Вінницький національний технічний університет

Факультет менеджменту та інформаційної безпеки

Кафедра менеджменту та безпеки інформаційних систем

БАКАЛАВРСЬКА КВАЛІФІКАЦІЙНА РОБОТА

на тему:

"Розробка захищеного web-додатку управління доступом до конфіденційних документів із використанням шифрування та механізмів захисту від XSS, CSRF та SQL ін'єкцій"

Виконав: студент 2(4) курсу, гр.1КІТС-21б
спеціальності 125– Кібербезпека
Освітня програма – Кібербезпека
інформаційних технологій та систем
(шифр і назва напряму підготовки, спеціальності)

Літвак Я. В.
(прізвище та ініціали)

Керівник: к.ф.-м.н., доц., доцент каф. МБІС
Шиян А. А.
(прізвище та ініціали)
«____» _____ 2025 р.

Рецензент: к.т.н., доц., доцент каф. ОТ
Крупельницький Л.В.
(прізвище та ініціали)
«____» _____ 2025 р.

Допущено до захисту
Голова секції УБ кафедри МБІС
д.т.н., проф. Яремчук Ю.Є.
"_____2025р.

Вінниця ВНТУ – 2025

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

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

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

З А В Д А Н Н Я
на бакалаврську кваліфікаційну роботу студенту
Літвак Яну Володимировичу
(прізвище, ім`я, по-батькові)

1. Тема роботи " Розробка захищеного web-додатку управління доступом до конфіденційних документів із використанням шифрування та механізмів захисту від XSS, CSRF та SQL ін'єкцій"

Керівник роботи к.ф.-м.н., доц., доцент каф. МБІС Шиян А. А.
(прізвище, ім`я, по-батькові, науковий ступінь, вчене звання)

затверджені наказом вищого навчального закладу від " 20 " березня 2025року № 80
2. Термін подання студентом роботи 28.05.2025 р.
3. Вихідні дані до роботи Методичні вказівки кафедри, нормативно-правові документи у сфері захисту інформації, сучасні технології веб-розробки та шифрування..

4. Зміст текстової частини:
1. Теоретичні основи захисту веб-додатків та конфіденційної інформації
2. Проектування захищеного веб-додатку управління доступом до конфіденційних документів із використанням шифрування та механізмів захисту від xss та sql ін'єкцій
3. Реалізація та тестування захищеного веб-додатку управління доступом до конфіденційних документів

5. Перелік ілюстративного матеріалу (з точним зазначенням обов`язкових креслень)
Презентація у форматі PowerPoint з ілюстраціями структури програми.

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Розділ I	Шиян А.А., к.ф.-м.н., доцент		
Розділ II	Шиян А.А., к.ф.-м.н., доцент		
Розділ III	Шиян А.А., к.ф.-м.н., доцент		

7. Дата видачі завдання 18 лютого 2025 р.

КАЛЕНДАРНИЙ ПЛАН

№	Назва та зміст етапу	Термін виконання		Примітка
		початок	закінчення	
1	Визначення напрямку бакалаврської роботи, формулювання теми	20.03.2025	23.03.2025	Виконано
2	Аналіз предметної області обраної теми	24.03.2025	13.04.2025	Виконано
3	Розробка алгоритму роботи	14.04.2025	27.04.2025	Виконано
4	Написання бакалаврської роботи на основі розробленої теми	28.04.2025	25.05.2025	Виконано
5	Передзахист бакалаврської кваліфікаційної роботи	26.05.2025	27.05.2025	Виконано
6	Виправлення, уточнення, корегування бакалаврської дипломної роботи	28.05.2025	02.06.2025	Виконано
7	Захист бакалаврської кваліфікаційної роботи	09.06.2025	10.06.2025	Виконано

Студент _____ **Літвак Я.В.**
(підпис) (прізвище та ініціали)

Керівник роботи _____ **Шиян А.А.**
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

У даній бакалаврській дипломній роботі розроблено захищений веб-додаток для управління доступом до конфіденційних документів. У роботі проаналізовано сучасні загрози інформаційній безпеці, зокрема атаки типу XSS, CSRF та SQL-ін’єкції, а також методи захисту від них. Запропоновано архітектуру веб-системи, яка забезпечує шифрування файлів, контроль доступу на основі ролей користувачів та надійне зберігання даних. У роботі наведено технічні рішення щодо реалізації засобів криптографічного захисту та забезпечення цілісності і конфіденційності інформації під час взаємодії користувача із системою.

ANNOTATION

In this bachelor's thesis, a secure web application for managing access to confidential documents has been developed. The work analyzes modern information security threats, including XSS, CSRF, and SQL injection attacks, as well as methods of protection against them. The architecture of the web system is proposed, providing file encryption, role-based access control, and reliable data storage. The thesis presents technical solutions for implementing cryptographic protection tools and ensuring the integrity and confidentiality of information during user interaction with the system.

ЗМІСТ

ВСТУП

Актуальність задачі. У сучасному цифровому світі інформація стає найціннішим ресурсом, а її захист – критичною вимогою. Різке зростання обсягів конфіденційних даних, що зберігаються та обробляються в інтернеті, створює значні виклики у сфері кібербезпеки. Особливо актуальним є питання захисту веб-додатків від поширених атак, таких як XSS, CSRF та SQL-ін’єкції, які можуть призвести до компрометації даних, несанкціонованого доступу або навіть втрати інформації.

З огляду на поширення використання хмарних сервісів та веб-платформ для зберігання і обміну конфіденційними документами, посилення заходів безпеки стає необхідністю для запобігання витокам, шахрайству та промисловому шпигунству. Захист інформації вимагає впровадження комплексних рішень, що поєднують надійну автентифікацію, розмежування доступу, шифрування даних та захист від атак, орієнтованих на вразливості веб-додатків.

Метою роботи є розробка захищеного веб-додатку для управління доступом до конфіденційних документів із застосуванням ефективних механізмів шифрування і протидії XSS, CSRF та SQL-ін’єкцій, що забезпечить надійний захист даних і контроль користувацьких прав.

Задачі дослідження:

– проаналізувати сучасні загрози безпеці веб-додатків;

– проаналізувати переваги та недоліки існуючих методів нейтралізації та протидії XSS, CSRF та SQL-ін’єкціям;

– спроектувати для веб-додатка систему захисту від XSS, CSRF та SQL-ін’єкцій шляхом впровадження відповідних механізмів безпеки;

– спроектувати архітектуру веб-додатка із механізмами аутентифікації та авторизації користувачів з розмежуванням прав доступу;

– розробити функціональні модулі розробленого захищеного веб-додатку управління доступом до конфіденційних документів;

– розробити інструкцію користування інтерфейсом захищеного веб-додатку

управління доступом до конфіденційних документів.

Об’єктом дослідження є процес управління доступом до конфіденційних документів у веб-додатках із використанням засобів захисту інформації.

Предметом дослідження є методи захисту веб-додатків від атак типу XSS, CSRF, SQL-ін’єкцій, а також засоби шифрування даних і реалізації контрольованого доступу до конфіденційних документів.

Новизна дослідження полягає у комплексному підході до побудови безпечного веб-додатку, що поєднує сучасні криптографічні методи, рольовий контроль доступу і захист від типових веб-атак, що значно підвищує рівень безпеки і надійності системи.

Практична цінність роботи полягає у створенні функціонального прототипу захищеного веб-додатку, який може бути використаний у державних установах, комерційних компаніях і приватних організаціях для безпечного зберігання і управління конфіденційними документами. Запропоновані рішення сприятимуть зниженню ризиків витоку інформації, підвищенню довіри користувачів і забезпеченню цілісності даних.

Таким чином, ця робота має важливе значення як для конкретних користувачів, так і для суспільства в цілому, сприяючи підвищенню рівня інформаційної безпеки та створенню більш надійного цифрового середовища.

1. ТЕОРЕТИЧНІ ОСНОВИ ЗАХИСТУ ВЕБ-ДОДАТКІВ ТА КОНФІДЕНЦІЙНОЇ ІНФОРМАЦІЇ

1.1 Поняття інформаційної безпеки та конфіденційності

Інформаційна безпека – це стан захищеності інформації та пов’язаних з нею ресурсів (таких як інформаційні системи, мережі, апаратне забезпечення та персонал) від внутрішніх і зовнішніх загроз. Вона спрямована на забезпечення основних принципів роботи з даними: конфіденційності, цілісності, доступності, автентичності та невідмовності [1].

Конфіденційність передбачає обмеження доступу до інформації лише для авторизованих осіб, запобігаючи її витоку або несанкціонованому розголошенню. Цілісність гарантує, що дані залишаються незміненими і точними, без несанкціонованих модифікацій. Доступність забезпечує постійну можливість отримання інформації та послуг для законних користувачів, навіть під час кібератак чи технічних збоїв [2].

Автентичність дозволяє перевіряти джерело інформації та ідентифікувати користувачів, запобігаючи спробам підробки або фальсифікації. Невідмовність забезпечує юридичну значимість дій, наприклад, коли сторона не може заперечувати факт відправлення або отримання даних [3].

Загрозами інформаційній безпеці можуть бути кібератаки (наприклад, віруси, шкідливе програмне забезпечення, фішинг), соціальна інженерія, апаратні та програмні збої, а також внутрішні ризики, пов’язані з діями персоналу. Для протидії цим загрозам використовуються технічні засоби (шифрування, фаєрволи, системи виявлення вторгнень), організаційні заходи (політики безпеки, навчання співробітників) та правові механізми (відповідні закони та стандарти) [4].

Таким чином, інформаційна безпека є комплексною дисципліною, яка поєднує технологічні, управлінські та правові аспекти для ефективного захисту даних і інформаційної інфраструктури. Конфіденційна інформація – це відомості, доступ до яких обмежений, оскільки їх розголошення може завдати шкоди окремій особі,

організації або державі. Вона може включати персональні дані, фінансові звіти, комерційні таємниці, внутрішню службову документацію, медичні записи та інші чутливі матеріали, які не призначені для публічного розповсюдження. У сучасному цифровому світі, де значна частина інформації зберігається та обробляється в електронному вигляді, потреба в захисті такої інформації стала особливо актуальною.

Захист конфіденційної інформації ґрунтується не лише на прагненні уникнути фінансових збитків або втрати конкурентних переваг, а й на законодавчих вимогах. Багато країн запровадили правові норми, які зобов'язують організації забезпечувати безпеку персональних і чутливих даних. Із розвитком Інтернету речей, хмарних технологій та мобільних пристроїв кількість каналів, через які інформація може бути скомпрометована, лише зростає. Нижче наведено кількісні характеристики інформаційної безпеки (табл. 1.1) [5].

Таблиця 1.1 - Кількісні характеристики інформаційної безпеки.

Показник	Значення	Джерело / Примітка
Середня вартість витоку даних	$4.45 млн	IBM, 2023
Час виявлення витоку	277 днів	IBM, 2023
Частка атак на веб-додатки	~26%	Verizon DBIR, 2023
Відсоток атак через людський фактор	понад 80%	Verizon DBIR, 2023
Рівень впровадження MFA в компаніях	55–60%	Microsoft Security Blog

У зв'язку з цим, розробка надійних систем захисту конфіденційної інформації є необхідною умовою функціонування будь-якої сучасної інформаційної системи.

Один із базових підходів до оцінки ризику базується на розрахунку (1.1) [6]:

$$R = P \times I \ , \qquad (1.1)$$

де

R (Risk) — рівень ризику;

P (Probability) — ймовірність виникнення загрози (значення в діапазоні від 0 до 1 або у відсотках);

I (Impact) — потенційний вплив або збитки у разі реалізації загрози (може виражатися у балах, грошовому еквіваленті або іншій шкалі).

Таким чином, ефективний захист конфіденційної інформації потребує цілісного, багаторівневого підходу, що поєднує технічні, правові, організаційні та освітні заходи. Це не лише питання впровадження технологічних засобів, а й стратегічне завдання формування культури відповідального поводження з даними, що має вирішальне значення для сталого розвитку цифрового суспільства. Класифікацію конфіденційної інформації наведено у табл. 1.2 [7].

Таблиця 1.2 - Класифікація конфіденційної інформації.

Тип інформації	Приклад	Ризики у разі витоку
Персональні дані	ПІБ, адреса, телефон, паспортні дані	Крадіжка особистості, шахрайство
Медичні дані	Діагнози, результати аналізів	Порушення приватності, дискримінація
Фінансова інформація	Номери банківських карток, звіти	Фінансові втрати, злом рахунків
Комерційна таємниця	Бізнес-плани, списки клієнтів	Втрата конкурентних переваг
Внутрішня службова	Документація, внутрішня переписка	Зниження довіри, репутаційні збитки

У цифрову епоху, коли інформація стала стратегічним ресурсом, особливого значення набувають законодавчі та етичні аспекти її захисту. Це вже не лише технічне питання, а й юридична та моральна відповідальність перед користувачами.

Законодавство регулює збирання, зберігання й обробку персональних даних. В Україні діє Закон «Про захист персональних даних», що визначає умови законного використання інформації [8]. У ЄС — GDPR, який встановлює суворі вимоги до прозорості, мети збору та відповідальності у разі порушень.

Крім того, існують міжнародні стандарти, як-от ISO/IEC 27001, що регламентують управління інформаційною безпекою, а також галузеві вимоги — HIPAA для медицини, PCI DSS для платіжних систем [9].

Етичні норми теж важливі: навіть законні дії можуть викликати недовіру, якщо порушено моральні принципи. Це може призвести до репутаційних втрат або втрати користувачів [10].

Таким чином, етичні та законодавчі вимоги взаємодоповнюють одне одного, формуючи цілісну систему цінностей у сфері захисту даних. Надійна безпека можлива лише за умови, коли технічні рішення поєднуються з відповідальністю та повагою до прав користувача.

1.2 Основні загрози безпеці веб-додатків

Основні загрози безпеці веб-додатків пов’язані з різноманітними вразливостями, що можуть бути використані зловмисниками для несанкціонованого доступу, маніпуляції даними або порушення нормального функціонування системи. Такі загрози виникають через недоліки в проектуванні, реалізації або експлуатації додатків, що відкривають шлях для атак [11].

SQL injection – це атака, коли хакер вставляє SQL-код в веб-форми, щоб отримати доступ до бази даних, або змінити, видалити або додати дані. SQL-ін’єкції можуть бути дуже шкідливими для веб-додатків, оскільки вони можуть дозволити хакеру виконувати будь-які дії з базою даних, навіть виконання команд на сервері (рис. 1.1) [12].

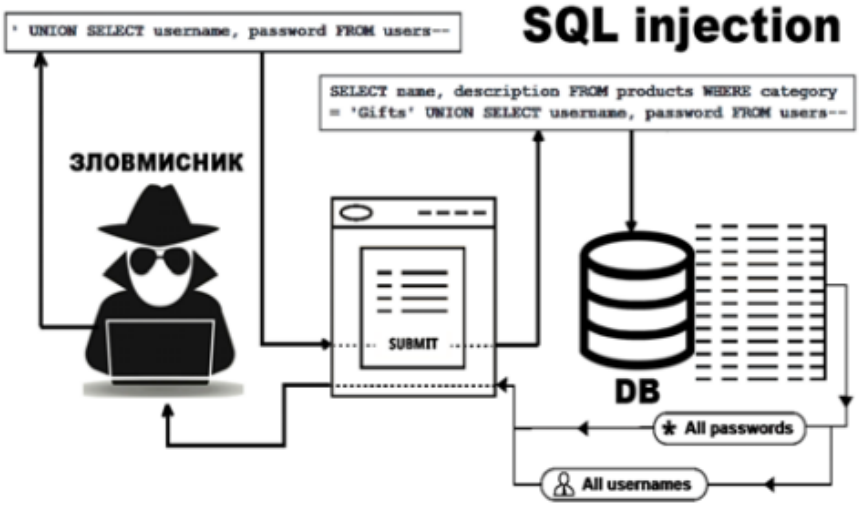


Рисунок 1.1 – Атака SQL injection [12]

Основним принципом атаки SQLi є використання недостатньо фільтрованого вводу користувача, що дозво-ляє зловмиснику вставляти в SQL-запит шкідливий код. Наприклад, якщо веб-додаток використовує запит типу "SELECT * FROM users WHERE username = 'input'", зловмисник може вводити значення, які містять шкідливий SQL-код, наприклад, " ' OR 1=1; --". В цьому випадку весь запит стає "SELECT * FROM users WHERE username = " OR 1=1; --'", що дозволяє зловмиснику отримати доступ до всіх записів в таблиці users [13].

SQL-ін’єкція може бути не лише інструментом викрадення даних, а й способом руйнування – зловмисник може змінити або повністю видалити таблиці, створити нові облікові записи адміністратора або навіть завантажити шкідливі скрипти для подальших атак. SQL-ін’єкція є критичною загрозою для веб-додатків, особливо тих, що працюють із важливою чи конфіденційною інформацією.

Cross-site scripting – це атака, при якій хакер вставляє скрипти в веб-сторінки, щоб отримати доступ до даних користувачів, які переглядають сторінки [14]. Ця атака може дозволити хакеру виконувати будь-які дії від імені користувача, включаючи крадіжку паролів та іншої конфіденційної інформації. Cross-site scripting є однією з найбільш поширених хакерських атак на веб-додатки. XSS полягає в тому, що зловмисники вбудовують скрипти в веб-сторінки, які відображаються для інших користувачів (рис. 1.2) [15].

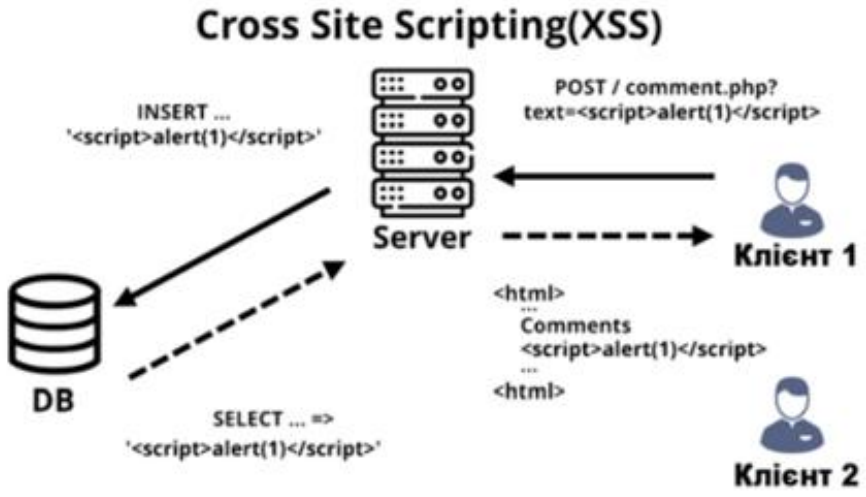


Рисунок 1.2 – Атака Cross-site scripting [15]

Наприклад, зловмисники можуть вбудувати скрипт, який виконує певну дію, наприклад, крадіжку кукі або паролів користувачів, коли інший користувач відкриває відповідну веб-сторінку. За допомогою XSS-атак зловмис-ники можуть також перенаправляти користувачів на інші веб-сторінки, що містять шкідливий вміст.

Основна ідея атаки полягає в тому, що зловмисник змушує авторизованого користувача перейти на підроблену сторінку, де за допомогою скрипту відправляється запит на сервер з підробленими параметрами. Якщо на сервері немає достатньої перевірки запитів, то він може виконати небажану дію без попередньої авторизації користувача, наприклад, відправити команду на видалення або зміну даних [16].

Cross-site request forgery – це атака, при якій хакер використовує веб-сторінки, щоб змусити користувачів виконувати дії, які вони не хочуть виконувати [17]. Наприклад, хакер може відправити електронний лист з посилан-ням, що виконує певну дію, таку як відправка грошей з банківського рахунку, якщо користувач натисне на поси-лання (рис. 1.3) [18].

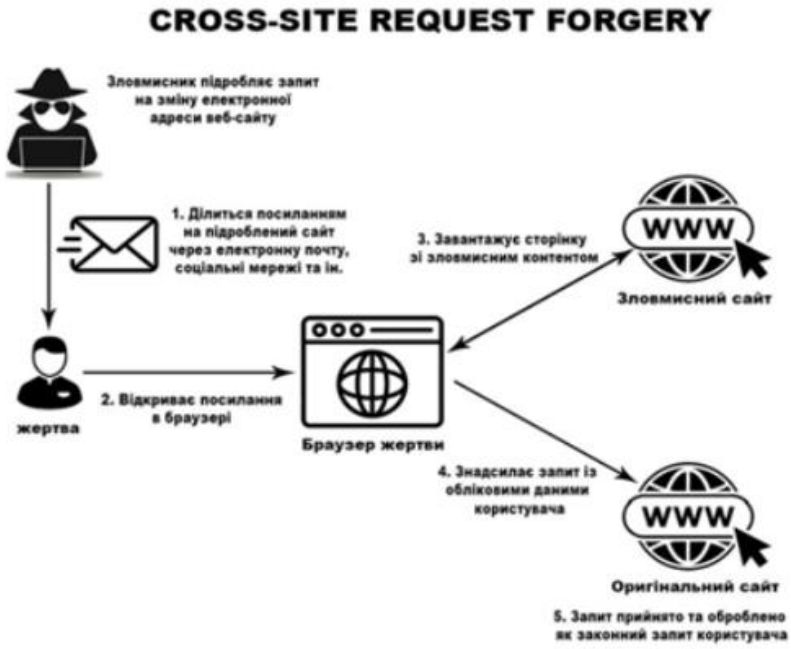


Рисунок 1.3 – Атака Cross-site request forgery [18]

Cross-site request forgery є небезпечною, але малопомітною атакою, яка

базується на неправильному використанні довіри між користувачем і веб-додатком. Її особливість полягає в тому, що вона реалізується не за рахунок вразливості в самій системі, а завдяки відсутності захисту від небажаних зовнішніх запитів.

Один з основних прикладів атаки SSRF полягає в тому, що зловмисник використовує вразливість на веб-сайті для відправки запиту до внутрішньої мережі організації [19]. Наприклад, зловмисник може використовувати адресу IP внутрішньої мережі, яку він дізнався під час перегляду заголовків відповіді попереднього запиту до сервера. Після цього зловмисник може надіслати запит на внутрішній сервер з використанням цієї адреси IP, що дозволить йому виконати дії, такі як отримання конфіденційних даних або знищення внутрішніх ресурсів.

Server-side request forgery – це атака, коли хакер використовує вразливість веб-додатку, щоб змусити сервер виконувати запити до інших ресурсів в мережі, таких як бази даних, файлові системи або інші веб-сервери [20]. Наприклад, нападник може використовувати сервер, щоб скопіювати або видалити дані зі стороннього сервера або виконати шкідливий код на іншому сервері (рис. 1.4) [21].

