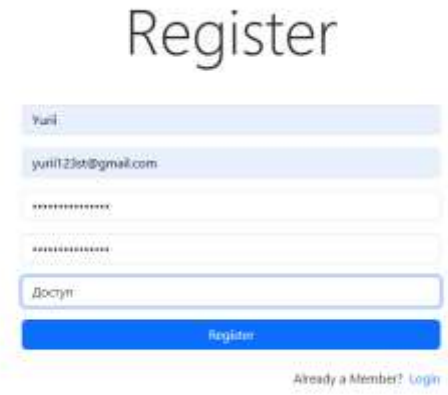


Рисунок 3.2 – Реєстрація користувача

У випадку некоректного пароля або вже зареєстрованого email система виведе відповідне повідомлення про помилку.

Авторизація користувача

На сторінці `/login` користувач вводить email і пароль. У разі правильного введення даних відкривається головна сторінка зі всіма раніше завантаженими файлами та папками.

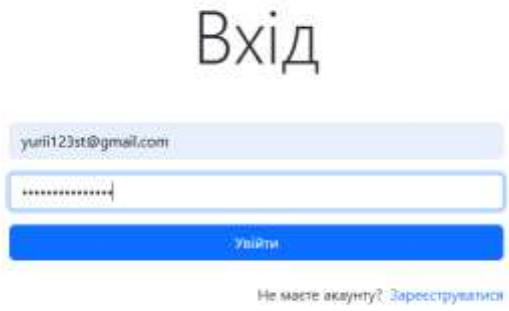


Рисунок 3.3 – Вхід користувача

Firebase зберігає сесію користувача, тому повторна авторизація не потрібна при оновленні сторінки.

Завантаження файлів

На головній сторінці натискається кнопка «Upload File». Після вибору файлу починається процес завантаження у Firebase Storage, відображається індикатор прогресу. Після завершення у Firestore створюється запис з метаданими, і новий файл з’являється в списку.

Рисунок 3.4 – Головна сторінка

Користувач може:

переглядати файл (якщо тип підтримується);

завантажити його на локальний пристрій;

видалити файл з системи.

Створення папок

На сторінці Dashboard є кнопка «Create File», яка відкриває вікно для введення назви. Після натискання кнопки «Створити» у Firestore створюється новий запис типу " Created Files", і вона відображається в інтерфейсі.

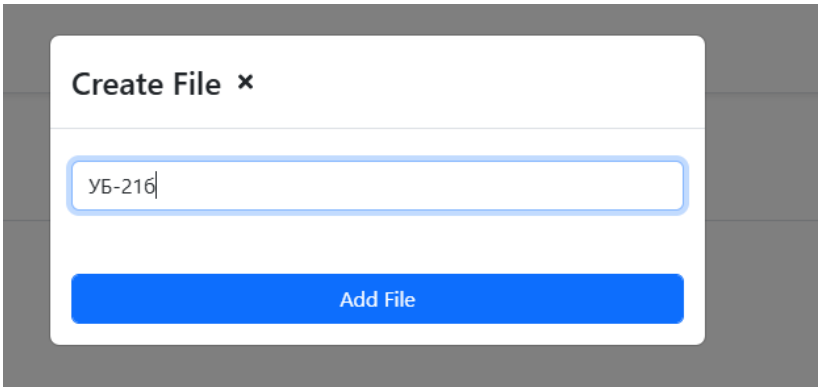


Рисунок 3.5 – Створення файлу

Усі подальші дії (завантаження файлів) можна виконувати в межах цієї папки, що формує логічну структуру.

На зображені показано розділ **Cloud Firestore** у Firebase, а саме колекцію `docs`, яка містить метадані документів, створених або завантажених користувачами. Кожен документ у цій колекції зберігає важливу інформацію:

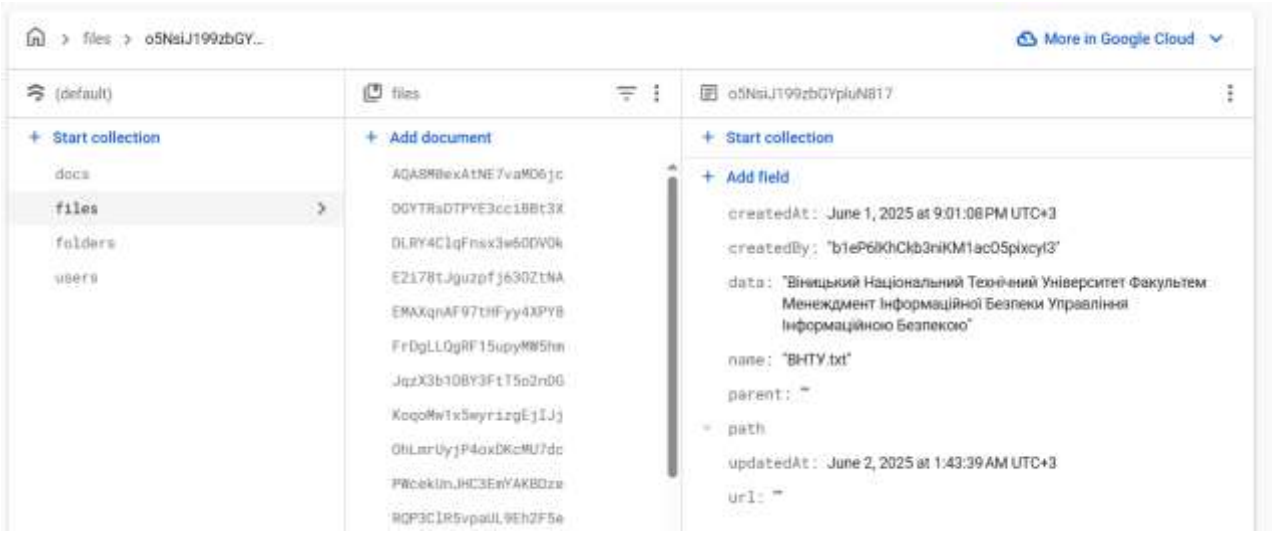


Рисунок 3.6 – Вікно в Firebase з всіма файлами і їх інформація

Ця структура даних дозволяє системі ефективно організовувати файли за принципом дерева, а також логувати всі дії користувача для забезпечення контролю та безпеки. Інформація з `docs` використовується інтерфейсом для відображення структури файлів у Dashboard.

Виявлення підозрілої активності

На зображенні показано додатковий етап перевірки доступу до системи. Цей механізм виступає як елемент поведінкового аналізу: якщо система виявляє підозрілу активність користувача (наприклад, надто часті дії за короткий проміжок часу), вона автоматично блокує інтерфейс і пропонує пройти верифікацію. Для продовження взаємодії користувач повинен ввести кодове слово, яке є умовним підтвердженням автентичності його дій.

