МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет кораблебудування імені адмірала Макарова
Навчально-науковий інститут автоматики і електротехніки
Кафедра комп'ютерних технологій та інформаційної безпеки

«ДОПУЩЕНИЙ ДО ЗАХИСТУ»
Завідувач кафедри комп'ютерних технологій та інформаційної безпеки, к.т.н., доцент
_____ С.М. Нужний
«17» 06 2026р.

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття ступеня вищої освіти «бакалавр»

РОЗРОБКА ТА ДОСЛІДЖЕННЯ МЕТОДУ ЗАХИСТУ ВЕБ-ЗАСТОСУНКІВ ВІД ТИПОВИХ ВЕБ-АТАК

Галузь знань: 12 «Інформаційні технології»
Спеціальність: 125 «Кібербезпека та захист інформації»
Освітньо-професійна програма: «Системи технічного захисту інформації, автоматизація її обробки»

Виконала: студентка 4381 групи
Ярута Ольга Вікторівна _____
(прізвище та ініціали) (підпис)

Керівник роботи: к.т.н., доцент кафедри комп'ютерних технологій та інформаційної безпеки
Сірівчук А.С. _____
(прізвище та ініціали) (підпис)

Миколаїв – 2026

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

Національний університет кораблебудування імені адмірала Макарова

Навчально-науковий інститут автоматики та електротехніки

Кафедра комп'ютерних технологій та інформаційної безпеки

Спеціальність **125«Кібербезпека та захист інформації»**
Освітня програма **«Системи технічного захисту інформації, автоматизація її обробки»**

ЗАТВЕРДЖУЮ
Гарант освітньої програми, к.т.н.,
доцент, доцент кафедри КТІБ
_____ С.М. Нужний
«10» ____02____ 2026р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ
на здобуття ступеня вищої освіти «бакалавр»

Студентці Яруті Ользі Вікторівні
(Прізвище, ім'я, по батькові)

1. Тема роботи: Розробка та дослідження методу захисту веб-застосунків від типових веб-атак.

Керівник роботи к.т.н., доцент кафедри комп'ютерних технологій та інформаційної безпеки, Сірівчук Андрій Сергійович

Затверджені наказом ректора №286-ус від 21.04.2026

2. Термін подання роботи: 5.06.2026
3. Вихідні дані по роботі: тип вразливостей SQL-ін'єкції, XSS, CSRF і помилки автентифікації

4. Перелік питань, що належать до розробки (найменування розділів):
Аналіз сучасних веб-застосунків та загроз безпеці/ Дослідження існуючих методів та засобів захисту / Розробка методів захисту веб-застосунку / Експериментальне дослідження та оцінка ефективності
5. Перелік презентаційних матеріалів: Титульний лист/ Мета та завдання / Узагальнена архітектура сучасного веб-застосунку / Основні технології функціонування веб-застосунків/ Класифікація типових кіберзагроз для веб-застосунків / Методи захисту веб-застосунку/ Процес обробки запиту / Демонстрація тестового застосунку / Тестування системи безпеки/ Висновки

6. Консультант - внутрішній рецензент роботи

Прізвище, ініціали та посада	Дата подання роботи на рецензію	Дата отримання рецензії	Підпис
Передрій В.І. професор, д.т.н.	4.06.26	5.06.26	

7. Дата видачі завдання _____

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів роботи	Терміни виконання	Примітка
1	Вибір теми, визначення мети, завдань, об'єкта та предмета дослідження	10.02.2026	
2	Попередній варіант оглядових розділів, підбір і опрацювання списку використаних джерел	20.03.2026	
3	Попередній варіант повної КР	15.04.2026	
4	Складання ЄДКІ зі спеціальності на ступінь «бакалавра»	16.04.2026 14.05.2026	
5	Попередній варіант презентації	29.05.2026	
6	Подання на кафедру КТІБ тексту остаточного варіанту роботи, підписаного її керівником в електронному форматі разом із заявами щодо самостійності виконання роботи та ідентичності друкованої та електронної версій роботи (Додатки 1 і 2 «Порядку здійснення заходів з перевірки робіт на наявність текстових збігів/ідентичності/схожості із використанням програмно-технічних засобів», який введений в дію наказом ректора НУК за №20 від 20.01.2020) Отримання внутрішньої рецензії	05.06. 2026	не пізніше, ніж за 14 діб до захисту
7	Отримання висновку керівника; Отримання висновку кафедри щодо наявності / відсутності плагіату в роботі Попередній захист роботи на засіданні кафедри КТІБ Висновок кафедри про КР, допуск до захисту	10.06.2026	
8	Отримання зовнішньої рецензії	15.06.2026	
9	Підготовка презентації та доповіді	17.06.2026	
10	Подання на кафедру КТІБ: – друкованого і зшитого варіанту пояснювальної записки до кваліфікаційної роботи та/або (за рішенням кафедри) електронного варіанту в форматі *.pdf – друкованого варіанту презентації (в 5-ти екземплярах в разі захисту оф-лайн, в одному екземплярі в разі захисту он-лайн (за рішенням кафедри)) та електронного варіанту презентації; – зовнішньої рецензії; – документів щодо відсутності плагіату і передачі авторських прав; – файлу-опису кваліфікаційної роботи (згідно Додатку до наказу ректора НУК за №287-уч від 19.05.2020 р.).	22.06.2026	
11	Захист кваліфікаційної роботи	24 - 25. 06.2026	

Студентка _____ (підпис) О.В. Ярута (ПІБ)

Керівник роботи _____ (підпис) А.С. Сірівчук (ПІБ)

АНОТАЦІЯ

Ярута Ольга Вікторівна. Дослідження та розробка методів захисту веб-застосунків від типових веб-атак. Кваліфікаційна робота бакалавра за спеціальністю 125 Кібербезпека та захист інформації, галузь знань 12 Інформаційні технології, освітньо-професійна програма «Системи технічного захисту інформації, автоматизація її обробки». Національний університет кораблебудування імені адмірала Макарова. Миколаїв, 2026. Пояснювальна записка містить 60 с., 12 рис., 3 табл..

У кваліфікаційній роботі досліджено методи захисту веб-застосунків від типових веб-атак. Актуальність теми зумовлена широким використанням веб-застосунків та ризиками, пов'язаними з *SQL*-ін'єкціями, *XSS, CSRF* і помилками автентифікації. Метою роботи є дослідження типових веб-загроз і розробка програмних механізмів захисту веб-застосунку. Об'єктом дослідження є процес захисту веб-застосунків від веб-атак. Предметом дослідження є методи перевірки введених даних, захищеної роботи з базою даних, захисту сесій, *CSRF*-захисту та хешування паролів.

У роботі використано методи аналізу архітектури веб-застосунків, класифікації кіберзагроз, моделювання *HTTP*-запитів, програмної реалізації захисного модуля та експериментального тестування.У процесі виконання роботи проаналізовано архітектуру сучасних веб-застосунків, основні принципи веб-технологій і типові кіберзагрози. Розглянуто існуючі алгоритми захисту та розроблено демонстраційний веб-застосунок *Secure Notes* із механізмами фільтрації введення, параметризованими *SQL*-запитами, *CSRF*-токенами, хешуванням паролів, захистом сесій і журналюванням подій безпеки. Проведено моделювання веб-атак і перевірено працездатність реалізованих механізмів захисту. Практичне значення роботи полягає у створенні демонстраційного веб-застосунку, який може бути використаний для навчального тестування базових методів захисту від типових веб-атак.

Ключові слова: веб-застосунок, кібербезпека, *SQL*-ін'єкція, *XSS, CSRF*, автентифікація, хешування паролів, *Flask, SQLite, Secure Notes*.

ANNOTATION

Yaruta Olha Viktorivna. Research and Development of Methods for Protecting Web Applications Against Common Web Attacks. Bachelor's qualification work in specialty 125 Cybersecurity and Information Protection, field of knowledge 12 Information Technologies, educational and professional program "Systems of Technical Information Protection and Automation of Its Processing". Admiral Makarov National University of Shipbuilding. Mykolaiv, 2026.The explanatory note contains 60 pages, 12 figures, 3 tables.

The qualification work studies methods for protecting web applications against common web attacks. The relevance of the topic is determined by the widespread use of web applications and the risks associated with SQL injection, XSS, CSRF, and authentication errors. The aim of the work is to study common web threats and develop software mechanisms for protecting a web application. The object of the research is the process of protecting web applications against web attacks. The subject of the research is methods of input validation, secure database interaction, session protection, CSRF protection, and password hashing.

The work uses methods of web application architecture analysis, cybersecurity threat classification, HTTP request modeling, software implementation of a protection module, and experimental testing. The work analyzes the architecture of modern web applications, the basic principles of web technologies, and typical cybersecurity threats. Existing protection algorithms are considered, and a demonstration web application Secure Notes is developed with input filtering, parameterized SQL queries, CSRF tokens, password hashing, session protection, and security event logging. Web attacks are modeled, and the functionality of the implemented protection mechanisms is tested. The practical significance of the work lies in the development of a demonstration web application that can be used for educational testing of basic protection methods against common web attacks. Keywords: web application, cybersecurity, SQL injection, XSS, CSRF, authentication, password hashing, Flask, SQLite, Secure Notes.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

CORS – Cross-Origin Resource Sharing

CSP – Content Security Policy

CSRF – Cross-Site Request Forgery

DAST – Dynamic Application Security Testing

LFI – Local File Inclusion

OWASP – Open Worldwide Application Security Project

RFI – Remote File Inclusion

SAST – Static Application Security Testing

URI – Uniform Resource Identifier

WAF – Web Application Firewall

БД – база даних;

ІБ – інформаційна безпека;

КСЗІ – комплексна система захисту інформації;

НСД – несанкціонований доступ;

ПЗ – програмне забезпечення.

ВСТУП

Сучасні веб-застосунки є невід'ємною частиною інформаційної інфраструктури підприємств, організацій, державних установ, освітніх платформ, фінансових сервісів і персональних онлайн-сервісів. Через веб-застосунки здійснюється передавання, обробка та зберігання значних обсягів даних, зокрема персональної, службової та конфіденційної інформації. Зростання кількості веб-сервісів одночасно збільшує кількість потенційних точок атак, пов'язаних із помилками програмної реалізації, недостатньою перевіркою вхідних даних, слабкою автентифікацією, небезпечною взаємодією з базами даних і некоректною обробкою *HTTP*-запитів.

Актуальність теми кваліфікаційної роботи зумовлена тим, що типові веб-атаки залишаються однією з найбільш поширених причин порушення безпеки веб-застосунків. До таких атак належать *SQL*-ін'єкції, міжсайтове виконання сценаріїв *XSS*, підробка міжсайтових запитів *CSRF*, помилки автентифікації, небезпечна робота з файлами та інші способи впливу на серверну або клієнтську частину застосунку. У разі успішної реалізації таких атак можуть бути порушені конфіденційність, цілісність і доступність інформації, що є базовими властивостями інформаційної безпеки.

Метою кваліфікаційної роботи є дослідження типових веб-атак і розробка методів захисту веб-застосунку, які забезпечують безпечну обробку введених даних, захищену взаємодію з базою даних, захист сесій користувача, використання *CSRF*-токенів, хешування паролів і журналювання подій безпеки.

Для досягнення поставленої мети необхідно вирішити такі задачі:

1) проаналізувати архітектуру сучасних веб-застосунків, основні принципи веб-технологій і класифікувати типові веб-атаки;

2) дослідити існуючі методи та алгоритми захисту веб-застосунків від *SQL*-ін'єкцій, *XSS*, *CSRF* і помилок автентифікації;

3) розробити структуру та програмну реалізацію демонстраційного веб-застосунку з модулем захисту від типових веб-атак;

4) провести моделювання веб-атак і оцінити ефективність реалізованих механізмів захисту.

Об’єктом дослідження є процес захисту веб-застосунків від типових веб-атак, які обумовлені вразливістю в написанні їх коду.

Предметом дослідження є методи та програмні механізми захисту веб-застосунків, зокрема валідація і фільтрація введених даних, параметризовані *SQL*-запити, *CSRF*-токени, хешування паролів, захист сесій і журналювання подій безпеки.

У роботі використано такі методи дослідження: метод аналізу літературних джерел і технічної документації для вивчення сучасних підходів до захисту веб-застосунків; метод класифікації для систематизації типових веб-загроз; метод моделювання для опису процесу обробки *HTTP*-запитів і ризиків веб-атак; метод програмної реалізації для створення демонстраційного веб-застосунку *Secure Notes*; метод експериментального тестування для перевірки роботи захисних механізмів; метод порівняльного аналізу для оцінки поведінки системи під час обробки безпечних і небезпечних запитів.

Практична частина роботи передбачає створення демонстраційного веб-застосунку *Secure Notes*. У ньому реалізовано реєстрацію користувача, вхід до системи, особистий кабінет, додавання нотаток, збереження даних у базі *SQLite* та журналювання подій безпеки. Для захисту застосунку використано параметризовані *SQL*-запити, перевірку введених даних, *CSRF*-токени, хешування паролів за допомогою *Werkzeug*, сесії *Flask* і журналювання підозрілих дій.

Практична значимість роботи полягає у створенні програмної моделі веб-застосунку з базовим модулем захисту, яку можна використовувати для навчального тестування методів протидії типовим веб-атакам. Розроблений підхід може бути застосований як основа для подальшого вдосконалення систем захисту веб-застосунків, а також для демонстрації принципів безпечної

обробки *HTTP*-запитів, роботи з базами даних і захисту облікових записів користувачів.

Робота відповідає наступним фаховим компетентностям:

– ФК-2 – Здатність використовувати інформаційні технології, сучасні методи і моделі кібербезпеки та системи захисту інформації;

– ФК-4 – Здатність забезпечувати захист інформації в інформаційних та інформаційно-комунікаційних системах згідно встановленої політики кібербезпеки й захисту інформації;

– ФК-10 – Здатність виконувати моніторинг інформаційних процесів, аналізувати, виявляти, оцінювати можливі вразливості та загрози інформаційному простору й інформаційним ресурсам згідно з встановленою політикою інформаційної безпеки.

Отже, тема кваліфікаційної роботи є актуальною, оскільки спрямована на вирішення практичної задачі підвищення захищеності веб-застосунків від типових веб-атак. Результати роботи мають теоретичне значення для систематизації методів захисту та практичне значення для демонстрації реалізації базових механізмів безпеки у веб-застосунку.

РОЗДІЛ 1 АНАЛІЗ СУЧАСНИХ ВЕБ-ЗАСТОСУНКІВ ТА ЗАГРОЗ БЕЗПЕЦІ

1.1 Архітектура сучасних веб-застосунків

Сучасні веб-застосунки є складними програмними системами, що забезпечують взаємодію користувача з інформаційними ресурсами через мережу Інтернет. На відміну від статичних веб-сайтів, основним призначенням яких є відображення інформації, веб-застосунки виконують обробку запитів користувачів, реалізують бізнес-логіку, забезпечують автентифікацію, авторизацію, зберігання та передавання даних. Саме тому архітектура веб-застосунку має важливе значення не лише для його функціональності, продуктивності та масштабованості, а й для забезпечення належного рівня кібербезпеки.