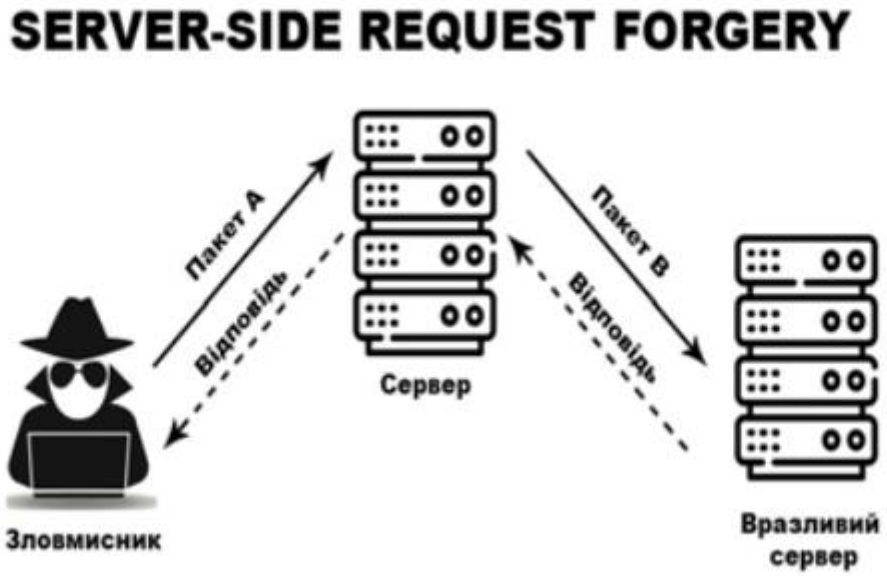


Рисунок 1.4 – Атака Server-side request forgery [21]

Одним з методів захисту від атаки SSRF є перевірка вхідних даних та параметрів запитів на вміст небезпечних URL-адрес. Також може бути використана

блокування доступу до внутрішньої мережі з зовнішнього сервера.

Файлові атаки – це атаки, коли хакер використовує вразливість веб-додатку, щоб завантажити зловмисний файл на сервер або змінити існуючі файли [22]. Ці атаки можуть бути особливо небезпечними, якщо зловмисний файл містить шкідливий код, Одним з прикладів файлової атаки, яка використовує вразливості в програмному забезпеченні, є атака на фай-лову систему сервера.

Витік даних – це ситуація, коли конфіденційна, персональна або службова інформація стає доступною для осіб, які не мають на це правас [23]. Такі інциденти можуть виникати як у результаті хакерських атак, так і через помилки в конфігурації або програмній логіці.

DoS (Denial of Service) – це кібератака направлена на систему з завданням вивести її з ладу, спричинити відмову [24]. Мається на увазі створення умов, при яких звичайні користувачі можуть отримати відмову в отриманні доступу до конкретних сервісів, або ж цей доступ стає ускладненим. DDoS – розподілена атака на відмову в обслуговуванні, це атаки DoS що виконуються водночас з багатьох пристроїв, це можуть бути боти. Метою таких атак є руйнування системи таким чином, щоб її служби стали недоступні [25].

Хоча існує багато різних варіантів і типів атак, по суті зловмисник використовує вразливості, щоб взяти під свій контроль кілька машин. Різниця між атакою DoS і DDoS полягає в тому, що це завдання виконує кілька машин.

1.3 Методи та засоби захисту веб-додатків

Сучасні веб-додатки взаємодіють із конфіденційною інформацією користувачів, обробляють персональні, фінансові або службові дані, що робить їх привабливою ціллю для зловмисників. Тому захист таких систем повинен реалізовуватись на декількох рівнях, із застосуванням як організаційних, так і технічних заходів.

Для ефективного захисту від SQL-ін’єкцій необхідно застосовувати комплексний підхід, що включає кілька ключових методів. Перш за все, використання підготовлених запитів (Prepared Statements) є одним із

найнадійніших способів запобігання SQL-ін’єкціям. Цей метод дозволяє розділити логіку запиту і дані, які надходять від користувача, що виключає можливість виконання шкідливого коду, введеного зловмисником [26]. Запити компілюються заздалегідь, а дані передаються як параметри, тому будь-який введений текст розглядається саме як дані, а не як частина команди SQL. Ще одним важливим заходом є динамічне формування SQL-запитів із використанням змінних замість жорстко закодованих значень (рис. 1.5) [27].

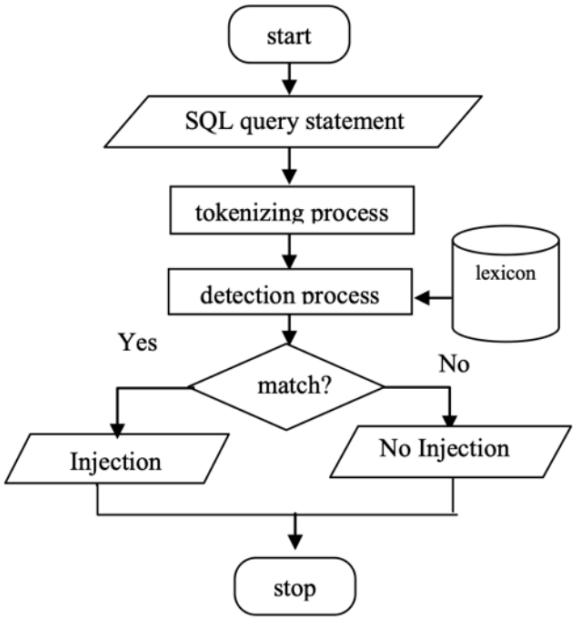


Рисунок 1.5 – Блок-схема роботи SQL-запитів [27]

Фільтрація та валідація введених користувачем даних також є невід’ємною частиною захисту від SQL-ін’єкцій. Необхідно ретельно перевіряти всі вхідні дані на предмет наявності потенційно шкідливих символів і конструкцій, що можуть бути інтерпретовані як SQL-код. При цьому важливо застосовувати не лише просте видалення або заміну символів, а й комплексну очищувальну логіку, що враховує контекст використання даних у запитах.

Обмеження прав доступу користувачів і сервісів до бази даних також значно підвищує рівень безпеки. Якщо у користувача немає достатніх прав для виконання певних операцій, навіть у разі успішної SQL-ін’єкції шкода буде обмежена.

Загалом, захист від SQL-ін’єкцій є критично важливою складовою безпеки будь-якого веб-додатку, особливо якщо він працює з конфіденційною

інформацією. Ігнорування цих методів може призвести до серйозних наслідків, таких як витік особистих даних, порушення цілісності інформації, а також повна компрометація бази даних і подальших сервісів. Тому забезпечення правильного застосування описаних методів має стати пріоритетом при розробці та підтримці веб-додатків.

Для захисту від атак типу Cross-site Scripting (XSS) застосовують кілька ефективних методів, які забезпечують безпеку введених користувачем даних і знижують ризик виконання шкідливого коду в браузері [28]. Перш за все, важливо здійснювати валідацію даних на вході, перевіряючи введений користувачем текст на наявність потенційно небезпечних символів і забезпечуючи його безпечність перед передачею на сервер. Це дозволяє відфільтрувати або блокувати небажані елементи ще на ранньому етапі обробки (рис. 1.6) [29].

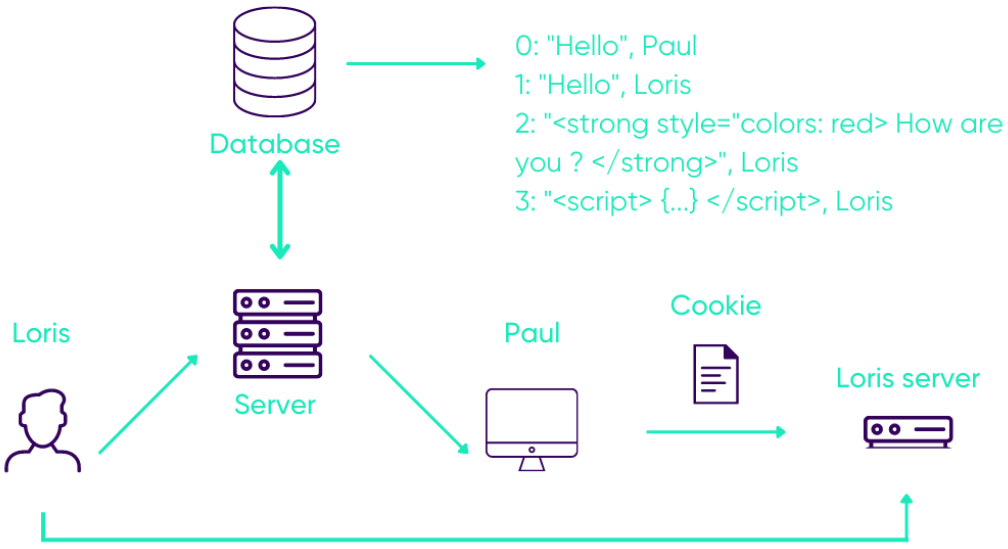


Рисунок 1.6 – Схема роботи XSS атаки [29]

Наступним кроком є екранування символів, яке полягає в заміні потенційно шкідливих символів на їх безпечні еквіваленти. Завдяки цьому браузер не інтерпретує такі символи як код, а відображає їх у вигляді тексту, що запобігає виконанню скриптів і впливу на сторінку.

Додатково широко використовується механізм Content Security Policy (CSP) – це політика безпеки, яка дозволяє обмежити джерела контенту, які можуть бути

завантажені на веб-сторінку, і заборонити виконання скриптів з ненадійних або сторонніх джерел. CSP значно підвищує рівень захисту, ускладнюючи зловмисникам реалізацію атак XSS. Ще одним важливим засобом є використання HTTP-only кукі, які забороняють доступ до кукі через JavaScript. Це обмеження знижує ризик крадіжки кукі з сесійними даними в разі виконання шкідливого скрипта, що є частою метою XSS-атак.

Для захисту від атак типу Cross-site Request Forgery (CSRF) рекомендується застосовувати кілька важливих заходів, які дозволяють убезпечити веб-додаток від несанкціонованих дій, що виконуються від імені користувача. Одним із найефективніших способів є використання вбудованих механізмів захисту від CSRF, які часто надають сучасні веб-фреймворки. Ці механізми автоматично генерують і перевіряють спеціальні токени, що забезпечують додатковий рівень захисту (рис 1.7) [30].

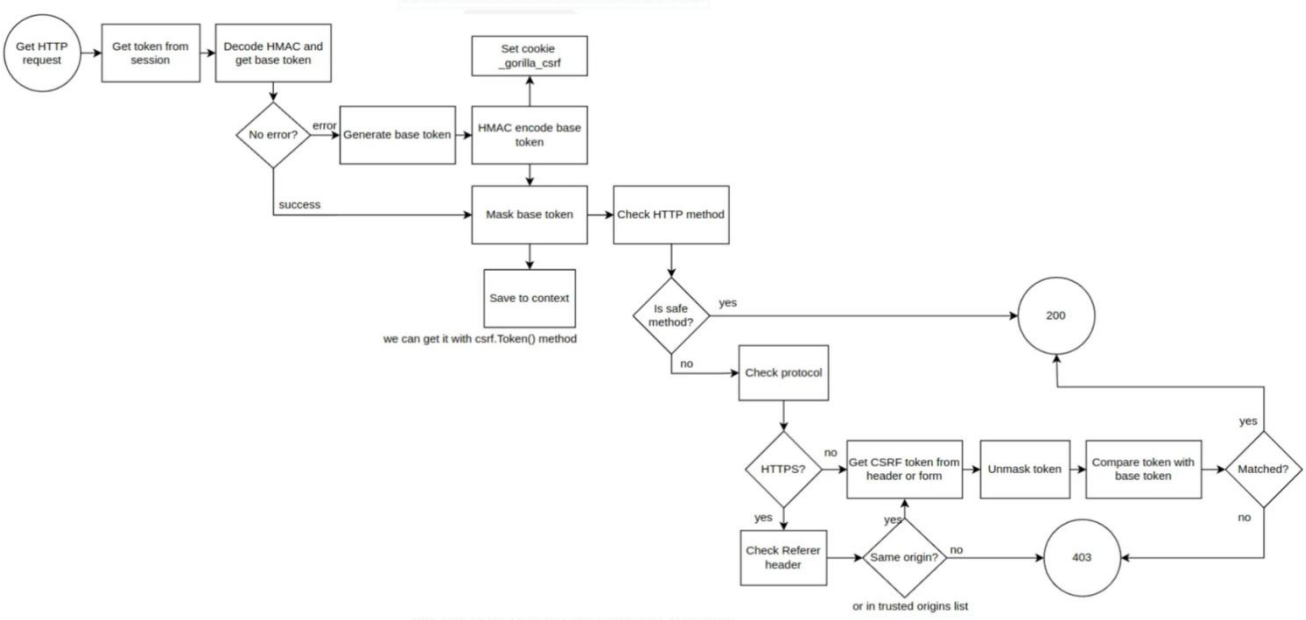


Рисунок 1.7 – Блок-схема роботи CSRF middleware [30]

Ключовим елементом захисту є застосування токенів випадкової генерації, які прив’язуються до конкретного користувача або сесії. Такі токени вставляються у форми та запити, які виконують дії, що можуть змінити стан додатку, наприклад, оновлення або видалення даних. Сервер при отриманні запиту перевіряє наявність і відповідність токена, що унеможливлює виконання запиту з іншого, несанкціонованого джерела. Додатково для перевірки легітимності запитів

застосовують аналіз HTTP-заголовка Referer, що дозволяє визначити, чи походить запит зі сторінок власного веб-сайту. Проте цей метод не є абсолютно надійним, оскільки заголовок може бути змінений або зовсім відсутній у деяких запитах, тому його краще використовувати як допоміжний засіб. Важливо також дотримуватись рекомендацій щодо методів HTTP-запитів: не рекомендується використовувати метод GET для операцій, які змінюють стан додатку, таких як видалення або редагування даних. Замість цього слід застосовувати методи POST або PUT, які дозволяють включати CSRF-токени в тіло запиту, що підвищує безпеку.

Крім того, не слід зберігати конфіденційну інформацію у кукі або передавати її через GET-запити, оскільки такі дані можуть бути підмінені або викрадені під час атак CSRF. Замість цього рекомендується передавати чутливу інформацію через безпечні POST-запити, що знижує ризик компрометації.

Для захисту від атак типу Server-Side Request Forgery (SSRF) необхідно застосовувати низку комплексних заходів, що знижують ризики несанкціонованого доступу до внутрішніх ресурсів системи [31]. Одним із найважливіших кроків є обмеження доступу до внутрішніх ресурсів через налаштування файрвола та мережевих політик. Це дозволяє заборонити прямі запити до баз даних, сервісів та інших внутрішніх систем із зовнішніх джерел, які потенційно можуть бути використані зловмисниками.

Важливим є також налаштування прав доступу на серверах і відповідних сервісах, що дозволяє обмежити можливість здійснення запитів на внутрішні ресурси лише авторизованим і довіреним компонентам системи.

Виявлення DoS/DDoS-атак передбачає відокремлення легітимного трафіку від шкідливого, що потребує швидкого та точного аналізу. Спочатку збираються дані про мережеву активність, які потім аналізуються для виявлення підозрілих запитів. Стандартні засоби захисту, як-от брандмаузери чи системи запобігання проникненню, ефективні проти окремих атак, але часто не справляються з масштабними DDoS-атаками, що імітують звичайний трафік.

1.4 Криптографічні методи захисту даних

У сучасному цифровому світі криптографія є одним із найнадійніших засобів захисту інформації. Вона базується на математичних алгоритмах і використовується для забезпечення конфіденційності, цілісності й автентичності даних. Саме криптографічні методи та побудовані на їх основі протоколи безпечного обміну інформацією, а також цифрові підписи, становлять основу сучасної інформаційної безпеки.

Симетричні алгоритми шифрування є важливою складовою криптографічних систем і поділяються на блочні та потокові. Потокові алгоритми працюють з безперервними потоками даних, генеруючи послідовність псевдовипадкових бітів, які за допомогою операції XOR поєднуються з інформаційним потоком. Такий підхід дозволяє шифрувати дані в режимі реального часу, що критично важливо для онлайн-відео, аудіо та інших застосувань, де важлива мінімальна затримка. Потокові алгоритми особливо ефективні в умовах, коли обробка даних відбувається на рівні окремих символів або бітів, а не цілих блоків (рис. 1.8) [32].

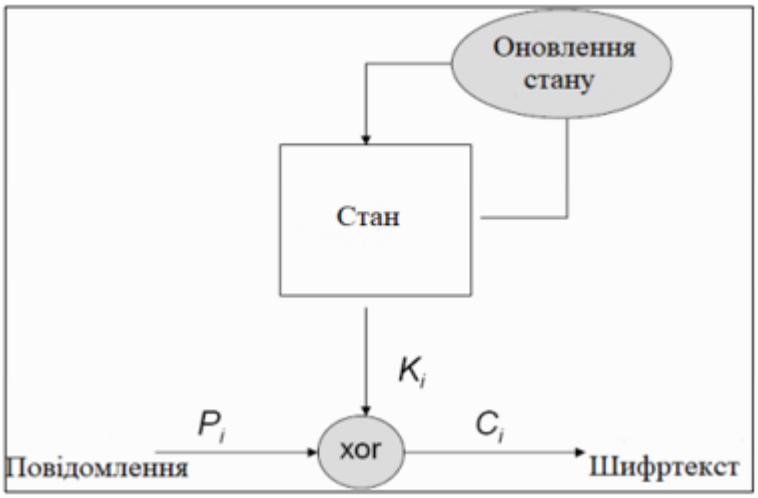


Рисунок 1.8 – Схема роботи потокових алгоритмів шифрування [32]

Блочні алгоритми шифрування працюють із фіксованими блоками даних, зазвичай розміром від кількох до кількох десятків байт. Вхідні дані діляться на такі блоки, кожен із яких шифрується окремо за допомогою одного секретного ключа. Розмір зашифрованого блоку співпадає з розміром вхідного, що дозволяє ретельно контролювати процес шифрування та забезпечує високу криптографічну стійкість.

Одним із найвідоміших блочних алгоритмів є DES (Data Encryption Standard), прийнятий у США як стандарт у 1977 році [33].

Він працює з блоками по 64 біти і використовує 56-бітний секретний ключ. DES базується на мережі Фейстеля, що складається з 16 раундів перетворень, кожен із яких застосовує підключ довжиною 48 біт, сформований із основного ключа. Особливістю DES є те, що навіть мінімальна зміна вхідних даних викликає радикальну зміну зашифрованого блоку, що підвищує його стійкість до атак.

Головною перевагою даного алгоритму є те, що шифрування та розшифрування відбувається за однаковим алгоритмом, що є дуже важливим параметром при апаратній реалізації.

Асиметричне шифрування, представлене такими алгоритмами, як RSA та ElGamal, використовує два ключі: відкритий для шифрування інформації і секретний для її розшифрування [34].

Це дає змогу уникнути передачі секретного ключа через захищені канали, оскільки розшифрувати дані може лише власник секретного ключа. Серед недоліків цього методу — складність внесення змін в алгоритм і більші вимоги до обчислювальних ресурсів у порівнянні з симетричним шифруванням.

Зважаючи на це, на практиці для забезпечення безпеки використовують ключі RSA довжиною 2048 біт і більше, що робить факторизацію числа надзвичайно складною навіть для сучасних потужностей. Це забезпечує прийнятний рівень захисту і дозволяє ефективно застосовувати RSA в реальних криптографічних системах.

Хешування — це процес перетворення будь-яких вхідних даних у послідовність символів фіксованої довжини, яка називається хешем або хеш-кодом [35]. Хешування не є традиційним шифруванням, проте відіграє важливу роль у захисті даних, зокрема для безпечного зберігання паролів. Для цього використовують спеціальні алгоритми, які забезпечують унікальне представлення будь-якої інформації. Хеші завжди мають фіксовану довжину незалежно від розміру вхідних даних. Наприклад, алгоритм SHA-256 створює хеш довжиною 256 біт (32 байти), навіть якщо вхідні дані — це один символ або великий документ.

Хешування з використанням солі (salted hashing) – це метод додавання випадкових даних (солі) до паролів перед їх хешуванням, щоб ускладнити атаки, такі як brute-force (рис 1.9) [36].

Username	String to be hashed	Hashed value = SHA256
user1	password123	EF92B778BAFE771E89245B89ECBC08A44A4E166C06659911881F383D4473E94F
user2	password123	EF92B778BAFE771E89245B89ECBC08A44A4E166C06659911881F383D4473E94F

Username	Salt value	String to be hashed	Hashed value = SHA256 (Password + Salt value)
user1	D; %yL9TS:5PalS/d	password123D; %yL9TS:5PalS/d	9C9B913EB1B6254F4737CE947EFD16F16E916F9D6EE5C1102A2002E48D4C88BD
user2	)<,-<U(jLezy4j>*	password123)<,-<U(jLezy4j>*	6058B4EB46BD6487298B59440EC8E70EAE482239FF2B4E7CA69950DFBD5532F2

Рисунок 1.9 – Приклад хешування паролів [36]

Керування ключами та їх зберігання — це критично важливі аспекти криптографічного захисту, які забезпечують надійність шифрування та цілісність інформації. До основних етапів керування ключами належать їхнє створення, зберігання, розповсюдження, використання та знищення.

Навіть найнадійніші алгоритми шифрування можуть стати вразливими без ефективного керування ключами. Саме ключі є основою всієї системи криптографічного захисту, тому їх безпечне створення, зберігання та обробка — одна з головних умов забезпечення інформаційної безпеки.

1.5 Огляд існуючих рішень та систем управління документами

Популярні платформи для зберігання даних і управління файлами, такі як Google Drive, Dropbox і корпоративні системи управління документами (DMS), мають суттєві відмінності як у функціональності, так і в підходах до безпеки, що визначає їх застосування в різних сферах .

Google Drive є одним із найпоширеніших сервісів для зберігання файлів у хмарі, який активно використовується як приватними користувачами, так і бізнесом. Він пропонує простий та інтуїтивно зрозумілий інтерфейс, гнучке управління файлами і папками, а також інтеграцію з іншими сервісами Google, такими як Gmail, Google

Docs, Sheets і Slides.

Dropbox також є популярним хмарним сервісом, який відомий своєю простотою і кросплатформністю. Він підтримує широкий спектр операційних систем, що робить його зручним для користувачів із різними пристроями.

Корпоративні системи управління документами (DMS) — це спеціалізовані платформи, створені для організацій з високими вимогами до безпеки, контролю доступу та автоматизації бізнес-процесів. Серед найвідоміших прикладів таких систем — Microsoft SharePoint, Alfresco, IBM FileNet тощо [37].

Вибір платформи для зберігання та управління файлами залежить від масштабів і потреб користувача. Для індивідуального використання або малого бізнесу доцільно використовувати хмарні сервіси, такі як *Google Drive* чи *Dropbox*, які пропонують простий інтерфейс і базовий захист. Натомість середнім і великим організаціям, що мають підвищені вимоги до безпеки й відповідності стандартам, більше підходять корпоративні DMS із розширеною функціональністю.

Таким чином, хоча всі платформи виконують спільну функцію — зберігання даних, рівень їхньої безпеки, контроль доступу та можливості інтеграції визначають доцільність використання в різних умовах [38].

Підходи до захисту конфіденційних документів у Google Drive, Dropbox і корпоративних DMS різняться залежно від архітектури, призначення платформи та рівня забезпечення безпеки.

Google Drive реалізує багаторівневу систему захисту даних. Під час передачі інформація шифрується за допомогою протоколу TLS, а при зберіганні — алгоритмом AES-256.

Отже, підходи до захисту конфіденційних документів у цих платформах відрізняються за рівнем деталізації і можливостями контролю: хмарні сервіси, як Google Drive і Dropbox, пропонують надійний базовий захист і зручність, тоді як корпоративні DMS забезпечують глибоке управління безпекою, що необхідне для складних бізнес-середовищ.

