Хмельницький національний університет
Факультет інформаційних технологій
Кафедра кібербезпеки

КВАЛІФІКАЦІЙНА РОБОТА

Вовковича Максима Олександровича

на здобуття ступеня вищої освіти Бакалавра

Система забезпечення безпеки конфіденційних даних від витоку на основі керування паролями

Галузь знань 12 – Інформаційні технології
Спеціальність 125 Кібербезпека
Освітня програма Кібербезпека

КРБКБ.2102141.21.02.20 ПЗ

Виконав студент 4 курсу група КБ-21-2 Максим ВОВКОВИЧ
Керівник канд. техн. наук, доцент Володимир ДЖУЛІЙ
Нормоконтролер старший викладач Сергій МОСТОВИЙ

До захисту допускаю:
Завідувач кафедри кібербезпеки Юрій КЛЬОЦ
06 06 2025 р.

Хмельницький 2025

ХМЕЛЬНИЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

Факультет Інформаційних технологій

Кафедра Кібербезпеки

Рівень вищої освіти Бакалавр

Галузь знань 12 – Інформаційні технології

Спеціальність 125 Кібербезпека

Освітня програма Кібербезпека

ЗАТВЕРДЖУЮ

Завідувач кафедри кібербезпеки

Юрій КЛЬОЦ

15 лютого 2025 р.

З А В Д А Н Н Я
НА КВАЛІФІКАЦІЙНУ РОБОТУ

Вовковичу Максиму Олександровичу

1 Тема роботи Система забезпечення безпеки конфіденційних даних від витоку на основі керування паролями.

Керівник роботи канд. техн. наук, доцент Володимир ДЖУЛІЙ

Затверджено наказом ректора університету від 7 лютого 2025 № 23

2 Строк подання студентом кваліфікаційної роботи на кафедру

3 Вихідні дані до роботи Проаналізувати існуючі рішення систем керування паролями. Підбір оптимального стеку технологій для розробки власного рішення.Проектування програмного забезпечення. Розробка спроектованого забезпечення. Тестування даного рішення. Представлення результатів.

4 Зміст пояснювальної записки (перелік питань, які потрібно розробити) Вступ. Дослідження предметної області. Постановка задачі. Проектування програмної захищеної системи керування паролями. Програмна реалізація захищеної системи керування паролями на підприємстві.

5 Перелік графічного матеріалу (із зазначенням обов'язкових креслень) Код програми. Схема загальної архітектури програми. Графічний інтерфейс. Блок-схеми роботи сервісів

6 Консультанти розділів кваліфікаційної роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв

7 Дата видачі завдання 16 лютого 2025 р.

КАЛЕНДАРНИЙ ПЛАН

Назва етапів (розділів) кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
Вибір і затвердження теми кваліфікаційної роботи	Січень-Лютий	+
Ознайомлення з предметною областю	Лютий	+
Дослідження існуючих рішень	Лютий	+
Постановка задачі	Березень	+
Визначення загальних принципів рішення задачі	Березень	+
Деталізація принципів рішення задачі	Квітень	+
Розробка проєктних рішень	Квітень	+
Тестування ефективності реалізовних рішень	Травень	+
Оформлення пояснювальної записки згідно вимог	Травень	+
Оформлення графічної частини	Червень	+
Захист КР	Червень	+

Студент _____ Максим ВОВКОВИЧ

Керівник кваліфікаційної роботи _____ Володимир ДЖУЛІЙ

АНОТАЦІЯ

Тема кваліфікаційної роботи: Система забезпечення безпеки конфіденційних даних від витоку на основі керування паролями.

Автор роботи: студент 4 курсу групи КБ-21-2 Вовкович Максим Олександрович

Керівник роботи: к.н.т, доцент Джулій Володимир Миколайович.

Пояснювальна записка: 62 с., 3 додатки, 28 рисунків, 40 джерел.

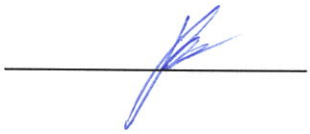
Графічна частина: 3 плакати, 10 презентаційних слайдів.

Кваліфікаційна робота бакалавра присвячена розробці системи керування паролями для забезпечення безпеки конфіденційної інформації на підприємстві.

У роботі проведено аналіз сучасних методів керування паролями та досліджено основні загрози, пов'язані з витоками даних через слабку або скомпрометовану автентифікацію. Оцінено рівень інформаційної безпеки підприємства та виявлено вразливості в управлінні обліковими даними.

В результаті роботи створено супровідну документацію до системи: політику безпеки паролів, технічне завдання, план впровадження системи, рекомендації щодо використання менеджерів паролів, регламент управління обліковими записами та протокол тестування ефективності запропонованих рішень. Запропонована система забезпечить високий рівень захисту конфіденційної інформації підприємства та мінімізує ризики несанкціонованого доступу до корпоративних ресурсів.

29.05.2025



ABSTRACT

Subject of qualification work: A password management-based system for protecting sensitive data from leakage.

Author: student of CS-21-2 Vovkovych Maksym Oleksandrovych.

Head of work: Ph. D. tech. Science. Associate Professor Dzhuliy Volodymyr Mykolayovych.

Explanatory note: 62 p., 3 appendices, 28 figures, 40 sources

Graphic part: 3 posters, 10 presentation slides.

The bachelor's qualification work is dedicated to the development of a password management system to ensure the security and confidentiality of information at the enterprise.

The study analyzes modern password management methods and investigates the main threats related to data leaks caused by weak or compromised authentication. The level of information security at the enterprise is assessed, and vulnerabilities in account data management are identified.

As a result of the research, supporting documentation for the system has been created, including a password security policy, technical specifications, an implementation plan, recommendations for using password managers, an account management policy, and a protocol for testing the effectiveness of the proposed solutions.

The proposed system will ensure a high level of security for confidential information at the enterprise and minimize the risks of unauthorized access to corporate resources.

29.05.2025 _____

ЗМІСТ

ВСТУП

У сучасному цифровому середовищі паролі є основним засобом захисту інформаційних ресурсів підприємств, незалежно від їхнього масштабу та сфери діяльності. В умовах зростання кіберзагроз та розвитку методів атак, управління паролями стає критично важливим завданням для забезпечення безпеки конфіденційної інформації. Особливо це актуально для компаній, які працюють із великим обсягом чутливих даних, зокрема таких, як підприємство «DataGroup», що займається обробкою та передачею інформації. Для запобігання витокам даних необхідно впроваджувати комплексну систему керування паролями, яка дозволяє мінімізувати ризики, пов’язані з ненадійними або скомпрометованими обліковими даними.

Однією з найбільших загроз у сфері інформаційної безпеки є людський фактор – використання слабких паролів, їх повторне застосування та неналежне зберігання. Крім того, існують зовнішні кіберзагрози, такі як фішинг, атаки на перехоплення даних, шкідливе програмне забезпечення та інші методи злому облікових записів. Тому розробка ефективної системи керування паролями є ключовим елементом загальної стратегії кібербезпеки підприємства.

Система керування паролями передбачає застосування сучасних технологій, таких як менеджери паролів, механізми багатофакторної автентифікації (MFA), шифрування облікових даних, а також централізовані політики безпеки, що регулюють створення, оновлення та контроль доступу до конфіденційної інформації. Додатково важливими є аудит та моніторинг дій користувачів, які дозволяють своєчасно виявляти потенційні загрози та реагувати на інциденти безпеки.

Зважаючи на обмежені ресурси та потребу в простих, але ефективних рішеннях, підприємствам варто впроваджувати автоматизовані системи керування паролями, які забезпечують генерацію складних ключів доступу, їх безпечне збереження та контрольовану ротацію. Це дозволяє значно знизити ризик несанкціонованого доступу до інформаційних ресурсів та мінімізувати

ймовірність витоку даних.

Актуальність дослідження визначається зростанням кількості атак, пов’язаних із компрометацією паролів, та необхідністю впровадження ефективної системи керування обліковими даними для захисту конфіденційної інформації.

Мета роботи розробити та впровадити систему керування паролями для підприємства з метою забезпечення конфіденційності, цілісності та доступності інформації та запобігання витокам даних.

Для досягнення поставленої мети необхідно виконати такі завдання:

– провести аналіз сучасних менеджерів паролів та дослідити характеристики «сильних» паролів;

– дослідити методи протидії витоку конфіденційної інформації на підприємстві;

– вивчити методи створення та використання захищених паролів для програмного забезпечення підприємства;

– сформулювати основні вимоги та поставити задачу розробки системи керування паролями;

– проаналізувати безпечні методи управління інформацією та виявити вразливості існуючих менеджерів паролів;

– розробити алгоритм безпечного шифрування паролів із можливістю зворотного дешифрування;

– спроєктувати архітектуру програмної захищеної системи керування паролями;

– визначити засоби реалізації захищеної системи керування паролями;

– сформувати специфікацію вимог до програмної системи керування паролями;

– реалізувати програмну систему керування паролями на підприємстві;

– провести тестування та оцінку надійності впровадженої системи керування паролями.

1 ДОСЛІДЖЕННЯ ПРЕДМЕТНОЇ ОБЛАСТІ. ПОСТАНОВКА ЗАДАЧІ

1.1 Дослідження сучасних менеджерів паролів та аналіз “сильних” паролів

У сучасному цифровому світі кожен користувач інтернету має десятки, а то й сотні онлайн-акаунтів – від соціальних мереж і електронної пошти до банківських сервісів та платформ електронної комерції. Управління великою кількістю облікових даних стає дедалі складнішим завданням. Використання одного пароля для різних сервісів значно підвищує ризик компрометації даних, тоді як постійне запам’ятовування унікальних паролів для кожного облікового запису є практично неможливим. Тому вирішенням проблеми стали менеджери паролів[1].

Менеджер паролів-це інструмент, призначений для безпечного зберігання паролів для різних онлайн-акаунтів та управління ними[2]. Діючи як цифровий сейф, він зберігає ваші облікові дані для входу, а також інші конфіденційні дані, такі як інформація про кредитну картку та важливі файли[3]. За допомогою одного головного пароля ви можете отримати доступ до збережених паролів і навіть створити надійні унікальні паролі для різних облікових записів. Це не тільки спрощує процес входу, але й значно покращує вашу безпеку в Інтернеті та зменшує ризик злому паролів.[4] Такі дані, як імена користувачів і паролі, надійно зберігаються і можуть автоматично заповнюватися на вебсайтах для зручності. Ці інструменти також генерують надійні, випадкові паролі та можуть синхронізуватися на різних пристроях, забезпечуючи безпеку і доступність ваших облікових даних, де б ми не були, зберігаючи при цьому надійне шифрування для захисту від несанкціонованого доступу.

Вибір правильного менеджера паролів може мати вирішальне значення для захисту ідентичності в Інтернеті. Хороший менеджер паролів не тільки спрощує цифрове життя, але й зміцнює безпеку в Інтернеті.

Він повинен мати надійний захист , та шифрування, це включає багатофакторну автентифікацію (MFA), біометричні дані та методи шифрування

(наприклад AES-256, ChaCha20 або Twofish). Ці функції мають вирішальне значення для захисту паролів та іншої конфіденційної інформації від несанкціонованого доступу. Хороший менеджер паролів повинен пропонувати безпечний спосіб зберігання та автоматичного заповнення даних кредитних карток, що робить онлайн-покупки зручнішими та гарантує, що ваша фінансова інформація залишається захищеною. Функції, які оцінюють надійність наявних паролів і генерують надійні, унікальні паролі, важливі для підтримання загального рівня безпеки. Це допомагає гарантувати, що паролі буде складно зламати. Також важливим моментом є отримання сповіщень, отримуючи сповіщення про витік наших даних, ми можемо вжити негайних заходів, наприклад, змінити скомпрометовані паролі, щоб захистити свої акаунти [5].

На даний момент є дескілька найбільш популярних сучасних менеджерів паролів, а саме : Keeper, RoboForm, NordPass, 1Password, LastPass та інші. Тепер розберемо кожен окремо[6].

Keeper, у величезному середовищі інструментів керування паролями Keeper міцно зарекомендував себе як вибір вищого рівня. Відомий своїми бездоганними протоколами безпеки та інтуїтивно зрозумілим інтерфейсом користувача, Keeper є синонімом надійності. Його відмінною рисою є підхід з нульовим знанням. Це означає, що перш ніж ваші дані потраплять на сервери Keeper, вони проходять шифрування на рівні пристрою, забезпечуючи додатковий рівень безпеки. Відмінною рисою надійного менеджера паролів є його можливості двофакторної автентифікації (2FA), і Keeper не розчарує. Від звичайних методів, таких як SMS, до більш просунутих опцій, таких як KeeperDNA, який пропонує біометричну автентифікацію за допомогою смарт-пристроїв, користувачі мають безліч варіантів вибору. Сумісність Keeper вражає завдяки спеціальним програмам для всіх основних платформ і безперебійним розширенням для різноманітних веб-браузерів. Ексклюзивні функції, такі як KeeperChat, безпечна система обміну повідомленнями, і аудит безпеки, який оцінює надійність пароля, ще більше підвищують цінність пропозиції. Перехід на Keeper – легкий вітер, завдяки

підтримці імпорту з кількох популярних менеджерів паролів і браузерів[7].

Плюси Keeper:

- широка сумісність з платформами;
- цілодобова підтримка в чаті;
- ексклюзивна функція безпечного обміну повідомленнями;
- кілька варіантів 2FA;
- щедрий безкоштовний пробний період.

RoboForm, який має давню репутацію, є відомим у спільноті керування паролями. Хоча він широко обслуговує підприємства, окремі користувачі також знаходять однакову цінність у його пропозиціях. Його зручний дизайн гарантує, що як новачки, так і експерти зможуть легко орієнтуватися в його функціях. Акцент на безпеці очевидний завдяки включенню 2FA та використанню розширеного шифрування AES-256 для захисту даних користувачів[8]. РобоФорм, доступний на різних платформах, рекомендує веб-додаток, який часто оновлюється, для користувачів комп'ютерів. Його набір розширень для браузера доповнює всі основні браузери, що робить його універсальним вибором. Такі функції, як галузевий стандартний генератор паролів, вхід одним клацанням миші та можливості безпечного обміну, роблять його важливим інструментом цифрової безпеки. Крім того, додаткова перевага моніторингу Dark Web гарантує, що користувачі будуть попереджені про потенційні витоки даних. Його економна ціна в поєднанні з функціями найвищого рівня робить RoboForm переконливим вибором для тих, хто шукає надійні рішення для керування паролями[9].

Плюси RoboForm:

- варіант самостійного або хмарного сховища;
- безпроблемна функція входу одним клацанням миші;
- безпечний пароль і можливості спільного доступу до папок;
- дуже доступна цінова модель;
- моніторинг Dark Web для підвищення безпеки.

NordPass є свідченням відданості бренду безпеці в Інтернеті. Які набори

NordPass Окремо стоїть вибір шифрування XChaCha20, яке не тільки є ефективним, але й пропонує додатковий рівень захисту завдяки подвійній довжині ключа. У поєднанні з архітектурою з нульовим знанням це забезпечує максимальну безпеку даних користувачів[10]. В основі лежить простота використання NordPass. Будь то доступ до сховища за допомогою біометричних даних, традиційного головного пароля чи спрощеного процесу 2FA, NordPass забезпечує безпроблемний досвід. Його сумісність поширюється на основні платформи з розширеннями браузера, доступними для Chrome, Firefox, Safari тощо. Унікальні функції, як-от сканер витоку даних і офлайн-режим, а також зручні для користувача папки з даними та можливості оптичного розпізнавання символів ще більше підвищують позиції компанії на ринку. З конкурентоспроможною ціною, надійною безкоштовною версією та зобов'язанням постійної безпеки, NordPass є гідним суперником у царині керування паролями.

Плюси NordPass:

- розширена модель шифрування XChaCha20;
- економічні плани передплати;
- багаторівневий процес автентифікації;
- комплексний сканер втрат даних;
- підтримка автентифікації за допомогою пароля[11].

1Password дозволяє безпечно ділитися та отримувати доступ до конфіденційних даних з будь-якого місця, забезпечуючи чудову гнучкість для віддаленої роботи. Я міг встановити дозволи на доступ для певних осіб, що, на мою думку, було необхідним для підтримки безпеки. Інструмент допоможе вам увійти у свій обліковий запис, переглядати та редагувати елементи з будь-якого браузера. Це також дозволяє відстежувати звіти про використання, допомагаючи відстежувати, як співробітники взаємодіють з даними[12].

LastPass є одним із найпопулярніших менеджерів паролів. Цим інструментом користується понад 33 млн людей у всьому світі. LastPass пропонує

один із найкращих способів генерувати безпечні випадкові паролі, які допоможуть вам захиститися від хакерів. Мені було корисно легко зберігати свої цифрові записи та отримувати доступ до них. Фактично, він ідеально підходить для безпечного обміну паролями з іншими та дозволяє родині чи друзям отримати доступ до вашого облікового запису в надзвичайних ситуаціях. Він чудово підходить для використання на Windows, Мак, Androidі такі веб-переглядачі, як Chrome, Safari та Edge [13,14].

Перед тим як проводити аналіз сильних паролів , варто зазначити що не слід використовувати дуже часто використовувані паролі, які розповсюдженні по світу наглядно показані на рисунку 1.1

Рисунок 1.1 – Топ 10 найпоширеніших паролів за 2023 рік.

Тепер перейдемо до аналізу «сильних» паролів. «Сильний» пароль - це такий пароль, який важко вгадати людині або програмі. Сильний пароль складається щонайменше з 16-18 символів і містить верхні та нижні літери, цифри та символи. Сильні паролі також дотримуються найкращих практик паролів, щоб вони ніколи не містять особистої інформації, словнику чи фрази, і ніколи не використовуються повторно в декількох облікових записах[15]. Оскільки мета пароля - забезпечити доступ до ресурсів лише авторизованим користувачам, пароль, який легко вгадати, є ризиком для кібербезпеки. Коли люди створюють облікові дані для входу в систему, вони часто йдуть проти своєї мети, створюючи пароль, який легко запам'ятовується. Вони обирають свої імена, номери телефонів, дні народження або навіть саме слово «пароль», яке протягом багатьох

років було найпоширенішим паролем. Зараз, коли багато політик безпеки вимагають додавання цифри, найпоширенішим паролем є «password1». Натомість найкращим способом запобігти витоку даних, виявити крадіжку або інший несанкціонований доступ до приватної інформації є створення складних, довгих паролів[16]. Надійний пароль має складатися, щонайменше, з 18 символів. Чим довшим є пароль, тим він надійніший. Довші паролі складніше зламати, навіть якщо кіберзлочинці використовують сучасні високотехнологічні інструменти. Надійний пароль повинен складатися з комбінації великих та малих літер, цифр та спеціальних символів. Завдяки цьому збільшується кількість можливих комбінацій, а це, в свою чергу, значно ускладнює підбір чи злам паролю. Також можна створювати мнемонічний пароль для запам'ятовування, який буде містити лише малі літери, наприклад комбінацію випадкових слів, проте слід переконатися, що такий пароль має велику довжину. Також важливим пунктом є унікальність, кожен пароль повинен бути унікальним і використовуватись лише для одного облікового запису. Повторне використання паролю для доступу до кількох облікових записів збільшує ризик крадіжки даних, оскільки хакер зможе скористатися отриманими реєстраційними даними для доступу до кількох облікових записів.Крім цього регулярна зміна паролів сприяє зміцненню захисту ваших облікових записів. Рекомендовано змінювати ваші паролі через кожні 3-6 місяців. Однак, після зламу паролю або виявлення підозрілих дій у вашому обліковому записі, слід негайно змінювати дані для входу.

Також висока ентропія, паролі з високою ентропією набагато важче підібрати або зламати, оскільки вони містять велику кількість можливих комбінацій. Обидва типи паролів: комбінований (наприклад, DS%ty^dn$u48/*uD) та мнемонічний (наприклад, Packet-Garden-Mountain-Firetruck) — мають високий рівень ентропії. Нетипові слова та фрази, тобто, Потрібно уникати використання типових слів та фраз в паролях. Хакери полюбляють проводити Словникові Атаки (Dictionary Attacks), під час яких вони намагаються підібрати паролі за допомогою бази даних найбільш поширених слів та фраз, які використовуються в паролях. Не

мало важливим пунком є те що ніколи не можна використовувати під час створення паролю персональні дані, такі як імена, дати народження або інформацію з платіжної картки. Хакери часто проводять збір інформації про свою майбутню ціль, щоб мати вищі шанси вгадати пароль. Також можлива рекомендація це не використання в паролях послідовних літер або цифр, таких як «abcd» або «1234», оскільки таку комбінацію легко підібрати. Хакери використовують високотехнологічні програми для визначення та зламу паролів, і ряд послідовних символів буде визначено за лічені секунди[17]. Для кращої наглядності , на рисунку 1.2 показано наскільки впливає на час взлому складний пароль.