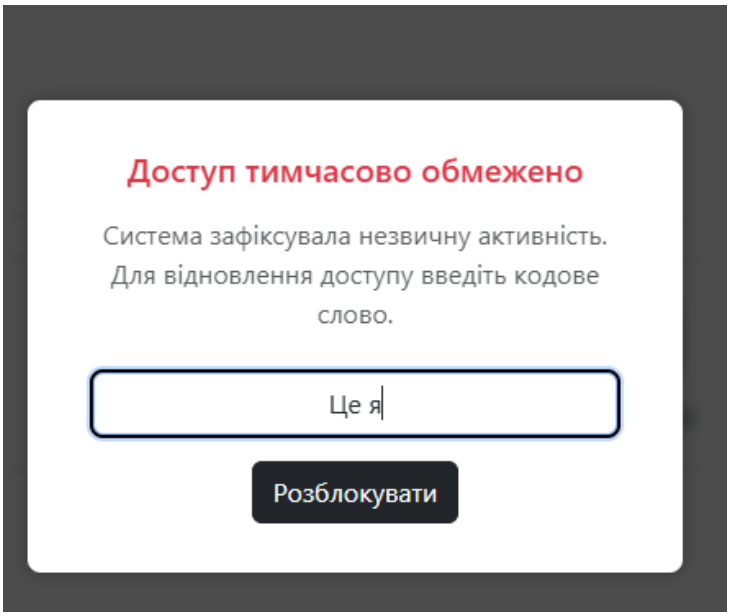


Рисунок 3.7 – Функція ШІ яка виявляє подозрілу активність

Такий підхід слугує додатковим рівнем безпеки в межах логіки «штучного інтелекту» та демонструє здатність системи реагувати на нетипову поведінку в режимі реального часу. Він імітує поведінковий фаєрвол або систему моніторингу активності, яка захищає сховище від потенційних зловмисників або автоматизованих ботів.

На цьому зображенні демонструється модальне вікно із завантаженням, яке інформує користувача про те, що **штучний інтелект (ШІ)** обробляє файл. Такий інтерфейс відображається під час виконання аналізу завантаженого файлу або його метаданих, що імітує поведінку систем з інтелектуальним аналізом вмісту.

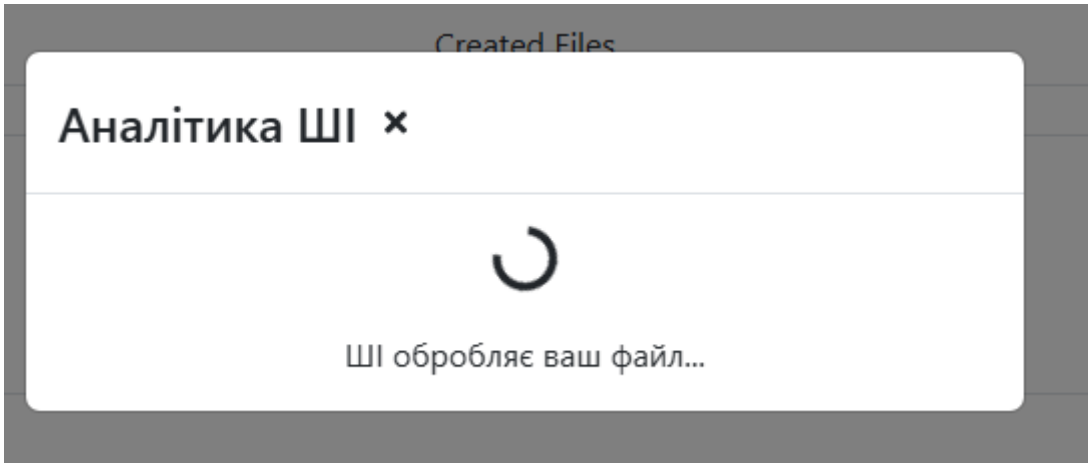


Рисунок 3.8 – Функція ШІ обробка файлу

Система може використовувати дані про тип файлу, частоту дій користувача або структуру імен файлів для формування аналітичного висновку чи визначення підозрілої активності. У цьому прикладі реалізовано візуальний індикатор обробки — індикатор завантаження (spinner) — і повідомлення "ШІ обробляє ваш файл…", яке підкреслює автоматичну реакцію системи.

Такий підхід є частиною концепції поведінкової безпеки й створює враження інтелектуального аналізу, навіть якщо в основі лежить евристичне правило або аналіз дій у Firestore

У цьому підрозділі було наочно продемонстровано роботу основних функцій системи: від реєстрації та авторизації користувача до взаємодії з файлами у хмарному середовищі. Скріншоти й описи інтерфейсів підтверджують працездатність реалізованої архітектури, а також інтеграцію аналітичного модуля, що реагує на дії користувача. Система забезпечує зручний, інтуїтивний інтерфейс, а також включає елементи адаптивного контролю дій, що імітують інтелектуальний аналіз, створюючи підґрунтя для подальшого розвитку в напрямі розумних систем безпеки.

3.7 Висновки до розділу

У межах третього розділу було детально розглянуто процес реалізації програмного забезпечення системи захищеної віддаленої роботи користувача у хмарному сховищі з використанням штучного інтелекту. Описано технології, середовище розробки, архітектуру застосунку та логіку роботи основних компонентів. Продемонстровано, як реалізовано реєстрацію, авторизацію, завантаження файлів, створення папок та відображення їх у структурованому вигляді.

Особливу увагу приділено інтеграції з платформою **Firebase**, яка забезпечила просту та безпечну реалізацію автентифікації, зберігання файлів та обробки метаданих. Крім того, в систему було вбудовано елементи поведінкової аналітики, що імітують функції штучного інтелекту — такі як виявлення аномальної активності користувача або модальне попередження при підозрілих діях.

Кожен підрозділ супроводжувався прикладами коду та візуалізацією інтерфейсу, що дозволяє чітко побачити роботу системи на практиці. Загалом, реалізоване програмне забезпечення відповідає поставленим вимогам, є функціональним, масштабованим і демонструє готовність до реального використання з можливістю подальшого розширення.

ВИСНОВКИ

У результаті виконання дипломної роботи «Розробка програми захищеної віддаленої роботи користувача у хмарному сховищі з використанням штучного інтелекту» було повністю реалізовано цикл створення сучасного веб-застосунку, що забезпечує зручну, безпечну та функціонально насичену взаємодію користувача з хмарною інфраструктурою.