По-друге, традиційні рішення часто недостатньо захищені від сучасних кіберзагроз, таких як атаки типу XSS, CSRF або SQL-ін’єкції, які можуть призвести

до компрометації даних. Розроблена система інтегрує сучасні механізми захисту, що мінімізують ризики таких атак, підвищуючи надійність збереження і передачі інформації.

По-третє, в багатьох випадках відсутній надійний аудит дій користувачів з документами, що ускладнює виявлення та розслідування інцидентів безпеки. Запропонована система забезпечує повний лог діяльності, що дозволяє відслідковувати всі операції, пов’язані з доступом до конфіденційних даних [39].

Таким чином, розроблена система спрямована на усунення вразливостей і обмежень існуючих платформ, підвищуючи безпеку, контроль і прозорість у роботі з конфіденційними документами.

1.6 Висновки та постановка задач

У результаті аналізу сучасних загроз інформаційній безпеці та існуючих методів захисту веб-додатків було визначено основні напрямки для розробки захищеного веб-додатку управління доступом до конфіденційних документів. Найбільшу увагу слід приділити захисту від XSS, CSRF та SQL-ін’єкцій, а також впровадженню надійних криптографічних методів для забезпечення конфіденційності даних.

Постановка задач включає:

– спроєктувати для веб-додатка систему захисту від XSS, CSRF та SQL-ін’єкцій шляхом впровадження відповідних механізмів безпеки;

– спроєктувати архітектуру веб-додатка із механізмами аутентифікації та авторизації користувачів з розмежуванням прав доступу;

– розробити функціональні модулі захищеного веб-додатку управління доступом до конфіденційних документів;

– розробити інструкцію користування інтерфейсом захищеного веб-додатку управління доступом до конфіденційних документів.

2 ПРОЕКТУВАННЯ ЗАХИЩЕНОГО ВЕБ-ДОДАТКУ УПРАВЛІННЯ ДОСТУПОМ ДО КОНФІДЕНЦІЙНИХ ДОКУМЕНТІВ ІЗ ВИКОРИСТАННЯМ ШИФРУВАННЯ ТА МЕХАНІЗМІВ ЗАХИСТУ ВІД XSS, CSRF ТА SQL ІН'ЄКЦІЙ

Основне завдання полягає у розробці захищеного веб-додатку для управління доступом до конфіденційних документів. Система має забезпечити можливість безпечного завантаження, зберігання та обміну файлами між користувачами з різними рівнями доступу, а також реалізувати механізми захисту даних від несанкціонованого доступу, атак типу XSS, CSRF та SQL-ін'єкцій.

Веб-додаток повинен включати авторизацію з перевіркою облікових даних, розмежування прав доступу до документів відповідно до ролей користувачів (адміністратор, звичайний користувач та суперкористувач), зручний інтерфейс для перегляду та керування файлами, а також функціонал шифрування для збереження конфіденційності даних при зберіганні та передачі.

2.1 Постановка задач та проєктування підходу захисту веб-додатка із управління доступом до конфіденційних документів із використанням шифрування та механізмів захисту від XSS, CSRF та SQL-ін'єкцій

У сучасному світі обсяги конфіденційної інформації в бізнесі, державному секторі та наукових установах постійно зростають: це договори, фінансові звіти, персональні дані клієнтів, патентні розробки, медичні картки тощо. Зберігання та обмін такими документами без належного захисту створює серйозні ризики – від фінансових втрат і правових претензій до пошкодження репутації організації. Крім того, поширення віддаленої праці й хмарних сервісів підвищує потребу в єдиній, централізованій платформі для безпечного обміну файлами між користувачами з різними ролями й рівнями довіри. Відсутність чіткого контролю доступу веде до «сліпих зон», коли документи копіюються на локальні пристрої або передаються через незахищені канали, що робить їх вразливими до віддаленого викрадення або

неправомірного використання.

Для вирішення зазначених проблем було розроблено веб-додаток, який забезпечує багаторівневий контроль доступу, централізоване управління правами та криптографічний захист даних. Система базується на принципі "глибокого захисту", де кожен рівень забезпечує окремий бар'єр для потенційних загроз. Перший рівень захисту формується через систему автентифікації та авторизації користувачів, другий рівень реалізується через шифрування файлів, а третій рівень забезпечує захист від спеціалізованих веб-атак.

У розробленій системі реалізовано три основні режими роботи з документами: завантаження й зберігання файлів у зашифрованому вигляді, надання та відкликання прав доступу відповідно до ролей користувачів (адміністратор, звичайний користувач та суперкористувач), де суперкористувач має повний доступ і належить власнику розроблюваного програмного рішення. Такий підхід мінімізує людський фактор у процесі управління безпекою, оскільки всі політики доступу централізовано налаштовуються адміністратором і виконуються автоматично.

Система контролю доступу базується на ролевій моделі, яка була реалізована у веб-додатку. Суперкористувач має необмежений доступ до всіх документів та адміністративних функцій системи, що дозволяє йому керувати як документами, так і користувацькими акаунтами. Адміністратори мають доступ до документів з рівнями "admin", "user" та "public", але не можуть переглядати документи суперкористувача або керувати акаунтами з вищими привілеями. Звичайні автентифіковані користувачі можуть працювати лише з документами рівнів "user" та "public", тоді як неавторизовані відвідувачі мають доступ виключно до публічних матеріалів. Така ієрархія забезпечує принцип мінімальних привілеїв, коли кожен користувач отримує лише той рівень доступу, який необхідний для виконання його функціональних обов'язків [40].

Вибір алгоритму PBKDF2 обґрунтовується кількома ключовими факторами. По-перше, його адаптивна природа дозволяє налаштовувати складність обчислення через параметр c відповідно до зростання потужності обчислювальної техніки. Збільшення кількості ітерацій пропорційно підвищує час, необхідний для

обчислення хешу, що робить атаки методом грубої сили економічно недоцільними. По-друге, PBKDF2 є стійким до атак за допомогою lookup tables та rainbow tables завдяки використанню унікальної солі S для кожного пароля. По-третє, алгоритм має вбудований механізм захисту від timing attacks через використання функції HMAC, яка має константний час виконання. По-четверте, PBKDF2 є міжнародним стандартом RFC 2898 і рекомендується провідними організаціями у сфері кібербезпеки, включаючи NIST [41]. Це унеможливлює відновлення пароля навіть у разі доступу до бази даних. Для запобігання автоматизованим атакам типу brute-force введено обмеження на кількість спроб входу за певний період.

Вибір AES-256 обґрунтовується комплексом технічних та безпекових характеристик. По-перше, AES має статус міжнародного стандарту шифрування FIPS 197, який рекомендується провідними криптографічними організаціями включаючи NIST та NSA. По-друге, довжина ключа 256 біт забезпечує криптографічну стійкість приблизно 2^{128} операцій для атак методом повного перебору, що робить зламування практично неможливим. Алгоритм пройшов інтенсивне тестування криптографічною спільнотою протягом більш ніж 20 років і не має відомих практичних вразливостей [43].

Процес криптографічного захисту документів здійснюється автоматично при завантаженні файлу в систему за математичним алгоритмом, який формалізується системою рівнянь:

$$Kf = CSPRNG(256), IV = CSPRNG(128), Cf = E_{\{Kf\}(Pf,IV)}^{\{AES-256\}}, \quad (2.1)$$

$$Ck = E_{\{Km\}(Kf)}^{\{AES-256\}}, \; So(H(fn), Cf, Ck, IV), \quad (2.2)$$

де $Kf \in \{0,1\}^{256}$ — криптографічний ключ файлу, генерований криптографічно стійким генератором псевдовипадкових чисел,

$IV \in \{0,1\}^{128}$ — вектор ініціалізації,

Pf — оригінальний файл,

Cf — зашифрований файл, отриманий шифруванням AES-256,

Ck — зашифрований ключ файлу за допомогою головного ключа системи Km,

$H(fn)$ — хеш-функція імені файлу для унікальної ідентифікації,

So – функція, яка здійснює збереження кортежу $(H(fn), Cf, Ck, IV)$ в базі даних веб-додатку.

Коли користувач завантажує документ через веб-інтерфейс, система генерує унікальний криптографічний ключ довжиною 256 біт за допомогою криптографічно стійкого генератора псевдовипадкових чисел та використовує його для перетворення оригінального вмісту в зашифровану форму. Зашифрований файл зберігається у спеціальній директорії на сервері під унікальним ім'ям, що обчислюється як SHA-256 хеш від оригінального імені файлу та часової мітки.

Криптографічний ключ зберігається окремо в базі даних у зашифрованому вигляді за допомогою головного ключа системи, що забезпечує додатковий рівень захисту. При запиті користувача на завантаження документа система спочатку перевіряє його права доступу, після чого виконує зворотний процес розшифрування для повернення оригінального вмісту користувачу.

Щоб протидіяти поширеним веб-атакам, у системі було передбачено комплексну перевірку та обробку вхідних даних. Захист від SQL-ін'єкцій реалізується через Object-Relational Mapping (ORM) систему Django, яка автоматично параметризує всі запити до бази даних. Додатково всі спеціальні символи автоматично екрануються, перетворюючи потенційно небезпечні конструкції на безпечні текстові дані [44].

Для захисту від XSS-атак у розробленому додатку працює багаторівневий механізм. По-перше, всі користувацькі дані автоматично екрануються при відображенні на веб-сторінках, що означає перетворення спеціальних HTML-символів на їх безпечні ентитети. Наприклад, символ "<" перетворюється на «<», що робить неможливим виконання HTML-тегів або JavaScript-коду. По-друге, система використовує Content Security Policy заголовки, які обмежують джерела, з яких браузер може завантажувати та виконувати скрипти. По-третє, всі форми введення мають серверну валідацію, яка перевіряє та очищає дані перед їх збереженням у базі даних.

Проти CSRF-атак було реалізовано систему унікальних токенів, які генеруються для кожної користувацької сесії. Коли користувач відкриває форму

для виконання будь-якої дії (створення, редагування, видалення документів), система автоматично додає прихований CSRF-токен до цієї форми. При відправці форми сервер перевіряє наявність та валідність цього токена. Якщо токен відсутній, неправильний або не відповідає поточній сесії користувача, запит відхиляється з помилкою 403. Оскільки зловмисник не може передбачити або отримати CSRF-токен жертви, його атаки стають неефективними.

Така багаторівнева архітектура забезпечує, що навіть якщо один з механізмів захисту буде скомпрометований, інші рівні безпеки продовжать захищати систему від несанкціонованого доступу та шкідливих дій. Нижче наведена блок–схема, що відображає алгоритм роботи системи захисту веб-додатка із захищеним управління доступом до конфіденційних документів (рис. 2.1).

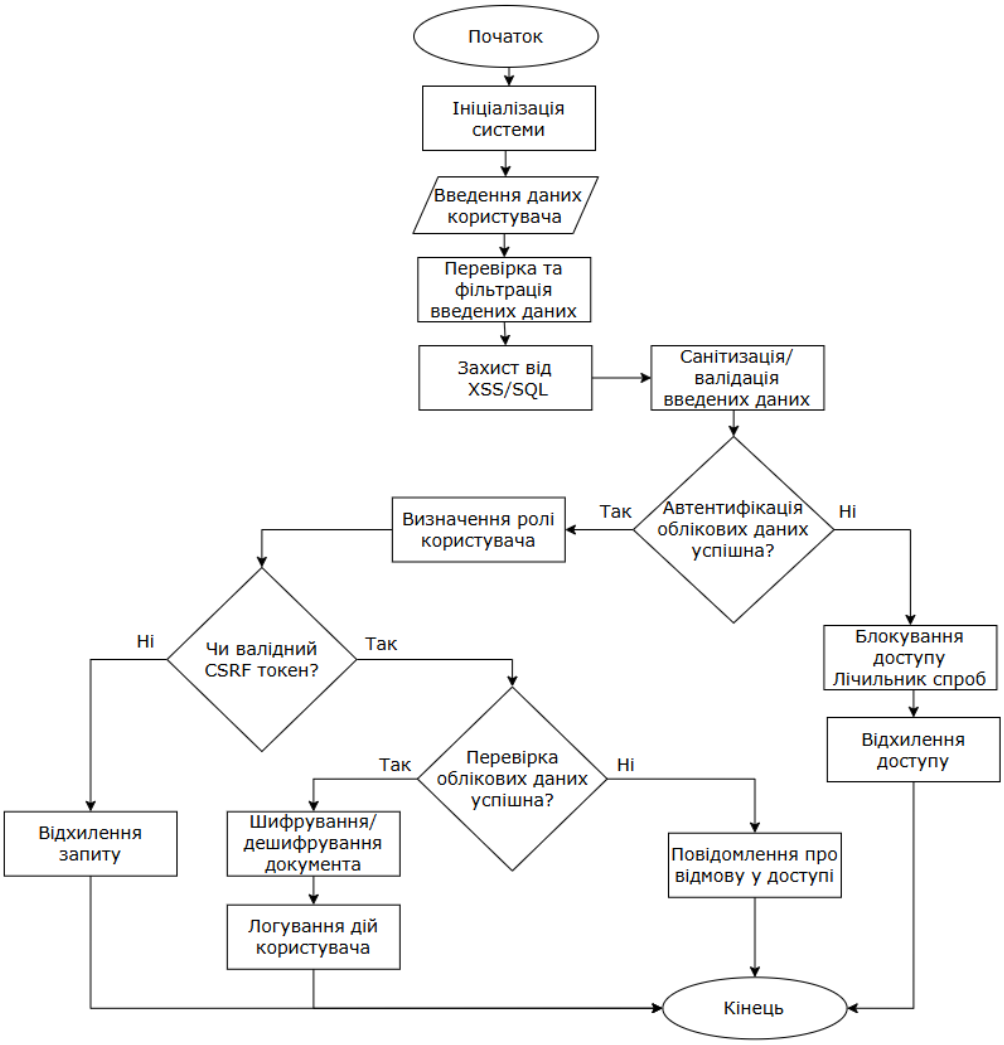


Рисунок 2.1 – Блок–схема, що відображає підхід захисту веб-додатка із управління доступом до конфіденційних документів із використанням

шифрування та механізмів захисту від XSS, CSRF та SQL-ін’єкцій

Крок 1. Початок. Ініціюється запуск системи, завантажуються потрібні компоненти та перевіряється доступність ключових ресурсів.

Крок 2. Ініціалізація системи. Підготовка середовища виконання, налаштування маршрутизаторів запитів, бази даних, шифрувальних бібліотек тощо.

Крок 3. Введення даних користувача. Користувач вводить свої дані (логін, пароль, персональну інформацію) через веб-форму або мобільний додаток.

Крок 4. Перевірка та фільтрація введених даних. Система перевіряє правильність формату введених даних, видаляє небезпечні символи та валідує інформацію.

Крок 5. Захист від XSS/SQL-ін'єкцій через екранування спеціальних символів та параметризовані запити ORM Django.

Крок 6. Валідація та санітизація введених даних. Додаткове очищення та перевірка даних на відповідність встановленим правилам безпеки.

Крок 7. Автентифікація за допомогою PBKDF2 з унікальною сіллю успішна? Система перевіряє, чи відповідають введені користувачем дані тим, що зберігаються в базі даних.

Крок 8а. (Якщо НІ) Блокування доступу/Лічильник спроб. Блокування доступу після перевищення ліміту спроб входу (захист від brute-force атак).

Крок 8б. (Якщо НІ) Відхилення доступу. Система відмовляє в доступі та повертає користувача до початку процесу.

Крок 9. (Якщо ТАК) Визначення ролі користувача. Система визначає, які права та привілеї має автентифікований користувач.

Крок 10. Чи валідний CSRF токен? Перевірка CSRF-токену для захисту від атак підробки міжсайтових запитів.

Крок 11а. (Якщо НІ) Відхилення запиту. При невалідному токені система відхиляє запит як потенційно небезпечний.

Крок 12. (Якщо ТАК) Перевірка облікових даних успішна? Додаткова перевірка

облікових даних користувача для підтвердження його особи.

Крок 13а. (Якщо НІ) Повідомлення про відмову у доступі. Система інформує користувача про неможливість надання доступу.

Крок 14. (Якщо ТАК) Шифрування/дешифрування документа. Шифрування документів AES-256 з унікальним ключем або розшифрування при завантаженні документу залежно від типу операції користувача.

Крок 15. Логування дій користувача. Логування всіх дій користувача для аудиту безпеки.

Крок 16. Кінець. Завершення процесу, закриття з'єднань та очищення тимчасових ресурсів.

Загалом запропонована та реалізована архітектура поєднує багаторівневий контроль доступу, надійне шифрування даних і перевірені механізми захисту від веб-атак. Це дозволяє гарантувати збереження конфіденційності, цілісності та доступності документів у різноманітних сценаріях використання – від внутрішнього обміну у великих корпораціях до державних інформаційних систем із жорсткими регулятивними вимогами. Таким чином, комбінація цих методів дозволяє побудувати багаторівневий захист, який суттєво знижує ризики атак.

Цей підхід дозволяє мінімізувати ризики несанкціонованого доступу, виключити помилки в розподілі прав вручну та забезпечити масштабованість системи – у разі потреби можна легко додати нові ролі або змінити політики доступу, не змінюючи базову логіку авторизації.

2.2 Проєктування взаємодії захищеного веб-додатка управління доступом до конфіденційних документів із базою даних SQLite

Переходячи від загальних функціональних вимог до конкретної реалізації, було здійснено проєктування взаємодії захищеного веб-додатку з реляційною базою даних SQLite, яка виконує не лише функцію сховища, а й слугує важливою складовою багаторівневої системи безпеки. У процесі проєктування було обрано легковагову вбудовану СУБД – SQLite, яка забезпечує високу продуктивність та спрощене розгортання завдяки збереженню всієї бази даних у єдиному файлі. Це

рішення значно полегшує резервне копіювання, відновлення та адміністрування без потреби у запуску окремого серверного компонента, що особливо важливо для систем управління конфіденційними документами з обмеженими ресурсами інфраструктури.

З міркувань безпеки було впроваджено низку заходів: підтримку шифрування на рівні сторінок за допомогою розширення SQLCipher, а також обмеження доступу до бази лише через процес веб-додатку з обмеженими правами користувача ОС. Окрім цього, було застосовано конфігураційні параметри PRAGMA secure_delete для безпечного знищення видалених даних і PRAGMA cipher_page_size для оптимізації шифрування сторінок, що дозволяє зменшити ризики несанкціонованого відновлення інформації. З метою додаткового захисту вміст документів шифрується поза межами ядра СУБД із використанням симетричного алгоритму AES-256, який гарантує як конфіденційність, так і цілісність даних через криптографічну стійкість приблизно 2^{128} операцій для атак методом повного перебору. ER-діаграма першого рівня (рисунок 2.2) моделює основні компоненти системи управління користувачами, автентифікації та контролю сесій, де кожна сутність відіграє чітко визначену роль у забезпеченні безпечного доступу до системи.

Таблиця AUTH_USER містить повний набір атрибутів, необхідних для автентифікації, авторизації та реалізації рольової моделі доступу. Зокрема, паролі зберігаються у вигляді хешів, сформованих за допомогою алгоритму PBKDF2 з унікальною "солею" та налаштовуваною кількістю ітерацій (не менше 100,000), що гарантує стійкість до атак перебору (brute-force) і виключає можливість зворотного відновлення пароля навіть у разі компрометації бази згідно з формулою (2.1). Поля is_active, is_staff, is_superuser та role_level, дозволяють реалізувати багаторівневу модель управління правами доступу, де ролі користувачів чітко обмежують їхні можливості в межах системи.

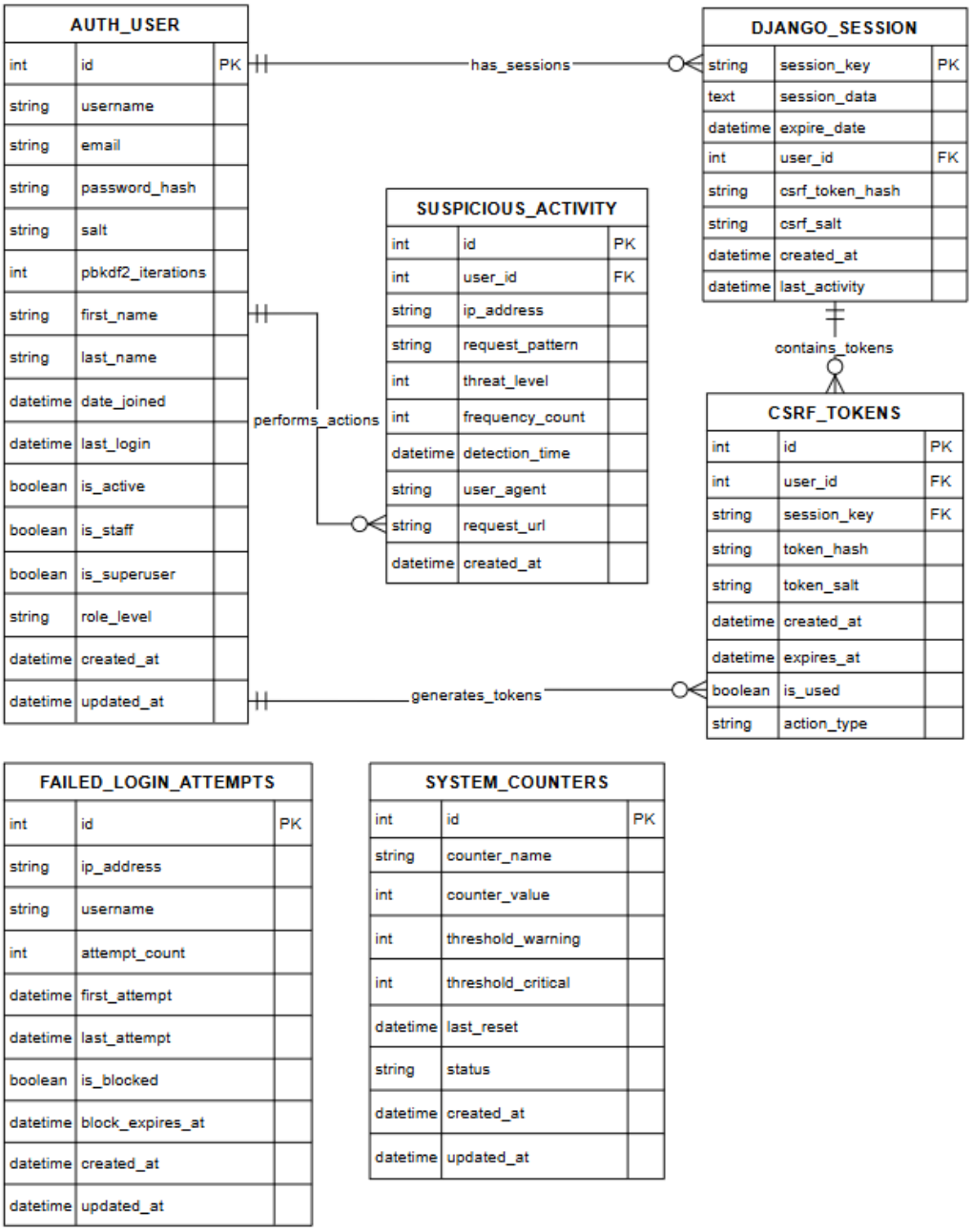


Рисунок 2.2 – Структура ER діаграми бази даних, задіяної у захищеного веб додатку, управління користувачами та сесіями

Математична модель рольової авторизації формалізується системою рівнянь доступу:

$$A(u,d) = \{1, \text{якщо } R(u) \geq L(d) \wedge S(u) = true\ 0\}, \quad (2.3)$$

де $A(u,d)$– функція доступу користувача u до документа d,

$R(u)$ – рівень ролі користувача (public=0, user=1, admin=2, superuser=3),

$L(d)$– рівень захисту документа,

$S(u)$ – статус активності користувача.