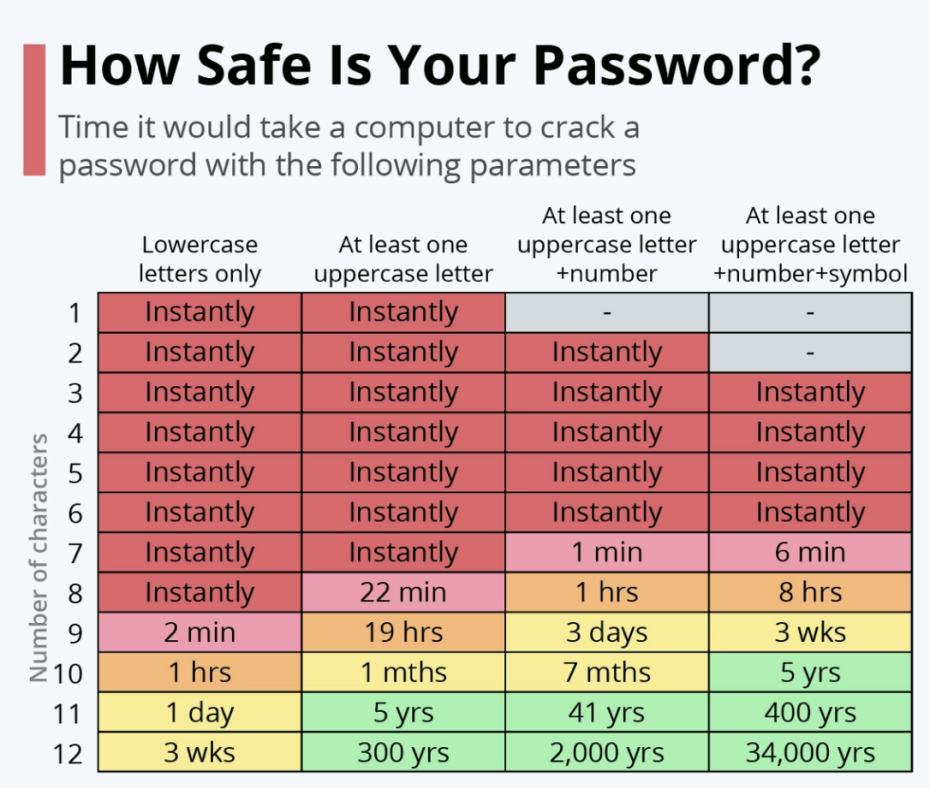


Рисунок 1.2 – Відповідність складності паролю до часу його взлому[18]

Як видно з рисунка 1.2, додавання до пароля навіть одного додаткового символу — цифри, великої літери чи спеціального знака — збільшує кількість можливих комбінацій у геометричній прогресії. Це істотно ускладнює використання методів грубої сили та словникових атак, які є одними з

найпоширеніших у арсеналі зловмисників. Водночас таблиця наочно демонструє, що короткі або одноманітні паролі, незалежно від їхнього складу, зламуються миттєво.

1.2 Дослідження методів протидії загрозам витоку конфіденційної інформації на підприємстві

Основною загрозою для інформаційної безпеки будь якого підприємства є витік конфіденційної інформації. Витік даних — це ситуація, коли конфіденційна інформація стає доступною публічно без згоди її власника.[19] Для того щоб розібратися як уникати та протидіяти цьому , спершу потрібно зрозуміти першопричини, та усунути їх.

Причини і види вразливостей систем безпеки:

– недосконале ПЗ, інша техніка;

– деякі процеси роботи системи неповноцінні;

– робота з інформаційною системою відбувається в складних експлуатаційних умовах.

Вразливості не завжди з’являються навмисно. Їх класифікація передбачає вразливості, які можуть бути випадкового або об’єктивного характеру. Щоб звести загрози втрати, крадіжки, зміни інформаційних даних до мінімуму, потрібно ліквідувати або мінімізувати вплив слабких місць в системі безпеки.

Приклади випадкових – ненавмисних загроз:

– неполадки в роботі апаратури;

– помилки, збої в ПЗ;

– помилки в діях персоналу або працівників, які працюють в системі;

– форс-мажори, викликані діями стихій, природними факторами;

– проблеми через постійні перебої електроенергії.

Причини несанкціонованого вторгнення бувають різними. Хакерами, що

мають недобросовісні мотиви, часто можуть бути люди з персоналу, користувачі інформаційного ресурсу, конкуренти або наймані фахівці. Мотивом може бути бажання збагатитися чужим коштом. Конкуренти іноді намагаються скомпрометувати суперника крадіжкою конфіденційної інформації. Співробітники компанії, яких звільнили за ті чи інші вчинки, можуть помститися колишньому роботодавцю. Таким чином, причин злочинної діяльності буває багато. Завдання інформаційної безпеки – запобігти діям зловмисників, зупинити їх на ранньому етапі проникнення в систему. Для отримання оптимального результату варто звернутися до фахівців, які позитивно зарекомендували себе в цій сфері діяльності, мають бездоганну репутацію. Тоді інформаційні системи будуть гарантовано в безпеці.[20]

Існують основні вирішення даного питання, які допомагають уникнути витоку конфіденційної інформації.Комплексна політика безпеки. Підтримання безпеки конфіденційних даних завжди має починатися зі створення комплексної політики безпеки, яка охоплює всі основи. Сьогодні кібербезпека є настільки складною темою, що якщо у компанії немає добре спланованої стратегії, є велика ймовірність, щось проґавити. І тоді, лише одна маленька помилка може відкрити доступ зловмисникам до цінної інформації, як-от персональні дані (PII).Хороша політика безпеки вимагає розробки комплексної стратегії управління ризиками. Це включає виявлення та оцінку потенційних загроз безпеці та ризиків, визначення їхньої ймовірності та потенційного впливу, а також впровадження відповідних превентивних заходів. Іншим важливим стратегічним елементом є створення структури управління безпекою для нагляду за впровадженням і дотриманням політики безпеки. Це включає визначення ролей і обов’язків, розробку політик і процедур, а також впровадження засобів контролю для забезпечення ефективного управління ризиками безпеки.

Щось завжди може піти не так, незалежно від того, скільки запобіжних заходів було вжито. Саме тому, швидке та ефективне реагування на атаку може значно змінити кінцевий результат інциденту. Як наслідок, мати добре розроблені

плани реагування на інциденти – як технічно, так і з точки зору спілкування з клієнтами, діловими партнерами та широкою громадськістю – так само важливо, як і превентивні заходи. Також дуже важливо навчати та підвищувати обізнаність працівників щодо ризиків безпеки, найкращих практик і політики, щоб забезпечити безпеку конфіденційної інформації. Співробітники можуть ненавмисно стати слабкою ланкою в безпеці організації, а недостатня обізнаність про ризики безпеці може призвести до порушень безпеки. Важливо пояснювати потенційні ризики безпеці та виконувати вправи для їх виявлення. Це охоплює заохочення користувачів дізнатися про поширені типи кібератак і вразливості, а також пояснення потенційних наслідків порушень безпеки. Співробітники можуть стати більш пильними та краще підготовленими для виявлення потенційних загроз, якщо виникають ризики безпеці, знижуючи ризик успішних атак. Варто демонструвати співробітникам, як підтримувати кібербезпеку на робочому місці та за його межами, особливо якщо в компанії поширена віддалена робота. Це включає поради щодо надійних паролів, багатофакторної автентифікації, безпечного зв'язку, безпечного перегляду вебсторінок і захисту даних. Співробітники можуть допомогти запобігти порушенням безпеки та зменшити наслідки успішних атак, отримавши чіткі інструкції щодо того, як реагувати на можливі ризики для безпеки.

Необхідно створювати загальну культуру безпеки. Це включає підкреслення того, що безпека є відповідальністю кожного, від генерального директора до звичайного співробітника, і заохочення відкритої та прозорої культури. Співробітники з більшою ймовірністю сприймуть безпеку серйозно і стануть активними учасниками її підтримки, якщо в організації створена культура безпеки.

Жорсткий контроль доступу. Перш ніж компанія зможе ефективно захистити свою цінну конфіденційну інформацію та персональні дані (PII), варто визначити, що це таке, де вони знаходяться та хто має мати до них доступ. Оскільки конфіденційна інформація може надходити в різних формах і типах, які

не завжди такі очевидні, як номери кредитної картки чи соціального страхування, це не завжди є легким завданням. Саме тому, такі інструменти, як програмне забезпечення для запобігання втрати даних, може бути надзвичайно корисним. Дотримання принципу найменших привілеїв (PoLP). PoLP – це концепція інформаційної безпеки, яка стверджує, що користувач або організація повинні мати доступ лише до необхідних даних, ресурсів і програм для виконання завдання. У цьому випадку варто переконатись, що доступ до даних надається лише тим, кому вони дійсно потрібні. Наприклад, якщо комусь не потрібно редагувати дані, варто надати їм лише дозволи на читання. Цей принцип не тільки запобігає несанкціонованому доступу, але й допомагає краще контролювати авторизований доступ[21]. Використання DLP-рішення Це рішення для запобігання витоку конфіденційних файлів за межі мережі компанії. DLP-системи аналізують усю вхідну та вихідну інформацію та за допомогою неї виявляють підозрілі операції та ризики[22]. Відстеження доступу до всіх конфіденційних даних. Програмне забезпечення для запобігання втрати даних, наприклад, дозволяє не лише визначати конфіденційні дані, але й реєструвати всі спроби доступу. Наприклад, сповіщення про будь-які підозрілі дії, такі як доступ у неробочий час або численні спроби доступу до різних фрагментів конфіденційної інформації за короткий проміжок часу. Шифрування даних. З сучасною швидкістю процесора немає виправдання уникати шифрування. Раніше шифрування інформації займало багато ресурсів, тому його часто уникали. Тепер, слід шифрувати інформацію, коли це можливо, особливо коли компанія має справу з конфіденційними даними. Хоча більшість Інтернет-технологій, таких як електронна пошта чи Інтернет, зараз шифрують інформацію під час передачі, варто піти далі та запровадити власну схему шифрування. Два рівні шифрування не зашкодять даним або ресурсам. Однак, такий підхід гарантує, що навіть якщо конфіденційність транзитної технології буде порушено в результаті атаки, дані залишаться в безпеці від зловмисних хакерів. Використання заходів, які запобігають непотрібному доступу до незашифрованих даних, якщо

користувачам потрібен лише тимчасовий доступ до незашифрованої інформації, як-от перевірка особистих даних клієнта під час дзвінка. Наприклад, можна заборонити копіювати та вставляти (CTRL+C, CTRL+V) персональну інформацію (PII) будь-де, крім необхідних для роботи середовищ. Застосування шифрування щоразу, коли дані повинні залишити зону впливу компанії. Ніколи, наприклад, не варто дозволяти користувачам копіювати незашифровану конфіденційну інформацію на будь-який зовнішній носій, такий як зовнішні жорсткі диски або USB-накопичувачі. Якщо для їхньої роботи потрібні такі копіювання, DLP дозволяє застосовувати шифрування кожного разу, коли виконується така операція[23].

Крім вищезазначеного також існують і антивірусні програми та міжмережеві екрани але вони все ще залишаються першою лінією оборони для системи інформаційної безпеки. Вони не лише блокують шкідливі програми та віруси, але й захищають мережу від несанкціонованого доступу. Робота міжмережевого екрану схожа на мережевий фільтр, який цілодобово аналізує внутрішній трафік компанії. І на відміну від звичайного підключення до інтернет-мережі, сервіс аналізує трафік компанії 24/7 та швидко усуває спроби витоку даних.

Також варто згадати про ISMS. Адже система пропонує комплексний підхід до захисту інформації на підприємстві. Вона допоможе реалізувати ефективні стратегії для виявлення, оцінки та зменшення ризиківх. ISMS охоплює не лише технологічні рішення, а й організаційні заходи, які передбачають різні аспекти безпеки — від захисту мережі до управління інцидентами. Саме це дозволяє створити надійний бар'єр проти кіберзагроз[24].

Не менш важливими є й хмарні технології. Вони забезпечують надійне зберігання інформації та доступ до неї з будь-якого місця, а також надають додаткові засоби захисту, такі як шифрування та резервне копіювання. В комбінації з ISMS, хмарні технології можуть значно підвищити рівень безпеки вашого підприємства, дозволяючи ефективно управляти інформаційними

ресурсами та знижувати ризики втрати даних[25].Інформаційна безпека підприємства вимагає комплексного підходу. Для ефективного захисту даних необхідно постійно моніторити інформаційні системи, регулярно оновлювати програмне забезпечення, а також не чекати, коли виникне загроза, а відразу впроваджувати нові технології.

1.3 Дослідження методів побудови захищеного пароля для програмного забезпечення на підприємстві

У корпоративному середовищі правильне створення та зберігання паролів є критично важливим аспектом інформаційної безпеки, оскільки ненадійні або скомпрометовані паролі можуть стати причиною несанкціонованого доступу до конфіденційних даних. Використання слабких паролів підвищує ризик атак методом перебору, що дозволяє зловмисникам отримати контроль над системами підприємства. Крім того, повторне використання однакових паролів для різних облікових записів може призвести до ланцюгової компрометації, коли злам одного ресурсу відкриває доступ до інших. Впровадження багатофакторної автентифікації та використання шифрованих менеджерів паролів значно знижує ймовірність витоку інформації. Також важливо регулярно оновлювати політики безпеки та проводити навчання персоналу щодо методів створення та зберігання паролів. Таким чином, ефективне управління паролями сприяє захисту корпоративних ресурсів та мінімізує ризики кібератак.

Більшість людей не вміють підбирати випадкові паролі та схильні вказувати часто вживані слова, цифри та символи. Якими кмітливими ми б не були, більшість комп’ютерів із легкістю відгадують створювані людьми паролі. Надійність пароля забезпечує використання комбінацій літер, цифр і спеціальних символів. Для забезпечення високого рівня безпеки рекомендується використовувати генератори випадкових паролів, які створюють складні та

унікальні комбінації символів. Такі паролі важко вгадати або зламати за допомогою методів перебору. Прикладом такого інструменту є генератор паролів від Avast, який забезпечує створення надійних паролів для різних облікових записів. Приклад його роботи на рисунку 1.3.

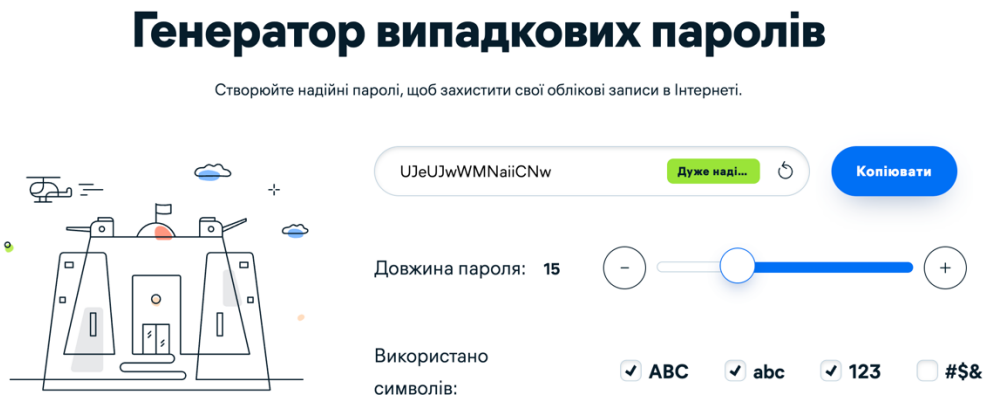


Рисунок 1.3 – Приклад генерації паролю[26]

Генератори безпечних паролів застосовують метод криптографічної ентропії (випадковості). Щоб створити надійний пароль, потрібно уникати найпоширеніших запитань і відповідей системи безпеки. Ніколи не використовувати той самий пароль для різних важливих облікових записів. Генератори видають паролі, які містять принаймні 16 символів, серед яких одна велика літера, одна цифра, один спеціальний знак і одна мала літера. Під час створення паролів люди часто використовують номери телефонів і соціального страхування, поштові індекси, номери документів, будинків і дні народження. Усе це легко відгадати.

Програма дає змогу вибрати довжину пароля та вказати критерії щодо символів, з яких буде створено надійний випадковий пароль. Можна встановити попередньо визначені або вибрані метрики Avast для генерування випадкового пароля, скопіювати його в буфер обміну та використати для будь-якого облікового запису в Інтернеті. Завдяки широкому вибору малих і великих літер, цифр і спеціальних символів автоматичний генератор паролів Avast видає

непередбачувані випадкові послідовності знаків.Зручний дизайн і здатність цього інструмента Avast миттєво створювати випадкові паролі роблять його зручним генератором надійних паролів для підприємств, облікових записів у соціальних мережах, електронної пошти й банківських рахунків. Кожен випадковий пароль генерується за допомогою надійних криптографічних алгоритмів локально у нашому браузері з використанням ресурсів процесора комп'ютера, тому жодні дані через Інтернет не передаються.

Після того як пароль згенерований , його потрібно зберегти в нашій програмі та не допустити того щоб він був втрачений , для цього існують деякі програмні рішення, а саме:

- хешування та соління паролю;
- безпечне його зберігання в базі даних;
- контроль та моніторинг безпеки паролів.

Хешування — це процес перетворення вхідних даних, таких як файл або пароль, у рядок символів фіксованого розміру, що є унікальним відбитком початкових даних. Це часто використовується в кібербезпеці для безпечного зберігання паролів і перевірки цілісності даних[27]. Для підвищення безпеки хешування можна впровадити кілька технік та найкращих практик:

- соління;
- розтягнення ключа**;**
- оновлення алгоритмів хешування**.**

Соління передбачає додавання випадкової величини (солі) до вхідних даних перед хешуванням. Сіль зберігається поруч з хешованим значенням. Ця техніка покращує безпеку, додаючи додаткову рандомізацію та складність до хешованого значення, роблячи його більш стійким до заздалегідь обчислених атак типу rainbow table. Це також допомагає захистити від атак, таких як словникові атаки і атаки грубою силою[28].

Розтягнення ключа — це техніка, що додає обчислювальну складність процесу хешування. Вона передбачає навмисне уповільнення алгоритму

хешування, роблячи атаки типу сила грубої сили та розгадування паролів обчислювально затратними та часозатратними. Важливо регулярно оновлювати використовуваний алгоритм хешування, оскільки з часом можуть бути виявлені нові вразливості. Використання останніх безпечних алгоритмів хешування забезпечує стійкість збережених хешованих значень до атак. Оновлення до новіших алгоритмів, таких як SHA-256, рекомендується для покращеної безпеки[29,30] Після того як згенерований пароль захешували його варто варто десь зберігати. У нашому випадку для програмного рішення зберігання паролю буде відбуватись у базі данних.

Окрім хешування та правильного зберігання, важливо моніторити використання паролів, щоб своєчасно виявляти загрози. Для цього в нашому програмному рішенні варто використоувати логування подій. Основною метою логування є надання інформації, що дозволить проаналізувати помилки чи проблеми у роботі програми. Достатньо уявити ситуацію, коли є дві інтегровані системи і на якомусь із етапів роботи виникає помилка. Для того, щоб з’ясувати на стороні якої з програм вона виникла, якраз і потрібні будуть логи. Це також може допомогти у локалізації помилки, що виникла та подальшого її швидкого усунення[31,32]. Логи нам зможуть додатково покращити наш захист паролів , оскільки будуть завжди оповіщати коли хтось захоче зайти в систему, це зменшить ризик несанкціонованого доступу.

1.4 Постановка задачі

Провівши попередній аналіз існуючих загроз, пов’язаних із витоками конфіденційної інформації, а також вивчивши методи, які використовуються для генерації, збереження та захисту паролів, було прийнято рішення про необхідність створення повноцінної системи керування паролями, що відповідатиме сучасним вимогам безпеки. Особливої актуальності така система набуває в умовах цифровізації бізнес-процесів підприємств, зокрема таких як

“DataGroup”, де щодня обробляються та передаються великі об’єми конфіденційної інформації. У сучасних реаліях просто наявності пароля недостатньо — він повинен бути надійно згенерований, безпечно збережений, а всі дії, пов’язані з ним, мають бути ретельно задокументовані й підконтрольні системному адміністратору. Система, яку буде реалізовано в межах цієї роботи, повинна вирішити комплекс завдань, які сьогодні є базовими для будь-якої безпечної інформаційної інфраструктури: від моменту створення пароля до його подальшого захищеного використання, аж до журналювання взаємодії з системою.

