Хмельницький національний університет
Факультет інформаційних технологій
Кафедра кібербезпеки

КВАЛІФІКАЦІЙНА РОБОТА

Сафонова Владислава Руслановича

на здобуття ступеня вищої освіти Бакалавра

Система аудиту та аналізу безпеки вебдодатків

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

Шифр КРБКБ.220164.22.01.07 ПЗ

Виконав студент 3 курсу група КБс-22-1 Владислав САФОНОВ
Керівник д-р фолософії Микола СТЕЦЮК
Нормоконтролер старший викладач Сергій МОСТОВИЙ

До захисту допускаю:
Завідувач кафедри кібербезпеки Юрій КЛЬОЦ
2 06 2025 р.

Хмельницький 2025

ХМЕЛЬНИЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

Факультет Інформаційних технологій

Кафедра Кібербезпеки

Рівень вищої освіти Бакалавр

Галузь знань 12 – Інформаційні технології

Спеціальність 125 – Кібербезпека

Освітня програма Кібербезпека

ЗАТВЕРДЖУЮ

Завідувач кафедри кібербезпеки

Юрій КЛЬОЦ

15 лютого 2025 р.

З А В Д А Н Н Я
НА КВАЛІФІКАЦІЙНУ РОБОТУ

Сафонову Владиславу Руслановичу

1 Тема роботи Система аудиту та аналізу безпеки вебдодатків

Керівник роботи д-р філософії, Стецюк Микола Васильович

Затверджено наказом ректора університету від 7 лютого 2025 № 23

2 Строк подання студентом кваліфікаційної роботи на кафедру

3 Вихідні дані до роботи

Системи реалізовано у вигляді вебзастосунку на базі фреймворку Ruby on Rails, метою проєкту є виявлення таких вразливостей як SQL-injection та XSS результат сканування формується у вигляді звіту формату PDF який буде автоматично завантажуватись користувачу після закінчення сканування та для покращення конфіденційності даних буде захищений паролем який буде виведений користувачу.

4 Зміст пояснювальної записки (перелік питань, які потрібно розробити)

Огляд існуючих вразливостей, аналіз вже існуючих систем виявлення вразливостей, планування реалізації системи, створення алгоритму роботи програми, тестування роботи системи

5 Перелік графічного матеріалу (із зазначенням обов'язкових креслень)

Алгоритм роботи системи, діаграма прецедентів, діаграма класів

6 Консультанти розділів кваліфікаційної роботи

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

7 Дата видачі завдання 16 лютого 2025 р.

КАЛЕНДАРНИЙ ПЛАН

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

Студент В. Саф Владислав САФОНОВ

Керівник кваліфікаційної роботи Микола СТЕЦЮК

АНОТАЦІЯ

Тема кваліфікаційної роботи: Система аудиту та аналізу безпеки вебдодатків.

Автор роботи: Сафонов Владислав Русланович.

Керівник роботи: Стецюк Микола Васильович.

Пояснювальна записка: 72 с., 2 додатки, 51 рисунків, 46 джерел.

Графічна частина: 3 плакати, 12 презентаційних слайдів.

АЛГОРИТМ РОБОТИ СИСТЕМИ, ДІАГРАМА ПРЕЦЕДЕНТІВ, ДІАГРАМА КЛАСІВ.

Кваліфікаційна робота бакалавра присвячена розробці системи аудиту та аналізу безпеки вебдодатків.

В роботі розроблено вебзастосунок, який перевіряє інші вебзастосунки на найпоширеніші вразливості такі як SQL-ін'єкція та XSS за допомогою перевірки через форму авторизації. Цей інструмент дозволяє автоматизувати процес тестування безпеки, значно знижуючи час, необхідний для виявлення потенційних вразливостей. Застосунок здійснює сканування через запити до вебсервісу, аналізуючи відповіді та шукаючи аномалії, які можуть свідчити про присутність вразливостей. Результати тестування представлені у вигляді запаролених звітів у форматі PDF.

27.05.2025 В. Сафонов

ABSTRACT

Subject of qualification work: Web application security audit and analysis system.

Author: Safonov Vladyslav Ruslanovych.

Head of work: Stetsiuk Mykola Vasylovych.

Explanatory note: 72 pages, 2 appendices, 51 drawings, 46 sources.

Graphic part: 3 posters, 12 presentation slides.

SYSTEM OPERATION ALGORITHM, USE CASE DIAGRAM, CLASS DIAGRAM.

The bachelor's qualification work is dedicated to the development of a system for auditing and analyzing the security of web applications.

The paper developed a web application that checks other web applications for the most common vulnerabilities such as SQL injection and XSS using a form-based authentication check. This tool allows you to automate the security testing process, significantly reducing the time required to identify potential vulnerabilities. The application scans through requests to a web service, analyzing responses and looking for anomalies that may indicate the presence of vulnerabilities. The test results are presented as password-protected reports in PDF format.

27.05.2025 _____

ЗМІСТ

ВСТУП

В сучасному світі в якому технології прогресують вже не роками а годинами важко оминути їх використання. Велика кількість сервісів які надають послуги для людей підлаштовувалися роками до клієнтів. Сьогодні більшість з них у вигляді веб сайтів тому через великий попит на них, з'являються люди які бажають порушити функціонування цих систем різними методами, мотивацією для них стає грошова винагорода або просто на сучасному слензі “для фану”. Через таку ситуацію розробникам веб сайтів потрібно серйозно поставитись до безпеки їхніх застосунків, оскільки в разі несанкціонованого доступу до даних, компанія на яку працюють дані програмісти може понести великі грошові втрати. Особливо тяжкі наслідки отримують компанії з ЄС оскільки в них запроваджений “Загальний регламент про захист даних (GDPR)”.

Одним із найпоширеніших та водночас небезпечних типів вразливостей у сфері веббезпеки є Cross-Site Scripting (XSS), що дослівно перекладається як міжсайтове скриптування. Його популярність серед зловмисників пояснюється кількома факторами: по-перше, XSS доволі просто реалізувати за умови наявності навіть базових знань JavaScript, по-друге велика кількість сучасних вебзастосунків досі залишається вразливою до цього типу атак через недосконаліть реалізації обробки користувацького вводу або використання застарілих бібліотек. З практичної точки зору, ця вразливість виникає тоді, коли дані, введені користувачем, виводяться на сторінку без належної фільтрації чи екранування, дозволяючи вставляти та виконувати сторонній скрипт у браузері жертви. Зловмисник у такий спосіб може викрасти cookie-файли, сеансові токени, змінити DOM-структуру, перенаправити користувача на фішингову сторінку або здійснити будь-які інші дії, що призводять до втрати конфіденційності чи цілісності даних.

Найчастіше джерелом XSS стають недоопрацьовані клієнтські бібліотеки або шаблони, які не враховують новітні методики безпеки, або просто не оновлювались тривалий час. Проблема ускладнюється тим, що в багатьох

проєктах використовується величезна кількість залежностей, і стежити за їх актуальністю вручну вкрай складно. Усе це у сукупності створює умови, за яких навіть досвідчений розробник може не помітити потенційно небезпечні ділянки коду, не кажучи вже про новачків, для яких поняття безпечної обробки введених даних іноді взагалі є чимось другорядним.

У контексті сучасної розробки виникає об’єктивна потреба у впровадженні спеціалізованих інструментів, які здатні автоматизувати перевірку вебзастосунків на предмет XSS та інших вразливостей. Така автоматизація дозволяє значно підвищити рівень безпеки проєкту ще на етапі розробки, до моменту розгортання в продуктивному середовищі. Ручна перевірка на вразливості надзвичайно трудомісткий і рутинний процес, який потребує часу, уважності, знань та великого досвіду, а будь-яка помилка чи пропущена деталь може мати фатальні наслідки. Для новачка цей процес взагалі може стати непосильним, і саме тут автоматизовані інструменти стають незамінними помічниками. Вони не лише пришвидшують виявлення критичних помилок, а й формують звички правильного кодування, вказуючи на слабкі місця та рекомендуючи кращі практики.

З огляду на це, створення власної системи або використання вже існуючих рішень для сканування на XSS не є розкішшю чи другорядним завданням, а стає нагальною необхідністю. Це інвестиція, яка повертається багатократно за рахунок збереженого часу, підвищення довіри користувачів, уникнення репутаційних втрат та, врешті-решт, уникнення фінансових санкцій, які можуть накласти регулятори в разі витоку даних.

Метою нашої роботи буде створення системи аудиту та аналізу безпеки вебзастосунків, що може виявляти та видавати звітність користувачу по його проекту на найпопулярніші вразливості які включають в себе XSS, SQL-ін’єкції та інші, що зможе зберегти як час працівника так і гроші компанії на яку працює даний спеціаліст.

1 ОГЛЯД ІСНУЮЧИХ ІНСТРУМЕНТІВ ТА ПЛАНУВАННЯ ВЛАСНОГО

1.1 Огляд найпоширеніших вразливостей

Початок розробки нашої системи важко уявити без ретельного розгляду всіх популярних вразливостей, їх особливостей та рівня критичності. Даний огляд дозволить нам глибше зрозуміти, які компоненти необхідно включити до функціоналу нашої системи, щоб забезпечити максимальне покриття та ефективність виявлення загроз. Вивчення поширених атак, таких як SQL Injection, XSS, CSRF, RCE та інших, допоможе визначити оптимальні методи сканування та перевірки, що гарантують виявлення вразливостей на ранніх етапах.

Крім того, розуміння структури та механізмів цих атак дозволить не лише виявляти потенційні загрози, а й прогнозувати можливі варіанти їх розвитку, створюючи більш надійну архітектуру безпеки. Це також сприятиме побудові модульної структури нашої системи, що дозволить легко додавати нові методи сканування у відповідь на появу нових типів вразливостей або еволюцію вже існуючих.

Таким чином, на основі цього огляду ми зможемо не тільки визначити пріоритети у розробці, а й закласти фундамент для подальшого масштабування системи. Наша система буде здатна адаптуватися до змін у сфері кібербезпеки, залишаючись ефективним інструментом для виявлення та попередження загроз в умовах постійно зростаючих викликів інформаційної безпеки.

Розпочнемо з найпопулярнішого, яку ми ще у нашому вступі згадували, а саме “Cross-Site Scripting (XSS)”. Історія виникнення XSS бере свій початок наприкінці 1999 року, коли невелика група інженерів з безпеки Microsoft звернула увагу на новий тип атак. Центр реагування на безпеку Microsoft та команда, відповідальна за безпеку браузера Microsoft Internet Explorer, дізналися про випадки, коли зловмисники вставляли скрипти та зображення у HTML-сторінки деяких сайтів. Ці атаки мали різні форми: у деяких випадках зловмисники надсилали посилання жертвам, які нічого не підозрювали, і після

переходу за цими посиланнями їхні файли cookie перехоплювалися та передавалися третім сторонам. Інші варіанти атак мали більш стійкий характер, коли шкідливий код зберігався на сервері та активувався щоразу, коли сторінку відкривали інші користувачі.

Приблизно в той самий час Майкл Барретт, який обіймав посаду директора з інформаційної безпеки PayPal з 2006 по 2013 рік, згадував про виявлення експлойту Reflected XSS у програмному забезпеченні American Express. Цей вразливий елемент був продемонстрований на всіх рівнях управління компанією American Express і навіть обговорювався з командою Центру безпеки Microsoft під час конференції восени 1999 року.

До грудня того ж року інженери з безпеки Microsoft розпочали детальне вивчення цієї уразливості, а вже в лютому 2000 року було опубліковано офіційний звіт у співпраці з CERT, де вперше було введено термін "Cross Site Scripting" (XSS) для позначення цього типу атак. Таким чином, було започатковано дослідження однієї з найбільш поширених вразливостей у сфері веб-безпеки.

XSS надає змогу зловмисникам у їх корисливих цілях використовувати та маніпулювати взаємодією між користувачами та скомпроментовано програмою рисунок 1.1. Також вразливість дозволяє хакерам обійти політику однакового походження. Політика однакового походження (англ. The same-origin policy) – критично важлива функція безпеки, реалізована у веб-браузерах, щоб обмежити взаємодію документів або сценаріїв з одного походження(поєднання протоколу, домену та порту) з іншими ресурсами. Головна мета запобігти доступу зловмисних веб-сайтів до конфіденційних даних з інших веб-сайтів, які користувач може відвідати. XSS надає змогу зловмисникам видавати себе за законного користувача, оскільки зловмисні дії виконуються на персональному комп'ютері жертви, окрім цього отримується доступ до конфіденційних даних користувача. Якщо користувач має привілейований доступ до програми, то хакери потенційно можуть отримати повний контроль над функціями та даними програми.