Основою більшості сучасних веб-застосунків є клієнт-серверна модель. У такій моделі клієнтська частина відповідає за взаємодію з користувачем, формування запитів і відображення результатів, а серверна частина виконує обробку запитів, перевірку прав доступу, реалізацію прикладної логіки та взаємодію з базою даних. Такий підхід дозволяє централізовано керувати даними та сервісами, однак одночасно створює ризики, пов’язані з передаванням інформації через відкриті мережі, обробкою неперевірених вхідних даних і можливістю несанкціонованого доступу до серверних ресурсів [2].

Узагальнену архітектуру сучасного веб-застосунку наведено на рисунку 1.1. Типовий веб-застосунок складається з декількох взаємопов’язаних компонентів: користувача, браузера або мобільного клієнта, клієнтської частини, API, серверної частини, бази даних, системи автентифікації, *Web Application Firewal*l, засобів журналювання та моніторингу, зовнішніх сервісів і хмарної інфраструктури [1]. Кожен із цих компонентів виконує окрему функцію, однак водночас може бути потенційною точкою реалізації веб-атаки.

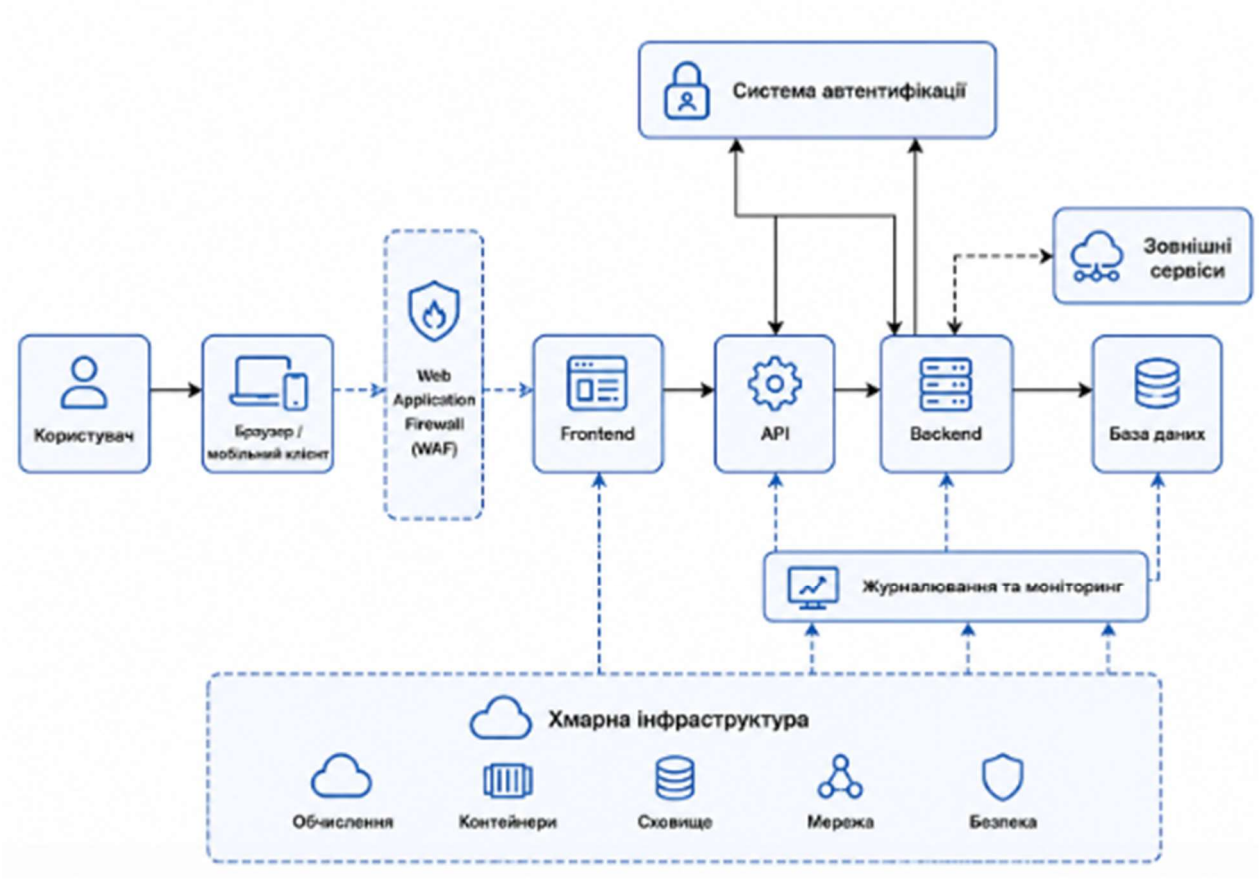

Рисунок 1.1 – Узагальнена архітектура сучасного веб-застосунку

Клієнтська частина, або *frontend*, забезпечує відображення інтерфейсу користувача та первинну взаємодію з ним. Вона реалізується за допомогою *HTML, CSS, JavaScript* та сучасних фреймворків. Через клієнтську частину користувач вводить дані, надсилає запити до сервера та отримує результати їх обробки. З погляду безпеки цей рівень є важливим, оскільки саме через нього можуть реалізовуватися атаки, пов’язані з міжсайтовим виконанням сценаріїв, підміною даних у браузері, фішингом або некоректною обробкою введеної користувачем інформації.

Інтерфейс прикладного програмування, або *API*, забезпечує обмін даними між клієнтською та серверною частинами веб-застосунку. У сучасних системах найчастіше використовуються *REST API* або *GraphQL*. *API* підвищує гнучкість веб-застосунку та дає змогу інтегрувати його з мобільними застосунками, мікросервісами й зовнішніми платформами. Водночас відкриті *API*-точки збільшують площину атаки, оскільки зловмисник може надсилати спеціально

сформовані запити з метою обходу авторизації, отримання надлишкових даних або порушення логіки роботи системи.

Серверна частина, або *backend*, відповідає за обробку запитів, реалізацію основної логіки роботи веб-застосунку, взаємодію з базою даних та перевірку прав доступу. Помилки в серверній логіці можуть призвести до критичних наслідків, зокрема несанкціонованого доступу, виконання ін'єкційних атак, витоку конфіденційних даних або порушення цілісності інформації. У дослідженнях із захисту програмного забезпечення підкреслюється, що недостатня перевірка вхідних даних і помилки в логіці обробки запитів є одними з основних причин появи вразливостей у програмних системах [2].

База даних є одним із найбільш критичних компонентів веб-застосунку, оскільки в ній зберігаються облікові записи користувачів, персональні дані, службова інформація, результати обробки запитів та інші важливі відомості. Небезпечна взаємодія серверної частини з базою даних може створювати умови для *SQL*-ін'єкцій, несанкціонованого читання, зміни або видалення інформації. Тому на рівні бази даних необхідно забезпечувати контроль доступу, фільтрацію запитів, резервне копіювання, шифрування критичних даних і журналювання операцій.

Окреме місце в архітектурі веб-застосунку займає система автентифікації та авторизації. Автентифікація призначена для перевірки особи користувача, а авторизація визначає рівень його доступу до ресурсів і функцій системи. Для цього можуть використовуватися сесії, токени доступу, *JWT, OAuth* 2.0, багатофакторна автентифікація та рольові моделі доступу. Неправильна реалізація цих механізмів може призвести до захоплення облікових записів, підвищення привілеїв, обходу обмежень доступу та компрометації всієї системи.

Важливим захисним елементом є *Web Application Firewall*, який виконує функцію фільтрації *HTTP*-запитів і блокування підозрілої активності. *WAF* може використовуватися для виявлення типових атак, зокрема *SQL*-ін'єкцій, *XSS*-атак, спроб обходу автентифікації та надсилання некоректних запитів.

Однак використання *WAF* не може повністю замінити безпечне проєктування та якісну реалізацію веб-застосунку, оскільки міжмережевий екран лише зменшує ризики, але не усуває першопричини вразливостей у коді або архітектурі.

Журналювання та моніторинг є необхідними компонентами архітектури веб-застосунку, оскільки дають змогу фіксувати дії користувачів, помилки системи, спроби несанкціонованого доступу та інші події безпеки. Аналіз журналів дозволяє виявляти підозрілу активність, реагувати на інциденти та проводити розслідування після порушень безпеки. Без належного моніторингу навіть успішна атака може залишатися непоміченою протягом тривалого часу, що збільшує потенційні збитки для власника системи [3].

Сучасні веб-застосунки також активно використовують хмарну інфраструктуру, контейнеризацію та зовнішні сервіси. Хмарні платформи забезпечують масштабованість, доступність і гнучкість розгортання, а контейнери спрощують перенесення застосунків між різними середовищами. Водночас такі технології створюють додаткові ризики, пов’язані з помилками конфігурації, відкритими портами, неналежним зберіганням секретів, використанням небезпечних образів контейнерів і недостатнім контролем прав доступу [4].

Таким чином, архітектура сучасного веб-застосунку є багаторівневою системою, у якій кожен компонент має власне функціональне призначення та власні ризики безпеки. Клієнтська частина, *API*, серверна логіка, база даних, система автентифікації, засоби моніторингу та хмарна інфраструктура повинні розглядатися як єдиний комплекс, що потребує узгодженого захисту. Проведений аналіз показує, що ефективний захист веб-застосунків від типових веб-атак має ґрунтуватися на розумінні архітектури системи, визначенні критичних точок взаємодії та впровадженні багаторівневих механізмів безпеки на всіх етапах життєвого циклу програмного продукту.

1.2 Основні принципи веб-технологій

Функціонування сучасних веб-застосунків базується на сукупності мережевих, прикладних і програмних технологій, які забезпечують передавання даних між клієнтом і сервером, обробку користувацьких запитів, збереження інформації та захист взаємодії. До основних принципів веб-технологій належать клієнт-серверна модель, використання протоколів *HTTP* та *HTTPS*, застосування *URI/URL* для ідентифікації ресурсів, обробка запитів і відповідей, підтримка сесій, використання *API*, механізми автентифікації та авторизації, а також застосування засобів захисту переданих даних.

Клієнт-серверна модель є базовим принципом побудови веб-застосунків. У цій моделі клієнт, яким найчастіше є веб-браузер або мобільний застосунок, формує запит до сервера. Сервер приймає цей запит, обробляє його відповідно до логіки застосунку, звертається до бази даних або зовнішніх сервісів і повертає клієнту відповідь. Така взаємодія дозволяє відокремити інтерфейс користувача від серверної логіки та централізовано керувати інформаційними ресурсами.

Основною перевагою клієнт-серверної моделі є можливість централізованого зберігання й обробки даних. Це спрощує супровід системи, оновлення програмного забезпечення, адміністрування доступу та масштабування. Водночас така модель створює низку ризиків безпеки, оскільки сервер стає центральною точкою обробки запитів і потенційною ціллю для атак. У разі недостатнього контролю доступу, некоректної обробки введених даних або помилок у бізнес-логіці зловмисник може отримати доступ до конфіденційної інформації або порушити роботу веб-застосунку [5,6].

Ключовим протоколом передавання даних у веб-середовищі є HTTP. Протокол *HTTP* забезпечує обмін повідомленнями між клієнтом і сервером за принципом «запит – відповідь». Клієнт надсилає HTTP-запит, у якому зазначає метод, адресу ресурсу, заголовки та, за потреби, тіло повідомлення. Сервер

обробляє запит і повертає *HTTP*-відповідь, що містить код стану, заголовки та дані, які мають бути відображені або оброблені клієнтом.

Найбільш поширеними *HTTP*-методами є *GET, POST, PUT, PATCH* та *DELETE*. Метод *GET* використовується для отримання ресурсу, *POST* – для передавання даних на сервер, *PUT* і *PATCH* – для оновлення ресурсу, *DELETE* – для його видалення. Неправильне використання *HTTP*-методів може створювати додаткові загрози. Наприклад, передавання конфіденційних даних через *GET*-запити є небезпечним, оскільки такі дані можуть зберігатися в історії браузера, журналах сервера або проміжних мережевих пристроях.

HTTP сам по собі не забезпечує криптографічного захисту інформації, тому дані, що передаються між клієнтом і сервером, можуть бути перехоплені або модифіковані в мережі. Для усунення цього недоліку використовується *HTTPS*, який є захищеною версією *HTTP* із застосуванням криптографічного протоколу *TLS*. *HTTPS* забезпечує шифрування даних, автентичність сервера та цілісність переданої інформації. Це особливо важливо для веб-застосунків, які працюють з обліковими записами користувачів, персональними даними, платіжною інформацією або іншими конфіденційними відомостями.

Використання *HTTPS* є обов’язковою умовою для захисту сучасних веб-застосунків. Без нього зловмисник може реалізувати атаку типу «людина посередині», перехопити логін і пароль користувача, змінити передані дані або підмінити вміст веб-сторінки. Проте саме по собі використання *HTTPS* не гарантує повної безпеки веб-застосунку, оскільки воно захищає канал передавання даних, але не усуває вразливості у коді, логіці авторизації, роботі з базою даних або конфігурації сервера.

Основні веб-технології, що використовуються під час функціонування веб-застосунків, наведено в таблиці 1.2.

Таблиця 1.2 – Основні технології функціонування веб-застосунків

Технологія	Основне призначення	Значення для безпеки
HTTP	Передавання запитів і відповідей між клієнтом та сервером	Визначає структуру взаємодії, але без додаткового захисту не забезпечує шифрування
HTTPS	Захищене передавання *HTTP*-трафіку через *TLS*	Забезпечує шифрування, цілісність і автентичність переданих даних
URL/URI	Ідентифікація веб-ресурсів	Може містити параметри, які потребують перевірки та фільтрації
Cookies	Збереження службових даних на стороні клієнта	Використовуються для сесій, але потребують захищених атрибутів
Sessions	Підтримка стану користувача між запитами	Дають змогу ідентифікувати користувача, але можуть бути ціллю атак викрадення сесії
API	Обмін даними між компонентами системи	Потребує контролю доступу, обмеження запитів і перевірки вхідних даних
TLS	Криптографічний захист каналу зв’язку	Захищає інформацію від перехоплення та модифікації
CORS	Контроль міждоменних запитів	Неправильне налаштування може призвести до несанкціонованого доступу до ресурсів

Як видно з таблиці 1.2, кожна веб-технологія має не лише функціональне призначення, а й безпосередній вплив на рівень захищеності веб-застосунку. Особливо важливими є ті механізми, які відповідають за передавання даних, ідентифікацію користувача, контроль доступу та взаємодію між різними компонентами системи.

Оскільки протокол *HTTP* є *stateless*-протоколом, тобто не зберігає стан між окремими запитами, у веб-застосунках використовуються додаткові механізми підтримки сесій. Найчастіше для цього застосовуються *cookies*, сесійні ідентифікатори або токени доступу. Після успішної автентифікації користувача сервер може створити сесію або видати токен, який надалі використовується для підтвердження прав користувача під час виконання наступних запитів.

Механізми сесій і *cookies* є критично важливими з точки зору безпеки. Якщо сесійний ідентифікатор буде викрадено, зловмисник може отримати доступ до облікового запису користувача без знання його пароля. Для зменшення таких ризиків необхідно використовувати атрибути *Secure, HttpOnly* та *SameSite* для *cookies*, обмежувати строк дії сесій, здійснювати повторну перевірку прав доступу для критичних операцій і завершувати сесію після виходу користувача із системи.

