Хмельницький національний університет
Факультет інформаційних технологій
Кафедра кібербезпеки

КВАЛІФІКАЦІЙНА РОБОТА

Розгона Іллі Дмитровича

на здобуття ступеня вищої освіти Бакалавра

Система захисту програмного комплексу фінансового документообігу з web-архітектурою

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

Шифр КРБКБ.2101128.21.01.13 ПЗ

Виконав: студент 4 курсу, група КБ-21-1 _____ Ілля РОЗГОН
Керівник: д-р технічних наук, професор _____ Михайло КАСЯНЧУК
Нормоконтролер старший викладач _____ Сергій МОСТОВИЙ

До захисту допускаю:
Завідувач кафедри кібербезпеки _____ Юрій КЛЬОЦ

11 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.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

Розгон Ілля Дмитрович
(Прізвище,ім'я, по батькові студента)

1 Тема роботи Система захисту програмного комплексу фінансового документообігу з web-архітектурою

Керівник роботи Михайло КАСЯНЧУК

Затверджено наказом ректора університету від 7 лютого 2025 № 23

2 Строк подання студентом кваліфікаційної роботи на кафедру

3 Вихідні дані до роботи Лістинги тестових скриптів для атак, результати аудиту, тестування системи до/після впровадження шифрування

4 Зміст пояснювальної записки (перелік питань, які потрібно розробити)
Аналіз предметної області, огляд методів захисту, проектування та впровадження системи, тестування, висновки.

5 Перелік графічного матеріалу (із зазначенням обов'язкових креслень)
Схеми архітектури, модель загроз, алгоритм шифрування, результати тестування.

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	виконав
Ознайомлення з предметною областю	Лютий	виконав
Дослідження існуючих рішень	Лютий	виконав
Постановка задачі	Березень	виконав
Визначення загальних принципів рішення задачі	Березень	виконав
Деталізація принципів рішення задачі	Квітень	виконав
Розробка проєктних рішень	Квітень	виконав
Апробація проєктних рішень	Травень	виконав
Оформлення пояснювальної записки згідно вимог	Травень	виконав
Оформлення графічної частини	Червень	виконав
Захист КР	Червень	виконав

Студент _____ Ілля РОЗГОН

Керівник кваліфікаційної роботи _____ Михайло КАСЯНЧУК

ABSTRACT

Theme of the qualification work: Information security system of a financial document management software suite with web architecture.

Author of the work: Illia Dmytrovych Rozghon.

Supervisor of the work: Mykhailo Mykolaiovych Kasianchuk.

Explanatory note: 69 p., 2 appendices, 8 figures, 2 tables, 51 sources.

Graphic part: 4 posters, 9 presentation slides.

END-TO-END ENCRYPTION, SCRIPT FILTER, ROUTE-BASED SECURITY CONFIGURATION, RATE LIMITING FILTER.

The bachelor's qualification work is devoted to the development of a security system for a financial document management software suite with web architecture. The study analyzes information security threats typical for web applications, especially in the context of limited resources of small businesses, and proposes effective protection methods.

Modern approaches to data security are examined, including the implementation of end-to-end encryption, role-based access control (RBAC), and protection against common attacks such as SQL injection, XSS, and CSRF.

The proposed system was modeled based on a threat model and attacker model and tested before and after the implementation of protective mechanisms. Special attention was paid to the confidentiality of data transmission and the integration of cryptographic algorithms into the system's communication functionality.

06.06.2025 _____

АНОТАЦІЯ

Тема кваліфікаційної роботи: Система захисту програмного комплексу фінансового документообігу з web-архітектурою.

Автор роботи: Розгон Ілля Дмитрович.

Керівник роботи: Касянчук Михайло Миколайович.

Пояснювальна записка: 69 с., 2 додатки, 8 рисунків, 2 таблиці, 51 джерел.

Графічна частина: 4 плакати, 9 презентаційних слайдів.

НАСКРІЗНЕ ШИФРУВАННЯ, ФІЛЬТР СКРИПТІВ, КОНІФГУРАЦІЯ БЕЗПЕКИ З ОБМЕЖЕННЯМ МАРШРУТІВ, ФІЛЬТР ОБМЕЖЕННЯ КІЛЬКОСТІ ЗАПИТІВ.

Кваліфікаційна робота бакалавра присвячена розробці системи захисту програмного комплексу фінансового документообігу з web-архітектурою. У роботі проведено аналіз загроз інформаційній безпеці, що виникають у веб-додатках, зокрема в умовах обмежених ресурсів малого бізнесу, та запропоновано ефективні засоби захисту.

Розглянуто сучасні підходи до захисту інформації, включаючи використання end-to-end шифрування, системи управління доступом на основі ролей (RBAC), а також захист від поширених атак (SQL-ін'єкції, XSS, CSRF).

Розроблена система моделювалася з урахуванням моделі загроз та моделі порушника, пройшла тестування до та після впровадження захисних механізмів. Особливу увагу приділено конфіденційності передавання даних та інтеграції криптографічних алгоритмів у комунікаційний функціонал системи.

06.06.2025 _____

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ

АГК – Алгоритм генерації ключа

АКП – Алгоритм кінцевого перетворення

ДСТУ – Державний стандарт України

ІТ – Інформаційні технології

ІКС – Інформаційно-комунікаційна система

ПЗ – Програмне забезпечення

ОС – Операційна система

КСЗІ – Комплексна система захисту інформації

ЦВЗ – Цифровий водяний знак

AES – Advanced Encryption Standard

DES – Data Encryption Standard

IDEA – International Data Encryption Algorithm

PES – Proposed Encryption Standard

PKC – Public-key cryptography

MFA – Multi-Factor Authentication

RBAC – Role-Based Access Control

MVC – Model-View-Controller

OWASP – Open Web Application Security Project

ORM – Object-Relational Mapping

ВСТУП

Сучасний розвиток інформаційних технологій та впровадження web-архітектури в бізнес-процеси відкривають нові можливості для автоматизації та оптимізації фінансового документообігу. Водночас зростає кількість загроз інформаційній безпеці, що вимагає постійного вдосконалення систем захисту. Особливо актуальною ця проблема є для малих комерційних підприємств, де ресурсів на забезпечення високого рівня безпеки часто недостатньо.

Метою даної кваліфікаційної роботи є створення системи захисту програмного комплексу, що забезпечує захищений документообіг на підприємстві, з урахуванням сучасних підходів до web-архітектури. Практична цінність роботи обумовлена необхідністю покращення існуючої системи фінансового документообігу шляхом проведення аудиту, виявлення уразливостей та впровадження додаткових заходів захисту.

Практична цінність роботи полягає у створенні прикладного рішення, яке можна інтегрувати в реальні бізнес-процеси малого підприємства з мінімальними витратами. Запропонована система враховує обмеження ресурсів та рівень технічної підготовки користувачів, що робить її придатною для впровадження в умовах малого бізнесу. Результати роботи можуть бути використані як база для модернізації вже існуючих систем документообігу з метою посилення їх захисту, а також слугувати шаблоном для впровадження безпечної комунікації в інших галузях, де конфіденційність фінансової інформації є критично важливою.

Одним із ключових напрямків удосконалення системи є впровадження end-to-end шифрування у чаті, що використовується для внутрішньої комунікації користувачів. Початкова версія системи характеризувалася використанням нешифрованого каналу зв’язку, що створювало серйозні ризики щодо витоку конфіденційної інформації, зокрема фінансових даних підприємства. Впровадження шифрування дозволило забезпечити конфіденційність передачі даних, знизивши ймовірність перехоплення інформації зловмисниками та підвищити загальний рівень безпеки системи.

Для досягнення мети роботи необхідно виконати такі завдання:

- Проведення детального аналізу предметної області та вивчення існуючих програмних комплексів фінансового документообігу;
- Дослідження сучасних методів захисту інформації, особливо в контексті web-архітектури;
- Проведення аудиту існуючого програмного комплексу з метою виявлення вразливостей, зокрема у початковій реалізації чату без шифрування;
- Розробка моделі загроз інформаційної безпеки з врахуванням специфіки роботи системи та потенційних векторів атак;
- Створення моделі порушника, яка охоплює як зовнішніх зловмисників, так і потенційних внутрішніх користувачів з недопустимими діями;
- Проектування та впровадження системи захисту, що включає сучасні криптографічні методи, зокрема реалізацію end-to-end шифрування для забезпечення безпеки комунікації.

Результатом виконання даної кваліфікаційної роботи стане розробка ефективної системи захисту вебдодатку, яка забезпечить комплексний підхід до захисту програмного комплексу, орієнтованого на автоматизацію фінансового документообігу. Цей підхід передбачає не лише впровадження технічних рішень, спрямованих на усунення вразливостей, а й формування цілісної архітектури безпеки, здатної адаптуватися до змін у загрозовому середовищі та відповідати сучасним вимогам інформаційної безпеки. Очікується, що створена система дозволить виявити як найбільш очевидні, так і приховані вразливості в логіці роботи та структурі програмного продукту, а також визначити напрями для подальшого вдосконалення.

Однією з ключових складових цієї роботи є проведення аналізу еволюції інформаційної безпеки програмного комплексу — від його первинної версії, де відсутні засоби шифрування, до інтеграції сучасних криптографічних алгоритмів і захисних протоколів. У процесі реалізації розглядаються такі аспекти, як захист даних у стані спокою та під час передачі, управління сесіями, автентифікація користувачів, а також можливість використання end-to-end шифрування для

захисту каналу комунікації між користувачами. Результати цих заходів дозволяють об'єктивно оцінити вплив впроваджених рішень на загальний рівень захисту системи та вказати на ефективність обраних методів.

Актуальність теми зумовлена стрімким зростанням кількості кіберінцидентів, спрямованих на фінансові системи, що функціонують у вебсередовищі. У зв’язку з цим підвищення рівня безпеки вебдодатків стало одним із ключових завдань сучасної інформаційної безпеки. Саме тому розробка доступної, ефективної та гнучкої системи захисту є вкрай важливою для бізнесів, особливо малих і середніх підприємств, які не мають змоги впроваджувати комерційні дорогі рішення. Представлена в роботі система передбачає використання переважно відкритих технологій або інструментів із низькою вартістю, що робить її привабливою для практичного використання в умовах обмежених ресурсів.

За темою кваліфікаційної роботи були опубліковані тези конференцій: «Система захисту програмного комплексу фінансового документообігу з веб-архітектурою» у збірнику тез доповідей XX міжнародної науково-практичної конференції «Військова освіта і наука: сьогодення та майбутнє»

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ, ВИВЧЕННЯ ІСНУЮЧИХ ПРОГРАМНИХ КОМПЛЕКСІВ ТА МЕТОДІВ ЇХ ЗАХИСТУ

1.1 Огляд програмних комплексів фінансового документообігу

Програмні комплекси фінансового документообігу становлять основу автоматизації фінансових процесів на підприємстві. Сучасні рішення дозволяють не лише вести облік, а й здійснювати аналіз, прогнозування, інтегрувати бізнес-процеси та забезпечувати високий рівень контролю за фінансовою інформацією. Вони є ключовим інструментом цифрової трансформації підприємств у різних галузях.

Бухгалтерський облік і формування звітності є однією з основних функцій сучасних програмних комплексів фінансового документообігу. Вони дозволяють здійснювати точний облік фінансових операцій, контролювати грошові потоки, формувати регламентовану та управлінську звітність, що є необхідною умовою ефективного управління фінансами підприємства [1].

SAP Financials є прикладом комплексної системи, яка включає модулі для ведення бухгалтерського обліку, контролю грошових потоків, управління активами та пасивами. Завдяки підтримці багатомовності та мультивалютності ця система зручна для використання міжнародними компаніями та корпоративними структурами [2].

Oracle Financials Cloud забезпечує автоматизацію фінансового обліку, управління витратами, а також розрахунки з постачальниками та клієнтами. Система інтегрується з іншими модулями ERP, дозволяючи формувати повну картину фінансового стану компанії [3].

Odoo, як система з відкритим кодом, пропонує модулі для бухгалтерії, аналітики, бюджетування та формування звітності. Її модульна структура дає можливість адаптувати систему до потреб малого та середнього бізнесу, забезпечуючи гнучкість у налаштуванні [4].

Управління фінансами та бюджетування також є важливою складовою сучасних фінансових систем. Зокрема, Oracle Financials Cloud дозволяє автоматизувати управління бюджетами й оперативно реагувати на зміни ринку. Деякі системи використовують алгоритми машинного навчання для підвищення точності прогнозування фінансових результатів [5].

Ще однією критично важливою функцією є електронний документообіг. Наприклад, Odoo пропонує централізоване управління документами з можливістю збереження, контролю змін, керування доступом, пошуку й резервного копіювання. Це дозволяє зменшити ризики втрати інформації та підвищити ефективність обробки документів.

Інтеграція та обмін даними. Сучасні комплекси, такі як SAP S/4HANA Cloud, дозволяють інтегрувати фінансові дані з іншими бізнес-процесами, наприклад, з логістикою, управлінням продажами, CRM. Це створює єдине інформаційне середовище, де дані оновлюються в режимі реального часу. Інтерфейси API та підтримка стандартів обміну даними (XML, JSON) сприяють безшовній інтеграції з банківськими системами та іншими зовнішніми сервісами [6].

Web-архітектура передбачає використання сучасних технологій, які забезпечують масштабованість, високу доступність і зручність віддаленого доступу до системи через веб-браузери або мобільні додатки. Така архітектура дозволяє будувати модульні, розподілені системи, де кожен компонент відповідає за окрему функцію. Це сприяє легшому оновленню, супроводу та інтеграції із зовнішніми сервісами [7].

Хмарні рішення, зокрема Oracle Financials Cloud та SAP S/4HANA Cloud, базуються на використанні дата-центрів провайдерів, що дозволяє підприємствам уникати значних витрат на власну інфраструктуру. Такі системи забезпечують гнучкість, автоматичне масштабування та регулярне оновлення програмного забезпечення, що гарантує стабільну та безперебійну роботу бізнес-процесів [8].

Модульність та інтеграційність. Системи типу Odoo характеризуються модульною структурою, де кожен модуль відповідає за окрему функціональну

область: бухгалтерія, документообіг, CRM, управління проектами. Це дозволяє підприємствам впроваджувати лише ті модулі, які відповідають їхнім потребам, а згодом легко інтегрувати додаткові функції. Інтеграція через RESTful API, веб-сервіси та стандартні протоколи забезпечує можливість обміну даними між різними системами, що є ключовим для створення єдиного інформаційного простору [9].

Віддалений доступ та мобільність. Використання веб-технологій дозволяє користувачам працювати з системою незалежно від географічного розташування. Наприклад, менеджери можуть отримувати оперативний доступ до фінансової звітності через мобільні додатки або веб-браузери, що підвищує гнучкість управління підприємством. Рішення, що працюють у хмарі, часто забезпечують автоматичне масштабування, що дозволяє системі адаптуватися до змін у навантаженні без втрати продуктивності [10].

Переваги існуючих рішень. Автоматизація та оптимізація бізнес-процесів. Комплексні рішення значно знижують потребу в ручному введенні даних, що мінімізує ризик помилок і дозволяє оптимізувати роботу підприємства. Автоматизація створює основу для більш ефективного аналізу фінансової інформації [11].

Гнучкість та масштабованість. Хмарні платформи, такі як Oracle Financials Cloud, дозволяють підприємствам швидко масштабувати свої системи в залежності від зростання обсягів даних та розширення бізнесу. Це особливо важливо для підприємств, що планують швидке зростання.

Модульність. Можливість обирати окремі модулі (наприклад, управління витратами, електронний документообіг) дозволяє налаштувати систему відповідно до специфіки роботи підприємства, зменшуючи витрати на впровадження неактуальних для бізнесу функцій [12].

Віддалений доступ і мобільність. Завдяки web-архітектурі користувачі можуть отримувати доступ до системи з будь-якого місця, що сприяє оперативності прийняття управлінських рішень. Це також полегшує роботу в умовах гнучкого графіка або віддаленого офісу.

Недоліки існуючих рішень. Висока вартість впровадження та підтримки. Системи типу SAP Financials або Oracle Financials Cloud часто вимагають значних початкових інвестицій, що може бути критичним для малих підприємств. Ліцензійні платежі, налаштування та постійна підтримка системи можуть стати фінансовим навантаженням.

Складність інтеграції. Інтеграція нових модулів із існуючими бізнес-процесами може вимагати значних технічних ресурсів та часу. Підприємствам необхідно ретельно планувати міграцію даних і налаштування інтерфейсів, щоб уникнути простоїв і помилок.

Питання безпеки. Використання web-технологій відкриває нові вектори для кібератак. Незважаючи на впровадження сучасних засобів захисту (шифрування, MFA, моніторинг активності), завжди існує ризик експлуатації вразливостей у веб-інтерфейсах або API. Тому важливе постійне оновлення системи та моніторинг безпеки.

Залежність від інтернет-з’єднання. Для хмарних рішень критичною є стабільність інтернет-з’єднання. У разі виникнення проблем з мережею може виникнути тимчасова недоступність системи, що впливає на оперативність прийняття рішень.

1.2 Методи забезпечення безпеки в програмних комплексах

Захист інформації в системах фінансового документообігу є критично важливим завданням, оскільки дані часто мають конфіденційний характер і можуть бути мішенню для численних кібератак. Сучасні системи застосовують комплексний підхід, який включає криптографічні методи, механізми аутентифікації та авторизації, захист каналу зв’язку, а також системи управління доступом і аудитів. Нижче наведено детальний опис основних методів забезпечення безпеки з прикладами і використанням відповідних скорочень.

Криптографія є основним засобом забезпечення конфіденційності, цілісності та автентичності даних. В системах фінансового документообігу застосовуються наступні криптографічні підходи:

AES: Високоефективний стандарт, який забезпечує сильний рівень захисту за допомогою блочного шифрування. AES широко використовується завдяки своїй швидкості та надійності [13].

DES: Хоча DES вже вважається застарілим через короткий розмір ключа (56 біт), його принципи залишаються базою для розробки нових алгоритмів [14].

IDEA: Забезпечує високий рівень захисту завдяки використанню багатьох раундів шифрування та складної системи замін [15].

PES: Використовується в окремих системах як альтернатива стандартним алгоритмам, забезпечуючи додатковий рівень інновацій у криптографії [16].

Для симетричного шифрування використовується АГК, який дозволяє створити криптографічний ключ, що використовується для шифрування та розшифрування даних. Генерація ключа має бути стійкою до криптографічних атак, а його безпечне зберігання – гарантією конфіденційності інформації.

Асиметричне шифрування PKC Використовується для обміну ключами, електронного підпису та аутентифікації учасників комунікації. Підхід передбачає використання пари ключів: відкритого та закритого. Закритий ключ залишається конфіденційним, а відкритий може бути розповсюджений для перевірки цифрового підпису. Цей метод дозволяє захистити канали зв’язку, де симетричне шифрування застосовується після безпечного обміну ключами [17].

ЦВЗ забезпечують автентичність та цілісність даних. Використовуючи алгоритми асиметричного шифрування, створюється унікальний підпис, який гарантує, що дані не були змінені під час передачі та походять від справжнього відправника. ЦВЗ використовується для захисту авторських прав та перевірки справжності документів. Цей метод вбудовує захисні маркери без зміни основного змісту документу [18].

Протоколи безпеки – SSL/TLS: SSL (Secure Sockets Layer) / TLS (Transport Layer Security) забезпечують захист даних при їх передачі через мережу.

Протоколи використовують комбінацію симетричного та асиметричного шифрування для створення безпечного з’єднання між клієнтом та сервером. Використання SSL/TLS гарантує, що дані, що передаються між користувачами та сервером системи, захищені від прослуховування, підміни або атак «людина посередині». Впровадження даних протоколів відповідає вимогам стандартів, зокрема ДСТУ, що регламентують безпеку ІТ [19].

Забезпечення доступу до системи – критично важливий аспект захисту. Механізми аутентифікації та авторизації дозволяють впевнитися, що лише уповноважені користувачі отримують доступ до чутливої інформації.

Паролі та логіни найбільш базовий метод, що забезпечує первинну перевірку користувача. Однак потребує додаткового посилення (наприклад, політики складності паролів).

MFA включає використання кількох методів перевірки, наприклад, пароля, смарт-карти або мобільного додатку для отримання коду. Це значно знижує ризик несанкціонованого доступу. Сертифікати: Використання цифрових сертифікатів, заснованих на PKC, дозволяє перевірити автентичність користувача, що особливо актуально для web-інтерфейсів та мобільних додатків [20].

Авторизація визначає рівень доступу користувача до різних ресурсів системи. Використовуються системи керування доступом, які можуть бути інтегровані в КСЗІ, що забезпечують чіткий контроль над правами доступу на основі ролей. Сучасні рішення часто застосовують політики надання доступу, які дозволяють автоматично визначати дозволи користувача залежно від його ролі, посадових обов’язків або контексту сесії.

Захист каналу зв’язку захист комунікаційних каналів є невід'ємною складовою безпеки інформації, що передається між клієнтами та серверами. Використання VPN (Virtual Private Network) VPN-технології створюють захищене з’єднання через публічну мережу, забезпечуючи шифрування всіх даних, що передаються між віддаленими користувачами та корпоративною мережею. Це дозволяє запобігти можливості перехоплення інформації та несанкціонованого доступу до внутрішніх ресурсів організації. VPN може застосовуватися як

додатковий рівень захисту для співробітників, що працюють віддалено, забезпечуючи конфіденційність та цілісність даних [21].