Система має дозволяти не просто зберігати паролі, а й робити це безпечно, із застосуванням механізмів шифрування та хешування. Вона також повинна забезпечити генерацію складних, криптографічно стійких паролів, логувати всі дії користувачів, реалізовувати безпечну авторизацію з використанням сучасних підходів зокрема, на базі JWT-токенів, а також унеможливлювати доступ до даних без наявності відповідного дозволу. JSON веб-токен (JWT) призначений для передачі підписаних «заявок» (claims) між службами (як зовнішніми, так і внутрішніми для вашого застосунку/сайту). «Заявки» — частина інформації, яку інші можуть переглядати та/або перевіряти, але не змінювати[33,34]. Кожен етап роботи користувача повинен супроводжуватись перевіркою прав доступу та зберігатись у вигляді запису в системі логування, що дозволяє згодом провести аудит дій, ідентифікувати спроби зловживань або атаки.

Сформульована задача передбачає створення повноцінного програмного забезпечення, яке стане внутрішнім інструментом для підприємства Його використання має не лише підвищити рівень безпеки всієї IT-інфраструктури, а й оптимізувати взаємодію між користувачами та адміністраторами, зробити процес керування обліковими записами контрольованим, стандартизованим та передбачуваним. У цьому контексті важливою є не лише функціональність, а й архітектурна правильність, стійкість до атак, зручність користування, а також можливість масштабування рішення в майбутньому. Система повинна бути

побудована на технологічній основі, яка дозволяє зручно та безпечно реалізовувати зазначені функції.

У зв'язку з цим було прийнято рішення для реалізації обрати стек технологій, який поєднує в собі надійність, гнучкість і масштабованість. У якості мови програмування обрано C# — сучасну, типізовану, об'єктно-орієнтовану мову, яка добре себе зарекомендувала у сфері розробки корпоративного ПЗ. C# — це мова, яка має сувору типізацію. У ній є автоматичний збір сміття — програмісту не потрібно дбати про ручне звільнення пам'яті. У мові згодом запровадили динамічне звязування, асинхронні методи, шаблони [35,36].

Більшість програм та сервісів для вебу, пов'язаних із продуктами Microsoft, написано на C#. Відомий сайт stackoverflow.com написаний на C# із використанням фреймворку.NET. Подібним методом зроблені сайти Microsoft і Dell. Як фреймворк — ASP.NET Core MVC, який забезпечує зручний поділ на рівні контролерів, представлень і моделей, що значно спрощує структурування логіки додатку, робить його підтримку прозорою, а розширення — швидким. ASP NET це фреймворк для розробки веб-додатків, створений корпорацією Microsoft. У світі веб-розробки ASP.NET займає важливе місце, забезпечуючи розробникам інструменти для створення масштабованих і сучасних веб-додатків. Розробники можуть зосереджуватися на бізнес-логіці, а не на деталях інфраструктури, завдяки чому процес розробки стає більш ефективним.[37,38] Крім того, ASP.NET Core має потужну екосистему для реалізації безпеки — від аутентифікації до авторизації, а також безперешкодну інтеграцію із сучасними бібліотеками шифрування, логування та роботи з JWT.

Для зберігання даних обрано систему управління базами даних PostgreSQL. PostgreSQL – це система управління реляційною базою даних (СУБД) з відкритим вихідним кодом, яка використовує та розширює мову запитів SQL для роботи з даними. Вирізняється чудовою масштабованістю, підтримкою нестандартних типів даних, легкою інтеграцією зі сторонніми інструментами та гарною підтримкою зі сторони спільноти, що позитивно впливає на екосистему

PostgreSQL в цілому.

PostgreSQL має потужні механізми безпеки, включаючи підтримку SSL, сильне шифрування та детальний контроль доступу, а також архітектуру, яка дозволяє легко додавати нові функції без зміни основного коду[39,40]. Це один із найпотужніших сучасних інструментів, який має відкритий код, гнучку схему конфігурацій, повноцінну підтримку транзакцій, а також високий рівень захисту та контроль доступу. PostgreSQL дозволяє зручно оперувати структурованими даними, використовувати індекси, розширювати функціонал за допомогою розширень та процедур, і водночас гарантує цілісність та надійність інформації. Ця СУБД чудово підходить для побудови безпечних систем, оскільки дозволяє реалізовувати контроль доступу до таблиць, перегляд історії змін, та організовувати ізольовані простори даних для різних типів користувачів.

Основні функції які повинні бути імплементовані в нашу систему керування:

- створення сильних паролів;
- зберігання паролів та їх шифрування;
- хешування паролів;
- логування дій користувачів;
- безпечна авторизація;
- використання сучасних стандартів безпеки.

Система повинна також забезпечувати регулярне оновлення компонентів для усунення вразливостей, а адміністратори — мати змогу оперативно реагувати на інциденти безпеки. Особливу увагу слід приділити сегментації прав доступу, щоб кожен користувач мав лише ті дозволи, які необхідні для виконання його функцій.

2 ПРОЕКТУВАННЯ ПРОГРАМНОЇ ЗАХИЩЕНОЇ СИСТЕМИ КЕРУВАННЯ ПАРОЛЯМИ

2.1 Алгоритми роботи системи

У процесі розробки програмної захищеної системи керування паролями я прийняв рішення застосувати гібридний підхід, що поєднує стандартні алгоритми шифрування, зручний інтерфейс для взаємодії користувача та належне логування дій усередині системи. Алгоритми роботи цієї системи умовно можна поділити на декілька груп: аутентифікація користувача, генерація токенів доступу, створення, шифрування, збереження, редагування та видалення документів, а також логування дій.

Основна мета системи — забезпечити безпечне зберігання, передачу та обробку конфіденційних даних користувача. На цьому етапі формулюються загальні алгоритми, що ляжуть в основу подальшого програмного втілення.

Першим етапом є реєстрація та авторизація користувача. Алгоритм реєстрації перевіряє унікальність логіна, хешує пароль за допомогою алгоритму BCrypt, та зберігає облікові дані у базу. Це не просто шифрування, а криптостійкий хеш, який важко зламати навіть при повному доступі до БД. Таким чином, жоден пароль не зберігається у відкритому вигляді. Приклад алгоритму реєстрації видно на рисунку 2.1. Після успішної авторизації, система видає користувачеві токен доступу, згенерований за допомогою JWT (JSON Web Token). Цей токен містить ім'я користувача та строк дії, він підписується секретним ключем, щоб уникнути підробки. Саме цей токен використовується в усіх наступних запитах до API.

Рисунок 2.1 – Алгоритм роботи реєстрації

Також крім реєстрації , є ще алгоритм входу в систему , на основі логіну та паролю. Цей алгоритм перевіряє чи наші дані які ми вводимо при вході в систему вірні, якщо так то генерується пароль та відбувається вхід в систему, та це логується одразу в систему, якщо ні то видається помилка , та також логується в системі. Приклад роботи даного алгоритму наведено на рисунку 2.2.

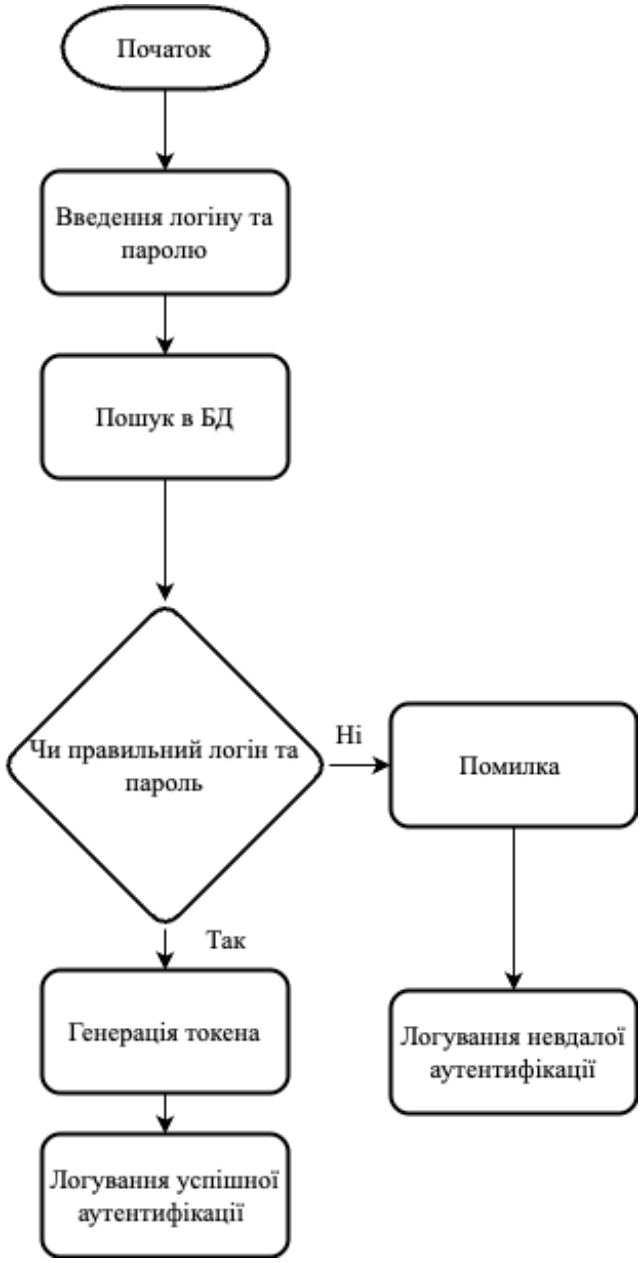


Рисунок 2.2 – Алгоритм роботи входу в систему

Далі користувач взаємодіє з основною функціональністю системи — створенням, редагуванням та збереженням файлів та записів підприємства. Кожен запис включає в себе назву цього запису, текст який може бути в цьому записі, а також опціонально прикріплений файл. Для шифрування паролів було обрано симетричний алгоритм AES (Advanced Encryption Standard), що працює з унікальним ключем та IV, збереженим у безпечному середовищі виконання. Тому навіть при витоку бази даних, дані залишаться зашифрованими.

Створення запису відбувається за наступним алгоритмом: користувач вводить дані, система шифрує їх, формує об'єкт і додає його до бази. Усі дії логуються з прив'язкою до користувача. Це забезпечує прозорість та можливість подальшого аудиту. Редагування документів має подібний алгоритм: перед поданням змін дані дешифруються, редагуються користувачем, шифруються знову і оновлюються в базі. Це унеможливлює зберігання незашифрованих даних у будь-який момент часу.

Також в цій системі спроектував логування подій. Логування — це не просто технічна частина, а справжній “чорний ящик” усієї системи. Кожна дія користувача — від входу до редагування чи видалення документу — фіксується автоматично. У логах зберігається все: хто саме виконав дію, що саме зробив і коли. Це дає змогу аналізувати поведінку користувачів, та вчасно виявляти підозрілі або потенційно небезпечні дії. У системі використовую комплексний набір алгоритмів, що покриває весь цикл обробки конфіденційної інформації — від введення користувачем до шифрування, зберігання, перегляду, редагування та видалення. Система функціонує з урахуванням сучасних вимог до захисту даних, забезпечуючи баланс між безпекою, зручністю користування та гнучкістю функціональності.

2.2 Проектування алгоритмів функцій системи

Мої основні функції захисту будуть розприділені та відокремленні від основної програми в папці Sevices. Там я зберу та розроблятиму наступні функцій:

- encrypt;
- decrypt;
- generateToken;
- log.

На рисунку 2.3 видно структуру функцій захисту моєї системи.

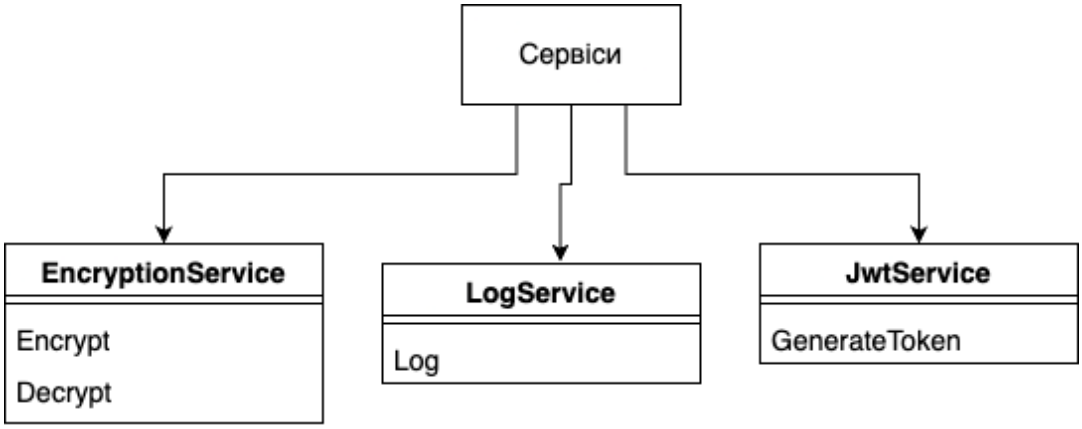


Рисунок 2.3 – Структура функцій захисту системи.

Сервіс EncryptionService реалізує логіку симетричного шифрування на основі алгоритму AES. Основна мета цього сервісу — забезпечити збереження чутливих текстових даних (назва, вміст документа, паролі) у зашифрованому вигляді. Саме цей підхід дозволяє гарантувати, що навіть при компрометації бази даних зловмисник не отримає доступу до суті інформації. Вибір симетричного шифрування зумовлений його швидкодією та ефективністю в рамках одностороннього використання (шифрування/дешифрування лише на сервері).

Основні функції:

- генерація ключа (_key) та ініціалізуючого вектора (_iv) через SHA256;
- шифрування тексту (метод Encrypt);
- дешифрування (метод Decrypt).

Перед кожним шифруванням система створює екземпляр алгоритму AES, ініціалізує його з уже збереженим ключем та вектором ініціалізації, після чого здійснює криптографічне перетворення тексту в потік байтів. На виході отримується рядок у форматі Base64, що легко зберігається в текстових полях бази даних. Приклад роботи алгоритму шифрування наглядно показано на рисунку 2.4

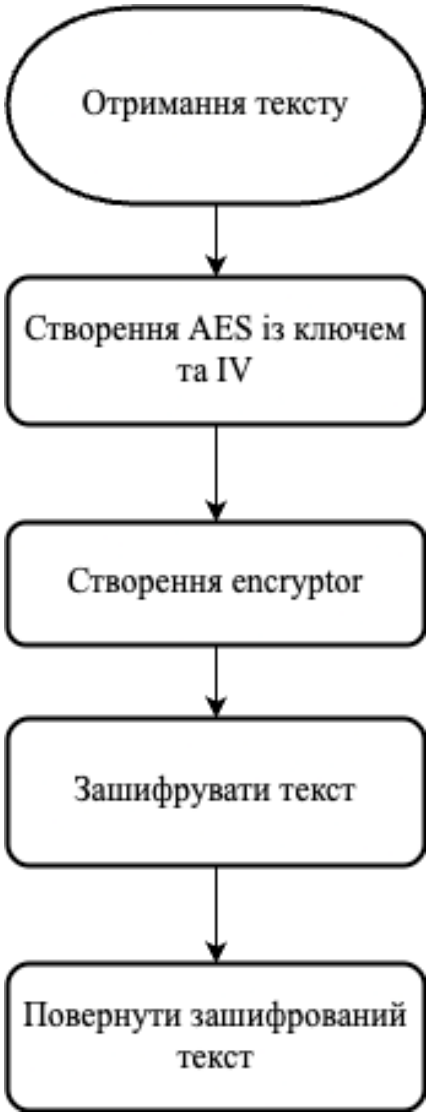


Рисунок 2.4 Алгоритм роботи методу шифрування

Процес дешифрування є інверсивним до шифрування , тобто, спершу рядок Base-64 конвертується назад в байти , потім через AES- дешифрування , текст відновлюється до початкової форми.

Далі йде Сервіс JwtService. Він виконує генерацію маркерів доступу (токенів) для автентифікованих користувачів. Кожен токен містить закодовану інформацію про користувача (claim) та підписаний з використанням симетричного секретного ключа. Це дозволяє серверу ідентифікувати користувача без необхідності зберігати сесію, що значно спрощує архітектуру та покращує масштабованість. Основні кроки роботи цього алгортму показані на рисунку 2.5.

Рисунок 2.5 Алгортм створення JWT-токену.

І наостанок LogService. Він реалізує центральну систему логування дій .Це дозволяє проводити безпековий аудит на основі збережених логів, аналіз активності та відстужувати підозрілі дії в системі. Приклад роботи алгоритму логування наведено на рисунку 2.6.

Рисунок 2.6 – Алгоритм роботи логування.

Ця діаграма на рисунку 2.6 добре ілюструє просту, але надійну логіку логування. Уся суть полягає в тому, що кожна дія одразу фіксується у вигляді події, яка через контекст бази даних зберігається у відповідну таблицю. Завдяки цьому можна у будь-який момент переглянути історію активностей користувачів та вчасно зреагувати на потенційні загрози.

2.3 Проєктування архітектури програмної захищеної системи керування паролями

Під час розробки даної системи я вирішив будувати її архітектуру на базі комбінованого підходу, що поєднує класичну модель MVC (Model-View-Controller) з сервісно-орієнтованою логікою (Service Layer), а також використовує принципи так званої “цибулинної” архітектури (Onion Architecture). Такий вибір

був зумовлений необхідністю створити гнучку, масштабовану та захищену систему, де чітко розмежовується логіка відповідальностей, і кожен модуль виконує строго визначену функцію. Крім того, мене цікавило не лише створити працюючий функціонал, а побудувати саме таку структуру, яку легко буде підтримувати, змінювати, адаптувати до нових вимог або перенести на інші середовища.

Основна ідея, яку я вклав у побудову архітектури, полягала в тому, щоб зовнішні рівні системи — такі як користувацький інтерфейс чи веб-контролери — ніколи безпосередньо не взаємодіяли з базою даних або внутрішніми механізмами обробки. Усе, що стосується шифрування, автентифікації, генерації токенів, логування чи збереження інформації, винесено у спеціалізовані сервіси. Саме сервіси виступають як незалежні компоненти, які можуть бути легко замінені, протестовані або розширені без зміни контролерів. Вони, у свою чергу, працюють із контекстом бази даних через чітко визначені інтерфейси, що дозволяє тримати внутрішні деталі реалізації прихованими від верхніх шарів. Ясно відокремлюючи представницький рівень, сервісний рівень та рівень даних, я дотримувався принципів «внутрішніх кілець» Onion Architecture. Тобто, вся логіка, яка є критично важливою (наприклад, шифрування чи логування), знаходиться в центрі архітектури, а зовнішні елементи (наприклад, контролери чи інтерфейс користувача) мають доступ до неї лише через відповідні інтерфейси. Це не лише забезпечує кращу безпеку й контроль над даними, але й унеможливлює випадкові або неконтрольовані виклики з боку інтерфейсного шару.

Окрім того, використання цієї моделі дозволяє мені максимально ізолювати залежності. Жоден контролер не залежить від конкретної реалізації сервісу, і навіть контекст бази даних підключається лише через відповідні інтерфейси. Це дає змогу у будь-який момент реалізувати, наприклад, іншу систему логування (наприклад, у вигляді мікросервісу), не змінюючи жодного рядка в бізнес-логіці або контролерах. А завдяки тому, що всі сервіси виконуються асинхронно, система залишається стабільною навіть під навантаженням. Спроектувавши

архітектуру для мого підприємства , створив діаграму на які візуально зображена архітектура. Діаграма на рисунку 2.7.