У теоретичній частині проведено комплексний аналіз концепції віддаленого доступу, хмарних технологій та методів забезпечення інформаційної безпеки. Розглянуто роль штучного інтелекту у виявленні загроз, моніторингу поведінки користувача та загалом у підвищенні надійності цифрових систем. Оцінено наявні рішення ринку (Microsoft AVD, Amazon WorkSpaces, Google Cloud Workstations), що дозволило окреслити чіткі вимоги до майбутнього програмного продукту.

У проєктній частині розроблено архітектуру програмного забезпечення, визначено основні функціональні компоненти та логіку їх взаємодії. Побудовано алгоритм роботи системи, створено блок-схеми, діаграми використання та описано модель користувацької поведінки. Передбачено підтримку базової системи аналітики для подальшої інтеграції інтелектуальних засобів аналізу дій користувача.

Окрему увагу приділено впровадженню функціональності, що імітує елементи штучного інтелекту — зокрема, виявлення підозрілої активності користувача за допомогою аналізу частоти його дій. Таке розширення дозволяє не лише забезпечити базову безпеку, але й закладає основу для подальшого розвитку — наприклад, автоматичного блокування сесій, адаптивного контролю доступу або поведінкової аналітики.

Інтерфейс реалізовано з урахуванням принципів UX/UI дизайну — застосунок адаптивний, простий у використанні, розбитий на логічні модулі та легко розширюється. Уся логіка взаємодії реалізована з використанням сучасних підходів до роботи з подіями, обробки помилок та захисту доступу на рівні хмарної інфраструктури.

Таким чином, створене програмне забезпечення відповідає сучасним вимогам до надійності, гнучкості та масштабованості. Воно може використовуватись як основа для розгортання персональних або малих корпоративних рішень із віддаленим доступом до хмарного середовища, з можливістю подальшого розширення функціоналу на основі ШІ та аналітики.

ПЕРЕЛІК ПОСИЛАНЬ

1. Хмарні технології URL: https://ucloud.ua/hmarni-tehnologiyi-shho-cze-take/ (Дата звернення: 10.05.2025)

2. Типи хмарних технологій URL: https://denovo.ua/blog/vidi-hmarnih-servisiv-yakij-obrati-ta-oglyad-hmarnih-provajd-8 (Дата звернення: 10.05.2025)

3. Переваги та недоліки хмарних технологій URL: https://edin.ua/shho-take-xmarni-texnologi%D1%97-i-navishho-voni-potribni/ (Дата звернення: 10.05.2025)

4. Віддалений доступ до даних URL: https://itez.com.ua/blog/remote-desktop-benefits.html (Дата звернення: 10.05.2025)

5. Що таке хмарні сховища та як вони працюють URL: https://info.nic.ua/uk/blog-uk/cloudstorage-2/ (Дата звернення: 10.05.2025)

6. Фішинг на хмарні сховища URL: https://hub.kyivstar.ua/articles/fishyng-nichnyj-zhah-dlya-kompanij-u-2023-shho-cze-za-kiberataka-ta-yak-zahystytysya (Дата звернення: 10.05.2025)

7. Проблеми безпеки хмарних сховищ URL: https://elartu.tntu.edu.ua/bitstream/lib/40618/2/MNPK_2022_Shymchuk_G_V-Security_problems_of_187-188.pdf (Дата звернення: 10.05.2025)

8. Інформаційна безпека в сучасному цифровому освітньому URL: https://enpuir.npu.edu.ua/handle/123456789/48219 (Дата звернення: 12.05.2025)

9. Ключові переваги штучного інтелекту URL: https://constructor.tech/ua/node/294 (Дата звернення: 12.05.2025)

10. Штучний інтелект у хмарних інфраструктурах URL: https://ardenis.com.ua/blog/shtuchnyj-intelekt-shi-perevagy-ta-nedoliky-chastyna-1/ (Дата звернення: 12.05.2025)

11. Персональне навчальне середовище. Хмарні технології URL: https://informatik.pp.ua/uroky/9-klas/konspekty-uchnia/9-klas-urok-45-personalne-navchalne-seredovyshche-khmarni-tekhnolohii-2/ (Дата звернення: 12.05.2025)

12. Популярні корпоративні рішення URL: https://softtim.com.ua/rishennya/khmarni-rishennya/nalashtuvannya-microsoft-wvd (Дата звернення: 12.05.2025)

13. Сучасні хмарні рішення URL: https://ucloud.ua/hmarni-tehnologiyi-shho-cze-take/ (Дата звернення: 12.05.2025)

14. Аналіз існуючих рішень у сфері захищеного віддаленого доступу до хмарних середовищ URL: https://eska.global/blog/analiz-perevag-ta-rizikiv-pri-perehodi-do-hmarnih-poslug (Дата звернення: 12.05.2025)

15. A Secured Cloud-Based Electronic Document Management System URL: https://www.researchgate.net/publication/369768638_A_Secured_Cloud-Based_Electronic_Document_Management_System (Дата звернення: 13.05.2025)

16. Модель використання програмної системи URL: https://visuresolutions.com/uk (Дата звернення: 13.05.2025)

17. Архітектура системи URL: https://duikt.edu.ua/ua/news-1-0-11275-systems-architect---profesiya-do-yakoi-mozhut-pidnyati-lishe-studenti-scho-gotuyutsya-za-specialnistyukompyuternnauki?lang=ua&act=view&page=1&category=0&id=11275&sys_link=systems-architect---profesiya-do-yakoi-mozhut-pidnyati-lishe-studenti-scho-gotuyutsya-za-specialnistyu-kompyuterni-nauki (Дата звернення: 13.05.2025)

18. Хмарна платформа Firebase URL: https://avada-media.ua/blog/firebase/ (Дата звернення: 17.05.2025)

19. Взаємодії зі сторонніми бібліотеками URL: https://uk.legacy.reactjs.org/docs/integrating-with-other-libraries.html (Дата звернення: 17.05.2025)

20. Побудова та розуміння алгоритмів URL: https://cloud.itstep.org/blog/building-and-understanding-algorithms-a-step-by-step-guide-for-beginners (Дата звернення: 17.05.2025)

21. Мова програмування JavaScript URL: https://goit.global/ua/articles/shcho-take-javascript-i-dlia-choho-vin-potriben/ (Дата звернення: 17.05.2025)

22. Що таке React JS? URL: https://brainlab.com.ua/uk/blog-uk/shho-take-react-js-yak-pochaty-vyvchaty-reakt (Дата звернення: 20.05.2025)

23. Середовище розробки Visual Studio Code URL: https://top-keis.com.ua/shho-take-visual-studio-code/ (Дата звернення: 20.05.2025)

24. Інтерфейс користувача URL: https://voll.com.ua/uk/glossary/koristuvackij-interfejs-ui (Дата звернення: 20.05.2025)

25. Firebase що це і як працює? URL: https://dou.ua/forums/topic/44058/ (Дата звернення: 20.05.2025)

ДОДАТКИ

Додаток А. Технічне завдання

Вінницький національний технічний університет
Факультет менеджменту та інформаційної безпеки
Кафедра менеджменту та безпеки інформаційних систем

ЗАТВЕРДЖУЮ
Голова секції "Управління інформаційною безпекою" кафедри МБІС
д.т.н., професор
_____ Юрій ЯРЕМЧУК
" 20 " березня 2025 р.