Додатково, для суперкористувачів діє розширена формула:

$$A_{s(u,d)} = \{1, \text{якщо } is_{s(u)} = true \wedge S(u) = true, A(u,d)\}, \quad (2.4)$$

Таблиця DJANGO_SESSION реалізує механізм зберігання активних сесій користувачів з додатковими засобами захисту від CSRF-атак. Кожен запис включає унікальний ідентифікатор сесії (session_key), зашифровані дані сесії (session_data), часову позначку закінчення дії сесії та критично важливі поля csrf_token_hash і csrf_salt для зберігання хешованих CSRF-токенів.

Зв'язок з session_key забезпечує валідацію токенів у контексті конкретної користувацької сесії, що унеможливлює крадіжку токенів між різними сесіями. ER-діаграма другого рівня (рисунок 2.3) відображає компоненти системи, відповідальні за управління конфіденційними документами, контроль доступу до них та ведення журналу аудиту дій користувачів.

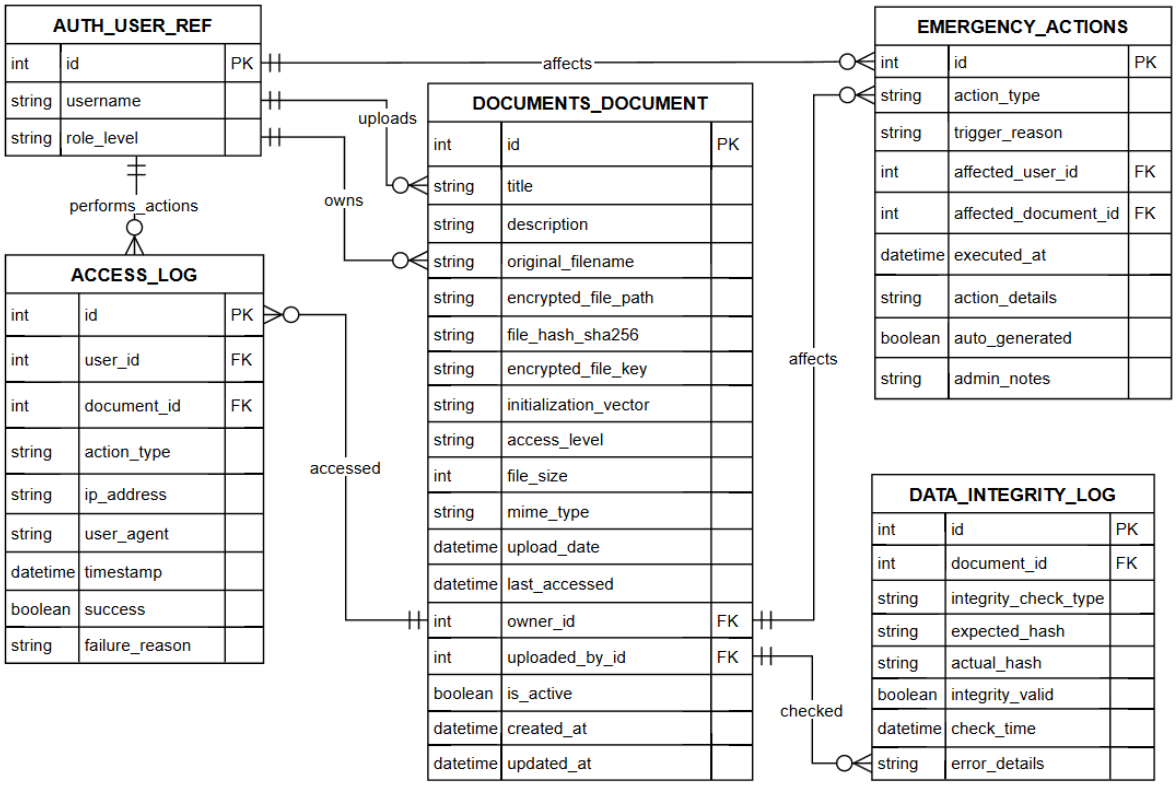


Рисунок 2.3 – Структура ER моделі бази даних задіяної у захищеного веб-додатку управління доступом до конфіденційних документів

Сутність DOCUMENTS_DOCUMENT відповідає за облік і зберігання конфіденційних файлів, кожен із яких супроводжується набором метаданих та криптографічних параметрів. Ключовими атрибутами є encrypted_file_path, який вказує на фізичне розташування зашифрованого файлу у файловій системі, та encrypted_file_key, що містить ключ шифрування у зашифрованому вигляді згідно з формулою (2.7). Додатково зберігається initialization_vector для забезпечення унікальності шифрування кожного файлу та file_hash_sha256 для контролю цілісності. Поле access_level визначає рівень доступності документа відповідно до рольової моделі (public, user, admin, superuser), що дозволяє автоматично контролювати права перегляду та завантаження. Через зовнішні ключі owner_id та uploaded_by_id підтримується зв'язок із таблицею користувачів (AUTH_USER_REF), що забезпечує контроль над правами власності й джерелами походження документів. Захист від SQL-ін'єкцій та витоку даних реалізовано як система моніторингу підозрілих запитів через таблицю SUSPICIOUS_ACTIVITY з полями:

- user_id – ідентифікатор користувача;
- ip_address – IP-адреса запиту;
- request_pattern – шаблон підозрілого запиту;
- threat_level – рівень загрози;
- frequency_count – частота подібних запитів;
- detection_time – час виявлення.

Критичні порогові значення для виявлення атак можна подати у вигляді математичної формули:

$$S(u,t) = \{C, \text{if } R_{s(u,t)} \geq 3 \vee R_{d(u,t)} \geq 10\ W, \text{if } R_{s(u,t)} \geq 1 \vee R_{d(u,t)} \geq 5\ N\}, \quad (2.11)$$

де $R_{s(u,t)}$ – кількість запитів з SQL-паттернами від користувача u за час t,

$R_{d(u,t)}$ – кількість спроб доступу до заборонених даних.

Таблиця ACCESS_LOG реалізує комплексний аудит дій користувачів, зберігаючи детальну інформацію про всі спроби доступу до документів. Кожен запис містить ідентифікатори користувача та документа, тип дії (перегляд,

завантаження, редагування), IP-адресу, user agent браузера та часову мітку. Поля success та failure_reason дозволяють відслідковувати як успішні операції, так і спроби несанкціонованого доступу, що забезпечує можливість виявлення аномальної активності та потенційних атак.

У сукупності, логіка взаємодії між цими таблицями забезпечує цілісну й безпечну архітектуру для захисту конфіденційних даних у веб-додатку. Поєднання шифрування, хешування, ізольованого зберігання сесій і контрольованого розподілу прав доступу дозволяє ефективно реалізувати вимоги до автентифікації, авторизації, захисту документів і моніторингу дій користувачів. Структура ER-діаграми, відображає не лише організаційні зв’язки, а й концептуальну модель безпеки, що лежить в основі функціонування всієї системи.

Враховуючи обмеження SQLite щодо одночасного запису, у системі реалізовано механізм серіалізації транзакцій з використанням режиму WAL (Write-Ahead Logging), що забезпечує послідовне виконання операцій запису та запобігає конфліктам доступу.

Таким чином, взаємодія веб-додатку з базою даних SQLite реалізована з урахуванням особливостей цієї СУБД та вимог до безпеки обробки конфіденційної інформації. Застосування додаткових механізмів захисту, таких як шифрування бази даних та використання параметризованих запитів, дозволяє забезпечити належний рівень безпеки та захисту даних у системі управління доступом до конфіденційних документів.

2.3 Проєктування архітектури веб-додатку і взаємодії компонентів

Проєктування архітектури веб-додатку є ключовим етапом створення безпечної та надійної інформаційної системи. Від узгодженої взаємодії компонентів залежить цілісність даних, ефективність обробки запитів та стійкість до зовнішніх загроз. Забезпечення конфіденційності, цілісності та доступності інформації вимагає інтеграції сучасних криптографічних засобів, систем автентифікації, контролю доступу, а також засобів виявлення й нейтралізації атак на веб-рівні. Раціональна структура компонентів і чітке визначення потоків даних дозволяють досягти

високого рівня захищеності та масштабованості програмного забезпечення. Взаємодія компонентів архітектури розробленого веб-додатку із захищеним управління доступом до документів із шифруванням та захистом від XSS/CSRF/SQL-ін’єкцій наведено блок-схемою нижче (рис. 2.4).

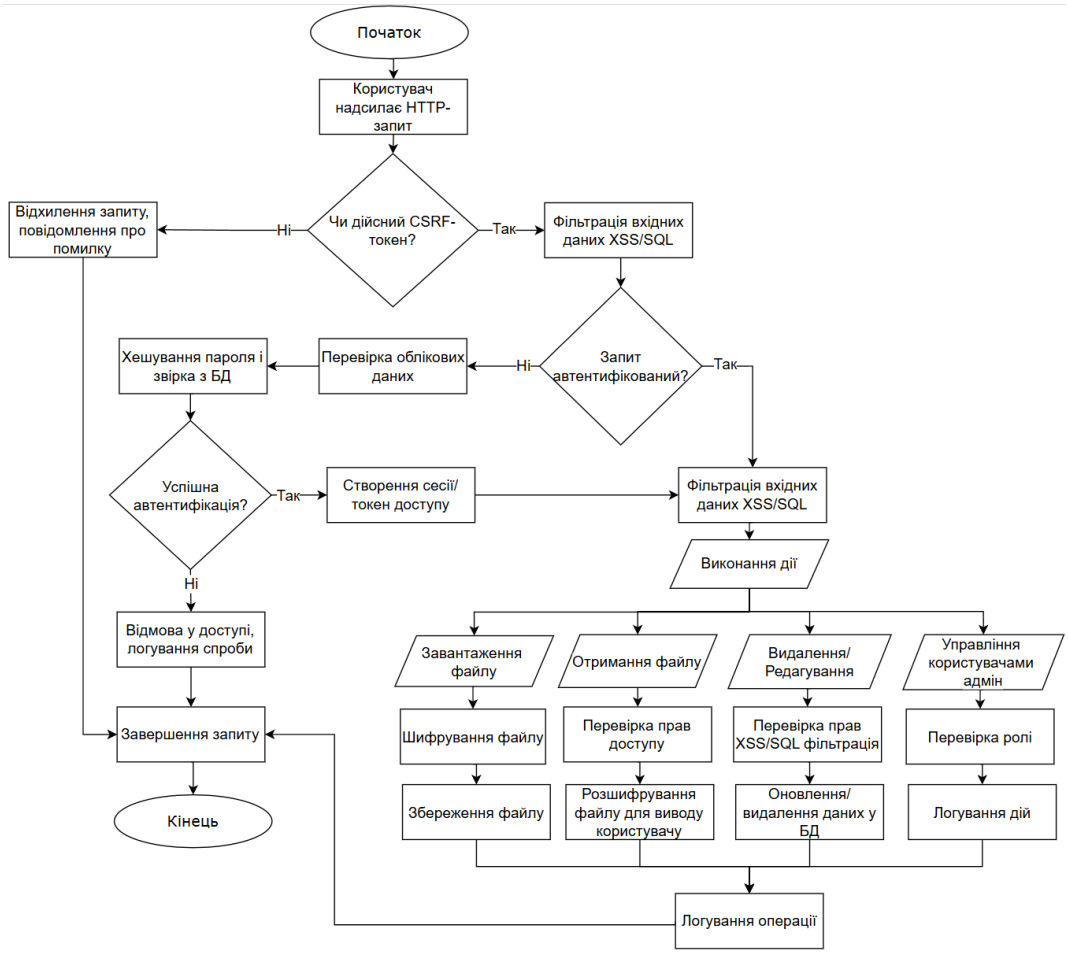


Рисунок 2.4 – Схема взаємодії компонентів архітектури розробленого захищеного веб-додатку із управлінням доступом до конфіденційних документів із шифруванням та захистом від XSS/CSRF/SQL-ін’єкцій.

Крок 1. Початок. На цьому етапі ініціалізується робота веб-додатку, завантажуються основні конфігурації системи, бібліотеки шифрування, механізми автентифікації, а також налаштовуються інтерфейси взаємодії з базою даних та файловим сховищем.

Крок 2. Отримання HTTP-запиту від користувача. Користувач надсилає запит до веб-додатку через інтерфейс користувача. Це може бути як запит на

авторизацію, завантаження файлу або інша дія.

Крок 3. Перевірка CSRF-токена. На цьому етапі відбувається перевірка автентичності запиту шляхом звірки переданого CSRF-токена з очікуваним значенням. У разі невідповідності система припиняє обробку та генерує повідомлення про помилку безпеки.

Крок 4. Фільтрація вхідних даних (захист від XSS та SQL-ін’єкцій). Проводиться очистка та валідація всіх вхідних даних з метою запобігання атакам типу XSS та SQL-ін’єкціям. Застосовуються механізми екранування спеціальних символів, використання параметризованих запитів до БД тощо.

Крок 5. Перевірка автентифікації. Система визначає, чи є запит автентифікованим. Якщо користувач ще не авторизований, він має ввести свої облікові дані.

Крок 6. Хешування пароля та звірка з базою даних. Введений користувачем пароль хешується за допомогою криптографічного алгоритму PBKDF2, після чого отримане значення порівнюється з даними, що зберігаються у базі даних. У разі невірного пароля авторизація блокується.

Крок 7. Створення сесії токена доступу. При успішній автентифікації система генерує сесію або токен для подальшої взаємодії. Токен додається до кожного наступного запиту користувача, що дозволяє підтримувати безпечну авторизацію.

Крок 8. Визначення типу дії. Система визначає, яку дію має виконати користувач: завантаження файлу, запит доступу, редагування, видалення, або адміністративні дії.

Крок 9. Завантаження документа (якщо обрана відповідна дія). Користувач передає файл, який перед збереженням шифрується за допомогою симетричного алгоритму (AES). Шифрований файл зберігається у файловій системі або базі даних.

Крок 10. Отримання доступу до документа. При запиті на перегляд файлу система перевіряє права доступу користувача. Якщо доступ дозволений, файл розшифровується і надається для перегляду.

Крок 11. Редагування чи видалення файлу. Здійснюється додаткова перевірка

прав доступу, після чого користувач може внести зміни до своїх файлів або видалити їх. Дані проходять повторну фільтрацію для захисту від шкідливого коду.

Крок 12. Адміністративні дії. Адміністратор отримує доступ до розширеного функціоналу: керування обліковими записами, перегляд усіх файлів, зміна статусів доступу та аудит дій користувачів.

Крок 13. Логування усіх критичних подій. Система зберігає записи про всі важливі операції: входи в систему, завантаження, зміни, відмови у доступі. Це необхідно для моніторингу безпеки та реагування на інциденти.

Крок 14. Завершення обробки запиту. Система формує відповідь на запит користувача, надсилає її через інтерфейс, і завершує виконання поточної сесії обробки.

Адміністратор веб-додатка виконує функції керування системою, що включає створення та модифікацію облікових записів, розподіл прав доступу до документів, а також моніторинг дій користувачів. Він також має змогу переглядати всі завантажені файли, змінювати їх статус, обмежувати або відкликати доступ до них, а також стежити за дотриманням політик безпеки в системі. Звичайний користувач, у свою чергу, має обмежений набір можливостей. Функціональні сценарії передбачають стандартну взаємодію користувача з інтерфейсом: автентифікація, завантаження файлів, перегляд доступних документів (рис. 2.5).

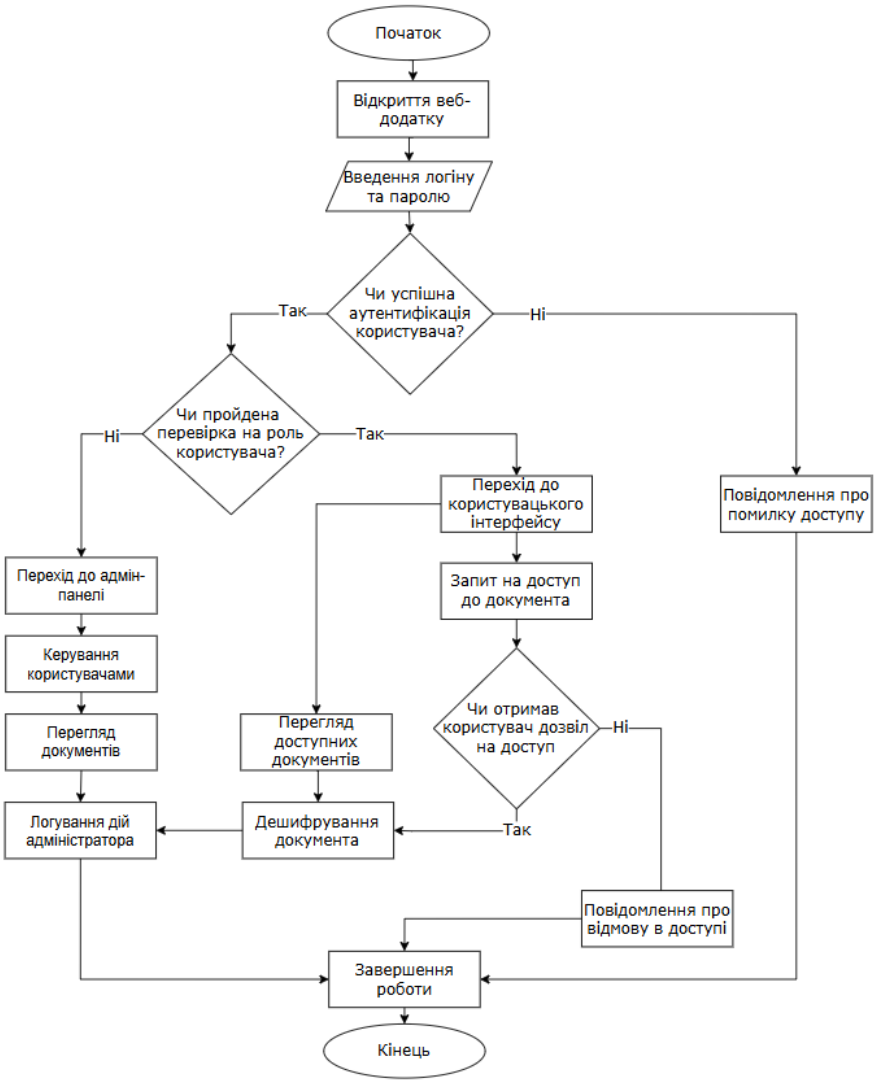


Рисунок 2.5 – Схема взаємодії користувача з функціоналом розробленого веб-додатку захищеного управління доступом до конфіденційних документів

Крок 1. Початок. На цьому етапі запускається веб-додаток. Відбувається ініціалізація системи, зокрема завантаження механізмів шифрування/дешифрування, а також налаштування автентифікації користувачів.

Крок 2. Відкриття веб-додатку.

Користувач відкриває інтерфейс веб-додатку через браузер, ініціюючи взаємодію із системою.

Крок 3. Введення логіну та паролю.

Користувач вводить облікові дані для входу: логін і пароль.

Крок 4. Перевірка успішності автентифікації користувача.

Система перевіряє коректність введених даних. Якщо автентифікація

неуспішна – генерується повідомлення про помилку доступу (перехід до Кроку 14). Якщо успішна – перехід до перевірки ролі користувача.

Крок 5. Перевірка ролі користувача.

Визначається, до якої категорії належить користувач (адміністратор чи звичайний користувач). Якщо це адміністратор – перехід до адміністративного функціоналу. Якщо звичайний користувач – перехід до користувацького інтерфейсу. Якщо суперкористувач – перехід до інтерфейсу суперкористувача.

Крок 6. Дії адміністратора (якщо перевірка на роль не пройдена).

Адміністратор має доступ до:

– панелі керування користувачами;

– перегляду документів;

– логування своїх дій.

Крок 7. Перехід до користувацького інтерфейсу.

Користувач отримує доступ до власного інтерфейсу для взаємодії з документами.

Крок 8. Запит на доступ до документа.

Користувач обирає потрібний документ і надсилає запит на отримання доступу.

Крок 9. Перевірка дозволу на доступ.

Система перевіряє, чи має користувач необхідні права. Якщо немає – формується повідомлення про відмову в доступі (перехід до Кроку 14). Якщо дозвіл є – доступ дозволено.

Крок 10. Перегляд доступних документів.

Користувачу відкривається перелік документів, до яких йому надано доступ.

Крок 11. Дешифрування документа.

Документ розшифровується перед переглядом, використовуючи механізми безпечної обробки даних.

Крок 12. Завершення роботи.

Після завершення дій (перегляд, дешифрування, виконання команд) користувач або система ініціює завершення сесії.

Крок 13. Повідомлення про помилку або відмову в доступі (за потреби).

Якщо на будь-якому етапі виникла помилка автентифікації або відмова у правах доступу, система інформує про це користувача.

Крок 14. Кінець.

Система завершує обробку поточного запиту користувача, надсилає відповідь (успішну або з помилкою), за потреби фіксує дію в журналі та закриває сесію. Уся взаємодія із захищеним функціоналом відбувається згідно з принципами розмежування прав доступу, цілісності та конфіденційності даних. Завершення цього кроку забезпечує цілісність, конфіденційність та контрольованість усіх операцій, виконаних у межах взаємодії.

2.4 Висновки до розділу 2

У цьому розділі було здійснено проєктування захищеного веб-додатку для управління доступом до конфіденційних документів із реалізацією сучасних механізмів інформаційної безпеки. Проведено постановку задачі та визначено функціональні вимоги до системи, зосереджуючись на забезпеченні захисту даних шляхом використання шифрування, а також запобігання атакам типу XSS, CSRF та SQL-ін’єкцій. Поєднання PBKDF2 хешування, AES-256 шифрування, ізольованого зберігання сесій, захищених CSRF-токенів і контрольованого розподілу прав доступу дозволяє ефективно реалізувати вимоги до автентифікації, авторизації, захисту документів і моніторингу дій користувачів.

Таким чином, взаємодія веб-додатку з базою даних SQLite реалізована з урахуванням особливостей цієї СУБД та вимог до безпеки обробки конфіденційної інформації згідно з міжнародними стандартами.

Представлено схеми взаємодії між модулями та користувачем, що відображають логіку обробки запитів та контролю доступу. Отримані результати створюють концептуальну основу для наступного етапу – реалізації функціонального прототипу веб-додатку з підтримкою захищеного зберігання, перегляду та управління конфіденційними документами у багатокористувацькому середовищі.