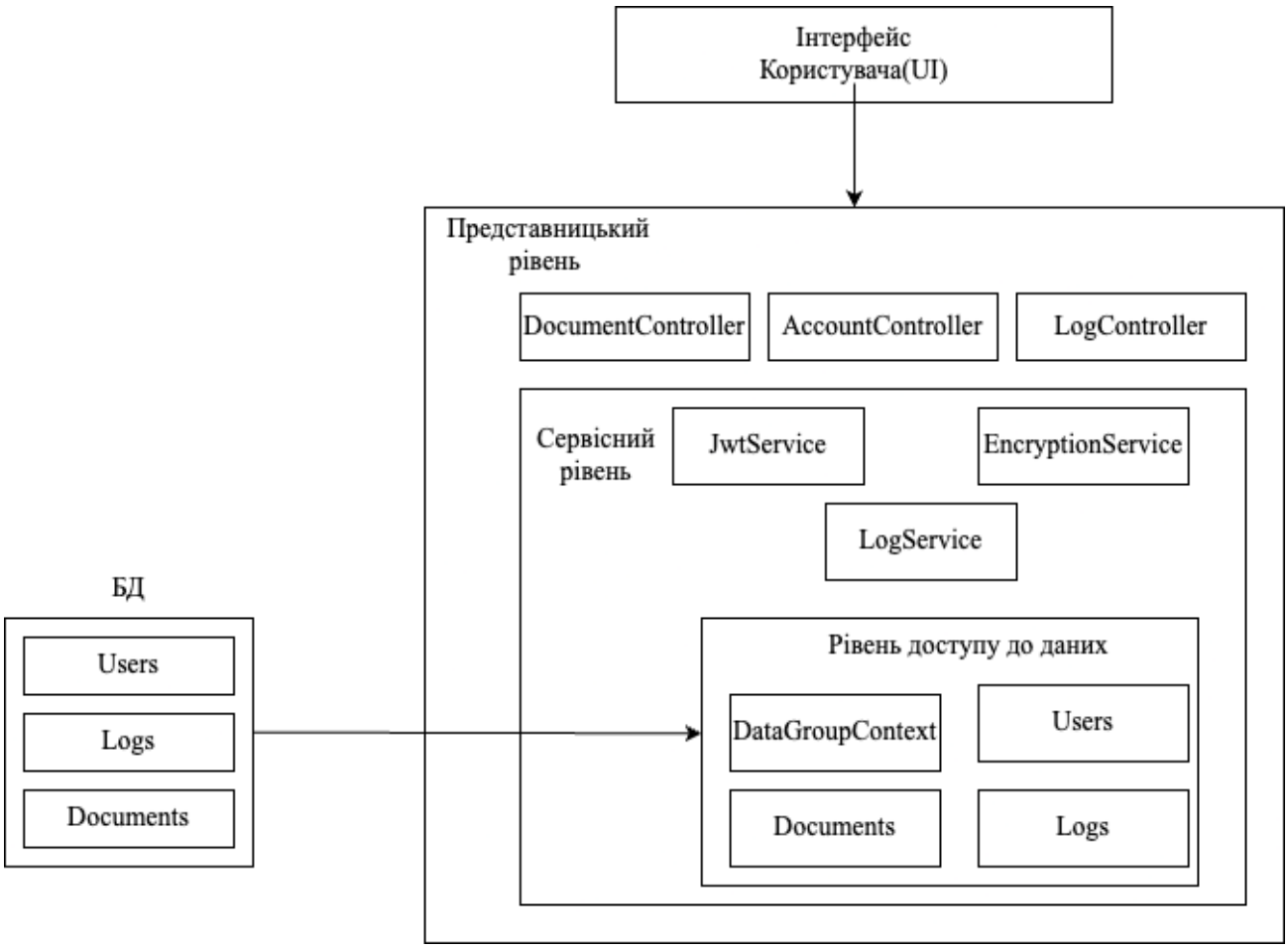


Рисунок 2.7 – Діаграма загальної архітектури системи

Діаграма, що зображена на рисунку, ілюструє логічну архітектуру захищеної системи керування паролями, яка була розроблена відповідно до сучасних принципів побудови програмного забезпечення. В основі структури лежить багаторівневий підхід, що дозволяє розділити відповідальності між різними компонентами системи. Це забезпечує не лише зручність підтримки та тестування, але й підвищену безпеку за рахунок ізоляції критично важливих частин логіки.

У верхній частині діаграми знаходиться інтерфейс користувача — UI (User Interface), який виконує функцію точки входу в систему для кінцевого

користувача. Це можуть бути сторінки авторизації, перегляду або створення документів, у вигляді стандартних MVC-представлень (Views), з якими безпосередньо працює користувач. Саме з інтерфейсу користувач ініціює запити, що надсилаються на сервер.

Після отримання запиту керування переходить до представницького рівня, який у цій системі реалізовано за допомогою контролерів: DocumentController, AccountController та LogController. Кожен контролер виконує окрему роль. Наприклад, DocumentController відповідає за CRUD-операції з документами, AccountController — за обробку авторизації та реєстрації користувачів, а LogController — за надання можливості перегляду логів. Контролери виконують роль посередника між інтерфейсом та внутрішньою логікою системи. Вони не займаються складною обробкою даних, а делегують її спеціалізованим сервісам.

Сервісний рівень є ядром бізнес-логіки системи. Саме тут знаходяться сервіси JwtService, EncryptionService та LogService, кожен із яких реалізує окрему функціональність. Сервіс JwtService відповідає за створення токенів доступу для авторизованих користувачів, реалізуючи сучасну модель безпечної автентифікації на базі JWT (JSON Web Token). Він формує токени з урахуванням часу дії, підпису та іншої метаінформації. Сервіс EncryptionService реалізує шифрування та дешифрування чутливої інформації з використанням симетричного алгоритму AES. Такий підхід гарантує, що навіть при фізичному доступі до бази даних ніхто не зможе прочитати вміст без ключа. Третій сервіс, LogService, забезпечує журналювання дій у системі, зберігаючи в базі інформацію про дії користувача, час, тип операції та інші деталі. Цей модуль критично важливий для безпеки, оскільки дозволяє виявляти підозрілі дії та виконувати аудит.

Нижче знаходиться рівень доступу до даних, який реалізовано за допомогою контексту Entity Framework Core під назвою DataGroupContext. Цей контекст відповідає за взаємодію з базою даних PostgreSQL і управляє трьома головними наборами даних — Users, Documents та Logs. Через ці сутності

здійснюється збереження та отримання відповідної інформації. Контекст інкапсулює всі запити до БД, дозволяючи сервісам працювати з абстракцією, а не безпосередньо з SQL. Уся структура об’єднана логікою взаємодії: запит від користувача потрапляє у відповідний контролер, який звертається до сервісу, а той, у свою чергу, при потребі взаємодіє з базою даних через DataGroupContext. Така послідовність забезпечує чіткий потік обробки інформації, де кожен компонент відповідає лише за свою частину, не знаючи про реалізаційні деталі інших частин системи.

Окремо варто звернути увагу на безпекову ізоляцію. Наприклад, користувач не має прямого доступу до сервісів — усі дії ініціюються виключно через контролери, які попередньо перевіряють токен авторизації та дані, що надходять. Усі сервіси впроваджуються через залежності (Dependency Injection), що дозволяє гнучко конфігурувати систему та легко замінювати компоненти, наприклад, у тестах або при масштабуванні. Уся логіка побудована так, що навіть при зміні способу збереження даних чи шифрування не потрібно змінювати контролери або інтерфейс — лише сервіс, що реалізує відповідну логіку.

Таким чином, діаграма демонструє не лише внутрішню будову системи, а й відображає сучасний підхід до розробки безпечного, масштабованого та підтримуваного програмного забезпечення. Це дозволяє ефективно організувати взаємодію між компонентами, спростити майбутній розвиток проєкту й забезпечити високий рівень захисту конфіденційної інформації.

2.4 Висновки

У другому розділі дипломної роботи я провів повноцінне проєктування захищеної програмної системи керування паролями, що охоплює всі важливі аспекти — від опису алгоритмів до архітектурного розподілу функціональності. Цей етап розробки став основою для практичної реалізації системи, дозволивши сформувати цілісне бачення її внутрішньої логіки, безпекових механізмів та

структурної організації.

У підпункті 2.1 були докладно розглянув алгоритми, які забезпечують повний цикл взаємодії користувача із системою. Зокрема, проаналізував послідовність дій при реєстрації, авторизації, шифруванні, створенні, редагуванні, перегляді та видаленні документів. Усі ці алгоритми були сформульовані у вигляді блок-схем, що наочно демонструють послідовність виконання кроків. Особливу увагу приділив реалізації безпеки на кожному з етапів — починаючи від перевірки існування користувача до генерації JWT-токенів та застосування AES-шифрування. Було доведено, що навіть базові операції в системі — такі як вхід у систему чи збереження документа — супроводжуються логуванням і криптографічним захистом, що забезпечує надійний рівень безпеки.

У підпункті 2.2 зосередив увагу на внутрішній структурі функцій системи. Спроєктував логіку роботи сервісів EncryptionService, JwtService та LogService. Для кожного з них описав функціональну відповідальність, передбачені можливі виняткові ситуації та запропоновано блок-схеми алгоритмів роботи. Також розробив діаграми послідовності, які демонструють реальні сценарії використання системи — наприклад, авторизацію користувача чи створення документа. Ці моделі дозволили розділити відповідальність між компонентами, забезпечити максимальну автономність кожного сервісу, а також показали, що логіка програми реалізована таким чином, щоб бути придатною до модульного тестування, розширення й масштабування. Було також враховано концепції гнучкості — наприклад, можливість легкої заміни алгоритму шифрування чи реалізації логування у вигляді окремого мікросервісу.

Найбільш комплексно була проаналізована архітектура всієї системи в підпункті 2.3. Підхід до побудови структури ґрунтувався на поєднанні кількох перевірених моделей — MVC, Service Layer і Onion Architecture. Розробив візуальні діаграми ,які ілюструють принципи побудови системи та взаємозв’язки між її складовими.

3 ПРОГРАМНА РЕАЛІЗАЦІЯ ЗАХИЩЕНОЇ СИСТЕМИ КЕРУВАННЯ ПАРОЛЯМИ НА ПІДПРИЄМСТВІ

3.1 Реалізація програмної захищеної системи керування паролями на підприємстві

У першу чергу я створив набір сервісів, відповідальних за окремі аспекти безпеки: шифрування (EncryptionService), генерацію JWT-токенів (JwtService) та логування (LogService).

На рисунку 3.1 представлено код сервісу EncryptionService, який реалізує симетричне шифрування з використанням алгоритму AES. Даний сервіс використовується в момент збереження чутливої інформації в базу даних, зокрема — вмісту документів.

```
2 references
public class EncryptionService : IEncryptionService
{
    4 references
    private readonly byte[] _key;
    3 references
    private readonly byte[] _iv;

    1 reference
    public EncryptionService(string key)
    {
        using var sha = SHA256.Create();
        _key = sha.ComputeHash(Encoding.UTF8.GetBytes(key));
        _iv = _key.Take(16).ToArray();
    }

    5 references
    public string Encrypt(string plainText)
    {
        using var aes = Aes.Create();
        aes.Key = _key;
        aes.IV = _iv;

        var encryptor = aes.CreateEncryptor(aes.Key, aes.IV);
        using var ms = new MemoryStream();
        using var cs = new CryptoStream(ms, encryptor, CryptoStreamMode.Write);
        using var sw = new StreamWriter(cs);
        sw.Write(plainText);
        sw.Close();

        return Convert.ToBase64String(ms.ToArray());
    }

    7 references
    public string Decrypt(string cipherText)
    {
        using var aes = Aes.Create();
        aes.Key = _key;
        aes.IV = _iv;

        var decryptor = aes.CreateDecryptor(aes.Key, aes.IV);
        var bytes = Convert.FromBase64String(cipherText);
        using var ms = new MemoryStream(bytes);
        using var cs = new CryptoStream(ms, decryptor, CryptoStreamMode.Read);
        using var sr = new StreamReader(cs);
        return sr.ReadToEnd();
    }
}
```

Рисунок 3.1 – Реалізація EncryptionService з використанням AES-алгоритму

Методи Encrypt та Decrypt забезпечують перетворення вхідного рядка у форматі plain text у зашифрований base64-рядок і навпаки. Усі ключі створюються на основі хешу від переданого ключового рядка за допомогою SHA-256, що дозволяє уникати слабких ключів. Другим критично важливим компонентом є JwtService. Його завдання — створювати безпечні токени, які містять інформацію про автентифікованого користувача. Токен передається разом із запитами та використовується для доступу до захищених ресурсів.Реалізація класу JwtService продемонстрована на рисунку 3.2.

```
2 references
public class JwtService : IJwtService
{
    4 references
    private readonly IConfiguration _config;

    0 references
    public JwtService(IConfiguration config)
    {
        _config = config;
    }

    2 references
    public string GenerateToken(string username)
    {
        var claims = new[]
        {
            new Claim(ClaimTypes.Name, username)
        };

        var key = new SymmetricSecurityKey(Encoding.UTF8.GetBytes(_config["Jwt:Key"]));
        var creds = new SigningCredentials(key, SecurityAlgorithms.HmacSha256);

        var token = new JwtSecurityToken(
            issuer: _config["Jwt:Issuer"],
            audience: _config["Jwt:Audience"],
            claims: claims,
            expires: DateTime.UtcNow.AddHours(1),
            signingCredentials: creds);

        return new JwtSecurityTokenHandler().WriteToken(token);
    }
}
```

Рисунок 3.2 – Клас JwtService

Цей токен формується підписом із секретним ключем і містить основні claims, зокрема — ім’я користувача. Час життя токена — одна година, що також визначається під час створення. Це дозволяє знизити ризики у разі компрометації. Ще один важливий елемент — LogService. Він реалізує простий асинхронний

метод LogAsync, що дозволяє зберігати у базу кожну дію користувача. Логи включають дію, ім'я користувача (якщо є), а також додаткову інформацію, якщо вона надана. Це дозволяє створити простий, але ефективний механізм контролю та аудиту.Реалізація класу LogService на рисунку 3.3.

```
public class LogService : ILogService
{
    3 references
    private readonly DataGroupContext _context;

    0 references
    public LogService(DataGroupContext context)
    {
        _context = context;
    }

    9 references
    public async Task LogAsync(string action, string? username = null, string? details = null)
    {
        var entry = new LogEntry
        {
            Action = action,
            Username = username,
            Details = details
        };

        _context.Logs.Add(entry);
        await _context.SaveChangesAsync();
    }
}
```

Рисунок 3.3 – Клас LogService

Всі вищезгадані сервіси інжектуються у контролери за допомогою механізму залежностей DI (Dependency Injection), що дозволяє зберігати чисту архітектуру та спрощує тестування. Основна взаємодія користувача із системою відбувається через контролер DocumentsController, який реалізує CRUD-функціонал для документів. Усі дані, які вводить користувач (назва, зміст документа), перед збереженням шифруються, а потім зберігаються у базі PostgreSQL.Реалізація методу Create на рисунку 3.4.

```
[HttpPost]
0 references
public async Task<IActionResult> Create(DocumentViewModel model)
{
    if (ModelState.IsValid)
    {
        var document = new Document
        {
            Name = _encryption.Encrypt(model.Name),
            Content = _encryption.Encrypt(model.Content)
        };

        if (model.File != null)
        {
            using var memoryStream = new MemoryStream();
            await model.File.CopyToAsync(memoryStream);
            document.FileData = memoryStream.ToArray();
            document.FileName = model.File.FileName;
            document.FileType = model.File.ContentType;
        }

        _context.Documents.Add(document);
        await _context.SaveChangesAsync();

        var username = User.Identity?.Name ?? "Anonimys";
        await _logService.LogAsync(
            "Document Created",
            username,
            $"Name: {document.Name}"
        );

        return RedirectToAction("Index");
    }

    return View(model);
}
```

Рисунок 3.4 – Метод Create в DocumentsController із викликом EncryptionService

Крім того, при кожному створенні, редагуванні, завантаженні чи видаленні документа відбувається логування дії. Таким чином, жодна операція в системі не залишається без цифрового сліду. Розшифрування відбувається вже під час завантаження сторінки з переліком документів або при перегляді детальної інформації.

Авторизація захищена атрибутами [Authorize], які застосовуються до контролерів. Доступ до певних методів можливий лише після автентифікації користувача. Для цього я реалізував AccountController із методами Login та

Register. Користувач може створити обліковий запис, пароль хешується через BCrypt, після чого дані потрапляють до таблиці Users. Під час логіну перевіряється хеш пароля, і в разі успіху генерується токен. Наочно реалізація Login та Register в AccountController видно на рисунку 3.5.

```
[HttpPost]
0 references
public async Task<IActionResult> Register(RegisterModel model)
{
    if (await _db.Users.AnyAsync(u => u.Username == model.Username)){
        await _log.LogAsync("RegisterFailed", model.Username, "User already exists");
        return BadRequest("User already exists");
    }


    var passwordHash = BCrypt.Net.BCrypt.HashPassword(model.Password);
    await _log.LogAsync("RegisterSuccess", model.Username);
    _db.Users.Add(new User { Username = model.Username, PasswordHash = passwordHash });
    await _db.SaveChangesAsync();

    return RedirectToAction("Index", "Documents");
}

[HttpPost]
0 references
public async Task<IActionResult> Login(LoginModel model)
{
    var user = await _db.Users.SingleOrDefaultAsync(u => u.Username == model.Username);
    if (user == null || !BCrypt.Net.BCrypt.Verify(model.Password, user.PasswordHash)){
        await _log.LogAsync("LoginFailed", model.Username);
        return Unauthorized();
    }

    await _log.LogAsync("LoginSuccess", model.Username);
    var token = _jwt.GenerateToken(user.Username);
    return RedirectToAction("Index", "Documents");
}
```

Рисунок 3.5 – Реалізація методу Login та Register в AccountController

Для взаємодії з користувачем я реалізував прості View-форми без використання JavaScript. Це забезпечує наочність і простоту. Сторінки для входу, реєстрації, перегляду документів, створення, редагування тощо реалізовані як стандартні Razor Views. Приклад однієї з таких видно на рисунку 3.6.

```
@model DataGroup.Models.LoginModel

<h2>Login</h2>

<form asp-action="Login" method="post">
    <div>
        <label>Username</label>
        <input asp-for="Username" />
    </div>
    <div>
        <label>Password</label>
        <input asp-for="Password" type="password" />
    </div>
    <button type="submit">Login</button>
</form>

@if (!ViewData.ModelState.IsValid)
{
    <div style="color: red">
        Invalid login or password.
    </div>
}

<div style="margin-top: 15px;">
    Не зареєстровані? <a asp-controller="Account" asp-action="Register">Зареєструватися</a>
</div>
```

Рисунок 3.6 – View Login.cshtml з полями для імені користувача та пароля

Як видно з прикладу, інтерфейс реалізовано максимально лаконічно й інтуїтивно зрозуміло. Всі поля підписані, кнопка має чітке призначення, а помилки відображаються безперешкодно в тому ж вікні. Це дозволяє забезпечити користувацький досвід без ускладнень.

3.2 Налаштування конфігурації системи на підприємстві

Основні параметри конфігурації визначаються у файлі appsettings.json, а також програмно задаються в методі конфігурування Program.cs. На рисунку 3.7 наведено вміст файлу appsettings.json, де зберігаються параметри підключення до бази даних, а також конфігураційні ключі JWT для автентифікації користувачів. Хотів би зазначити, що з міркувань безпеки секретні ключі рекомендується зберігати в захищеному середовищі, наприклад у середовищі змінних або в системі управління секретами.Але в даному випадку цього буде достатньо.

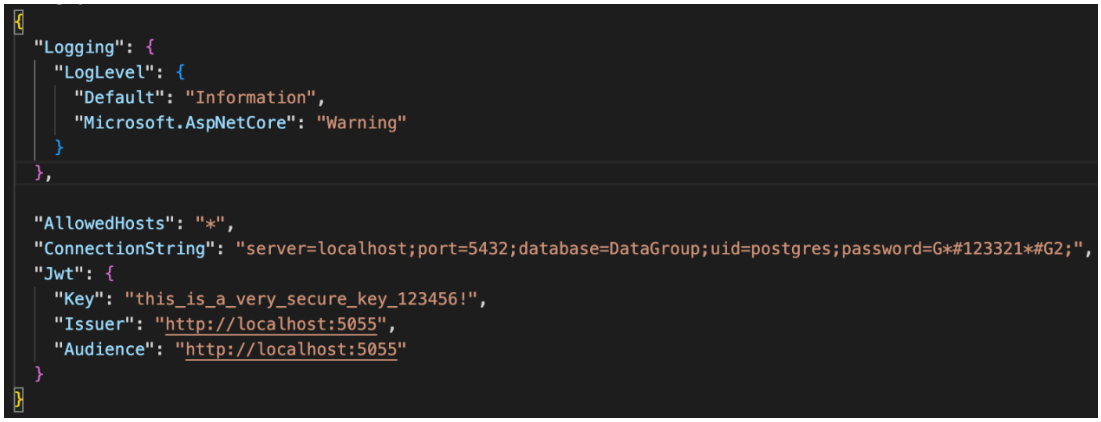


Рисунок 3.7 – Вміст конфігураційного файлу appsettings.json з параметрами JWT та рядком підключення PostgreSQL

У параметрі ConnectionString вказано шлях до бази даних PostgreSQL, ім'я бази, порт, логін та пароль користувача. Розділ Jwt містить конфігурацію для роботи з токенами: секретний ключ (Key), видавець (Issuer) та аудиторія (Audience). Значення цих параметрів мають відповідати тим, що перевіряються під час валідації токена при запитах.