Інші засоби захисту каналу зв’язку. Шифрування на рівні транспортного протоколу окрім SSL/TLS, можуть використовуватись інші протоколи, що забезпечують шифрування даних, наприклад, IPsec [22].

Фаєрволи та системи виявлення вторгнень (IDS/IPS) використовуються для моніторингу трафіку та виявлення підозрілих активностей, що можуть свідчити про спробу несанкціонованого доступу або кібератак [23].

Роль систем управління доступом і аудитів для забезпечення комплексного захисту системи важливим є не лише шифрування та аутентифікація, але й постійний контроль за діями користувачів та системних подіями.

Системи управління доступом забезпечують централізоване управління правами користувачів до ресурсів системи. Використання ролевих моделей доступу дозволяє обмежувати права доступу до даних відповідно до посадових обов’язків та конкретних завдань. В системах, інтегрованих у ІКС, впровадження політик доступу є важливим елементом забезпечення безпеки.

Системи аудиту та моніторингу дозволяють вести журнал подій, що відображають спроби входу в систему, зміни у конфігурації, доступ до конфіденційних даних та інші критичні операції. Регулярний аудит допомагає ідентифікувати потенційні загрози та виявляти порушення безпеки ще на ранніх стадіях. Системи моніторингу інтегровані із засобами управління доступом дозволяють автоматично реагувати на підозрілі активності, в тому числі шляхом блокування доступу або сповіщення відповідального персоналу [24].

1.3 Аналіз нормативно-правової бази інформаційної безпеки в Україні

Нормативно-правова база інформаційної безпеки в Україні формує правовий фундамент для захисту конфіденційної інформації, забезпечення стабільної роботи ІТ-систем, ПЗ та ІКС, а також регулює питання захисту персональних даних і корпоративних секретів. Цей розділ надає всебічний огляд

законодавчих вимог, міжнародних стандартів та специфічних нормативних документів, що регулюють діяльність підприємств, зокрема малого бізнесу, у сфері захисту інформації.

Основні законодавчі документи:

- Закон України «Про захист персональних даних»: цей закон встановлює загальні принципи обробки, зберігання та передачі персональних даних. Він обов'язковий для всіх організацій, які працюють з конфіденційною інформацією, незалежно від розміру підприємства. Закон передбачає, що застосування сучасних криптографічних методів (наприклад, симетричного шифрування з алгоритмами AES, DES, IDEA або PES, а також використання РКС) є обов'язковим для забезпечення цілісності та конфіденційності даних [25].

- Закон України «Про інформацію»: закон регулює порядок розповсюдження, використання та захисту інформації, включаючи питання публічності та доступності. Він визначає вимоги до захисту як внутрішньої, так і зовнішньої інформації, що має значення для організацій, що використовують ІТ. В цьому контексті важливе впровадження комплексних систем захисту інформації (КСЗІ) та використання стандартних підходів до управління доступом [26].

- Закон України «Про кібербезпеку»: цей закон створює правову базу для впровадження заходів протидії кіберзагрозам. Він охоплює питання захисту мережевої інфраструктури, захисту даних, що передаються через публічні канали, а також впровадження криптографічних засобів для шифрування інформації. Закон активно стимулює використання VPN для захисту каналів зв'язку та забезпечує нормативну базу для моніторингу та реагування на кібератаки [27].

Законодавство вимагає застосування передових технологій захисту, що включають використання сучасних криптографічних алгоритмів. Наприклад, для симетричного шифрування застосовують такі алгоритми як AES, DES, IDEA або PES. Ці алгоритми повинні використовуватися разом із надійними методами генерації ключів, що забезпечуються через АГК, а також алгоритмами кінцевого перетворення (АКП) для підтримки стійкості до атак. Забезпечення безпеки ПЗ та ІКС вимагає, щоб розробники враховували специфічні вимоги щодо захисту

даних, інтегрували криптографічні засоби та дотримувалися положень законодавства [28].

Законодавчі документи акцентують увагу на необхідності захисту ОС та ПЗ через впровадження КСЗІ, що включає заходи як апаратного, так і програмного характеру. Використання цифрових водяних знаків (ЦВЗ) для захисту авторських прав та підтвердження справжності документів також рекомендується, особливо в контексті захисту електронного документообігу [29].

ISO/IEC 27001: Цей стандарт визначає вимоги до систем управління інформаційною безпекою (СУІБ). Він охоплює всі аспекти забезпечення безпеки, від фізичного захисту до технічних заходів, таких як криптографічний захист даних. Впровадження ISO/IEC 27001 дозволяє організаціям систематизувати ризики, впроваджувати заходи захисту та проводити регулярний аудит безпеки. ISO/IEC 27002: Надає практичні рекомендації для впровадження заходів безпеки, визначаючи політики, процедури, контроль доступу та механізми моніторингу. Стандарт є комплементарним до ISO/IEC 27001 і служить методологічною базою для розробки внутрішніх документів щодо безпеки. ISO/IEC 15408 (Common Criteria): Стандарт використовується для оцінки безпеки ПЗ та ІКС. Він встановлює вимоги до проведення сертифікації систем безпеки, що дає змогу забезпечити об'єктивну оцінку надійності застосовуваних криптографічних алгоритмів (AES, DES, IDEA, PES) та механізмів захисту інформації [30].

Багато українських підприємств, особливо ті, що працюють у сфері фінансового документообігу, активно впроваджують стандарти ISO як частину своєї політики забезпечення безпеки. Це дозволяє не тільки підвищити рівень захисту даних, а й забезпечити відповідність вимогам міжнародних партнерів. Відповідність стандартам сприяє формуванню єдиних процедур, що охоплюють управління доступом, контроль над використанням криптографічних методів (з використанням АГК, АКП та РКС) та моніторинг систем з використанням КСЗІ. Підготовка персоналу, аудит систем та впровадження регулярних перевірок відповідно до ISO/IEC 27001 забезпечують стабільну роботу IT-систем у відповідності до найвищих вимог інформаційної безпеки.

Для підприємств малого бізнесу існують спеціалізовані нормативні документи, що враховують обмежені ресурси та специфіку організацій малого масштабу. Ці документи розроблені Міністерством економіки, Міністерством цифрової трансформації України та іншими державними установами. Рекомендації для малого бізнесу включають впровадження ефективних, але економічно вигідних заходів безпеки. Наприклад, використання відкритих рішень, таких як Odoo, з інтегрованими модулями безпеки, що відповідають вимогам ДСТУ, є прикладом адаптації сучасних технологій захисту.

ДСТУ охоплює вимоги до криптографічного захисту, що включають застосування алгоритмів шифрування (AES, DES, IDEA, PES) та забезпечення цілісності даних. Стандарти ДСТУ впроваджуються як у державних, так і в приватних організаціях, зокрема в ПЗ для фінансового документообігу, що гарантує відповідність місцевим нормативним вимогам та створення стабільної системи захисту.

Підтримка інновацій та адаптація нових технологій:

- Нормативна база також стимулює впровадження інновацій в галузі ІТ. Законодавство спрямоване на модернізацію систем захисту за рахунок інтеграції новітніх технологій – наприклад, використання сучасних методів аутентифікації (багатофакторної аутентифікації) та впровадження нових підходів до управління доступом.

- Підприємства малого бізнесу отримують рекомендації щодо створення комплексних систем захисту КСЗІ із застосуванням стандартних криптографічних алгоритмів, використанням АГК, АКП, цифрових підписів та ЦВЗ для захисту інформації.

- Державні ініціативи спрямовані на підтримку розвитку інноваційних ІТ-рішень дозволяють малому бізнесу впроваджувати сучасні заходи безпеки, не витрачаючи значних коштів на розробку власних систем захисту.

1.4 Сучасні підходи до веб-архітектури та їх вплив на безпеку

Сучасна web-архітектура розвивається з урахуванням потреб високої масштабованості, доступності та гнучкості систем, що надають послуги через Інтернет. Проте ці переваги супроводжуються рядом викликів з точки зору безпеки, які вимагають впровадження комплексних заходів захисту. Нижче наведено детальний аналіз кожного аспекту сучасної веб-архітектури та його вплив на безпеку інформації, з урахуванням специфічних вимог до ІТ, ПЗ, ОС, ІКС та КСЗІ.

Розподіленість та масштабованість, сучасні веб-додатки часто базуються на мікросервісній архітектурі, де функціональність розподіляється між численними незалежними сервісами, що впливають на безпеку. Кожен з цих сервісів може використовувати власні бази даних, протоколи обміну даними та політики безпеки. Вплив на безпеку розподіленість створює більшу кількість точок взаємодії, кожна з яких може бути вразливою до атак. Необхідно впроваджувати централізоване управління безпекою через КСЗІ (Комплексну систему захисту інформації), щоб забезпечити узгодженість політик безпеки в усіх сервісах. Масштабованість системи повинні легко адаптуватися до змін навантаження шляхом горизонтального або вертикального масштабування. Вплив на безпеку під час масштабування зростає кількість серверів та контейнерів, що ускладнює моніторинг безпеки. Важливим є використання централізованих систем логування та моніторингу, а також впровадження політик ізолювання між різними сегментами мережі [31].

Використання API (Application Programming Interface) інтеграція сервісів, API дозволяють додаткам взаємодіяти між собою, забезпечуючи інтеграцію як всередині організації, так і з зовнішніми партнерами чи постачальниками послуг. Вплив на безпеку незахищені або неналежно валідувані API можуть стати точкою входу для атак, наприклад, шляхом підробки запитів або експлуатації вразливостей через несанкціонований доступ. Заходи захисту впровадження аутентифікації API-запитів, використання токенів доступу, шифрування даних під

час передачі за допомогою SSL/TLS, а також регулярне тестування API на проникнення допомагають знизити ризики [32].

Сервіс-орієнтована архітектур базується на розподілі функціональності між окремими сервісами, які можуть працювати незалежно один від одного, забезпечуючи більшу гнучкість і повторне використання компонентів. Вплив на безпеку взаємодія між сервісами вимагає забезпечення безпеки кожного каналу обміну даними. Якщо один сервіс стає вразливим, це може стати причиною компрометації всієї системи. Заходи захисту впровадження стандартних протоколів обміну, таких як RESTful API, а також використання криптографічних методів захисту (наприклад, симетричне шифрування алгоритмами AES, DES, IDEA, PES з використанням АГК та АКП) сприяє підвищенню рівня безпеки при взаємодії між сервісами [33].

Хмарні платформи дозволяють кільком користувачам або організаціям використовувати спільну інфраструктуру. Виклики недосконала ізоляція віртуальних машин або контейнерів може призвести до ситуації, коли дані однієї організації потенційно доступні іншій. Заходи захисту ретельне налаштування політик віртуалізації, регулярний аудит із застосуванням КСЗІ та використання технологій ізоляції, що відповідають вимогам ДСТУ, є критично важливими.

Дані, що передаються між клієнтом та хмарним сервером, повинні бути захищені за допомогою шифрування протоколами SSL/TLS. Виклики атаки типу «людина посередині» (MITM) можуть призвести до перехоплення даних, якщо шифрування не використовується або його реалізація має вразливості. Зберігання даних у хмарі хмарні платформи повинні використовувати методи шифрування даних на рівні баз даних та файлових систем, щоб забезпечити їх конфіденційність навіть у разі фізичного доступу до серверів. Заходи захисту використання сучасних алгоритмів шифрування (AES, DES, IDEA, PES) та застосування механізмів управління ключами, де ключі генеруються за допомогою АГК і трансформуються за допомогою АКП, забезпечує високий рівень захисту даних [34].

Багато хмарних сервісів вимагають чіткої системи управління доступом із застосуванням багатофакторної аутентифікації, цифрових сертифікатів та токенів доступу. Виклики помилки в конфігурації доступу можуть відкрити шлях для несанкціонованого доступу, що критично для підприємств малого бізнесу. Заходи захисту інтеграція систем управління доступом з КСЗІ, регулярне оновлення політик та аудит користувацької активності допомагають мінімізувати ці ризики.

Методи захисту web-додатків від типових атак, захист від SQL-ін'єкцій, SQL-ін'єкції виникають, коли зловмисник вводить шкідливі SQL-коди через веб-форми або URL-параметри, що дозволяє йому маніпулювати базою даних.

- Використання підготовлених (prepared) та параметризованих запитів для запобігання можливості вставлення шкідливого коду.
- Використання Object-Relational Mapping (ORM) фреймворків, які автоматично обробляють вхідні дані.
- Регулярне проведення аудиту коду та тестування на проникнення для виявлення потенційних вразливостей [35].

Захист від Cross-Site Scripting (XSS) При XSS-атаках зловмисники впроваджують шкідливі скрипти в контент веб-сторінок, які виконуються у браузерах користувачів. Методи захисту:

- Екранування (escaping) та валідація вхідних даних, що дозволяє уникнути виконання небажаного коду.
- Впровадження Content Security Policy (CSP), що обмежує виконання скриптів лише з довірених джерел [36].
- Регулярне оновлення бібліотек та фреймворків для усунення відомих вразливостей [37].

Захист від Cross-Site Request Forgery (CSRF) CSRF-атаки змушують автентифікованого користувача виконати небажану дію на сайті без його відома, використовуючи його сесію [38].

- Використання унікальних токенів, що додаються до кожної форми та перевіряються на сервері.

- Використання реферера та заголовків запиту для підтвердження походження запиту.

- Забезпечення багатофакторної аутентифікації, що ускладнює експлуатацію сесійних даних.

Використання сучасних фреймворків та технологій, такі як Angular, React, Django або Ruby on Rails, мають вбудовані механізми захисту від багатьох типів атак, що дозволяє розробникам зосередитись на бізнес-логіці.

Моніторинг та аудит інтеграція систем моніторингу, логування та аудиту (часто як частина КСЗІ) дозволяє відслідковувати підозрілу активність та оперативно реагувати на інциденти безпеки [39].

Інтеграція криптографічних засобів таких як шифрування даних для забезпечення конфіденційності даних, що передаються через веб-додатки, використовуються алгоритми симетричного шифрування (AES, DES, IDEA, PES), де ключі генеруються за допомогою АГК та трансформуються через АКП. Обмін ключами, асиметричне шифрування (РКС) застосовується для безпечного обміну ключами, що дозволяє встановити захищене з’єднання між клієнтом і сервером, мінімізуючи ризик перехоплення ключів. Додаткові методи використання цифрових підписів та ЦВЗ для захисту цілісності та автентичності документів, що є критичним для фінансових транзакцій та обробки конфіденційної інформації.

1.5 Постановка задачі проектування кваліфікаційної роботи

Розробка системи захисту фінансового документообігу вимагає чіткої постановки задач, які необхідно вирішити в межах цієї роботи. Сучасні програмні комплекси фінансового документообігу використовують web-архітектуру, що забезпечує віддалений доступ та централізоване управління документами, але водночас створює нові виклики у сфері кібербезпеки. Особливої уваги потребують питання конфіденційності даних, захисту від атак та впровадження механізмів криптографічного захисту.

Аналіз предметної області показав, що сучасні програмні комплекси для фінансового документообігу надають широкий спектр функціональності, включаючи хмарну інтеграцію, підтримку web-архітектури та мобільного доступу. Водночас, ці переваги супроводжуються низкою викликів у сфері безпеки — наявністю вразливостей до SQL-ін'єкцій, XSS, CSRF, проблемами з автентифікацією та шифруванням даних. Вивчення нормативно-правової бази та методів захисту показало важливість впровадження криптографічних механізмів, багатофакторної автентифікації, чіткого контролю доступу та систем моніторингу. Враховуючи обмеження малих підприємств, ключовим є створення практичних і економічно доцільних рішень із забезпечення базового рівня кіберзахисту.

Таким чином, основні задачі, які потрібно вирішити, включають:

- Провести комплексний аудит інформаційної безпеки програмного комплексу фінансового документообігу згідно з вимогами ДСТУ ISO/IEC 27001.
- Розробити формалізовану модель загроз відповідно до принципів CIA (Confidentiality, Integrity, Availability).
- Сформувати модель порушника з урахуванням потенціалу зовнішніх та внутрішніх зловмисників, а також автоматизованих атак.
- Реалізувати захист від brute-force та DDoS атак шляхом впровадження механізмів обмеження частоти автентифікаційних запитів та загальному контролю кількості запитів(rate limiting) [40-41].
- Спроєктувати та впровадити криптографічні засоби захисту, зокрема end-to-end шифрування на основі AES-256 та PKC [42].
- Реалізувати механізми розмежування доступу на основі моделі RBAC із журналюванням дій користувачів [43].
- Впровадити захист від атак типу SQL injection шляхом використання параметризованих запитів та ORM-рішень.
- Забезпечити протидію XSS-атакам через впровадження механізмів фільтрації та екранування вхідних даних, а також політик Content Security Policy (CSP).

2 СТВОРЕННЯ СИСТЕМИ ЗАХИСТУ ПРОГРАМНОГО КОМПЛЕКСУ ЗАХИЩЕНОГО ДОКУМЕНТООБІГУ, МОДЕЛІ ЗАГРОЗ ТА МОДЕЛІ ПОРУШНИКА

2.1 Порівняльний аналіз існуючого програмного комплексу

З метою забезпечення надійного функціонування інформаційної системи фінансового документообігу та зниження рівня потенційних ризиків було проведено комплексний аудит поточного стану програмного комплексу. Такий аудит є важливою складовою процесу розробки та впровадження системи захисту інформації, оскільки дозволяє виявити слабкі місця, порушення вимог безпеки, а також визначити напрями вдосконалення.

Аудит проводився на прикладі реального програмного забезпечення, розробленого за архітектурою Java Spring MVC, з використанням бази даних MySQL, сесійної авторизації, механізмів контролю доступу на основі ролей, а також функціоналу внутрішнього чату між користувачами. Особлива увага приділялася аналізу аспектів, що стосуються захисту даних, автентифікації, авторизації, зберігання та передачі інформації.

Система реалізована у вигляді класичного веб-додатку, де клієнтська частина представлена через HTML-інтерфейс, а серверна логіка реалізована на Java за допомогою фреймворку Spring MVC. Серверна частина додатку відповідає за обробку запитів користувачів, взаємодію з базою даних, керування сесіями, а також за виконання бізнес-логіки, пов’язаної з документообігом та внутрішнім листуванням [44].

Варто зазначити, що підключення до бази даних MySQL здійснюється локально, тобто без використання зовнішніх або віддалених з’єднань. Це певною мірою зменшує ризик перехоплення даних на мережевому рівні, однак не замінює повноцінного шифрування трафіку, яке може знадобитися у випадку розгортання системи в хмарному середовищі або в умовах розподіленої інфраструктури [45].

Система автентифікації реалізована на основі стандартних механізмів Spring Security з використанням сесійного зберігання стану користувача. Після проходження авторизації (введення логіна та пароля) користувач отримує унікальний ідентифікатор сесії, який дозволяє йому надалі взаємодіяти із системою без повторного введення облікових даних. Важливо, що паролі користувачів не зберігаються у відкритому вигляді — для цього використовується алгоритм хешування bcrypt, що забезпечує стійкість до атак перебору (brute-force) та аналізу злитих баз даних [46].

Для регулювання прав доступу до окремих частин функціоналу в системі реалізовано рольову модель контролю доступу (RBAC). Кожному користувачу призначається певна роль (наприклад, адміністратор, менеджер, звичайний користувач), і в залежності від ролі визначається перелік доступних дій. Наприклад, лише користувачі з роллю адміністратора можуть створювати або редагувати інші облікові записи, мати доступ до перегляду журналів подій або змінювати структуру документообігу.

Аудит підтвердив, що доступ до більшості маршрутів ендпоінтів у системі обмежений відповідно до призначених ролей. Разом з тим, деякі критично важливі частини додатку потребують додаткового контролю — наприклад, доступ до адміністративних сторінок або налаштувань безпеки.

Для фіксації дій користувачів і системних подій використовується файлове логування. Усі події зберігаються у лог-файлах на сервері, і можуть містити такі записи, як спроби входу в систему, помилки авторизації, звернення до заборонених маршрутів, обробка критичних помилок тощо. Це дозволяє адміністраторам системи в подальшому здійснювати аудит активності, виявляти підозрілу поведінку користувачів, а також проводити розслідування у випадку інцидентів.

Однак, варто зазначити, що логи зберігаються поза базою даних, тобто немає централізованого інтерфейсу для їх перегляду, аналізу або пошуку за критеріями. Це обмежує можливості моніторингу у реальному часі і потребує ручного втручання при аналізі подій.

У ході проведеного аудиту системи були виявлені ряд вразливостей, які можуть бути використані зловмисниками для отримання несанкціонованого доступу, компрометації даних або порушення роботи системи. Нижче наведено опис основних з них.

Однією з ключових функцій системи є чат, який дозволяє користувачам обмінюватися повідомленнями у реальному часі. У межах цього чату можуть обговорюватися як загальні організаційні питання, так і чутлива інформація, зокрема дані про документи, платежі, фінансові умови тощо. У процесі аудиту було виявлено, що передача повідомлень від клієнта до сервера, а також їх збереження у базі даних, відбувається у відкритому вигляді, без будь-якої форми криптографічного захисту.