Перейдімо від важкозрозумілих термінів до більш зрозумілого практичного пояснення. Як може виникнути дана проблема та як нею користуються зловмисники? Ми є розробником, на якого покладена роль зробити не тільки back-end у якому ми як риба у воді але й front-end(перенесення картинки яку надав дизайнер у вигляд веб-сторінки) частину в якій ми погано знаємось. Тому для пришвидшення процесу розробки ми вирішуємо використовувати вже готове рішення, при розробці ми знаходимо в когось шматок коду який робить гарну картинку на нашій сторінці і навіть не помічаємо що версія бібліотеки вже застаріла [1,2,3,4].

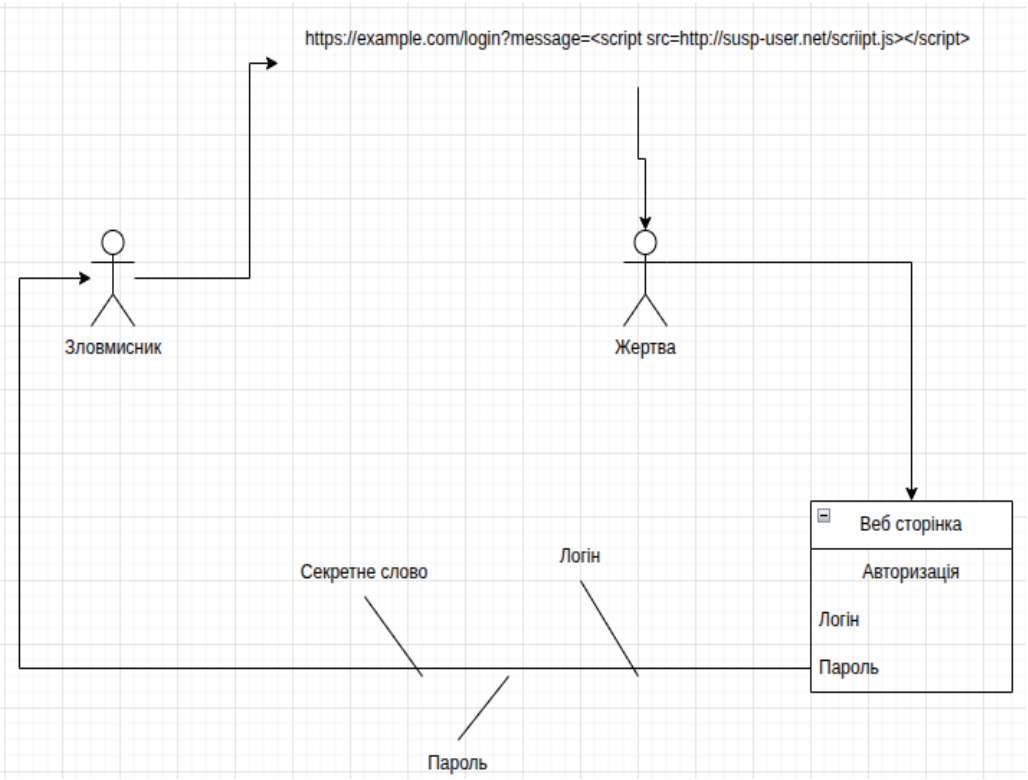


Рисунок 1.1 – Схема роботи XSS

SQL, або Structured Query Language (структурована мова запитів), була створена в 1970 році для роботи з базами даних. Вона слугує засобом зберігання, обробки та взаємодії з даними в цих базах. Основу SQL становлять два ключові елементи, які є фундаментальними для розуміння її роботи.

Перший елемент – це таблиці, у яких організовано зберігання даних. Вони мають структуру з рядками та стовпцями, де кожна комірка містить певне значення. Для більш ефективного управління даними зазвичай створюється декілька таблиць, що дозволяє оптимізувати процеси зберігання та доступу до інформації.

Другий важливий елемент – це запити і оператори. Саме за їх допомогою програми взаємодіють із базами даних. Коли необхідно отримати інформацію, оновити її або видалити, формується SQL-запит, який виконує відповідні дії з даними. Таким чином, SQL забезпечує зручний спосіб маніпуляції даними та підтримання їхньої цілісності в базах даних.

SQL-ін'єкцію вперше виявив у 1998 році Джефф Форрістал. Під час дослідження він помітив, що введення спеціально сформованих SQL-запитів дозволяє отримати доступ до прихованої інформації, збереженої на сервері. Спочатку розробники недооцінювали серйозність цієї вразливості, проте з часом SQLi стала однією з найпоширеніших атак, регулярно з'являючись у списках топ-10 найнебезпечніших вразливостей за версією OWASP.

Протягом наступних років SQL-ін'єкції неодноразово призводили до масштабних зломів даних. Одним із найбільших випадків став інцидент з Heartland Payment Systems у 2008 році – платіжним оператором, який обробляв мільйони транзакцій. В результаті атаки було викрадено інформацію про понад 130 мільйонів кредитних та дебетових карток.

Ще одним гучним випадком стала атака на Sony у 2011 році, коли зловмисники отримали доступ до особистих даних близько 77 мільйонів користувачів PlayStation. Збитки від цього інциденту для Sony Pictures оцінювалися у величезні суми, підкреслюючи критичну важливість захисту від SQL-ін'єкцій. Ці події стали знаковими у сфері інформаційної безпеки, підкреслюючи необхідність впровадження надійних методів захисту даних від несанкціонованого доступу через вразливості в SQL-запитах.

Отже SQL injection (SQLi) дозволяє зловмисникам маніпулювати запитами до бази даних, які виконує програма рисунок 1.2. Це може надати зловмисникам

доступу до даних, які вони зазвичай не зможуть отримати, наприклад інформацію що належить іншим користувачам, або конфіденційних даних, доступних програмі. У багатьох випадках зловмисники можуть змінити або видалити дані, що призведе до постійних змін у функціональності або вмісті програми.

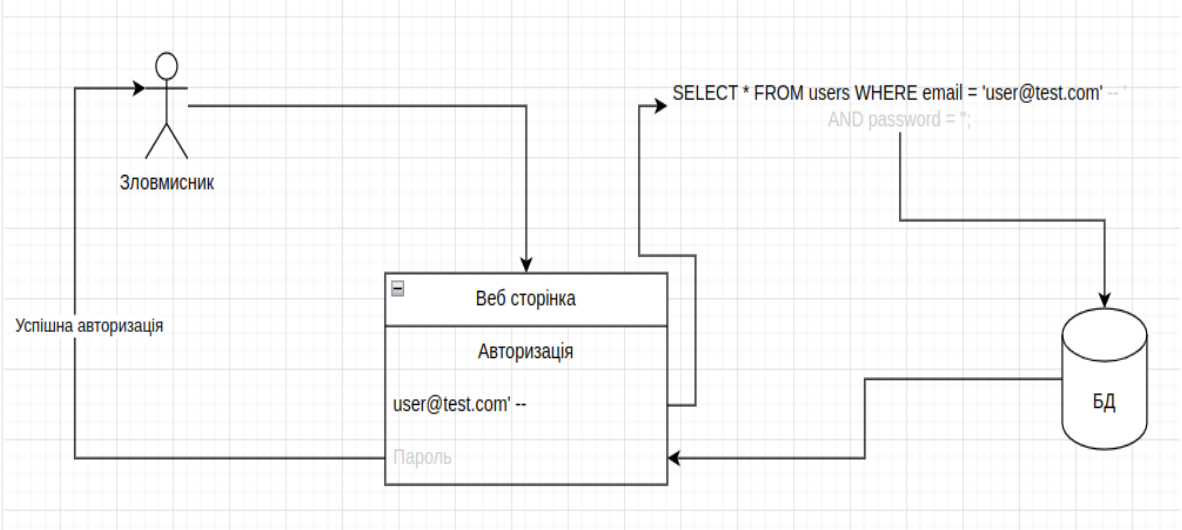


Рисунок 1.2 – Схема роботи SQL injection

В разі успішної атаки SQL-ін’єкції може призвести до неавторизованого доступу до конфіденційних даних: паролі, реквізити кредитних карт, особиста інформація користувача. Атаки SQL-ін’єкцій зіграли значну роль у численних резонансних витоках даних протягом історії, часто призводячи до серйозної шкоди репутації та суттєвих нормативних санкцій для постраждалих організацій. У деяких випадках зловмисники використовували ці вразливості для встановлення постійних “бекдорів” у системах організації.

Бекдор – це прихована або несанкціонована точка входу в комп’ютерну систему, програму або мережу, яка дозволяє зловмиснику або авторизованому користувачеві обійти звичайну автентифікацію та заходи безпеки. Бекдори можуть створюватися розробниками навмисно для законних цілей, наприклад для віддаленого усунення несправностей або обслуговування, але вони частіше пов’язані зі зловмисними намірами.

Такі бекдори можуть уможливити тривалий несанкціонований доступ, дозволяючи зловмисникам залишатися непоміченими протягом тривалого періоду часу, викрадаючи конфіденційні дані або компрометуючи додаткові ресурси. Ці довгострокові порушення можуть мати руйнівні наслідки, зокрема фінансові втрати, втрату довіри клієнтів і накладення юридичної відповідальності.

Всього існує чотири найпоширеніші сценарії атаки SQL-ін'єкцій. “Отримання прихованих даних”, зміна SQL запитів для виявлення додаткової інформації, яка зазвичай недоступна. “Порушення логіки програми”, коли можна змінити запит, щоб втручатись в логіку програми. “Асоціаційна атака”, де можна отримати дані з різних таблиць бази даних. “Сліпа SQL ін'єкція”, коли результат контрольованого запиту не повертається у відповідях програми.

Тепер розглянемо як кожен з цих сценаріїв працює. Розпочнемо з “Отримання прихованих даних” приклад запиту на рисунку 1.3, даний метод дає змогу дослідити структуру таблиць які цікавлять зловмисник, в запиту нижче отримуються відомості про авто на основі бренду, наданої користувачем.

```
testdb=# SELECT * FROM cars WHERE brand = 'Nissan';
```

Рисунок 1.3 – Запит на продукти по категорії

Якщо програма не перевіряє належним чином зміст запитів який зображений на рисунку 1.4, зловмисник може маніпулювати введенням марки автомобіля, що відкриває додаткові дані.

```
testdb=# Nissan' OR '1'='1';
```

Рисунок 1.4 – Запит на отримання додаткових даних

В результаті зловмиснику для отримання потрібних йому даних може додати в початковий запит умову “або” рисунок 1.5, яка видасть дані оскільки дана умова видасть true

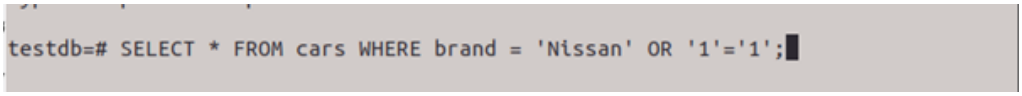


Рисунок 1.5 – Запит з завжди правдивою умовою

“Порушення логіки програми”, даний метод по вигляду запиту може здаватись схожим для новачків які до цього не працювали з SQL запитами. Він надає змогу зловмиснику обходити існуючі перевірки чи обмеження. На рисунку 1.6 відображено запит для перевірки облікових даних при авторизації.

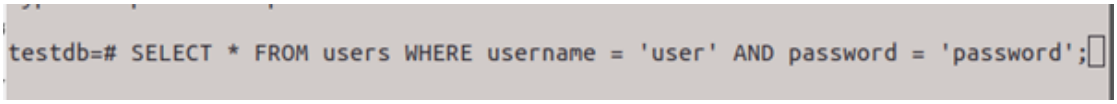


Рисунок 1.6 – Запит для перевірки облікових даних

Якщо програма не перевіряє належним чином, то рисунку 1.7 зловмисник може додати ігнорування параметра пароль, з допомогою оператора “--”.