Далі як можна побачити на рисунку 3.8, у файлі Program.cs здійснюється основне конфігурування всіх сервісів, які використовуються системою. Спершу зчитується конфігурація через метод GetConfiguration(). Потім налаштовується підключення до бази даних PostgreSQL за допомогою AddDbContextFactory, що дозволяє створювати контексти бази у потокобезпечний спосіб.

```
var configuration = GetConfiguration();

var builder = WebApplication.CreateBuilder(args);

builder.Services.AddDbContextFactory<DataGroupContext>(opts =>
    opts.UseNpgsql(configuration["ConnectionString"]));

builder.Services.AddControllersWithViews();
builder.Services.AddScoped<IJwtService, JwtService>();
builder.Services.AddSingleton<IEncryptionService>(new EncryptionService("myww_super_secure_passphrase"));
builder.Services.AddScoped<ILogService, LogService>();
```

Рисунок 3.8 – Конфігурування DI-контейнера у файлі Program.cs

Використовується AddScoped для IJwtService та ILogService, що означає створення нового екземпляра сервісу на кожен HTTP-запит. EncryptionService

додано як Singleton, адже він не містить стану і безпечний для повторного використання. Це дозволяє знизити навантаження на систему та забезпечує оптимальну продуктивність.

Далі я налаштував автентифікацію з використанням JWT яка зображена на рисунку 3.9. Я обрав схему JwtBearerDefaults.AuthenticationScheme, яка дозволяє приймати токени у заголовках HTTP-запитів. У блоці TokenValidationParameters вказано, що необхідно перевіряти видавця, аудиторію, підпис токена, термін його дії та ключ підпису. Також встановлено NameClaimType, що дозволяє зчитувати ім’я користувача з claim токена у подальшому коді.

```
builder.Services.AddAuthentication(JwtBearerDefaults.AuthenticationScheme)
    .AddJwtBearer(options =>
    {
        options.TokenValidationParameters = new TokenValidationParameters
        {
            ValidateIssuer = true,
            ValidateAudience = true,
            ValidateLifetime = true,
            ValidateIssuerSigningKey = true,
            ValidIssuer = configuration["Jwt:Issuer"],
            ValidAudience = configuration["Jwt:Audience"],
            IssuerSigningKey = new SymmetricSecurityKey(Encoding.UTF8.GetBytes(configuration["Jwt:Key"])),
            NameClaimType = ClaimTypes.Name
        };
    });
```

Рисунок 3.9 – Налаштування JWT-аутентифікації та параметрів валідації токенів

Після конфігурації автентифікації у програмі вмикається middleware-ланцюжок: UseStaticFiles, UseRouting, UseAuthentication та UseAuthorization. Це забезпечує правильну обробку запитів користувачів та обмеження доступу до ресурсів, що потребують авторизації.

На рисунку 3.10 наочно показана реалізація методу GetConfiguration() також у конфігурації маршрутизації вказано, що за замовчуванням користувач потрапляє на сторінку авторизації, тобто контролер Account, метод Login. Це дозволяє одразу з моменту запуску програми забезпечити перевірку доступу користувача.

```
var app = builder.Build();

if (!app.Environment.IsDevelopment())
{
    app.UseExceptionHandler("/Home/Error");
    app.UseHsts();
}

app.UseStaticFiles();
app.UseRouting();
app.UseAuthentication();
app.UseAuthorization();

app.MapControllerRoute(
    name: "default",
    pattern: "{controller=Account}/{action=Login}/{id?}");

app.Run();

IConfiguration GetConfiguration()
{
    var builder = new ConfigurationBuilder()
        .SetBasePath(Directory.GetCurrentDirectory())
        .AddJsonFile("appsettings.json", optional: false, reloadOnChange: true)
        .AddEnvironmentVariables();

    return builder.Build();
}
```

Рисунок 3.10 – Програмна точка входу та логіка запуску програми (Program.cs)

Крім конфігурації сервісів, я приділив окрему увагу налаштуванню бази даних. У системі використовується PostgreSQL як основна СУБД, що забезпечує зберігання користувачів, документів та логів. У appsettings.json вказано параметри підключення, включно з адресою сервера (localhost), портом (5432), ім’ям бази (DataGroup), логіном (postgres) і паролем. Ці дані використовуються в Program.cs при виклику методу AddDbContextFactory, що дозволяє створювати екземпляри DataGroupContext, які відповідають за взаємодію з БД.

Усі операції збереження, читання, редагування та видалення інформації відбуваються через ORM Entity Framework Core, що абстрагує розробника від прямого SQL. Таблиці бази даних та їхні поля зображено на рисунку 3.11.

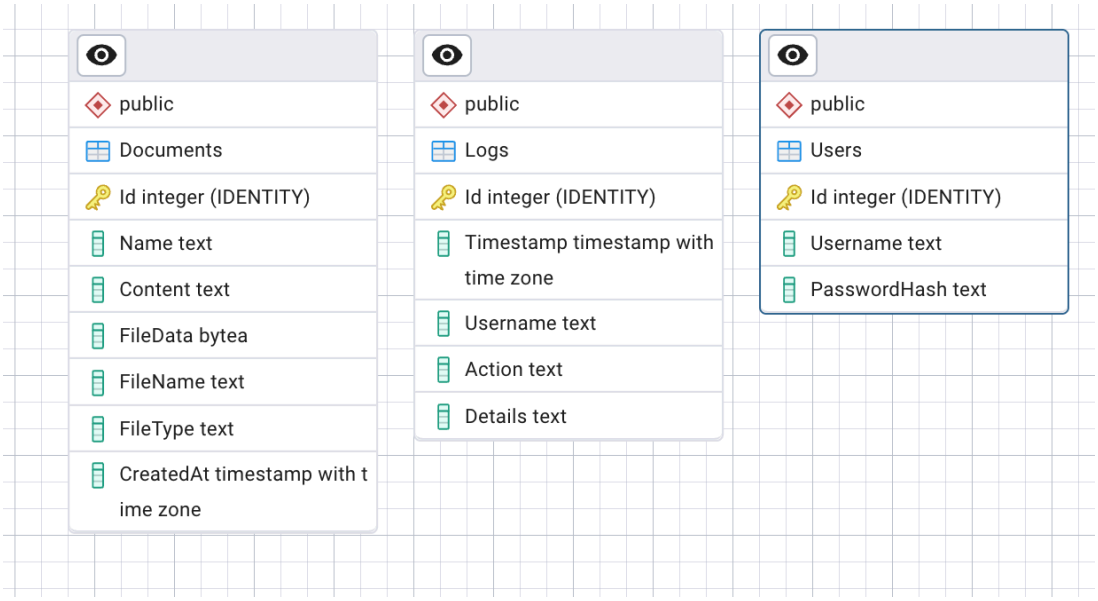


Рисунок 3.11 – Таблиці бази даних PostgreSQL: Users, Documents, Logs

Створення таблиць відбувається за допомогою механізму міграцій, а самі моделі системи (User, Document, LogEntry) чітко відповідають полям у базі. У таблиці Documents вміст назви та контент документа зберігаються у зашифрованому вигляді, таблиця Users містить хешовані паролі, а таблиця Logs зберігає всю історію дій користувачів, що дає змогу проводити аудит системи.

3.3 Розгортання та тестування захищеної системи керування паролями на підприємстві

Розгортання проводиться локально на сервері розробника, однак програмна реалізація повністю готова до переміщення на продакшн-сервер за допомогою контейнеризації або хостингу на будь-якому середовищі, сумісному з ASP.NET Core та PostgreSQL.

Після запуску програми першим етапом є перевірка коректності взаємодії клієнта із сервером. На рисунку 3.12 видно відкриття в браузері за адресою localhost:5055,воно автоматично спрямовує користувача на сторінку входу, що обробляється контролером AccountController. На цій сторінці користувач може

виконати вхід або перейти до реєстрації. У випадку неправильного введення пароля або неіснуючого користувача виводиться повідомлення про помилку, що підтверджує роботу валідації.

Рисунок 3.12 – Сторінка входу (Login) при першому запуску системи

Після успішної авторизації генерується JWT-токен, який автоматично передається із кожним наступним запитом. Цей токен дозволяє доступ до захищених частин програми, таких як перегляд, створення та редагування документів. Для підтвердження автентифікації токена використовуються атрибути [Authorize], які присутні в усіх захищених контролерах. Спроба доступу до документів без токена автоматично перекидає користувача на сторінку входу. На рисунку 3.13 видно результат спроби доступу до контролера Documents без токена.

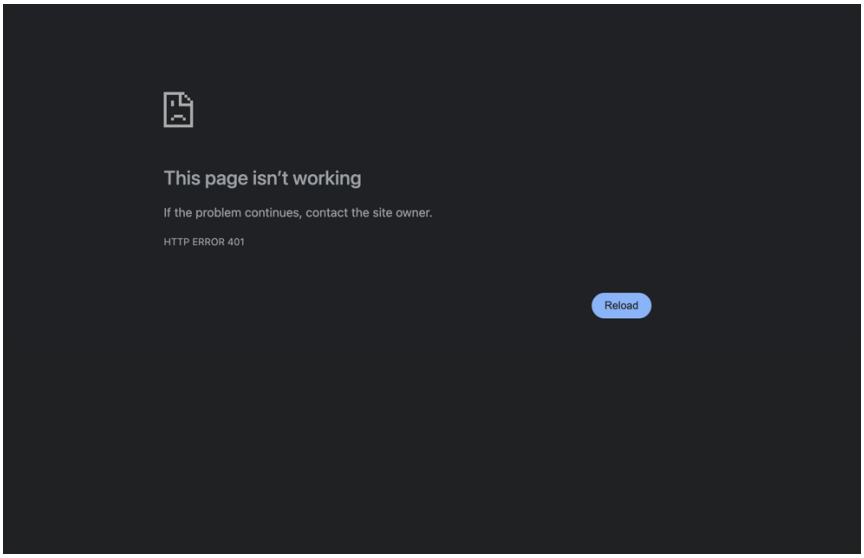


Рисунок 3.13 – Перевірка доступу до сторінки /Documents без токена

Далі виконав перевірку функціональності шифрування та дешифрування даних. Створив тестовий документ із назвою та вмістом, після чого проведено перегляд вмісту бази даних у PostgreSQL, де підтверджено, що поля Name та Content зберігаються у зашифрованому вигляді, наочно видно на рисунку 3.14. Це означає, що навіть у разі витоку бази зловмисник не зможе прочитати чутливу інформацію без ключа дешифрування.

	Id [PK] integer	Name text	Content text
1	14	ql+lawtrvxwSaOwqxs1I4WMgkGzrGPAshzshLY0y82g=	4Ra/UK+2xS0rlWFl/bYZqWABngTVCQGs3GUOxPESXbI=

Рисунок 3.14 – Зафіксований запис документа в зашифрованому вигляді у базі PostgreSQL

Окремим етапом я провів модульне тестування основних сервісів, що відповідають за безпеку та цілісність системи. Створив юніт-тести для наступних компонентів:

- encryptionService – перевірка коректності шифрування та дешифрування;
- jwtService – перевірка формату та вмісту створеного JWT-токена;
- logService – перевірка запису логів у контекст бази;

- accountController – перевірка поведінки при реєстрації та логіні;
- documentsController – тестування методу створення документів.

Для юніт-тестів використовував фреймворк xUnit та бібліотека Moq для створення моків залежностей. Також використовув вбудовану базу InMemoryDatabase, що дозволило виконувати тести без підключення до реальної PostgreSQL.

Першим протестував сервіс EncryptionService, який відповідає за шифрування та дешифрування даних. Створив тест, який виконує шифрування певного рядка, а потім розшифровує його й перевіряє, чи дорівнює результат початковому значенню. Це підтверджує, що механізм шифрування працює коректно і не втрачає даних. Реалізацію цього тесту видно на рисунку 3.15.

```
public class EncryptionServiceTests
{
    3 references
    private readonly EncryptionService _service;

    0 references
    public EncryptionServiceTests()
    {
        _service = new EncryptionService("test_passphrase_1234567890123456");
    }

    [Fact]
    0 references
    public void EncryptDecrypt_ShouldReturnOriginalText()
    {
        var original = "Sensitive data";
        var encrypted = _service.Encrypt(original);
        var decrypted = _service.Decrypt(encrypted);
        decrypted.Should().Be(original);
    }
}
```

Рисунок 3.15 – Тест для EncryptionService: перевірка коректності дешифрування

Далі тестуванню піддавався DocumentsController, що видно на рисунку 3.16, зокрема функції створення документа. Змоделював передачу форми документа (назва, зміст, файл), і перевірив, чи об’єкт документа з’являється в базі даних. Усі

залежності (шифрування, логування) змінив на моки, що дозволило протестувати лише бізнес-логіку контролера.

```
public DocumentsControllerTests()
{

    var options = new DbContextOptionsBuilder<DataGroupContext>()
        .UseInMemoryDatabase(databaseName: "TestDb")
        .Options;

    _context = new DataGroupContext(options);
    _logServiceMock = new Mock<ILogService>();
    _encryptionMock = new Mock<IEncryptionService>();

    _encryptionMock.Setup(x => x.Encrypt(It.IsAny<string>())).Returns((string s) => s);
    _encryptionMock.Setup(x => x.Decrypt(It.IsAny<string>())).Returns((string s) => s);

    _controller = new DocumentsController(_context, _encryptionMock.Object, _logServiceMock.Object);


    var user = new ClaimsPrincipal(new ClaimsIdentity(new Claim[]
    {
        new Claim(ClaimTypes.Name, "testuser")
    }, "mock"));

    _controller.ControllerContext = new ControllerContext
    {
        HttpContext = new DefaultHttpContext { User = user }
    };
}

[Fact]
0 references
public async Task Create_ValidModel_ShouldSaveDocumentAndLog()
{
    var model = new DocumentViewModel
    {
        Name = "Test Name",
        Content = "Test Content"
    };

    var result = await _controller.Create(model);

    var doc = await _context.Documents.FirstOrDefaultAsync();
    Assert.NotNull(doc);
    Assert.Equal("Test Name", doc.Name);
    Assert.IsType<RedirectToActionResult>(result);

    _logServiceMock.Verify(x => x.LogAsync("Document Created", "testuser", It.IsAny<string>()), Times.Once);
}
```

Рисунок 3.16 – Тест на створення документа з моканим контекстом бази

Останнім етапом став JwtService, який відповідає за генерацію JWT-токенів. У тесті перевірив, чи не є порожнім створений токен, а також розбирається його структура, щоб упевнитися, що в claims міститься коректне ім’я користувача. Це дозволяє гарантувати, що токен дійсний та містить інформацію, необхідну для авторизації користувачів у системі. Приклад реалізації тесту на рисунку 3.17.

```
public class JwtServiceTests
{
    [Fact]
    0 references
    public void GenerateToken_ShouldReturnValidJwt()
    {
        var mockConfig = new Mock<IConfiguration>();
        mockConfig.Setup(c => c["Jwt:Key"]).Returns("this_is_a_very_secure_key_123456!");
        mockConfig.Setup(c => c["Jwt:Issuer"]).Returns("TestIssuer");
        mockConfig.Setup(c => c["Jwt:Audience"]).Returns("TestAudience");

        var jwtService = new JwtService(mockConfig.Object);
        var token = jwtService.GenerateToken("testuser");

        token.Should().NotBeNullOrEmpty();
        token.Should().Contain(".");
    }
}
```

Рисунок 3.17 – Тест для JwtService: перевірка валідності створеного токена

Усі юніт-тести які були створені для перевірки внутрішнього функціоналу були пройдені успішно, що підтверджується на рисунку 3.18.

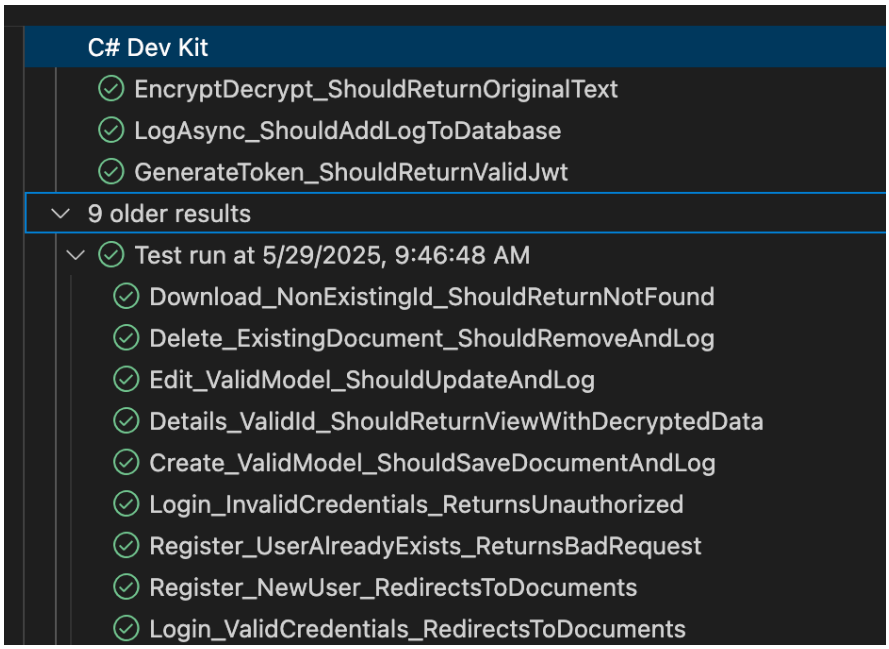


Рисунок 3.18 – Результат проходження тестів системи

Таким чином, реалізоване рішення не лише відповідає вимогам безпеки, але й є стабільним та тестованим у різних сценаріях використання. Повна функціональність системи доведена шляхом автоматизованого тестування.

3.5 Висновки

У даному розділі було безпосередньо реалізовано всі найважливіші функціональні елементи програмної захищеної системи керування паролями для підприємства. Показано, як теоретичні підходи, розглянуті у попередніх частинах роботи, були втілені в практичному середовищі з використанням сучасного стеку технологій — ASP.NET Core MVC, C#, PostgreSQL та додаткових сервісів для забезпечення шифрування, авторизації та журналювання.

У рамках реалізації було створено окремі сервіси, відповідальні за безпеку (EncryptionService), аутентифікацію (JwtService) і логування (LogService), які були інтегровані в контролери системи. Зокрема, контролер DocumentsController реалізує захищену роботу з документами, уся інформація яких зберігається в базі у зашифрованому вигляді. Контролер AccountController забезпечує функції реєстрації та входу користувачів із застосуванням хешування паролів і видачею токенів доступу.

Конфігураційні параметри зосереджено в appsettings.json, де визначено безпечні ключі, рядок підключення до бази, а також параметри валідації JWT. Усі залежності налаштовано через DI-контейнер, що дозволило побудувати гнучку і розширювану архітектуру з мінімальним рівнем зв’язаності між компонентами.

Також було здійснено повноцінне тестування реалізованого функціоналу. Створено юніт-тести для окремих сервісів і контролерів із використанням xUnit, Moq та InMemoryDatabase. Результати тестів підтвердили коректність роботи механізмів авторизації, шифрування, логування та обробки запитів. Завдяки цьому вдалося створити надійну та безпечну систему, яка відповідає сучасним вимогам до зберігання та обробки конфіденційної інформації. Впроваджені рішення демонструють ефективну інтеграцію теоретичних знань у реальний програмний продукт.