3 РЕАЛІЗАЦІЯ ТА ТЕСТУВАННЯ ЗАХИЩЕНОГО ВЕБ-ДОДАТКУ УПРАВЛІННЯ ДОСТУПОМ ДО КОНФІДЕНЦІЙНИХ ДОКУМЕНТІВ

У третьому розділі здійснено практичну реалізацію захищеного веб-додатку управління доступом до конфіденційних документів на основі розробленої архітектури та проектних рішень. Обґрунтовано вибір мови програмування Python з фреймворком Flask для серверної частини системи, а також сучасних веб-технологій для створення інтерактивного користувацького інтерфейсу. Розроблено інтуїтивний користувацький інтерфейс з диференційованим функціоналом для різних категорій користувачів та наведено детальну інструкцію з користування системою. Проведено комплексне тестування розробленого програмного продукту, що підтвердило ефективність впроваджених механізмів захисту та коректність функціонування всіх компонентів системи.

3.1 Обґрунтування вибору мови програмування та інструментів розробки

Для розробки захищеного веб-додатку управління доступом до конфіденційних документів одним із ключових етапів є обґрунтований вибір мови програмування, фреймворків і засобів розгортання. Саме ці технології визначають не лише архітектуру та функціональність системи, а й її продуктивність, масштабованість, безпеку, легкість у підтримці та подальшому розвитку. В умовах швидко змінюваного ринку IT та підвищених вимог до якості і швидкості розробки, особливої уваги потребує попередній аналіз сучасних інструментів, доступних на сьогодні. На поточному етапі розвитку індустрії найбільш поширеними мовами програмування для розробки вебзастосунків є JavaScript, Java, PHP, C# та Python. Вони активно використовуються у промисловій розробці, мають розвинені екосистеми та спільноти, а також підтримуються великою кількістю платформ та інструментів.

JavaScript є основною мовою фронтенд-розробки та водночас однією з найпопулярніших мов для створення серверної логіки за допомогою середовища

виконання Node.js. Перевагою JavaScript є його універсальність – можливість реалізації повного циклу розробки на одній мові (full-stack). JavaScript демонструє високу продуктивність у контексті серверних фреймворків (Node.js середня пропускна здатність ≈ 12 000 req/s) та найширшу екосистему на GitHub (≈ 3,5 млн репозиторіїв) і StackOverflow (≈ 4,2 млн запитань). Універсальність мови дозволяє розробляти фронтенд і бекенд на одній технології, що скорочує час на навчання й обслуговування. Водночас динамічна типізація може ускладнювати підтримку великих кодових баз і призводити до прихованих помилок на етапі виконання.

Java – це мова зі строгою об'єктно-орієнтованою парадигмою, яка широко використовується у великих корпоративних проєктах, банківських системах, мобільних застосунках (особливо на Android) та серверних рішеннях. Серед її переваг: висока надійність, масштабованість, підтримка багатопотоковості та розвинена екосистема інструментів (Spring, Hibernate тощо). Java утримує високі позиції в корпоративному сегменті та забезпечує виняткову пропускну здатність (≈ 15 000 req/s). Строга ООР-парадигма й багатопоточність роблять його надійним для масштабованих сервісів, проте велика кількість шаблонного коду та складніша початкова конфігурація можуть уповільнювати швидкість розробки. Окрім того, порівняно з іншими мовами, Java вимагає більше коду для реалізації базової логіки та має складнішу конфігурацію на старті.

PHP є однією з найстаріших мов для веброзробки, яка й досі активно застосовується у створенні сайтів, CMS-систем (наприклад, WordPress, Joomla) та простих вебдодатків. Основними перевагами PHP є простота у використанні, низький поріг входу та велика кількість готових рішень. Однак PHP все частіше піддається критиці через обмежену гнучкість, складнощі з масштабуванням та застарілу архітектуру низки застосунків, що на ній працюють.

C# у поєднанні з фреймворком .NET є потужним інструментом для створення десктопних, мобільних і вебзастосунків. C#/.NET поєднує високу продуктивність (≈ 14 000 req/s) і інтеграцію з хмарними сервісами Microsoft Azure. Разом із цим, C# орієнтований переважно на середовище Windows, а налаштування середовища розробки та підтримка проєктів можуть бути ресурсомісткими. Python забезпечує

швидкий старт і зручність синтаксису, має великий обсяг спільнотної підтримки (≈ 3,5 млн запитань на StackOverflow) і численні бібліотеки, однак продуктивність його веб-фреймворків (≈ 6 000 req/s) нижча порівняно з компільованими мовами, що може потребувати додаткового масштабування інфраструктури. Він підтримує кілька парадигм програмування, має велику кількість бібліотек для роботи з базами даних, вебсервісами, аналітикою, машинним навчанням і автоматизацією. Python має величезну активну спільноту, якісну документацію та стабільну підтримку. Його єдиним помітним недоліком може бути нижча продуктивність у порівнянні з компільованими мовами, однак це рідко є критичним для більшості веб-орієнтованих проєктів. Для більш наочного порівняння основних мов програмування та інструментів розробки наведено нижче (табл. 3.1) [44].

Таблиця 3.1 – Кількісне порівняння мов програмування для веб-розробки.

Мова	TIOBE Rank	Throughput (req/s)	SO Questions (млн)	GitHub Repos (млн)
JavaScript	7	12 000	4,2	3,5
Java	2	15 000	3,8	2,7
PHP	11	8 000	2,4	1,1
C#/.NET	5	14 000	1,9	1,3
Python	1	6 000	3,5	3,1

На підставі наведених даних для розробки захищеного веб-додатку було обрано Python: він забезпечує достатню продуктивність, велику кількість готових пакетів для реалізації безпеки авторизації та автентифікації, а також сприяє швидкому прототипуванню й легкій підтримці коду. Тому саме Python було обрано як основну мову програмування для реалізації програмного продукту.

Серед веб-фреймворків, які підтримують мову Python, найбільш відомими є Django, Flask та FastAPI. Для обґрунтування вибору конкретного фреймворка необхідно провести їх порівняльний аналіз за критеріями функціональності, складності освоєння, безпеки, розширюваності та готовності до використання у великих проєктах. Django – це повноцінний фреймворк для розробки веб-застосунків, який дотримується принципу "все включено" (batteries included). Він

має вбудовану ORM, систему маршрутизації, механізми автентифікації, панель адміністратора, захист від основних веб-загроз та інші важливі компоненти, що значно скорочують час розробки. Django також підтримує модульну архітектуру, дозволяючи легко масштабувати проєкт.

Flask – мікрофреймворк, який надає розробнику більше свободи, але вимагає ручного підключення багатьох функцій, таких як аутентифікація, валідація форм або адміністрування. Він є чудовим вибором для невеликих проєктів або тоді, коли потрібно повністю контролювати структуру застосунку, однак для середніх і великих систем його використання може ускладнити розгортання та підтримку.

FastAPI – сучасний фреймворк, орієнтований на розробку API з використанням асинхронного підходу. Він демонструє високу продуктивність, автоматично генерує документацію і добре підходить для сервіс-орієнтованих архітектур. Проте, як відносно новий інструмент, FastAPI ще не має такого широкого набору готових рішень, як Django, і потребує більше часу на налаштування при створенні повноцінного веб-застосунку.

FastAPI демонструє найвищу продуктивність серед представлених фреймворків, однак через свою відносну новизну він поки що має меншу підтримку спільноти та обмежену екосистему додаткових модулів. Нижче наведено порівняння вище згаданих фреймворків Django, у свою чергу, є більш збалансованим варіантом: він має високу популярність, широку спільноту, велику кількість супутніх пакетів, а також стабільні показники продуктивності. Flask, як мікрофреймворк, підходить для невеликих або кастомізованих проєктів, проте вимагає більше ручної конфігурації (табл. 3.2) [44].

Таблиця 3.2 – Кількісне порівняння сучасних фреймворків для Python

Фреймворк	Throughput (середина простого CRUD, req/s)	GitHub Stars (тис.)	SO Questions (тис.)	Пакетів у PyPI (тис.)
Django	7 500	74	735	8,2
Flask	5 000	61	290	3,7
FastAPI	12 000	50	47	0,9

З огляду на потреби створення комплексного веб-застосунку з підтримкою автентифікації, адмін-панелі, роботи з БД та високими вимогами до захисту, вибір фреймворка зумовлено перевагами Django як найзрілішого та найфункціональнішого рішення.

Для повноцінної реалізації вебзастосунку не менш важливо обрати ефективне середовище розробки. У Python-екосистемі найпопулярнішими середовищами є PyCharm, Visual Studio Code (VS Code), Sublime Text та Atom. Щоб оцінити придатність цих середовищ до роботи з Django, нижче подано таблицю з основними технічними та статистичними характеристиками кожного з них.

Доцільно систематизувати основні параметри кожного середовища розробки у вигляді порівняльної таблиці. Це оцінить їхні можливості за ключовими критеріями. Порівняльну таблицю середовищ подано нижче (табл.3.3).

Таблиця 3.2 – Кількісне порівняння середовищ розробки для Python з підтримкою Django

Середовище	Середнє споживання ОЗУ (MB)	Щотижневі завантаження (тис.)	Розширень плагінів	SO Questions (тег, тис.)
PyCharm	850	210	2 100	145
VS Code	320	1 050	40 000	1 250
Sublime Text	60	95	5 200	30
Atom	650	45	8 300	25

Після проведеного кількісного аналізу Python-фреймворків виявлено, що FastAPI забезпечує найвищу пропускну здатність (≈12 000 req/s), проте його екосистема пакунків поки що відносно невелика (≈900 пакунків на PyPI) і кількість питань на StackOverflow (≈47 тис.) свідчить про обмежену спільнотну підтримку порівняно з Django. Django демонструє збалансовані показники: середній throughput ≈7 500 req/s, велика кількість зірок на GitHub (≈74 тис.) і питань на StackOverflow (≈735 тис.), а також понад 8 200 пакунків у PyPI. Flask трохи поступається за продуктивністю (≈5 000 req/s), але має широку спільнотну базу (≈290 тис. питань на SO) і майже 3 700 пакунків.

З огляду на необхідність швидкого розгортання «з коробки» готових механізмів автентифікації, панелі адміністрування та захисту від XSS/CSRF/SQL-ін’єкцій, а також на вимогу до стабільної підтримки й документації, вибір було зроблено на користь Django: він поєднує прийнятну продуктивність і широку екосистему, що оптимізує час розробки та знижує витрати на впровадження додаткових модулів.

Що стосується середовищ розробки, VS Code лідирує за популярністю (більше 1 050 тис. щотижневих завантажень та 40 000 розширень), проте вимагає значного часу на налаштування і підключення плагінів для повноцінної роботи з Django. PyCharm пропонує повну інтеграцію з Django та всі необхідні інструменти «з коробки», але споживає у середньому 850 МБ ОЗУ й має суттєву платну частину функціональності. Sublime Text і Atom демонструють скромніші метрики з огляду на меншу кількість питань і розширень, що обмежує їхню застосовність у великих проєктах із високими вимогами до безпеки та масштабованості.

Таким чином, для реалізації програмного засобу було обрано мову програмування Python, фреймворк Django та інтегроване середовище розробки PyCharm. Така комбінація забезпечує надійну основу для створення безпечного, масштабованого та функціонально повного вебзастосунку управління доступом до конфіденційних документів.

3.2 Опис функціональних модулів розробленого веб-додатку

Для реалізації захищеного веб-додатку управління доступом до конфіденційних документів було розроблено веб-додаток з використанням мови програмування Python та фреймворку Django в середовищі розробки Visual Studio Code. Система складається з декількох функціональних модулів, які забезпечують повний цикл роботи з документами від створення до видалення з урахуванням рівнів доступу користувачів. Код нижче відображає імпорт необхідних бібліотек та модулів для функціонування веб-додатку.

```
import os
from django.http import HttpResponse, HttpResponseForbidden
from django.shortcuts import redirect, render, get_object_or_404
from django.contrib.auth.forms import PasswordChangeForm
from django.contrib.auth.models import User
```

```
from secure_docs import settings
from django.contrib.auth.views import LoginView
from django.contrib.auth.decorators import login_required
```

У цій частині коду підключаються основні компоненти Django для роботи з HTTP-запитами, рендерингом сторінок, автентифікацією користувачів та взаємодією з базою даних. Також імпортуються власні форми та моделі для роботи з документами.

Розроблений веб-додаток призначений для централізованого управління доступом до конфіденційних документів в організаційному середовищі та забезпечує автоматизоване завантаження, зберігання і розповсюдження файлів між користувачами з різними рівнями привілеїв доступу. Захищений веб-додаток реалізує чотирирівневу систему доступу, де кожен документ може мати статус public, user, admin або superuser, що відповідає ієрархії користувачів у системі. Додаток автоматично здійснює криптографічне перетворення завантажених файлів перед їх збереженням у файловій системі сервера та забезпечує зворотне розшифрування при запиті користувача на завантаження.

Для забезпечення криптографічного захисту конфіденційних документів було імплементовано симетричний алгоритм шифрування AES-256 з використанням бібліотеки cryptography для Python. Код нижче демонструє реалізацію методу шифрування файлів у моделі Document:

```
def encrypt_and_save_file(self, uploaded_file):
    from cryptography.fernet import Fernet
    key = Fernet.generate_key()
    cipher_suite = Fernet(key)
    file_content = uploaded_file.read()
    encrypted_content = cipher_suite.encrypt(file_content)
    self.encryption_key = key
    self.filename = uploaded_file.name
```

Процес шифрування ініціюється автоматично при завантаженні файлу користувачем, генеруючи унікальний ключ шифрування для кожного документа окремо. Зашифрований контент зберігається у спеціально призначеній директорії з унікальним іменем файлу, що унеможливлює ідентифікацію оригінального документа сторонніми особами. Код нижче показує реалізацію методу розшифрування документів:

```
def decrypt_file(self):
    try:
        from cryptography.fernet import Fernet
        cipher_suite = Fernet(self.encryption_key)
        with open(self.encrypted_file_path, 'rb') as encrypted_file:
            encrypted_content = encrypted_file.read()
        decrypted_content = cipher_suite.decrypt(encrypted_content)
        return decrypted_content
    except Exception:
        return None
```

Розшифрування здійснюється через відповідний метод, який відновлює оригінальний вміст файлу тільки при наявності відповідних криптографічних ключів та успішній верифікації прав доступу користувача. Код нижче демонструє початок головної функції для відображення документів з урахуванням рівня доступу користувача.

```
def home(request):
    search_query = request.GET.get('search', '')
    if request.user.is_superuser:
        if search_query:
            documents = Document.objects.filter(Q(title__icontains=search_query)) |
Document.objects.filter(Q(uploaded_by__username__icontains=search_query))
        else:
            documents = Document.objects.all()
```

Функція отримує пошуковий запит та визначає документи для суперкористувача. Суперкористувач має доступ до всіх документів у системі. Код нижче показує логіку фільтрації документів для адміністраторів.

```
    elif request.user.is_staff:
        if search_query:
            documents = Document.objects.filter(Q(title__icontains=search_query) & Q(access_level__in=['admin', 'user',
'public'])) | Document.objects.filter(Q(uploaded_by__username__icontains=search_query)
    & Q(access_level__in=['admin', 'user', 'public']))
        else:
            documents = Document.objects.filter(Q(access_level__in=['admin', 'user', 'public']))
```

Адміністратори бачать документи з рівнем доступу "admin", "user" та "public". Пошук здійснюється як за назвою документа, так і за ім'ям користувача. Код нижче демонструє продовження логіки фільтрації для звичайних та неавторизованих користувачів.

```
    elif request.user.is_authenticated: if search_query:
            documents = Document.objects.filter(Q(title__icontains=search_query) & Q(access_level__in=['user',
'public'])) | Document.objects.filter(Q(uploaded_by__username__icontains=search_query) & Q(access_level__in=['user',
'public']))
        else: documents = Document.objects.filter(Q(access_level__in=['user', 'public']))
    else:  if search_query:
            documents = Document.objects.filter(Q(title__icontains=search_query) & Q(access_level='public')) |
```

```
Document.objects.filter(Q(uploaded_by__username__icontains=search_query) & Q(access_level='public'))
        else:  documents = Document.objects.filter(Q(access_level='public'))
    return render(request, 'home.html', {'documents': documents})
```

Авторизовані користувачі мають доступ до документів рівня "user" та "public", неавторизовані бачать лише публічні документи. Функція повертає відфільтровані документи для відображення на головній сторінці. Код нижче відображає функцію для перегляду детальної інформації про документ з перевіркою прав доступу.

```
def document_detail(request, doc_id):
    document = get_object_or_404(Document, id=doc_id)
    if not document.can_user_view(request.user):
        return HttpResponseForbidden("Вам заборонено перегляд цього документа.")
    return render(request, 'document_detail.html', {'document': document})
```

Функція отримує документ за його ідентифікатором та перевіряє чи має користувач право на його перегляд через метод моделі. У разі відсутності прав повертається заборона доступу. Код нижче демонструє клас для обробки процесу автентифікації користувачів.

```
class CustomLoginView(LoginView):
    template_name = 'login.html'
    def dispatch(self, request, *args, **kwargs):
        if request.user.is_authenticated:
            return redirect('')
        return super().dispatch(request, *args, **kwargs)
```

Клас розширює стандартний механізм входу Django та перенаправляє вже автентифікованих користувачів на головну сторінку. Це запобігає повторному відображенню форми входу для користувачів, які вже увійшли в систему.

Протидія атакам типу SQL-ін'єкції забезпечується через використання ORM-системи Django, яка автоматично здійснює параметризацію запитів до бази даних та екранування спеціальних символів у користувацькому вводі. Всі операції з базою даних виконуються через Q-об'єкти Django, що виключає можливість прямого введення SQL-коду в запити. Код нижче демонструє безпечну реалізацію пошукового запиту:

```
documents = Document.objects.filter(
    Q(title__icontains=search_query) & Q(access_level__in=['admin', 'user', 'public'])
) | Document.objects.filter(
    Q(uploaded_by__username__icontains=search_query) & Q(access_level__in=['admin', 'user', 'public'])
)
```

Веб-додаток автоматично екранує всі параметри пошуку та використовує

параметризовані запити, що унеможливлює виконання шкідливого SQL-коду через пошукові поля. Код нижче відображає початок функції створення нового документа.

```
@login_required
def create_document(request):
    if request.method == "POST":
        form = DocumentForm(request.POST, request.FILES, user=request.user)
        if form.is_valid():
            document = form.save(commit=False)
            document.uploaded_by = request.user
            document.owner = request.user
```

Функція перевіряє POST-запит та валідує форму документа. Встановлює поточного користувача як власника та автора документа. Код нижче демонструє процес збереження шифруваного файлу.

```
            document.comment = form.cleaned_data.get('comment')
            uploaded_file = request.FILES.get('file')
            if uploaded_file:
                document.encrypt_and_save_file(uploaded_file)
            document.save()
            return redirect('home')
```

Система отримує коментар та файл з форми, шифрує файл перед збереженням.

Механізм захисту від міжсайтового скриптингу реалізовано через автоматичне екранування користувацького вводу у всіх точках відображення даних на веб-сторінках. Django-фреймворк за замовчуванням здійснює HTML-екранування всіх змінних, що передаються до шаблонів, перетворюючи потенційно небезпечні символи на їх HTML-ентитети. Додатково впроваджено серверну валідацію всіх форм введення даних через спеціалізовані класи форм Django.

Після успішного збереження користувач перенаправляється на головну сторінку. Код нижче показує перевірку прав доступу до файлу для завантаження.

```
def download_document(request, doc_id):
    document = get_object_or_404(Document, id=doc_id)
    if not document.can_user_view(request.user):
        return HttpResponse("У вас немає доступу до цього файлу", status=403)
    if not document.encrypted_file_path:
        return HttpResponse("Файл відсутній", status=404)
```

Функція отримує документ за ідентифікатором та перевіряє права користувача. Якщо файл відсутній, повертається відповідний статус помилки. Код нижче демонструє процес розшифрування та повернення файлу.

```
    decrypted_content = document.decrypt_file()
    if decrypted_content is None:
        return HttpResponse("Помилка розшифрування файлу", status=500)
    response = HttpResponse(decrypted_content, content_type="application/octet-stream")
    response["Content-Disposition"] = f'attachment; filename="{document.filename}"'
    return response
```

Система розшифровує файл та повертає його користувачу для завантаження. У разі помилки розшифрування повертається відповідний статус. Код нижче демонструє перевірку прав для видалення документа.

```
@login_required
def delete_document(request, doc_id):
    document = get_object_or_404(Document, id=doc_id)
    if (document.uploaded_by != request.user or request.user.is_superuser) and not request.user.is_superuser:
        return HttpResponseForbidden("Ви не маєте прав для видалення цього документа.")
```

Функція перевіряє чи є користувач автором документа або суперкористувачем. Лише вони мають право видаляти документи. Код нижче показує процес видалення файлу та запису з бази даних.

```
    if document.encrypted_file_path:
        file_path = os.path.join(settings.MEDIA_ROOT, document.encrypted_file_path)
        if os.path.exists(file_path):
            os.remove(file_path)
    document.delete()
    return redirect('home')
```

Система видаляє зашифрований файл з файлової системи та запис про документ з бази даних. Після видалення користувач перенаправляється на головну сторінку. Код нижче відображає функцію профілю користувача з відображенням списку інших користувачів.

```
@login_required
def profile(request):
    if request.user.is_superuser:
        users = User.objects.exclude(is_superuser=True)
    elif request.user.is_staff:
        users = User.objects.filter(is_staff=False, is_superuser=False)
    else:
        users = User.objects.none()
    return render(request, 'profile.html', {'user': request.user, 'profiles': users})
```

Функція визначає який список користувачів може бачити поточний користувач залежно від його ролі. Суперкористувач бачить всіх крім інших суперкористувачів, адміністратор бачить лише звичайних користувачів. Код нижче демонструє функцію зміни пароля користувача.

```
@login_required
```

```
def change_password(request):
    if request.method == 'POST':
        form = PasswordChangeForm(request.user, request.POST)
        if form.is_valid():
            form.save()
            return redirect('home')
    else:
        form = PasswordChangeForm(request.user)
    return render(request, 'change_password.html', {'form': form})
```

Функція використовує стандартну форму Django для зміни пароля та після успішної зміни перенаправляє користувача на головну сторінку. Доступ до функції мають лише автентифіковані користувачі. Код нижче відображає перевірку прав для створення користувача.

```
@login_required
def create_user(request):
    if not request.user.is_superuser and not request.user.is_staff:
        return HttpResponseForbidden("У вас немає прав для створення акаунтів.")
```

Функція перевіряє чи має користувач права адміністратора або суперкористувача. Лише вони можуть створювати нові облікові записи. Код нижче демонструє обробку форми створення користувача.

```
    if request.method == "POST":
        form = CustomUserCreationForm(request.POST)
        if form.is_valid():
            form.save()
            return redirect('profile')
    else:
        form = CustomUserCreationForm()
    return render(request, 'create_user.html', {'form': form})
```

Система обробляє POST-запит з даними нового користувача та зберігає його. Після успішного створення відбувається перенаправлення на сторінку профілю. Код нижче показує перевірку прав для редагування користувача.

```
@login_required
def edit_user(request, user_id):
    if not request.user.is_superuser and not request.user.is_staff:
        return HttpResponseForbidden("У вас немає прав для редагування акаунтів.")
    user = get_object_or_404(User, id=user_id)
```

Функція перевіряє права доступу та отримує користувача за ідентифікатором. Лише адміністратори можуть редагувати облікові записи. Код нижче демонструє обробку форми редагування користувача.

```
    if request.method == 'POST':
        form = EditUserForm(request.POST, instance=user)
        if form.is_valid():
```

```
            form.save()
            return redirect('profile')
    else:
        form = EditUserForm(instance=user)
    return render(request, 'edit_user.html', {'form': form, 'user': user})
```

Система обробляє зміни даних користувача через спеціальну форму редагування. Форма попередньо заповнюється існуючими даними користувача. Код нижче демонструє перевірку прав для видалення користувача.

```
@login_required
def delete_user(request, user_id):
    if not request.user.is_superuser and not request.user.is_staff:
        return HttpResponseForbidden("У вас немає прав для видалення акаунтів.")
    user = get_object_or_404(User, id=user_id)
```

Функція перевіряє права доступу адміністратора та отримує користувача для видалення. Звичайні користувачі не мають доступу до цієї функції. Код нижче показує додаткові перевірки безпеки та видалення.

```
    if user.is_superuser:
        return HttpResponseForbidden("Ви не можете видаляти суперкористувачів.")
    if request.user.is_staff and not request.user.is_superuser and user.is_staff:
        return HttpResponseForbidden("Ви не можете видаляти інших адміністраторів.")
    user.delete()
    return redirect('profile')
```

Система забороняє видалення суперкористувачів та адміністраторам видаляти один одного. Після успішного видалення відбувається перенаправлення на профіль. Захист від міжсайтових підроблених запитів забезпечується через вбудований механізм CSRF-токенів Django, який автоматично генерує унікальні токени для кожної користувацької сесії та перевіряє їх наявність у всіх POST-запитах. Кожна форма у системі містить прихований CSRF-токен, який валідується сервером перед обробкою запиту. Код нижче демонструє застосування CSRF-захисту:

```
@login_required
def create_document(request):
    if request.method == "POST":
        # CSRF-токен автоматично перевіряється Django
        form = DocumentForm(request.POST, request.FILES, user=request.user)
```

У результаті реалізації функціональних модулів захищеного веб-додатку було створено повнофункціональну систему управління доступом до конфіденційних

документів з використанням фреймворку Django. Розроблені модулі забезпечують диференційований доступ користувачів до документів відповідно до їх ролей у системі, де суперкористувачі мають повний доступ, адміністратори можуть керувати документами з обмеженими правами, а звичайні користувачі працюють лише з публічними та користувацькими документами.