Одним із фундаментальних принципів веб-безпеки є перевірка та фільтрація вхідних даних. Усі дані, які надходять від користувача, браузера, *API* або зовнішніх сервісів, мають розглядатися як потенційно небезпечні. Якщо веб-застосунок без перевірки використовує введені дані в *SQL*-запитах, *HTML*-сторінках, командному рядку або файловій системі, це може створити умови для реалізації ін’єкційних атак, міжсайтового виконання сценаріїв або інших типових веб-загроз [1, 20].

Важливим принципом веб-технологій є розмежування прав доступу. Користувач повинен мати доступ лише до тих ресурсів і функцій, які відповідають його ролі в системі. Для цього використовуються механізми авторизації, рольові моделі доступу, перевірка прав на рівні серверної логіки та контроль доступу до *API*. Небезпечно покладатися лише на перевірки на стороні клієнта, оскільки зловмисник може змінити клієнтський код, підробити запит або напряму звернутися до серверного *API*.

У сучасних веб-застосунках значну роль відіграють *API*, які забезпечують взаємодію між клієнтською частиною, сервером, мобільними застосунками та

зовнішніми сервісами. *API* дає змогу будувати гнучкі та масштабовані системи, однак одночасно збільшує площину атаки. Небезпечними є ситуації, коли *API* повертає надмірний обсяг даних, не перевіряє права доступу до об’єктів, не обмежує кількість запитів або дозволяє виконувати критичні операції без належної автентифікації.

Для захисту клієнтської частини веб-застосунку використовуються політики безпеки браузера. До них належать *Same-Origin Policy, Content Security Policy*, механізми *CORS*, а також захисні *HTTP*-заголовки. *Same-Origin Policy* обмежує доступ скриптів до ресурсів іншого походження, *Content Security Policy* допомагає зменшити ризик *XSS*-атак, а *CORS* визначає, які зовнішні домени можуть звертатися до ресурсів веб-застосунку. Неправильне налаштування таких механізмів може послабити захист і створити додаткові можливості для атак.

Окрему увагу слід приділяти захисним HTTP-заголовкам. До них належать *Strict-Transport-Security, X-Content-Type-Options, X-Frame-Options, Content-Security-Policy* та *Referrer-Policy*. Вони дозволяють зменшити ризики атак, пов’язаних із підміною вмісту, *clickjacking*, витоком службової інформації та виконанням небезпечних сценаріїв. Використання таких заголовків є важливою складовою базового захисту веб-застосунку.

Таким чином, основні принципи веб-технологій формують технічну основу функціонування веб-застосунків і безпосередньо впливають на їхню безпеку. Клієнт-серверна модель, *HTTP/HTTPS*, сесії, *cookies, API, TLS, CORS* і захисні *HTTP*-заголовки забезпечують взаємодію між компонентами системи, але за неправильного використання можуть створювати вразливості. Тому під час розроблення веб-застосунків необхідно враховувати не лише функціональні можливості веб-технологій, а й потенційні загрози, які виникають у процесі їх застосування.

1.3 Класифікація кіберзагроз для веб-застосунків

Веб-застосунки працюють у відкритому мережевому середовищі, тому є постійною ціллю для кібератак. Основні загрози пов'язані з обробкою вхідних даних, автентифікацією користувачів, роботою з базами даних, файловою системою та серверним середовищем. У разі успішної атаки можуть бути порушені конфіденційність, цілісність або доступність інформації.

До найбільш поширених загроз для веб-застосунків належать *SQL*-ін'єкції, *XSS, CSRF, Command Injection, File Inclusion* та *Broken Authentication* [7-10]. Їх класифікацію наведено в таблиці 1.3.

Таблиця 1.3 – Класифікація типових кіберзагроз для веб-застосунків

Загроза	Сутність атаки	Можливі наслідки
SQL-ін'єкція	Впровадження шкідливих *SQL*-команд у запити до бази даних	Отримання, зміна або видалення даних
XSS	Виконання шкідливого сценарію у браузері користувача	Викрадення сесій, підміна сторінки, фішинг
CSRF	Виконання небажаної дії від імені автентифікованого користувача	Зміна пароля, налаштувань або виконання операцій без згоди користувача
Command Injection	Передавання команд операційній системі через веб-застосунок	Виконання команд на сервері, компрометація системи
File Inclusion	Підключення локальних або віддалених файлів через параметри запиту	Розкриття файлів, виконання шкідливого коду
Broken Authentication	Помилки в механізмах входу, сесій або токенів доступу	Захоплення облікового запису, обхід автентифікації

SQL-ін’єкція є атакою, за якої зловмисник вводить спеціально сформований *SQL*-код у поля введення або параметри запиту. Якщо веб-застосунок не перевіряє ці дані, шкідливий код може бути виконаний базою даних. Наслідком такої атаки може бути несанкціонований доступ до інформації, зміна записів або повне видалення даних.

XSS, або міжсайтове виконання сценаріїв, виникає тоді, коли веб-застосунок відображає користувацькі дані без належної перевірки. У результаті в браузері користувача може виконатися шкідливий *JavaScript*-код. Така атака використовується для викрадення *cookies*, сесійних токенів або перенаправлення користувача на фішингові сторінки.

CSRF є атакою, за якої зловмисник змушує автентифікованого користувача виконати дію у веб-застосунку без його згоди. Наприклад, це може бути зміна пароля, електронної пошти або виконання певної операції в системі. Основною причиною *CSRF* є відсутність спеціальних токенів перевірки запиту.

Command Injection належить до критичних загроз, оскільки дає змогу виконувати команди операційної системи через веб-застосунок. Така вразливість виникає тоді, коли введені користувачем дані передаються до системних команд без перевірки. Успішна атака може призвести до повної компрометації сервера.

File Inclusion пов’язана з небезпечним підключенням файлів у веб-застосунку. У разі *Local File Inclusion* зловмисник може отримати доступ до локальних файлів сервера, а у разі *Remote File Inclusion* – підключити віддалений шкідливий файл. Наслідком може бути розкриття конфігураційних файлів або виконання шкідливого коду.

Broken Authentication охоплює помилки в механізмах входу користувача, керування сесіями та токенами доступу. До таких помилок належать слабкі паролі, відсутність обмеження кількості спроб входу, небезпечне зберігання токенів і неправильне завершення сесій. Наслідком може бути захоплення облікового запису або отримання доступу до закритих функцій системи [13, 27].

Таким чином, основні загрози для веб-застосунків пов’язані з недостатньою перевіркою вхідних даних, слабкою автентифікацією, помилками авторизації та небезпечною взаємодією із серверним середовищем. Для зменшення ризиків необхідно застосовувати параметризовані запити, фільтрацію даних, захищені сесії, *CSRF*-токени, обмеження прав доступу та постійний моніторинг подій безпеки. Отже, класифікація загроз є основою для подальшого дослідження методів і засобів захисту веб-застосунків.

Висновок до розділу

Проведений аналіз показав, що безпека веб-застосунків залежить від захищеності всіх компонентів їх архітектури: клієнтської та серверної частин, API, бази даних, механізмів автентифікації та хмарної інфраструктури. Кожен із цих елементів може бути джерелом потенційних вразливостей.

Дослідження веб-технологій засвідчило, що *HTTP/HTTPS*, сесії, *cookies*, *API* та механізми контролю доступу є основою функціонування веб-застосунків, але потребують правильного налаштування для забезпечення належного рівня безпеки.

Встановлено, що найбільш поширеними загрозами є *SQL*-ін’єкції, *XSS*, *CSRF, Command Injection, File Inclusion* та порушення автентифікації. Для їх ефективного запобігання необхідно застосовувати комплексні механізми захисту, зокрема перевірку вхідних даних, контроль доступу та моніторинг подій безпеки.

Отже, розуміння архітектури веб-застосунків, принципів роботи веб-технологій і сучасних кіберзагроз є основою для розроблення ефективних методів захисту від веб-атак.

РОЗДІЛ 2 ДОСЛІДЖЕННЯ ІСНУЮЧИХ МЕТОДІВ ТА ЗАСОБІВ ЗАХИСТУ

2.1 Аналіз вразливостей існуючої моделі

Для подальшого дослідження методів захисту веб-застосунків необхідно визначити слабкі місця існуючої моделі та оцінити рівень ризику для її основних компонентів. У межах цієї роботи існуюча модель веб-застосунку розглядається як сукупність взаємопов'язаних елементів: клієнтської частини, API, серверної логіки, бази даних, системи автентифікації, файлової системи, засобів журналювання та інфраструктури розгортання[10-12].

На відміну від класифікації веб-атак, наведеної в підрозділі 1.3, у цьому підрозділі аналізуються не окремі типи атак, а причини виникнення вразливостей у структурі веб-застосунку. Такий підхід дозволяє встановити, які компоненти системи потребують першочергового захисту та які методи доцільно застосовувати для зниження ризиків.

Існуючу модель веб-застосунку можна подати у вигляді множини компонентів:

$$M=\{C, A, S, D, F, I, L\},$$

де C – клієнтська частина веб-застосунку;

A – *API* та канали обміну даними;

S – серверна логіка;

D – база даних;

F – файлова система;

I – інфраструктура розгортання;

L – система журналювання та моніторингу.

Кожен компонент цієї моделі має власні функції та власні потенційні вразливості. Клієнтська частина відповідає за взаємодію з користувачем, але не може вважатися довіреним середовищем, оскільки користувач або зловмисник має можливість змінювати параметри запитів, *cookies*, заголовки та інші дані перед їх надсиланням на сервер. Тому перевірка даних лише на стороні браузера не забезпечує належного рівня захисту.

API є однією з найбільш критичних частин веб-застосунку, оскільки саме через нього відбувається обмін даними між клієнтом, сервером і зовнішніми сервісами. Основні ризики *API* пов’язані з недостатньою перевіркою прав доступу, відсутністю обмеження кількості запитів, передаванням надмірного обсягу даних і можливістю прямого звернення до внутрішніх функцій системи. Тому кожен *API*-запит має проходити не лише автентифікацію, а й перевірку авторизації.

Серверна логіка є центральним елементом моделі, оскільки саме вона обробляє запити, приймає рішення, взаємодіє з базою даних і формує відповіді клієнту. Вразливості цього компонента найчастіше виникають через недостатню перевірку вхідних даних, помилки у бізнес-логіці, небезпечне використання системних функцій або неправильну обробку винятків. Такі недоліки можуть призвести до обходу обмежень доступу, виконання небезпечних операцій або порушення цілісності даних [12].

База даних є найбільш цінним інформаційним активом веб-застосунку, оскільки містить облікові записи, персональні дані, службову інформацію та результати роботи системи. Основна небезпека полягає у формуванні запитів із використанням неперевірених користувацьких даних. Для зниження ризику необхідно застосовувати параметризовані запити, обмеження прав доступу до бази даних і контроль операцій із критичними таблицями [13].

Файлова система також є потенційно вразливим компонентом. Якщо веб-застосунок дозволяє завантаження, перегляд або підключення файлів, необхідно перевіряти їх тип, розмір, розширення, шлях зберігання та права

доступу. Відсутність таких перевірок може призвести до розкриття службових файлів, завантаження шкідливого коду або виконання небажаних дій на сервері.

Інфраструктура розгортання охоплює сервер, операційну систему, хмарне середовище, контейнери, мережеві налаштування та службові компоненти. Навіть за умови коректної реалізації програмного коду неправильна конфігурація інфраструктури може створити критичні вразливості. До таких проблем належать відкриті службові порти, стандартні паролі, застарілі бібліотеки, незахищені сховища та зберігання секретних ключів у відкритому вигляді.

Для формалізованого аналізу вразливостей доцільно застосувати ризик-орієнтований підхід. Рівень ризику для окремої вразливості можна визначити як добуток імовірності її використання та рівня можливого впливу:

$$R_i = P_i \cdot I_i,$$

де R_i– рівень ризику для і-ї вразливості;

P_i– імовірність використання вразливості;

I_i– вплив успішної експлуатації вразливості на систему.

Для комплексної оцінки ризику всієї моделі веб-застосунку можна використати узагальнену оцінку:

$$R_\Sigma = \sum_{j=1}^{n} w_i \cdot P_i \cdot I_i$$

де R_Σ – загальний рівень ризику моделі;

n – кількість виявлених вразливостей

w_i– ваговий коефіцієнт критичності компонента;

P_i– імовірність реалізації загрози;

I_i– рівень впливу на систему.

У межах цієї роботи вихідними даними для аналізу є структура веб-застосунку, визначена в підрозділі 1.1, класифікація типових веб-загроз, наведена в підрозділі 1.3, а також типові точки входу: форми введення, *URL*-параметри, *HTTP*-заголовки, *cookies*, *API*-запити, механізми завантаження файлів і запити до бази даних.

Для проведення аналізу приймаються такі умови: веб-застосунок доступний через мережу Інтернет; користувачі взаємодіють із системою через браузер або мобільний клієнт; серверна частина обробляє *HTTP/HTTPS*-запити; база даних зберігає критичну інформацію; зловмисник не має фізичного доступу до сервера, але може надсилати спеціально сформовані запити до відкритих точок входу. Такі умови відповідають типовому сценарію експлуатації сучасного веб-застосунку.

Аналіз показує, що найбільш критичними зонами існуючої моделі є обробка вхідних даних, *API*, механізми автентифікації та авторизації, взаємодія з базою даних і конфігурація серверної інфраструктури. Саме ці компоненти мають найбільший вплив на конфіденційність, цілісність і доступність інформації.

Перспективним напрямом подальшого дослідження є розробка комбінованого підходу до захисту веб-застосунку, який поєднує перевірку вхідних даних, параметризовані запити до бази даних, захищене керування сесіями, контроль доступу до *API*, безпечну роботу з файлами, налаштування захисних *HTTP*-заголовків, використання *Web Application Firewall* та журналювання подій безпеки.

Очікуваним результатом такого підходу є зниження ймовірності реалізації типових веб-атак, підвищення стійкості веб-застосунку до несанкціонованого доступу, зменшення ризику витоку або модифікації даних, а також забезпечення своєчасного виявлення підозрілої активності. Отже, аналіз вразливостей існуючої моделі підтверджує необхідність подальшого дослідження існуючих алгоритмів і засобів захисту, що розглядаються в наступному підрозділі.

2.2 Опис існуючих алгоритмів захисту

Захист веб-застосунків від типових веб-атак ґрунтується на поєднанні декількох груп алгоритмів і технічних механізмів. Їх основне призначення полягає у зменшенні кількості вразливих точок, перевірці вхідних даних, контролі доступу, захисті сесій, безпечній взаємодії з базою даних, фільтрації HTTP-запитів і своєчасному виявленні підозрілої активності.

Першим базовим алгоритмом є перевірка та фільтрація вхідних даних. Його сутність полягає в тому, що всі дані, які надходять до веб-застосунку від користувача, *API* або зовнішнього сервісу, мають розглядатися як потенційно небезпечні. До таких даних належать параметри *URL*, вміст форм, *HTTP*-заголовки, cookies, *JSON*-запити та завантажені файли. Алгоритм перевірки вхідних даних передбачає визначення допустимого формату, перевірку типу даних, обмеження довжини, видалення або екранування небезпечних символів і відхилення некоректних запитів.