Рисунок 1.7 – Запит з ігноруванням подальших параметрів

“Асоціаційна атака”, цей метод базується на використанні SQL оператора “UNION”, щоб об'єднати результати двох або більше запитів SELECT в один відповідний. Тобто зловмисник може за допомогою об'єднання безпечного запиту який видасть продукти за певною категорією, з отриманням всіх користувачів з їх ідентифікаторами та автентифікаторами. Приклад запиту показано на рисунку 1.8, в результаті у відповідь прийде вміст таблиці users.

```
testdb=# SELECT name, price FROM products WHERE category = 'electronics' UNION S
ELECT username, password FROM users--';
```

Рисунок 1.8 – Запит з оператором “UNION”

Окрім цього на рисунку 1.9 зображено метод як можна визначати кількість стовпців в даній таблиці за допомогою додавання в запит наступного вмісту, збільшуючи індекс стовпця.

```
testdb=# ' ORDER BY 1--
```

Рисунок 1.9 – Запит на отримання кількості стовпців

Також після того як відомо кількість стовпців можна визначити типи даних рисунку 1.10 які зберігаються в кожному з них.

```
testdb=# ' UNION SELECT 'a',3,NULL--
```

Рисунок 1.10 – Запит на визначення типів даних стовпців

Багато вразливостей SQL-ін’кій належить до категорії “Сліпа SQL-ін’єкція”. У таких випадках програма безпосередньо не показує результати виконаного SQL-запиту або не надає детальних повідомлень про помилки бази даних у своїх відповідях. Однак ці вразливості все ще можуть використовувати зловмисники для отримання несанкціонованих даних або маніпулювання базою даних, хоча й за допомогою більш складних і непрямих методів. Сліпа SQL-ін’єкція часто ґрунтується на спостереженні за незначними змінами в поведінці програми, такими як варіації часу відповіді, відмінності у вмісті відповідей або певні індикатори успіху та помилки. Ці методи ускладнюють використання, але не є неможливим для рішучого зловмисника.

Перейдемо до застосування сліпої SQL-ін’єкції шляхом запску умовних відповідей. Але перед цим, файли cookie – це не громіздкі фрагменти даних, які веб-браузер зберігає локально на пристрої користувача під час перегляду ним різних сайтів. Ці дані дають змогу покращувати якість та ефективність вашого перебування в мережі Інтернет, з допомогою них вам надаються саме ту інформацію яка вас найбільше цікавить, заради якої ви в принципі використовуєте його. Часто в файлах cookie зберігаються не лише аналітичні дані, але й сеанси на різних сайтах. Сеанс – це тимчасова взаємодія між користувачем та веб-застосунком, яка триває допоки користувач активно використовує застосунок. Це надає змогу серверу запам’ятовувати інформацію про користувача, таку як статус входу, уподобання або активності, за кількома запитами. Для зручності керування даними сеансами кожен з них має унікальний ідентифікатор, який зазвичай зберігається як файли cookie або передаються через URL-адреси. Після завершення сеансу через те, що користувач виходить із системи, закриває браузер або час очікування сеансу закінчується, збереженні дані скидаються, це впровадження задля забезпечення безпеки та продуктивності роботи системи. Розглянемо варіант в якому використовуються файли cookie для відстеження та збору даних користувача. До прикладу рисунок 1.11 запит до сервера може містити в заголовці cookie “Cookie: TrackingId=kD3kl4Fsd3Fd341D3”. Під час обробки запиту сервер перевіряє чи є це відомим користувачем для системи.

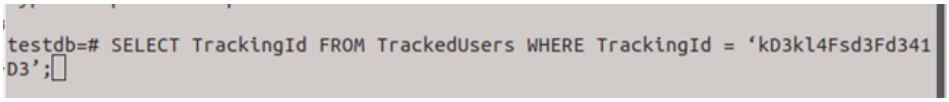


Рисунок 1.11 – Запит на перевірку користувача

За певних обставин атаки SQL-ін’єкцій можуть посилюватися, щоб скомпрометувати базовий сервер або інші серверні системи. Крім цього, зловмисники можуть використовувати ін’єкцією SQL для записку атак типу “відмова в обслуговуванні” проти програми або інфраструктури [5,6].

“Path traversal” також відомий як “Directory Traversal”, вперше даний термін був визнаний в середині 1990-х років, коли мережа Інтернет розширилась та динамічні веб-застосунки стали більш поширеними. Розробники все більше покладались на дані користувачів для створення шляхів до файлів, ненавмисно наражаючи програми на вразливості, коли перевірка введених даних ігнорувалася. Перші обговорення по інцидентах з даною вразливістю відразу з'явилися в першій документації по веб-безпеці, та в інших джерелах які були присвячені безпечній веб-розробці.

Один з найперших задокументованих випадків уразливості “Path traversal” стався в 1996 році, в епоху веб-застосунків на основі CGI. CGI(Common Gateway Interface) – це стандартний протокол, який дозволяє веб-серверам взаємодіяти із зовнішніми програмами для створення динамічного веб-вмісту. Багато веб-серверів того часу використовували сценарії, написані на Perl або подібних мовах, які часто динамічно завантажувані файли на основі введення користувача.

“Path traversal” атака виникає коли зловмисник маніпулює шляхами до файлів у програмі, щоб отримати доступ до файлів або каталогів за межами передбачуваної області. Ця вразливість виникає внаслідок недостатньо правильної перевірки або дезінфекції переданих користувачами даних, що дозволяє зловмисникам пройти через структуру каталогів сервера та потенційно отримати несанкціонований доступ до конфіденційних даних клієнтів.

Ключовим прикладом використання зловмисником було додавання послідовностей обходу (../) для маніпулювання шляхом файлів і доступу до конфіденційних системних файлів до прикладу таких як /etc/passwd. В разі якщо зловмисник отримує доступ до даної директорії то він зможе додавати нового користувача, змінювати паролі існуючих. Тому ця вразливість несе дуже вилку небезпеку.

Найпоширенішою проблемою розробників що писали динамічні веб-сайти на ранніх версіях PHP було використання багатьох функцій, які не перевіряли або не фільтрували вхідні дані, до прикладу: inlude, require та fopen. Вони писали код

без достатнього розуміння безпеки, приклад коду: “$file = $_GET[‘file’]; include(“files/” . $file);” [7,8].

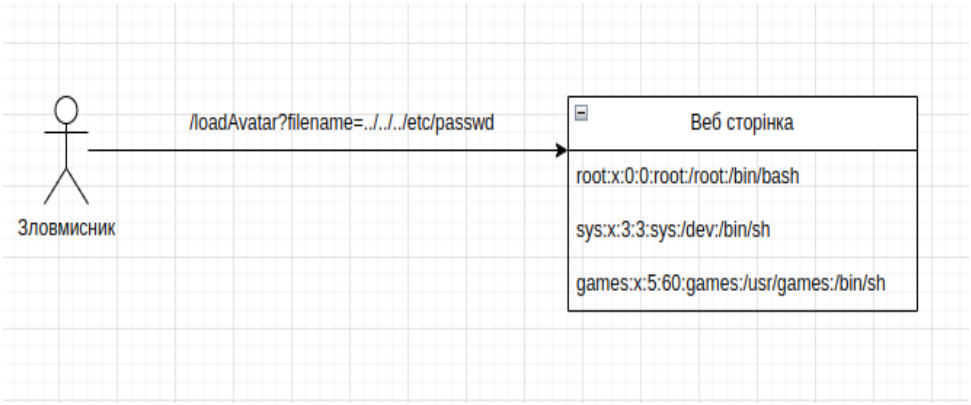


Рисунок 1.12 – Схема роботи Path traversal

“OS Command injection” також відома як “Shell injection”, було вперше визначено як вразливість безпеки наприкінці 1990-х років, коли вебсистеми та мережеві системи почали інтегрувати введення користувача в команди на рівні операційної системи. Дана проблема привернула увагу зловмисників, оскільки вони могли скористатися цією слабкістю та скористатись неправильною перевіркою введення довільних команд на сервері чи хост-системі.

Окрім цього дана вразливість може призвести до повної компрементації програми та її даних. Також зловмисники часто використовують її, щоб проникнути в інші частини інфраструктури хостингу. Використовуючи довірчі відносини, вони можуть посилити атаку та націлитися на інші системи в організації.

Суть “OS Command injection” є використання неправильно налаштованої обробки вхідних даних, а саме ручних даних які надає користувач, в особливостях код а не простий текст. Вставляючи спеціальні символи або розділювачі команд, найпоширеніші з них це крапка з комою “;”, подвійний амперсанд “&&” або вертикальна лінія “|”, що призводить до зупинки виконання запланованої конди та додавання власної шкідливої команди. Наприклад програма розроблена для перевірки IP-адреси вказаної користувачем, може

включати вхідні дані безпосередньо в команду оболонки, наприклад “ping <текст користувача>”. Якщо введення не буде перевірено, зловмисник зможе зробити щось на зразок “127.0.01; cat /etc/passwd”. Що буде схоже на sql ін’єкцію, коли додається додаткова умова на виконання. Що призведе до виконання команди ping та cat яка висвітлить конфіденційну інформацію користувача.

Дана вразливість має ще додаткову небезпеку, оскільки вона може служити опорою для букового руху в інфраструктурі організації. Компроментуючи хост-сервер, зловмисник може використовувати довірчі відносини між системами, підвищувати свої привілеї та поширювати атаки на інші мережеві пристрої чи програми. Це робить “ін’єкцію команд ОС” високопріориттеною загрозою в середовищах, де на карту поставлені конфіденційні дані або важливі операції.

Для того щоб захистити нашу систему від даної загрози, потрібно ретельно обробляти вхідні дані, гарантуючи, що всі введені користувачем дані продезінфіковано, перевірено або параметризовано перед передачею системним командам. Це також передбачає мінімізацію залежності від команд на рівні ОС, впровадження суворого контролю доступу та дотримання принципу найменших привілеїв, щоб зменшити потенційну шкоду в разі використання вразливості. Неможливо переоцінити важливість усунення цієї вразливості, оскільки наслідки можуть поширюватися далеко за межі початкової цільової програми, впливаючи на цілі системи та мережі [9,10].

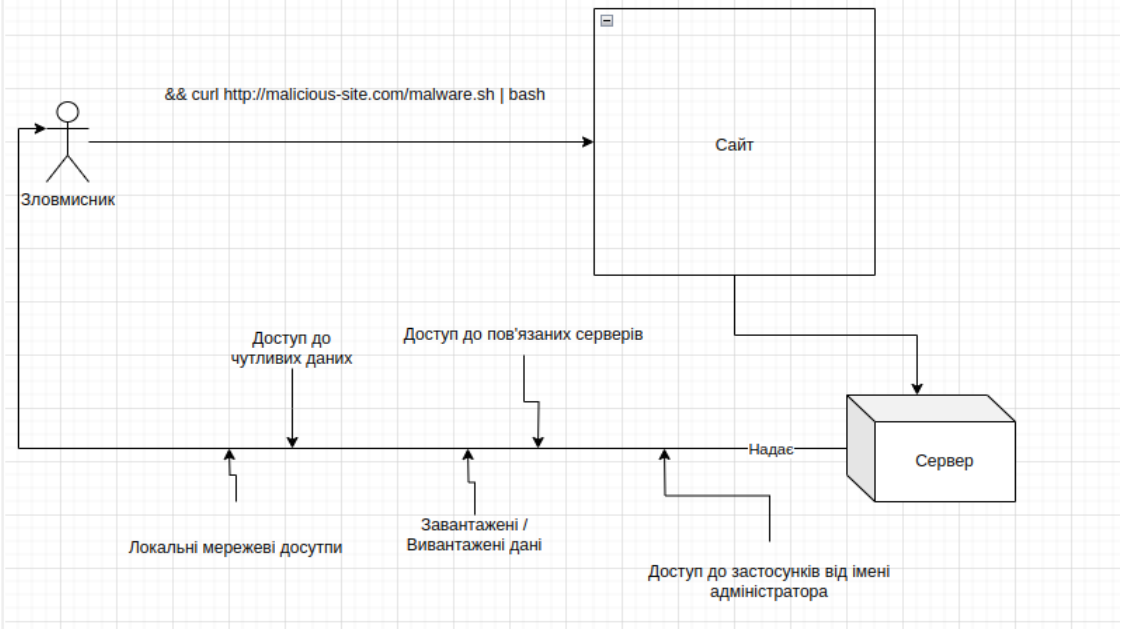


Рисунок 1.13 – Схема роботи “OS Command injection”

“Sensitive information exposure”, або "Витік чутливої інформації", була визначена як одна з критично важливих вразливостей у сфері кібербезпеки ще на початку 2000-х років. Це поняття охоплює випадки, коли конфіденційна інформація стає доступною для сторонніх осіб або зловмисників через неналежний рівень захисту в інформаційних системах. З розвитком Інтернету та зростанням обсягу даних, що передаються та зберігаються в мережі, ця проблема набула глобальних масштабів.

Збільшення популярності онлайн-сервісів, електронної комерції та соціальних мереж призвело до того, що величезні масиви особистої та фінансової інформації стали зберігатися у цифрових форматах. Це включає в себе номери кредитних карток, паролі, персональні дані, медичну інформацію та комерційні секрети. На той час, коли обсяги даних почали стрімко зростати, питання їхнього захисту часто відходило на другий план через фокусування на зручності використання та швидкості доступу. Багато компаній зосереджувалися на розширенні своїх цифрових платформ, не приділяючи достатньої уваги шифруванню даних, безпечному зберіганню інформації та належним заходам контролю доступу.