3.3 Тестування функціональності захищеного веб-додатку управління доступом до конфіденційних документів

Метою тестування є перевірка надійності реалізованих механізмів інформаційної безпеки та функціональної стабільності системи в умовах потенційних загроз.

Першим етапом тестування є перевірка захисту від XSS-атак шляхом спроби введення шкідливого JavaScript-коду в поле імені користувача форми авторизації. Для проведення цього тестування використовується класичний XSS-вектор у вигляді коду <script>alert('XSS')</script>, який при успішному виконанні повинен був би відобразити спливаюче вікно в браузері користувача. Введення потенційно небезпечного коду здійснюється через стандартний веб-інтерфейс системи автентифікації (рис. 3.1).

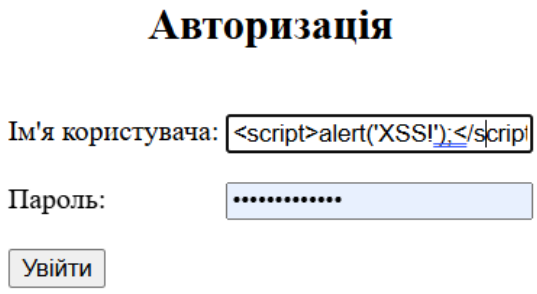


Рисунок 3.1 – Тестування захисту від XSS-атак через поле імені користувача

Результати тестування демонструють, що система коректно обробляє потенційно небезпечний вміст, запобігаючи виконанню шкідливого коду в браузері користувача. Механізми фільтрації та екранування вхідних даних працюють ефективно, не дозволяючи JavaScript-коду виконатися в контексті веб-сторінки.

Для підтвердження стабільності захисних механізмів здійснюється повторне тестування XSS-захисту з використанням того ж шкідливого коду. Додаткове інформаційне повідомлення для користувача вказує на необхідність введення коректних облікових даних для продовження роботи з системою (рис. 3.2).

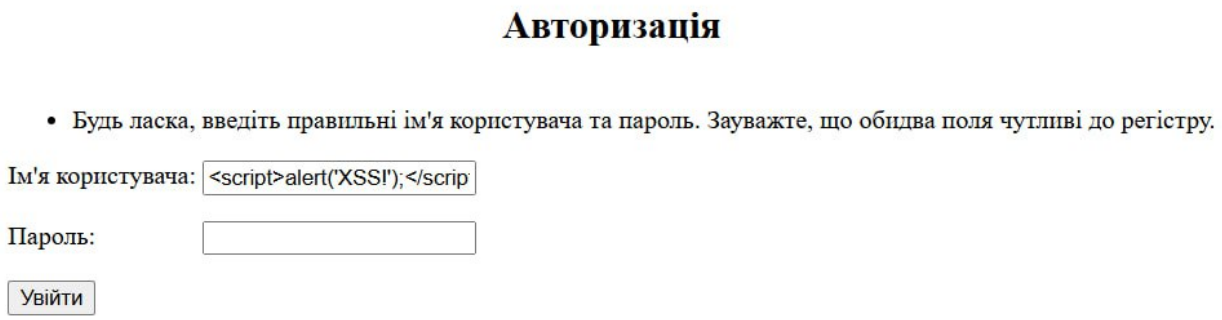


Рисунок 3.2 – Повторне тестування XSS-захисту з інформаційним повідомленням

Система продовжує демонструвати надійну роботу механізмів захисту, послідовно нейтралізуючи потенційні XSS-атаки через точки введення користувацьких даних. Повторне тестування підтверджує стабільність роботи системи захисту від XSS-атак та надійність реалізованих механізмів фільтрації і валідації користувацького вводу. Система зберігає свою функціональність навіть після спроб введення шкідливого коду.

Після успішної автентифікації з використанням коректних облікових даних система надає доступ до основного інтерфейсу управління документами. Головна сторінка відображає повний перелік доступних документів та функціональні можливості для їх управління, включаючи можливість створення нових документів та здійснення пошуку серед існуючих. Додатково проводиться тестування XSS-захисту через пошуковий рядок системи шляхом введення того ж шкідливого коду <script>alert('XSS')</script>. (рис. 3.3).

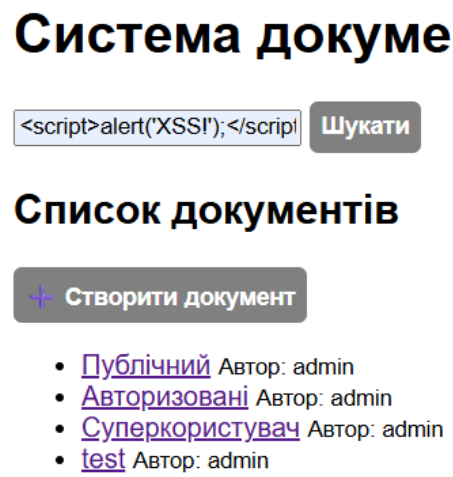


Рисунок 3.3 – Тестування XSS-захисту через пошуковий рядок системи

Нормальне функціонування інтерфейсу після проведення XSS-тестування підтверджує, що механізми захисту не впливають негативно на користувацький досвід та загальну функціональність системи. Всі елементи інтерфейсу залишаються доступними та працездатними.

Наступним етапом тестування є перевірка захисту від CSRF-атак через спробу несанкціонованого видалення документів. Підготовка до тестування включає налаштування POST-запиту на видалення конкретного документа без включення відповідного CSRF-токена в заголовки запиту. Використовується спеціалізований HTTP-клієнт для формування запиту з необхідними параметрами, що дозволяє детально контролювати структуру та зміст HTTP-запиту (рис. 3.4).

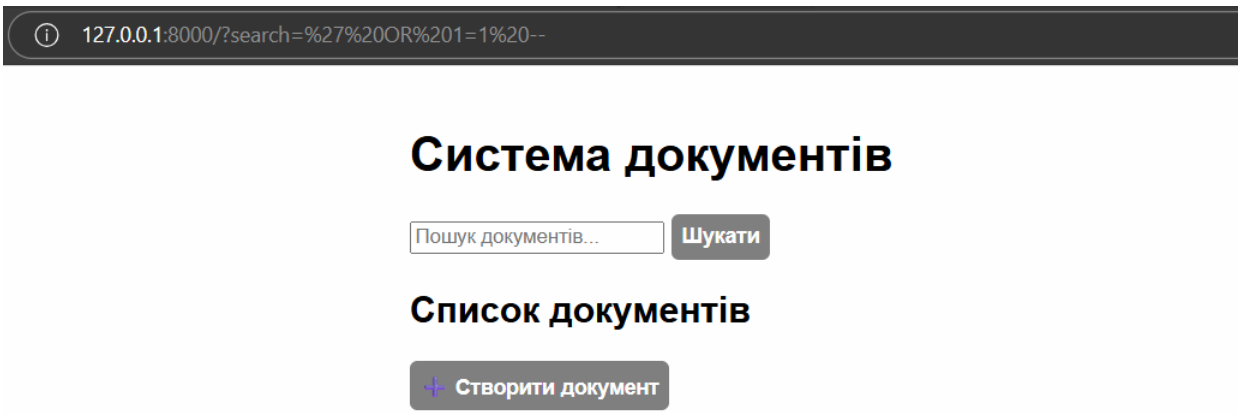


Рисунок 3.4 – Налаштування POST-запиту для тестування CSRF-захисту

Налаштування POST-запиту демонструє можливості детального аналізу поведінки системи при спробах CSRF-атак та дозволяє перевірити ефективність

реалізованих механізмів захисту в контрольованих умовах, а для конкретного маршруту видалення документів, який має структуру /delete/<int:doc_id> та призначений для обробки запитів на видалення документів за їх унікальним ідентифікатором налаштовані вбудовані механізми захисту від CSRF-атак, які повинні перевіряти наявність та валідність відповідного токена в кожному запиті.

Структура маршруту забезпечує чітку ідентифікацію документів для видалення та створює основу для перевірки захисних механізмів системи проти несанкціонованих міжсайтових запитів. Спроба SQL-ін'єкції через параметр пошуку з використанням конструкції "OR 1=1" (рис. 3.5).

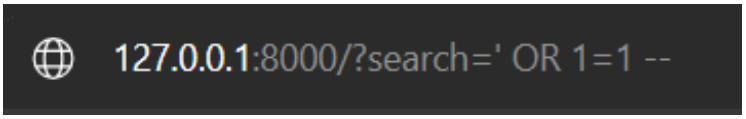


Рисунок 3.5 – Спроба SQL-ін'єкції через параметр пошуку

Типовим прикладом тестування захисту від SQL-ін'єкцій в пошуковій функціональності системи є спроба через параметр пошуку з використанням конструкції "OR 1=1". Система коректно обробляє такі спроби SQL-ін'єкції без компрометації безпеки даних. Тестування здійснено через локальний сервер розробки за адресою 127.0.0.1:8000, що забезпечує повний контроль над тестовим середовищем та дозволяє отримувати детальну інформацію про поведінку системи в умовах потенційних атак.

Результати тестування захисту від CSRF-атак підтверджують ефективність реалізованих механізмів захисту. При спробі виконання несанкціонованого POST-запиту на видалення документа без відповідного CSRF-токена система повертає код помилки 403 (Заборонено) з відповідним повідомленням "Помилка перевірки CSRF. Запит відхилений." Такий результат демонструє коректну роботу механізмів перевірки CSRF-токенів та ефективне запобігання несанкціонованим міжсайтовим запитам (рис. 3.6).

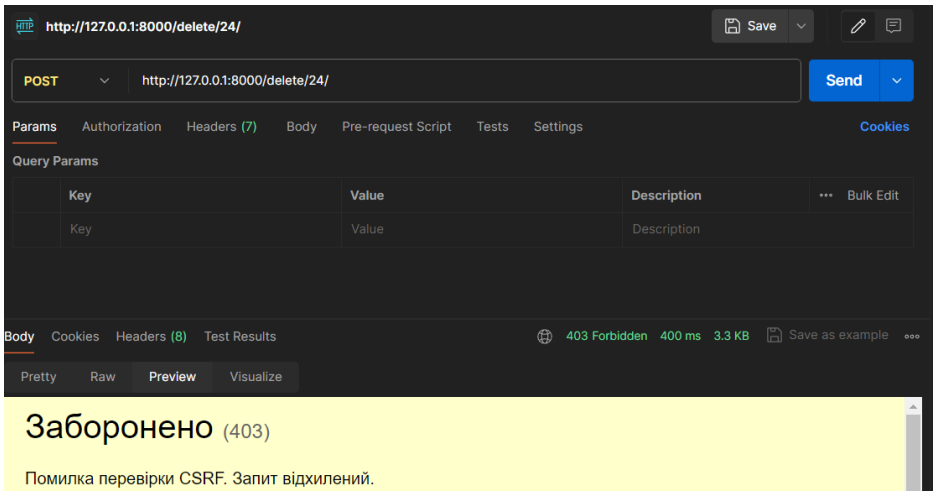


Рисунок 3.6 – Результат тестування CSRF-захисту з кодом помилки 403

Отримання коду помилки 403 та відповідного повідомлення про помилку підтверджує, що система надійно захищена від CSRF-атак та забезпечує належний рівень безпеки для операцій модифікації даних.

Комплексне тестування показало, що розроблений веб-додаток успішно протистоїть основним типам веб-атак та забезпечує надійний рівень безпеки для управління конфіденційними документами. Всі реалізовані механізми захисту функціонують коректно, не порушуючи при цьому користувацький досвід та функціональність системи. Результати тестування підтверджують, що система відповідає поставленим вимогам щодо інформаційної безпеки та може бути впроваджена для використання в реальних умовах з високим рівнем довіри до її захисних можливостей.

3.4 Інструкція користування інтерфейсом захищеного веб-додатку управління доступом до конфіденційних документів

Розроблений веб-добаток забезпечує інтуїтивно зрозумілий інтерфейс для управління доступом до конфіденційних документів із різними рівнями привілеїв користувачів. Система передбачає декілька типів користувачів: звичайні користувачі, адміністратори та суперкористувачі, кожен з яких має відповідні права доступу та функціональні можливості. Для доступу до системи необхідно пройти процедуру автентифікації через спеціальну форму входу. (рис. 3.8).

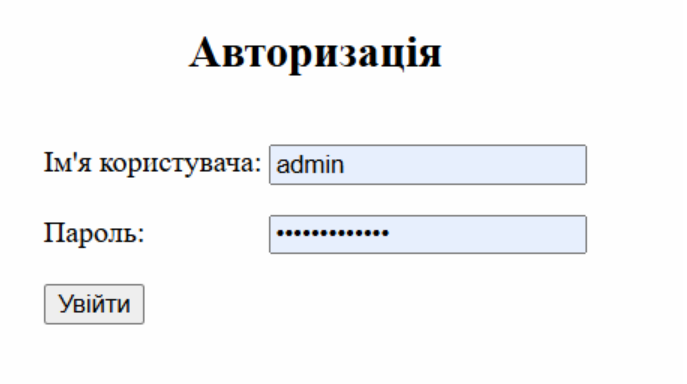


Рисунок 3.7 – Форма авторизації та автентифікації користувача

Інтерфейс авторизації містить поля для введення імені користувача та паролю, а також кнопку "Увійти" для підтвердження облікових даних та отримання доступу до системи. Форма авторизації забезпечує надійний механізм перевірки облікових даних користувача перед наданням доступу до конфіденційної інформації та функцій системи.

Початковий інтерфейс системи для звичайного адміністратора представляє собою головну сторінку з основними функціональними елементами управління документами. На головній сторінці розміщено поле пошуку документів, кнопку "Шукати" для активації пошукового запиту, а також список наявних документів з інформацією про їх авторів та рівні доступу (рис. 3.7).

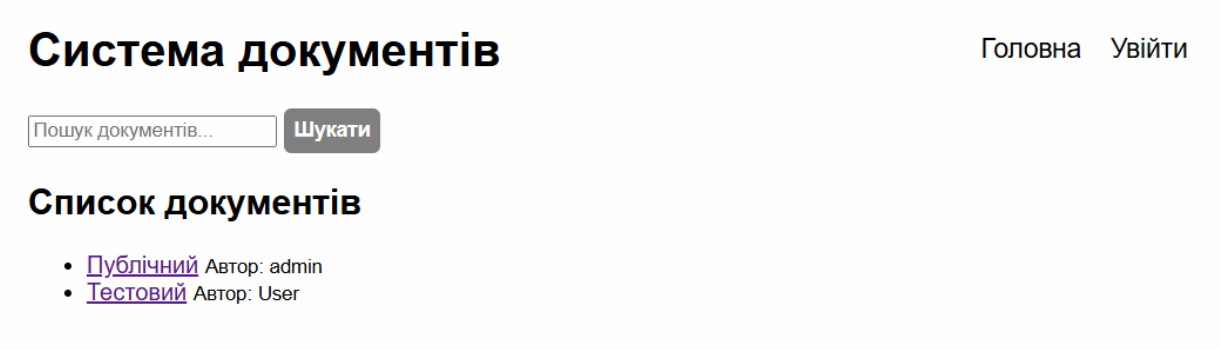


Рисунок 3.8 – Інтерфейс головної сторінки для звичайного адміністратора

Після завантаження головної сторінки користувач може ознайомитися з переліком доступних документів, серед яких присутні публічні та тестові документи з відповідною інформацією про авторство (рис. 3.9).

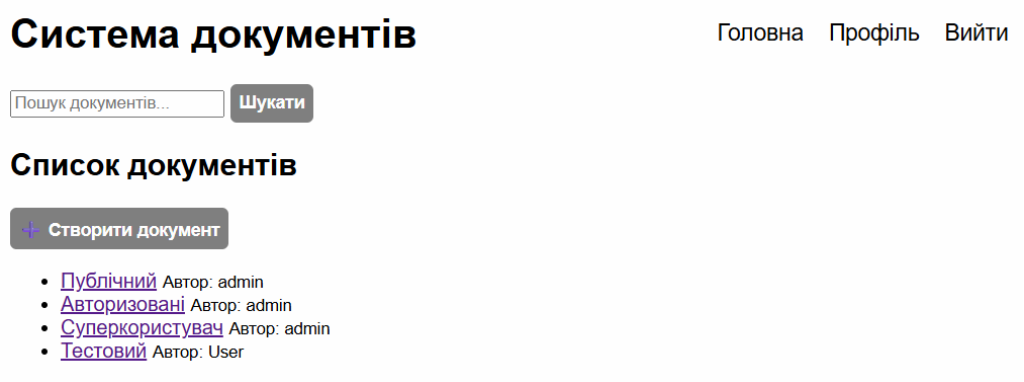


Рисунок 3.9 – Інтерфейс головної сторінки суперкористувача з розширеним функціоналом

На головній сторінці користувача відображається поле пошуку, кнопка "Створити документ" та список документів, доступних для перегляду згідно з налаштованими правами доступу, що ілюструється на (рис. 3.10).

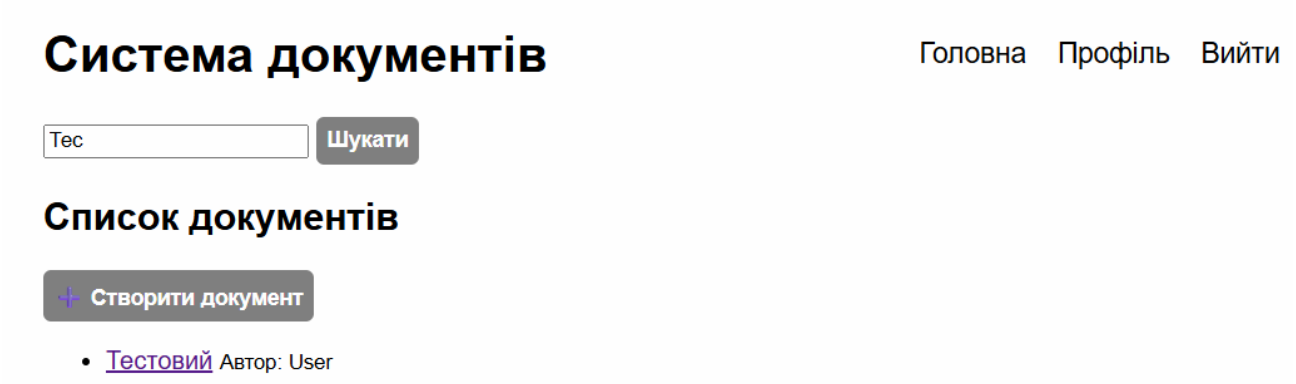


Рисунок 3.10 – Інтерфейс головної сторінки звичайного користувача

Обмежений інтерфейс звичайного користувача забезпечує доступ лише до тих документів та функцій, які відповідають його рівню авторизації в системі.

Детальний перегляд документа надає повну інформацію про його властивості та статус. Сторінка перегляду документа містить назву документа, інформацію про рівень доступу, автора та коментарі, а також функціональні кнопки "Завантажити" та "Видалити документ" для певних операцій з файлом, як показано на (рис. 3.11).

Інтерфейс перегляду документа забезпечує користувача всією необхідною інформацією для прийняття рішень щодо подальших дій з конфіденційним матеріалом.

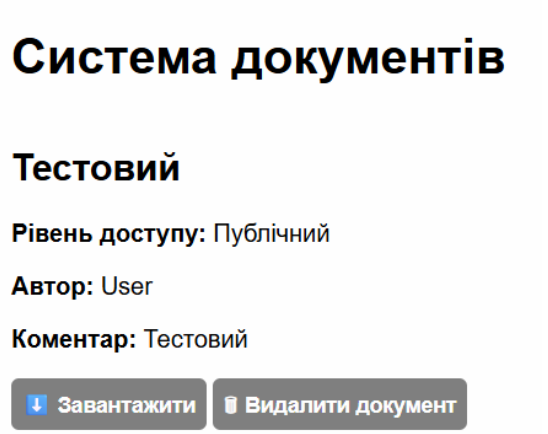


Рисунок 3.11 – Детальний перегляд документа з функціональними можливостями

Профіль суперкористувача включає розширені адміністративні функції для управління системою. Сторінка профілю відображає інформацію про поточного користувача, його роль у системі, можливість зміни паролю, а також розділ "Управління акаунтами" з переліком всіх користувачів системи та функціональними кнопками для їх редагування та видалення, що демонструється на (рис. 3.12).