ВИСНОВКИ

У процесі виконання дипломної роботи було реалізовано повний цикл розробки захищеної системи керування паролями для підприємства «DataGroup». На основі аналізу загроз, вивчення сучасних менеджерів паролів і вимог до безпеки було сформовано функціональні та архітектурні вимоги до системи, які лягли в основу практичної реалізації. З урахуванням того, що навіть найкращі рішення можуть бути вразливими без належного контролю та аудиту, було створено власну систему, яка не лише зберігає й генерує паролі, а й інтегрується у внутрішні процеси компанії, забезпечуючи контроль доступу, аудит дій користувачів і моніторинг загроз.

Система побудована на фреймворку ASP.NET Core MVC з використанням архітектури Onion, яка дозволяє ізолювати бізнес-логіку, полегшити тестування й забезпечити масштабованість. Основні компоненти системи реалізовано у вигляді окремих сервісів: шифрування даних (AES), генерація токенів доступу (JWT), хешування паролів (BCrypt), логування подій. Усі сервіси підключені через DI-контейнер, що спрощує їхню підтримку й розвиток.

Розроблено повноцінну систему авторизації та автентифікації на основі JWT-токенів. Контролери системи захищені за допомогою атрибутів [Authorize], що дозволяє гнучко обмежувати доступ до критичних ресурсів. Чутлива інформація, включно з паролями й документами, зберігається у зашифрованому вигляді в базі даних PostgreSQL. Для перевірки надійності реалізованого функціоналу було проведено модульне тестування з використанням xUnit, Moq та InMemoryDatabase. Результати тестів підтвердили правильність роботи основних механізмів системи: шифрування, автентифікації, логування та контролю доступу.

Загалом система є надійною, гнучкою й придатною для використання на підприємствах малого та середнього бізнесу. Усі цілі та завдання, поставлені на початку роботи, були досягнуті повністю.

ПЕРЕЛІК ДЖЕРЕЛ

1. Українські студії в європейському контексті. Зб. наук. пр., 2023. URL: https://www.academia.edu/109645587 (дата звернення: 21.02.2025).

2. Кращі менеджери паролів для ПК і смартфонів. URL: https://www.moyo.ua/ua/news/luchshiy_menedzher_paroley_top_10_programm_dlya_pk_i_smartfonov.html (дата звернення: 21.02.2025).

3. Password manager від NordVPN . – URL: https://nordvpn.com/uk/password-manager/ (дата звернення: 21.02.2025).

4. Менеджер паролів: що це таке і для яких цілей він потрібен. URL: https://hackyourmom.com/pryvatnist/menedzher-paroliv-shho-cze-take-i-dlya-yakyh-czilej-vin-potriben/ (дата звернення: 21.02.2025).

5. 10 найкращих менеджерів паролів у 2024 році . URL: https://blog.acer.com/ua/discussion/1790/10-naykraschih-menedzheriv-paroliv-u-2024-roci (дата звернення: 21.02.2025).

6. Топ-7 кращих менеджерів паролів . URL: https://bizmag.com.ua/top-7-krashchykh-menedzheriv-paroliv/ (дата звернення: 21.02.2025).

7. Огляд Keeper – менеджера паролів. URL: https://digital-expert.online/ua/best-password-managers-tools/keeper-password-managers-reviews (дата звернення: 21.02.2025).

8. RoboForm – менеджер паролів. URL: https://www.roboform.com/ (дата звернення: 21.02.2025).

9. RoboForm for Business. URL: https://www.linkedin.com/company/roboformforbusiness (дата звернення: 21.02.2025).

10. Який менеджер паролів обрати у. URL: https://speka.media/yakii-menedzer-paroliv-obrati-u-2025-porivnyannya-nordpass-1password-keeper-ta-proton-pass-9dn205 (дата звернення: 21.02.2025).

11. Best password manager tools. URL: https://www.unite.ai/uk/best-password-manager-tools (дата звернення: 23.03.2025).

12. 1Password – огляд менеджера. URL: https://softlist.com.ua/ua/news/1rassword-obzor-menedzhera-paroley (дата звернення: 23.03.2025).

13. Best password managers . URL: https://www.guru99.com/uk/best-password-managers.html (дата звернення: 23.03.2025).

14. Чи безпечно користуватись менеджером паролів LastPass. URL: https://nadiyno.org/chy-bezpechno-korystuvatys-menedzherom-paroliv-lastpass/ (дата звернення: 23.03.2025).

15. What makes a strong. URL: https://www.keepersecurity.com/blog/2023/08/31/what-makes-a-strong-password/ (дата звернення: 23.03.2025).

16. Strong password. URL: https://www.techtarget.com/searchenterprisedesktop/definition/strong-password (дата звернення: 23.03.2025).

17. Secure password generator. URL: https://uk.vpnmentor.com/tools/secure-password-generator (дата звернення: 23.03.2025).

18. Time it would take a computer to crack a password . URL: https://www.statista.com/chart/26298/time-it-would-take-a-computer-to-crack-a-password/ (дата звернення: 23.03.2025).

19. Як відбуваються витоки даних: пояснення від Molfar. URL: https://molfar.com/blog/yak-vidbuvayutsya-vytoky-danyh-poyasnennya-vid-molfar (дата звернення: 23.03.2025).

20. Інформаційна безпека: види загроз і методи їх усунення. URL: https://datami.ee/ua/blog/informationsijna-bezpeka-vidi-zagroz-i-metodi-yih-usunennya (дата звернення: 23.03.2025).

21. What is the Principle of Least Privilege. URL: https://www.paloaltonetworks.com/cyberpedia/what-is-the-principle-of-least-privilege (дата звернення: 14.04.2025).

22. Енциклопедія загроз. URL: https://www.eset.com/ua/support/information/entsyklopediya-zahroz/dlp/ (дата звернення: 14.04.2025).

23. Best practices for minimizing the risk of PII breaches. URL: https://corewin.ua/blog/best-practices-for-minimizing-the-risk-of-pii-breaches (дата звернення: 14.04.2025).

24. What is ISO 27004 and ISMS monitoring. URL: https://continuumgrc.com/uk/what-is-iso-27004-and-isms-monitoring/ (дата звернення: 18.04.2025).

25. Інформаційна безпека підприємства та основні засади захисту даних. URL: https://gigatrans.ua/ua/news/nformac-yna-bezpeka-p-dpri-mstva-ta-osnovn-zasadi-zahistu-danih (дата звернення: 18.04.2025).

26. Random password generator. URL: https://www.avast.ua/random-password-generator#mac (дата звернення: 18.04.2025).

27. Hashing. URL: https://www.vpnunlimited.com/ua/help/cybersecurity/hashing (дата звернення: 18.04.2025).

28. Insecure password storage vulnerabilities . URL: https://cqr.company/ua/web-vulnerabilities/insecure-passwords-storage/ (дата звернення: 19.04.2025).

29. What is key stretching? Password security explained. URL: https://4imag.com/what-is-key-stretching-password-security (дата звернення: 23.04.2025).

30. SHA-256 Hashing and Password Cracking. URL: https://specopssoft.com/blog/sha256-hashing-password-cracking/ (дата звернення: 23.04.2025).

31. Logging in: concepts, requirements. URL: https://training.qatestlab.com/blog/technical-articles/logging-in-concepts-requirements-levels/ (дата звернення: 23.04.2025).

32. Для чого потрібне логування: переваги та значення. URL: https://amur.kozak.cx.ua/psikhologiya/dlya-chogo-potribne-loguvannya-perevagi-ta-znachennya.html (дата звернення: 23.04.2025).

33. Як використовувати JSON Web Tokens (JWT) для автентифікації. URL: https://devzone.org.ua/post/iak-vykorystovuvaty-json-web-tokens-jwt-dlia-avtentyfikatsiyi (дата звернення: 23.04.2025).

34. Introduction to JWT. URL: https://jwt.io/introduction (дата звернення: 23.04.2025).

35. C#: що це за мова і де його використовують. URL: https://robotdreams.cc/uk/blog/284-s-chto-eto-za-yazyk-i-gde-ego-ispolzuyut (дата звернення: 23.04.2025).

36. Вступ до мови C#. URL: https://www.w3schools.com/cs/cs_intro.aspx (дата звернення: 23.04.2025).

37. Що таке ASP.NET: основи для початківців. URL: https://foxminded.ua/asp-net-tse/ (дата звернення: 23.04.2025).

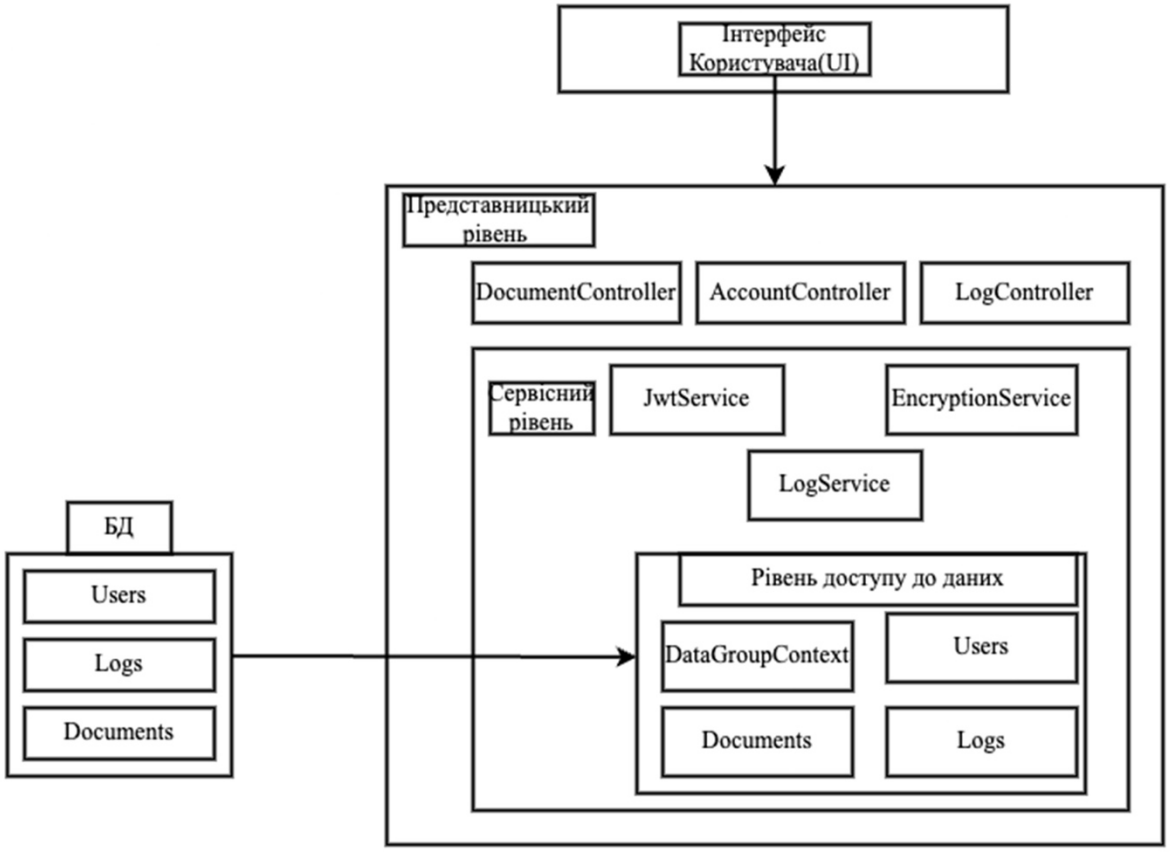
38. ASP.NET Core. URL: https://github.com/dotnet/aspnetcore (дата звернення: 23.04.2025).

39. PostgreSQL: вступ до реляційних баз даних. URL: https://itvdn.com/ua/video/postgresql-ua (дата звернення: 23.04.2025).

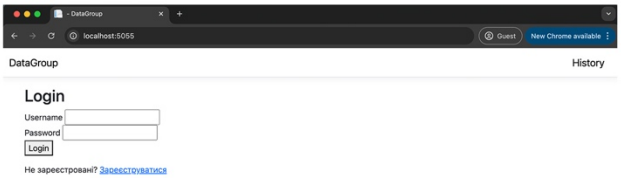
40. PostgreSQL – встановлення і налаштування. URL: https://thehost.ua/ua/wiki/administration/database/postgresql-install (дата звернення: 23.04.2025.

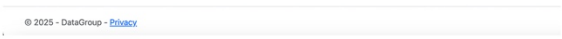
ДОДАТОК А

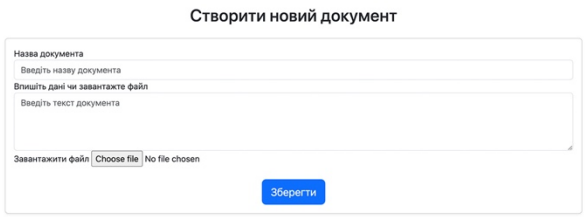
Копії графічної частини

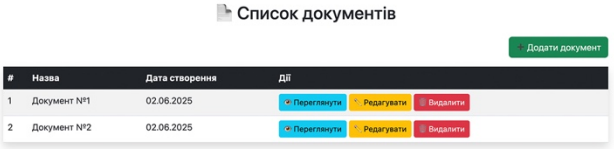


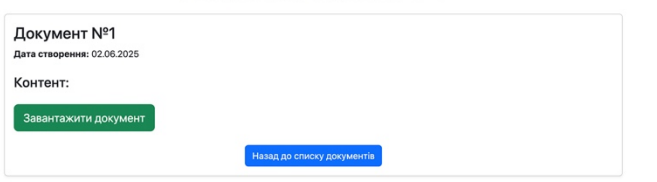
					КРБКБ.2102141.21.02.20 Е8		
Зм.	Арк.	№ докум.	Підпис	Дата	Система забезпечення безпеки конфіденційних даних від витоку на основі керування паролями Схема загальної архітектури програми	Літера	Маса	Масштаб
Розроб.		Вовкович М.О.				У		
Перевір.		Джулій В.М.						
Н.контр.						Аркуш 1	Аркушів 3	
Т. контр.		Мостовий С.В.				ХНУ, КБ-21-2		
Затв.		Кльоц Ю.П.						



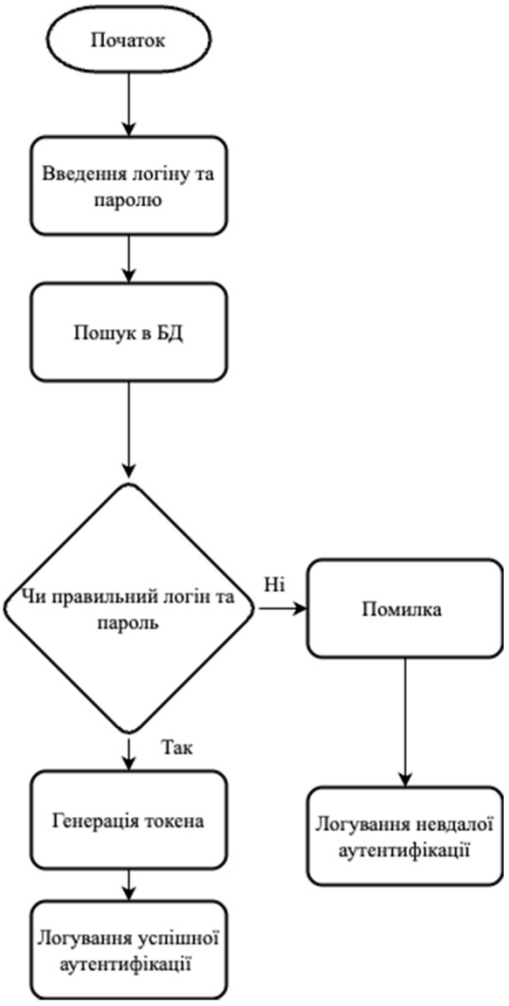


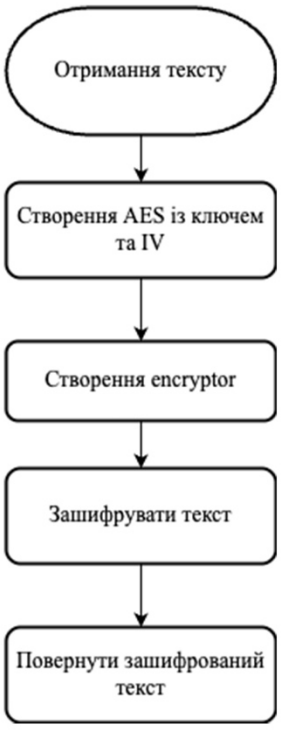


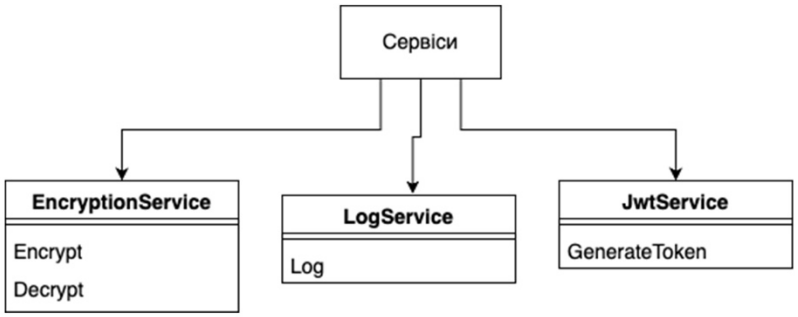




					КРБКБ.2102141.21.02.20 Е8		
Зм.	Арк.	№ докум.	Підпис	Дата	Система забезпечення безпеки конфіденційних даних від витоку на основі керування паролями Графічний інтерфейс	Літера	Маса	Масштаб
Розроб.		Вовкович М.О.				У		
Перевір.		Джулій В.М.						
Н.контр.						Аркуш 2	Аркушів 3	
Т. контр.		Мостовий С.В.				ХНУ, КБ-21-2		
Затв.		Кльоц Ю.П.						







					КРБКБ.2102141.21.02.20 Е8	
Зм.	Арк.	№ докум.	Підпис	Дата		
Розроб.		Вовкович М.О.			Система забезпечення безпеки конфіденційних даних від витоку на основі керування паролями Блок-схеми роботи сервісів	Літера: У; Маса; Масштаб
Перевір.		Джулій В.М.				
Н.контр.						Аркуш 3; Аркушів 3
Т. контр.		Мостовий С.В.				ХНУ, КБ-21-2
Затв.		Кльоц Ю.П.				