У загальному вигляді алгоритм перевірки вхідних даних можна подати так:

1) отримання даних від користувача або зовнішнього джерела;
2) визначення очікуваного типу та формату даних;
3) перевірка відповідності даних заданим правилам;
4) нормалізація або екранування допустимих значень;
5) відхилення запиту у разі виявлення небезпечного вмісту;
6) передавання перевірених даних до подальшої обробки.

Такий алгоритм є універсальним, оскільки застосовується для зменшення ризику *SQL*-ін’єкцій, *XSS, Command Injection, File Inclusion* та інших атак, пов’язаних із небезпечною обробкою введення [14-16].

Для захисту від *SQL*-ін’єкцій використовуються параметризовані запити та підготовлені вирази. Основна ідея цього підходу полягає в тому, що *SQL*-команда відокремлюється від користувацьких даних. У такому випадку введене користувачем значення не може змінити структуру *SQL*-запиту, навіть якщо

містить спеціальні символи або фрагменти *SQL*-коду. Це є одним із найбільш ефективних методів захисту бази даних веб-застосунку.

Алгоритм безпечної роботи з базою даних передбачає:

1) приймання запиту від клієнта;

2) перевірку та нормалізацію вхідних даних;

3) формування *SQL*-запиту з використанням параметрів;

4) виконання запиту з обмеженими правами користувача бази даних;

5) повернення лише необхідного обсягу інформації;

6) журналювання критичних операцій.

Додатковим засобом захисту бази даних є принцип мінімальних привілеїв. Він передбачає, що обліковий запис, через який веб-застосунок підключається до бази даних, повинен мати лише ті права, які необхідні для виконання конкретних функцій. Наприклад, якщо модуль застосунку лише читає інформацію, він не повинен мати права на зміну або видалення таблиць.

Для захисту від *XSS*-атак використовується алгоритм безпечного виведення даних. Його сутність полягає в тому, що будь-яка інформація, отримана від користувача, перед відображенням на веб-сторінці повинна бути екранована. Це означає, що спеціальні символи, які можуть бути інтерпретовані браузером як *HTML* або *JavaScript*-код, перетворюються на безпечний текстовий формат.

Алгоритм захисту від *XSS* включає:

1) визначення місця, де дані будуть відображені;

2) перевірку джерела даних;

3) екранування *HTML, JavaScript* або *URL*-символів залежно від контексту;

4) застосування безпечних шаблонізаторів;

5) налаштування політики *Content Security Policy*;

6) використання атрибута *HttpOnly* для *cookies*.

Важливим доповненням до цього алгоритму є *Content Security Policy*. Цей механізм дозволяє обмежити джерела, з яких браузеру дозволено

завантажувати скрипти, стилі, зображення та інші ресурси. У разі правильної конфігурації *CSP* зменшує ймовірність виконання стороннього шкідливого коду в браузері користувача.

Для захисту від *CSRF*-атак застосовується алгоритм перевірки унікального токена запиту. Після відкриття форми або створення сесії сервер генерує спеціальний CSRF-токен і передає його клієнту. Під час виконання критичної дії клієнт повинен повернути цей токен разом із запитом. Якщо токен відсутній або не відповідає значенню, збереженому на сервері, запит відхиляється.

Алгоритм *CSRF*-захисту має таку послідовність:

1) створення унікального токена для користувацької сесії;
2) додавання токена до форми або заголовка запиту;
3) надсилання запиту користувачем;
4) перевірка токена на сервері;
5) виконання операції лише у разі успішної перевірки;
6) відхилення запиту у разі відсутності або невідповідності токена.

Додатково для зменшення ризику *CSRF* застосовується перевірка заголовків *Origin* та *Referer*, а також атрибут *SameSite* для *cookies*. Це дозволяє обмежити можливість надсилання запитів із небажаних зовнішніх ресурсів.

Для захисту автентифікації та сесій використовуються алгоритми безпечного керування обліковими записами. Вони передбачають перевірку складності паролів, обмеження кількості невдалих спроб входу, використання багатофакторної автентифікації, безпечне зберігання паролів і контроль строку дії сесій. Паролі не повинні зберігатися у відкритому вигляді. Для цього застосовуються криптографічні хеш-функції з додаванням солі.

Алгоритм безпечної автентифікації включає:

1) отримання логіна та пароля;
2) перевірку кількості попередніх невдалих спроб входу;
3) хешування введеного пароля;
4) порівняння хешу з даними, збереженими в базі;

5) створення захищеної сесії або токена доступу;
6) обмеження строку дії сесії;
7) завершення сесії після виходу користувача із системи.

Для захисту від несанкціонованого доступу важливим є алгоритм авторизації. Його завданням є перевірка того, чи має користувач право виконувати конкретну дію. Автентифікація лише підтверджує особу користувача, але не визначає його права. Тому кожен запит до захищеного ресурсу повинен проходити перевірку ролі, рівня доступу або належності ресурсу конкретному користувачу.

Загальну модель контролю доступу можна подати у вигляді умови:

$$Access(u, r, a) = \begin{cases} 1, & \text{якщо користувач } u \text{ має право виконати дію } a \text{ над ресурсом } r, \\ 0, & \text{якщо такого права немає.} \end{cases}$$

де u – користувач;

r – ресурс;

a – дія над ресурсом;

Access – результат перевірки доступу.

Якщо значення функції дорівнює 1, запит виконується. Якщо значення дорівнює 0, сервер повинен відхилити запит і зафіксувати подію в журналі безпеки.

Для захисту від *Command Injection* використовується алгоритм обмеження системних викликів. Основний принцип полягає в тому, що веб-застосунок не повинен безпосередньо передавати користувацькі дані до команд операційної системи. Якщо виконання системної команди є необхідним, слід використовувати список дозволених значень, перевірку параметрів, ізоляцію процесів і мінімальні права доступу.

Для захисту від *File Inclusion* і небезпечного завантаження файлів застосовується алгоритм контролю файлових операцій. Він передбачає перевірку розширення, *MIME*-типу, розміру файлу, шляху зберігання, прав доступу та заборону виконання завантажених файлів як програмного коду. Додатково доцільно зберігати файли поза основним каталогом веб-застосунку та перейменовувати їх після завантаження.

Окрему групу становлять алгоритми фільтрації *HTTP*-запитів, які реалізуються за допомогою *Web Application Firewall. WAF* аналізує вхідний трафік і блокує запити, що містять ознаки типових атак. Такий механізм не усуває вразливості в коді, але знижує ймовірність їх успішної експлуатації. Найбільшу ефективність *WAF* має тоді, коли використовується разом із серверною валідацією, контролем доступу та журналюванням подій.

Для виявлення вразливостей у процесі розроблення застосовуються методи статичного, динамічного та гібридного аналізу. Статичний аналіз дозволяє перевіряти вихідний код без його виконання. Динамічний аналіз досліджує поведінку веб-застосунку під час роботи. Гібридний аналіз поєднує переваги обох підходів і дозволяє виявляти ширший спектр вразливостей [1].

Ефективність застосування алгоритмів захисту можна оцінювати через зміну рівня ризику до і після впровадження засобу захисту:

$$E = \frac{R_0 - R_1}{R_0} \cdot 100\%$$

де E – ефективність застосованого методу захисту;

R_0– рівень ризику до впровадження захисту;

R_1– рівень ризику після впровадження захисту.

Ця модель дозволяє кількісно оцінити, наскільки обраний метод зменшує ризик для веб-застосунку. Наприклад, якщо після впровадження параметризованих запитів і перевірки введення ризик *SQL*-ін’єкцій зменшується, це свідчить про ефективність обраного механізму.

2.3 Розробка структури модулю системи інформаційного захисту

Модуль інформаційного захисту доцільно розміщувати між зовнішніми запитами користувачів і основною серверною логікою веб-застосунку. У такому випадку кожен *HTTP/HTTPS*-запит проходить попередню перевірку до того, як буде переданий до функціональних компонентів системи. Це дозволяє зменшити ймовірність потрапляння шкідливих або некоректних даних до серверної логіки, бази даних чи файлової системи.

Основне призначення модуля полягає у виконанні трьох функцій: попередній аналіз запиту, прийняття рішення щодо його безпечності та передавання результату до веб-застосунку. Таким чином, модуль виконує роль проміжного захисного рівня, який не замінює наявні механізми безпеки, а координує їх застосування в межах єдиної логіки обробки запитів.

Структурно роботу модуля можна подати як послідовність обробки запиту:

1) надходження *HTTP/HTTPS*-запиту від користувача або зовнішнього сервісу;
2) виділення параметрів запиту, заголовків, *cookies*, тіла запиту та токена доступу;
3) порівняння отриманих даних із правилами безпеки;
4) перевірка стану автентифікації та прав доступу;
5) оцінювання рівня ризику запиту;
6) прийняття рішення: дозволити, заблокувати або передати на додаткову перевірку;
7) фіксація результату обробки в журналі подій;
8) передавання безпечного запиту до основної логіки веб-застосунку.

Для формалізації роботи модуля доцільно використати функцію прийняття рішення:

$$D(Q) = \begin{cases} allow, & \text{якщо запит відповідає правилам безпеки,} \\ block, & \text{якщо запит містить ознаки атаки,} \\ review, & \text{якщо запит потребує додаткової перевірки.} \end{cases}$$

де $D(Q)$ – результат обробки запиту;

Q – вхідний *HTTP/HTTPS*-запит;

allow – дозвіл на подальшу обробку;

block – блокування запиту;

review – передавання запиту на додатковий аналіз.

Така модель дає змогу не просто блокувати небезпечні запити, а й виділяти проміжну категорію підозрілих дій. Це важливо для веб-застосунків, у яких надмірно жорстке блокування може порушити роботу легальних користувачів. Наприклад, запит може не містити явних ознак атаки, але мати нетипову кількість параметрів, незвичний формат даних або надходити з підозрілою частотою.

Вхідними даними для роботи модуля є параметри запиту, метод *HTTP*, *URL*-адреса ресурсу, заголовки, *cookies*, тіло запиту, токен доступу, роль користувача та поточна політика безпеки. До вихідних даних належать рішення щодо обробки запиту, повідомлення про помилку або блокування, запис у журналі подій і переданий до серверної логіки безпечний запит.

Політика безпеки в межах модуля повинна містити набір правил, за якими здійснюється перевірка запитів. До таких правил належать допустимі типи даних, обмеження довжини параметрів, перелік дозволених *HTTP*-методів, вимоги до токенів, права доступу користувачів, обмеження файлових операцій і правила фіксації подій безпеки. Наявність єдиної політики дозволяє централізовано керувати захистом веб-застосунку.

Особливістю запропонованої структури є те, що модуль не повинен бути жорстко прив’язаний до конкретної атаки. Його завдання полягає не лише у виявленні *SQL*-ін’єкцій, *XSS* або *CSRF*, а у перевірці загальної коректності та

допустимості запиту. Завдяки цьому модуль може бути адаптований до різних типів веб-застосунків і розширений новими правилами захисту.

Граничними умовами для роботи модуля є доступність веб-застосунку через мережу Інтернет, наявність відкритих точок входу, використання *HTTP/HTTPS*-запитів і відсутність фізичного доступу зловмисника до сервера. Передбачається, що зловмисник може змінювати параметри запитів, надсилати спеціально сформовані дані, повторювати запити та намагатися обійти механізми автентифікації.

Перспективним напрямом подальшої роботи є реалізація запропонованої структури у вигляді програмного модуля, який може бути інтегрований у веб-застосунок або працювати як проміжний захисний компонент на рівні *API*. Очікуваним результатом є зменшення кількості успішних веб-атак, підвищення контролю над обробкою запитів, своєчасне виявлення підозрілої активності та зниження загального рівня ризику для веб-застосунку.

Висновки до розділу

У розділі проведено аналіз вразливостей існуючої моделі веб-застосунку та визначено найбільш критичні компоненти, які потребують захисту. Розглянуто основні алгоритми протидії типовим веб-атакам, зокрема перевірку вхідних даних, параметризовані запити, захист сесій, контроль доступу, механізми протидії XSS і CSRF, а також використання Web Application Firewall.

На основі проведеного аналізу розроблено структуру модуля інформаційного захисту, який виконує перевірку запитів, оцінювання рівня ризику та прийняття рішень щодо їх подальшої обробки. Запропонований підхід дає змогу підвищити рівень захищеності веб-застосунку, зменшити ймовірність успішної реалізації веб-атак і забезпечити більш ефективний контроль подій безпеки.

РОЗДІЛ 3 РОЗРОБКА МЕТОДІВ ЗАХИСТУ ВЕБ-ЗАСТОСУНКУ

3.1 Постановка задачі розробки

Метою практичної частини кваліфікаційної роботи є розробка демонстраційного веб-застосунку *Secure Notes* із вбудованим модулем захисту від типових веб-атак. Розроблений застосунок призначений для перевірки ефективності базових методів захисту веб-застосунків під час обробки *HTTP*-запитів, автентифікації користувачів, збереження даних у базі даних та виведення користувацької інформації на веб-сторінку.

У межах роботи створюється веб-застосунок із такими функціями:

- реєстрація користувача;
- вхід користувача до системи;
- створення захищеної сесії;
- перехід до особистого кабінету;
- додавання текстових нотаток;
- збереження нотаток у базі даних;
- журналювання підозрілих дій.

Основна проблема, яку вирішує розробка, полягає у зменшенні ризику реалізації типових атак на веб-застосунок. До таких атак належать *SQL*-ін’єкції, *XSS, CSRF* та атаки, пов’язані з порушенням механізмів автентифікації. У звичайній незахищеній моделі веб-застосунку небезпечні дані можуть потрапити до серверної логіки, *SQL*-запитів або *HTML*-сторінки без належної перевірки. Це може призвести до несанкціонованого доступу, викрадення сесії, виконання шкідливого сценарію або компрометації облікових даних користувача.

Для вирішення цієї проблеми в роботі розробляється модуль захисту, який виконує попередню перевірку запитів і забезпечує безпечну взаємодію

між клієнтом, сервером та базою даних. Узагальнену структуру роботи розроблюваного веб-застосунку наведено на рисунку 3.1.