Це означає, що у разі компрометації сервера або витоку дампу бази даних зловмисник отримає повний доступ до історії переписки всіх користувачів системи. Ба більше, у випадку використання незашифрованого з'єднання (наприклад, HTTP замість HTTPS) існує ризик перехоплення повідомлень у мережі через атаку типу MITM (Man-in-the-Middle). Особливо це критично, якщо користувачі працюють з додатком у загальнодоступних або ненадійних мережах (громадський Wi-Fi, VPN-провайдери тощо).

Враховуючи важливість конфіденційності у фінансових системах, відсутність шифрування чату є суттєвим порушенням принципів інформаційної безпеки — зокрема принципу захисту даних у транзиті та збереженні (data in transit, data at rest). Впровадження енд-ту-енд шифрування є критично необхідним для уникнення ризиків.

У ряді компонентів програмного комплексу були виявлені SQL-запити, які будуються за допомогою ручного формування рядків, в які підставляються значення, введені користувачем. Такий підхід є дуже вразливим до SQL-ін’єкцій — одного з найпоширеніших і найнебезпечніших типів атак на бази даних.

Зокрема, при використанні прямої конкатенації введеного значення у SQL-запит, зловмисник може ввести спеціально сформовану команду, яка змінить структуру запиту.

SQL-ін’єкції можуть призвести не лише до несанкціонованого читання даних, але й до зміни або видалення записів у базі, виконання адміністративних команд, порушення цілісності інформації. За найгіршого сценарію — повна компрометація бази даних і втрати бізнес-критичної інформації.

Веб-форми, які приймають вхідні дані від користувачів, не мають встановлених обмежень на об’єм переданого тексту. Наприклад, у полі «повідомлення чату» або «назва документа» можна ввести довільну кількість символів, аж до мегабайтів тексту, що створює передумови для відмови в обслуговуванні (DoS) на рівні серверного застосунку або бази даних.

Зловмисник може сконструювати запит, який міститиме надмірно великий об’єм тексту (наприклад, 5 МБ у полі "message"), що призведе до перевантаження серверної пам’яті, переповнення логів, зниження продуктивності або навіть аварійного завершення роботи сервісу. Подібні атаки легко реалізуються автоматизованими скриптами, тому їх часто використовують боти для знищення або блокування онлайн-сервісів.

Крім того, відсутність обмежень на вхідні дані створює ризик вставлення небажаних HTML- або JavaScript-елементів у великій кількості, що у поєднанні з XSS може мати катастрофічні наслідки.

Система дозволяє користувачам вводити довільний текст у полях повідомлень або назв документації, але при цьому не проводить фільтрацію HTML-контенту. Це створює умови для міжсайтових скриптових атак (Cross-Site Scripting, XSS), коли вміст, введений одним користувачем, відображається на стороні іншого без обробки або екранізації.

У результаті XSS-атаки зловмисник може вставити у повідомлення шкідливий скрипт (наприклад, <script>alert("XSS")</script>), який буде виконаний у браузері іншого користувача. Через XSS можна викрасти сесійні cookie, змінити DOM-структуру сторінки, здійснити фішингову атаку або навіть перехопити дії користувача на сайті.

Особливу небезпеку становлять персистентні (збережені) XSS, коли шкідливий код зберігається в базі даних і виконується при кожному зверненні. У

системах документообігу, де є велика кількість користувачів і постійний обмін інформацією, такий вектор атаки може поширитися миттєво.

Використання сесій для збереження автентифікаційного стану користувача є стандартною практикою у веб-додатках. Однак відсутність таймауту для сесії створює додаткові ризики, особливо у випадках, коли користувачі використовують загальні пристрої (наприклад, офісні комп'ютери, комп'ютери у коворкінгу, бібліотеці тощо).

Якщо користувач не виходить із системи вручну, його сесія залишатиметься активною протягом невизначеного часу. Це дає змогу будь-кому, хто отримає фізичний або мережевий доступ до пристрою, продовжити роботу від імені цього користувача без повторної автентифікації. Подібний вектор атаки відомий як session hijacking — викрадення або несанкціоноване використання сесії.

Також відсутність обмеження часу сесії негативно впливає на продуктивність сервера, оскільки всі сесії залишаються в активному стані і споживають ресурси. Це створює умови для поступового виснаження пулу сесій та зриву нормального функціонування системи.

2.2 Створення моделі загроз інформаційної безпеки

Проєктування ефективної системи захисту інформаційного ресурсу неможливе без чіткого й системного розуміння потенційних загроз, які можуть бути реалізовані щодо цього ресурсу. На практиці це означає, що перш ніж впроваджувати будь-які технічні або організаційні механізми захисту, необхідно визначити, хто або що може спричинити загрозу, у який спосіб вона може бути реалізована, і яких наслідків слід очікувати в разі її реалізації. Саме тому формування моделі загроз є базовим етапом у проєктуванні будь-якої системи інформаційної безпеки. Вона дозволяє сформулювати стратегію захисту з урахуванням найреальніших сценаріїв атак або збоїв, які відповідають архітектурі та функціоналу конкретної системи.

Модель загроз являє собою узагальнений опис можливих ризиків, які можуть виникнути як внаслідок цілеспрямованих дій зловмисників, так і в результаті випадкових помилок — людських або технічних. Такі ризики можуть проявлятися через недосконалості архітектури, погано продуману логіку доступу, недостатньо захищені точки входу в систему, або інші вразливості в програмному забезпеченні [47-48].

У межах цієї кваліфікаційної роботи модель загроз формується з урахуванням реального функціоналу створеного програмного комплексу — системи фінансового документообігу з web-архітектурою. При цьому враховуються такі аспекти, як обробка персональних та фінансових даних, наявність вбудованого чату для внутрішньої комунікації між користувачами, механізми автентифікації та авторизації, а також рівень розмежування доступу на основі ролей. Додатково в модель інтегровано результати проведеного аудиту, описаного в попередньому розділі, що дозволяє зробити її максимально практичною та актуальною.

Побудова моделі загроз здійснюється за класичним підходом до аналізу ризиків, який складається з п’яти основних етапів. Першим кроком є ідентифікація активів — це означає визначення всіх елементів системи, які мають цінність і підлягають захисту (наприклад, персональні дані, документи, повідомлення, логіка транзакцій, облікові записи користувачів тощо). Далі відбувається оцінка загроз, тобто опис дій або умов, які можуть порушити нормальне функціонування системи або скомпрометувати її дані.

Наступним етапом є класифікація загроз за категоріями — конфіденційність, цілісність та доступність. Це дозволяє структурувати ризики та вибудувати ієрархію пріоритетів. Після цього виконується моделювання векторів атак, під час якого визначається, яким саме чином конкретна загроза може бути реалізована в рамках даної системи. Завершальним кроком є пріоритезація ризиків — тобто визначення найкритичніших загроз, які вимагають першочергового усунення.

В основі класифікації використовується тріада CIA — три ключові властивості, які має зберігати будь-яка інформація: Confidentiality (конфіденційність), Integrity (цілісність), Availability (доступність). Крім того, було частково використано підходи STRIDE-моделі, яка додає такі категорії, як підробка (Spoofing), спотворення даних (Tampering), заперечення дій (Repudiation), розкриття інформації (Information Disclosure), відмова в обслуговуванні (Denial of Service), і підвищення привілеїв (Elevation of Privilege). У цій роботі STRIDE використовується в спрощеному вигляді, з акцентом на найбільш критичні аспекти безпеки для web-систем.

Конфіденційність — це властивість інформації бути доступною лише тим користувачам, які мають на це відповідні права. У програмному комплексі документообігу конфіденційність охоплює такі елементи, як особисті дані користувачів (імена, email, IP-адреси), вміст фінансових документів (договірні умови, суми, строки), повідомлення у внутрішньому чаті, а також лог-файли активності системи.

До основних загроз конфіденційності належать: Перехоплення даних у транзиті, несанкціонований доступ, витік сесій, неправильна авторизація.

Перехоплення даних у транзиті є однією з найпоширеніших загроз для web-додатків. Якщо система використовує незашифровані протоколи передачі даних, такі як HTTP замість HTTPS, інформація, що передається між клієнтом і сервером, може бути перехоплена зловмисниками. Такий тип атаки особливо небезпечний у публічних або відкритих мережах, де зловмисник може здійснити атаку типу "людина посередині" (Man-in-the-Middle). У результаті можуть бути перехоплені логіни, паролі, повідомлення з чату, а також фінансові документи.

Несанкціонований доступ до бази даних виникає тоді, коли зловмисник отримує можливість напряму звертатися до бази, минаючи механізми авторизації. Це може відбутися як через вразливості в програмному коді (наприклад, SQL-ін’єкції), так і через недоліки в конфігурації сервера. Особливо високий ризик спостерігається, коли чутлива інформація — наприклад, повідомлення

користувачів або фінансові дані — зберігається у відкритому (незашифрованому) вигляді.

Витік сесій також є поширеною загрозою для конфіденційності. Якщо веб-додаток не захищає сесійні токени належним чином (наприклад, не використовує прапорці HttpOnly або Secure для cookie), або якщо вразливості типу XSS дозволяють вставити шкідливий скрипт у сторінку, зловмисник може викрасти ідентифікатор сесії іншого користувача. У такому випадку він отримає повний доступ до облікового запису жертви без необхідності введення логіна і пароля.

Неправильна авторизація має місце тоді, коли система недостатньо чітко перевіряє, чи має користувач право доступу до певного ресурсу. Наприклад, якщо користувач, що не має адміністративних повноважень, може переглядати або редагувати чужі документи через URL або форму, це є прямим порушенням принципу конфіденційності. Подібна помилка може виникати через відсутність перевірок на рівні контролера або через занадто спрощену логіку доступу.

Цілісність інформації означає її захищеність від несанкціонованих або випадкових змін. У системі документообігу цілісність означає, що фінансові документи не повинні змінюватися без дозволу, повідомлення в чаті мають зберігати оригінальний зміст, а структура бази даних — залишатися стабільною.

Типові загрози цілісності включають: SQL-ін’єкції, XSS-атаки, зловживання правами, неочищені вхідні дані.

SQL-ін’єкції — це тип атаки, при якій зловмисник змінює структуру SQL-запиту шляхом введення спеціально сформованого коду у форму або параметр запиту. При недостатньому захисті введених даних запит може бути модифікований таким чином, що дозволить змінити або видалити записи в базі даних, обійти авторизацію або навіть отримати повний контроль над таблицями. У системі документообігу це може призвести до підміни або знищення важливих документів.

XSS-атаки (міжсайтове виконання скриптів) виникають тоді, коли введений користувачем HTML або JavaScript-код не фільтрується і відображається іншим користувачам. Це дозволяє зловмиснику змінювати вигляд сторінки, вводити

фальшиві елементи (наприклад, підроблені поля входу) або отримувати дані з форми до її надсилання. Крім загроз конфіденційності, XSS може також впливати на цілісність візуального представлення або даних, які бачить користувач.

Зловживання правами передбачає ситуацію, коли користувач використовує більше повноважень, ніж йому належить. Наприклад, співробітник із роллю «менеджер» має доступ до функцій, які були призначені лише для адміністратора, — таких як видалення користувачів або зміна системних налаштувань. Це може бути наслідком помилок у реалізації контролю доступу або занадто широких дозволів у моделі ролей.

Неочищені вхідні дані створюють ризик накопичення некоректної, шкідливої або випадкової інформації у системі. Наприклад, якщо користувач вставить у поле назви документа великий обсяг тексту або фрагменти HTML-коду, ці дані можуть зіпсувати вигляд сторінок або порушити структуру таблиць у базі даних. Такі вхідні дані не лише ускладнюють підтримку системи, а й можуть стати основою для подальших атак.

Доступність означає, що система має бути доступною та функціональною для легітимних користувачів у будь-який момент часу. У нашому випадку це означає, що чат має працювати без зависань, документи повинні створюватися й відкриватися без затримок, а вхід у систему не повинен бути заблокованим через технічні помилки.

До найтиповіших загроз доступності належать: DoS-атаки, акумуляція сесій, масовий спам, критичні винятки.

DoS-атаки через великі форми реалізуються шляхом надсилання до сервера запитів, які містять надмірні обсяги інформації. Наприклад, користувач вводить у текстове поле кілька мегабайт символів, після чого система намагається обробити цей запит, витрачаючи значну кількість ресурсів. Якщо таких запитів багато, сервер може зависнути або стати недоступним для інших користувачів.

Акумуляція сесій є проблемою у тих системах, де не встановлено обмеження часу життя сесії або не очищаються старі неактивні сесії. Згодом пам’ять сервера починає заповнюватися даними про велику кількість одночасно

активних користувачів (навіть якщо більшість із них уже неактивні). Це призводить до деградації продуктивності або до збоїв у роботі.

Масовий спам є актуальною загрозою для будь-якої інтерактивної системи. Якщо система дозволяє створювати документи, повідомлення чи інші об’єкти без обмежень і перевірок, зловмисник може автоматично створити сотні або тисячі елементів, заповнюючи базу даних непотрібною інформацією. Це не лише перевантажує сервер, а й ускладнює роботу реальним користувачам.

Критичні винятки (exceptions) виникають тоді, коли система не передбачає обробку помилок у певних ситуаціях. Наприклад, якщо у форму вводиться нестандартний символ або надто велике число, а сервер не обробляє такі випадки коректно — додаток може зупинитися або повернути технічну помилку (500 Internal Server Error). Такі ситуації порушують доступність системи для кінцевого користувача та негативно впливають на довіру до неї.

Під час побудови моделі загроз важливо розрізняти зовнішні та внутрішні загрози, оскільки вони мають різне походження, мотивацію та методи дій. Це допомагає точніше оцінити ризики й сформувати ефективні заходи захисту.

Зовнішні загрози походять від сторонніх осіб чи систем, які не мають доступу до внутрішнього середовища, але намагаються його отримати. Це можуть бути хакери, шкідливі програми, автоматизовані сканери чи конкуренти. Їхні атаки часто спрямовані на відкриті частини системи: логін-форми, API, порти тощо.

Внутрішні загрози виникають від користувачів із доступом до системи — співробітників або підрядників. Вони можуть бути як навмисними (крадіжка даних, саботаж), так і випадковими (помилки, недотримання політик безпеки). Загрозу становлять також облікові записи колишніх працівників, якщо їх не було вчасно видалено.

Часто внутрішні загрози є більш небезпечними, оскільки дії таких користувачів важко виявити й зупинити. Вони можуть діяти тривалий час, завдаючи шкоди без очевидних ознак. Для ефективного захисту модель загроз

повинна враховувати обидві категорії та мати окремі механізми реагування для кожної з них (табл. 2.1).

Таблиця 2.1 – Модель загроз

Назва загрози	Категорія (CIA)	Ймовірність реалізації	Потенційні наслідки	Пріоритет
1	2	3	4	5
Перехоплення даних у транзиті	Конфіденційність	Середня	Витік особистих даних, повідомлень чату, облікової інформації	Високий
Несанкціонований доступ до БД	Конфіденційність	Середня	Доступ до всіх документів та чатів у відкритому вигляді	Високий
Витік сесій	Конфіденційність	Висока	Отримання повного доступу до чужого облікового запису	Високий
Неправильна авторизація	Конфіденційність	Середня	Отримання доступу до чужих документів або дій	Високий
SQL-ін’єкції	Цілісність	Середня	Видалення або зміна важливих даних, обхід автентифікації	Високий
XSS-атаки	Конфіденційність / Цілісність	Висока	Викрадення сесій, спотворення вигляду сторінок	Високий
Зловживання правами	Цілісність	Середня	Несанкціоноване редагування, видалення або конфігурація системи	Середній
Неочищені вхідні дані	Цілісність	Середня	Збереження шкідливих або некоректних даних	Середній

Кінець таблиці 2.1

1	2	3	4	5
DoS через великі форми	Доступність	Середня	Перевантаження сервера, відмова в обслуговуванні	Високий
Акумуляція сесій	Доступність	Висока	Вичерпання пам’яті сервера, зниження продуктивності	Середній
Масовий спам	Доступність	Середня	Засмічення бази, погіршення зручності користування	Середній
Критичні винятки (Exceptions)	Доступність	Середня	Аварійне завершення запитів, помилки на інтерфейсі	Середній

Модель загроз є ключовим етапом у проєктуванні системи захисту інформаційного ресурсу. Вона дозволяє виявити найімовірніші та найнебезпечніші ризики для конфіденційності, цілісності та доступності даних. У роботі враховано як зовнішні, так і внутрішні загрози, що дозволяє сформувати цілісну стратегію безпеки для фінансової web-системи документообігу.

2.3 Створення моделі порушника інформаційної безпеки

Після побудови моделі загроз логічним наступним кроком є формування моделі порушника інформаційної безпеки, яка описує потенційного суб’єкта, здатного реалізувати одну або кілька загроз до об’єктів інформаційної системи. Модель порушника дозволяє більш точно оцінити реальні ризики, розуміючи не лише те, що може бути атаковане, а й ким, як, і з якими технічними або організаційними можливостями [49-50].

Під «порушником» у цьому контексті розуміється як реальна особа (наприклад, співробітник, зловмисник, конкурент), так і штучний суб’єкт

(програма, бот, скрипт), дії якого спрямовані на порушення конфіденційності, цілісності або доступності інформаційних ресурсів. Створення моделі порушника дає змогу більш глибоко сформувати профіль потенційного атакувальника та передбачити його можливі дії, засоби, рівень знань і мотивацію.

Побудова моделі порушника для даної системи базується на аналізі архітектури програмного комплексу, його слабких місць (виявлених під час аудиту), способів доступу до ресурсів, а також реального функціоналу — системи фінансового документообігу з web-інтерфейсом, збереженням персональних даних і внутрішньою комунікацією.

У моделі прийнято розрізняти дві основні категорії класифікації порушників: зовнішні та внутрішні. Обидві категорії мають свою специфіку дій, відмінний набір інструментів та різний рівень потенційної шкоди. Розгляд обох груп є важливим для побудови повноцінної захисної системи, оскільки атаки можуть бути здійснені як ззовні, так і зсередини організації.

Зовнішній порушник — це суб’єкт, який не має законного доступу до системи, але намагається його отримати або зашкодити системі іншими способами. Зазвичай зовнішні порушники діють із віддалених локацій, використовуючи Інтернет або вразливі публічні точки доступу. Вони можуть сканувати мережу на предмет відкритих портів, тестувати форми входу на вразливості, шукати відомі недоліки у бібліотеках, які використовуються застосунком, або просто здійснювати атаки наосліп (наприклад, через ботів).

Типовими прикладами зовнішніх порушників є:

- хакери з навичками тестування на проникнення;
- сканери безпеки та автоматизовані боти;
- конкуренти, що намагаються дестабілізувати роботу системи;
- звичайні користувачі, які випадково або навмисно вводять некоректні дані у спробі обійти правила.

Внутрішній порушник, навпаки, є користувачем системи, який має легітимний доступ, але з певних причин зловживає цим доступом або здійснює дії, які виходять за межі його дозволених повноважень. Ця категорія є особливо

небезпечною, оскільки такі користувачі вже знаходяться «всередині» захищеного периметра і мають змогу виконувати операції на рівні бізнес-логіки системи.

Прикладами внутрішніх порушників є:

- співробітник, який навмисно завантажує копії фінансових документів перед звільненням;
- користувач із розширеними правами, який тестує межі дозволеного;
- адміністратор, що неправильно налаштував права доступу або залишив сесію відкритою;
- звичайний працівник, який необережно натиснув на фішингове посилання або залишив комп'ютер без нагляду.

У внутрішнього порушника зазвичай є вища ймовірність реалізації загрози, адже він вже має доступ до системи, розуміє її логіку, а його дії важче відслідкувати.

Окрему увагу при моделюванні порушника необхідно приділити його мотивації. Розуміння того, що саме спонукає особу або групу осіб до дій, спрямованих на порушення інформаційної безпеки, дозволяє краще передбачити вектори атак, рівень підготовки порушника, а також визначити об'єкти системи, які є для нього найбільш привабливими. Мотивація порушника безпосередньо впливає на характер, глибину і послідовність його дій, а отже — і на заходи, які необхідно реалізувати для протидії цим загрозам.

Фінансова вигода є, без сумніву, найпоширенішою мотивацією у сфері фінансових інформаційних систем. Порушник, керуючись особистими корисливими інтересами, може намагатися отримати доступ до платіжної інформації, облікових записів клієнтів, документів, що містять конфіденційні умови угод, або інші дані, які мають цінність на чорному ринку. Така інформація може бути використана для продажу, шантажу, підробки платіжних документів або несанкціонованих транзакцій. У більшості випадків атаки, спрямовані на фінансову вигоду, добре сплановані, включають етапи підготовки, збору інформації та експлуатації вразливостей.

Зловмисна конкуренція — ще одна поширена мотивація, особливо у випадках, коли на ринку присутні кілька компаній, які надають схожі послуги. У такому випадку атакувальник може діяти в інтересах третьої сторони, яка має на меті дискредитацію, дестабілізацію або підрив репутації компанії-конкурента. Серед можливих дій — навмисне створення навантаження на систему, поширення компрометуючих матеріалів, блокування доступу до сервісів або викрадення стратегічної документації.

Ідеологічні або політичні мотиви є менш поширеними в комерційному секторі, однак не менш небезпечними. У таких випадках атакувальники керуються не матеріальними вигодами, а переконаннями, прагненням самовираження або протесту. Такі дії можуть включати в себе деформацію інтерфейсу системи, публікацію небажаного контенту, виведення з ладу сервісу або поширення неправдивої інформації. Атаки за ідеологічними мотивами часто носять публічний характер, супроводжуються заявами в ЗМІ або соціальних мережах.