ДОДАТОК Б

Код програмного забезпечення

```
using Microsoft.AspNetCore.Authentication.JwtBearer;
using Microsoft.IdentityModel.Tokens;
using System.Text;
using DataGroup.Data;
using Microsoft.EntityFrameworkCore;
using DataGroup.Services;
using DataGroup.Interfaces;
using System.Security.Claims;

var configuration = GetConfiguration();

var builder = WebApplication.CreateBuilder(args);

builder.Services.AddDbContextFactory<DataGroupContext>(opts =>
   opts.UseNpgsql(configuration["ConnectionString"]));

builder.Services.AddControllersWithViews();
builder.Services.AddScoped<IJwtService, JwtService>();
builder.Services.AddSingleton<IEncryptionService>(new
EncryptionService("myww_super_secure_passphrase"));
builder.Services.AddScoped<ILogService, LogService>();

// JWT
builder.Services.AddAuthentication(JwtBearerDefaults.AuthenticationScheme)
   .AddJwtBearer(options =>
   {
      options.TokenValidationParameters = new TokenValidationParameters
      {
         ValidateIssuer = true,
         ValidateAudience = true,
         ValidateLifetime = true,
         ValidateIssuerSigningKey = true,
         ValidIssuer = configuration["Jwt:Issuer"],
         ValidAudience = configuration["Jwt:Audience"],
         IssuerSigningKey = new
SymmetricSecurityKey(Encoding.UTF8.GetBytes(configuration["Jwt:Key"])),
         NameClaimType = ClaimTypes.Name
      };
   });
```

```
var app = builder.Build();

if (!app.Environment.IsDevelopment())
{
   app.UseExceptionHandler("/Home/Error");
   app.UseHsts();
}

app.UseStaticFiles();
app.UseRouting();
app.UseAuthentication();
app.UseAuthorization();

app.MapControllerRoute(
   name: "default",
   pattern: "{controller=Account}/{action=Login}/{id?}");

app.Run();

IConfiguration GetConfiguration()
{
   var builder = new ConfigurationBuilder()
      .SetBasePath(Directory.GetCurrentDirectory())
      .AddJsonFile("appsettings.json", optional: false, reloadOnChange: true)
      .AddEnvironmentVariables();

   return builder.Build();
}
using Microsoft.AspNetCore.Mvc;
using DataGroup.Data;
using DataGroup.Models;
using DataGroup.Services;
using Microsoft.EntityFrameworkCore;
using DataGroup.Interfaces;

public class AccountController : Controller
{
   private readonly DataGroupContext _db;
   private readonly ILogService _log;
   private readonly IJwtService _jwt;

   public AccountController(DataGroupContext db, IJwtService jwt, ILogService log)
   {
```

```
        _db = db;
        _jwt = jwt;
        _log = log;
    }

    [HttpPost]
    public async Task<IActionResult> Register(RegisterModel model)
    {
        if (await _db.Users.AnyAsync(u => u.Username == model.Username)){
            await _log.LogAsync("RegisterFailed", model.Username, "User already
exists");
            return BadRequest("User already exists");
        }


        var passwordHash = BCrypt.Net.BCrypt.HashPassword(model.Password);
        await _log.LogAsync("RegisterSuccess", model.Username);
        _db.Users.Add(new User { Username = model.Username, PasswordHash =
passwordHash });
        await _db.SaveChangesAsync();

        return RedirectToAction("Index", "Documents");
    }

    [HttpPost]
    public async Task<IActionResult> Login(LoginModel model)
    {
        var user = await _db.Users.SingleOrDefaultAsync(u => u.Username ==
model.Username);
        if (user == null || !BCrypt.Net.BCrypt.Verify(model.Password,
user.PasswordHash)){
            await _log.LogAsync("LoginFailed", model.Username);
            return Unauthorized();
        }

        await _log.LogAsync("LoginSuccess", model.Username);
        var token = _jwt.GenerateToken(user.Username);
        return RedirectToAction("Index", "Documents");
    }

    [HttpGet]
    public IActionResult Login()
    {
        return View();
```

```
    }

    [HttpGet]
    public IActionResult Register()
    {
        return View();
    }
}

using Microsoft.AspNetCore.Mvc;
using Microsoft.EntityFrameworkCore;
using DataGroup.Data;
using DataGroup.Models;
using DataGroup.Services;
using DataGroup.Interfaces;
using Microsoft.AspNetCore.Authorization;

namespace DataGroup.Controllers;

public class DocumentsController : Controller
{
    private readonly DataGroupContext _context;
    private readonly IEncryptionService _encryption;
    private readonly ILogService _logService;

    public DocumentsController(DataGroupContext context, IEncryptionService
encryption, ILogService logService)
    {
        _context = context;
        _logService = logService;
        _encryption = encryption;
    }

    public async Task<IActionResult> Index()
    {
        var documents = await _context.Documents.ToListAsync();
        foreach (var doc in documents)
        {
            doc.Name = _encryption.Decrypt(doc.Name);
            doc.Content = _encryption.Decrypt(doc.Content);
        }

        return View(documents);
    }
```

```
public IActionResult Create() => View();

[HttpPost]
public async Task<IActionResult> Create(DocumentViewModel model)
{
    if (ModelState.IsValid)
    {
        var document = new Document
        {
            Name = _encryption.Encrypt(model.Name),
            Content = _encryption.Encrypt(model.Content)
        };

        if (model.File != null)
        {
            using var memoryStream = new MemoryStream();
            await model.File.CopyToAsync(memoryStream);
            document.FileData = memoryStream.ToArray();
            document.FileName = model.File.FileName;
            document.FileType = model.File.ContentType;
        }

        _context.Documents.Add(document);
        await _context.SaveChangesAsync();

        var username = User.Identity?.Name ?? "Anonimys";
        await _logService.LogAsync(
            "Document Created",
            username,
            $"Name: {document.Name}"
        );

        return RedirectToAction("Index");
    }

    return View(model);
}

public async Task<IActionResult> Download(int id)
{
    var document = await _context.Documents.FindAsync(id);
    if (document == null || document.FileData == null)
        return NotFound();
```

```
    var username = User.Identity?.Name ?? "Anonimys";
    await _logService.LogAsync(
            "Document Downloaded",
            username,
            $"Name: {document.Name}");

    return File(document.FileData, document.FileType, document.FileName);
}

public async Task<IActionResult> Edit(int id)
{
    var document = await _context.Documents.FindAsync(id);
    if (document == null)
        return NotFound();

    var model = new DocumentViewModel
    {
        Id = document.Id,
        Name = _encryption.Decrypt(document.Name),
        Content = _encryption.Decrypt(document.Content)
    };


    return View(model);
}

[HttpPost]
public async Task<IActionResult> Edit(DocumentViewModel model)
{
    if (ModelState.IsValid)
    {
        var document = await _context.Documents.FindAsync(model.Id);
        if (document == null)
            return NotFound();

        document.Name = _encryption.Encrypt(model.Name);
        document.Content = _encryption.Encrypt(model.Content);

        if (model.File != null)
        {
            using var memoryStream = new MemoryStream();
```

```
            await model.File.CopyToAsync(memoryStream);
            document.FileData = memoryStream.ToArray();
            document.FileName = model.File.FileName;
            document.FileType = model.File.ContentType;
        }

        _context.Update(document);
        await _context.SaveChangesAsync();
        var username = User.Identity?.Name ?? "Anonimys";
        await _logService.LogAsync(
            "Document Edited",
            username,
            $"Name: {document.Name}"
        );

        return RedirectToAction("Index");
    }

    return View(model);
}

public async Task<IActionResult> Delete(int id)
{
    var document = await _context.Documents.FindAsync(id);
    if (document == null)
        return NotFound();

    _context.Documents.Remove(document);
    await _context.SaveChangesAsync();
    var username = User.Identity?.Name ?? "Anonimys";
    await _logService.LogAsync(
            "Document Deleted",
            username,
            $"Name: {document.Name}"
    );

    return RedirectToAction("Index");
}

public async Task<IActionResult> Details(int? id)
{
    if (id == null)
        return NotFound();
```

```
        var document = await _context.Documents.FirstOrDefaultAsync(m => m.Id ==
id);
        if (document == null)
            return NotFound();

        document.Name = _encryption.Decrypt(document.Name);
        document.Content = _encryption.Decrypt(document.Content);

        return View(document);
    }
}

using Microsoft.AspNetCore.Mvc;
using DataGroup.Data;
using DataGroup.Models;
using DataGroup.Services;
using Microsoft.EntityFrameworkCore;

namespace DataGroup.Controllers
{

    public class LogsController : Controller
    {
        private readonly DataGroupContext _context;

        public LogsController(DataGroupContext context)
        {
            _context = context;
        }

        public async Task<IActionResult> Index()
        {
            var logs = await _context.Logs.OrderByDescending(l =>
l.Timestamp).ToListAsync();
            return View(logs);
        }
    }
}

using Microsoft.EntityFrameworkCore;
using DataGroup.Models;

namespace DataGroup.Data
{
```

```
    public class DataGroupContext : DbContext
    {
        public DbSet<Document> Documents { get; set; }
        public DbSet<User> Users { get; set; }
        public DbSet<LogEntry> Logs { get; set; }

        public DataGroupContext(DbContextOptions<DataGroupContext> options) :
base(options) { }

        protected override void OnConfiguring(DbContextOptionsBuilder
optionsBuilder)
        {

        }
    }
}


namespace DataGroup.Interfaces{

    public interface IEncryptionService

    {
        public string Encrypt(string plainText);
        public string Decrypt(string plainText);
    }
}



namespace DataGroup.Interfaces{

    public interface IJwtService
    {
        public string GenerateToken(string username);
    }
}



namespace DataGroup.Interfaces{

    public interface ILogService
    {
```

```
        Task LogAsync(string action, string? username = null, string? details = null);
    }
}


using System;
using System.ComponentModel.DataAnnotations;
using System.ComponentModel.DataAnnotations.Schema;

namespace DataGroup.Models
{
    public class Document
{
    [Key]
    public int Id { get; set; }
    public string Name { get; set; } = null!;
    public string? Content { get; set; }
    public byte[]? FileData { get; set; }
    public string? FileName { get; set; }
    public string? FileType { get; set; } = null!;

    [Column(TypeName = "timestamptz")]
    public DateTime CreatedAt { get; set; } = DateTime.UtcNow;
}
}

public class DocumentViewModel
{
    public int Id { get; set; }
    public string Name { get; set; } = null!;
    public string? Content { get; set; }
    public IFormFile? File { get; set; }
}

namespace DataGroup.Models
{
    public class LogEntry
    {
        public int Id { get; set; }
        public DateTime Timestamp { get; set; } = DateTime.UtcNow;
        public string? Username { get; set; }
        public string Action { get; set; } = null!;
        public string? Details { get; set; }
    }
```

```
}

namespace DataGroup.Models
{
    public class LoginModel
    {
        public string Username { get; set; }
        public string Password { get; set; }
    }
}

namespace DataGroup.Models
{
    public class RegisterModel
    {
        public string Username { get; set; }
        public string Password { get; set; }
    }
}

namespace DataGroup.Models
{
    public class User
    {
        public int Id { get; set; }
        public string Username { get; set; }
        public string PasswordHash { get; set; }
    }
}

using System.Security.Cryptography;
using System.Text;
using DataGroup.Interfaces;

namespace DataGroup.Services
{
    public class EncryptionService : IEncryptionService
    {
        private readonly byte[] _key;
        private readonly byte[] _iv;

        public EncryptionService(string key)
        {
            using var sha = SHA256.Create();
```

```
            _key = sha.ComputeHash(Encoding.UTF8.GetBytes(key));
            _iv = _key.Take(16).ToArray();
        }

        public string Encrypt(string plainText)
        {
            using var aes = Aes.Create();
            aes.Key = _key;
            aes.IV = _iv;

            var encryptor = aes.CreateEncryptor(aes.Key, aes.IV);
            using var ms = new MemoryStream();
            using var cs = new CryptoStream(ms, encryptor, CryptoStreamMode.Write);
            using var sw = new StreamWriter(cs);
            sw.Write(plainText);
            sw.Close();

            return Convert.ToBase64String(ms.ToArray());
        }

        public string Decrypt(string cipherText)
        {
            using var aes = Aes.Create();
            aes.Key = _key;
            aes.IV = _iv;

            var decryptor = aes.CreateDecryptor(aes.Key, aes.IV);
            var bytes = Convert.FromBase64String(cipherText);
            using var ms = new MemoryStream(bytes);
            using var cs = new CryptoStream(ms, decryptor, CryptoStreamMode.Read);
            using var sr = new StreamReader(cs);
            return sr.ReadToEnd();
        }
    }
}

using Microsoft.Extensions.Configuration;
using Microsoft.IdentityModel.Tokens;
using System.IdentityModel.Tokens.Jwt;
using System.Security.Claims;
using System.Text;
using DataGroup.Interfaces;

namespace DataGroup.Services
```

```
{
    public class JwtService : IJwtService
    {
        private readonly IConfiguration _config;

        public JwtService(IConfiguration config)
        {
            _config = config;
        }

        public string GenerateToken(string username)
        {
            var claims = new[]
            {
                new Claim(ClaimTypes.Name, username)
            };

            var key = new
SymmetricSecurityKey(Encoding.UTF8.GetBytes(_config["Jwt:Key"]));
            var creds = new SigningCredentials(key, SecurityAlgorithms.HmacSha256);

            var token = new JwtSecurityToken(
                issuer: _config["Jwt:Issuer"],
                audience: _config["Jwt:Audience"],
                claims: claims,
                expires: DateTime.UtcNow.AddHours(1),
                signingCredentials: creds);

            return new JwtSecurityTokenHandler().WriteToken(token);
        }
    }
}

using DataGroup.Interfaces;
using DataGroup.Data;
using DataGroup.Models;

namespace DataGroup.Services
{
    public class LogService : ILogService
    {
        private readonly DataGroupContext _context;

        public LogService(DataGroupContext context)
```

```
        {
            _context = context;
        }

        public async Task LogAsync(string action, string? username = null, string?
details = null)
        {
            var entry = new LogEntry
            {
                Action = action,
                Username = username,
                Details = details
            };

            _context.Logs.Add(entry);
            await _context.SaveChangesAsync();
        }
    }
}
```

ДОДАТОК В

Список публікацій

Міністерство освіти і науки України
Хмельницький національний університет

ЗБІРНИК НАУКОВИХ ПРАЦЬ
за матеріалами XVI Всеукраїнської науково-практичної конференції
«Актуальні проблеми комп'ютерних наук АПКН-2024»

15-16 листопада 2024

Хмельницький 2024

УДК 004:37:001:62

Збірник наукових праць за матеріалами XVI Всеукраїнської науково-практичної конференції «Актуальні проблеми комп'ютерних наук АПКН-2024». Хмельницький. 2024. 582с.

У збірнику наукових праць подані перспективні практичні розробки аспірантів, студентів та здобувачів в області сучасних інформаційних технологій. Розглянуто актуальні проблеми комп'ютерних наук, комп'ютерної інженерії, прикладної математики й інженерії програмного забезпечення, приведено ряд робіт по впровадженню інформаційних технологій у виробництво та управління. Висвітлено перспективні розробки сучасних систем пошуку, обробки й захисту інформації, медійних та комунікаційних системи.

УДК 004:37:001:62

Матеріали конференції відтворені з авторських оригіналів, друкуються в авторській редакції та наведені в алфавітному порядку прізвищ авторів. При макетуванні можливі незначні зміни компоновки контенту авторських оригіналів. Відповідальність за якість та зміст публікацій несе автор.

Участь у конференції та складові всіх її етапів (розгляд праць, перевірка на плагіат, макетування, публікація збірника наукових праць та видача сертифікатів) є безкоштовними для всіх учасників. Оргкомітет конференції висловлює подяку учасникам конференції та сподівається на подальшу співпрацю.

З питань проведення конференції та подальшого обміну інформацією звертатись на e-mail конференції: ***apkt.khnu@gmail.com***

АКТУАЛЬНІ ПРОБЛЕМИ КОМП'ЮТЕРНИХ НАУК - 2024

XVI Всеукраїнська науково-практична конференція

Метою конференції є висвітлення актуальних проблем комп'ютерних наук, інформатики та інформаційних технологій.

Робочі мови конференції:

українська, англійська

СЕКЦІЇ КОНФЕРЕНЦІЇ:

1. Комп'ютерні науки та прикладні інформаційні технології.
2. Комп'ютерна інженерія та системи захисту інформації.
3. Математичне моделювання та інженерія програмного забезпечення
4. Телерадіокомунікації, медійні та комунікаційні системи.
5. Проблеми впровадження інформаційних технологій у виробництво та управління.

СПИСОК ОРГАНІЗАЦІЙ, ПРЕДСТАВНИКИ ЯКИХ БРАЛИ УЧАСТЬ У РОБОТІ КОНФЕРЕНЦІЇ:

Донбаська державна машинобудівна академія
Західноукраїнський національний університет
Національний технічний університет «Харківський політехнічний інститут»
Національний університет «Львівська політехніка»
Приватний заклад вищої освіти «ІТ СТЕП Університет»
Сумський державний університет
Харківський національний університет радіоелектроніки
Хмельницький національний університет
Хмельницький фаховий економіко-технологічний коледж УЕП

ОРГКОМІТЕТ КОНФЕРЕНЦІЇ:

Олег СИНЮК – голова оргкомітету, проректор Хмельницького національного університету з наукової роботи, доктор технічних наук, професор.

Тетяна ГОВОРУЩЕНКО – заступник голови оргкомітету, декан факультету інформаційних технологій Хмельницького національного університету, доктор технічних наук, професор.

Олександр БАРМАК – заступник голови оргкомітету, завідувач кафедри комп'ютерних наук Хмельницького національного університету, доктор технічних наук, професор.

Олег САВЕНКО – професор кафедри комп'ютерної інженерії та інформаційних систем Хмельницького національного університету, доктор технічних наук, професор

Олена ВИСОЦЬКА – доктор технічних наук, завідувач кафедри радіоелектронних та біомедичних комп'ютеризованих засобів і технологій Національного аерокосмічного університету ім. М. Є. Жуковського «Харківський авіаційний інститут», професор

Євгеній ЛАВРОВ – доктор технічних наук, професор (Сумський державний університет)

Людмила ТІМОФЄЄВА – відповідальна за студентську науково-дослідну роботу ХНУ

Олександр МАЗУРЕЦЬ – секретар конференції, доцент кафедри комп'ютерних наук Хмельницького національного університету, к.т.н., доцент кафедри комп'ютерних наук ХНУ

Марина МОЛЧАНОВА – секретар конференції, викладач кафедри комп'ютерних наук Хмельницького національного університету

КОНТАКТНА ІНФОРМАЦІЯ:

e-mail для листування: *apkt.khnu@gmail.com*

УДК 004.891

Чешун Д.В., Вишневський Д.Я., Вовкович М.О., Джулій В.М.

Хмельницький національний університет

СТРУКТУРНИЙ СИНТЕЗ РОЗРОБКИ WEB-ДОДАТКІВ

Структурний синтез веб-додатка є багатогранною проблемою, що містить невирішені питання синтезу як всієї архітектури, так і вихідного коду рівнів моделей, представлень і контролерів. Розглянуті методи синтезу покращують показники оптимізації розробки веб-додатків, такі як якість (структурність і повторюваність), але при цьому погіршуються критерії простоти конструкції та можливість модифікацій. На сьогодні існує необхідність створення нових методів зберігання (синтез рівня моделей) і взаємодії (синтез рівня контролерів).

Structural synthesis of a web application is a multifaceted problem, containing unsolved synthesis issues of both the entire architecture and the source code of the model, view, and controller levels. The considered methods of synthesis improve the optimization indicators of web application development, such as quality (structurality and repeatability), but at the same time, the criteria of design simplicity and the possibility of modifications deteriorate. Today, there is a need to create new methods of storage (model level synthesis) and interaction (controller level synthesis).

Останнім часом зростає роль інформаційних технологій в житті суспільства. На даний момент мережа Інтернет являє собою сукупність веб-додатків. Веб-додатки оперують величезними об'ємами даних, в тому числі і конфіденційними, що зумовлює високі вимоги до програмного коду. У той же час, тривалість розробки, погодження та затвердження міжнародних і національних стандартів призводить до їх консерватизму, а також до хронічного відставання вимог і рекомендацій цих документів від сучасної практики та технологій створення складних систем. Сучасні методи розробки систем, їх уніфікація, стандартизація, не повною мірою враховують специфіку розробки на платформі LAMP з використанням вільних інтернет-технологій. В цих умовах створення нових методів розробки веб-додатків є актуальною задачею [1,2].

Програмні системи, що розробляється за технологією MVC, діляться на три частини: класи-контролери, що реалізують взаємодію з користувачем; класи-моделі, що представляють об'єктну модель даних та методи роботи з ними; класи-представлення, що реалізують інтерфейс. На момент створення архітектури MVC веб-додатки були відсутні в принципі. У зв'язку з цим поширеним явищем стало невірне трактування архітектури, що призвело до появи великої кількості не уніфікованого програмного коду, що не задовольняють вимогам якості.

Проблема полягає у відсутності чітких рекомендацій з розробки файлів вихідного коду контролерів, моделей і представлень та специфікації їх

взаємозв'язків. На основі проведеного аналізу можна зробити висновок, що існує необхідність створення методів і алгоритмів оптимізації розробки веб-додатків, які включали б в себе питання зберігання (синтез рівня моделей) і взаємодії (синтез рівня контролерів).

Структурний синтез веб-додатків заснований на певній класифікації вихідного коду веб-додатка. Відповідно до клієнт-серверної технології, файли веб-додатка програмного коду ділиться на клієнтські файли і серверні. До складу клієнтських файлів входять JavaScript-бібліотеки, зображення, файли таблиць стилів, що відповідають за відображення інтерфейсу. До складу серверних файлів входять бібліотеки інтерпретатора PHP (для платформи LAMP), файли конфігурацій і файли тестування, тобто, це код, який виконується на сервері. В основі такої класифікації лежить відділення логіки від представлення або інтерфейсу від реалізації [3,4,5].

Веб-додаток (ВД) можна представити як сукупність файлів, що реалізують набір функцій (1):

$$ВД = < КК,\ КМ,\ ФП,\ Ф3 >, \quad (1)$$

де *КК* - множина класів-контролерів, файлів ВД, які відповідають за логіку поведінки; *КМ* - множина класів-моделей сутностей предметної області; *ФП* - множина файлів представлень, що описують інтерфейс ВД (графічний для користувача і програмний для взаємодії з іншим ВД); *Ф3* - множина файлів завантажувача, в обов'язки якого входить завантажити всі інші частини ВД в пам'ять сервера.