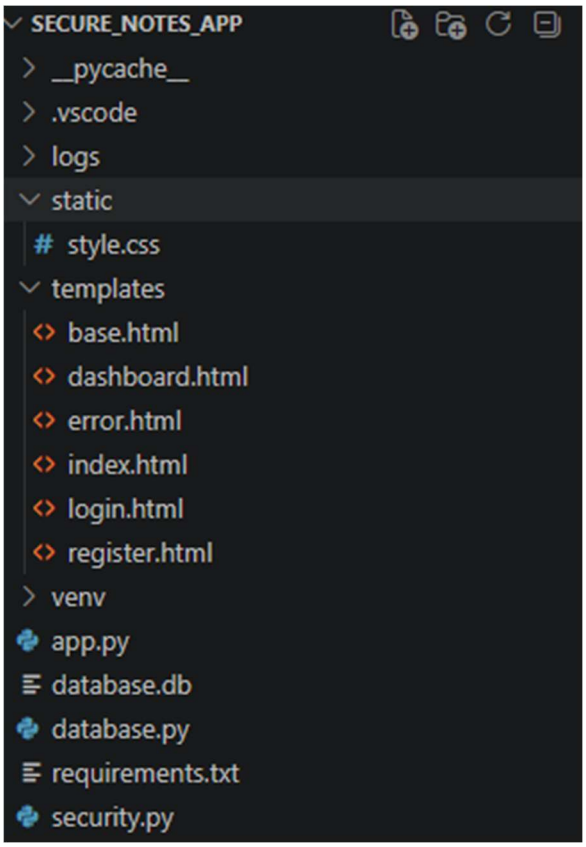

Рисунок 3.1 – Структура програмної реалізації

На рисунку 3.1 показано структуру створеного проєкту у середовищі *Visual Studio Code*. До складу програмної реалізації входять файли *app.py*, *database.py, security.py,* каталог *templates* з *HTML*-шаблонами, каталог *static* зі стилями, база даних *database.db* та журнал подій безпеки *security.log*. Такий поділ дозволяє відокремити основну логіку веб-застосунку від функцій роботи з базою даних і механізмів захисту.

Для реалізації цієї моделі в застосунку *Secure Notes* передбачено такі вимоги безпеки:

– захист від *SQL*-ін’єкцій шляхом використання параметризованих *SQL*-запитів;

– захист від *XSS* шляхом перевірки введених даних і блокування небезпечних шаблонів;

– захист від *CSRF* шляхом використання унікальних *CSRF*-токенів у формах;

– безпечне зберігання паролів користувачів у вигляді хешів;

– перевірка автентифікації перед доступом до особистого кабінету;

– журналювання підозрілих дій у файлі подій безпеки;

– обмеження некоректного введення у формах реєстрації, входу та додавання нотаток.

Вхідними даними для роботи розробленого модуля є логін і пароль користувача, текст нотатки, HTTP-метод запиту, параметри форми, сесійні дані, *cookies* та *CSRF*-токен. Вихідними даними є результат обробки запиту: успішна реєстрація, успішний вхід, збереження нотатки, блокування небезпечного введення, повідомлення про помилку або запис у журналі подій безпеки.

Дослідна установка для перевірки запропонованої моделі складається з локального веб-сервера *Flask*, бази даних *SQLite*, браузера користувача та середовища розробки *Visual Studio Code*. Веб-застосунок запускається локально за адресою *http*://127.0.0.1:5000, що дозволяє виконати тестування основних функцій без розгортання на зовнішньому сервері.

Для перевірки працездатності розробленого модуля передбачено використання контрольних вхідних даних. Для перевірки *SQL*-захисту використовується рядок:

'OR'1'='1

Для перевірки *XSS*-захисту використовується тестовий *JavaScript*-фрагмент:

<script>alert('XSS')</script>

Для перевірки *CSRF*-захисту виконується спроба надсилання форми без коректного *CSRF*-токена. Для перевірки захисту паролів аналізується запис користувача в базі даних і перевіряється, що пароль зберігається не у відкритому вигляді, а як хеш.

Критеріями успішності розробки є такі результати:

– *SQL*-ін’єкційний рядок не дозволяє обійти автентифікацію;
– *XSS*-навантаження не виконується в браузері користувача;
– запит без коректного *CSRF*-токена блокується сервером;
– пароль користувача зберігається в базі даних у вигляді хешу;
– підозрілі дії записуються до журналу подій безпеки;
– легальні дії користувача виконуються без блокування.

Очікуваним результатом розробки є створення працездатного демонстраційного веб-застосунку, який підтверджує можливість практичного застосування базових методів захисту від типових веб-атак. Запропонований підхід дозволяє не лише реалізувати окремі механізми захисту, а й об’єднати їх у єдину модель безпечної взаємодії клієнта, сервера та бази даних. Надалі ця модель буде використана для вибору технологій реалізації та проведення експериментальної перевірки ефективності розробленого захисту.

3.2 Вибір технології захисту

Для практичної реалізації поставленої задачі було обрано програмно-алгоритмічний підхід, який передбачає створення демонстраційного веб-застосунку *Secure Notes*. Застосунок використовується як дослідна модель для перевірки методів захисту від типових веб-атак. Основним критерієм вибору технологій була можливість швидко створити працездатну клієнт-серверну систему, реалізувати обробку *HTTP*-запитів, роботу з базою даних, автентифікацію користувачів, перевірку введених даних і фіксацію подій безпеки.

Під час вибору технологій враховувалися такі вимоги:

– простота розгортання веб-застосунку в локальному середовищі;
– можливість реалізації клієнт-серверної взаємодії;
– підтримка роботи з *HTML*-формами та *HTTP*-запитами;
– можливість збереження даних у базі даних;
– підтримка механізмів автентифікації та сесій;
– можливість реалізації перевірки введених даних;
– можливість журналювання подій безпеки;
– зручність демонстрації результатів тестування.

Як основну мову програмування було обрано *Python*. Ця мова є зручною для створення демонстраційних програмних рішень, має зрозумілий синтаксис і підтримує велику кількість бібліотек для веб-розробки, роботи з базами даних, обробки рядків і журналювання подій. Для бакалаврської роботи *Python* є доцільним вибором, оскільки дозволяє зосередитися не на складності інфраструктури, а на реалізації та перевірці методів захисту веб-застосунку.

Для створення серверної частини було обрано веб-фреймворк *Flask*. Він забезпечує маршрутизацію *HTTP*-запитів, обробку форм, роботу із сесіями, підключення *HTML*-шаблонів і запуск локального веб-сервера. *Flask* не потребує складного налаштування, тому добре підходить для створення дослідного веб-застосунку. У межах роботи він використовується як основа для реалізації сторінок реєстрації, входу до системи, особистого кабінету та сторінки додавання нотаток.

Для збереження даних було обрано *SQLite*. Ця база даних не потребує встановлення окремого серверного програмного забезпечення та зберігає інформацію у локальному файлі. Такий варіант є достатнім для дослідної установки, оскільки робота виконується в локальному середовищі. *SQLite* дозволяє створити таблиці користувачів і нотаток, а також перевірити захищену взаємодію веб-застосунку з базою даних.

Для формування *HTML*-сторінок використовується шаблонізатор *Jinja*2, який входить до складу *Flask*. Він дозволяє розділити серверну логіку та інтерфейс користувача. Завдяки цьому сторінки реєстрації, входу та особистого кабінету можуть формуватися на основі шаблонів, а дані передаються до них із серверної частини застосунку.

Для роботи з паролями обрано бібліотеку *Werkzeug*, яка входить до екосистеми *Flask*. Її використання доцільне, оскільки вона містить готові засоби для безпечної обробки паролів. Це дозволяє не створювати власні криптографічні функції, а використовувати готові інструменти, призначені для веб-застосунків.

Для створення токенів і випадкових значень використовується стандартний модуль *Python secrets*. Його перевагою є те, що він призначений для генерації криптографічно стійких випадкових значень. У межах обраної архітектури цей модуль є придатним для формування унікальних значень, необхідних для перевірки справжності запитів.

Для перевірки текстових даних використовується модуль *re*, який забезпечує роботу з регулярними виразами. Його вибір пояснюється необхідністю аналізувати вхідні рядки та виявляти небажані конструкції. Використання регулярних виразів дозволяє створити набір правил для початкової перевірки введених даних у межах дослідної моделі.

Для фіксації подій безпеки використовується модуль *logging*. Він дає змогу записувати інформацію про роботу застосунку та підозрілі дії до окремого журналу. Це важливо для експериментальної частини роботи, оскільки результати тестування мають бути не лише візуально продемонстровані, а й зафіксовані у вигляді подій.

Середовищем розробки обрано *Visual Studio Code*. Воно забезпечує зручну роботу з *Python*-файлами, *HTML*-шаблонами, *CSS*-файлами, вбудованим терміналом і структурою проєкту[17]. Це дозволяє одночасно розробляти програмний код, запускати локальний сервер і контролювати результати роботи застосунку.

Для розгортання дослідної установки використовується локальне середовище. Веб-застосунок запускається на локальному сервері *Flask* за адресою *http*://127.0.0.1:5000. Такий підхід дозволяє безпечно виконувати тестові запити, перевіряти реакцію системи на небезпечні введення та формувати скріншоти результатів роботи без розміщення застосунку у відкритому доступі.

Обрана технологічна основа дозволяє реалізувати повний цикл роботи дослідного веб-застосунку: приймання запиту від користувача, обробку даних серверною частиною, звернення до бази даних, формування відповіді, керування сесією та фіксацію подій. При цьому всі компоненти залишаються достатньо простими для аналізу та пояснення в межах кваліфікаційної роботи.

Отже, для реалізації веб-застосунку *Secure Notes* було обрано набір технологій, який забезпечує можливість створення компактної дослідної моделі з необхідними функціями безпеки.

3.3 Реалізація механізмів захисту

У межах практичної частини роботи було реалізовано демонстраційний веб-застосунок *Secure Notes*, до якого інтегровано модуль захисту від типових веб-атак. Основна ідея реалізації полягає в тому, що кожен запит користувача проходить попередню перевірку перед виконанням основної логіки веб-застосунку. Це дозволяє зменшити ризик передавання небезпечних даних до бази даних, *HTML*-шаблонів або механізму автентифікації.

Розроблений модуль захисту реалізує такі основні функції:

- валідацію введених даних;
- фільтрацію потенційно небезпечних шаблонів;
- захищену роботу з базою даних через параметризовані запити;
- хешування паролів користувачів;
- захист сесій користувача;

– перевірку *CSRF*-токенів;

– журналювання підозрілих подій.

У загальному вигляді процес обробки запиту можна подати як послідовність:

$$Q \rightarrow V(Q) \rightarrow A(Q) \rightarrow C(Q) \rightarrow D(Q) \rightarrow R,$$

де Q – вхідний *HTTP*-запит;

$V(Q)$ – перевірка та фільтрація введених даних;

$A(Q)$ – перевірка автентифікації користувача;

$C(Q)$ – перевірка *CSRF*-токена;

$D(Q)$ – безпечна взаємодія з базою даних;

R – результат обробки запиту.

Якщо на будь-якому етапі перевірки виявляється порушення правил безпеки, запит не передається до подальшої обробки. У такому разі користувачу відображається повідомлення про помилку, а відповідна подія записується до журналу безпеки.

3.3.1 Валідація та фільтрація введених даних

Першим механізмом захисту є перевірка введених даних. У веб-застосунку користувач вводить дані під час реєстрації, входу до системи та створення нотатки. Усі ці дані розглядаються як потенційно небезпечні, оскільки вони надходять із зовнішнього середовища.

Для перевірки введення у файлі *security.py* створено функцію *is_suspicious_input*(). Вона аналізує рядок і порівнює його з набором шаблонів, які можуть свідчити про спробу атаки. До таких шаблонів належать фрагменти *<script>*, *javascript:*, *onerror=*, *union select, drop table*, *' OR* '1'='1, *../* та інші конструкції, характерні для *XSS*, *SQL*-ін’єкцій, *File Inclusion* або *Command Injection*.

Фрагмент реалізації перевірки введених даних має такий вигляд:

```
SUSPICIOUS_PATTERNS = [
    r"(?i)<script",
    r"(?i)</script",
    r"(?i)javascript:",
    r"(?i)'\s*or\s*'1'\s*=\s*'1",
    r"(?i)union\s+select",
    r"(?i)drop\s+table",
    r"(?i)\.\./"
]
def is_suspicious_input(value):
    if not value:
        return False
    for pattern in SUSPICIOUS_PATTERNS:
        if re.search(pattern, value):
            return True
    return False
```

У цій функції використовується модуль *re*, який дозволяє виконувати пошук за регулярними виразами. Якщо введений користувачем рядок відповідає одному з небезпечних шаблонів, функція повертає значення *True*, а запит блокується.

Окремо реалізовано перевірку імені користувача під час реєстрації. Ім’я користувача повинно містити від 3 до 30 символів і складатися лише з літер, цифр або символу підкреслення. Такий підхід обмежує можливість введення спеціальних символів, які можуть використовуватися для атак.

```
def is_valid_username(username):
    if not username:
        return False
```

```
    pattern = r"^[A-Za-zA-Яа-яІіЇїЄєҐґ0-9_]{3,30}$"
    return re.match(pattern, username) is not None
```

Таким чином, валідація введення виконує функцію первинного захисного бар'єра. Вона не замінює інші механізми захисту, але дозволяє відхиляти очевидно небезпечні запити ще до їх передавання в серверну логіку.

3.3.2 Захищена робота з базою даних

Другим важливим механізмом є захищена робота з базою даних. У веб-застосунку *Secure Notes* база даних використовується для збереження облікових записів користувачів і нотаток. Основною загрозою під час взаємодії з базою даних є *SQL*-ін'єкція, яка виникає тоді, коли користувацькі дані безпосередньо додаються до *SQL*-запиту.

Для усунення цього ризику в роботі використано параметризовані *SQL*-запити. Їх суть полягає в тому, що структура *SQL*-команди задається окремо, а користувацькі значення передаються як параметри. У такому випадку введений користувачем рядок не може змінити логіку *SQL*-запиту.

Наприклад, пошук користувача за іменем реалізовано так:

```
def get_user_by_username(username):
    with get_connection() as connection:
        return connection.execute(
            "SELECT * FROM users WHERE username = ?",
            (username,)
        ).fetchone()
```

У цьому фрагменті символ ? використовується як місце підстановки параметра. Значення *username* передається окремо від SQL-команди. Тому навіть якщо користувач введе рядок ' *OR* '1'='1, він буде оброблений як звичайне текстове значення.

Аналогічний підхід застосовано під час створення користувача та збереження нотатки:

```
connection.execute(
   "INSERT INTO users (username, password_hash) VALUES (?, ?)",
   (username, password_hash)
)
connection.execute(
   "INSERT INTO notes (user_id, text) VALUES (?, ?)",
   (user_id, text)
)
```

Використання параметризованих запитів є основним засобом захисту бази даних у розробленому застосунку. Це дозволяє запобігти виконанню користувацького введення як частини *SQL*-коду.

3.3.3 Захист автентифікації та паролів

Для забезпечення захисту облікових записів користувачів у застосунку реалізовано безпечне зберігання паролів. Паролі не зберігаються у базі даних у відкритому вигляді. Під час реєстрації введений пароль перетворюється на хеш за допомогою функції *generate_password_hash*() з бібліотеки *Werkzeug*.

```
password_hash = generate_password_hash(password)
create_user(username, password_hash)
```

Під час входу користувача до системи введений пароль не порівнюється напряму з відкритим значенням. Замість цього використовується функція *check_password_hash*(), яка перевіряє відповідність введеного пароля збереженому хешу.

```
if not user or not check_password_hash(user["password_hash"], password):
    log_security_event(f"Invalid login attempt for username: {username}")
```

```
    flash("Невірний логін або пароль.", "error")
    return redirect(url_for("login"))
```

Такий підхід забезпечує захист облікових даних навіть у випадку несанкціонованого доступу до бази даних. Якщо база буде скомпрометована, зловмисник не отримає паролі користувачів у відкритому вигляді.