Технічний інтерес або випадковість — це категорія, яка характерна для внутрішніх користувачів, зокрема співробітників компанії, які мають доступ до системи, але не завжди усвідомлюють межі дозволеного. Часто такі користувачі з цікавості тестують функціонал, вводять нестандартні значення у поля форм, намагаються змінити параметри URL або використовують сторонні інструменти (наприклад, проксі чи розширення браузера) для аналізу роботи системи. Подібні дії можуть спричинити збої або виявити вразливості, які в подальшому можуть бути використані вже зі шкідливими намірами.

Ще одним важливим фактором, який безпосередньо впливає на оцінку ризиків, є рівень технічної підготовки порушника. Залежно від обізнаності, досвіду та наявності спеціальних інструментів, порушники мають різну здатність до виявлення, використання та комбінування вразливостей.

Порушник із низьким рівнем підготовки — це, як правило, користувач без спеціальної технічної освіти або досвіду роботи в ІТ-сфері. Його дії найчастіше базуються на інтуїції, спробах і помилках, або ж виконуються за інструкціями з

відкритих джерел (форумів, відео, блогів). Такий користувач може спробувати вставити JavaScript у форму (XSS), ввести спеціальні символи в поле логіну, маніпулювати адресним рядком браузера. Незважаючи на простоту, такі дії можуть виявитися ефективними в системах з низьким рівнем захисту.

Порушник із середнім рівнем підготовки — це особа, яка має базові знання з розробки або тестування web-додатків. Вона вже розуміє, як працює авторизація, взаємодія з базами даних, має навички роботи з HTTP-запитами, проксі-серверами (наприклад, Burp Suite), може здійснювати SQL-ін’єкції, змінювати cookie, перехоплювати трафік. Такий порушник здатен виконати складніші атаки, знайти приховані функції або комбінувати кілька векторів у межах однієї сесії.

орушник із високим рівнем підготовки — це професіонал у сфері кібербезпеки: етичний хакер, сертифікований пентестер або досвідчений зловмисник, який має глибокі технічні знання та значний практичний досвід. Такий фахівець досконало розуміється на принципах роботи інформаційних систем, знає типові й нетипові вразливості, уміє аналізувати архітектуру програмного забезпечення, а також має навички зворотної інженерії.

У його арсеналі — широкий спектр інструментів: як публічні утиліти типу Burp Suite, Metasploit, Wireshark, так і власноруч створені скрипти та експлойти. Він також добре обізнаний у криптографії, способах обходу автентифікації, техніках роботи з API та протоколами прикладного рівня. Порушник здатен реалізувати багаторівневу, складну атаку, яка може включати підробку токенів доступу, втручання в мережевий трафік, експлуатацію логіки вебзастосунків або обхід захисту на клієнтській стороні.

Його дії зазвичай ретельно сплановані, малопомітні для систем виявлення та максимально адаптовані під конкретну інфраструктуру або об’єкт атаки. Це робить загрозу з боку такого порушника однією з найнебезпечніших.

Урахування рівня підготовки порушника дозволяє проектувати захист не лише від очевидних загроз, а й від потенційно складних, прихованих або комбінованих атак. Це, в свою чергу, підвищує загальну стійкість системи до впливу як випадкових помилок, так і цілеспрямованих шкідливих дій.

Таблиця 2.2 – Модель порушника

Тип порушника	Джерело	Мотивація	Рівень технічної підготовки	Потенційні дії	Рівень загрози
Випадковий зовнішній користувач	Зовнішній	Цікавість, помилка	Низький	Введення небезпечних даних, базовий XSS	Низький
Зовнішній зловмисник	Зовнішній	Фінансова вигода	Середній / високий	SQL-ін’єкції, DoS, викрадення даних через сканування	Високий
Конкурент	Зовнішній	Бізнес-шкідництво	Середній	DDoS, компрометація репутації	Високий
Етичний хакер/дослідник	Зовнішній	Інтерес, самоствердження	Високий	Виявлення вразливостей, іноді публікація без згоди	Середній
Внутрішній користувач (без злих намірів)	Внутрішній	Випадковість, цікавість	Низький / середній	Тестування системи, залишення сесії відкритою	Середній
Внутрішній зловмисник	Внутрішній	Фінансова вигода, помста	Середній / високий	Викрадення документів, підміна даних, змінення прав доступу	Високий
Адміністратор з доступом до критичних ресурсів	Внутрішній	Нехтування політиками безпеки, помилка	Середній	Випадкове видалення, зміна логіки, залишення «дір»	Високий

Моделювання порушника є критично важливим етапом у забезпеченні інформаційної безпеки системи, оскільки воно дозволяє не лише ідентифікувати потенційні джерела загроз, а й оцінити їхню реалістичність, мотивацію та технічні можливості. Урахування як зовнішніх, так і внутрішніх порушників із різним рівнем підготовки та різними мотивами дає змогу сформувати цілісну картину ризиків і розробити ефективні заходи протидії. Такий підхід сприяє підвищенню загальної стійкості системи до атак і мінімізації ймовірності компрометації її ключових компонентів.

2.4 Проєктування системи захисту інформації

Сучасні інформаційні системи, особливо ті, що обробляють фінансові дані, вимагають особливої уваги до побудови системи інформаційної безпеки. В умовах стрімкого зростання кіберзагроз, збільшення кількості атак на бізнес-інфраструктури та зростання складності програмних рішень, питання інформаційного захисту перестає бути факультативним аспектом і перетворюється на один із центральних елементів усього життєвого циклу програмного забезпечення. Саме тому проєктування системи захисту інформації в межах цієї роботи є не лише технічною задачею, а й стратегічним кроком у забезпеченні надійності, відповідності сучасним вимогам і довіри користувачів до застосунку.

Проєктування системи захисту — це багаторівневий і міждисциплінарний процес, що поєднує знання в галузі розробки програмного забезпечення, криптографії, мережевої безпеки, управління ризиками, аналізу загроз та нормативно-правової відповідності. Така система має охоплювати весь обсяг можливих векторів атак, враховуючи як зовнішні, так і внутрішні загрози, обробку та зберігання конфіденційної інформації, а також сценарії взаємодії між користувачами.

У межах даної кваліфікаційної роботи проєктування системи захисту здійснюється на основі попередньо сформованих моделі загроз і моделі порушника, які дали змогу структуровано описати всі потенційно небезпечні дії щодо програмного комплексу фінансового документообігу. Це дозволило перейти від абстрактного аналізу до формування чітких, технічно обґрунтованих вимог, які надалі ляжуть в основу конкретної реалізації захисту.

При побудові системи інформаційного захисту критично важливо дотримуватися базових принципів безпеки, які закладено в міжнародних стандартах (наприклад, ISO/IEC 27001, OWASP ASVS) та які довели свою ефективність у практичних реалізаціях.

Принцип мінімальних привілеїв (Least Privilege) передбачає, що кожному суб'єкту в системі (користувачу, сервісу, скрипту) повинні бути надані лише ті права доступу, які є необхідними для виконання його безпосередніх функцій. Заборонено надавати зайві повноваження «на всякий випадок», оскільки кожне додаткове право — це потенційна вразливість. Наприклад, користувач, який лише переглядає документи, не повинен мати можливості їх редагувати або видаляти.

Принцип розмежування доступу логічно відокремлених ролей і груп доступу до різних частин системи. У нашому застосунку це реалізується через модель RBAC, де кожна роль (наприклад, «адміністратор», «менеджер», «користувач») має строго визначені дозволи.

Принцип захисту по глибині (Defense in Depth) система захисту не повинна спиратися на єдиний механізм безпеки. Натомість має бути реалізовано кілька незалежних рівнів захисту, які дублюють та доповнюють один одного. Це означає, що навіть у разі обходу одного бар'єра (наприклад, авторизації) порушник не отримає безперешкодного доступу до цільових ресурсів.

Усі критичні дії мають за замовчуванням бути забороненими це принцип відмовостійкості (Fail-safe defaults), якщо явно не зазначено протилежне. Наприклад, у разі втрати з'єднання або помилки в авторизації система має не надавати жодного доступу, а не переходити в «відкритий режим».

Принцип відкритості (Security through transparency, not obscurity) ґрунтується не на приховуванні механізмів, а на їх надійності. Це означає, що навіть у разі знання структури системи, протоколів або логіки порушник не повинен мати змоги обійти захист.

На підставі проведеного аналізу загроз і моделювання можливих сценаріїв поведінки порушника, у межах цієї кваліфікаційної роботи було запропоновано низку технічних та організаційних заходів, які формують основу системи захисту інформації. Вони охоплюють різні аспекти архітектури — від каналу передачі даних до логіки авторизації та валідації введення. Впровадження цих рішень спрямоване на посилення стійкості до типових атак, підвищення надійності програмного комплексу та забезпечення відповідності сучасним практикам інформаційної безпеки.

Одним із найважливіших напрямків захисту є забезпечення конфіденційності внутрішньої комунікації користувачів, зокрема — шифрування повідомлень у чаті(End-to-End Encryption). Традиційна передача повідомлень, навіть через HTTPS, не захищає дані від доступу з боку сервера, що зберігає повідомлення в незашифрованому вигляді. Щоб усунути цей недолік, передбачається використання механізму end-to-end шифрування, при якому текст повідомлення шифрується на клієнтській стороні (у браузері) та розшифровується лише на стороні одержувача. У якості технічного рішення пропонується використовувати симетричний алгоритм AES-256, а для розповсюдження ключів між користувачами — механізм асиметричного шифрування (RSA або алгоритми на еліптичних кривих, наприклад ECDH). Такий підхід гарантує, що навіть у разі компрометації сервера або бази даних, повідомлення залишаться недоступними для сторонніх осіб.

Ще одним критичним аспектом є забезпечення цілісності даних, зокрема захист бази даних від маніпуляцій за допомогою SQL-ін’єкцій. Вразливість цього типу виникає тоді, коли введені користувачем дані без фільтрації підставляються безпосередньо в SQL-запит, що дозволяє зловмиснику змінити логіку запиту. Щоб уникнути подібних атак, усі запити до бази даних мають будуватися на основі

підготовлених запитів (prepared statements) або ORM-інструментів, які автоматично екранізують небезпечні символи. У контексті Spring Boot це забезпечується через використання JPA/Hibernate, де всі запити проходять через перевірені інтерфейси взаємодії. Це дозволяє зберігати не лише цілісність бази, а й структуру системи контролю доступу, автентифікації та логування.

XSS (міжсайтове виконання скриптів) — одна з найпоширеніших загроз для web-застосунків, яка виникає, коли введені користувачем скрипти не фільтруються і відображаються у браузері інших користувачів. Для запобігання подібним атакам у системі впроваджується санітаризація вводу — процес очищення тексту від небезпечних HTML-елементів і JavaScript-інструкцій. Усі поля, що виводяться на сторінку (наприклад, ім’я користувача, повідомлення, назви документів), проходять через спеціальні фільтри, які дозволяють лише безпечний набір символів. Також у деяких випадках використовуються HTML-енкодери, що замінюють небезпечні символи на їх безпечні еквіваленти (наприклад, < на <). Це зменшує ризик виконання несанкціонованого коду в браузері.

Оскільки ідентифікація користувача в системі здійснюється через сесію, важливо забезпечити її конфіденційність і захист від викрадення. Для цього використовується встановлення захисних прапорців для cookie: HttpOnly (забороняє доступ до cookie з JavaScript), Secure (дозволяє передавати cookie тільки через HTTPS) та SameSite=Strict (блокує передачу cookie при запитах з інших доменів). Також впроваджено обмеження часу активності сесії — після 15 хвилин неактивності користувач буде автоматично розлогінений. Це дозволяє зменшити ймовірність викрадення сесій у публічних мережах або на спільних пристроях.

Для всіх форм у системі реалізовано подвійну валідацію введення — як на клієнтському рівні (за допомогою JavaScript), так і на серверному (через Spring Validation). Це дозволяє запобігти введенню надмірно довгих або некоректних значень, які можуть викликати помилки або бути використані у якості векторів атак. Наприклад, повідомлення чату не повинно перевищувати 5000 символів,

поле "назва документа" — 255 символів, поле "email" — має відповідати формату RFC 5322. Подібні обмеження також дозволяють зменшити ризик DoS-атак шляхом перевантаження форми.

З метою забезпечення ізоляції доступу до функціоналу системи використовується рольова модель доступу RBAC. Усі користувачі системи поділяються на ролі (наприклад: користувач, менеджер, адміністратор), кожна з яких має свій набір дозволів. Ці дозволи задаються централізовано та не можуть бути змінені на клієнтській стороні. Авторизація здійснюється на сервері — навіть якщо користувач вручну звернеться до URL, який йому недоступний, система поверне повідомлення про заборону. Це дозволяє запобігти як випадковому, так і навмисному доступу до ресурсів, що виходять за межі повноважень користувача.

Кожна значуща подія в системі має бути задокументована. Це стосується як дій користувачів (вхід у систему, зміна документа, створення повідомлення), так і системних подій (виключення, помилки доступу, запити до системи). Усі події логуються в окремі файли логів, доступ до яких мають лише адміністратори системи. Це дозволяє здійснювати аудит безпеки, відновлювати хронологію інцидентів, а також виявляти аномальну поведінку користувачів.

Усі з’єднання між клієнтом і сервером повинні здійснюватися виключно через захищений протокол HTTPS. Для цього використовується сертифікат SSL/TLS, що шифрує весь трафік. У системі примусово перенаправляються всі HTTP-запити на HTTPS, щоб виключити можливість перехоплення даних у незашифрованому вигляді. Бажаним є використання протоколу TLS 1.3, який забезпечує більш сучасний рівень захисту та меншу вразливість до атак типу downgrade або replay.

Для забезпечення безпеки у межах архітектури Spring MVC важливо реалізувати обмеження частоти запитів до окремих маршрутів (наприклад, форм авторизації, реєстрації, надсилання повідомлень), щоб запобігти атакам типу brute-force, спам, автоматичне створення контенту або сканування сторінок. Це досягається за допомогою механізмів rate limiting, які контролюють, скільки запитів може здійснити певна IP-адреса або сесія за заданий проміжок часу.

Наприклад, не більше 5 POST-запитів на логін протягом 1 хвилини. У разі перевищення лімітів — запит блокується, або повертається помилка зі статусом 429 Too Many Requests.

Для досягнення максимальної безпеки передбачається перевірка прав доступу не лише на рівні контролерів (веб-інтерфейсу), але й у сервісному рівні логіки. Це означає, що навіть якщо буде обійдено інтерфейсний контроль, всі критичні операції (наприклад, збереження документів, зміна профілю, відправка повідомлень) перевіряють повноваження користувача на глибшому рівні. Це дає змогу унеможливити прямий доступ до внутрішніх методів через, наприклад, зламані POST-запити.

2.5 Висновок

У результаті проведеного аналізу було сформовано цілісне уявлення про стан безпеки інформаційної системи фінансового документообігу. Виявлено наявність низки вразливостей, які створюють потенційні загрози як з боку зовнішніх атакувальників, так і внутрішніх користувачів. До найважливіших проблем віднесено відсутність шифрування внутрішнього чату, можливість SQL-ін’єкцій, ризики XSS-атак, акумуляцію неактивних сесій, а також необмежене введення даних, що може використовуватися для створення навантаження на систему.

Був сформований структурований перелік загроз, що охоплює порушення конфіденційності, цілісності та доступності інформації. Особливу увагу приділено технічним загрозам, що стосуються незахищених каналів зв’язку, неправильного розмежування прав доступу, а також ненадійної авторизації. Водночас розглядалися й непрямі фактори ризику, які можуть виникати внаслідок людського чинника, помилок конфігурації або некоректної реалізації бізнес-логіки.

На основі отриманої інформації було окреслено характеристики потенційного порушника, його можливості, мотивацію та рівень технічної обізнаності. Це дозволило краще зрозуміти, яким саме чином можуть реалізовуватись атаки, з якого середовища вони походять, та як саме має будуватись система захисту.

У підсумку запропоновано комплексний підхід до забезпечення інформаційної безпеки системи. Впроваджуються сучасні механізми криптографічного захисту, включаючи енд-ту-енд шифрування чату, система рольового доступу, фільтрація введення, обмеження запитів, захищені сесії та централізоване логування. Усі заходи проектуються з урахуванням реальної архітектури (Spring MVC), сесійної моделі автентифікації та особливостей web-взаємодії.

Таким чином, було створено обґрунтовану модель захисту, адаптовану до функціоналу застосунку та сучасних вимог до інформаційної безпеки в сфері фінансових web-систем.

3 ІМПЛЕМЕНТАЦІЯ І ТЕСТУВАННЯ

3.1 Імплементація запропонованих рішень

На цьому етапі відбувається безпосереднє впровадження заходів інформаційної безпеки, розроблених у межах попереднього етапу проектування. Основна мета полягає в тому, щоб інтегрувати механізми захисту в архітектуру системи фінансового документообігу без порушення її функціональних характеристик та з урахуванням обраної технологічної бази, зокрема Spring MVC, MySQL та сесійної моделі автентифікації.

Функціонал чату було доопрацьовано відповідно до принципів енд-ту-енд шифрування. Під час реєстрації кожного користувача на клієнтській стороні генеруються пара криптографічних ключів за алгоритмом RSA: публічний та приватний ключ. Публічний ключ призначений для шифрування даних і зберігається на сервері у відкритому вигляді. Приватний ключ, який використовується для дешифрування отриманих повідомлень, шифрується на клієнті із застосуванням пароля користувача та передається на сервер у зашифрованому вигляді. Таким чином, навіть у разі компрометації серверної частини системи, отриманий приватний ключ залишається захищеним, оскільки його розшифрування можливе лише за наявності правильного пароля користувача.

У процесі авторизації, після введення пароля, клієнт отримує з бекенду зашифрований приватний ключ, який потім розшифровується на клієнтській стороні за допомогою наданого пароля. Розшифрований ключ зберігається в localStorage браузера й використовується для дешифрування отриманих повідомлень та розшифрування AES-ключів чату. Завдяки такій архітектурі приватний ключ користувача не покидає клієнтську сторону в незашифрованому вигляді на жодному етапі життєвого циклу.

При створенні нового чату генерується унікальний симетричний ключ шифрування за алгоритмом AES-256, який надалі використовується для

шифрування всіх повідомлень, що передаються в межах цього чату. З метою безпечного розповсюдження симетричного ключа між учасниками чату, кожному з них він передається у зашифрованому вигляді з використанням їхнього відкритого RSA-ключа. Таким чином, лише власник відповідного приватного ключа зможе розшифрувати AES-ключ чату та використовувати його для перегляду вмісту повідомлень.

Зазначена модель гарантує цілісність і конфіденційність інформації, дозволяючи зменшити ризики, пов’язані з перехопленням або витоком даних. Важливо, що серверна частина системи не має технічної можливості отримати доступ до відкритого вмісту повідомлень або ключів, які використовуються для їх шифрування, що відповідає принципам Zero Knowledge Architecture.

Під час реєстрації користувача генерується пара RSA-ключів, після чого приватний ключ шифрується введеним паролем. Зашифрований приватний ключ разом із публічним надсилається на сервер, де вони зберігаються без можливості доступу до незашифрованих даних. Такий механізм забезпечує криптографічний захист з боку клієнта і виключає ризик розкриття ключової інформації навіть у випадку компрометації сервера (рис. 3.1).

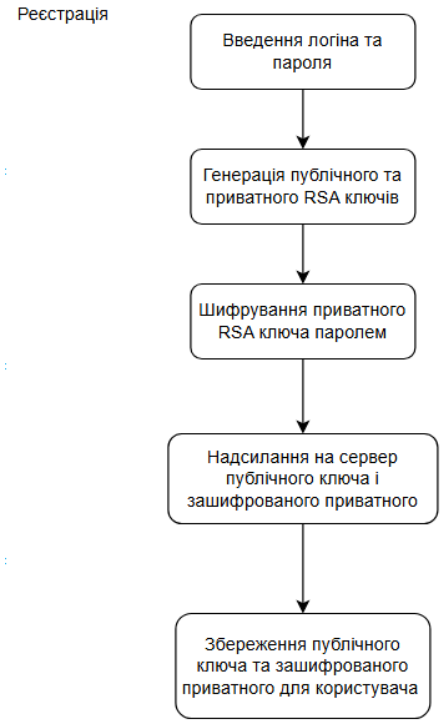


Рисунок 3.1 – Логіка під час реєстрації, збереження RSA пари ключів

Під час авторизації користувача на клієнтську частину передається зашифрований приватний ключ. Після введення пароля відбувається його розшифрування безпосередньо у браузері, а сам приватний ключ тимчасово зберігається в local storage для використання під час комунікації (рис. 3.2).

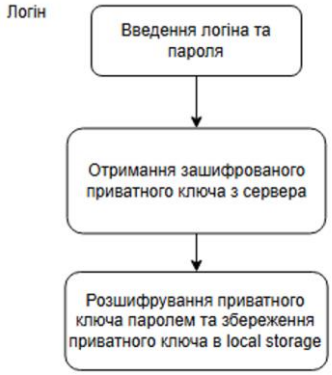


Рисунок 3.2 – Логіка під час логіну, отримання і збереження приватного ключа

Після успішного входу в систему користувач може ініціювати новий чат. Під час створення нового чату генерується унікальний симетричний ключ, який буде використовуватися для шифрування всіх повідомлень у рамках цього чату. Цей ключ далі шифрується відкритими ключами всіх учасників чату і надсилається їм індивідуально (рис. 3.3).