Однією з найбільших проблем того часу стало недостатнє шифрування даних під час їх передачі. Багато онлайн-ресурсів використовували незахищений протокол HTTP, що дозволяло зловмисникам перехоплювати дані в момент їхнього пересилання між користувачем та сервером. Наприклад, особисті дані, введені у форму замовлення на вебсайті, могли бути вкрадені через атаку "man-in-the-middle", якщо з'єднання не було захищене HTTPS. Крім того, погане налаштування серверів та відсутність обмежень доступу сприяли тому, що конфіденційна інформація ставала доступною для перегляду або навіть редагування сторонніми особами.

Варто згадати і про помилки у програмному забезпеченні, які призводили до витоків інформації. У деяких випадках дані зберігалися у відкритому вигляді у журналах систем, кешах браузерів або залишалися доступними після завершення сесії користувача. Це створювало сприятливі умови для витоку конфіденційної інформації при фізичному доступі до пристрою або під час зламу системи.

Перші масштабні інциденти витоку даних почали привертати увагу громадськості та спеціалістів з безпеки. Одним із знакових прикладів став витік даних компанії TJX у 2005 році, коли були скомпрометовані дані понад 94 мільйонів кредитних і дебетових карток через вразливості в системі захисту. Це стало потужним сигналом для всієї індустрії щодо необхідності посилення кібербезпеки та розробки нових підходів до захисту інформації [11].

1.2 Аналіз вже існуючих рішень

Зі стрімким розвитком веб-технологій і все більшою залежністю від веб-додатків для бізнес-операцій забезпечення їх безпеки стало ключовим аспектом сучасного розвитку. Вразливості веб-додатків можуть призвести до витоку даних, фінансових втрат і несанкціонованого доступу до критично важливих систем. Для пом’якшення цих ризиків використовуються спеціалізовані сканери безпеки

веб-додатків, які визначають слабкі місця та пропонують стратегії виправлення. Ці інструменти допомагають розробникам і фахівцям із безпеки виявляти недоліки безпеки, перш ніж ними зможуть скористатися зловмисники. Тому перед тим як розпочати планування та реалізацію нашого проєкту варто розглянути існуючі реалізації і взяти з них найкращі сторони. Системи аудиту та аналізу безпеки вебзастосунків як і будь яке інше програмне забезпечення поділяється на безкоштовне або як ще його називають з відкритим кодом та платне. Інструменти з відкритим кодом широко використовуються в спільноті кібербезпеки завдяки своїй гнучкості та економічній ефективності. Деякі з найпопулярніших сканерів веб-безпеки з відкритим кодом включають: OWASP ZAP (Zed Attack Proxy) – є одним із найпоширеніших сканерів безпеки веб-додатків із відкритим вихідним кодом, розроблений і підтримуваний Open Web Application Security Project (OWASP). Він розроблений, щоб допомогти фахівцям із безпеки, розробникам і тестувальникам проникнення виявляти вразливості у веб-додатках.

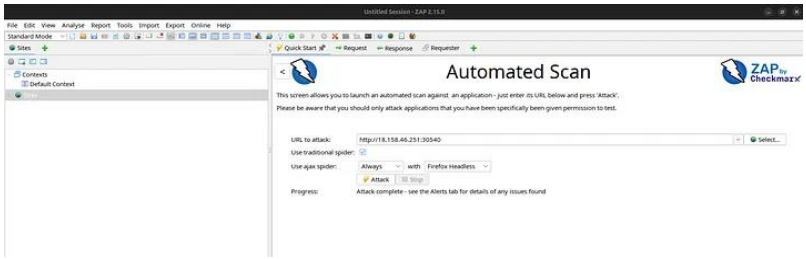


Рисунок 1.15 – Інтерфейс аналізатора OWASP ZAP [41]

Окрім того що даний інструмент являється безкоштовним, він також має багато переваг, до прикладу графічний інтерфейс який спрощує використання для новачків, їм не потрібно шукати команди для того щоб отримати максимум від ефективності інструмента. Тож користувач може спокійно навести курсором на те що йому потрібно і спокійно працювати. До того ж він являється довлі зручним як для початкових розробників так і для спеціалістів зі стажем, оскільки даний інструмент можна інтегрувати у GitHub Actions, GitLab CI/CD та інші робочі процеси DevSecOps для автоматичного сканування програм перед

розгортанням. Та не варто забувати про важливість існування спільноти професіоналів з безпеки, які регулярно вносять оновлення, покращення та нові методи сканування.

Як кажуть монета має дві сторони тож даний інструмент не є винятком, оскільки він має також недоліки. Одним з найбільших являється обмежена точність порівняно з платними “комерційними” аналогами. Він може видавати хибні спрацювання або пропускати деякі складні вразливості. Незважаючи на те що OWASP має багато зручностей для початківців, для повного використання його розширених можливостей потрібен досвід. Користувачі повинні вивчити методи ручного тестування на проникнення, сценарії та налаштування конфігурації, щоб максимізувати його ефективність. До того ж в результаті сканування не можна сформувати звіт у форматі PDF. А також зберігання результатів сканування не у вбудованій безпечній базі даних.

Тепер, коли ми розглянули один з найкращих платних варіантів аналізатора, а саме OWASP ZAP, ми можемо перейти до розгляду платних рішень і того, що їх відрізняє. Платні аналізатори безпеки веб-додатків пропонують безліч розширених функцій і можливостей, які виходять за рамки основ безкоштовних інструментів. Ці рішення професійного рівня зазвичай включають розширене виявлення вразливостей, глибше автоматизоване сканування та надійні функції ручного тестування, адаптовані для складних корпоративних середовищ. Вони також забезпечують повну звітність, повну інтеграцію з конвеєрами CI/CD і спеціальну підтримку клієнтів, щоб допомогти швидко вирішувати проблеми. З регулярними оновленнями для протидії загрозам, що розвиваються, і вищим ступенем налаштування платні інструменти створені для надання більш точних, ефективних і масштабованих оцінок безпеки, що робить їх важливою інвестицією для організацій, яким потрібен суворий захист своїх веб-активів [12,13].

Одним з найкращих інструментів серед платних являється “Burp Suite Professional”. Burp Suite Professional – це інтегрована платформа преміум-класу, розроблена PortSwigger, яка служить комплексним рішенням для тестування

безпеки веб-додатків. Він поєднує в собі автоматизоване сканування та багатий набір інструментів ручного тестування в єдиний інтерфейс, що дає змогу фахівцям із безпеки ефективно виявляти такі вразливості, як впровадження SQL, міжсайтовий сценарій (XSS) та інші поширені загрози. Інструмент працює як проксі-сервер-перехоплювач, що дозволяє тестувальникам захоплювати та маніпулювати трафіком HTTP/S між браузером і цільовою програмою, пропонуючи точний контроль над даними, якими обмінюються під час оцінювання. Ця можливість додатково розширена завдяки функціям, які підтримують автоматичні атаки, ітеративну модифікацію запитів і аналіз випадковості маркерів сеансу, що робить інструмент універсальним як для швидкого сканування вразливостей, так і для поглибленого ручного дослідження.

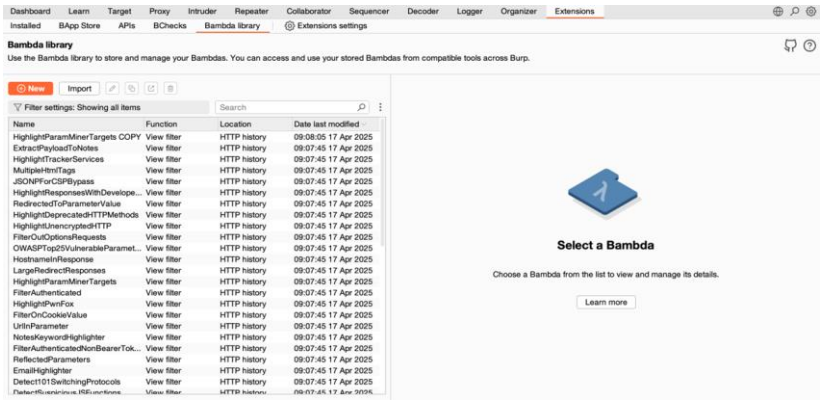


Рисунок 1.16 – Інтерфейс аналізатора Burp Suite Professional [42]

Однією з ключових сильних сторін Burp Suite Professional є його повний набір інструментів і можливість розширення. Його графічний інтерфейс розроблений інтуїтивно зрозумілим, проводячи як новачків, так і досвідчених користувачів через різні процеси тестування, від початкового відображення структури веб-додатку до детального вивчення виявлених уразливостей. Платформа також підтримує спеціальні розширення та добре інтегрується з іншими інструментами безпеки та конвеєрами розробки, дозволяючи адаптувати робочі процеси тестування, які відповідають конкретним потребам різних середовищ. Крім того, його надійні можливості звітування надають практичну

інформацію та допомагають спростити процес виправлення, що є вирішальним у контексті професійного тестування на проникнення та аудиту безпеки.

Однак слід враховувати деякі недоліки. Вартість Burp Suite Professional може бути суттєвим фактором для невеликих організацій або окремих дослідників, оскільки необхідні інвестиції можуть бути непомірно високими порівняно з безкоштовними альтернативами. Крім того, незважаючи на те, що інтерфейс є зручним для користувача, широкий спектр функцій і параметрів конфігурації означає, що для новачків може знадобитися крутий процес навчання, яким може знадобитися значний час, щоб повністю освоїти його потенціал. Інструмент також може бути ресурсомістким, особливо під час розширеного сканування складних програм, які можуть вимагати високопродуктивного апаратного забезпечення для підтримки оптимальної продуктивності. Незважаючи на ці проблеми, його ефективність у виявленні вразливостей і повний набір функцій роблять його незамінним інструментом для багатьох спеціалістів із безпеки [14,15].

1.3 Планування реалізації програмного продукту

На сьогоднішній день є багато різних варіантів реалізації від створення простого застосунку для різних операційних систем до впровадження власної бібліотеки для на певній мові програмування, з подальшим публікуванням її на офіційних сайтах де публікуються довірені бібліотеки, яка дуже детально перевіряються на зловмисний код.

Розробка власної бібліотеки та подальша публікація її в офіційному репозиторії, такому як до прикладу “RubyGems.org” [16], це спроба, яка включає в себе безліч складних процесів, як технічних, так і концептуальних. По суті, складність полягає в розробці API, який є не тільки інтуїтивно зрозумілим і узгодженим, але й досить гнучким, щоб враховувати майбутні вдосконалення без порушення існуючої функціональності. Це вимагає ретельного планування та

глибокого розуміння проблемної області, а також потенційних крайніх випадків, які можуть виникнути під час її використання. Внутрішня архітектура має бути модульною, гарантуючи, що кожен компонент відокремлений і може підтримуватися або замінюватися незалежно, що, у свою чергу, додає рівні абстракції, якими потрібно ретельно керувати.

Тому нам варто задуматись краще над розробкою простого веб-застосунку оскільки він має кілька переконливих переваг. Основною з них являється час який ми витратимо на публікацію даної бібліотеки, оскільки якщо її не опублікувати від неї не буде ніякої користі. В свою чергу веб-застосунок буде забезпечувати зручний графічний інтерфейс який буде більше притягувати користувачів до нього аніж встановлення невідомої для них бібліотеки, яка через неграмотне створення може погіршити а ще в гіршому варіанті зовсім зламати проект. Також графічний інтерфейс більше підходить для аудиторії яка тільки починає ознайомлюватись з усіма нюансами у веб-розробці, на котрих і буде орієнтуватись наш застосунок. Централізувавши функціональні можливості на веб-платформі, користувачі можуть легко ініціювати сканування, переглядати вичерпні звіти та керувати конфігураціями за допомогою інтуїтивно зрозумілих інформаційних панелей, що значно покращує загальну взаємодію з користувачем.

Наш веб-застосунок буде створений на базі фреймворка Ruby on Rails, який базується на мові програмування Ruby. Мова Ruby – це високорівнева мова програмування, походить з Японії яка була тепло прийнята технологічним ком’юніті, що в результаті трансформувало індустрію технологій. Її автором являється Юкіхіро Мацумото, більш відомий як Матц, у 1995 році, Ruby була розроблена як об’єктно орієнтована мова, яка черпала натхнення з Lisp, Perl та Ada. Матц зусередився на виразному та інтуєтивно зрозумілому синтаксесі, який швидко б міг зрозуміти новачок. Подібно до Perl та Python, Ruby є інтерпративною мовою, яка також має спільні об’єктно-орієнтовані принципи з Java та Ada. Ця комбінація дозволяє Ruby знайти ідеальний баланс між продуктивністю та простотою використання.