3.3.4 Захист сесій користувача

Після успішного входу до системи для користувача створюється сесія. У сесії зберігається ідентифікатор користувача та його ім’я:

```
session.clear()
session["user_id"] = user["id"]
session["username"] = user["username"]
```

Перед створенням нової сесії виконується *session.clear*(). Це дозволяє очистити попередні дані сесії та зменшити ризик використання залишкової інформації від попереднього користувача.

Для обмеження доступу до захищених сторінок реалізовано декоратор *login_required*. Він перевіряє, чи є в сесії ідентифікатор користувача. Якщо користувач не автентифікований, доступ до особистого кабінету блокується, а користувач перенаправляється на сторінку входу.

```
def login_required(route_function):
    @wraps(route_function)
    def wrapper(*args, **kwargs):
        if "user_id" not in session:
            log_security_event("Unauthorized access attempt to protected page")
            flash("Спочатку увійдіть у систему.", "error")
            return redirect(url_for("login"))
```

```
        return route_function(*args, **kwargs)
    return wrapper
```

Цей механізм виконує функцію базового контролю доступу. Він гарантує, що сторінка особистого кабінету доступна лише після успішної автентифікації.

3.3.5 Захист від *CSRF*-атак

Для захисту форм від *CSRF*-атак реалізовано механізм *CSRF*-токенів. Під час роботи із формами сервер створює унікальний токен і зберігає його в сесії користувача. Цей токен додається до *HTML*-форми у вигляді прихованого поля.

Генерація токена реалізована у файлі *security.py*:

```
def generate_csrf_token():
    if "_csrf_token" not in session:
        session["_csrf_token"] = secrets.token_urlsafe(32)
    return session["_csrf_token"]
```

Для перевірки токена використовується функція:

```
def validate_csrf_token(token):
    if not token:
        return False
    return token == session.get("_csrf_token")
```

У *HTML*-формах токен передається через приховане поле:

```
<input type="hidden" name="csrf_token" value="{{ csrf_token() }}">
```

Під час обробки *POST*-запиту сервер перевіряє отриманий токен. Якщо токен відсутній або не відповідає значенню, збереженому в сесії, запит блокується:

```
csrf_token = request.form.get("csrf_token")
if not validate_csrf_token(csrf_token):
    log_security_event("CSRF validation failed during note creation")
    abort(400, description="Некоректний CSRF-токен.")
```

Таким чином, навіть якщо зловмисник спробує змусити користувача виконати небажану дію через сторонню сторінку, запит без коректного *CSRF*-токена не буде виконаний.

3.3.6 Фільтрація нотаток і захист від *XSS*

Окремий механізм захисту реалізовано для форми додавання нотаток. Оскільки текст нотатки вводиться користувачем і потім відображається на сторінці, він може бути використаний для *XSS*-атаки. Тому перед збереженням нотатки виконується перевірка вмісту.

```
note_text = request.form.get("note", "").strip()
if is_suspicious_input(note_text):
    log_security_event(
        f"Suspicious note input from user {session.get('username')}: {note_text}"
    )
    flash("Нотатка містить потенційно небезпечний вміст і не була збережена.", "error")
    return redirect(url_for("dashboard"))
```

Якщо в тексті нотатки виявлено небезпечний шаблон, наприклад *<script>alert('XSS')</script>,* нотатка не зберігається в базі даних. Користувач отримує повідомлення про помилку, а подія записується в журнал безпеки.

Додатково *HTML*-шаблонізатор *Jinja*2 за замовчуванням екранує виведені значення. Це означає, що навіть текстові дані, які відображаються на сторінці, не повинні виконуватися браузером як *HTML* або *JavaScript*-код. У

розробленому застосунку цей механізм поєднується з попередньою фільтрацією введення.

3.3.7 Журналювання подій безпеки

Для контролю підозрілих дій у застосунку реалізовано журналювання. Усі небезпечні або некоректні дії записуються до файлу *logs/security.log*. Для цього використовується функція *log_security_event*().

```
def log_security_event(message):
    logging.warning(message)
```

Журналювання застосовується у таких випадках:

- введення підозрілого логіна;
- невдала спроба входу;
- спроба доступу до захищеної сторінки без автентифікації;
- спроба збереження *XSS*-навантаження;
- помилка *CSRF*-перевірки.

Загальна логіка роботи модуля захисту

Узагальнений алгоритм роботи модуля захисту веб-застосунку можна подати так:

1) користувач надсилає *HTTP*-запит;
2) система отримує параметри форми, *cookies*, сесійні дані та *CSRF*-токен;
3) виконується перевірка *CSRF*-токена для *POST*-запитів;
4) виконується перевірка введених даних;
5) перевіряється стан автентифікації користувача;
6) *SQL*-запити виконуються лише у параметризованому вигляді;
7) у разі небезпечного введення запит блокується;
8) результат обробки або подія безпеки записується до журналу;
9) безпечний запит передається до основної логіки веб-застосунку.

Таким чином, у розробленому веб-застосунку *Secure Notes* реалізовано комплекс базових механізмів захисту, які охоплюють перевірку введених даних, захищену роботу з базою даних, хешування паролів, контроль сесій, *CSRF*-захист, фільтрацію небезпечного вмісту та журналювання подій безпеки. Ці механізми формують практичну модель захисту веб-застосунку, яка в подальшому може бути перевірена шляхом експериментального тестування типових веб-атак.

Висновок до розділу

У програмній реалізації впроваджено комплекс базових механізмів захисту: валідацію та фільтрацію введених даних, параметризовані *SQL*-запити, хешування паролів, контроль сесій, *CSRF*-захист, фільтрацію *XSS*-вмісту та журналювання підозрілих подій. Реалізовані механізми дозволяють знизити ризик успішної реалізації типових веб-атак і забезпечують безпечну взаємодію користувача з веб-застосунком.

РОЗДІЛ 4 ЕКСПЕРИМЕНТАЛЬНЕ ДОСЛІДЖЕННЯ ТА ОЦІНКА ЕФЕКТИВНОСТІ

4.1 Методика тестування безпеки

Метою експериментального дослідження є перевірка працездатності розробленого веб-застосунку *Secure Notes* та оцінка ефективності реалізованих механізмів захисту від типових веб-атак. У межах тестування перевіряється, чи здатна система коректно обробляти легальні запити користувача та блокувати небезпечні введення, які можуть бути використані для *SQL*-ін'єкцій, *XSS*, *CSRF*-атак або порушення механізмів автентифікації [18-20].

Об'єктом тестування є розроблений у межах практичної частини веб-застосунок *Secure Notes*. Застосунок реалізовано на основі клієнт-серверної моделі та запущено в локальному середовищі за адресою *http*://127.0.0.1:5000. Він містить головну сторінку, сторінку реєстрації користувача, сторінку входу, особистий кабінет користувача, форму додавання нотаток, базу даних *SQLite* та журнал подій безпеки.

Запуск веб-застосунку у локальному середовищі наведено на рисунку 4.1.

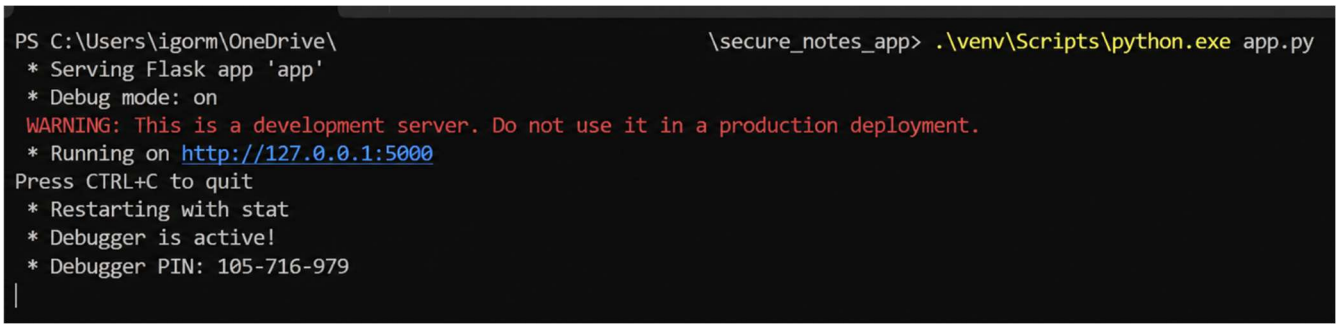

Рисунок 4.1 – Запуск веб-застосунку Secure Notes у локальному середовищі

Як видно з рисунку 4.1, застосунок запускається за допомогою локального веб-сервера *Flask*. Локальний запуск дає змогу безпечно виконувати тестові запити та перевіряти реакцію системи без розміщення застосунку у відкритому доступі.

Головну сторінку веб-застосунку *Secure Notes* наведено на рисунку 4.2.

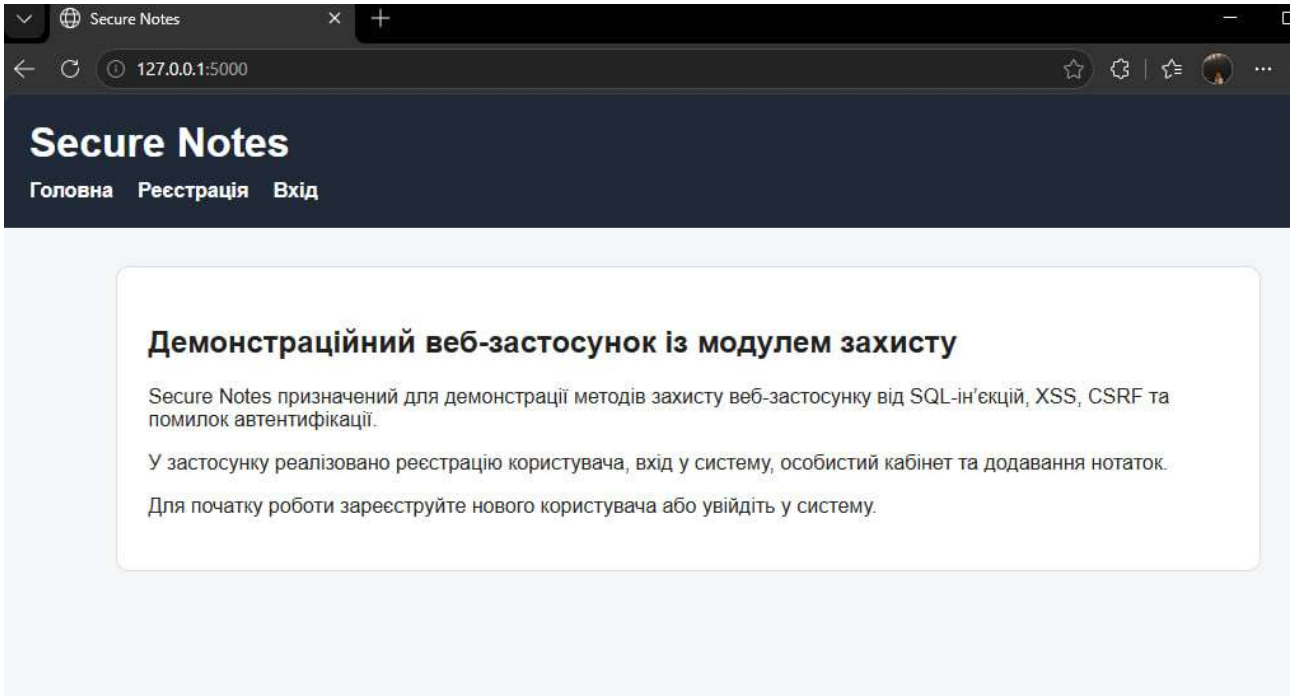

Рисунок 4.2 – Головна сторінка веб-застосунку *Secure Notes*

На рисунку 4.2 показано головну сторінку застосунку, яка містить навігаційні елементи для переходу до реєстрації та входу користувача. Ця сторінка використовується як початкова точка взаємодії з веб-застосунком під час тестування.

Сторінку реєстрації користувача наведено на рисунку 4.3 там наведено форму реєстрації, у якій користувач вводить ім'я та пароль. Під час тестування ця сторінка використовується для перевірки створення облікового запису, хешування пароля та наявності *CSRF*-токена у формі.

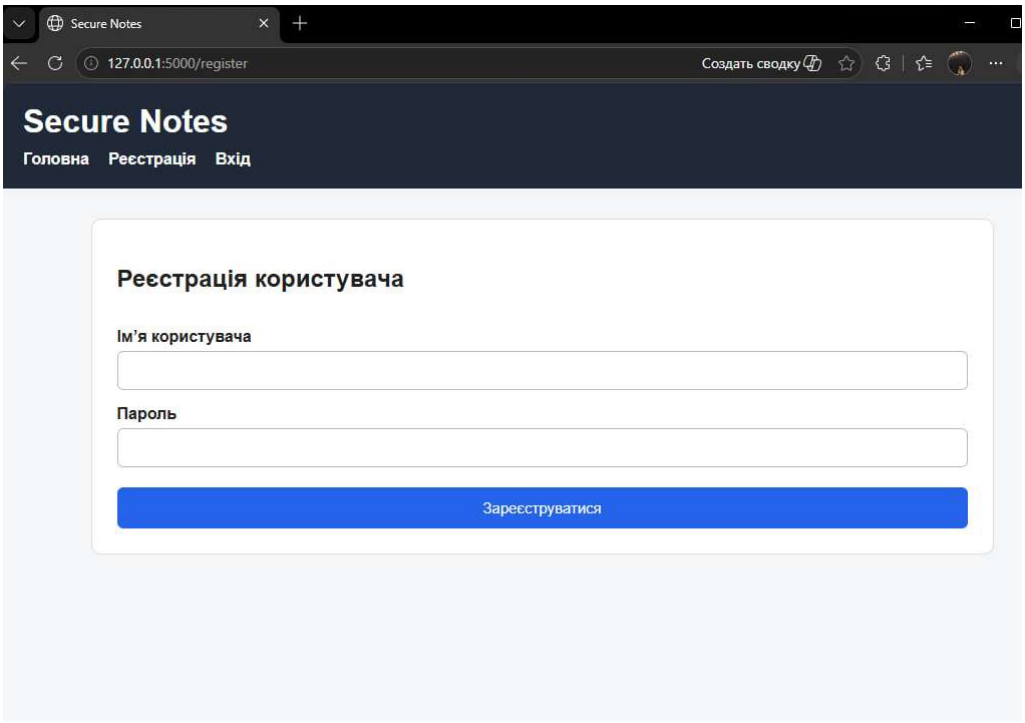

Рисунок 4.3 – Сторінка реєстрації користувача

Сторінку входу користувача до системи наведено на рисунку 4.4.