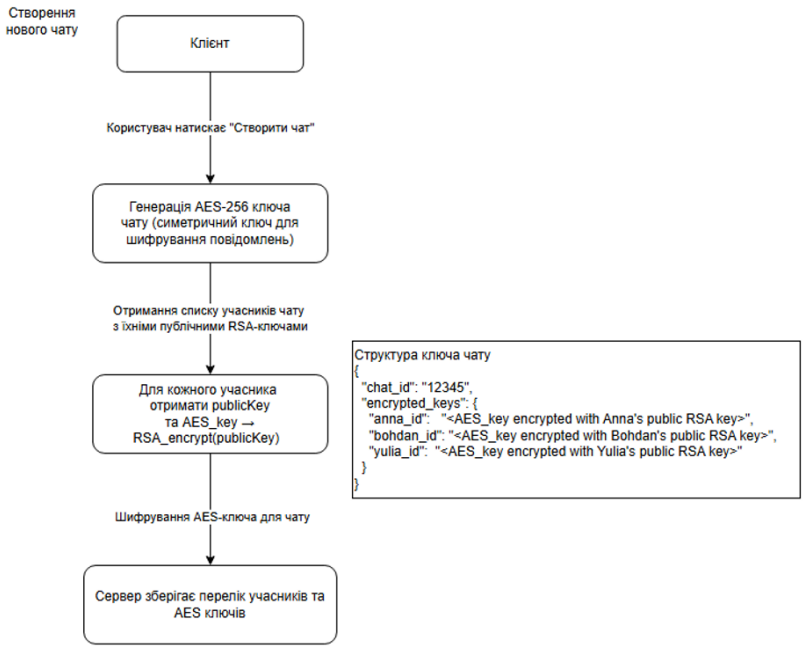


Рисунок 3.3 – Логіка створення нового чату, створення симетричного ключа чату

Повідомлення, яке вводить користувач, шифрується на клієнтській стороні за допомогою симетричного ключа AES, унікального для кожного чату. Цей AES-ключ зберігається в зашифрованому вигляді для кожного учасника чату: його копія шифрується відкритим RSA-ключем відповідного користувача. Після входу в чат користувач розшифровує свій варіант AES-ключа за допомогою свого приватного ключа, що дозволяє йому читати та надсилати повідомлення. Таким чином, лише учасники з відповідним приватним ключем можуть отримати доступ до змісту повідомлень, що забезпечує наскрізне (end-to-end) шифрування всієї переписки (рис. 3.4).

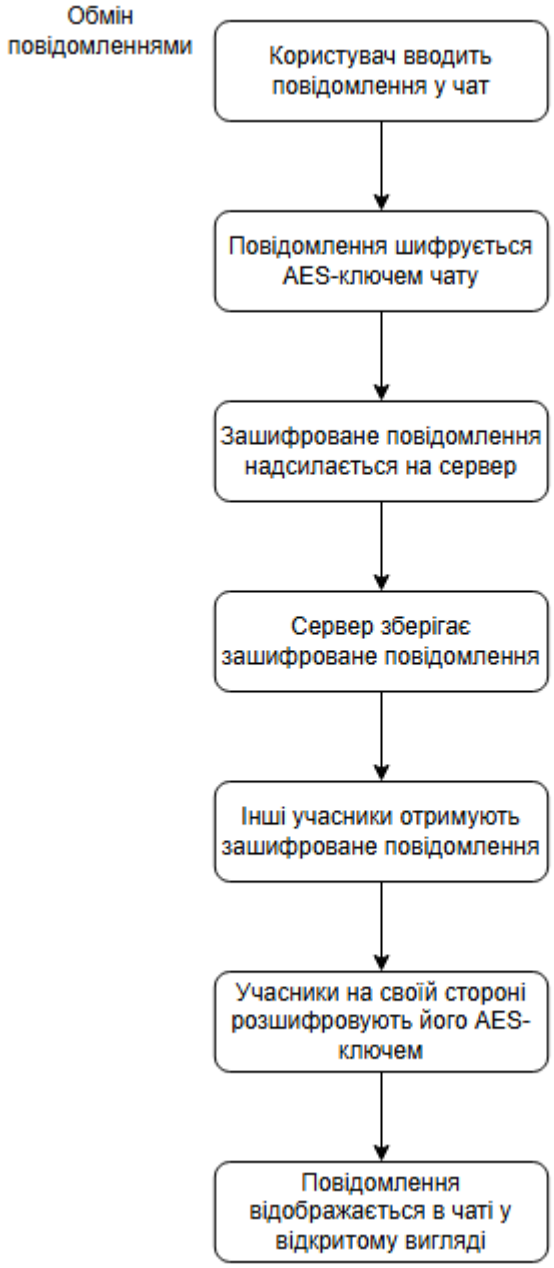


Рисунок 3.4 – Логіка відправки повідомлення

Захист форм та валідація введення. Було переглянуто усі форми у веб-інтерфейсі та серверні обробники з метою впровадження валідації введених даних. Було замінено просту ручну перевірку на повноцінне використання системи валідацій Spring Framework. Усі DTO-класи для обробки форм тепер оснащені анотаціями @NotBlank, @Size, @Pattern, @Email тощо, що забезпечує автоматичну перевірку структури, формату та довжини полів (рис. 3.5). На клієнтському рівні додано HTML5-атрибути валідації (maxlength, pattern, required) та доповнено валідацію JavaScript-функціями для відображення користувачу причин відмови ще до надсилання форми.

```
@Data 9 usages  Illya *
public class BuyerSaveDto {
    private Long id;
    @NotBlank(message = "Must not be empty")
    @Size(min = 2, max = 30, message = "Must be between 2 and 30 characters")
    @Pattern(regexp = "^((?![\\s]).)*$", message = "Must not contain spaces")
    private String name;
    @NotBlank(message = "Must not be empty")
    @Size(min = 2, max = 30, message = "Must be between 2 and 30 characters")
    @Pattern(regexp = "^((?![\\s]).)*$", message = "Must not contain spaces")
    private String surname;
    @NotBlank(message = "Must not be empty")
    @Size(min = 2, max = 30, message = "Must be between 2 and 30 characters")
    @Pattern(regexp = "^((?![\\s]).)*$", message = "Must not contain spaces")
    private String lastname;
    @Size(min = 2, max = 30, message = "Must be between 2 and 30 characters")
    @NotBlank(message = "Must not be empty")
    private String address;
    @Size(min = 2, max = 30, message = "Must be between 2 and 30 characters")
    @NotNull(message = "Must not be empty")
    private Long idNumber;
}
```

Рисунок 3.5 – Приклад серверної валідації DTO запиту створення покупця

Захист від SQL-ін’єкцій. Було повністю відмовлено від використання вручну сформованих SQL-запитів. Усі існуючі репозиторії було реорганізовано для взаємодії через JPA-інтерфейси. У тих місцях, де використовувались

EntityManager.createNativeQuery(...), ці виклики було замінено на JPQL або Criteria API. Це забезпечує автоматичне екранізування параметрів і унеможливлює SQL-ін’єкції. Також було проведено аудит усіх запитів на предмет потенційного впливу введених користувачем даних на структуру запиту. Додатково у властивостях підключення до бази вимкнено можливість мультизапитів (multiQueries), що блокує складні ланцюгові атаки.

Захист від XSS-атак. Було впроваджено обов’язкове екранування всіх динамічних даних, які відображаються в HTML-шаблонах. Усі змінні, що підставляються у вивід через Thymeleaf, тепер використовують синтаксис ${value} без ручного вимкнення екранування, що автоматично перетворює небезпечні символи (<, >, ", ') у їх HTML-еквіваленти. Крім того, додано спеціальний сервіс-фільтр, який очищає вхідні текстові поля від небажаних тегів, скриптів та інструкцій (рис. 3.6). Було заборонено передачу нестандартного HTML через всі публічні форми. Таким чином, навіть якщо користувач вставляє скрипт, система не виведе його у виконуваному вигляді.

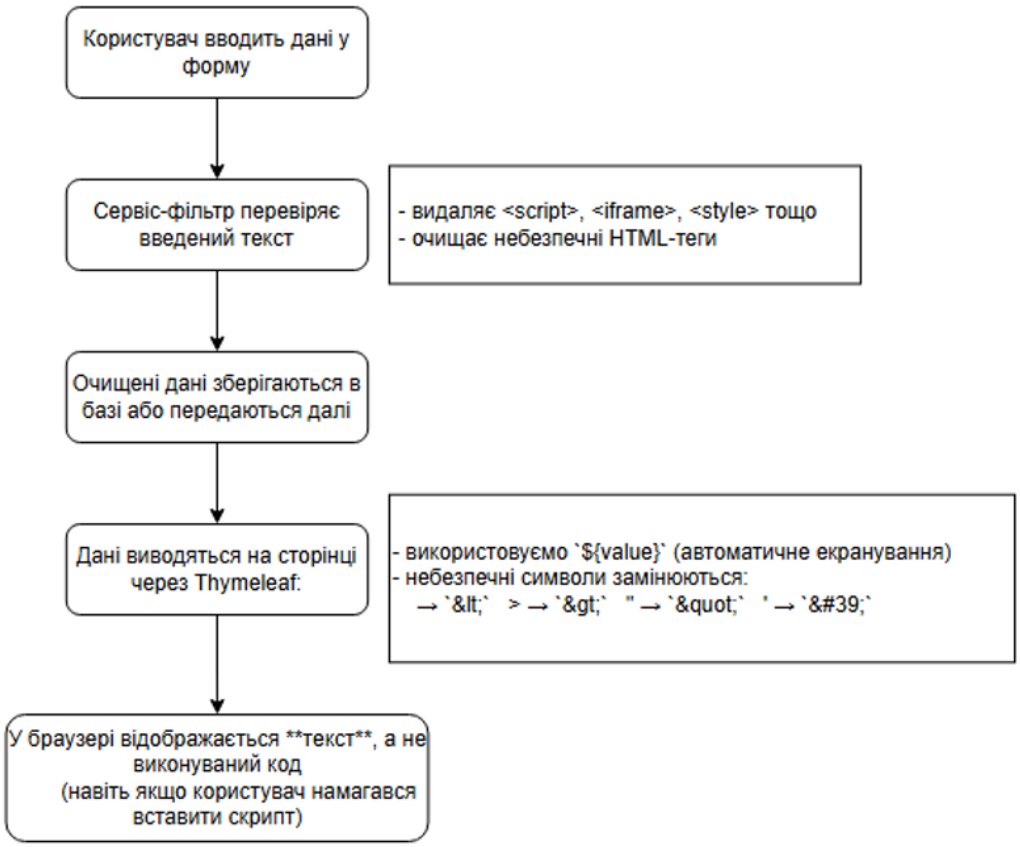


Рисунок 3.6 – Фільтр скриптів

Рольова модель доступу. Система контролю доступу була перероблена відповідно до моделі RBAC. Було створено окрему таблицю roles з відповідними зв’язками до користувачів та permissions з відповідними зв’язками до ролей. У Spring Security налаштовано обмеження маршрутів через antMatchers() у Java-конфігурації, що чітко визначає дозволені дії для кожної ролі (рис. 3.7). Також у контролерах було додано анотації @PreAuthorize для функцій, що виконують зміну даних. У результаті, навіть якщо користувач вручну звернеться до захищеного ресурсу, система перевірить його повноваження ще до виконання бізнес-логіки. Таким чином, ізоляція рівнів доступу стала більш жорсткою та контрольованою.

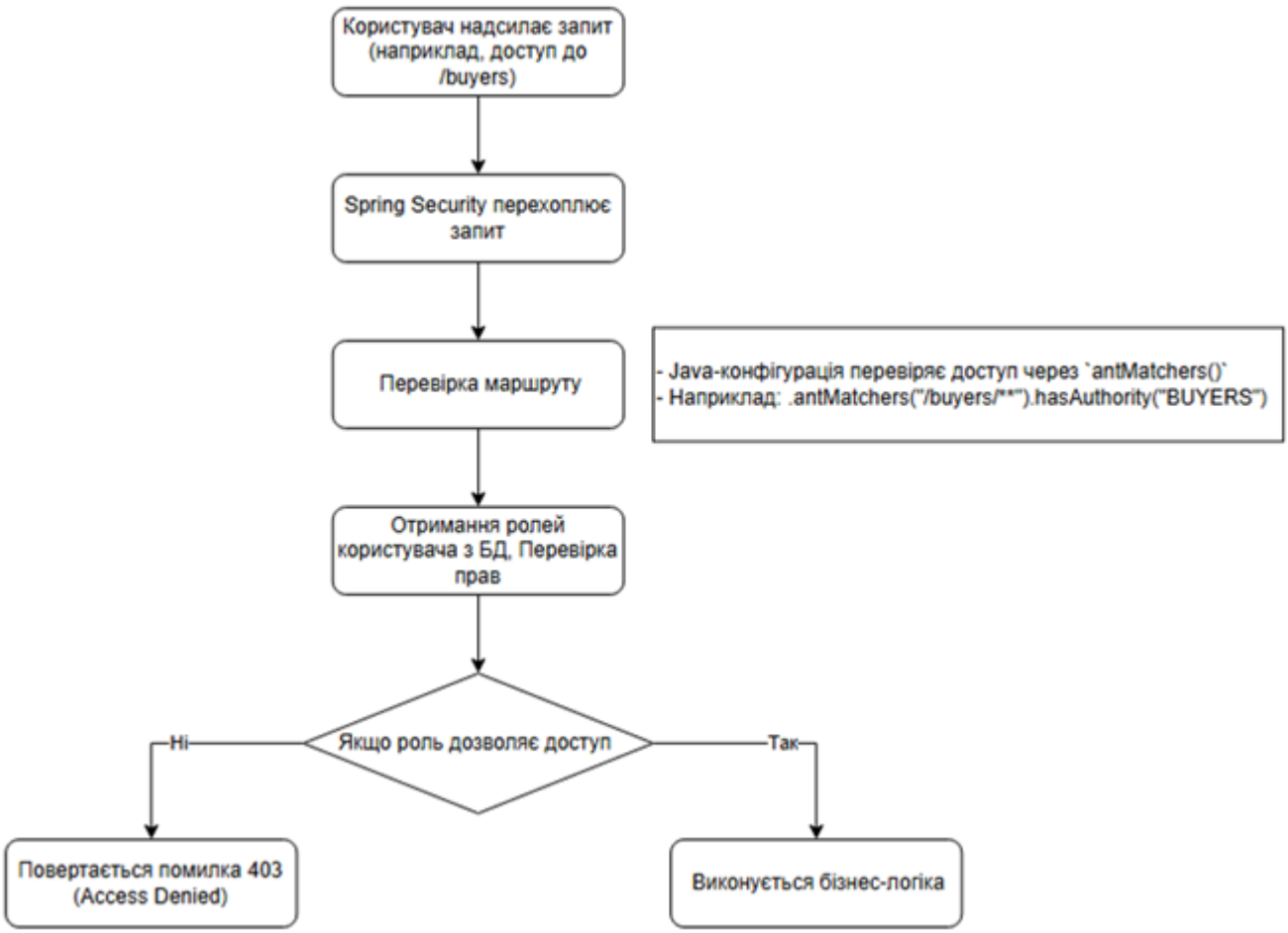


Рисунок 3.7 – SecurityConfig з обмеженням маршрутів

Управління сесіями. Налаштування безпеки сесій було оновлено з урахуванням сучасних рекомендацій OWASP. У конфігурації Spring Security

встановлено максимальну тривалість сесії без активності — 15 хвилин, після чого сесія автоматично завершиться. Для додаткового захисту cookie з ідентифікатором сесії налаштовано з прапорцями HttpOnly та Secure. Це запобігає доступу до сесійних даних зі сторони JavaScript, а також блокує передачу через незашифровані з'єднання. Увімкнено обмеження на кількість одночасних активних сесій для одного користувача — при повторному вході попередня сесія може завершуватись примусово або блокуватись за політикою.

В application.properties додано server.servlet.session.timeout=15m

Логування подій у системі реалізовано централізоване логування всіх ключових дій, що здійснюються над користувачами: отримання списку користувачів, пошук директорів та менеджерів, створення, оновлення та логічне видалення облікових записів. Для цього використовується логер Slf4j на основі Logback, який фіксує кожен запит із вказанням параметрів сортування, ID користувача, та інформації про результат операції (наприклад, успішне збереження або видалення).

Лог-файли формуються зі вказанням часу події, типу дії, ідентифікатора користувача та короткого опису результату. Це дозволяє здійснювати аудит подій і швидко виявляти потенційні інциденти.

З метою підвищення захищеності програмного комплексу від автоматизованих атак типу brute-force, а також запобігання надмірному навантаженню на серверну частину, у системі реалізовано механізм обмеження частоти запитів (rate limiting). Зазначена функціональність реалізована шляхом розробки спеціалізованого фільтра GlobalRateLimitingFilter, який інтегрується у ланцюг обробки HTTP-запитів.

Фільтр здійснює моніторинг кількості запитів, що надходять від кожної окремої IP-адреси протягом визначеного інтервалу часу. Параметри обмеження (допустима кількість запитів та тривалість інтервалу) конфігуруються через зовнішній файл налаштувань application.properties: rate.limiter.global.requests=100 та rate.limiter.global.duration.seconds=60

В основі реалізації лежить використання потокобезпечної структури ConcurrentHashMap, де для кожної IP-адреси зберігається список часових міток запитів. Під час кожного звернення виконується очищення історії запитів від записів, що виходять за межі допустимого часового вікна, якщо кількість актуальних записів перевищує допустимий поріг, користувачу повертається статус HTTP 429 (Too Many Requests) із відповідним текстовим повідомленням про перевищення ліміту (рис. 3.8). У випадку, якщо обмеження не перевищено, запит допускається до подальшої обробки, а поточна часова мітка додається до списку.

Завдяки такому підходу забезпечується ефективне протидіяння зловживанням API з боку окремих користувачів або автоматизованих скриптів, при цьому зберігаючи прозорість і безперервність роботи для добросовісних клієнтів системи.

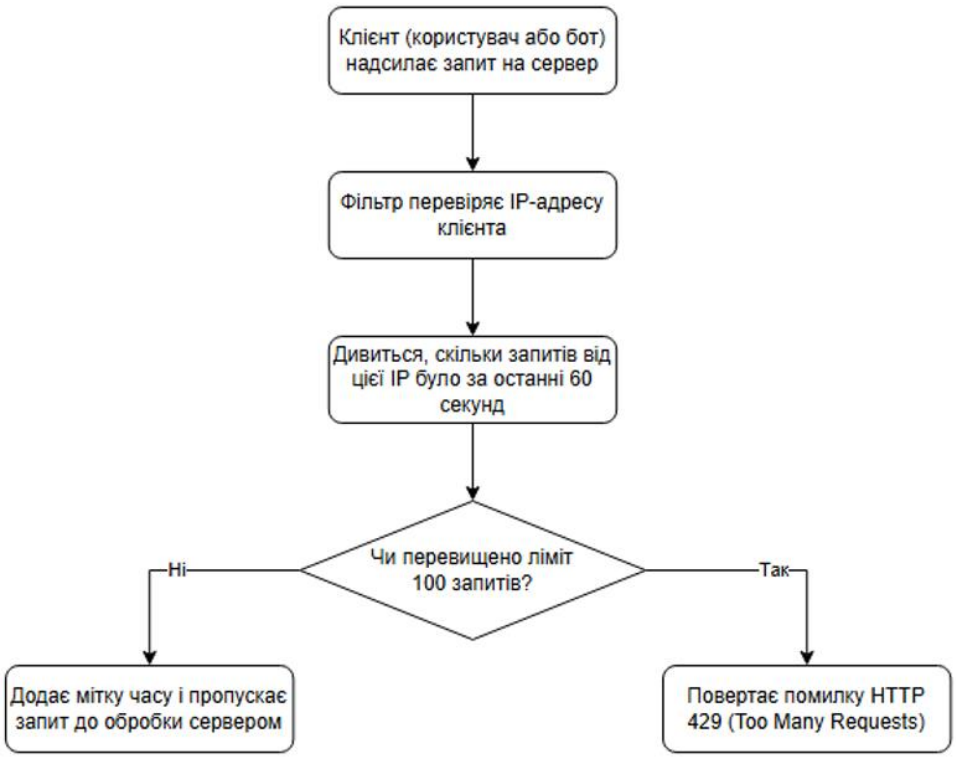


Рисунок 3.8 – Глобальний фільтр обмеження кількості запитів

3.2 Тестування системи

Після завершення процесу імплементації та апробації захисних механізмів, проводиться повномасштабне тестування інформаційної системи. Основною метою цього етапу є перевірка коректності роботи функціоналу, стійкості до типових векторів атак, відповідності вимогам безпеки, а також стабільності при взаємодії з навантаженням.

Тестування охоплює як функціональні, так і нефункціональні аспекти системи. Зокрема, виконуються перевірки логіки доступу, цілісності збережених даних, відповідності поведінки інтерфейсу, реакції системи на некоректні введення та загальну відмовостійкість. До процесу тестування залучаються автоматизовані скрипти, а також ручна перевірка критичних сценаріїв з боку адміністратора, менеджера та звичайного користувача.