ТЕХНІЧНЕ ЗАВДАННЯ

до бакалаврської кваліфікаційної роботи на тему:
Розробка програми захищеної віддаленої роботи користувача у хмарному сховищі з використанням штучного інтелекту
08-72.БКР. 018.00.069 ТЗ

Керівник бакалаврської кваліфікаційної
роботи
д.т.н., проф. кафедри МБІС
_____ Яремчук Ю.Є.
"20" березня 2025 р

Вінниця – 2025 р.

1. Найменування та область застосування

Програмний засіб захищеного віддаленого доступу користувача до хмарного сховища з елементами штучного інтелекту.
Область застосування: забезпечення безпечного зберігання та обробки даних у хмарних середовищах, захист інформаційних ресурсів від несанкціонованого доступу у вебзастосунках і корпоративних системах.

2.Підстава для розробки

Розробка виконується на основі наказу ректора ВНТУ № 97 від 20.03.2025 р.

3. Мета та призначення розробки

3.1 Мета розробки: створення програми для безпечного віддаленого доступу до хмарного сховища з використанням штучного інтелекту.

3.2 Призначення: забезпечення зручного керування файлами та захисту даних шляхом виявлення підозрілої активності користувача.

4. Джерела розробки

4.1. Google LLC. Firebase Documentation. Google Developers. 2024.

4.2. Meta Platforms Inc. React: A JavaScript Library for Building User Interfaces. React Official Docs. 2024.

4.3. Airbnb Inc. Airbnb JavaScript Style Guide // GitHub Repository. 2024.

4.4. Bhardwaj A., Raj S. Artificial Intelligence in Cybersecurity: A Review // Journal of Information Security. 2020. № 11. P. 145–156.

5. Вимоги до програми

5.1 Вимоги до функціональних характеристик:

5.1.1 Програма повинна мати зручний та інтуїтивно зрозумілий інтерфейс для керування файлами в хмарному сховищі.

5.1.2 Реєстрація та авторизація мають виконуватись через Firebase Authentication.

5.1.3 Система повинна забезпечувати завантаження, перегляд, редагування та видалення файлів користувача.

5.1.4 Модуль штучного інтелекту має здійснювати виявлення підозрілої активності.

5.2 Вимоги до надійності:

5.2.1 Програма повинна стабільно працювати без збоїв, інформуючи про помилки при їх виникненні.

5.2.2 Дані повинні зберігатися у Firebase з можливістю резервного копіювання.

5.2.3 Усі ключові функції мають працювати коректно за стандартних умов.

5.3 Вимоги до складу і параметрів технічних засобів:

– процесор — Intel Core i3 2 ГГц або аналогічний;
– оперативна пам’ять — не менше 4 GB;
– середовище — Windows 10
– наявність стабільного підключення до Інтернету;
– дотримання норм безпеки при роботі з комп’ютерною технікою.

6. Вимоги до програмної документації

6.1 Обов’язкова поетапна інструкція для майбутніх користувачів, наведена у пункті 3.4

7. Вимоги до технічного захисту інформації

7.1 Необхідно забезпечити захист від несанкціонованих дій користувачів, включаючи спроби доступу до чужих файлів або змін даних.

7.2 Неавторизовані користувачі не повинні мати доступу до інформаційних ресурсів — усі дії мають бути дозволені лише після успішної автентифікації.

8. Техніко-економічні показники

8.1 Вартість розробки мінімальна завдяки використанню безкоштовних технологій, при цьому користь від застосування перевищує витрати.

8.2 Програма розрахована на широке коло користувачів та не потребує дорогого обладнання чи програмного забезпечення.

9. Стадії та етапи розробки

№ з/п	Назва етапів бакалаврської кваліфікаційної роботи	**Початок**	**Закінчення**
1.	Визначення напрямку бакалаврської роботи, формулювання теми	20.03.20025	15.04.2025
2.	Аналіз предметної області обраної теми	15.24.2025	20.04.2025
3.	Розробка алгоритму роботи	20.04.2025	27.04.2025
4.	Написання бакалаврської роботи на основі розробленої теми	27.04.2025	25.05.2025
5.	Передзахист бакалаврської кваліфікаційної роботи	25.05.2025	26.05.2025
6.	Виправлення, уточнення, корегування бакалаврської кваліфікаційної роботи	27.05.2025	11.06.2025
7.	Захист бакалаврської кваліфікаційної роботи	11.06.2025	12.06.2025

10. Порядок контролю та прийому

10.1 До приймання бакалаврської кваліфікаційної роботи надається:

– ПЗ до бакалаврської кваліфікаційної роботи;

- демонстрація результату бакалаврської кваліфікаційної роботи;
- презентація;
- відзив керівника роботи;
- відзив рецензента

Технічне завдання до виконання прийняв _____Стойківський Ю.В.

Додаток Б. Лістинг програмного коду

```
import React, { useEffect, useState } from "react";
import { Button, Col, Container, Form, Row } from "react-bootstrap";
import { useDispatch, useSelector } from "react-redux";
import { Link, useHistory, useLocation } from "react-router-dom";
import { toast } from "react-toastify";
import { loginUser } from "../../../redux/actionCreators/authActionCreators";

const Login = () => {
  const [email, setEmail] = useState("");
  const [password, setPassword] = useState("");
  const [error, setError] = useState("");

  const [verified, setVerified] = useState(false);
  const [captchaInput, setCaptchaInput] = useState("");

  const isLoggedIn = useSelector((state) => state.auth.isLoggedIn);
  const dispatch = useDispatch();
  const history = useHistory();
  const { pathname } = useLocation();

  const handleSubmit = (e) => {
    e.preventDefault();

    if (!email || !password) return toast.dark("Будь ласка, заповніть всі поля");
    const data = {
      email,
      password,
    };
    dispatch(loginUser(data, setError));
  };

  useEffect(() => {
    if (error) {
      toast.error(error);
    }
    if (isLoggedIn) {
      history.goBack();
    }
  }, [error]);

  // Візуальне блокування доступу
  if (!verified) {
    return (
      <Container className="d-flex align-items-center justify-content-center" style={{ height:
"100vh" }}>
```

```
    <div className="text-center">
     <h2 className="mb-4">Перевірка доступу</h2>
     <p className="mb-3">Для продовження введіть кодове слово</p>
     <input
      type="text"
      value={captchaInput}
      onChange={(e) => setCaptchaInput(e.target.value)}
      placeholder="Кодове слово"
      style={{
       padding: "10px",
       fontSize: "1rem",
       borderRadius: "6px",
       border: "1px solid #ccc",
       width: "250px",
       marginBottom: "10px"
      }}
     />
     <br />
     <Button
      variant="secondary"
      onClick={() => {
       if (captchaInput.toLowerCase() === "доступ") {
        setVerified(true);
       } else {
        toast.error("Невірне кодове слово");
       }
      }}
     >
      Продовжити
     </Button>
    </div>
   </Container>
  );
}

return (
 <Container>
  <Row>
   <Col md="12">
    <h1 className="display-1 my-5 text-center">Вхід</h1>
   </Col>
   <Col md="5" className="mx-auto">
    <Form onSubmit={handleSubmit}>
     <Form.Group controlId="formBasicEmail" className="mb-3">
      <Form.Control
       type="email"
       placeholder="Email"
       value={email}
```