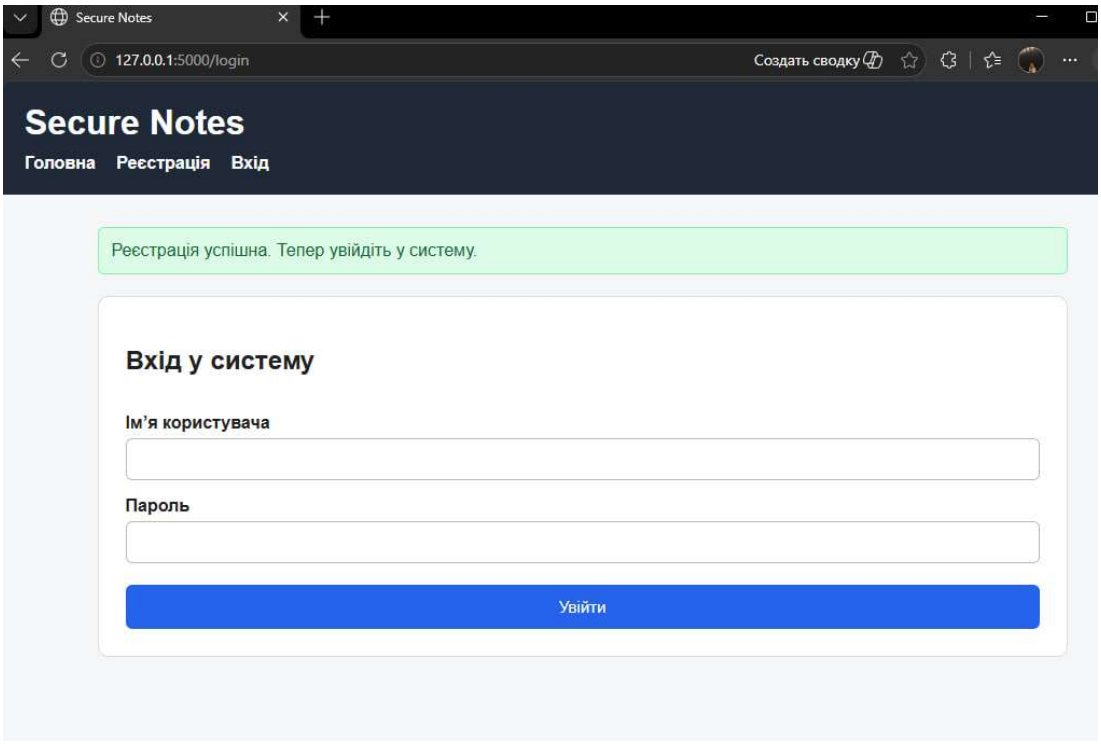

Рисунок 4.4 – Сторінка входу користувача

На рисунку 4.4 показано сторінку автентифікації користувача. Ця форма використовується для перевірки механізму входу, захисту від *SQL*-ін’єкцій і правильності обробки невірних або підозрілих облікових даних.

На рисунку 4.5 наведено особистий кабінет користувача, у якому реалізовано форму додавання нотаток. Саме ця форма використовується для перевірки фільтрації введених даних і захисту від *XSS*-атаки.

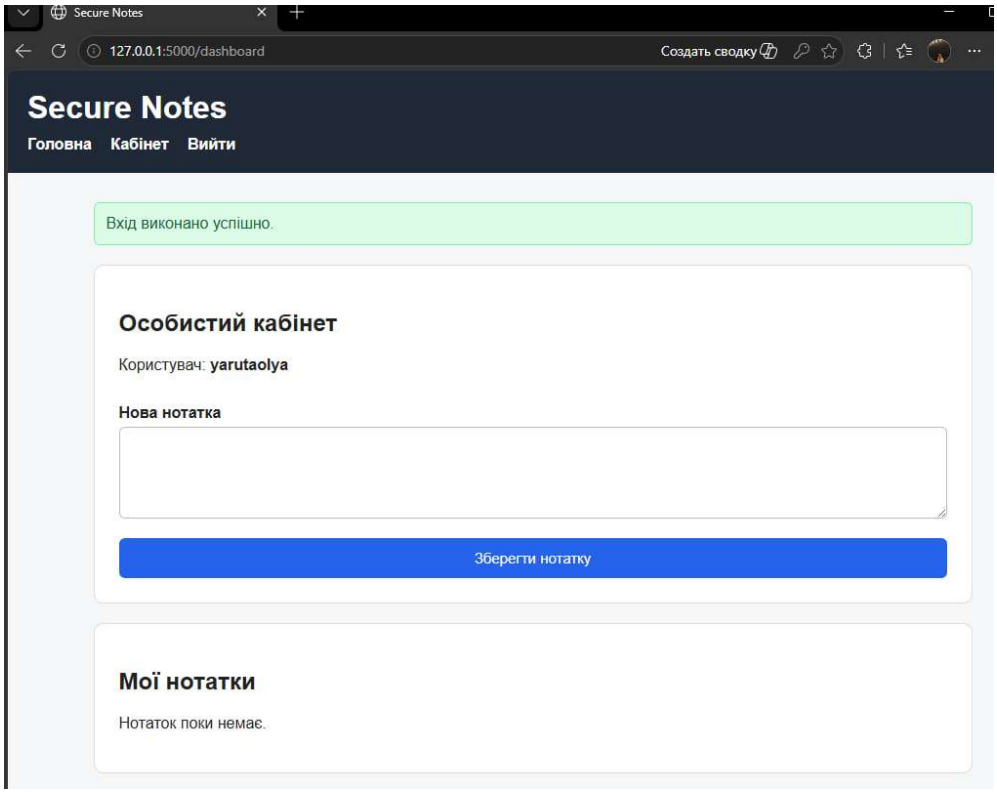

Рисунок 4.5 – Особистий кабінет користувача *Secure Notes*

Для тестування безпеки застосовується комбінована методика, яка включає автоматизоване визначення потенційних вразливостей і ручну перевірку реалізованих механізмів захисту. Автоматизоване тестування доцільно виконувати за допомогою програмного засобу *OWASP ZAP*, який призначений для аналізу безпеки веб-застосунків. *OWASP ZAP* дозволяє виконувати сканування веб-сторінок, аналіз *HTTP*-запитів і відповідей, перевірку заголовків безпеки, виявлення помилок конфігурації та пошук типових вразливостей.

У межах роботи *OWASP ZAP* розглядається як інструмент автоматичного визначення вразливостей, а розроблений застосунок *Secure Notes* – як об’єкт аналізу. Ціль сканування встановлюється як локальна адреса:

http://127.0.0.1:5000

Сканування локального застосунку дозволяє перевірити доступні сторінки, HTTP-форми, заголовки відповідей і потенційні точки введення даних. Оскільки застосунок є навчально-дослідним і розгорнутий локально, тестування не створює ризиків для зовнішніх інформаційних ресурсів.

Таким чином, методика тестування безпеки включає опис об’єкта тестування, запуск веб-застосунку в локальному середовищі, використання програмного забезпечення автоматичного визначення вразливостей *OWASP ZAP* та ручну перевірку реалізованих механізмів захисту. Такий підхід дозволяє оцінити як загальний стан безпеки веб-застосунку, так і практичну ефективність окремих засобів захисту від *SQL*-ін’єкцій, *XSS, CSRF* та помилок автентифікації.

4.2. Моделювання веб-атак

Після визначення методики тестування було проведено моделювання типових веб-атак на розроблений веб-застосунок *Secure Notes*. Метою моделювання є перевірка того, як система реагує на небезпечні вхідні дані та чи спрацьовують реалізовані механізми захисту: фільтрація введення, параметризовані *SQL*-запити, *CSRF*-токени, хешування паролів і журналювання подій безпеки.

Моделювання виконувалося у локальному середовищі. Веб-застосунок був запущений на локальному сервері *Flask* за адресою *http*://127.0.0.1:5000. Такий підхід дозволяє безпечно проводити тестування, оскільки всі запити надсилаються лише до власної дослідної системи.

Першим етапом моделювання було тестування форми входу на стійкість до *SQL*-ін’єкції. Для цього у було використано програмне забезпечення *SQLmap*

для поля логіну. Результати перевірки (рис. 4.6) показали, що система стійка для відомих SQL-ін’єкцій.

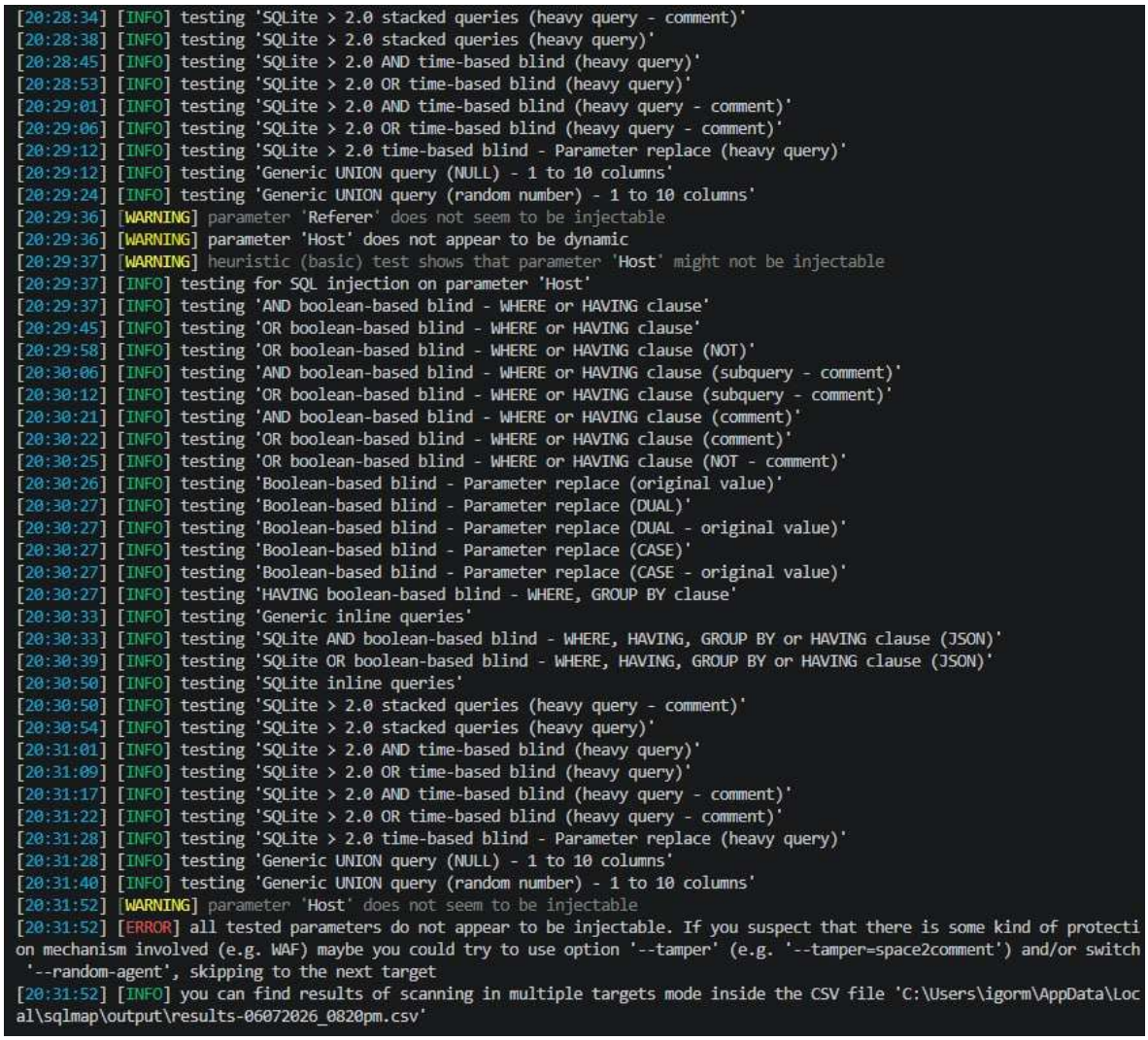

Рисунок 4.6 – Результат перевірки захисту від SQL-ін’єкції

Як видно з рисунку 4.6, введення *SQL*-ін’єкційного рядка не дозволило виконати вхід до системи. Застосунок виявив небезпечне введення та відхилив запит. Це підтверджує працездатність фільтрації введених даних і використання параметризованих *SQL*-запитів під час роботи з базою даних.

Другим етапом було моделювання *XSS*-атаки. Для цього в особистому кабінеті користувача у поле додавання нотатки було введено фрагмент JavaScript-коду або вставлення об’єкту:

<script>alert('XSS')</script>

<img src=1 onerror=alert(1)>

<input autofocus onfocus=alert(1)>

<select autofocus onfocus=alert(1)>

<textarea autofocus onfocus=alert(1)>

<keygen autofocus onfocus=alert(1)>

<video/poster/onerror=alert(1)>

<video><source onerror="javascript:alert(1)">

<video src=_ onloadstart="alert(1)">

<details/open/ontoggle="alert`1`">

<audio src onloadstart=alert(1)>

Метою цього тесту було перевірити, чи дозволить система зберегти та виконати шкідливий сценарій у браузері користувача. У разі відсутності захисту такий код міг би виконатися при перегляді сторінки з нотатками.

Результат перевірки захисту від *XSS*-атаки наведено на рисунку 4.7.

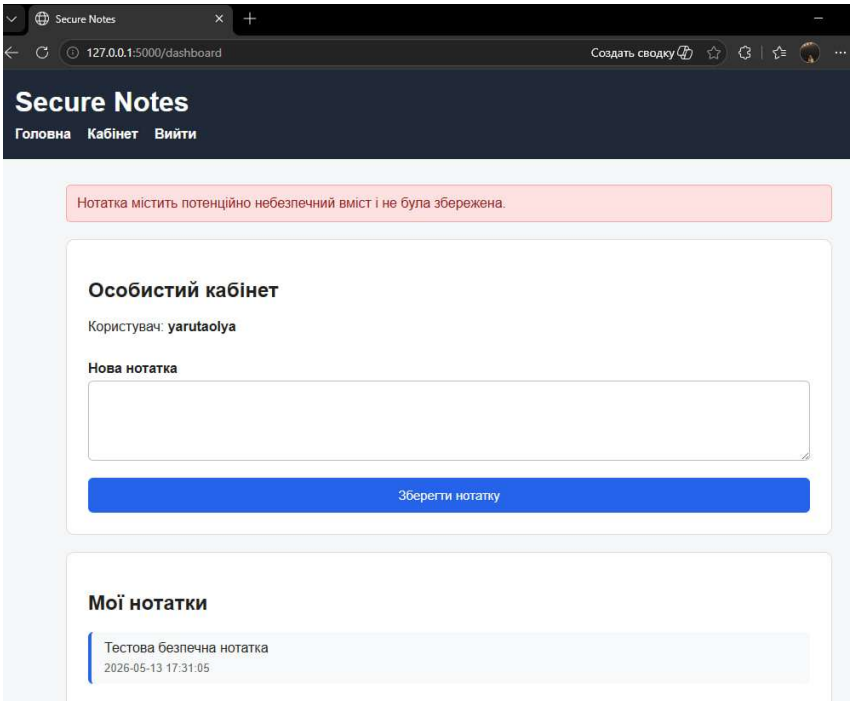

Рисунок 4.7 – Результат перевірки захисту від *XSS*-атаки

Введене *XSS*-навантаження не було збережене як звичайна нотатка та не виконалося у браузері. Система визначила наявність потенційно небезпечного вмісту та заблокувала його подальшу обробку. Це підтверджує роботу механізму фільтрації введених даних.Третім етапом було моделювання *CSRF*-атаки. Для цього з *HTML*-форми додавання нотатки було вручну видалено приховане поле з *CSRF*-токеном. Після цього була виконана спроба надіслати форму без коректного токена. Такий тест дозволяє перевірити, чи виконує сервер перевірку справжності *POST*-запиту.