На рівні функціонального тестування перевіряються всі основні сценарії користувацької взаємодії: реєстрація, вхід, створення та перегляд документів, надсилання повідомлень у чаті, робота з профілем та вихід із системи. Для кожної ролі визначається перелік дозволених дій, який порівнюється з фактично доступним функціоналом.

Окремо тестуються форми створення документів, збереження змін, валідація обов’язкових полів та повідомлення про помилки. Усі результати порівнюються з очікуваною поведінкою. Наприклад, користувач без ролі адміністратора не має змоги змінити ролі інших користувачів — при спробі доступу до відповідного маршруту система повертає заборону (HTTP 403) або перенаправляє на головну сторінку.

Тестування безпеки. У межах тестування безпеки виконуються спроби моделювання типових атак: SQL-ін’єкцій, XSS-ін’єкцій, атак на сесію, підбір паролів, спроби доступу до чужих даних. Для SQL-ін’єкцій вводяться небезпечні символи у поля пошуку, входу та форм, які взаємодіють із базою. Система успішно обробляє ці запити без зміни структури SQL, а в логах фіксується підозрілий ввід.

Для XSS-тестування вводяться скрипти у формати <script>, onload=, javascript: — жоден з них не виконується, а HTML-код екранізується або очищається фільтрами. У чаті, де раніше існувала загроза відображення небезпечного вмісту, повідомлення виводяться у безпечному форматі.

Тестується також надійність сесій — копіювання сесійного cookie в інший браузер не дає змоги авторизуватися, оскільки активна сесія пов’язується з IP або закінчується через таймаут. Перевірка механізму блокування після багаторазових невдалих спроб входу підтверджує наявність rate limiting — система тимчасово забороняє подальші запити, запобігаючи brute-force атакам.

Перевіряється повнота та точність логування дій у системі. Всі ключові події — авторизація, створення документів, помилки автентифікації, зміна ролей — записуються у лог-файли. Формат логів відповідає вимогам: кожна подія супроводжується часовою міткою, IP-адресою, ID користувача, коротким описом дії. Аналіз логів дозволяє відстежити не лише стандартну активність, а й виявити аномалії або спроби втручання в систему.

Тестування криптографічного захисту. Перевіряється робота шифрування повідомлень у чаті. На сервері підтверджується, що всі повідомлення зберігаються у зашифрованому вигляді, без прямого доступу до вмісту. У клієнтському інтерфейсі повідомлення коректно дешифруються лише для одержувача. Спроби перехоплення або підміни ключа не дозволяють зчитати зміст повідомлення, що підтверджує дієвість реалізованого енд-ту-енд шифрування.

Виконується обмежене навантажувальне тестування: одночасне надсилання сотень повідомлень у чаті, одночасне створення документів кількома користувачами, повторний вхід з кількох вкладок браузера. Система демонструє стабільну роботу, не допускає падінь або зависань. Всі операції відпрацьовуються в межах очікуваного часу, що свідчить про готовність системи до використання в реальному середовищі.

3.3 Висновок

У результаті практичного етапу роботи було реалізовано комплекс заходів з підвищення безпеки інформаційної системи фінансового документообігу. Запропоновані технічні рішення інтегровано безпосередньо в архітектуру програмного комплексу, з урахуванням особливостей побудови системи на основі Java Spring MVC, використання сесійної авторизації та взаємодії з базою даних через JPA.

Було здійснено повне впровадження енд-ту-енд шифрування внутрішнього чату, що забезпечує конфіденційність обміну повідомленнями на всіх етапах — від моменту надсилання до розшифрування одержувачем. Перероблено механізм збереження повідомлень таким чином, щоб сервер не мав доступу до незашифрованого вмісту. Це дозволяє надійно захистити внутрішню комунікацію навіть у разі компрометації серверної частини.

Усі запити до бази даних приведено до безпечного формату — замінено ручне формування SQL на підготовлені запити в межах ORM-рішення Hibernate. Також переглянуто механізми введення даних: реалізовано багаторівневу валідацію на клієнтському та серверному рівнях, встановлено обмеження на довжину текстових полів, запроваджено фільтрацію шкідливих конструкцій для запобігання XSS-атакам.

Було побудовано чітку модель доступу до ресурсів системи на основі ролей користувачів. Завдяки впровадженню механізму RBAC, кожен користувач отримує доступ лише до того функціоналу, який відповідає його ролі. Реалізовано обмеження на рівні контролерів, сервісів та шаблонів інтерфейсу, що дозволяє гарантувати цілісність моделі авторизації.

Апробація впроваджених рішень показала, що захисні механізми не порушують основної логіки роботи системи, зберігають її стабільність і забезпечують передбачувану поведінку у разі типових сценаріїв зловживання. У тестовому середовищі всі функції працюють відповідно до очікувань: дані шифруються, валідуються, фільтруються, а права доступу чітко обмежені.

Під час тестування системи підтверджено, що вона не допускає небезпечних SQL або XSS ін’єкцій, надійно блокує спроби обходу авторизації, фіксує підозрілу активність у логах та реагує на перевищення кількості запитів. Додатково перевірено роботу таймерів сесій та логіку автоматичного завершення роботи при неактивності користувача.

Таким чином, реалізовані рішення повністю відповідають сучасним вимогам інформаційної безпеки та успішно інтегруються в роботу веборієнтованої системи з документообігом фінансового характеру.

ВИСНОВКИ

У процесі виконання кваліфікаційної роботи було здійснено повний цикл дослідження, проєктування, реалізації та тестування системи захисту програмного комплексу фінансового документообігу на базі web-архітектури. Робота охопила як теоретичні, так і практичні аспекти інформаційної безпеки з урахуванням сучасних загроз, нормативних вимог та реальних технічних обмежень, властивих малому бізнесу.

У ході виконання кваліфікаційної роботи було виконано такі завдання:

- Сформульовано мету, завдання та предмет дослідження, а також визначено специфіку об'єкта — програмного комплексу, що реалізований із використанням Java Spring MVC, MySQL, сесійної авторизації та внутрішнього чату для обміну повідомленнями між користувачами.

- Проведено детальний огляд програмних рішень у сфері фінансового документообігу, а також проаналізовано сучасні методи захисту інформації у web-середовищах, включаючи криптографічні алгоритми, моделі контролю доступу, засоби виявлення атак та нормативно-правову базу України й міжнародні стандарти (ISO/IEC 27001).

- Проведено аудит існуючої версії програмного забезпечення, в ході якого виявлено низку критичних вразливостей: відсутність шифрування повідомлень, ризики SQL-ін’єкцій, міжсайтових скриптових атак (XSS), необмежене введення даних, відсутність тайм-ауту сесій та недостатню деталізацію прав доступу.

- Розроблено модель загроз інформаційної безпеки, що включає категоризацію ризиків за принципами CIA (конфіденційність, цілісність, доступність) та часткове використання методології STRIDE.

- Сформовано модель порушника, яка враховує як зовнішні, так і внутрішні джерела загроз, їхню ймовірну мотивацію, технічні ресурси, способи реалізації атак та вплив на функціонування системи.

- Запроєктовано і реалізовано систему захисту, до складу якої входять: впроваджене енд-ту-енд шифрування чату (на основі AES-256 та PKC), фільтрація та валідація вхідних даних, захист від SQL-ін’єкцій (ORM, параметризовані

запити), обмеження доступу за допомогою RBAC, захист сесій (тайм-аут, захищене cookie-зберігання), логування дій користувачів, механізм обмеження частоти запитів (rate limiting) для протидії brute-force та DDoS-атакам.

- Проведено комплексне тестування реалізованих заходів, що включало ручну перевірку, тестування з використанням спеціалізованих сканерів вразливостей та імітацію потенційних атак. Результати підтвердили підвищення рівня захисту системи на всіх критичних етапах обробки та передачі даних.

Таким чином, мета кваліфікаційної роботи досягнута в повному обсязі, а саме: створено систему захисту програмного комплексу фінансового документообігу з web-архітектурою, яка забезпечує конфіденційність, цілісність і доступність даних відповідно до сучасних вимог інформаційної безпеки. Реалізовані технічні рішення демонструють ефективність не лише в умовах академічного середовища, але й можуть бути практично застосовані в комерційних умовах.

Під час виконання роботи було не лише виявлено й усунуто критичні вразливості в існуючій системі, але й створено гнучку, масштабовану архітектуру захисту, яка може бути адаптована до нових викликів у сфері кібербезпеки. Запропонована система розроблена з урахуванням потреб малого бізнесу: вона не потребує значних фінансових витрат, сумісна з існуючою інфраструктурою підприємства та є перспективною для подальшого розвитку — зокрема, інтеграції з зовнішніми сервісами, мобільними додатками або хмарними платформами.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАНЬ

1. Автоматизація бухгалтерського обліку [Електронний ресурс] – Режим доступу: https://a4.com.ua/avtomatizacija-buhgalterskogo-obliku-jak-pidvishhiti-efektivnist-roboti-ta-zniziti-pomilki/ (дата звернення 17.02.2025)

2. Огляд функцій SAP Financials для фінансового обліку [Електронний ресурс] – Режим доступу: https://www.sap.com/ukraine/products/financial-management/green-ledger.html (дата звернення 12.02.2025)

3. Архітектура Oracle Financials Cloud [Електронний ресурс] – Режим доступу: https://www.oracle.com/uk/erp/financials/ (дата звернення 14.02.2025)

4. Функціональність Odoo у фінансовому документообігу [Електронний ресурс] – Режим доступу: https://www.odoo.com/uk_UA/app/accounting-features (дата звернення 15.02.2025)

5. Інструменти бюджетування в ERP-системах [Електронний ресурс] – Режим доступу: https://www.lighthouseindia.com/erp-for-software-budgeting.html (дата звернення 18.02.2025)

6. Формати даних XML та JSON [Електронний ресурс] – Режим доступу: https://aws.amazon.com/compare/the-difference-between-json-xml/ (дата звернення 24.02.2025)

7. Принципи безпечної веб-архітектури [Електронний ресурс] – Режим доступу: https://www.legitsecurity.com/aspm-knowledge-base/security-design-principles (дата звернення 20.02.2025)

8. Переваги хмарних технологій у фінансових системах [Електронний ресурс] – Режим доступу: https://adivi.com/blog/benefits-of-cloud-computing-in-financial-services/ (дата звернення 19.02.2025)

9. Безпека REST API у фінансових додатках [Електронний ресурс] – Режим доступу: https://www.stackhawk.com/blog/rest-api-security-best-practices/ (дата звернення 21.02.2025)

10. Thibodeau, M., Proctor, A., Francksen, T., Hains, D., Mullan, S., Church, I., Labbé-Morissette, G. (2021). Cloud-Based Technology Solution for Remote Work and Learning. *The International Hydrographic Review*, (25), 151-157.

11. Pronoza, P., Chernyshov, V., Aleksieienko, I. (2023). Optimization of business processes in investment using automation technology, financial calculations, and risk assessment methods.

12. Переваги електронного документообігу [Електронний ресурс] – Режим доступу: https://kefron.com/2023/04/7-key-benefits-of-electronic-document-management-systems/?switch=uk (дата звернення 22.02.2025)

13. Застосування AES для шифрування даних [Електронний ресурс] – Режим доступу: https://nordlayer.com/blog/aes-encryption/ (дата звернення 25.02.2025)

14. DES: обмеження та історія використання [Електронний ресурс] – Режим доступу: https://en.wikipedia.org/wiki/Data_Encryption_Standard (дата звернення 01.03.2025)

15. Алгоритм IDEA для захисту даних [Електронний ресурс] – Режим доступу: https://www.techtarget.com/searchsecurity/definition/International-Data-Encryption-Algorithm (дата звернення 02.03.2025)

16. Використання PES у криптографії [Електронний ресурс] – Режим доступу: https://eprint.iacr.org/2002/152.pdf (дата звернення 03.03.2025)

17. Криптографія (PKC) [Електронний ресурс] – Режим доступу: https://hexn.io/blog/public-key-cryptography-pkc-explained-100 (дата звернення 26.02.2025)

18. Роль цифрового підпису у фінансовій безпеці [Електронний ресурс] – Режим доступу: https://www.iplocation.net/the-role-of-digital-signatures-in-securing-financial-transactions (дата звернення 23.02.2025)

19. Протоколи SSL/TLS у веб-додатках [Електронний ресурс] – Режим доступу: https://www.f5.com/glossary/ssl-tls-encryption (дата звернення 27.02.2025)

20. Багатофакторна аутентифікація у фінансовому ПЗ [Електронний ресурс] – Режим доступу: https://www.paloaltonetworks.com/cyberpedia/what-are-multi-factor-authentication-mfa-examples-and-methods (дата звернення 28.02.2025)

21. VPN як засіб захисту інформації [Електронний ресурс] – Режим доступу: https://www.cloudflare.com/learning/access-management/vpn-security/ (дата звернення 04.03.2025)

22. Протокол IPsec у захисті мереж [Електронний ресурс] – Режим доступу: https://www.paloaltonetworks.com/cyberpedia/what-is-ipsec (дата звернення 05.03.2025)

23. IDS/IPS системи виявлення загроз [Електронний ресурс] – Режим доступу: https://www.dataguard.com/blog/how-intrusion-detection-systems-help-identify-cyber-threats/ (дата звернення 06.03.2025)

24. Alles, M., Brennan, G., Kogan, A., Vasarhelyi, M. A. (2018). Continuous monitoring of business process controls: A pilot implementation of a continuous auditing system at Siemens. In *Continuous Auditing: Theory and Application* (pp. 219-246). Emerald Publishing Limited.

25. України, З. (2010). Про захист персональних даних. *Відомості Верховної Ради України*, (34), 2297-17.

26. Галаджун, З. В. (2019). Становлення Закону України Про інформацію (від 1992 по 2017 рік). *Теле-та радіожурналістика*, (18), 126-137.

27. України, З. (2017). Про основні засади забезпечення кібербезпеки України. *Відомості Верховної Ради (ВВР)*, *45*, 2163-19.

28. Вимоги до засобів криптографічного захисту інформації [Електронний ресурс] – Режим доступу: https://zakon.rada.gov.ua/laws/show/z0696-21#Text (дата звернення 07.03.2025)

29. Комплексна система захисту інформації (КСЗІ) [Електронний ресурс] – Режим доступу: https://usts.kiev.ua/kompleksni-systemy-zakhystu-informatsii-kszi/ (дата звернення 08.03.2025)

30. ISO/IEC 27001 [Електронний ресурс] – Режим доступу: https://www.iso.org/standard/27001 (дата звернення 07.03.2025)

31. Cerny, T., Abdelfattah, A. S., Bushong, V., Al Maruf, A., Taibi, D. (2022, August). Microservice architecture reconstruction and visualization techniques: A

review. In *2022 IEEE International Conference on Service-Oriented System Engineering (SOSE)* (pp. 39-48). IEEE.

32. Gu, X., Zhang, H., Zhang, D., Kim, S. (2016, November). Deep API learning. In *Proceedings of the 2016 24th ACM SIGSOFT international symposium on foundations of software engineering* (pp. 631-642).

33. SOA (Service-Oriented Architecture) [Електронний ресурс] – Режим доступу: https://aws.amazon.com/what-is/service-oriented-architecture/ (дата звернення 09.03.2025)

34. Захист від атак типу MITM [Електронний ресурс] – Режим доступу: https://www.rapid7.com/fundamentals/man-in-the-middle-attacks/ (дата звернення 14.03.2025)

35. SQL-ін’єкції в системах [Електронний ресурс] – Режим доступу: https://www.vmware.com/topics/sql-injection-attack (дата звернення 15.03.2025)

36. Roth, S., Barron, T., Calzavara, S., Nikiforakis, N., Stock, B. (2020, January). Complex security policy? a longitudinal analysis of deployed content security policies. In *Proceedings of the 27th Network and Distributed System Security Symposium (NDSS)*.

37. Sudhodanan, A., Carbone, R., Compagna, L., Dolgin, N., Armando, A., Morelli, U. (2017, April). Large-scale analysis detection of authentication cross-site request forgeries. In *2017 IEEE European symposium on security and privacy (EuroSandP)* (pp. 350-365). IEEE.

38. Захист від XSS-атак [Електронний ресурс] – Режим доступу: https://portswigger.net/web-security/cross-site-scripting (дата звернення 30.03.2025)

39. Логування подій у безпеці [Електронний ресурс] – Режим доступу: https://www.mezmo.com/learn-security/what-are-security-event-logs (дата звернення 16.03.2025)

40. DDOS-attack [Електронний ресурс] – Режим доступу: https://www.cloudflare.com/learning/ddos/what-is-a-ddos-attack/ (дата звернення 17.03.2025)

41. Rate Limiting [Електронний ресурс] – Режим доступу: https://www.imperva.com/learn/application-security/rate-limiting/ (дата звернення 17.03.2025)

42. End-to-End encryption [Електронний ресурс] – Режим доступу: https://www.ibm.com/think/topics/end-to-end-encryption (дата звернення 17.03.2025)

43. Контроль доступу на основі ролей (RBAC) [Електронний ресурс] – Режим доступу: https://www.strongdm.com/rbac (дата звернення 25.03.2025)

44. Sahab, A., Trudel, S. (2020). COSMIC Functional Size Automation of Java Web Applications Using the Spring MVC Framework. In *IWSM-Mensura*.

45. Botros, S., Tinley, J. (2021). *High Performance MySQL*. " O'Reilly Media, Inc.".

46. Alex, B., Taylor, L. (2022). *Spring Security*.

47. Моделювання загроз для веб-систем [Електронний ресурс] – Режим доступу: https://www.blackduck.com/glossary/what-is-threat-modeling.html (дата звернення 04.04.2025)

48. Xiong, W., Legrand, E., Åberg, O., Lagerström, R. (2022). Cyber security threat modeling based on the MITRE Enterprise ATTandCK Matrix. *Software and Systems Modeling*, *21*(1), 157-177.

49. Модель порушника для веб-систем [Електронний ресурс] – Режим доступу: https://www.rusnauka.com/11_EISN_2010/Informatica/63866.doc.htm (дата звернення 04.04.2025)

50. Avkurova, Z., Gnatyuk, S., Abduraimova, B., Fedushko, S., Syerov, Y., Trach, O. (2022). Models for early web-attacks detection and intruders identification based on fuzzy logic. *Procedia Computer Science*, *198*, 694-699.

51. Анікін В. А. Система захисту програмного комплексу фінансового документообігу з вебархітектурою / В. А. Анікін І. Д. Розгон, М. І. Федорчук // Збірник тез доповідей XX міжнародної науково-практичної конференції «Військова освіта і наука: сьогодення та майбутнє», 29 листопада 2024 р. - Київ : ВІКНУ, 2024. – Т. 1. – С. 33.