Ruby on Rails або як коротше її називають RoR – це платформа для веб-розробки, як і раніше загнулось написане на мові програмування Ruby. Він був розроблений Девідом Хайнемайєром Ганссоном у 2003 році під час роботи над кодовою базою для Basecamp, інструменту управління проектами від 37signals. Офіційний реліз фреймворка RoR відбувся у липні 2004 року.

Ruby on Rails став еталоном багатьох інших популярних оскільки швидко став одним з найбільш затребуваних технологій. Даний фреймворк люблять розробники, так і підприємці через виняткову продуктивність у створенні веб-застосунків. Він часто використовується саме для стартапів і малих підприємств, оскільки він покращує саме головне, а саме час розробки за який і платять замовники компаніям. Також варто не забувати що Ruby on Rails має активне ком'юніті яке завжди може підказати в разі якихось проблем під час розробки, а також покращує продуктивність як розробки так і роботи самого веб-застосунку з допомогою створення все нових бібліотек або як їх називають геми [17,18].

2 СТВОРЕННЯ СИСТЕМИ

2.1 Проектування UML-діаграм

Створення будь-якої системи яка базується на створенні програмного забезпечення є критично важливим етапом у процесі розробки, одним з головних інструментів для досягнення найкращих результатів являється використання UML діаграм. UML (Unified Modeling Language) є одним із найпоширеніших засобів для візуалізації, проектування та документування програмних систем. Його основна мета полягає у створенні стандартних способів опису структури та поведінки програмного забезпечення, що значно полегшує процес розробки та комунікацію між учасниками команди. Незалежно від масштабу проекту, UML-діаграми дозволяють розробникам більш чітко зрозуміти вимоги до системи, виявити потенційні проблеми на етапі проектування та забезпечити узгодженість між фронт-енд, бек-енд та дизайном додатку.

Застосування UML діаграм із самого початку проекту допомагає забезпечити чітке розуміння архітектури додатку як для розробників, так і для стейкхолдерів. Діаграма прецедентів використання (Use Case Diagram) дозволяє визначити основні сценарії взаємодії користувачів із системою, ще до написання першого рядка коду. На рисунку 2.1 відображено основні позначення даної діаграми. Це дає змогу заздалегідь виявити всі необхідні функціональні вимоги, визначити ролі користувачів та їх можливі дії. Наприклад, для веб-застосунку з функціоналом управління замовленнями можна чітко виділити, які ролі будуть існувати (адміністратор, менеджер, клієнт) та які дії вони можуть виконувати (створення замовлення, перегляд історії, управління статусом).

Для кращого розуміння всіх складових розглянемо кожну з них. Актор – сутність, яка визначає когось або щось, що взаємодіє з системою, але не є її компонентом, тобто перебуває поза її межами. Зазвичай позначення класичне як на рисунку 2.1 у вигляді чоловічка, але у рідких випадках його зображують у вигляді прямокутника з написом “actor”. Умовно акторів можна поділити на первинних та вторинних: первинні ініціюють взаємодію з системою, а вторинні

вже реагують на дану взаємодію рисунок 2.2. За замовчуванням первинних розміщують графічно з ліва ді системи а вторинних справа. Взаємодія з системою не обмежується тільки людьми тому актор може представляти іншу систему або пристрій. Поширеною проблемою являється плутанина між поняттям “актор” та “користувач”. Актор – узагальнення представлення класу користувачів, яке може включати в себе декілька ролей, в свою чергу користувач – конкретна реалізація цього актора.

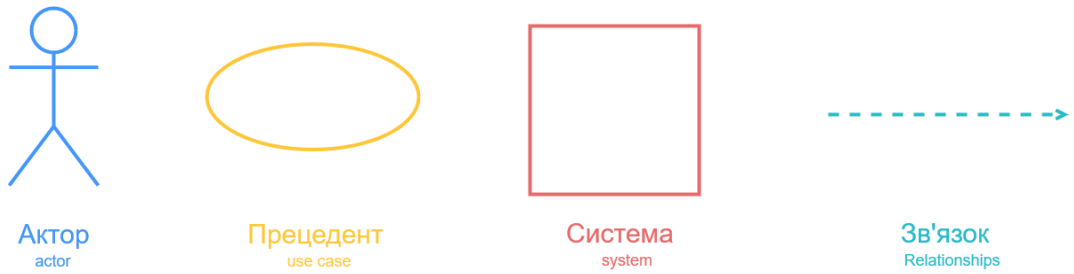


Рисунок 2.1 – Основні позначення діаграми використання [19]

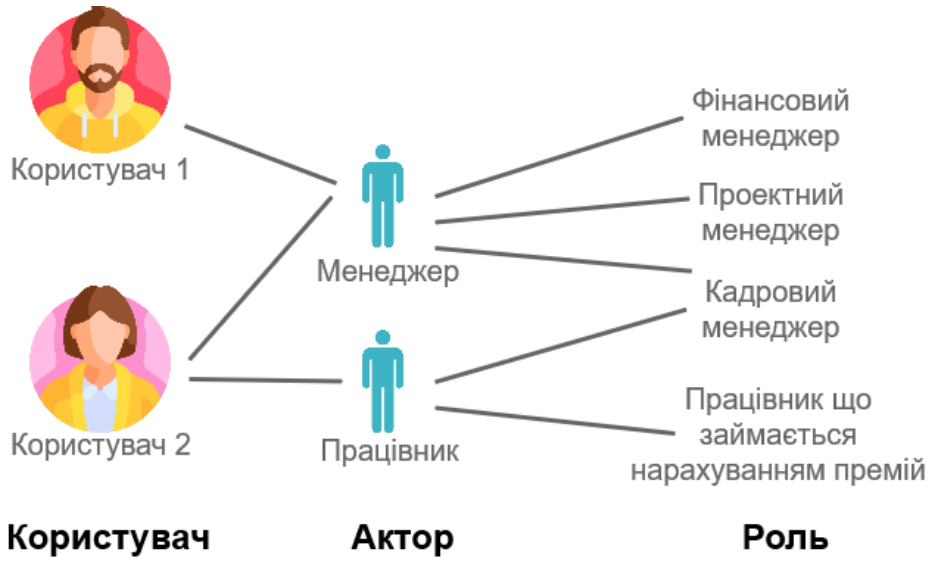


Рисунок 2.2 – Приклад користувача, актора та ролі в UML [19]

Випадок використання або як його ще називають прецедент, визначає очікувану поведінку системи, розкриваючи, що саме вона має виконувати. Це своєрідний опис функціональності, який дає відповідь на запитання “Що робить система?” та деталізує набір можливих дій, функцій чи завдань, доступних в рамках використання. Прецеденти не фокусуються на технічних деталях

реалізації, натомість зосереджуються виключно на кінцевому результаті, який очікується від системи. Вони візуалізуються у вигляді еліпса з дієсловом, що визначає конкретну дію або завдання, яке користувач або інша система має виконати. Наприклад, у системі управління електронною бібліотекою такими прецедентами можуть бути "забронювати книгу", "продовжити термін користування", "переглянути історію замовлень". У додатку для керування проєктами це можуть бути "додати завдання", "призначити виконавця", "оновити статус завдання". У сервісі доставки їжі прецедентами можуть бути "оформити замовлення", "перевірити статус доставки", "скасувати замовлення". У всіх цих випадках прецедент чітко визначає, що має статися в системі, проте залишається відкритим питання, як саме ця дія буде реалізована на рівні коду чи інфраструктури. Це дозволяє концентруватися на функціональних вимогах, відокремлюючи їх від технічних аспектів, і забезпечує більш зрозуміле спілкування між замовником, розробниками та тестувальниками.

Система, в рамках якої моделюються ці прецеденти, може бути представлена різноманітними програмними рішеннями – від сайту та мобільного додатка до окремого модуля програмного забезпечення. Вона відображається як прямокутник з назвою у верхній частині, що символізує межі функціональності та визначає контекст для виконання завдань. Прецеденти всередині цього прямокутника показують, які дії система здатна реалізувати, і як користувачі або зовнішні системи можуть взаємодіяти з нею.

Взаємодія між акторами та прецедентами відбувається через зв’язки, що моделюються за допомогою суцільних ліній. Актор завжди пов'язаний хоча б з одним прецедентом, оскільки саме він ініціює дію або реагує на події системи. Однак не кожен прецедент обов’язково має зв'язок з актором; деякі дії можуть бути викликані внутрішніми процесами системи. Наприклад, автоматичне оновлення балансу або обробка періодичних платежів може виконуватися без прямої взаємодії з користувачем. Зв’язки між компонентами надають візуальну ясність щодо того, як саме здійснюється взаємодія між користувачами, іншими системами та окремими функціями. Окрім того, коментарі у діаграмах можуть

слугувати розширенням опису, додавати додаткові умови або підказки, які полегшують розуміння логіки взаємодії.

Таким чином, прецеденти разом із системою та зв’язками формують чітке уявлення про функціональність додатка, визначаючи межі його роботи, сценарії використання та ролі учасників процесу. Вони не тільки структурують вимоги, але й допомагають побачити загальну картину роботи програми ще на етапі проектування [19,20].

Наша діаграма варіантів використання рисунок 2.3, являється не складною оскільки система не буде містити прям великого функціоналу. На ній зображено одного актора, яким являється простий користувач та два прецедента: “Перегляд сторінки з інформацією про вразливість” та “Сканувати певний вебзастосунок на одну або кілька вразливостей”.

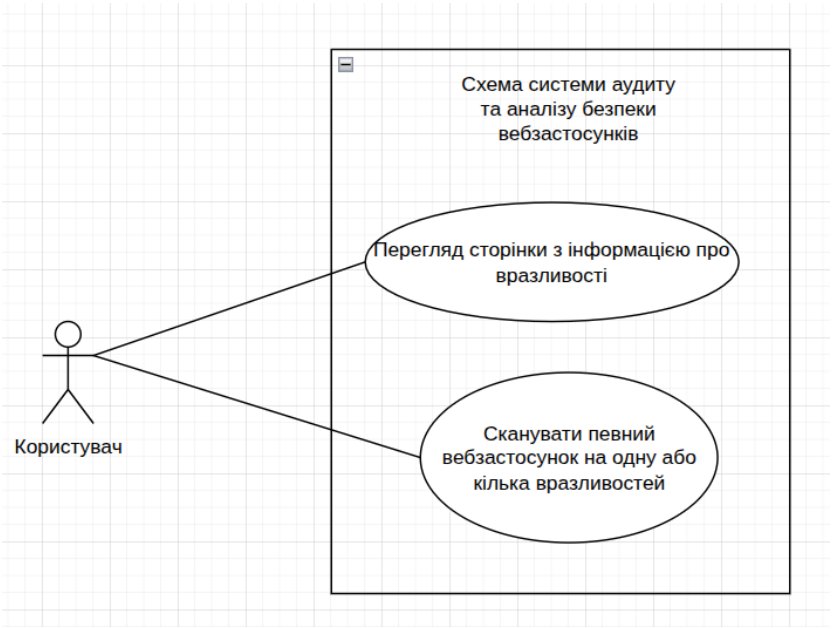


Рисунок 2.3 – Діаграма варіантів використання

Для глибшого розуміння структури системи використовується діаграма класів (Class Diagram), яка відображає класи, їх атрибути, методи та зв'язки між ними. Дана діаграма є центральною у розробці бек-енд логіки, оскільки вона визначає основну структуру даних і взаємозв'язок між ними. Наприклад, у додатку для обліку користувачів та їх замовлень діаграма класів дозволяє детально описати сутності "User", "Order", "Product" та їхні взаємозв'язки. Це

спрощує розробку моделі бази даних і допомагає запобігти багатьом помилкам на етапі кодування.

Діаграма класів є основою для розуміння структури об'єктно-орієнтованих систем. Вона дозволяє моделювати класи, їхні атрибути, методи, а також взаємозв'язки між ними. У класовій діаграмі кожен клас представлений прямокутником рисунок 2.4, який складається з трьох секцій: верхня частина містить назву класу, середня описує атрибути, а нижня включає методи, що реалізують поведінку об'єкта. Кожен з атрибутів має певний тип, що визначає, які дані можуть бути збережені, і модифікатор доступу (наприклад, public, private або protected) рисунок 2.5, який визначає рівень доступу до цього атрибута з інших об'єктів або класів. Методи також мають визначені модифікатори доступу та можуть повертати певні значення або виконувати дію без повернення даних [21,22].