Одним із сучасних методів розробки ВД є метод структурного синтезу «Модель-Представлення-Контролер» (рисунок 1).

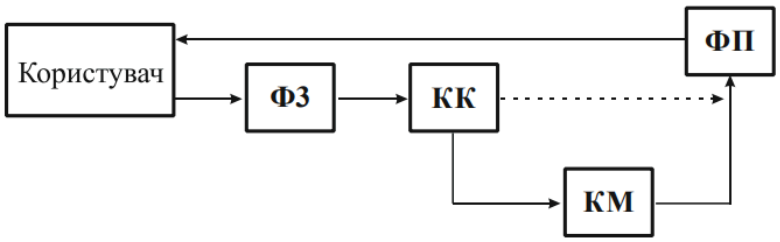


Рисунок 1 – Архітектура «Модель-Представлення-Контролер»

У даному підході програмний код поділяється на три підмножини: множина коду рівня логіки предметної області (рівень контролерів *КК*), множина коду рівня роботи з даними (рівень моделі *КМ*), множина коду зовнішнього представлення (рівень представлення *ФП*). Аналогічно попереднім моделям архітектури виділяється завантажувач (*Ф3*). Модель надає дані (зазвичай для рівня представлення), а також реагує на запити (зазвичай, від контролера), змінюючи свій стан (рисунок 2).

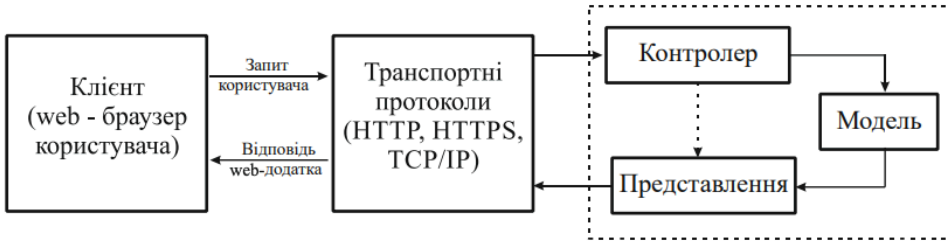


Рисунок 2 – Загальний вид ВД архітектури
«Модель-Представлення-Контролер»

Множина моделей у пам'яті становлять стан ВД, а представлення реалізує відображення інформації. Поведінка (контролер) інтерпретує дані, введені користувачем, та інформує модель і представлення про необхідність відповідної реакції. В якості прикладу ВД такої архітектури можна привести Joomla, Lifestreet, ModX.

При розробці ВД декомпозиції піддається не тільки виконуваний код, але і конфігурація. В МВД архітектурі конфігурація неявно закодована в тексті ВД за допомогою параметрів застосовуваних функцій, які не змінюються залежно від стану S або переданих вхідних впливів v. При виділенні конфігурації операція її завантаження розташовується в Ф32 (рисунок 3).

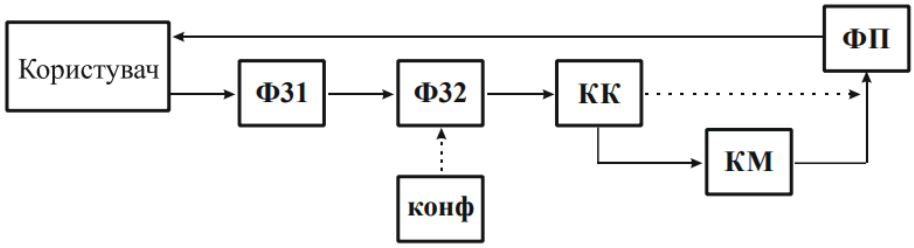


Рисунок 3 – Архітектура ВД з виділеною конфігурацією

У даній архітектурі класи-контролери отримують повністю готове до роботи оточення, яке ініціалізують завантажувачі *Ф31* і *Ф32*.

Запропонована класифікація враховує підходи до синтезу архітектури ВД на концептуальному рівні [6,7].

На основі аналізу множини ВД формалізація задачі синтезу контролерів буде наступною (2):

$$\begin{array}{l}(пкк \in ПКК) \vee (пкк \in СК) \vee (ппк \in М); \\ |ПКК| \to \min; |СК| \to \min; |М| \to \min\end{array}, \quad (2)$$

де *ПКК* - множина програмного коду контролерів, *СК* - множина сервісів, *М* - множина моделей, *пкк* - ділянка програмного коду, що аналізується.

Запропонований лінгвістичний підхід до синтезу контролерів (рисунок 4). У цьому підході методами абстрагування створено універсальний контролер, який отримує запити від користувача на спеціальній мові дій. Виступаючи в ролі транслятора, він перетворює його в виконуваний код ВД, виконуючи зазначені в мові дії. За рахунок цього зовнішній інтерфейс користувача та ВД уніфікується, що є безумовною перевагою.

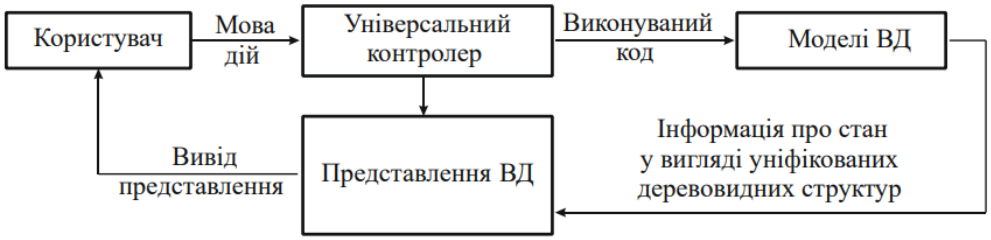


Рисунок 4 – Лінгвістичний підхід до синтезу контролерів

При цьому виникає ряд проблем: розробку форм необхідно проводити з урахуванням вимог інтерфейсу; введення в проект нової мови підвищує його складність; з появою нових вимог виникає необхідність розширювати мову дій і сам універсальний контролер, в результаті чого він може втратити свою універсальність; можливе зниження продуктивності внаслідок необхідності операції розбору команд нової мови.

Розглянуті методи хоч і покращують такі показники якості, як структурність і повторюваність, але при цьому погіршують критерії простоти конструкції та можливість модифікацій.

Розробка ВД для конкретної предметної області відбувається за наступним алгоритмом:

1. Визначення значущих моделей предметної області, сукупності їх властивостей та взаємозв'язків у відповідності з однією із методик моделювання предметної області.

2. Класифікація моделей за функціональною ознакою. Формування однієї або декількох груп моделей. Число модулів, що додаються буде відповідати числу груп моделей. При формуванні груп необхідно дотримуватися принципу мінімізації зв'язків між групами моделей (3):

$$\left| Li_{IN} \right| > \left| Li_{OUT} \right|; \sum_{i \in M} \left| Li_{OUT} \right| \to \min, \qquad (3)$$

де Li_{IN} - множина зв'язків між моделями всередині групи, Li_{OUT} - множина зв'язків між моделями поточної групи і моделями інших груп, M - множина груп моделей.

3. Визначення характеру взаємодії моделей і самих операцій взаємодії. Формування сервісного шару і шару контролерів з допомогою відображення *СЧМВ*-впливів (впливи створення, читання, модифікації, видалення) на *ПСП*-ресурси (ресурси передачі стану представлення) на основі інформації класифікації моделей за функціональною ознакою та графічного інтерфейса.

Структурний синтез веб-додатка є багатогранною проблемою, що містить повністю невирішені питання синтезу як всієї архітектури в цілому, так і вихідного коду рівнів моделей, представлень і контролерів. Існуючі стандарти розробки регламентують тільки протоколи взаємодії веб-додатків, недостатньо враховують специфіку веб-додатків. Низький поріг входження в технологію для розробника створює можливість появи веб-додатків, програмний код яких не відповідає вимогам якості. Внаслідок цього, розроблюваний програмний код веб-додатків не уніфікований і не стандартизований. На сьогодні існує необхідність створення методів і алгоритмів оптимізації розробки веб-додатків, які включали б в себе питання зберігання (синтез рівня моделей) і взаємодії (синтез рівня контролерів).

Перелік посилань

1. Баран, С.В.. Основи web-програмування: Навчальний посібник. / С.В. Баран - Кривий Ріг: Державний університет економіки і технологій, 2023р. - 316 с.
2. Цельсів, О.В. WEB програмування. / О.В. Цельсів - НУУТВ. КПІ 2022р.- 298с.
3. Веллинг, Л. Розробка Web-додатків за допомогою PHP і MySQL / Л. Веллинг, Л. Томсон - Вільямс. 2017р.- 880 с
4. Пасічник, В.В. Web дизайн і web програмування: підручник. / В.В. Пасічник, О.В. Пасічник, Д.І. Угрін - Львів: «Магнолія 2006», 2018р. - 336 с.
5. Матвієнко, О.В. Internet-технології: проектування Webсторінки/ О.В. Матвієнко, І.Л. Бородкіна - К.: ЦНЛ. 2017р. - 154 с.
6. Лук'янов, Б. В. Комп'ютерний аналіз даних / Б. В. Лук'янов – К. : Академія, 2017р.– 345с.
7. Лавров, Є. А. Математичні методи дослідження операцій: підручник / Є. А. Лавров, Л. П. Перхун, В. В. Шендрик – Суми : Сумський державний університет, 2017. – 212 с.

Завідувачу кафедри кібербезпеки
к.т.н., доц. Кльоцу Ю.П.
Вовковича Максима Олександровича
ПІБ здобувача вищої освіти
Студента ФІТ, 4 курсу, групи КБ-21-2

ЗАЯВА

З правилами чинного Положення «Про систему забезпечення академічної доброчесності у Хмельницькому національному університеті» від 31.08.2023, згідно з яким виявлення плагіату є підставою для відмови в допуску кваліфікаційної роботи до захисту та застосування заходів дисциплінарної та академічної відповідальності, ознайомлений. Про використання програмно-технічних засобів для перевірки кваліфікаційних робіт здобувачів вищої освіти на плагіат оповіщений та надаю свою згоду на обробку та збереження університетом моєї роботи в інституційному репозитарії університету.

Також надаю університету право на передачу моєї роботи для обробки та збереження в базах даних програмно-технічних засобів (Unicheck та Anti-Plagiarism) та використання роботи для виявлення плагіату в інших роботах, які перевіряються програмно-технічними засобами та користувачами, що мають доступ до цих програмно-технічних засобів, виключно в обмежених цілях для виявлення плагіату в текстах робіт.

Робота для перевірки університетом надається в друкованому та електронному варіанті. Електронна версія моєї роботи збігається (ідентична) з друкованою.

29.05.2025
дата

підпис

Протокол аналізу звіту подібності науковим керівником

Заявляю, що я ознайомився (-лась) з Повним звітом подібності, який був згенерований Системою виявлення і запобігання плагіату щодо роботи:

Автор: Вовкович Максим Олександрович

Співавтор:

Назва: Система забезпечення безпеки конфіденційних даних від витоку на основі керування паролями

Науковий керівник:

Підрозділ: Кафедра кібербезпеки

Коефіцієнт подібності 1:7.7%

Коефіцієнт подібності 2:3.6%

Мікропробіли: 0

Заміна букв: 0

Інтервали: 0

Білі знаки: 0

Дата створення звіту: 2025-05-30 23:33:15.0

Після аналізу Звіту подібності констатую наступне:

☑ **Запозичення, виявлені в роботі є законними і не є плагіатом. Рівень подібності не перевищує допустимої межі. Таким чином робота незалежна і приймається.**

☐ **Запозичення не є плагіатом, але перевищено граничне значення рівня подібностей. Таким чином робота повертається на доопрацювання.**

☐ **Виявлено запозичення і плагіат або навмисні текстові спотворення (маніпуляції), як передбачувані спроби укриття плагіату, які роблять роботу невідповідною вимогам законодавства (Ст. 32. ЗУ Про вищу освіту, пункт 3.1, Ст. 42. ЗУ Про освіту) та вимог НАЗЯВО (Критерій 5), а також кодексу етики і процедурам. Таким чином робота не приймається.**

Обгрунтування:

01.06.2025р.

Anti-Plagiarism (UA) v-15.281 Educational

The maximum coincidence with one document 1.0%

Dictionaries check: en_US, ru_RU, ua_UA. **Errors in the documents: 10%**

	Document		Sum coincidence on the DB	
ID: 242690 Title: Система забезпечення безпеки конфіденційних даних від витоку на основі керування паролями Added in a DB: 2025-05-30 Authors: Вовкович Максим Олександрович Heads: Джулій В.М. Consultants: Opponents:	Symbols	Lexemes	Symbols	Lexemes
	73318	617	1545 (2%)	20 (3%)

Plagiarism sources

ID	Description	Plagiarism presence in the document	
		Symbols	Lexemes

РІШЕННЯ ЕКСПЕРНОЇ КОМІСІЇ

КАФЕДРИ КІБЕРБЕЗПЕКИ

ПРО ДОПУСК КВАЛІФІКАЦІЙНОЇ РОБОТИ ДО ЗАХИСТУ

Підтверджуємо ознайомлення з результатом звіту подібності щодо роботи, генерованого системою виявлення текстових збігів/ідентичності/схожості:

Назва: Система забезпечення безпеки конфіденційних даних від витоку на основі керування паролями

Автор: Вовкович Максим Олександрович

Спеціальність: 125 – Кібербезпека

Освітня програма:Кібербезпека

Науковий керівник:Володимир ДЖУЛІЙ, канд. техн. наук, доцент

Після аналізу звіту подібності зроблено такий висновок:

№	Висновок	Позначка про відповідність
1	Запозичення, виявлені в роботі, є законними і не є плагіатом (далі – зазначаються підстави віднесення запозичень до правомірних). Робота приймається до захисту.	відповідає
2	Виявлені запозичення не є плагіатом, розміщені в розділах, які не описують безпосередньо авторське дослідження, але кількість цитат перевищує обсяг, виправданий поставленою метою роботи (далі – зазначаються детальні та аргументовані підстави віднесення запозичень до правомірних). Робота приймається до захисту, але має бути відкоригована. Відкоригований варіант має бути поданий на кафедру за 2 дні до захисту, разом із заявою щодо самостійності виконання письмової роботи та ідентичності друкованої та електронної версії роботи.	
3	Виявлені запозичення не є плагіатом, але частково розміщені в розділах, які описують безпосередньо авторське дослідження, а кількість цитат перевищує обсяг, виправданий поставленою метою роботи. В зв'язку з цим мета роботи та поставлені завдання не були досягнені. Робота може бути допущена до захисту (наступного року) після того як буде відкоригована та допрацьована і успішно пройде повторну перевірку на академічний плагіат.	
4	Робота містить навмисні текстові спотворення, передбачувані спроби укриття запозичень або інші прояви академічного плагіату. Робота містить фабрикацію або фальсифікацію даних. Робота не допускається до захисту.	
5	Інше:	

Підтвердження:

Оригінальність тексту роботи за результатами перевірки системою StrikePlagiarism складає 92.3%, оригінальність тексту роботи за результатами перевірки системою Anti-Plagiarism складає 99%.

Згідно з правилами чинного Положення «Про систему забезпечення академічної доброчесності у Хмельницькому національному університеті» від 24.09.2024, авторська робота, обсяг оригінального тексту у відсотках до загального обсягу матеріалу в якій складає 90-100%, визначається роботою з високою унікальністю тексту і допускається до захисту.

Керівник роботи _____ Володимир ДЖУЛІЙ

Гарант ОП _____ Віктор ЧЕШУН

Завідувач кафедри кібербезпеки _____ Юрій КЛЬОЦ

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ХМЕЛЬНИЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

РЕЦЕНЗІЯ НА КВАЛІФІКАЦІЙНУ РОБОТУ

освітнього ступеня «бакалавр»

Студент Вовкович Максим Олександрович

Тема Система забезпечення безпеки конфіденційних даних від витоку на основі керування паролями

Спеціальність 125 – Кібербезпека

Обсяг кваліфікаційної роботи освітньо-кваліфікаційного рівня «бакалавр»:

кількість листів креслень 3; кількість сторінок записки 62.

1. Короткий зміст роботи та прийнятих рішень У кваліфікаційній роботі була розроблена захищена програмна система керування паролями для підприємства «DataGroup». Основною метою створеної системи є забезпечення конфіденційності, цілісності та доступності облікових даних користувачів, а також запобігання витокам інформації через ненадійні або повторно використані паролі. У процесі проєктування було реалізовано такі ключові компоненти: механізм автентифікації користувачів із застосуванням JWT-токенів, шифрування даних (AES), збереження хешованих паролів (BCrypt), система логування дій користувачів, а також CRUD-інтерфейс для роботи з конфіденційними документами. Усі дані зберігаються у базі PostgreSQL у зашифрованому вигляді. Окремо було впроваджено модульне тестування основних сервісів системи, а також описано конфігурацію і принципи розгортання системи в умовах локального середовища.

2. Висновок про відповідність кваліфікаційної роботи завданню.У кваліфікаційній роботі було виконано поставлене завдання як у теоретичній, так і в практичній частині. Було дотримано усіх вимог щодо виконання

3. Характеристика виконання кожного розділу роботи, ступінь використання останніх досягнень науки і техніки і передових методів роботи: У першому розділі роботи було проведено глибокий аналіз предметної області, охарактеризовано основні проблеми зберігання та створення паролів, а також вивчено сучасні підходи до забезпечення їх захисту. На основі аналізу сформульовано вимоги до майбутньої системи та обґрунтовано актуальність її розробки. У другому розділі було розроблено алгоритми роботи основних функціональних модулів — генерації паролів, шифрування, дешифрування, логування, автентифікації. Ретельно спроєктовано архітектуру системи з використанням моделей MVC і Onion Architecture. У третьому розділі описано реалізацію системи засобами ASP.NET Core MVC, PostgreSQL, JWT, AES, BCrypt. Також проведено тестування всіх основних сервісів з використанням xUnit і Moq, реалізовано локальне розгортання та конфігурацію середовища. У роботі використано сучасні бібліотеки, актуальні методи захисту даних, принципи безпечного зберігання інформації та автоматизованого тестування.

4. Позитивні сторони роботи Розроблена система є актуальним інструментом для підприємств, які прагнуть забезпечити ефективне та безпечне управління обліковими даними своїх користувачів. Робота демонструє комплексний підхід до захисту даних: від генерації пароля до логування дій користувача. Обрана архітектура забезпечує гнучкість, масштабованість і простоту підтримки проєкту. Реалізація усіх ключових компонентів, включаючи захищене збереження паролів, автентифікацію за допомогою JWT, а також шифрування конфіденційних даних — відповідає сучасним стандартам кібербезпеки. Система може бути легко адаптована до потреб реального підприємства та допрацьована з урахуванням індивідуальних вимог.

5. Негативні сторони роботи На поточному етапі система не має веб-інтерфейсу адміністратора з гнучкими налаштуваннями політик паролів, як-от MFA чи тимчасове блокування обліковки після серії помилок. Крім того, система логування не реалізує візуалізацію логів або автоматичну обробку підозрілих активностей. Інтерфейс реалізований у вигляді базових Razor-форм, що хоча й забезпечують простоту, однак не мають повноцінної UI-оболонки. Також відсутня інтеграція з зовнішніми API для виявлення скомпрометованих паролів. Ці можливості можуть бути реалізовані в майбутніх версіях системи у межах подальшого розвитку.

6. Оцінка графічного оформлення та пояснювальної записки роботи. Графічне оформлення кваліфікаційної роботи відповідає темі роботи та виконане з дотриманням стандартів. У цілому графічне оформлення є чітким та якісним, а пояснювальна записка відповідає нормам оформлення.

7. Відгук про роботу в цілому Кваліфікаційна робота заслуговує позитивної оцінки, оскільки весь матеріал роботи є структурованим, чітким та послідовним. Усі розділи роботи мають логічну послідовність, що сприяє зрозумінню викладеного матеріалу в рамках теми роботи. Графічний матеріал допомагає наочно продемонструвати доцільність та ефективність прийнятих рішень для досягнення мети.

8. Інші зауваження

9. Оцінка кваліфікаційної роботи Ураховуючи всі позитивні та негативні сторони представленої кваліфікаційної роботи, можна зробити висновок, що вона заслуговує оцінки «відмінно».

РЕЦЕНЗЕНТ (прізвище, ім'я, по батькові, посада, місце роботи)
Бойко Юлій Миколайович,
доктор технічних наук, професор кафедри ТМІТ

« 09 » червня 2025. (підпис)