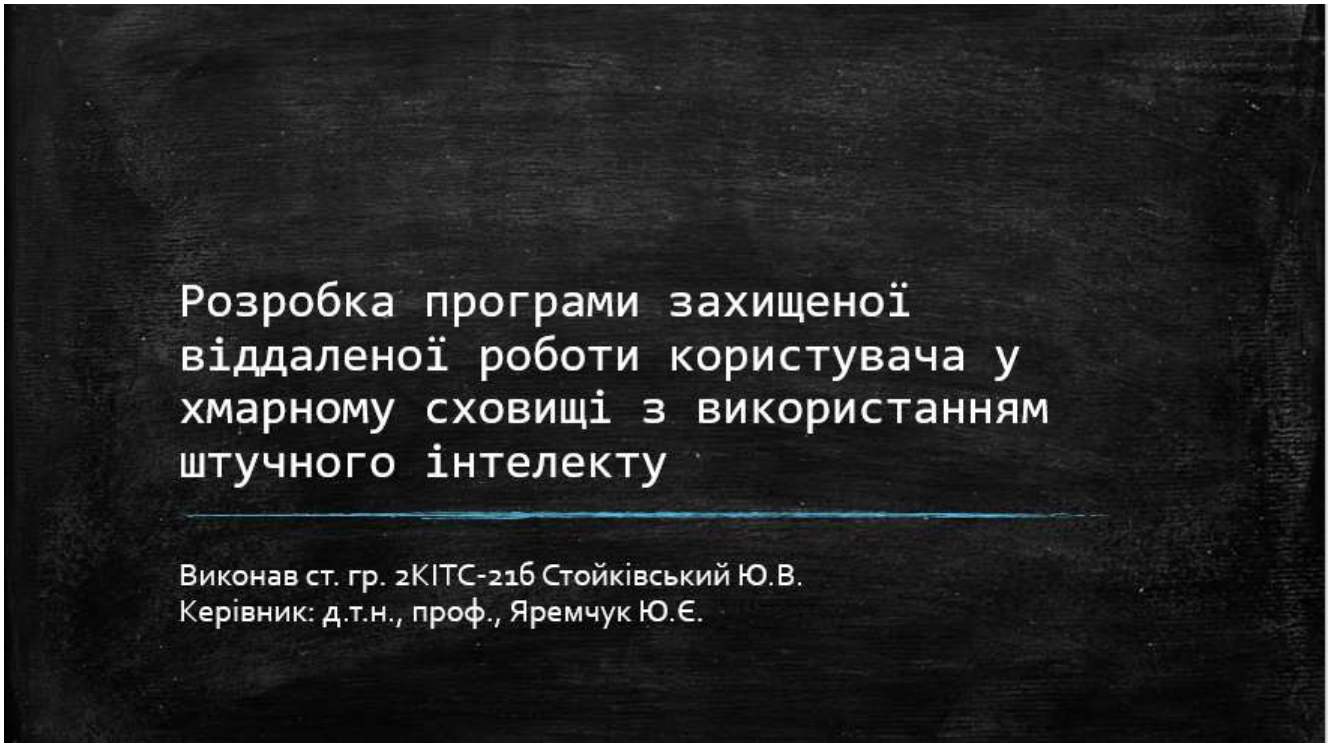
```
                onChange={(e) => setEmail(e.target.value)}
               />
              </Form.Group>
              <Form.Group controlId="formBasicPassword" className="mb-3">
               <Form.Control
                type="password"
                placeholder="Пароль"
                value={password}
                onChange={(e) => setPassword(e.target.value)}
               />
              </Form.Group>
              <Form.Group controlId="formBasicBtn" className="mt-3">
               <Button
                variant="primary"
                type="submit"
                className="form-control"
                block
               >
                Увійти
               </Button>
              </Form.Group>
              <p className="text-right d-flex align-items-center justify-content-end gap-2 ml-auto
my-4">
               Не маєте акаунту?
               <Link to="/signup" className="ml-2 text-decoration-none">
                Зареєструватися
               </Link>
              </p>
             </Form>
            </Col>
           </Row>
          </Container>
         );
        };

        export default Login;
        import React, { useEffect, useState } from "react";
        import { Button, Col, Container, Form, Row } from "react-bootstrap";
        import { useDispatch, useSelector } from "react-redux";
        import { Link, useHistory } from "react-router-dom";
        import { toast } from "react-toastify";
        import { registerUser } from "../../../redux/actionCreators/authActionCreators";

        const Register = () => {
         const [name, setName] = useState("");
         const [email, setEmail] = useState("");
         const [password, setPassword] = useState("");
         const [confirmPassword, setConfirmPassword] = useState("");
```