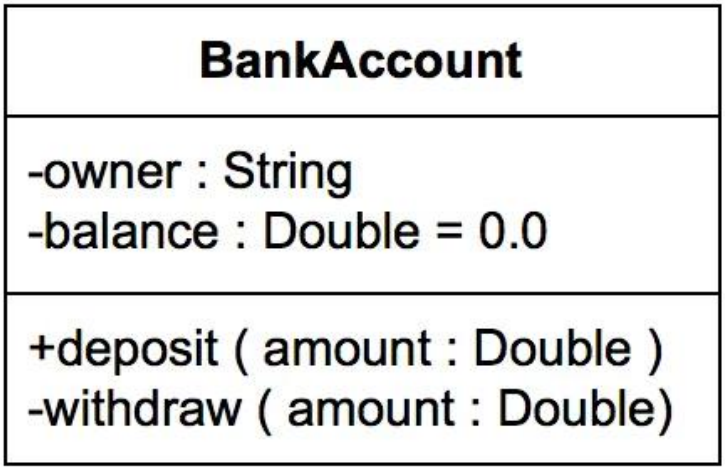


Рисунок 2.4 – Представлення класу [22]

public	+	anywhere in the program and may be called by any object within the system
private	-	the class that defines it
protected	#	(a) the class that defines it or (b) a subclass of that class
package	~	instances of other classes within the same package

Рисунок 2.5 – модифікатори доступу [22]

Зв'язки між класами відображають, як об'єкти взаємодіють один з одним або залежать один від одного. Існує кілька типів зв'язків: асоціація, агрегація, композиція та наслідування. Асоціація вказує на те, що один клас знає про існування іншого і може взаємодіяти з ним. Це найпростіший тип зв'язку, який показує просто взаємозв'язок між об'єктами. Агрегація демонструє слабкий зв'язок, де один об'єкт може існувати незалежно від іншого. Наприклад, факультет може складатися з багатьох викладачів, але навіть якщо факультет перестане існувати, викладачі залишаються. Композиція, на відміну від агрегації, вказує на сильну залежність, де один об'єкт не може існувати без іншого. Наприклад, кімнати не можуть існувати без будинку, якщо будинок знищується – зникають і кімнати. Наслідування визначає ієрархію класів, де підклас успадковує атрибути та методи батьківського класу. Це дозволяє використовувати загальні властивості в базовому класі та додавати специфічні для підкласів, що забезпечує повторне використання коду та підвищує його гнучкість. Наша діаграма рисунок 2.7 складається з семи класів та двох зв’язків асоціація та наслідування.

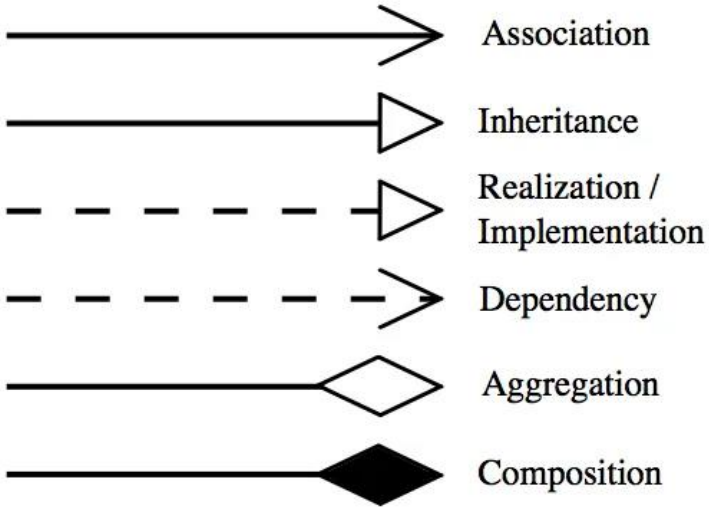


Рисунок 2.6 – Зв’язки між класами [22]

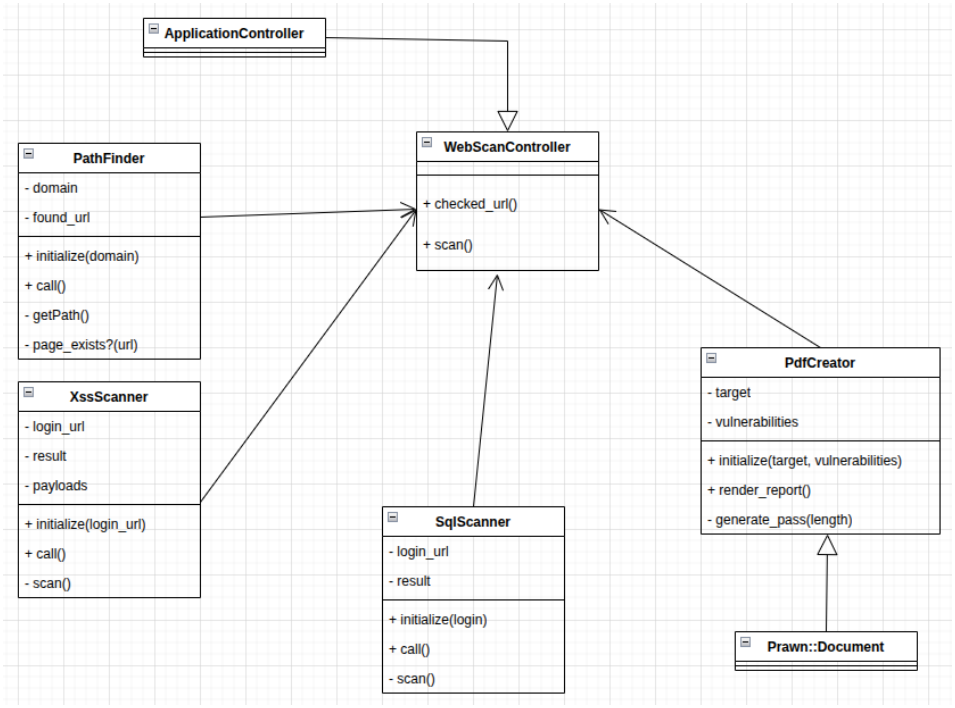


Рисунок 2.7 – Діаграма класів

Діаграма компонентів (Component Diagram) є важливою для визначення архітектури додатку. Вона демонструє, з яких компонентів складається система, як вони взаємодіють один з одним та з іншими зовнішніми системами. Ця діаграма надає повне уявлення про структуру додатку з точки зору модулів, сервісів та залежностей між ними. Вона особливо важлива під час проектування

бек-енд частини, оскільки дозволяє наочно визначити, які сервіси мають бути незалежними, а які взаємодіють один з одним.

У центрі діаграми лежить концепція модульності. Кожен компонент має певну відповідальність і повинен взаємодіяти з іншими через чітко визначені інтерфейси. Саме для цього в UML-нотації використовуються спеціальні символи портів і інтерфейсів. Порт це точка взаємодії компонента із зовнішнім світом або з іншими компонентами, яка забезпечує чіткий вхід і вихід даних. Порт часто позначається як маленький квадрат, розміщений на межі компонента. Порти можуть бути як надаючими, так і споживаючими тобто компонент може або надавати певну функціональність (наприклад, API-інтерфейс для зовнішніх клієнтів), або навпаки очікувати певний функціонал від інших частин системи. Це дозволяє гнучко компонувати архітектуру і за потреби легко замінювати або переробляти окремі модулі, не торкаючись усього застосунку.

Інтерфейси в діаграмі компонентів позначаються або відкритим кружечком (якщо інтерфейс надається компонентом), або порожнім півколом (якщо інтерфейс очікується компонентом від зовнішнього середовища). Інтерфейс задає набір операцій або методів, які доступні зовнішнім системам чи іншим компонентам для взаємодії. Наприклад, інтерфейс UserRepository може бути реалізований компонентом DatabaseLayer, а інший компонент, такий як UserService, буде цей інтерфейс споживати. Завдяки такому поділу досягається слабке зв’язування тобто компоненти знають лише про інтерфейс, а не про реалізацію, що дозволяє змінювати внутрішню логіку одного модуля без потреби змінювати інші частини системи.

Зв’язки між компонентами показують залежності: хто з ким взаємодіє, хто використовує чий функціонал, які дані передаються через інтерфейси. Це може бути як односпрямований зв’язок коли один компонент просто викликає метод іншого так і взаємозалежність, коли два модулі мають доступ один до одного. Саме ці зв’язки дозволяють оцінити складність системи, визначити критичні шляхи взаємодії, знайти потенційні точки відмови або перевантаження [23,24].

Діаграма станів (State Diagram) дозволяє описати життєвий цикл об'єкта в системі, включаючи всі можливі стани, в яких він може перебувати, і переходи між ними. Це корисно для фронт-енд розробників під час створення інтерфейсів, де необхідно чітко визначити, як відображати стан об'єктів у різні моменти часу. Наприклад, стан замовлення може змінюватися від "Очікується оплата" до "Відправлено" або "Скасовано" в залежності від дій користувача або адміністратора.

Ключовим поняттям у цій діаграмі є стан це певне визначене положення або конфігурація об’єкта, в межах якого він має конкретну поведінку або очікує на подію. Наприклад, об’єкт Замовлення може мати стани Нове, Опрацьовується, Відправлено, Скасовано. Переходи між цими станами відбуваються під дією подій, які можуть бути зовнішніми, такими як дії користувача, або внутрішніми, наприклад результат обчислення або зміна значення параметра. Ці переходи на діаграмі позначаються стрілками, які ведуть від одного стану до іншого і можуть супроводжуватись умовами або діями, що виконуються в момент зміни стану.

Важливою особливістю діаграми є можливість уточнення поведінки не лише в момент переходу, а й під час перебування у стані. Це означає, що можна вказати, які дії виконує об’єкт протягом часу, коли він перебуває у певному стані наприклад, відображення таймера, спостереження за сенсорами або очікування відповіді від сервера. Це робить діаграму станів особливо цінною для проєктування систем із відкладеною реакцією або систем реального часу, де поведінка залежить не лише від подій, а й від затримки у часі або внутрішніх обмежень.

Ще однією важливою концепцією є вкладені стани або композитні стани, коли в межах одного великого стану можуть існувати підстани. Це дозволяє уникати перевантаження діаграми великою кількістю дрібних переходів і водночас деталізувати поведінку об’єкта на більш глибокому рівні. Наприклад, у стані Активна сесія можна виділити підстани Редагування, Перегляд, Очікування, кожен з яких має свою специфічну поведінку, але всі вони належать до одного ширшого контексту.

Діаграми станів особливо ефективні при моделюванні реактивних систем, які реагують на події: графічні інтерфейси, ігри, протоколи зв'язку, контролери пристроїв усі ці системи зручно описуються саме через зміну станів. Вони дозволяють задати сувору логіку поведінки, виключити недетермінізм, уникнути помилок, пов'язаних із неправильним порядком виконання дій, та сприяти побудові стійкої до збоїв системи [25,26].

2.2 Створення користувацького інтерфейсу

Важко на сьогоднішній день уявити процес створення застосунку який не включав проектування UI/UX дизайну. UI – user interface (інтерфейс користувача) відповідає за зовнішній вигляд. Визначає якого кольору та форми будуть елементи. Обирає шрифт та стиль. UX – user experience (досвід користувача) відповідає за зручність, досліджує та формує шлях користувача.

Наш застосунок буде розрахований на новачків, які не полюбляють великі громіздкі системи. Добре продуманий дизайн діє як схема, яка описує інтерфейс користувача, процес навігації та загальний досвід користувача. Це раннє планування допомагає переконатися, що кінцевий продукт є інтуїтивно зрозумілим, зручним для користувача та відповідає потребам цільової аудиторії. Це також дозволяє розробникам і зацікавленим сторонам візуалізувати, як різні компоненти будуть взаємодіяти, полегшуючи виявлення потенційних проблем і вирішуючи їх завчасно.

Одним із ключових етапів у процесі розробки дизайну є детальне планування та створення макетів, що дозволяють уникнути багатьох помилок на наступних етапах роботи. Завчасне визначення функціональних вимог та особливостей взаємодії користувача допомагає вирішити потенційні проблеми ще на етапі проектування, забезпечуючи узгодженість у візуальному стилі та логіці роботи застосунку. Це дозволяє досягти цілісності та зрозумілості в кінцевому продукті, спрощуючи процес реалізації та тестування.