ДОДАТОК А

Глобальний фільтр обмеження кількості запитів

```
package com.example.NewBuildingFinance.config.ratelimiter;

import org.springframework.beans.factory.annotation.Value;
import org.springframework.stereotype.Component;
import org.springframework.web.filter.OncePerRequestFilter;

import javax.servlet.FilterChain;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import java.io.IOException;
import java.util.*;
import java.util.concurrent.ConcurrentHashMap;

@Component
public class GlobalRateLimitingFilter extends OncePerRequestFilter {

    @Value("${rate.limiter.global.requests}")
    private int maxRequests;

    @Value("${rate.limiter.global.duration.seconds}")
    private int durationInSeconds;

    private final Map<String, List<Long>> requestHistory = new
ConcurrentHashMap<>();

    @Override
```

```
    protected void doFilterInternal(HttpServletRequest request,
                                    HttpServletResponse response,
                                    FilterChain filterChain)
            throws ServletException, IOException {
        String ip = request.getRemoteAddr();
        long now = System.currentTimeMillis();

        requestHistory.putIfAbsent(ip, new ArrayList<>());
        List<Long> timestamps = requestHistory.get(ip);

        synchronized (timestamps) {
            timestamps.removeIf(timestamp -> now - timestamp > durationInSeconds *
1000L);

            if (timestamps.size() >= maxRequests) {
                response.setStatus(429);
                response.getWriter().write("Rate limit exceeded. Please try again later.");
                return;
            }

            timestamps.add(now);
        }

        filterChain.doFilter(request, response);
    }
}
```

Кешований текст запиту http-сервлета

```
package com.example.NewBuildingFinance.config.xssfilter;

import javax.servlet.ReadListener;
import javax.servlet.ServletInputStream;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletRequestWrapper;
import java.io.BufferedReader;
import java.io.ByteArrayInputStream;
import java.io.IOException;
import java.io.StringReader;
import java.util.stream.Collectors;

public class CachedBodyHttpServletRequest extends HttpServletRequestWrapper {
    private final String cachedBody;

    public CachedBodyHttpServletRequest(HttpServletRequest request) throws
IOException {
        super(request);
        this.cachedBody =
request.getReader().lines().collect(Collectors.joining(System.lineSeparator()));
    }

    public String getBody() {
        return cachedBody;
    }

    @Override
```

```
    public BufferedReader getReader() throws IOException {
        return new BufferedReader(new StringReader(cachedBody));
    }

    @Override
    public ServletInputStream getInputStream() throws IOException {
        final ByteArrayInputStream byteArrayInputStream = new
ByteArrayInputStream(cachedBody.getBytes());
        return new ServletInputStream() {
            @Override public boolean isFinished() { return
byteArrayInputStream.available() == 0; }
            @Override public boolean isReady() { return true; }
            @Override public void setReadListener(ReadListener readListener) {}
            @Override public int read() { return byteArrayInputStream.read(); }
        };
    }
}
```

Xss фільтр

```
package com.example.NewBuildingFinance.config.xssfilter;

import org.springframework.stereotype.Component;

import javax.servlet.*;
import javax.servlet.annotation.WebFilter;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import java.io.IOException;
import java.util.regex.Pattern;

@Component
@WebFilter("/*")
public class XssFilter implements Filter {

    private static final Pattern XSS_PATTERN = Pattern.compile(

"(<script.*?>.*?</script>)|((javascript:)|onerror=|onload=|<iframe|<object|<embed)",
            Pattern.CASE_INSENSITIVE | Pattern.DOTALL
    );

    @Override
    public void doFilter(ServletRequest request, ServletResponse response, FilterChain
chain)
            throws IOException, ServletException {

        HttpServletRequest httpReq = (HttpServletRequest) request;
```

```
        HttpServletResponse httpRes = (HttpServletResponse) response;

        // Лише перевірка текстових запитів (form-urlencoded / plain text / JSON)
        if (isModifyingRequest(httpReq) && isTextBasedContent(httpReq)) {
            CachedBodyHttpServletRequest cachedRequest = new
CachedBodyHttpServletRequest(httpReq);
            String body = cachedRequest.getBody();

            if (XSS_PATTERN.matcher(body).find()) {
                httpRes.sendError(HttpServletResponse.SC_BAD_REQUEST, "XSS content
detected");
                return;
            }

            chain.doFilter(cachedRequest, response);
        } else {
            chain.doFilter(request, response);
        }
    }

    private boolean isModifyingRequest(HttpServletRequest request) {
        String method = request.getMethod();
        return "POST".equalsIgnoreCase(method) ||
                "PUT".equalsIgnoreCase(method) ||
                "PATCH".equalsIgnoreCase(method);
    }

    private boolean isTextBasedContent(HttpServletRequest request) {
        String contentType = request.getContentType();
        return contentType != null && (
```

```
            contentType.contains("application/json") ||
                contentType.contains("application/x-www-form-urlencoded") ||
                contentType.contains("text")
        );
    }
}
```

Конфігурація безпеки

```
package com.example.NewBuildingFinance.config;

import com.example.NewBuildingFinance.service.auth.user.UserServiceImpl;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import
org.springframework.security.config.annotation.authentication.builders.Authentication
ManagerBuilder;
import org.springframework.security.config.annotation.web.builders.HttpSecurity;
import
org.springframework.security.config.annotation.web.configuration.EnableWebSecurit
y;
import
org.springframework.security.config.annotation.web.configuration.WebSecurityConfi
gurerAdapter;
import org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder;

@Configuration
@EnableWebSecurity
public class SecurityConfig extends WebSecurityConfigurerAdapter {
    @Autowired
    private UserServiceImpl userServiceImpl;
    @Bean
    public BCryptPasswordEncoder bCryptPasswordEncoder() {
        return new BCryptPasswordEncoder();
    }
```

```
@Override
protected void configure(HttpSecurity httpSecurity) throws Exception {
    httpSecurity
            .csrf()
                .disable()
                .authorizeRequests()
                .antMatchers("/registration/**").permitAll()
                .antMatchers("/profile/**").authenticated()
                .antMatchers("/statistics/**").hasAuthority("STATISTICS")
                .antMatchers("/flats/**").hasAuthority("FLATS")
                .antMatchers("/cashRegister/**").hasAuthority("CASH_REGISTER")
                .antMatchers("/buyers/**").hasAuthority("BUYERS")
                .antMatchers("/agencies/**").hasAuthority("AGENCIES")
                .antMatchers("/objects/**").hasAuthority("OBJECTS")
                .antMatchers("/contracts/**").hasAuthority("CONTRACTS")
                .antMatchers("/settings/**").hasAuthority("SETTINGS")
                .antMatchers("/chats/**").hasAuthority("CHATS")
            .and()
                .formLogin()
                .loginPage("/login")
                .defaultSuccessUrl("/profile/")
                .permitAll()
            .and()
                .logout()
                .permitAll()
                .logoutSuccessUrl("/login")
                .deleteCookies("JSESSIONID") // for remember me
            .and()
                .rememberMe()
```

```
                .key("uniqueAndSecret").tokenValiditySeconds(1209600); // Week 2
    }

    @Autowired
    protected void configureGlobal(AuthenticationManagerBuilder auth) throws
Exception {

auth.userDetailsService(userServiceImpl).passwordEncoder(bCryptPasswordEncoder(
));
    }
}
```

Сторінка чату

```
<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org">

<head>
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, initial-scale=1">
    <title>Chat</title>

    <style th:replace="blocks/linkAdminLTE"></style>
    <div th:replace="blocks/links"></div>
    <script th:src="@{/js/app.js}"></script>
    <link rel="stylesheet" th:href="@{/css/popup.css}">
</head>

<body class="sidebar-mini layout-fixed control-sidebar-slide-open" style="height:
auto;">
<div class="wrapper">

    <div th:replace="blocks/navbar"></div>
    <div th:replace="blocks/sidebarContainer"></div>

    <div class="content-wrapper">

        <div class="content-header">
            <div class="container-fluid">
                <h1 class="m-0">Chat</h1>
            </div>
        </div>
```

```
<section class="content">
    <div class="container">
        <div class="row justify-content-center mt-4">
            <div class="col-md-3">
                <!-- ЧАТИ -->
                <div class="card">
                    <div class="card-header bg-secondary text-white">
                        <h5 class="mb-0">Мої чати</h5>
                    </div>
                    <div class="list-group" id="chat-list" style="max-height: 500px;
overflow-y: auto;">
                        <!-- Чати будуть тут -->
                    </div>
                </div>
            </div>

            <div class="col-md-9">
                <!-- ВИБРАНИЙ ЧАТ -->
                <div class="card">
                    <div class="card-header bg-primary text-white d-flex justify-content-
between align-items-center">
                        <h3 class="card-title mb-0" id="chat-room-title">Кімната
чату</h3>
                        <button class="btn btn-sm btn-light"
onclick="refreshMessages()">Оновити</button>
                    </div>
                    <div class="card-body" id="chat-box" style="height: 400px;
overflow-y: scroll;">
                        <!-- Повідомлення будуть завантажені тут -->
```

```
                    </div>
                    <div class="card-footer">
                        <div class="input-group">
                            <input type="text" id="messageInput" class="form-control"
placeholder="Введіть повідомлення...">
                            <button class="btn btn-primary"
onclick="sendMessage()">Надіслати</button>
                        </div>
                    </div>
                </div>

                <!-- Форма для створення нового чату -->
                <div class="card mt-4">
                    <div class="card-header bg-success text-white">
                        <h5 class="mb-0">Створити новий чат</h5>
                    </div>
                    <div class="card-body">
                        <input id="chatName" class="form-control" style="width:
100%"/>
                        <select id="userSelect" class="form-control" style="width:
100%"></select>
                        <button class="btn btn-success mt-2"
onclick="createChat()">Створити чат</button>
                    </div>
                </div>
            </div>
        </div>
    </section>
</div>
```

```
</div>

<div th:replace="blocks/scriptAdminLTE"></div>

<script>

    // ----------------------------
    // Допоміжні функції для роботи з криптографією
    // ----------------------------

    let currentChatAESKey = null;

    // Перетворення рядка у ArrayBuffer
    function str2ab(str) {
        const buf = new ArrayBuffer(str.length);
        const bufView = new Uint8Array(buf);
        for (let i = 0; i < str.length; i++) {
            bufView[i] = str.charCodeAt(i);
        }
        return buf;
    }

    // Імпорт приватного ключа (PKCS8, Base64)
    async function importPrivateKey(pemKey) {
        try {
            console.log("Імпорт приватного ключа починається");
            const binaryDerString = window.atob(pemKey);
            const binaryDer = str2ab(binaryDerString);
            const key = await window.crypto.subtle.importKey(
                "pkcs8",
```

```
            binaryDer,
            {
                name: "RSA-OAEP",
                hash: "SHA-256"
            },
            true,
            ["decrypt"]
        );
        console.log("Імпорт приватного ключа успішний", key);
        return key;
    } catch (err) {
        console.error("Помилка імпорту приватного ключа:", err);
        throw err;
    }
}

async function decryptAESKey(encryptedAESKey, privateKey) {
    try {
        console.log("AES key decryption begins");
        const encryptedKeyBuffer =
Uint8Array.from(window.atob(encryptedAESKey), c => c.charCodeAt(0));
        console.log("Encrypted AES key (Uint8Array):", encryptedKeyBuffer);

        // Явно задаємо алгоритм з hash: "SHA-256" та порожнім label
        const algorithm = { name: "RSA-OAEP", hash: "SHA-256", label: new
Uint8Array(0) };
        const decrypted = await window.crypto.subtle.decrypt(
            algorithm,
            privateKey,
            encryptedKeyBuffer
```

```
        );
        console.log("AES key decrypted, bytes received:", new
Uint8Array(decrypted));

        // Імпортуємо отриманий AES ключ для подальшого використання в AES-
GCM
        const aesKey = await window.crypto.subtle.importKey(
            "raw",
            decrypted,
            { name: "AES-GCM" },
            true,
            ["encrypt", "decrypt"]
        );
        console.log("AES key imported successfully", aesKey);
        return aesKey;
    } catch (err) {
        console.error("Error decrypting AES key:", err);
        throw err;
    }
}

async function decryptMessageAES(aesKey, encryptedMessage, ivBase64) {
    try {
        const iv = Uint8Array.from(window.atob(ivBase64), c => c.charCodeAt(0));
        console.log("IV:", iv);
        const encryptedBuffer = Uint8Array.from(window.atob(encryptedMessage), c
=> c.charCodeAt(0));
        console.log("Encrypted message (Uint8Array):", encryptedBuffer);
        const decrypted = await window.crypto.subtle.decrypt(
            { name: "AES-GCM", iv: iv },
```

```
            aesKey,
            encryptedBuffer
        );
        const result = new TextDecoder().decode(decrypted);
        console.log("Decrypted message:", result);
        return result;
    } catch (err) {
        console.error("Message decryption error:", err);
        throw err;
    }
}

async function encryptMessageAES(aesKey, message) {
    try {
        // Створюємо випадковий вектор ініціалізації (IV) довжиною 12 байт
        const iv = window.crypto.getRandomValues(new Uint8Array(12));
        const encodedMessage = new TextEncoder().encode(message);

        // Шифруємо повідомлення AES-GCM
        const ciphertext = await window.crypto.subtle.encrypt(
            { name: "AES-GCM", iv: iv },
            aesKey,
            encodedMessage
        );

        // Конвертуємо зашифровані дані та IV в Base64 рядки для передачі
        const encryptedMessage = btoa(String.fromCharCode(...new
Uint8Array(ciphertext)));
        const ivBase64 = btoa(String.fromCharCode(...iv));
```

```
            console.log("Message encrypted successfully");
            return { encryptedText: encryptedMessage, iv: ivBase64 };
        } catch (err) {
            console.error("Message encryption error:", err);
            throw err;
        }
    }

    async function loadChatAESKey() {
        const response = await fetch(url + `/chats/${currentChatId}/aes-key`);
        const encryptedAESKey = await response.json(); // отримуємо Base64 рядок
        const privateKey = await importPrivateKey(userPrivateKey); // userPrivateKey
має бути встановлений
        currentChatAESKey = await decryptAESKey(encryptedAESKey, privateKey);
    }

</script>

<script>
    let url = window.location.protocol + "//" + window.location.host;
    let currentChatId = null;
    const userId = '[[${userId}]]';
    const userPrivateKey = '[[${privateKey}]]';  // Base64-кодований приватний ключ
користувача

    $(document).ready(function() {
        loadChats();
        initializeUserSelect();
    });
```

```
// ---------------------------
// Існуючі функції для роботи з чатами та повідомленнями
// ---------------------------

// Завантажити список чатів
function loadChats() {
    $.ajax({
        type: 'get',
        url: url + `/chats/${userId}`,
        dataType: "json",
        success: function (chats) {
            let chatList = document.getElementById('chat-list');
            chatList.innerHTML = '';
            chats.forEach(chat => {
                let chatItem = document.createElement('a');
                chatItem.href = '#';
                chatItem.classList.add('list-group-item', 'list-group-item-action');
                chatItem.textContent = chat.name;
                chatItem.onclick = function() {
                    currentChatId = chat.id;
                    document.getElementById('chat-room-title').innerText = chat.name;
                    highlightActiveChat(chat.id);
                    loadMessages();
                };
                chatItem.setAttribute('data-chat-id', chat.id);
                chatList.appendChild(chatItem);
            });
            if (chats.length > 0) {
                currentChatId = chats[0].id;
                document.getElementById('chat-room-title').innerText = chats[0].name;
```

```
                highlightActiveChat(currentChatId);
                loadMessages();
            }
            connectWebSocket();
        },
        error: function() {
            console.log('Error loading chats');
        }
    });
}

// Підсвітити активний чат
function highlightActiveChat(chatId) {
    let chatItems = document.querySelectorAll('#chat-list a');
    chatItems.forEach(item => {
        if (parseInt(item.getAttribute('data-chat-id')) === chatId) {
            item.classList.add('active');
        } else {
            item.classList.remove('active');
        }
    });
}

async function loadMessages() {
    if (!currentChatId) return;

    // Спочатку завантажуємо та розшифровуємо AES ключ для чату, якщо ще не зроблено
    if (!currentChatAESKey) {
        await loadChatAESKey();
```

```
    }

    $.ajax({
        type: 'get',
        url: url + `/chats/${currentChatId}/messages`,
        dataType: "json",
        success: async function (messages) {
            let chatBox = document.getElementById('chat-box');
            chatBox.innerHTML = '';  // Очищаємо чат перед завантаженням нових повідомлень

            // Для кожного повідомлення розшифровуємо текст
            for (let message of messages) {
                let decryptedText;
                try {
                    decryptedText = await decryptMessageAES(currentChatAESKey, message.content, message.iv);
                } catch (e) {
                    console.error('Error decrypting message', e);
                    decryptedText = "[Помилка розшифрування]";
                }

                // Визначаємо, чи це моє повідомлення
                const isMyMessage = +message.userId === +userId;
                const backgroundColor = isMyMessage ? '#e0f7fa' : '#f1f1f1';

                let messageDiv = document.createElement('div');
                messageDiv.classList.add('mb-2', 'p-2', 'rounded');
                messageDiv.style.backgroundColor = backgroundColor;
                messageDiv.style.display = 'flex';
```

```
                messageDiv.style.flexDirection = isMyMessage ? 'row' : 'row-reverse';
                messageDiv.style.alignItems = 'center';

                const messageDate = new Date(message.createdAt);
                const formattedDate = messageDate.toLocaleString();

                messageDiv.innerHTML = `
                <div style="max-width: 70%; word-wrap: break-word;">
                    <strong>${message.senderName}:</strong> ${decryptedText}
                    <br><small>${formattedDate}</small>
                </div>
            `;
                chatBox.appendChild(messageDiv);
            }
            chatBox.scrollTop = chatBox.scrollHeight;
        },
        error: function() {
            console.log('Error loading messages');
        }
    });
}

async function sendMessage() {
    let messageContent = document.getElementById('messageInput').value;
    if (!messageContent.trim() || !currentChatId) return;

    try {
        // Шифруємо повідомлення за допомогою currentChatAESKey
        const data = await encryptMessageAES(currentChatAESKey,
messageContent);
```

```
        $.ajax({
            type: 'post',
            url: url + `/chats/${currentChatId}/message`,
            contentType: "application/json",
            dataType: "json",
            data: JSON.stringify(data),
            success: function () {
                loadMessages();
            },
            error: function () {
                console.log('Error sending message');
            }
        });
    } catch (err) {
        console.error("Error in sendMessage:", err);
    }
}

// Оновити повідомлення
function refreshMessages() {
    loadMessages();
}

// Ініціалізація Select2 для вибору користувача для створення чату
function initializeUserSelect() {
    $('#userSelect').select2({
        ajax: {
            url: url + '/chats/available-users',
            dataType: 'json',
```

```
            delay: 250,
            data: function (params) {
                return {
                    searchTerm: params.term
                };
            },
            processResults: function (data) {
                return {
                    results: data.map(user => ({
                        id: user.id,
                        text: user.name + ' ' + user.surname
                    }))
                };
            }
        },
        placeholder: 'Search users'
    });
}

function createChat() {
    let selectedUserId = $('#userSelect').val();
    let chatName = $('#chatName').val();
    if (!selectedUserId || !chatName) {
        alert('Please select a user and enter a chat name');
        return;
    }

    $.ajax({
        url: url + `/chats/create-with-
user/${selectedUserId}?chatName=${encodeURIComponent(chatName)}`,
```

```
            method: 'POST',
            contentType: 'application/json',
            success: function () {
                loadChats();
            },
            error: function () {
                console.log('Error creating chat');
                alert('There was an error creating the chat');
            }
        });
    }

    function connectWebSocket() {
        addSubscription('/topic/chat/' + currentChatId, function (data) {
            loadMessages();
        })
    }
</script>

<script src="https://cdn.jsdelivr.net/npm/sockjs-client@1/dist/sockjs.min.js"></script>
<script src="https://cdn.jsdelivr.net/npm/stompjs@2.3.3/lib/stomp.min.js"></script>

<script src="https://unpkg.com/imask"></script>
<script th:src="@{/plugins/summernote/summernote-bs4.min.js}"></script>
<script src="https://unpkg.com/sweetalert/dist/sweetalert.min.js"></script>

</main>
</body>
</html>
```

Контролер чату

```
package com.example.NewBuildingFinance.controllers.chat;

import com.example.NewBuildingFinance.dto.chat.MessageRequest;
import com.example.NewBuildingFinance.dto.chat.MessageResponse;
import com.example.NewBuildingFinance.entities.auth.User;
import com.example.NewBuildingFinance.entities.chat.Chat;
import com.example.NewBuildingFinance.repository.chat.ChatRepository;
import com.example.NewBuildingFinance.service.auth.user.UserService;
import com.example.NewBuildingFinance.service.chat.ChatService;
import com.example.NewBuildingFinance.service.chat.MessageService;
import com.fasterxml.jackson.core.JsonProcessingException;
import com.fasterxml.jackson.core.type.TypeReference;
import com.fasterxml.jackson.databind.ObjectMapper;
import lombok.RequiredArgsConstructor;
import org.springframework.security.core.Authentication;
import org.springframework.security.core.context.SecurityContextHolder;
import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.validation.BindingResult;
import org.springframework.validation.FieldError;
import org.springframework.web.bind.annotation.*;

import javax.validation.Valid;
import java.util.HashMap;
import java.util.List;
import java.util.Map;

@Controller
```

```
@RequestMapping("/chats")
@RequiredArgsConstructor
public class ChatController {

    private final ChatService chatService;
    private final UserService userService;
    private final MessageService messageService;
    private final ChatRepository chatRepository;

    private final ObjectMapper mapper;

    @GetMapping()
    public String chats(
            Model model
    ){
        Authentication authentication =
SecurityContextHolder.getContext().getAuthentication();
        User user = userService.loadUserByUsername(authentication.getName());
        model.addAttribute("user", user);
        model.addAttribute("userId", user.getId());
        model.addAttribute("privateKey", user.getPrivateKey());
        return "chat/chats";
    }

    @GetMapping("/{userId}")
    @ResponseBody
    public String getChats(@PathVariable Long userId) throws
JsonProcessingException {
        return mapper.writeValueAsString(chatService.getChatsForUser(userId));
    }
```

```
    @PostMapping("/{chatId}/message")
    @ResponseBody
    public String sendMessage(
            @PathVariable Long chatId,
            @Valid @RequestBody MessageRequest messageRequest,
            BindingResult bindingResult
    ) throws JsonProcessingException {
        if(bindingResult.hasErrors()){
            Map<String, String> errors = new HashMap<>();
            for (FieldError error : bindingResult.getFieldErrors()) {
                errors.put(error.getField(), error.getDefaultMessage());
            }
            return mapper.writeValueAsString(errors);
        }

        MessageResponse message = messageService.sendMessage(chatId,
messageRequest);
        return mapper.writeValueAsString(message);
    }

    @GetMapping("/{chatId}/messages")
    @ResponseBody
    public String getMessages(@PathVariable Long chatId) throws
JsonProcessingException {
        return mapper.writeValueAsString(messageService.getMessages(chatId));
    }

    @GetMapping("/{chatId}/aes-key")
    @ResponseBody
```

```
    public String getChatAESKey(@PathVariable Long chatId) {
        Chat chat = chatRepository.findById(chatId).orElseThrow(() -> new
RuntimeException("Chat not found"));
        Authentication authentication =
SecurityContextHolder.getContext().getAuthentication();
        User currentUser = userService.loadUserByUsername(authentication.getName());
        try {
            Map<Long, String> encryptedKeys =
mapper.readValue(chat.getEncryptedAESKeys(), new TypeReference<Map<Long,
String>>() {});
            String encryptedAESKey = encryptedKeys.get(currentUser.getId());
            return mapper.writeValueAsString(encryptedAESKey);
        } catch(Exception e) {
            throw new RuntimeException("Error retrieving AES key", e);
        }
    }

    @GetMapping("/available-users")
    @ResponseBody
    public String getAvailableUsers() throws JsonProcessingException {
        Authentication authentication =
SecurityContextHolder.getContext().getAuthentication();
        User currentUser = userService.loadUserByUsername(authentication.getName());
        List<User> availableUsers =
chatService.getAvailableUsersForChat(currentUser);
        return mapper.writeValueAsString(availableUsers);
    }

    @PostMapping("/create-with-user/{otherUserId}")
    @ResponseBody
```

```
    public String createChatWithUser(@PathVariable Long otherUserId,
@RequestParam String chatName) throws JsonProcessingException {
        Authentication authentication =
SecurityContextHolder.getContext().getAuthentication();
        User currentUser = userService.loadUserByUsername(authentication.getName());
        Chat chat = chatService.createChatWithUser(currentUser, otherUserId,
chatName);
        return mapper.writeValueAsString(chat);
    }
}
```