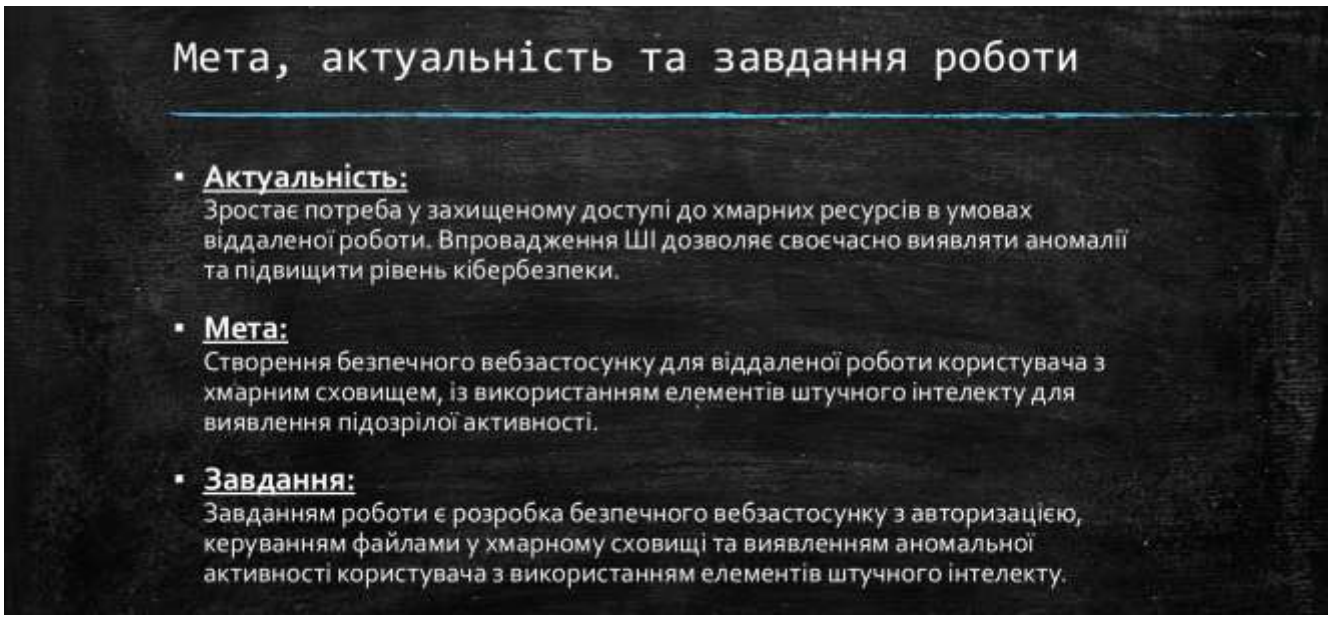
```
const [secretCode, setSecretCode] = useState("");
const [error, setError] = useState("");

const isLoggedIn = useSelector((state) => state.auth.isLoggedIn);
const dispatch = useDispatch();
const history = useHistory();

const handleSubmit = (e) => {
 e.preventDefault();

 if (!name || !email || !password || !secretCode)
  return toast.dark("Please fill in all fields!");

 if (password !== confirmPassword)
  return toast.dark("Passwords do not match!");

 if (password.length < 8) {
  return toast.dark("Password must be of length 8 or more");
 }

 if (
  !/^(?=.*[0-9])(?=.*[!@#$%^&*])[a-zA-Z0-9!@#$%^&*]{6,16}$/.test(password)
 ) {
  return toast.dark(
   "Password must have at least a number and a special character!"
  );
 }

 // Зберігаємо кодове слово в localStorage (імітація)
 localStorage.setItem("userSecretCode", secretCode);

 const data = {
  name,
  email,
  password,
 };

 dispatch(registerUser(data, setError));
};

useEffect(() => {
 if (error) {
  toast.error(error);
 }
 if (isLoggedIn) {
  history.push("/dashboard");
 }
}, [error, isLoggedIn]);
```

```
return (
 <Container>
  <Row>
   <Col md="12">
    <h1 className="display-1 my-5 text-center">Register</h1>
   </Col>
   <Col md="5" className="mx-auto">
    <Form onSubmit={handleSubmit}>
     <Form.Group controlId="formBasicName" className="mb-3">
      <Form.Control
       type="text"
       placeholder="Name"
       value={name}
       onChange={(e) => setName(e.target.value)}
      />
     </Form.Group>
     <Form.Group controlId="formBasicEmail" className="mb-3">
      <Form.Control
       type="email"
       placeholder="Email"
       value={email}
       onChange={(e) => setEmail(e.target.value)}
      />
     </Form.Group>
     <Form.Group controlId="formBasicPassword" className="mb-3">
      <Form.Control
       type="password"
       placeholder="Password"
       value={password}
       onChange={(e) => setPassword(e.target.value)}
      />
     </Form.Group>
     <Form.Group controlId="formBasicConfirmPassword" className="mb-3">
      <Form.Control
       type="password"
       placeholder="Re-type password"
       value={confirmPassword}
       onChange={(e) => setConfirmPassword(e.target.value)}
      />
     </Form.Group>
     <Form.Group controlId="formSecretCode" className="mb-3">
      <Form.Control
       type="text"
       placeholder="Create your secret code"
       value={secretCode}
       onChange={(e) => setSecretCode(e.target.value)}
      />
```

```
                </Form.Group>
                <Form.Group controlId="formBasicBtn" className="mt-3">
                 <Button
                  variant="primary"
                  type="submit"
                  className="form-control"
                  block
                 >
                  Register
                 </Button>
                </Form.Group>
                <p className=" text-right d-flex align-items-center justify-content-end gap-2 ml-auto
my-4">
                 Already a Member?
                 <Link to="/login" className="ml-2 text-decoration-none">
                  Login
                 </Link>
                </p>
               </Form>
              </Col>
             </Row>
            </Container>
          );
        };

        export default Register;
```

Додаток В. Ілюстраційний матеріал





Завдання роботи

- Проаналізувати предметну область.
- Обґрунтувати вибір інструментів розробки.
- Розробити архітектуру системи.
- Реалізувати функціонал: авторизація, робота з файлами.
- Інтегрувати ШІ-модуль виявлення аномалій.
- Провести тестування та аналіз результатів.

Характеристика розробки

– Вебзастосунок на React.
– Firebase: аутентифікація, зберігання файлів, база даних.
– Авторизація, реєстрація, робота з файлами та папками.
– ШІ-модуль фіксує аномальну активність користувача.
– Простий, адаптивний інтерфейс.