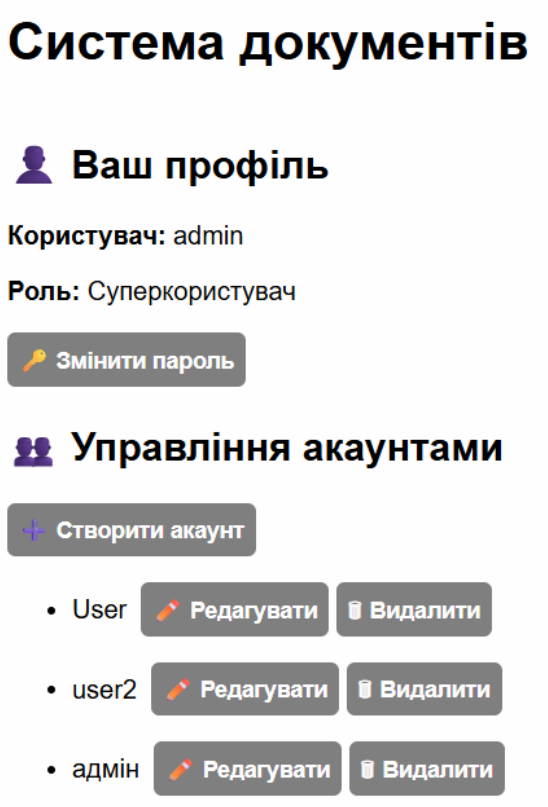


Рисунок 3.12 – Профіль суперкористувача з функціями управління акаунтами

Інтерфейс створення включає поля для введення назви документа, текстового коментаря, вибору рівня доступу з випадного меню, а також функцію завантаження файлу з локального сховища та кнопку "Зберегти документ" для збереження інформації, як представлено на (рис. 3.13).

Функціонал створення документів дозволяє користувачам гнучко налаштовувати параметри доступності та конфіденційності для кожного окремого матеріалу в системі.

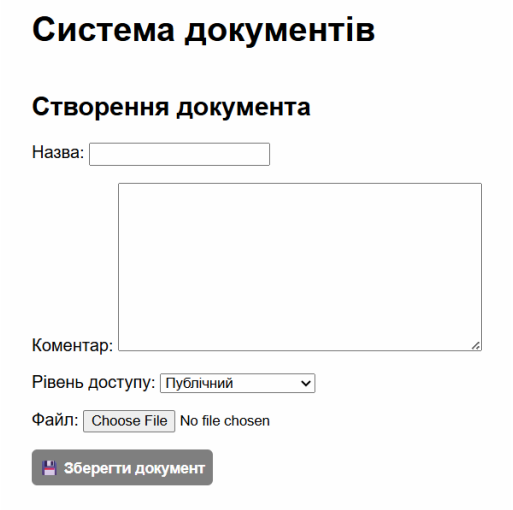


Рисунок 3.13 – Форма створення нового документа з налаштуваннями доступу

Інтерфейс зміни паролю забезпечує безпечний механізм оновлення облікових даних користувача. Форма зміни паролю містить поля для введення поточного паролю, нового паролю з відповідними вимогами безпеки, поле підтвердження нового паролю та інформативні повідомлення щодо політики паролів, завершуючись кнопкою "Зберегти новий пароль", що ілюструється на (рис. 3.14).

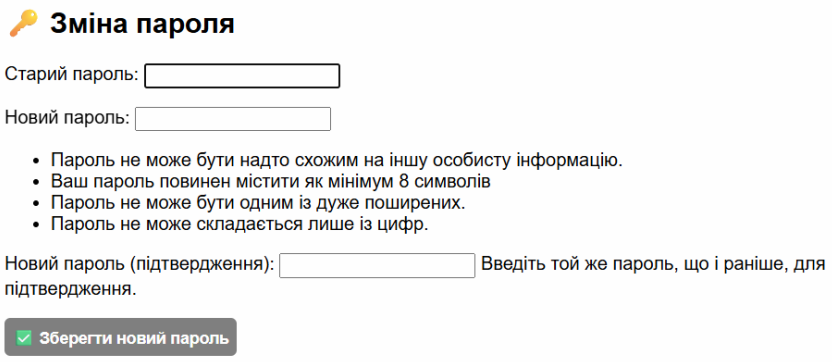


Рисунок 3.14 – Інтерфейс зміни паролю з вимогами безпеки

Система зміни паролю включає валідацію та перевірку відповідності

встановленим стандартам безпеки для забезпечення надійного захисту облікових записів. Форма створення нового акаунту надає суперкористувачам можливість розширення системи новими користувачами. Інтерфейс реєстрації включає поля для введення імені користувача з обмеженнями щодо допустимих символів, налаштування паролю з вимогами безпеки, підтвердження паролю, вибір рівня доступу з випадного меню та кнопку "Створити акаунт" для завершення процесу реєстрації, як показано на (рис. 3.15).

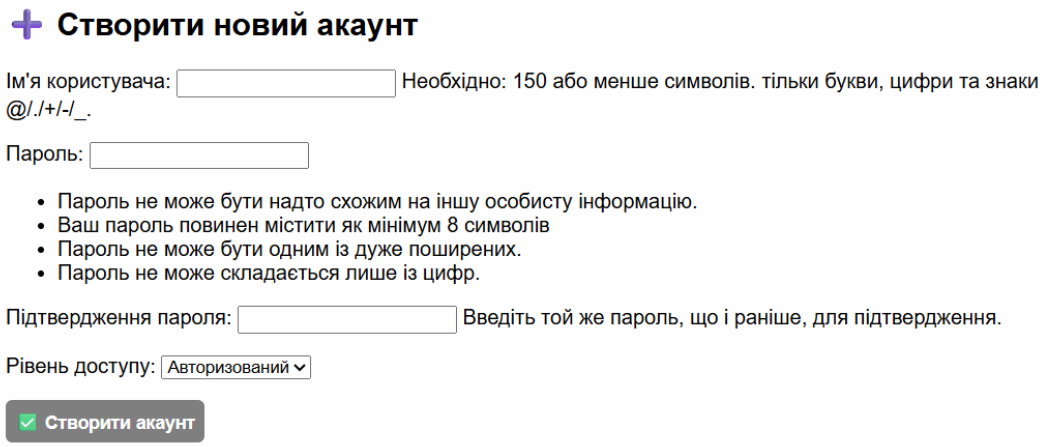


Рисунок 3.15 – Форма створення нового акаунту користувача

Функціонал створення акаунтів забезпечує гнучке управління користувацькою базою з можливістю призначення відповідних рівнів доступу відповідно до організаційних потреб. Форма редагування містить поля для зміни імені користувача з відповідними обмеженнями, налаштування рівня доступу через випадне меню та кнопку "Зберегти зміни" для підтвердження внесених модифікацій, що демонструється на (рис. 3.16).

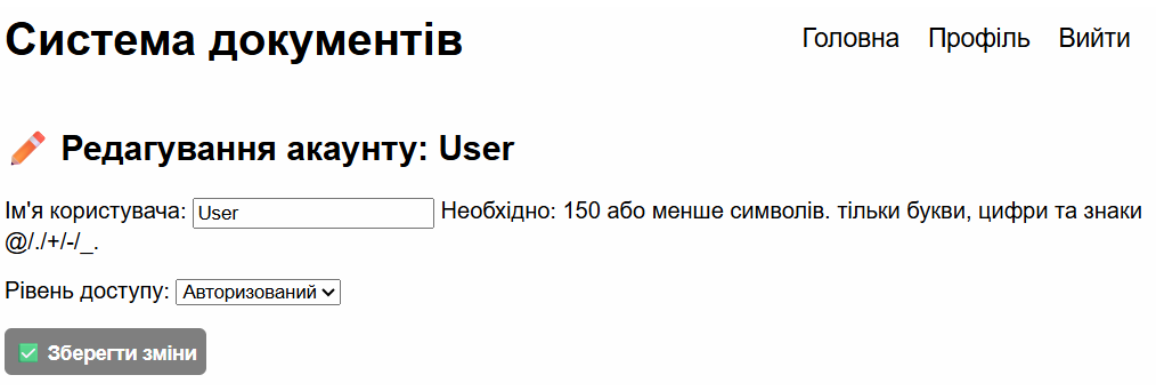


Рисунок 3.16 – Інтерфейс редагування акаунту користувача

Інтерфейс редагування існуючого акаунту дозволяє модифікацію параметрів користувачів системи.

Система характеризується гнучкими можливостями створення, перегляду та адміністрування документів з налаштуваннями конфіденційності, централізованим управлінням користувацькими акаунтами та надійними механізмами безпеки, включаючи валідацію паролів та контроль доступу.

3.5 Висновки до розділу 3

У третьому розділі було здійснено практичну реалізацію захищеного веб-додатку управління доступом до конфіденційних документів та проведено комплексне тестування його функціональних можливостей. Було обґрунтовано вибір сучасних технологій розробки, що забезпечують високий рівень безпеки та продуктивності системи. На етапі вибору інструментів розробки було проаналізовано переваги використання мови програмування Python з фреймворком Flask для створення програмної частини додатку. Обґрунтовано було базу даних SQLite як оптимальне рішення для зберігання структурованої інформації про користувачів та документи системи.

Розроблено було інтуїтивний користувацький інтерфейс, що забезпечує зручну навігацію та ефективну взаємодію з системою для користувачів різних рівнів доступу. Впроваджено було комплексну систему захисту від основних веб-загроз, включаючи захист від XSS-атак через валідацію та санітизацію вхідних даних, запобігання CSRF-атакам за допомогою токенів та використання параметризованих запитів для захисту від SQL-ін'єкцій.

Проведено було всебічне тестування функціональності системи, що підтвердило коректність роботи всіх модулів та ефективність впроваджених механізмів захисту. Результати тестування засвідчили, що розроблений веб-додаток забезпечує надійний захист конфіденційної інформації та відповідає сучасним стандартам інформаційної безпеки.

ВИСНОВОК

У першому розділі було розглянуто теоретичні основи захисту веб-додатків та конфіденційної інформації. Проаналізовано базові поняття інформаційної безпеки та конфіденційності, їх роль у забезпеченні надійного функціонування сучасних інформаційних систем. Вивчено основні загрози безпеці веб-додатків, зокрема атаки типу XSS, CSRF та SQL-ін'єкції, а також методи та засоби захисту від них. Розглянуто криптографічні методи захисту даних та проведено огляд існуючих рішень і систем управління документами. Досліджено специфіку роботи з конфіденційними документами та вимоги до їх безпечного зберігання й передачі.

На основі проведеного аналізу було визначено завдання розробки захищеного веб-додатку управління доступом до конфіденційних документів з використанням шифрування та механізмів захисту від основних веб-загроз, спроектувати архітектуру системи з рольовим контролем доступу та реалізувати програмний засіб, що автоматизує безпечний документообіг, а також протестувати й оцінити ефективність запропонованих механізмів захисту на практиці.

У другому розділі було запропоновано архітектуру захищеного веб-додатку управління доступом до конфіденційних документів із комплексним підходом до забезпечення інформаційної безпеки. Обґрунтовано постановку задачі та функціональні вимоги до системи захисту веб-додатка з урахуванням сучасних стандартів кібербезпеки. Спроектовано взаємодію веб-додатка з базою даних SQLite з використанням параметризованих запитів для запобігання SQL-ін'єкціям. Розроблено архітектуру веб-додатку з модульною структурою компонентів, що забезпечує ефективну взаємодію між клієнтською та серверною частинами системи. Використання рольового контролю доступу та криптографічних методів захисту підвищило рівень безпеки системи та забезпечило надійний захист конфіденційної інформації.

Останнім етапом роботи було практичне виконання захищеного веб-додатку управління доступом до конфіденційних документів з реалізацією всіх запропонованих механізмів захисту. У третьому розділі на основі розробленої

архітектури здійснено програмну реалізацію системи. Обґрунтовано вибір мови програмування Python з фреймворком Flask для серверної частини та сучасних веб-технологій для клієнтської частини, а також використання бази даних SQLite для зберігання структурованої інформації.

Розроблено веб-додаток з інтуїтивним користувацьким інтерфейсом, що забезпечує ефективну взаємодію користувачів різних рівнів доступу із системою. Також наведено детальну інструкцію користування інтерфейсом для різних категорій користувачів. Оцінено та проаналізовано коректність роботи програми, проведено комплексне тестування розробленого програмного продукту та зроблено висновок, що запропонована система забезпечує надійний захист від основних веб-загроз без негативного впливу на функціональність та зручність використання. Під час дослідження було проведено тестування ефективності механізмів захисту від XSS, CSRF атак та SQL-ін'єкцій з використанням розробленого веб-додатку. Оцінено вплив засобів захисту на продуктивність системи і підготовлено рекомендації для користувачів різних рівнів доступу.

Результатом роботи стала успішна розробка захищеного веб-додатку, що забезпечує автоматизоване управління доступом до конфіденційних документів із застосуванням сучасних методів шифрування та комплексного захисту від веб-загроз. Запропонована система дозволяє забезпечити конфіденційність, цілісність та доступність інформації в організаційному документообігу з гнучким рольовим контролем доступу.

Опираючись на усі проведені дослідження та враховуючи наведені результати роботи, можна вважати, що в даній бакалаврській дипломній роботі вдалося досягнути поставленої мети та виконати усі задачі, а саме – розроблено захищений веб-додаток управління доступом до конфіденційних документів із використанням шифрування та механізмів захисту від XSS, CSRF та SQL-ін'єкцій, що дозволяє забезпечити комплексний захист конфіденційної інформації в сучасних організаційних системах документообігу.

ПЕРЕЛІК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Stallings W. Cryptography and Network Security: Principles and Practice. – 8th ed. – Pearson, 2022. – 752 p.

2. OWASP Foundation. OWASP Top Ten Web Application Security Risks – 2021 [Електронний ресурс]. – Режим доступу: https://owasp.org/www-project-top-ten/

3. Stallings W. Cryptography and Network Security: Principles and Practice. – Boston: Pearson Education, 2017. – 752 p.

4. Whitman M. E., Mattord H. J. Principles of Information Security. – 7th ed. – Boston: Cengage Learning, 2021. – 576 p.

5. Peltier T. R. Information Security Risk Analysis. – 3rd ed. – Boca Raton: CRC Press, 2010. – 368 p.

6. Stallings W. Network Security Essentials: Applications and Standards. – 6th ed. – Boston: Pearson, 2017. – 480 p.

7. Гаврилюк В. В., Бачинський А. Я. Захист інформації в комп'ютерних системах і мережах. – Львів: «Новий Світ – 2000», 2015. – 324 с.

8. Закон України «Про захист персональних даних» № 2297-VI від 01.06.2010 р. (в редакції від 01.01.2024 р.) [Електронний ресурс]. – Режим доступу: https://zakon.rada.gov.ua/laws/show/2297-17

9. Calder A. IT Governance: An International Guide to Data Security and ISO27001/ISO27002. – 7th ed. – London: Kogan Page, 2020. – 384 p.

10. Andress J. The Basics of Information Security: Understanding the Fundamentals of InfoSec in Theory and Practice. – 3rd ed. – Amsterdam: Elsevier, 2019. – 240 p.

11. Бобало Ю. Я., Бачинський А. Я., Гуменний О. Я. Інформаційна безпека : навч. посіб. / Нац. ун-т «Львівська політехніка». Львів : Вид-во Львів. політехніки, 2019. 573 с.

12. Луцький М. Г., Антонов А. А., Бєлий М. П. Новітні технології захисту інформації : підручник / Нац. авіац. ун-т. Київ : НАУ, 2023. 310 с.

13. Остапов С. Є., Євсеєв С. П., Король О. Г. Кібербезпека: сучасні технології захисту : навч. посіб. Львів : Новий Світ – 2000, 2020. 678 с.

14. Хорошко В. О., Іващенко О. С., Литвин І. Р. Проєктування комплексних систем захисту інформації : підручник / Нац. ун-т «Львівська політехніка». Львів : Вид-во Львів. політехніки, 2020. 318 с.

15. OWASP Foundation. OWASP Top Ten – 2021: The Ten Most Critical Web Application Security Risks [Електронний ресурс]. – Режим доступу: https://owasp.org/Top10/

16. Філіппович В. І., Голованов Є. М. Безпека Web-додатків. Київ : Дія, 2020. 212 с.

17. Stuttard D., Pinto M. The Web Application Hacker's Handbook: Finding and Exploiting Security Flaws. 2nd ed. Indianapolis: Wiley, 2011. 912 p.

18. OWASP Foundation. OWASP CSRF Prevention Cheat Sheet [Електронний ресурс]. – Режим доступу: https://cheatsheetseries.owasp.org/Cross-Site_Request_Forgery_Prevention_Cheat_Sheet.html

19. Petrov A. Server-Side Request Forgery (SSRF): Understanding and Mitigating Attacks. – Infosec Resources. – 2021. [Електронний ресурс]. – Режим доступу: https://resources.infosecinstitute.com/topic/server-side-request-forgery-ssrf/

20. OWASP Foundation. OWASP SSRF Prevention Cheat Sheet [Електронний ресурс]. – Режим доступу: https://cheatsheetseries.owasp.org/cheatsheets/Server_Side_Request_Forgery_Prevention_Cheat_Sheet.html

21. Klein A. A Practical Guide to SSRF (Server-Side Request Forgery) Vulnerabilities. – PortSwigger Web Security Academy. – 2022. [Електронний ресурс]. – Режим доступу: https://portswigger.net/web-security/ssrf

22. Singh K. Server-Side Request Forgery (SSRF) Attacks: Explained. – Security Boulevard, 2020. [Електронний ресурс]. – Режим доступу: https://securityboulevard.com/2020/03/server-side-request-forgery-ssrf-attacks-explained/

23. Cisco Systems. Man-in-the-Middle Attack (MitM). – Cisco Secure.

[Електронний ресурс]. – Режим доступу: https://www.cisco.com/c/en/us/products/security/what-is-a-man-in-the-middle-attack.html

24. Kaspersky. What is a DoS Attack? Denial-of-Service Explained. – Kaspersky IT Encyclopedia. [Електронний ресурс]. – Режим доступу: https://www.kaspersky.com/resource-center/definitions/denial-of-service

25. Zargar S. T., Joshi J., Tipper D. A Survey of Defense Mechanisms Against Distributed Denial of Service (DDoS) Flooding Attacks. – IEEE Communications Surveys & Tutorials, vol. 15, no. 4, 2013, pp. 2046–2069.

26. OWASP. SQL Injection Prevention Cheat Sheet. [Електронний ресурс]. – Режим доступу: https://owasp.org/www-community/attacks/SQL_Injection

27. Chandankumar. SQL Injection – Attacks and Prevention. – Medium, 2022. [Електронний ресурс]. – Режим доступу: https://medium.com/@chandankumarmandal/sql-injection-attacks-and-prevention

28. Acunetix. What is Cross-site Scripting (XSS)? [Електронний ресурс]. – Режим доступу: https://www.acunetix.com/websitesecurity/cross-site-scripting

29. Mozilla Foundation. XSS (Cross-site scripting). – MDN Web Docs. [Електронний ресурс]. – Режим доступу: https://developer.mozilla.org/en-US/docs/Glossary/Cross-site_scripting

30. Django Project. CSRF Protection Middleware. – Django Documentation, v4.2. [Електронний ресурс]. – Режим доступу: https://docs.djangoproject.com/en/4.2/ref/csrf/

31. Microsoft Security Blog. Preventing SSRF in Cloud Applications. – 2021. [Електронний ресурс]. – Режим доступу: https://techcommunity.microsoft.com

32. Stallings W. Cryptography and Network Security. 7th ed. – Pearson, 2016. – 752 p.

33. NIST. Data Encryption Standard (DES). – FIPS PUB 46-3. – Gaithersburg: NIST, 1999.

34. Федорів Р. А. Основи криптографії. Київ : Наука, 2017. 228 с.

35. Щуров І. М. Сучасні криптографічні протоколи захисту даних. Львів :

ЛНУ, 2021. 146 с.

36. OWASP Foundation. Password Storage Cheat Sheet. – OWASP, 2023. [Online]. Available: https://cheatsheetseries.owasp.org/Password_Storage_Cheat_Sheet.html

37. OWASP. XSS, CSRF & Injection Defense Guide. – OWASP, 2022. [Електронний ресурс]. – Режим доступу: https://owasp.org/www-community/attacks

38. Google, Dropbox, Microsoft. Cloud SLA Comparison Chart. – 2024. [Електронний ресурс]. – Режим доступу: https://cloud.google.com/sla, https://www.dropbox.com, https://learn.microsoft.com

39. Microsoft. Enterprise File Sharing Security Comparison: SharePoint, Google Drive, Dropbox. – Microsoft TechDocs, 2023. [Електронний ресурс]. – Режим доступу: https://learn.microsoft.com

40. Omondiagbe O. P., Licorish S. A., MacDonell S. G. Features that predict the acceptability of Java and JavaScript answers on Stack Overflow. Proceedings of the 23rd International Conference on Evaluation and Assessment in Software Engineering. 2019.

41. Ertaul, Levent, Manpreet Kaur, and Venkata Arun Kumar R. Gudise. Implementation and performance analysis of pbkdf2, bcrypt, scrypt algorithms. Proceedings of the international conference on wireless networks (ICWN). The Steering Committee of The World Congress in Computer Science, Computer Engineering and Applied Computing (WorldComp), 2016.

42. Abbas A., Mohd M., Arshad S., Mahmod R. An efficient implementation of PBKDF2 with RIPEMD-160 on multiple FPGAs // Proceedings of the 2014 20th IEEE International Conference on Parallel and Distributed Systems (ICPADS). – Hsinchu, Taiwan: IEEE, 2014. – P. 944–949.

43. Abdullah, Ako Muhamad. Advanced encryption standard AES algorithm to encrypt and decrypt data. Cryptography and Network Security 16.1, 2017: 11.

44. Schneier B., Whiting D. A performance comparison of the five AES finalists // Proceedings of the 3rd AES Candidate Conference. – National Institute of Standards and Technology (NIST), 2000. – P. 15–34.

ДОДАТКИ

Додаток А. Технічне завдання

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

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

ТЕХНІЧНЕ ЗАВДАННЯ

на бакалаврську кваліфікаційну роботу студенту:
" Розробка захищеного web-додатку управління доступом до конфіденційних документів із використанням шифрування та механізмів захисту від XSS, CSRF та SQL ін'єкцій "

08-72.БКР.013.00.070 ТЗ

Керівник бакалаврської кваліфікаційної роботи
к.ф.-м.н, доцент
_____ Шиян А.А.
" ____ " _____

Вінниця – 2025 р.

1. **Найменування та область застосування**

Програмний засіб управління доступом до конфіденційних документів із застосуванням шифрування та захисту від XSS, CSRF та SQL-ін’єкцій.

2.Підстава для розробки

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

3. Мета та призначення розробки

3.1 Мета розробки: створення безпечного веб-додатку, що забезпечує керування доступом до конфіденційних документів з використанням криптографічного захисту та протидії поширеним веб-атакам.

3.2 Призначення: розроблений програмний засіб призначено для використання як система управління доступом до документів у середовищах, що потребують підвищеного рівня інформаційної безпеки.

4. Джерела розробки

4.1. ISO/IEC 27001:2013. Information technology – Security techniques – Information security management systems – Requirements. – Geneva: ISO, 2013. – 36 p.

4.2. OWASP Foundation. OWASP Top 10 Web Application Security Risks [Електронний ресурс]. – 2023. – Режим доступу: https://owasp.org/www-project-top-ten/

4.3. Закон України «Про захист персональних даних» №2297-VI від 01.06.2010 р. [Електронний ресурс]. – Режим доступу: https://zakon.rada.gov.ua/laws/show/2297-17

4.4. Єсін В. І., Кузнецов О. О., Сорока Л. С. Безпека інформаційних систем і технологій. – Харків: ХНУ імені В. Н. Каразіна, 2013. – 632 с.