Результат блокування запиту з некоректним *CSRF*-токеном наведено на рисунку 4.8. Сервер відхилив запит без правильного *CSRF*-токена. Це означає, що критичні дії у веб-застосунку не виконуються лише на основі факту активної сесії користувача. Для виконання операції також потрібен коректний токен, сформований сервером.

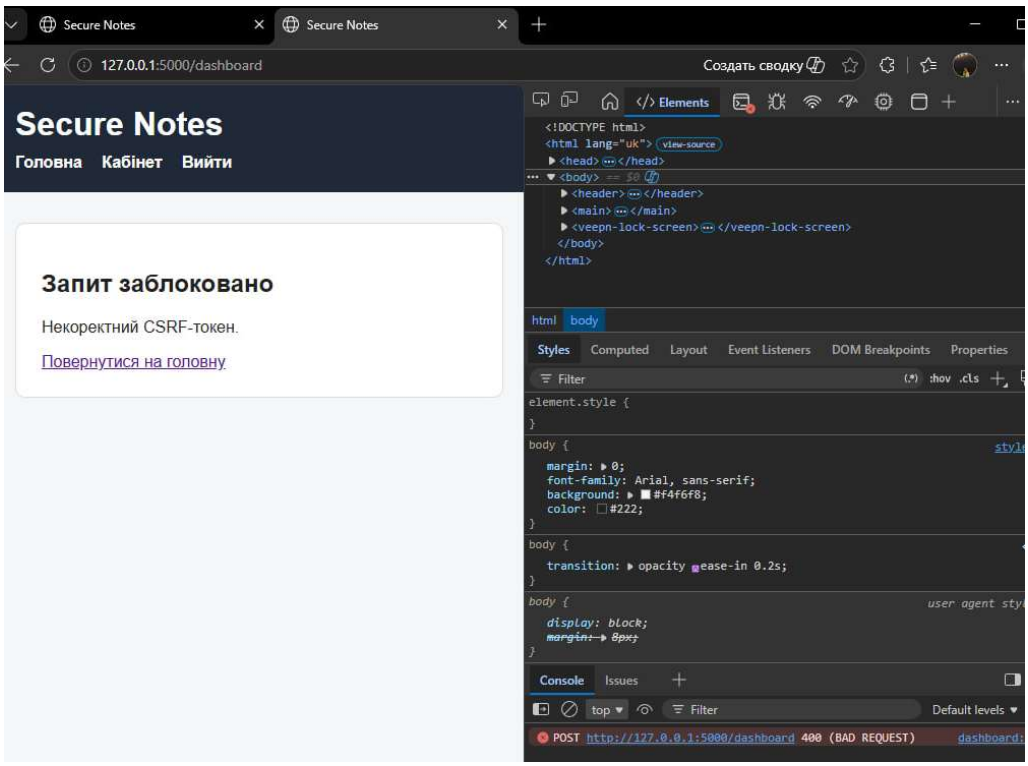

Рисунок 4.8 – Блокування запиту з некоректним *CSRF*-токеном

Четвертим етапом було перевірено спосіб зберігання пароля користувача. Для цього після реєстрації користувача було переглянуто запис у таблиці *users* бази даних *database.db*. Метою перевірки було встановити, чи зберігається пароль у відкритому вигляді або у вигляді хешу.

Результат перевірки збереження пароля наведено на рисунку 4.9.

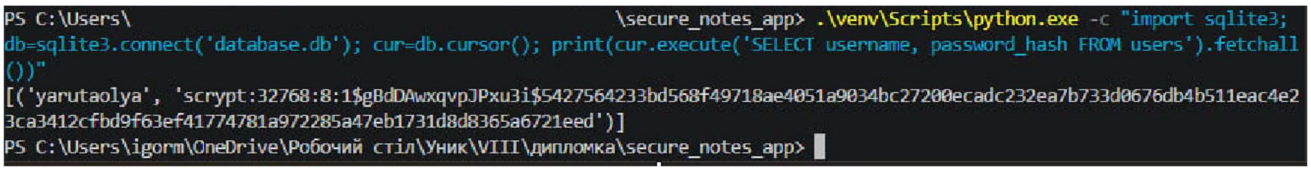

Рисунок 4.9 – Збереження пароля користувача у вигляді хешу

Як видно з рис. 4.9, пароль користувача не зберігається у відкритому вигляді. У базі даних міститься хеш пароля, сформований засобами бібліотеки *Werkzeug*. Це зменшує ризик компрометації облікових даних у випадку несанкціонованого доступу до бази даних.

П’ятим етапом було перевірено журналювання подій безпеки. Після виконання тестових запитів система створила записи у файлі *logs/security.log*. У журналі фіксуються підозрілі введення, спроби *SQL*-ін’єкції, *XSS*-навантаження та помилки *CSRF*-перевірки.

Приклад журналу подій безпеки наведено на рисунку 4.10.

Як видно з рисунку 4.10, система фіксує події, пов’язані з підозрілими діями користувача. У журналі зафіксовано спробу введення *SQL*-ін’єкційного рядка, спробу *XSS*-навантаження та помилку *CSRF*-перевірки. Це підтверджує, що розроблений веб-застосунок не лише блокує небезпечні запити, а й зберігає інформацію про них для подальшого аналізу.

Таким чином, у процесі моделювання веб-атак було перевірено основні механізми захисту, реалізовані у веб-застосунку *Secure Notes*. Результати тестування показали, що система блокує *SQL*-ін’єкційні запити, не допускає виконання *XSS*-навантаження, відхиляє запити без коректного *CSRF*-токена, зберігає паролі у вигляді хешів і фіксує підозрілі події у журналі безпеки.

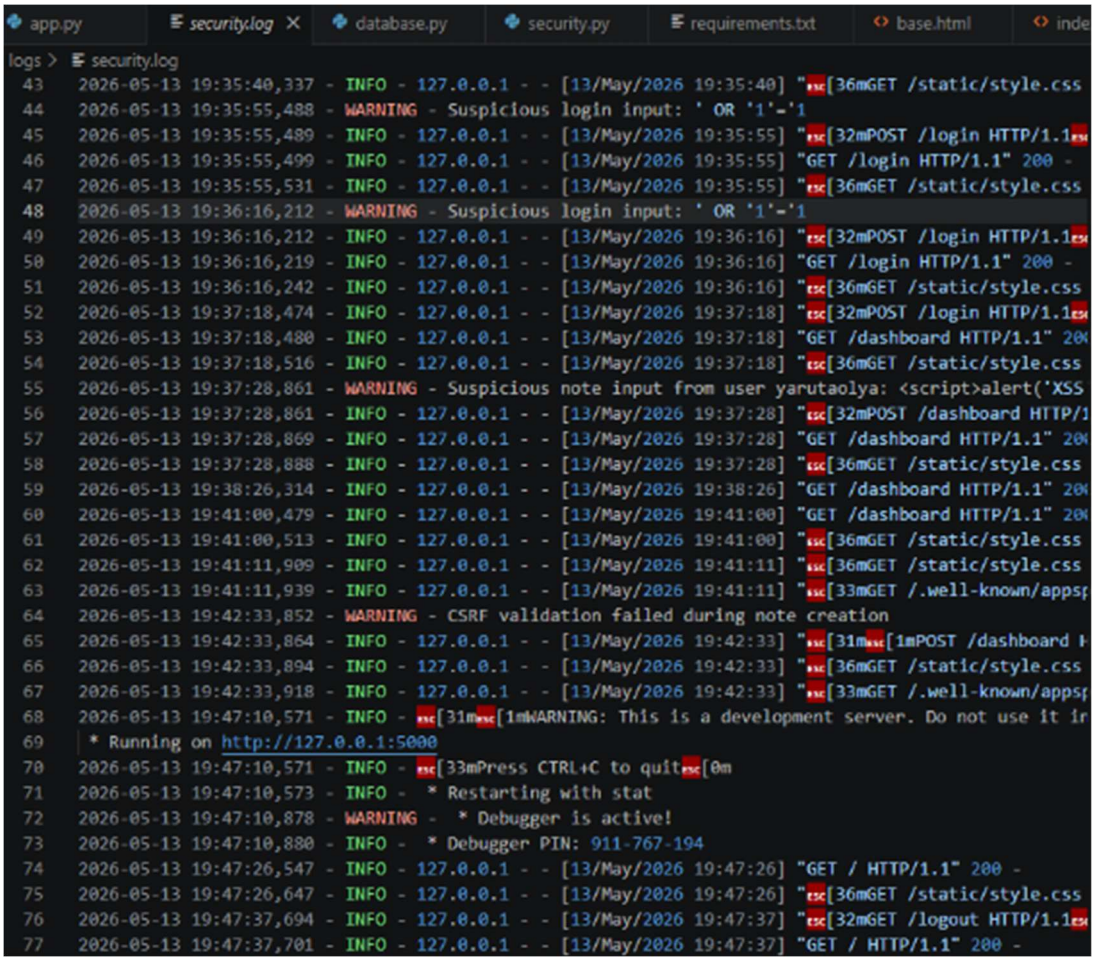

Рисунок 4.10 – Приклад журналу подій безпеки веб-застосунку

Це підтверджує працездатність розробленої моделі захисту веб-застосунку від типових веб-атак.

Висновок до розділу

У процесі моделювання атак встановлено, що застосунок успішно блокує спроби SQL-ін’єкцій, XSS-атак і запити без коректного CSRF-токена, забезпечує безпечне зберігання паролів у вигляді хешів та фіксує підозрілі дії в журналі безпеки. Отримані результати підтверджують працездатність розробленої моделі захисту та доцільність використання впроваджених механізмів для підвищення безпеки веб-застосунків.

ВИСНОВОК

У кваліфікаційній роботі було проведено дослідження та розробку методів захисту веб-застосунків від типових веб-атак. У процесі виконання роботи було проаналізовано архітектуру сучасних веб-застосунків. Було розглянуто основні принципи веб-технологій, зокрема клієнт-серверну модель, протоколи *HTTP* і *HTTPS*, сесії, *cookies, API*, механізми автентифікації та авторизації.

У роботі було класифіковано типові кіберзагрози для веб-застосунків. До основних загроз віднесено *SQL*-ін'єкції, *XSS, CSRF, Command Injection, File Inclusion* та *Broken Authentication*. Проведений аналіз показав, що більшість таких атак пов'язана з недостатньою перевіркою введених даних, помилками автентифікації, некоректною авторизацією та небезпечною передачею користувацьких параметрів до серверної логіки або бази даних.

У практичній частині роботи було розроблено демонстраційний веб-застосунок *Secure Notes*. У розробленому веб-застосунку було реалізовано основні механізми захисту: перевірку введених даних, фільтрацію небезпечних шаблонів, параметризовані *SQL*-запити, *CSRF*-токени, хешування паролів, захист сесій користувача та журналювання підозрілих дій.

Було проведено експериментальне тестування розробленого застосунку. Для перевірки захисту було змодельовано *SQL*-ін'єкцію, *XSS*-атаку та запит без коректного *CSRF*-токена. Також було перевірено спосіб зберігання пароля користувача та роботу журналу подій безпеки. Результати тестування показали, що *SQL*-ін'єкційний рядок не дозволяє обійти автентифікацію, *XSS*-навантаження не виконується у браузері, запит без правильного *CSRF*-токена блокується сервером, пароль зберігається у вигляді хешу, а підозрілі дії фіксуються у журналі.

ПЕРЕЛІК ПОСИЛАНЬ

1. *Grinberg M. Flask Web Development: Developing Web Applications with Python. – 2nd ed. – Sebastopol : O'Reilly Media,* 2018. – 316 *p.*

2. *Abbott,M.L., Fisher,M.T. The Art of Scalability: Scalable Web Architecture, Processes, and Organizations for the Modern Enterprise. Addison-Wesley Professional. Pearson Education,* 2015. 592*p*

3. *Atchison L. Architecting for Scale: How to Maintain High Availability and Manage Risk in the Cloud. O'Reilly Media*. 2020. 400 *p*.

4. Носенко М. В. Методи та засоби захисту інформації в автоматизованих системах : автореф. дис. *URL*: *https://krs.chmnu.edu.ua/jspui/bitstream/123456789/635/1/%D0%90%D0%B2%D1%82%D0%BE%D1%80%D0%B5%D1%84%D0%B5%D1%80%D0%B0%D1%82%20%D0%9D%D0%BE%D1%81%D0%B5%D0%BD%D0%BA%D0%BE%20%D0%9C.%D0%92.%20403.pdf*

5. *Hoffman A. Web Application Security: Exploitation and Countermeasures for Modern Web Applications. Sebastopol : O'Reilly Media, 2020. 400 p.*

6. Базові поняття кібербезпеки та захисту інформації. *URL*: *https://lib.iitta.gov.ua/id/eprint/107151/1/%D0%91%D0%90%D0%97%D0%9E%D0%92%D0%86_%D0%9F%D0%9E%D0%9D%D0%AF%D0%A2%D0%A2%D0%AF.pdf*

7. *Research Europe. Cybersecurity and Secure Software Engineering Trends. 2025. URL: https://researcheurope.org/wp-content/uploads/2025/10/re-10.10.2025.pdf#page=18*

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

9. *Hoffman A. Web Application Security: Exploitation and Countermeasures for Modern Web Applications. – Sebastopol : O'Reilly Media, 2020. – 400 p.*

10. *Nasenok K., Voitsekhovska M. Client-side rendering issues in the modern worldwide networ. Technical sciences and technologies. 2024. Is. 4(38), Pp. 197–207. DOI: https://doi.org/10.25140/2411-5363-2024-4(38)-197-207.*

11. Семчишин П. М. *Architectural solutions for secure software systems* // Інформаційні системи та технології. 2025. С. 340–342. URL: *https://elartu.tntu.edu.ua/bitstream/lib/51634/2/ISTCYSS_2025_Semchyshyn_P_M-Architectural_solutions_340-342.pdf*

12. *Viega J., McGraw G. Building Secure Software: How to Avoid Security Problems the Right Way. – Boston : Addison-Wesley, 2002. – 512 p.*

13. Halfond W. G., Viegas J., Orso A. A Classification of SQL Injection Attacks and Countermeasures // Proceedings of the IEEE International Symposium on Secure Software Engineering. – 2006. – P. 13–15.

14. OWASP Foundation. OWASP Top 10: The Ten Most Critical Web Application Security Risks. URL: https://owasp.org/www-project-top-ten

15. OWASP Foundation. Cross Site Scripting Prevention Cheat Sheet URL: https://cheatsheetseries.owasp.org

16. OWASP Foundation. SQL Injection Prevention Cheat Sheet URL: https://cheatsheetseries.owasp.org

17. Matthes E. Python Crash Course. – 3rd ed. – San Francisco : No Starch Press, 2023. – 544 p.

18. Філіпчук М. М. Алгоритми тестування безпеки веб-ресурсів. Тернопіль, 2022. 58 с.

19. Толкачова А. Методи для тестування безпеки веб-застосунків з використанням сучасних інструментів // Кібербезпека: освіта, наука, техніка. 2024. № 2 (26). С. 245–257.

20. Трофименко О. Г. Тестування та забезпечення якості програмних систем : навч.-метод. посіб. Одеса : НУ «ОЮА», 2024. 186 с.