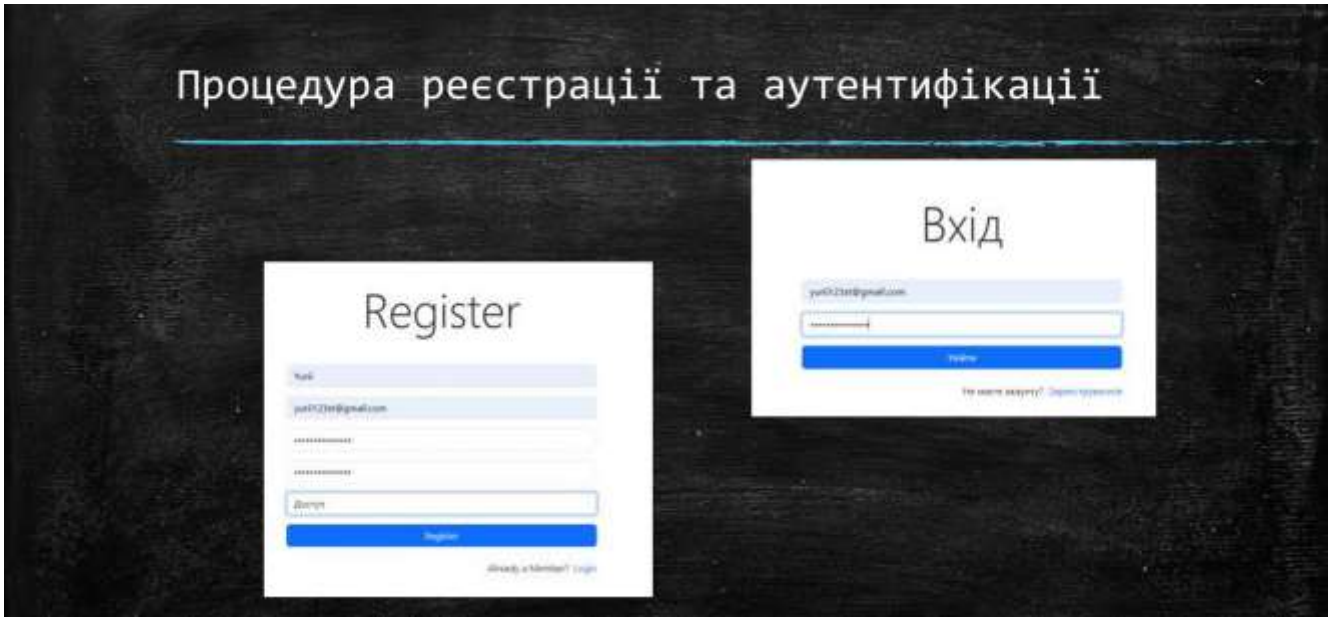
Вибір технологій для розробки програмного модуля та середовища розробки

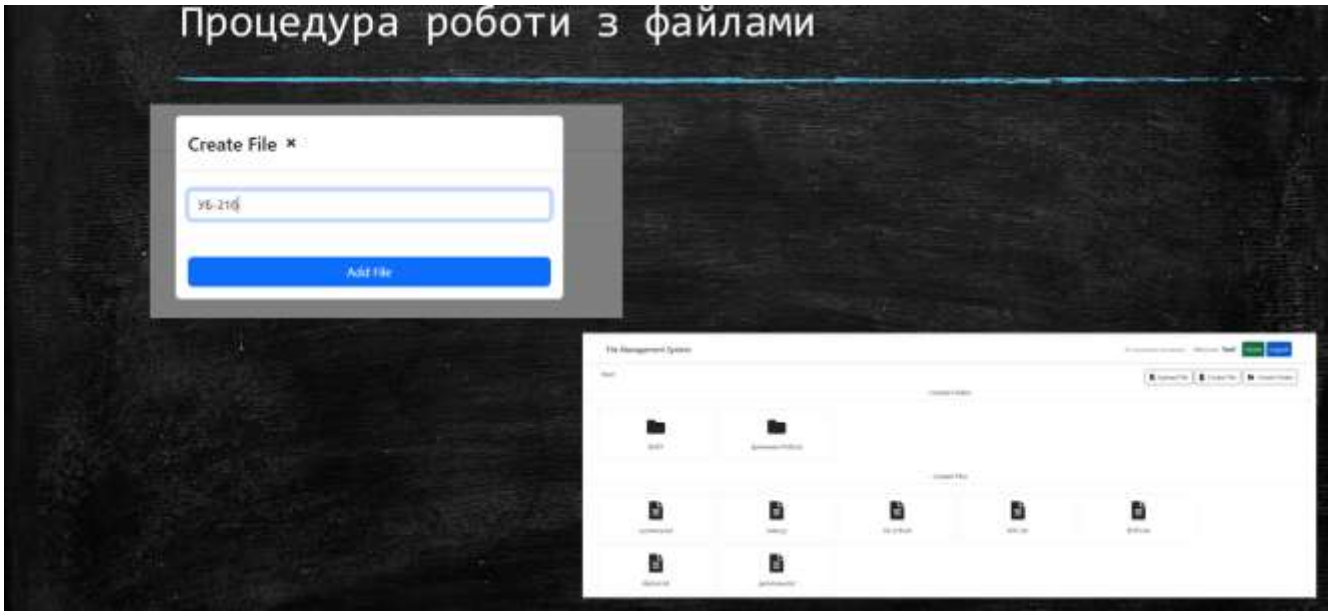
1. Мова програмування та фреймворк:
Основою клієнтської частини став фреймворк React.js, написаний на мові JavaScript. React дозволяє створювати швидкі, динамічні та компонентно орієнтовані інтерфейси користувача.
2. Хмарна платформа:
Для реалізації серверної логіки, автентифікації користувачів, зберігання файлів і даних було обрано Firebase від Google.
- Firebase Authentication – для реєстрації та входу користувачів;
- Firestore – для зберігання метаданих про файли та дії користувачів;
- Firebase Storage – для зберігання самих файлів;
- Firebase Security Rules – для контролю доступу до ресурсів.

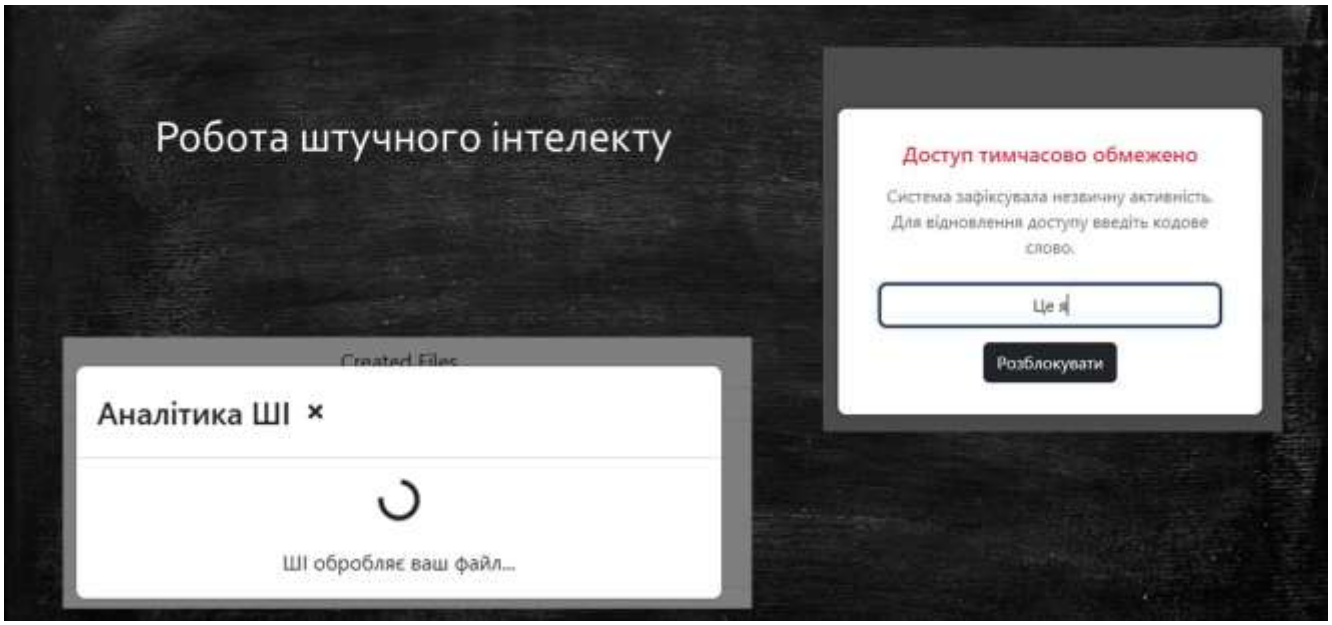
3. Середовище розробки:
Розробка здійснювалась у Visual Studio Code (VS Code) — потужному редакторі коду з розширеннями для роботи з React, Git, Firebase та ESLint. Це середовище є кросплатформним, легким та зручним для роботи з вебзастосунками.