Крім того, продумана структура дизайну значно полегшує масштабованість та майбутні оновлення. Чітко визначені елементи та зрозуміла архітектура забезпечують можливість легко додавати нові функції або вносити необхідні зміни, не порушуючи загальної логіки та зручності використання. Це особливо важливо в умовах швидкого розвитку технологій, коли очікування користувачів постійно зростають, а вимоги до продуктів швидко змінюються. Завдяки цьому підходу, кожне оновлення чи доповнення інтегрується більш плавно та ефективно, зберігаючи стабільність і якість застосунку [27,28].

Наша система буде мати небагато сторінок, він буде включати головну сторінку на якій користувач зможе ввести адресу ресурсу який вона хоче перевірити, а також довідкову сторінку на якій він зможе детально прочитати про кожну з вразливостей яку виявляє дана система. Тобто даний застосунок з таким розрахунком можна буде використовувати в навчальних цілях.

Для створення макету нашої системи найкращим інструментом є Figma. Ця платформа надає широкі можливості для проектування інтерфейсів, дозволяючи працювати над дизайном у реальному часі та забезпечуючи зручність у командній розробці. Завдяки інтуїтивно зрозумілому інтерфейсу та потужним функціям, Figma спрощує процес візуалізації ідей та їх швидке тестування. Це особливо важливо на етапі проектування, коли необхідно багато експериментувати з розташуванням елементів та кольоровою гамою.

Під час роботи над макетом можна легко вносити зміни, обговорювати деталі з командою та швидко виправляти недоліки, що значно скорочує час на редагування. Крім того, можливість створення інтерактивних прототипів дозволяє не тільки побачити, як система виглядатиме, але й протестувати основні сценарії взаємодії користувача ще до етапу розробки. Це дає змогу виявити слабкі місця та покращити досвід користувача, не витрачаючи ресурсів на виправлення помилок у вже готовому продукті.

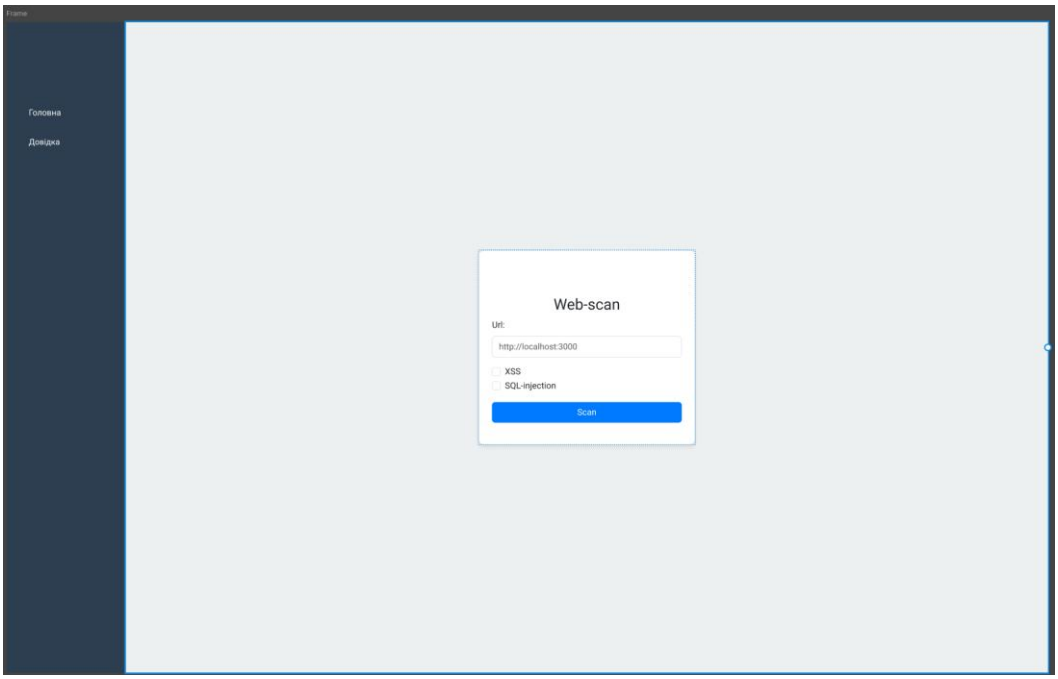


Рисунок 2.8 – Інтерфейс головної сторінки

Використання Figma у процесі проектування сприяє більш чіткій структуризації роботи, полегшує спілкування з іншими учасниками команди та дозволяє легко інтегрувати дизайнерські рішення у фінальну реалізацію продукту. Завдяки цим перевагам, створення макету стає не просто етапом підготовки до розробки, а повноцінною частиною формування кінцевого бачення продукту, що враховує зручність, естетику та функціональність майбутньої системи [29,30].

2.3 Вибір середовища для розробки

Вибір правильного середовища розробки – це один із ключових кроків на шляху до успішної реалізації програмного продукту. Від цього рішення залежить не лише зручність написання коду, але й ефективність відлагодження, тестування та підтримки проєкту в довгостроковій перспективі. Оптимально підібрана IDE (Integrated Development Environment) або редактор коду здатні суттєво підвищити продуктивність розробника, мінімізувати кількість помилок та спростити

інтеграцію з сучасними інструментами. Для розробки на Ruby існує декілька популярних рішень, кожне з яких має свої особливості, переваги та недоліки. Однак, серед усього розмаїття інструментів саме Visual Studio Code здобув визнання як найзручніший і найпотужніший інструмент для створення Ruby-додатків.

Visual Studio Code (VS Code) став одним з найпопулярніших редакторів коду серед розробників, завдяки своїй багатофункціональності, швидкодії та підтримці широкого спектра мов програмування, включаючи Ruby. Вибір середовища розробки для написання програмного коду на Ruby має велике значення, оскільки від цього залежить продуктивність розробника, зручність написання та підтримки коду, а також ефективність відлагодження та тестування. Серед великої кількості IDE, які підходять для розробки на Ruby, варто виділити наступні: RubyMine, Sublime Text, Atom та, звичайно, Visual Studio Code. RubyMine, розроблений JetBrains, пропонує повноцінне інтегроване середовище розробки з потужними інструментами для рефакторингу коду, зручним автодоповненням та вбудованим тестуванням [31].

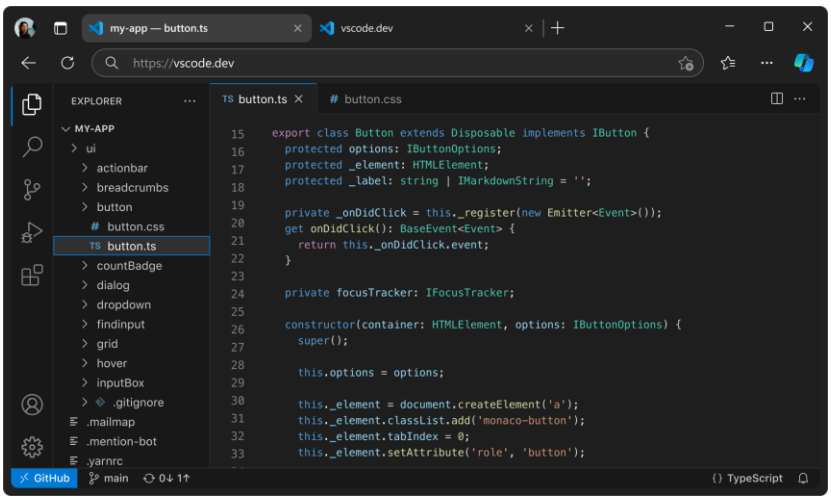


Рисунок 2.9 – Visual Studio Code [43]

RubyMine підтримує дебагінг з можливістю встановлення брейкпоінтів, огляду стеку викликів та аналізу змінних в реальному часі. Це середовище особливо добре підходить для великих проектів, де важливо мати під рукою всі

можливі інструменти для спрощення розробки та відлагодження. Завдяки інтеграції з системами контролю версій, такими як Git, розробники можуть легко відстежувати зміни, порівнювати версії файлів і вирішувати конфлікти безпосередньо в середовищі IDE. Автоматичне доповнення коду та аналіз помилок в реальному часі дозволяють швидко виявляти потенційні проблеми та оптимізувати написання коду. Також варто відзначити зручну інтеграцію з тестовими фреймворками, такими як RSpec і Minitest, що спрощує процес написання і запуску тестів. Однак, RubyMine має і недоліки: це комерційний продукт, що потребує платної підписки, що може бути недоступним для індивідуальних розробників або невеликих команд. Крім того, його продуктивність значно нижча порівняно з іншими редакторами на менш потужних машинах, що може впливати на швидкість роботи, особливо під час індексації великих проектів. Деякі розробники також відзначають певну перенасиченість функціоналом, що може ускладнити використання IDE для простих задач. Незважаючи на це, для великих корпоративних проектів RubyMine залишається одним із найпотужніших рішень, забезпечуючи розробників усіма необхідними інструментами для ефективної роботи [32].

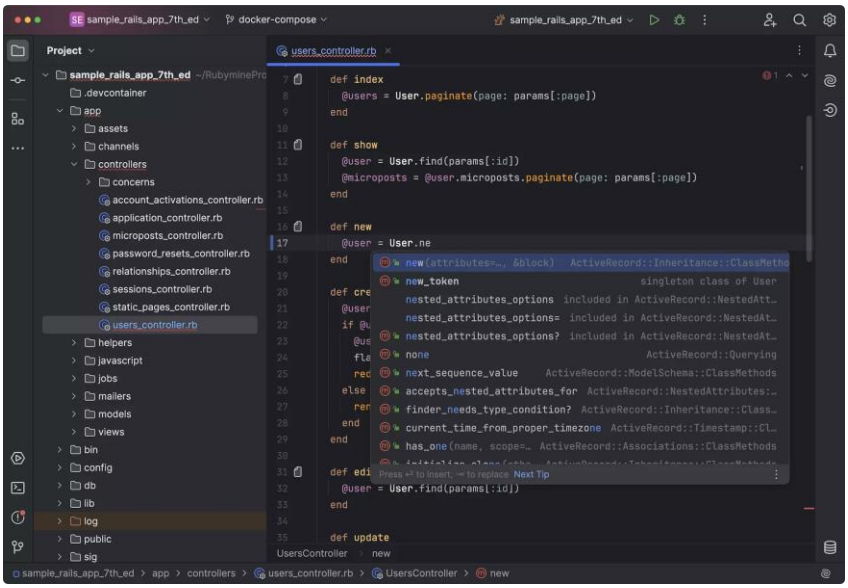


Рисунок 2.10 – RubyMine [44]

Sublime Text – це легкий та швидкий редактор коду з мінімалістичним інтерфейсом. Завдяки своїй легкості він запускається майже миттєво та не навантажує системні ресурси, що робить його ідеальним вибором для швидкого редагування коду або виправлення помилок. Підтримка великої кількості мов програмування, зокрема Ruby, реалізована через численні плагіни, які можна легко інтегрувати за допомогою Package Control. Серед найбільш корисних доповнень для Ruby виділяються LSP (Language Server Protocol) для автодоповнення та перевірки синтаксису, а також плагіни для форматування коду та підсвічування синтаксису. Sublime Text також відомий своєю зручною навігацією по файлах і можливістю відкривати великі проекти без втрати продуктивності. Крім того, інструмент підтримує multi-caret редагування, що дозволяє редагувати декілька місць у коді одночасно, підвищуючи ефективність розробки. Проте, Sublime Text більше підходить для невеликих проєктів або швидкого редагування скриптів, оскільки в ньому відсутні повноцінні інструменти для дебагінгу та тестування. Для реалізації більш складних сценаріїв відлагодження необхідно використовувати зовнішні інструменти або IDE, що додає зайві кроки в робочий процес. Також Sublime Text не надає вбудованих можливостей для інтеграції з системами контролю версій, такими як Git, хоча відповідні плагіни частково компенсують цей недолік. Відсутність повноцінної підтримки тестових фреймворків, таких як RSpec чи Minitest, робить його менш зручним для написання та запуску тестів у Ruby. Це може стати перешкодою при роботі над великими проєктами, де тестування і дебагінг є невід'ємною частиною розробки. Незважаючи на свої обмеження, Sublime Text залишається чудовим вибором для легких задач, швидких виправлень коду та роботи з файлами без зайвих затримок [33].