5. Вимоги до програми

5.1 Вимоги до функціональних характеристик:

5.1.1 Програмний засіб повинен мати інтерфейс з авторизацією та управлінням ролями користувачів;

5.1.2 Реалізація методу не повинна вимагати спеціальних ліцензійних програмних додатків;

5.2 Вимоги до надійності:

5.2.1 Програмний засіб має функціонувати стабільно

5.2.3 Програмний засіб повинен коректно виконувати передбачені завдання.

5.3 Вимоги до складу і параметрів технічних засобів:

– процесор – від 1.5 ГГц;

– оперативна пам’ять – не менше 1 ГБ;

– середовище функціонування – Windows/Linux, Python 3.x, Flask, SQLite;

– вимоги до техніки безпеки при роботі з програмою повинні відповідати існуючим вимогам та стандартам з техніки безпеки при користуванні комп’ютерною технікою.

6. Вимоги до програмної документації

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

7. Вимоги до технічного захисту інформації

7.1 Необхідно забезпечити захист розроблюваного програмного засобу від несанкціонованого використання.

7.2 Неможливість отримання доступу незареєстрованих користувачів до інформаційних ресурсів.

8. Техніко-економічні показники

8.1 Цінність результатів використання даного проекту повинна перевищувати витрати на його реалізацію.

8.2 Має бути реалізований таким чином, щоб підходити для використання широкого загалу.

9. Стадії та етапи розробки

№ з/п	Назва етапів бакалаврської кваліфікаційної роботи	Початок	Закінчення
1.	Визначення напрямку бакалаврської роботи, формулювання теми	21.03.2025	23.03.2025
2.	Аналіз предметної області обраної теми	27.03.2025	05.04.2025
3.	Розробка алгоритму роботи	08.04.2025	12.04.2025
4.	Написання бакалаврської роботи на основі розробленої теми	15.05.2025	26.05.2025
5.	Передзахист бакалаврської кваліфікаційної роботи	26.05.2025	31.05.2025
6.	Виправлення, уточнення, корегування бакалаврської дипломної роботи	01.06.2025	04.06.2025
7.	Захист бакалаврської кваліфікаційної роботи	05.06.2025	10.06.2025

10. Порядок контролю та прийому

10.1 До приймання бакалаврської кваліфікаційної роботи надається:

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

Технічне завдання до виконання прийняв _____

Додаток Б. Лістинг коду розробленого захищеного веб-додатку

```
import os
from django.http import HttpResponse, HttpResponseForbidden
from django.shortcuts import redirect, render, get_object_or_404
from django.contrib.auth.forms import PasswordChangeForm
from django.contrib.auth.models import User
from documents.forms import DocumentForm, CustomUserCreationForm, EditUserForm
from secure_docs import settings
from .models import Document
from django.contrib.auth.views import LoginView
from django.contrib.auth.decorators import login_required
from django.db.models import Q

def home(request):
    search_query = request.GET.get('search', '')
    if request.user.is_superuser:
        if search_query:
            documents = Document.objects.filter(Q(title__icontains=search_query)) | Document.objects.filter(Q(uploaded_by__username__icontains=search_query))
        else:
            documents = Document.objects.all()  # Суперкористувач бачить все
    elif request.user.is_staff:
        if search_query:
            documents = Document.objects.filter(Q(title__icontains=search_query) & Q(access_level__in=['admin', 'user', 'public'])) | Document.objects.filter(Q(uploaded_by__username__icontains=search_query) & Q(access_level__in=['admin', 'user', 'public']))
        else:
            documents = Document.objects.filter(Q(access_level__in=['admin', 'user', 'public']))  # Адмін бачить все нижче свого рівня
    elif request.user.is_authenticated:
        if search_query:
            documents = Document.objects.filter(Q(title__icontains=search_query) & Q(access_level__in=['user', 'public'])) | Document.objects.filter(Q(uploaded_by__username__icontains=search_query) & Q(access_level__in=['user', 'public']))
        else:
            documents = Document.objects.filter(Q(access_level__in=['user', 'public']))  # Авторизовані користувачі бачать "user" і "public"
    else:
        if search_query:
            documents = Document.objects.filter(Q(title__icontains=search_query) & Q(access_level='public')) | Document.objects.filter(Q(uploaded_by__username__icontains=search_query) & Q(access_level='public'))
        else:
            documents = Document.objects.filter(Q(access_level='public'))  # Неавторизовані користувачі бачать тільки "public"

    return render(request, 'home.html', {'documents': documents})

def document_detail(request, doc_id):
    document = get_object_or_404(Document, id=doc_id)

    if not document.can_user_view(request.user):
        return HttpResponseForbidden("Вам заборонено перегляд цього документа.")

    return render(request, 'document_detail.html', {'document': document})
```

```
class CustomLoginView(LoginView):
  template_name = 'login.html'

  def dispatch(self, request, *args, **kwargs):
    if request.user.is_authenticated:
      return redirect('')
    return super().dispatch(request, *args, **kwargs)

@login_required
def create_document(request):
  if request.method == "POST":
    form = DocumentForm(request.POST, request.FILES, user=request.user)
    if form.is_valid():
      document = form.save(commit=False)
      document.uploaded_by = request.user
      document.owner = request.user  # Встановлюємо власника документа
      document.comment = form.cleaned_data.get('comment')

      # Шифруємо файл перед збереженням
      uploaded_file = request.FILES.get('file')
      if uploaded_file:
        document.encrypt_and_save_file(uploaded_file)

      document.save()
def encrypt_and_save_file(self, uploaded_file):
    from cryptography.fernet import Fernet
    key = Fernet.generate_key()
    cipher_suite = Fernet(key)
    file_content = uploaded_file.read()
    encrypted_content = cipher_suite.encrypt(file_content)
    self.encryption_key = key
    self.filename = uploaded_file.name
  def decrypt_file(self):
    try:
      from cryptography.fernet import Fernet
      cipher_suite = Fernet(self.encryption_key)
      with open(self.encrypted_file_path, 'rb') as encrypted_file:
        encrypted_content = encrypted_file.read()
      decrypted_content = cipher_suite.decrypt(encrypted_content)
      return decrypted_content
    except Exception:
      return None
  def home(request):
    search_query = request.GET.get('search', '')
    if request.user.is_superuser:
      if search_query:
        documents = Document.objects.filter(Q(title__icontains=search_query)) |
Document.objects.filter(Q(uploaded_by__username__icontains=search_query))
      else:
        documents = Document.objects.all()
  documents = Document.objects.filter(
    Q(title__icontains=search_query) & Q(access_level__in=['admin', 'user', 'public'])
  ) | Document.objects.filter(
    Q(uploaded_by__username__icontains=search_query) & Q(access_level__in=['admin', 'user', 'public'])
  )
```

```
            # Зберігаємо шифрований документ
            return redirect('home')
    else:
        form = DocumentForm(user=request.user)
    return render(request, 'create_document.html', {'form': form})

def download_document(request, doc_id):
    document = get_object_or_404(Document, id=doc_id)

    # Перевірка прав доступу
    if not document.can_user_view(request.user):
        return HttpResponse("У вас немає доступу до цього файлу", status=403)

    if not document.encrypted_file_path:
        return HttpResponse("Файл відсутній", status=404)

    decrypted_content = document.decrypt_file()

    if decrypted_content is None:
        return HttpResponse("Помилка розшифрування файлу", status=500)

    response = HttpResponse(decrypted_content, content_type="application/octet-stream")
    response["Content-Disposition"] = f'attachment; filename="{document.filename}"'

    return response

@login_required
def delete_document(request, doc_id):
    document = get_object_or_404(Document, id=doc_id)

    if (document.uploaded_by != request.user or request.user.is_superuser) and not request.user.is_superuser:
        return HttpResponseForbidden("Ви не маєте прав для видалення цього документа.")

    if document.encrypted_file_path:
        file_path = os.path.join(settings.MEDIA_ROOT, document.encrypted_file_path)
        if os.path.exists(file_path):
            os.remove(file_path)

    document.delete()
    return redirect('home')  # Після видалення повернення на головну сторінку

@login_required
def profile(request):
    if request.user.is_superuser:  # Суперкористувачі бачать всіх
        # do not show superusers in the list
        users = User.objects.exclude(is_superuser=True)
    elif request.user.is_staff:  # Адміністратори бачать лише авторизованих користувачів
        users = User.objects.filter(is_staff=False, is_superuser=False)
    else:  # Звичайні користувачі нічого не бачать
        users = User.objects.none()
    return render(request, 'profile.html', {'user': request.user, 'profiles': users})

@login_required
def change_password(request):
    if request.method == 'POST':
        form = PasswordChangeForm(request.user, request.POST)
        if form.is_valid():
            form.save()
```

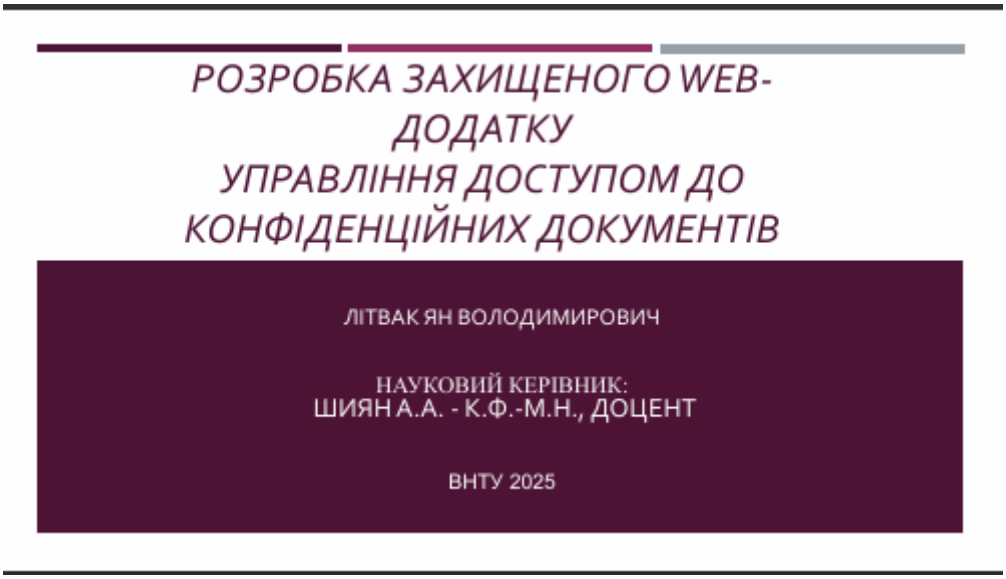
```
            return redirect('home')
    else:
        form = PasswordChangeForm(request.user)
    return render(request, 'change_password.html', {'form': form})

@login_required
def create_user(request):
    if not request.user.is_superuser and not request.user.is_staff:
        return HttpResponseForbidden("У вас немає прав для створення акаунтів.")

    if request.method == "POST":
        form = CustomUserCreationForm(request.POST)
        if form.is_valid():
            form.save()
            return redirect('profile')  # Повертаємося до профілю після створення
    else:
        form = CustomUserCreationForm()

    return render(request, 'create_user.html', {'form': form})

@login_required
def edit_user(request, user_id):
    if not request.user.is_superuser and not request.user.is_staff:
        return HttpResponseForbidden("У вас немає прав для редагування акаунтів.")

    user = get_object_or_404(User, id=user_id)

    if request.method == 'POST':
        form = EditUserForm(request.POST, instance=user)
        if form.is_valid():
            form.save()
            return redirect('profile')  # Повертаємося до профілю після редагування
    else:
        form = EditUserForm(instance=user)

    return render(request, 'edit_user.html', {'form': form, 'user': user})

@login_required
def delete_user(request, user_id):
    if not request.user.is_superuser and not request.user.is_staff:
        return HttpResponseForbidden("У вас немає прав для видалення акаунтів.")

    user = get_object_or_404(User, id=user_id)

    # Забороняємо видаляти суперкористувачів
    if user.is_superuser:
        return HttpResponseForbidden("Ви не можете видаляти суперкористувачів.")

    if request.user.is_staff and not request.user.is_superuser and user.is_staff:
        return HttpResponseForbidden("Ви не можете видаляти інших адміністраторів.")

    user.delete()
    return redirect('profile')
from django import forms
from .models import Document
from django.contrib.auth.models import User
from django.contrib.auth.forms import UserCreationForm
```

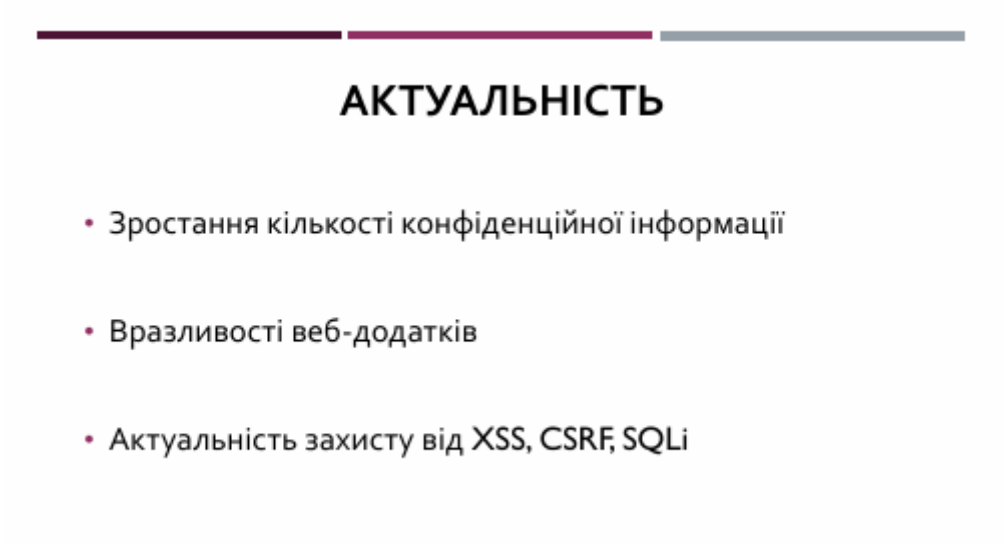
```
class DocumentForm(forms.ModelForm):
    file = forms.FileField(required=False)  # Поле для завантаження файлу

    class Meta:
        model = Document
        fields = ['title', 'comment', 'access_level', 'file']

    def __init__(self, *args, **kwargs):
        user = kwargs.pop('user', None)  # Отримуємо користувача
        super().__init__(*args, **kwargs)

        # Фільтрація варіантів рівня доступу
        if user:
            allowed_levels = ['public', 'user']  # Базові рівні доступу
            if user.is_staff:
                allowed_levels.append('admin')
            if user.is_superuser:
                allowed_levels.append('superuser')

            self.fields['access_level'].choices = [
                (level, label) for level, label in Document.ACCESS_LEVELS if level in allowed_levels
            ]

class CustomUserCreationForm(UserCreationForm):
    role = forms.ChoiceField(
        choices=[('user', 'Авторизований'), ('admin', 'Адміністратор')],
        required=True
    )

    class Meta:
        model = User
        fields = ['username', 'password1', 'password2']

    def save(self, commit=True):
        user = super().save(commit=False)
        role = self.cleaned_data['role']

        if role == 'admin':
            user.is_staff = True  # Робимо користувача адміністратором

        if commit:
            user.save()
        return user

class EditUserForm(forms.ModelForm):
    role = forms.ChoiceField(
        choices=[('user', 'Авторизований'), ('admin', 'Адміністратор')],
        required=True
    )
    class Meta:
        model = User
        fields = ['username', ]

    def __init__(self, *args, **kwargs):
        user = kwargs.pop('user', None)
        super().__init__(*args, **kwargs)

        if self.instance.is_staff:
```

```
            self.initial['role'] = 'admin'
        else:
            self.initial['role'] = 'user'

    def save(self, commit=True):
        user = super().save(commit=False)
        role = self.cleaned_data['role']
        user.is_staff = (role == 'admin')  # Оновлюємо статус адміністратора
        if commit:
            user.save()
        return user
from django.db import models
from django.contrib.auth.models import User
from cryptography.fernet import Fernet
import os
from secure_docs import settings
import time
cipher = Fernet(settings.SECRET_ENCRYPTION_KEY)

class Document(models.Model):
    ACCESS_LEVELS = [
        ('public', 'Публічний'),
        ('user', 'Для авторизованих'),
        ('admin', 'Для адміністраторів'),
        ('superuser', 'Суперкористувач'),
    ]

    title = models.CharField(max_length=255)
    content = models.TextField()
    access_level = models.CharField(max_length=20, choices=ACCESS_LEVELS, default='public')
    owner = models.ForeignKey(User, on_delete=models.CASCADE, related_name='owned_documents')
    uploaded_by = models.ForeignKey(User, on_delete=models.CASCADE,
related_name='uploaded_documents')
    comment = models.TextField(blank=True, null=True)  # Коментар до документа
    encrypted_file_path = models.CharField(max_length=500, blank=True, null=True)
    filename = models.CharField(max_length=255, blank=True, null=True)  # Ім'я файлу

    def encrypt_and_save_file(self, uploaded_file):
        """Шифрує файл і зберігає його у папку"""
        encrypted_content = cipher.encrypt(uploaded_file.read())
        self.filename = uploaded_file.name
        encrypted_filename = f"{time.time()}.enc"
        file_path = os.path.join(settings.MEDIA_ROOT, encrypted_filename)

        with open(file_path, "wb") as f:
            f.write(encrypted_content)

        self.encrypted_file_path = f"{encrypted_filename}"
    def decrypt_file(self):
        """Розшифровує файл для скачування"""
        if not self.encrypted_file_path:
            return None

        file_path = os.path.join(settings.MEDIA_ROOT, self.encrypted_file_path)
        with open(file_path, "rb") as f:
            encrypted_content = f.read()

        decrypted_content = cipher.decrypt(encrypted_content)
```

```
            return decrypted_content

        def can_user_view(self, user):
            """ Перевірка, чи користувач може переглядати документ """
            if user.is_superuser:
                return True  # Суперкористувач бачить все
            if user.is_staff and self.access_level in ['admin', 'user', 'public']:
                return True  # Адміністратор бачить документи для адмінів, користувачів та публічні
            if self.access_level == 'user' and user.is_authenticated:
                return True  # Авторизований користувач бачить "user" і "public"
            if self.access_level == 'public':
                return True  # Усі можуть бачити "public"
            return False  # Усі інші випадки — доступ заборонено
    from django.urls import path
    from django.contrib.auth import views as auth_views
    from documents.views import create_document, delete_document, delete_user, document_detail,
download_document, edit_user, home, profile, change_password, create_user
    from documents.views import CustomLoginView

    urlpatterns = [
        path('', home, name='home'),
        path('login/', CustomLoginView.as_view(), name='login'),
        path('logout/', auth_views.LogoutView.as_view(), name='logout'),
        path('document/<int:doc_id>/', document_detail, name='document_detail'),
        path('download/<int:doc_id>/', download_document, name='download_document'),
        path('create/', create_document, name='create_document'),
        path('delete/<int:doc_id>/', delete_document, name='delete_document'),
        path('profile/', profile, name='profile'),
        path('change-password/', change_password, name='change_password'),
        path('create-user/', create_user, name='create_user'),
        path('edit-user/<int:user_id>/', edit_user, name='edit_user'),
        path('delete-user/<int:user_id>/', delete_user, name='delete_user'),
    ]
```

Додаток В. Ілюстративний матеріал





МЕТА ТА ЗАДАЧІ ДОСЛІДЖЕННЯ

Мета: розробити захищений веб-додаток

Задачі:

- Захист від типових атак
- Шифрування файлів
- Контроль доступу

ОБ'ЄКТ, ПРЕДМЕТ І НОВИЗНА

- Об'єкт – доступ до конфіденційних документів
- Предмет – методи захисту в веб-додатках
- Новизна – багаторівнева система безпеки

АКТУАЛЬНІ ЗАГРОЗИ

- SQL Injection
- Cross-Site Scripting (XSS)
- Cross-Site Request Forgery (CSRF)
- SSRF, DoS/DDoS

МЕТОДИ ЗАХИСТУ

- Валідація та фільтрація даних
- CSRF-токени, CSP
- Prepared statements
- Обмеження прав

КРИПТОГРАФІЯ

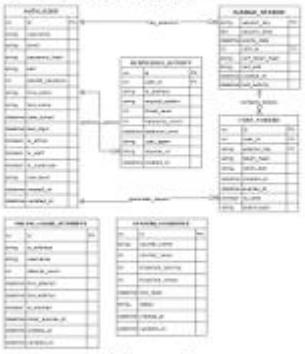
- AES-256
- PBKDF2
- SHA-256, соль
- Зберігання ключів

АРХІТЕКТУРА ДОДАТКУ

- Клієнт-серверна модель
- Модулі: інтерфейс, сервер, БД
- Ролі: користувач, адмін, суперкористувач

БАЗА ДАНИХ ТА БЕЗПЕКА

- SQLite + SQLCipher
- Розмежування доступу
- Захищене зберігання ключів



Структура ER діаграми бази даних, задіяної у захищеного веб додатку, управління користувачами та сесіями

РЕАЛІЗАЦІЯ ДОДАТКУ

- Python, Django
- ORM-запити
- Захист форм і запитів
- Шифрування файлів

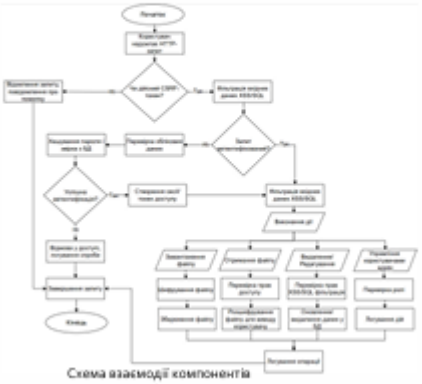


Схема взаємодії компонентів архітектури розробленого захищеного веб-додатку

ІНТЕРФЕЙС ТА ТЕСТУВАННЯ

- Авторизація, завантаження, доступ
- Перевірка безпеки
- Захист від атак у дії

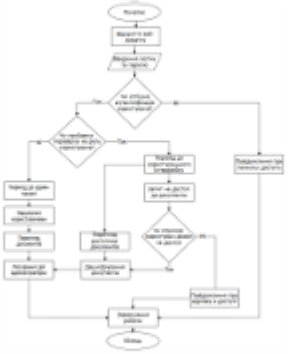


Схема взаємодії користувача з функціоналом розробленого веб-додатку

ВИСНОВКИ

- Реалізовано захищений веб-додаток
- Система стійка до атак
- Можливе практичне використання

ПРОТОКОЛ ПЕРЕВІРКИ НАВЧАЛЬНОЇ (КВАЛІФІКАЦІЙНОЇ) РОБОТИ

Назва роботи:

Розробка захищеного web-додатку управління доступом до конфіденційних документів із використанням шифрування та механізмів захисту від XSS, CSRF та SQL ін'єкцій

Тип роботи: бакалаврська кваліфікаційна робота

Підрозділ Кафедра менеджменту на безпеки інформаційних систем
Факультет менеджменту та інформаційної безпеки
Гр. 1КІТС-21б

Керівник доцент Шиян А.А. _____

Показники звіту подібності

Strike Plagiarism	
Оригінальність	91,09 %
Загальна схожість	8,91 %

Аналіз звіту подібності (відмітити потрібне)

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

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

Автор _____ Літвак Я.В.
(підпис) (прізвище, ініціали)

Опис прийнятого рішення

- ☐ **Допустити до захисту**

_____
_____
_____
_____

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

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