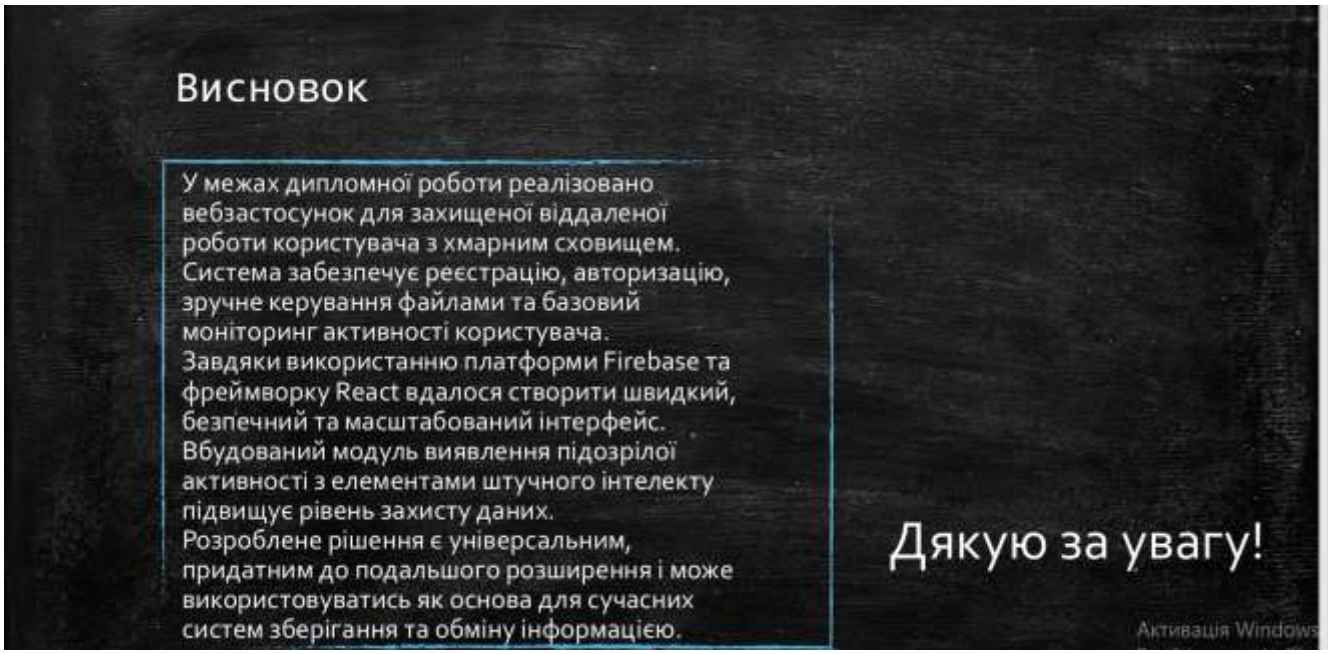
Алгоритм роботи системи

- Користувач проходить автентифікацію.
- Завантажує/редагує/видаляє файли.
- Файли зберігаються у Firebase Storage.
- Дії записуються в Firestore.
- Модуль аналізує активність і фіксує аномалії.
- У разі загрози система повідомляє користувача.


Процедура реєстрації та аутентифікації
Register
Вхід


Процедура роботи з файлами
Create File
Add File


Робота штучного інтелекту
Доступ тимчасово обмежено
Система зафіксувала незвичну активність. Для відновлення доступу введіть кодове слово.
Це я
Розблокувати
Created Files
Аналітика ШІ ×
ШІ обробляє ваш файл...


Висновок
У межах дипломної роботи реалізовано вебзастосунок для захищеної віддаленої роботи користувача з хмарним сховищем. Система забезпечує реєстрацію, авторизацію, зручне керування файлами та базовий моніторинг активності користувача.
Завдяки використанню платформи Firebase та фреймворку React вдалося створити швидкий, безпечний та масштабований інтерфейс.
Вбудований модуль виявлення підозрілої активності з елементами штучного інтелекту підвищує рівень захисту даних.
Розроблене рішення є універсальним, придатним до подальшого розширення і може використовуватись як основа для сучасних систем зберігання та обміну інформацією.
Дякую за увагу!
Активація Windows

Додаток Г. Протокол перевірки на антиплагіат

ПРОТОКОЛ ПЕРЕВІРКИ НАВЧАЛЬНОЇ (КВАЛІФІКАЦІЙНОЇ) РОБОТИ

Назва роботи:

Розробка програми захищеної віддаленої роботи користувача у хмарному сховищі з використанням штучного інтелекту

Тип роботи: бакалаврська кваліфікаційна робота

Підрозділ Кафедра менеджменту на безпеки інформаційних систем
Факультет менеджменту та інформаційної безпеки
Гр. 2КІТС-21б

Керівник професор Яремчук Ю.Є. _____

Показники звіту подібності

Strike Plagiarism	
Оригінальність	85,87%
Загальна схожість	14,13%

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 _____ Стойківський Ю.В.
(підпис) (прізвище, ініціали)

Опис прийнятого рішення

☐ **Допустити до захисту**

Особа, відповідальна за перевірку _____ Коваль Н.П.
(підпис) (прізвище, ініціали)

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