Сервіс чату

```
package com.example.NewBuildingFinance.service.chat;

import com.example.NewBuildingFinance.dto.chat.ChatResponse;
import com.example.NewBuildingFinance.entities.auth.User;
import com.example.NewBuildingFinance.entities.chat.Chat;
import com.example.NewBuildingFinance.repository.auth.UserRepository;
import com.example.NewBuildingFinance.repository.chat.ChatRepository;
import com.fasterxml.jackson.databind.ObjectMapper;
import lombok.RequiredArgsConstructor;
import org.springframework.stereotype.Service;

import javax.crypto.KeyGenerator;
import javax.crypto.SecretKey;
import java.util.*;
import java.util.stream.Collectors;

@Service
@RequiredArgsConstructor
public class ChatService {

    private final ChatRepository chatRepository;
    private final UserRepository userRepository;
    private final EncryptionService encryptionService;
    private final ObjectMapper objectMapper;

    public List<ChatResponse> getChatsForUser(Long userId) {
        return chatRepository.findAllByUserId(userId).stream()
                .map(chat -> ChatResponse.builder()
```

```
                .id(chat.getId())
                .name(chat.getName())
                .build())
            .collect(Collectors.toList());
    }

    public List<User> getAvailableUsersForChat(User currentUser) {
        List<User> allChatUsers = userRepository.findUsersWithPermission("CHATS");
        List<Chat> existingChats =
chatRepository.findAllByUsersContaining(currentUser);

        Set<Long> existingChatUserIds = existingChats.stream()
            .flatMap(chat -> chat.getUsers().stream())
            .map(User::getId)
            .collect(Collectors.toSet());

        return allChatUsers.stream()
            .filter(user -> !user.getId().equals(currentUser.getId()))
            .filter(user -> !existingChatUserIds.contains(user.getId()))
            .collect(Collectors.toList());
    }

    public Chat createChatWithUser(User currentUser, Long otherUserId, String
chatName) {
        User otherUser = userRepository.findById(otherUserId)
            .orElseThrow(() -> new RuntimeException("User not found"));

        Chat chat = new Chat();
        chat.setName(chatName);
        chat.setUsers(new HashSet<>(Arrays.asList(currentUser, otherUser)));
```

```
        try {
            // Генеруємо AES-ключ для чату
            KeyGenerator keyGen = KeyGenerator.getInstance("AES");
            keyGen.init(256);
            SecretKey aesKey = keyGen.generateKey();

            // Шифруємо AES-ключ для кожного учасника
            Map<Long, String> encryptedKeys = new HashMap<>();
            String encryptedForCurrent =
encryptionService.encryptAESKeyForUser(aesKey, currentUser.getPublicKey());
            String encryptedForOther =
encryptionService.encryptAESKeyForUser(aesKey, otherUser.getPublicKey());
            encryptedKeys.put(currentUser.getId(), encryptedForCurrent);
            encryptedKeys.put(otherUser.getId(), encryptedForOther);

            String json = objectMapper.writeValueAsString(encryptedKeys);
            chat.setEncryptedAESKeys(json);
        } catch(Exception e) {
            throw new RuntimeException("Encryption error", e);
        }

        return chatRepository.save(chat);
    }
}
```

Сервіс шифрування

```
package com.example.NewBuildingFinance.service.chat;

import org.springframework.stereotype.Service;

import javax.crypto.Cipher;
import javax.crypto.SecretKey;
import javax.crypto.spec.OAEPParameterSpec;
import javax.crypto.spec.PSource;
import java.security.*;
import java.security.spec.MGF1ParameterSpec;
import java.security.spec.X509EncodedKeySpec;
import java.util.Base64;

@Service
public class EncryptionService {
    // Зашифрувати AES ключ для користувача (RSA)
    public String encryptAESKeyForUser(SecretKey aesKey, String
userPublicKeyBase64) throws Exception {
        byte[] publicKeyBytes = Base64.getDecoder().decode(userPublicKeyBase64);
        KeyFactory keyFactory = KeyFactory.getInstance("RSA");
        X509EncodedKeySpec keySpec = new X509EncodedKeySpec(publicKeyBytes);
        PublicKey publicKey = keyFactory.generatePublic(keySpec);

        // Явно задаємо параметри OAEP: hash = SHA-256, MGF1 з SHA-256,
порожній label
        OAEPParameterSpec oaepParams = new OAEPParameterSpec(
                "SHA-256",
                "MGF1",
```

```
            new MGF1ParameterSpec("SHA-256"),
            PSource.PSpecified.DEFAULT
        );
        Cipher cipher = Cipher.getInstance("RSA/ECB/OAEPWithSHA-
256AndMGF1Padding");
        cipher.init(Cipher.ENCRYPT_MODE, publicKey, oaepParams);
        byte[] encryptedKey = cipher.doFinal(aesKey.getEncoded());
        return Base64.getEncoder().encodeToString(encryptedKey);
    }

    public KeyPair generateKeyPair() {
        try {
            KeyPairGenerator keyGen = KeyPairGenerator.getInstance("RSA");
            keyGen.initialize(2048); // Генеруємо ключі 2048 біт
            return keyGen.generateKeyPair();
        } catch (NoSuchAlgorithmException e) {
            throw new RuntimeException("Error generating RSA key pair", e);
        }
    }
}
```

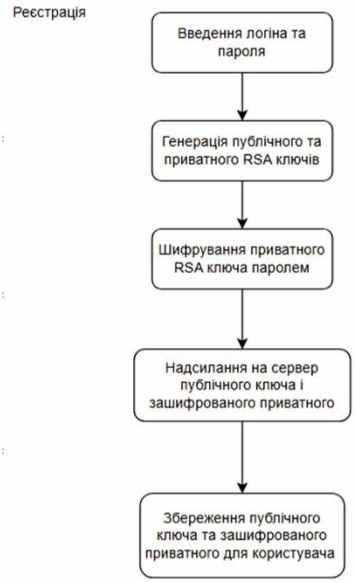
Сервіс повідомлень

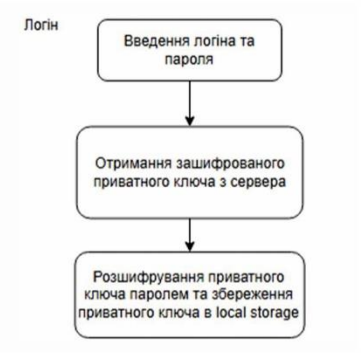
```
package com.example.NewBuildingFinance.service.chat;

import com.example.NewBuildingFinance.dto.chat.MessageRequest;
import com.example.NewBuildingFinance.dto.chat.MessageResponse;
import com.example.NewBuildingFinance.entities.auth.User;
import com.example.NewBuildingFinance.entities.chat.Chat;
import com.example.NewBuildingFinance.entities.chat.Message;
import com.example.NewBuildingFinance.repository.chat.ChatRepository;
import com.example.NewBuildingFinance.repository.chat.MessageRepository;
import com.example.NewBuildingFinance.service.auth.user.UserService;
import lombok.RequiredArgsConstructor;
import org.springframework.messaging.simp.SimpMessagingTemplate;
import org.springframework.security.core.Authentication;
import org.springframework.security.core.context.SecurityContextHolder;
import org.springframework.stereotype.Service;

import java.util.List;
import java.util.stream.Collectors;

@Service
@RequiredArgsConstructor
public class MessageService {

    private final MessageRepository messageRepository;
    private final ChatRepository chatRepository;
    private final SimpMessagingTemplate messagingTemplate;
    private final UserService userService;
```

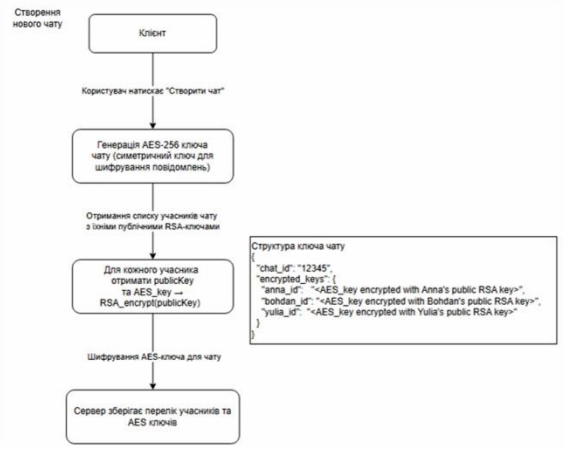
```
    public MessageResponse sendMessage(Long chatId, MessageRequest
messageRequest) {
        Chat chat = chatRepository.findById(chatId)
                .orElseThrow(() -> new RuntimeException("Chat not found"));

        Authentication authentication =
SecurityContextHolder.getContext().getAuthentication();
        User user = userService.loadUserByUsername(authentication.getName());

        if (!chat.getUsers().contains(user)) {
            throw new RuntimeException("User not in chat");
        }

        Message messageToSave = new Message();
        messageToSave.setChat(chat);
        messageToSave.setUser(user);
        messageToSave.setText(messageRequest.getEncryptedText());
        messageToSave.setIv(messageRequest.getIv());
        Message savedMessage = messageRepository.save(messageToSave);

        MessageResponse response = MessageResponse.builder()
                .userId(savedMessage.getUser().getId())
                .senderName(savedMessage.getUser().getName())
                .content(savedMessage.getText()) // Зашифрований текст
                .iv(savedMessage.getIv())
                .createdAt(savedMessage.getCreatedDateTime())
                .build();

        messagingTemplate.convertAndSend("/topic/chat/" + chatId, response);
```

```
        return response;
    }

    public List<MessageResponse> getMessages(Long chatId) {
        // Повертаємо зашифровані повідомлення, фронтенд сам розшифрує
        return messageRepository.findByChatIdOrderByCreatedDateTimeAsc(chatId)
                .stream()
                .map(message -> MessageResponse.builder()
                        .userId(message.getUser().getId())
                        .senderName(message.getUser().getName())
                        .content(message.getText())
                        .iv(message.getIv())
                        .createdAt(message.getCreatedDateTime())
                        .build()).collect(Collectors.toList());
    }
}
```

ДОДАТОК Б

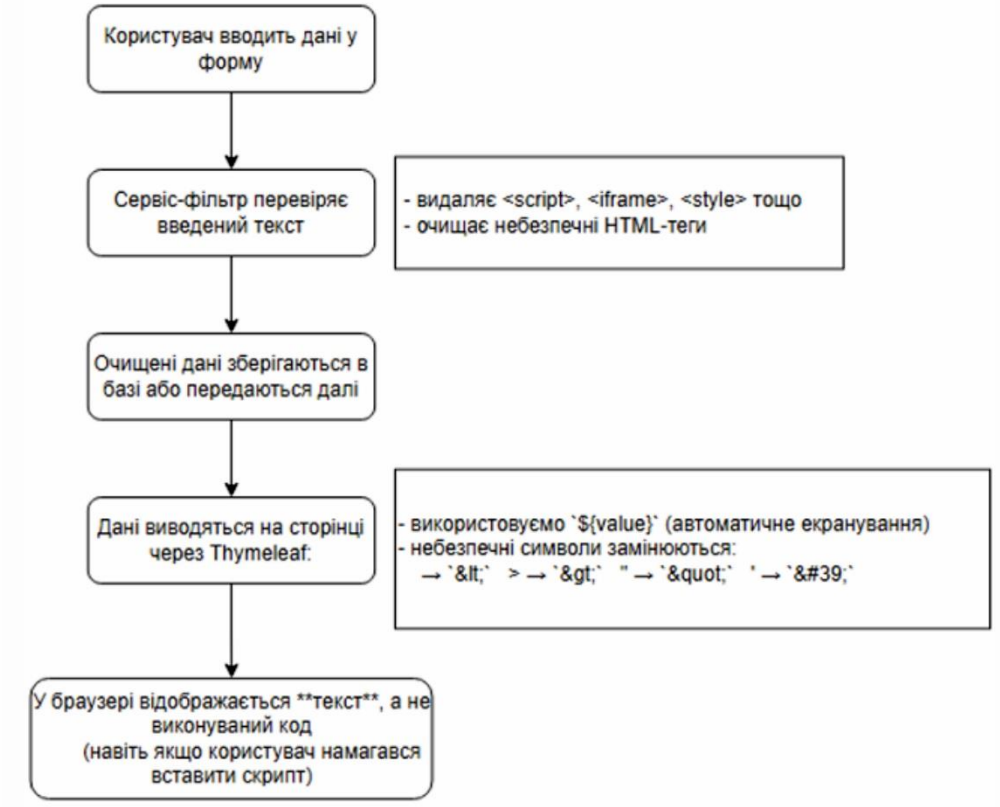






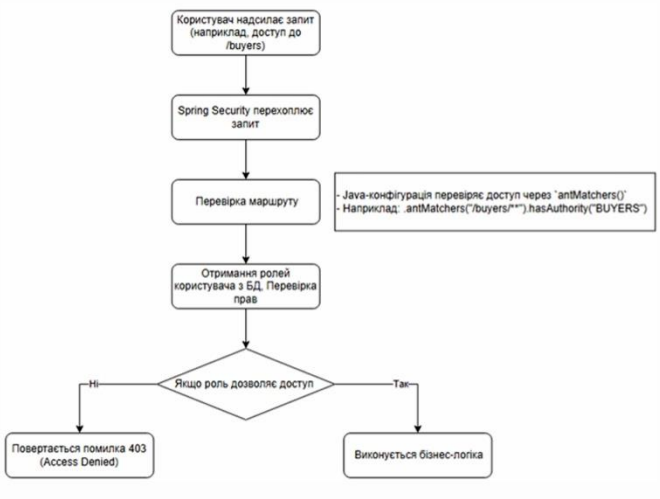
					КРБКБ.2101128.21.01.13 ПЗ			
Зм	Арк.	№ докум.	Підпис	Дата	Система захисту програмного комплексу фінансового документообігу з web-архітектурою Алгоритм роботи	Літ	Маса	Масштаб
Розроб.		Розгон Г.Д.				Н		
Перевір.		Касянчук М.М				Аркуш 1	Аркушів 3	
Т.контр.								
Н.контр.		Мостовий С.В				ХНУ, КБ-21-1		
Затверд.		Кльоц Ю.П.						

Фільтр скриптів

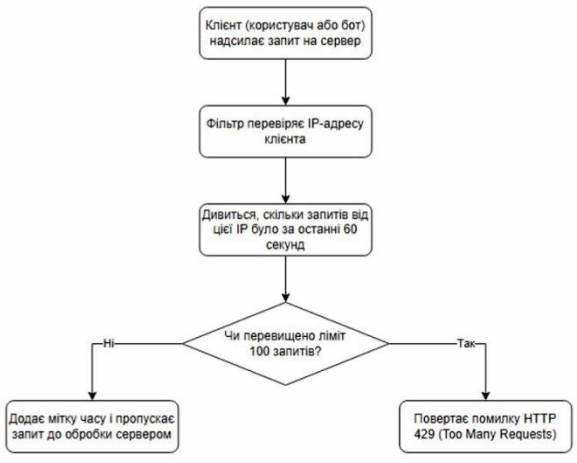


					КРБКБ.2101128.21.01.13 ПЗ	
Зм	Арк.	№ докум.	Підпис	Дата	Система захисту програмного комплексу фінансового документообігу з web-архітектурою Алгоритм роботи	Літ / Маса / Масштаб
Розроб.		Розгон І.Д.				Н
Перевір.		Касянчук М.М.				Аркуш 2 / Аркушів 3
Т.контр.						
Н.контр.		Мостовий С.В.				ХНУ, КБ-21-1
Затверд.		Кльоц Ю.П.				

Конфігурація безпеки з обмеженням маршрутів



Фільтр обмеження кількості запитів



Завідувачу кафедри кібербезпеки
к.т.н., доц. Кльоцу Ю.П.
Розгона Іллі Дмитровича
ПІБ здобувача вищої освіти

Студента ФІТ, 4 курсу, групи КБ-21-1

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, ознайомлена. Про використання програмно-технічних засобів для перевірки кваліфікаційних робіт здобувачів вищої освіти на плагіат оповіщена 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.

06.06.2025
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: Розгон Ілля Дмитрович

Співавтор:

Назва: Система захисту програмного комплексу фінансового документообігу з web-архітектурою

Науковий керівник:

Підрозділ: Кафедра кібербезпеки

Коефіцієнт подібності 1:1.8%

Коефіцієнт подібності 2:0%

Мікропробіли: 0

Заміна букв: 0

Інтервали: 0

Білі знаки: 0

Дата створення звіту: 2025-06-08 16:51:55.0

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:

09.06.2025р.

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: 9%**

	Document		Sum coincidence on the DB	
	Symbols	Lexemes	Symbols	Lexemes
ID: 244108 Title: Система захисту програмного комплексу фінансового документообігу з web-архітектурою Added in a DB: 2025-06-08 Authors: Розгон Ілля Дмитрович Heads: Касянчук М.М. Consultants: Opponents:	86740	1275	1048 (1%)	16 (1%)

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 захисту програмного комплексу фінансового документообігу з web-архітектурою

Автор: Розгон Ілля Дмитрович

Спеціальність: 125 – Кібербезпека

Освітня програма: Кібербезпека

Науковий керівник: Михайло КАСЯНЧУК, д-р технічних наук, професор

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 98%, 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 _____ Михайло КАСЯНЧУК

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 Розгон Ілля Дмитрович

Тема Система захисту програмного комплексу фінансового документообігу з web-архітектурою

Спеціальність 125 – Кібербезпека

Обсяг кваліфікаційної роботи освітньо-кваліфікаційного рівня «бакалавр»:

кількість листів креслень 3; кількість сторінок записки 67.

1. Короткий зміст роботи та прийнятих рішень У кваліфікаційній роботі була розроблена система захисту програмного комплексу фінансового документообігу з web-архітектурою. У межах дослідження проаналізовано актуальні методи захисту вебресурсів, протидії активним та пасивним атакам на інфраструктуру. Зокрема, в рамках роботи було розроблено стійкий end-to-end месенджер для внутрішньої комунікації. Спроектована система була імплементована на реальному корпоративному вебресурсі, проведена апробація та розроблено настанови з її експлуатації.

2. Висновок про відповідність кваліфікаційної роботи завданню У роботі в повній мірі виконано поставлені завдання, визначені темою та завданням на кваліфікаційну роботу, як у теоретичній, так і в практичній частинах. Спроектовано та впроваджено систему захисту комплексу фінансового документообігу

3. Характеристика виконання кожного розділу роботи, ступінь використання останніх досягнень науки і техніки і передових методів роботи: У вступі обґрунтовано актуальність теми, сформульовано мету та завдання, описано методи дослідження. У першому розділі було проаналізовано предметну область, досліджено існуючі системи протидії і державної та міжнародної нормативно - правової бази. У другому розділі визначено найбільш загрозливіші канали витоку інформації, побудовано модель загроз та модель загроз, а токож спроектовано систему запобігань інформаційним витокам. У третьому розділі розроблено імплементацію та апробацію системи запобігання інформаційним витокам. Протестовано імлементовану систему та оцінено використані ресурси.

4. Позитивні сторони Робота має значну практичну цінність, оскільки була спрямована на усунення вразливостей та захист реального вебдодатка, що функціонує в рамках корпоративної інфраструктури. За результатами роботи було суттєво підвищено захищеність даної системи, зокрема реалізовано захищений шифрований чат для внутрішніх комунікацій. Підхід до захисту був комплексний та систематичний

5. Негативні сторони роботи Можуть виникнути складності у ході розгортання та особливо підтримання запронованих заходів, що також може вимагати постійної присутності автора. Шифрований мессенджер не пройшов офіційну сертифікацію й потребує додаткового криптографічного дослідження. Також не повна увага була приділена засобам моніторингу та спостережності.

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

7. Відгук про роботу в цілому Кваліфікаційна робота заслуговує на позитивну оцінку, оскільки весь матеріал роботи є структурованим, чітким та послідовним. Усі розділи роботи логічно послідовні, 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 З урахуванням всіх позитивних та негативних сторін представленої кваліфікаційної роботи, можна зробити висновок, що вона заслуговує оцінки «відмінно».

РЕЦЕНЗЕНТ (прізвище, ім'я, по батькові, посада, місце роботи)
Пивовар Олег Сергійович,
Кандидат технічних наук, доцент, доцент кафедри телекомунікацій, медійних та інтелектуальних технологій

«9» червня 2025

Пивовар О.С.