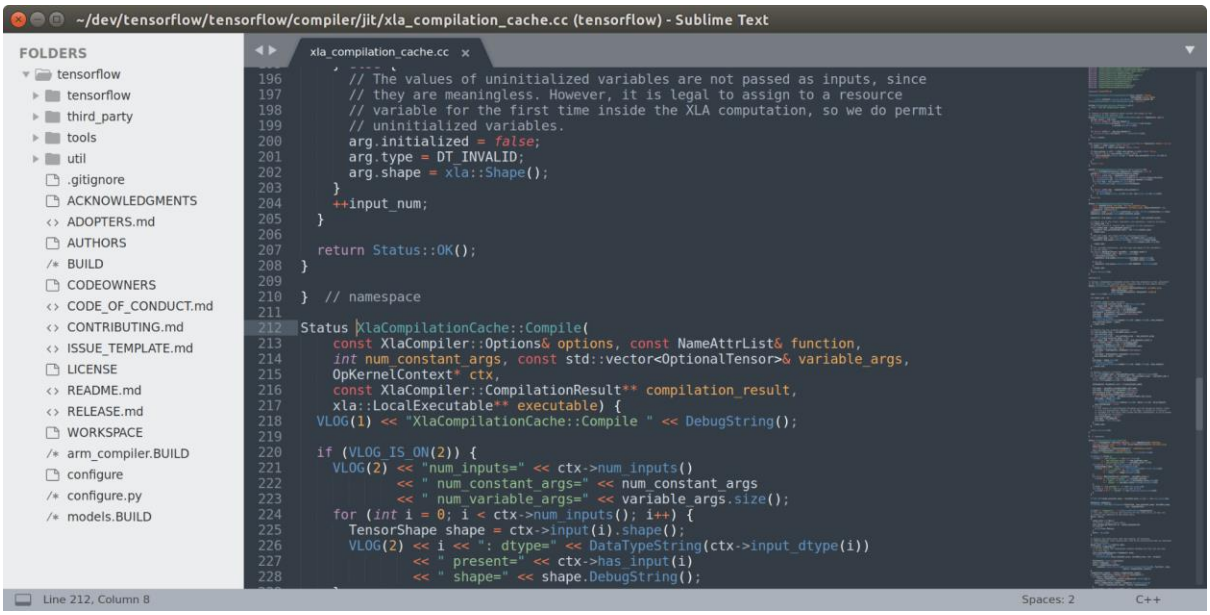


Рисунок 2.11 – Sublime Text [45]

Visual Studio Code став справжнім проривом у сфері редакторів коду. Це безкоштовне та відкрите середовище розробки, яке поєднує у собі простоту редактора та потужність IDE. Завдяки великій кількості розширень, VS Code легко адаптується під Ruby-розробку. Серед основних плагінів для Ruby варто виділити Solargraph для автодоповнення коду, Rubocop для статичного аналізу, а також плагін Ruby Test Explorer, який дозволяє запускати RSpec-тести безпосередньо з інтерфейсу редактора.

Окрім цього, VS Code має вбудований термінал, що дозволяє швидко запускати Ruby-команди, перевіряти стан Git-репозиторію та виконувати команди Docker чи інші DevOps-задачі. Інтеграція з GitHub та можливість одночасної роботи кількох розробників через Live Share значно спрощує процес командної розробки. Ще однією перевагою є стабільна підтримка спільнотою, регулярні оновлення та активний розвиток екосистеми розширень. Завдяки оптимізації під різні платформи, VS Code працює однаково швидко як на Windows, так і на macOS та Linux.

Отже серед всіх перерахованих середовищ розробки найкраще нам підійде саме Visual Studio Code оскільки вона найбільш універсальна, продуктивним та доступним рішенням для розробки на Ruby. Його гнучкість, велика кількість

розширень, підтримка сучасних інструментів розробки та безкоштовність роблять його очевидним вибором для розробників будь-якого рівня – від новачка до професіонала.

2.4 Інсталяція потрібного ПЗ та ініціалізація проекта

Що ж для початку нам потрібно встановити потрібні компоненти для створення нашого застосунку. У нашому випадку це мова програмування ruby та фреймворк Ruby on Rails, який являється бібліотекою даної мови програмування. Також ми встановимо цікавий інструмент “Docker”, який легко інтегрується з даним фреймворком та полегшить життя подальшим клієнтам які будуть використовувати наш застосунок.

Docker – це безкоштовна платформа, яка дозволяє розробникам пакувати програми та їхні залежності в контейнери. Ці контейнери легкі портативні та забезпечують однакову роботу програми незалежно від середовища. Тобто встановивши його користувачу який буде використовувати нашу систему не прийдеться задумуватись як встановити мову програмування ruby та фреймворк RoR [34].

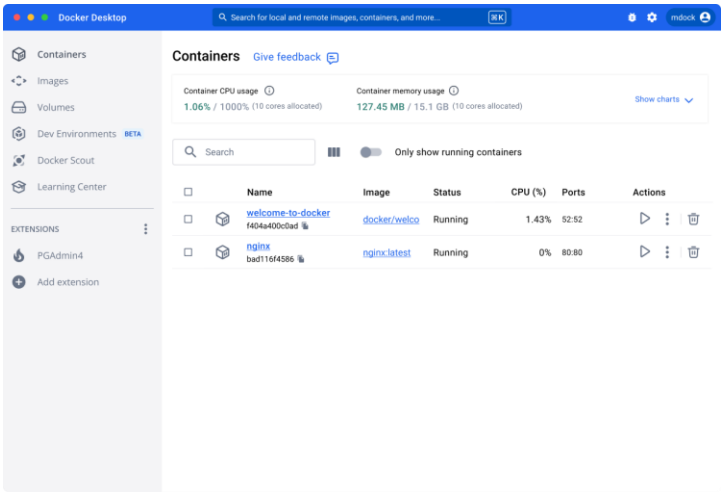


Рисунок 2.12 – Користувацький інтерфейс Docker [46]

Docker можна використовувати як з графічним інтерфейсом що є більш зручним для новачків так і в консолі. Ми в свою чергу працюємо на операційній системі Linux тому будемо використовувати консольний варіант, оскільки його встановлення є складнішим ніж встанволення з графічним інтерфейсом на macOS або Windows.

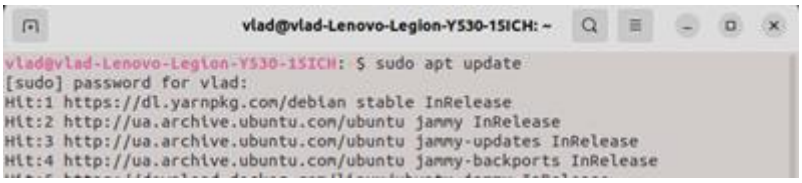


Рисунок 2.13 – Оновлення списку доступних пакетів

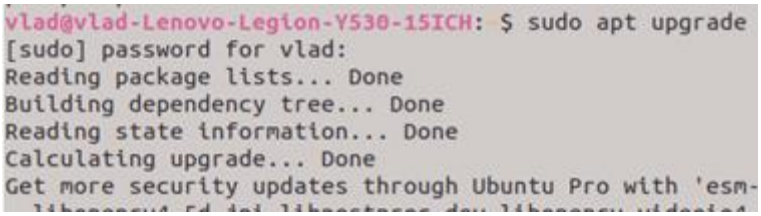


Рисунок 2.14 – Оновлення пакетів

Рисунок 2.15 – Встановлення необхідних залежностей

Рисунок 2.16 – Додавання офіційного репозиторія Docker

Рисунок 2.17 – Встановлення docker та docker compose

Рисунок 2.18 – Перевірка версій docker та docker compose

Тепер перейдемо до встановлення мови програмування ruby та фреймворка Ruby on Rails. Для цього найкращим рішенням являється інсталювати ruby version manager. Для цього перейдемо на сторінку “rvm.io” і виконуємо команди на рисунку 2.19 які показано на сайті.

Рисунок 2.19 – Інсталяція rvm, Ruby та Ruby on Rails

Тепер коли ми встановили все необхідне ми можемо створити проект але перед цим варто ознайомитись з усіма особливостями ініціалізації проекту. Для цього введемо команду на рисунку 2.20

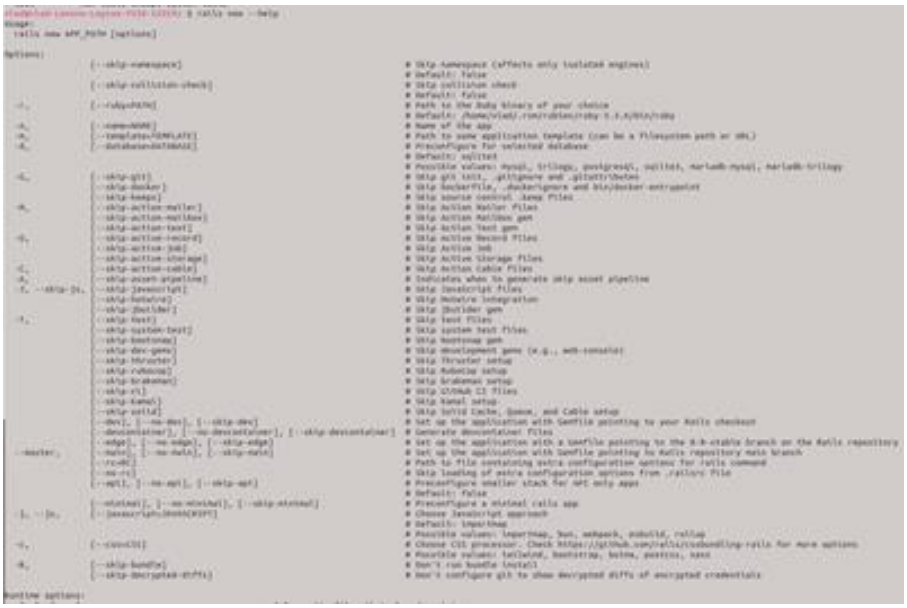


Рисунок 2.20 – Параметри для створення проекта

Як ми бачимо є безліч різних опцій які надають нам можливість гнучко ініціалізувати проект так щоб не було нічого лишнього. В нашому випадку ми не будемо використовувати БД для збереження а також нам не потрібен mailer. Тому в кінцевому варіанті наша команда на ініціалізацію проекта буде виглядати наступним чином:

```
vlad@vlad-Lenovo-Legion-Y530-15ICH: /khnu$ rails new scanner_app -O
Based on the specified options, the following options will also be activated:

  --skip-active-storage [due to --skip-active-record]
  --skip-solid [due to --skip-active-record]
  --skip-action-mailbox [due to --skip-active-storage]
  --skip-action-text [due to --skip-active-storage]

      create
      create  README.md
      create  Rakefile
      create  .ruby-version
      create  config.ru
      create  .gitignore
      create  .gitattributes
      create  Gemfile
         run  git init -b main from "."
Initialized empty Git repository in /home/vlad/khnu/scanner_app/.git/
      create  app
```

Рисунок 2.21 – Створення проєкта без БД

```
vlad@vlad-Lenovo-Legion-Y530-15ICH:~/scanner_app$ docker compose up --build
[+] Building 0.6s (12/12) FINISHED
 => [web internal] load build definition from Dockerfile
 => => transferring dockerfile: 334B
 => [web internal] load metadata for docker.io/library/ruby:3.2
```

Рисунок 2.22 – Створення та увімкнення контейнера

Тепер коли ми ініціалізували проект нам потрібно було б створювати конфігураційні файли для докера оскільки без них інтеграція докера неможлива, але наш фреймворк автоматично робить це все для нас, крім створення файла docker-compose.yml який містить список контейнерів які потрібно запустити разом. Хоча нам не потрібно запустити лише один контейнер, зручніше це буде зробити з допомогою даного файла. Далі нам потрібно змінити Dockerfile який за замовчуванням при ініціалізації проекта містить налаштування які нам не підходять оскільки ми не збираємось деплоїти наш проект на сервер.

2.5 Розробка frontend та backend частини

Для початку інсталюємо усі потрібні бібліотеки для нашого проекту. В нашому випадку нам знадобляться бібліотека “prawn” та “byebyug”. Гем “prawn” відповідає за генерацію PDF файлів Ruby, яка надає багато цікавих функцій та достатньо хорошу продуктивність роботи. Даний гем є офіційно представлений на сайті RubyGems що надає впевненість для нас що вона добре перевірена на вміст зловмисного коду. В свою чергу “byebug” дозволяє розробнику зупиняти виконання програми в певних точках, переглядати змінні, досліджувати поточний стан застосунку та взаємодіяти з ним у режимі реального часу. Це засіб для глибокого розуміння того, як працює код, і для швидкого знаходження помилок. Використання “byebug” є особливо корисним у складних додатках, де простий аналіз коду може бути недостатнім [35,36].

```
vlad@vlad-Lenovo-Legion-Y530-15ICH:~/khnu/scanner_app$ docker exec -it scanner_app-web-1 bash
root@070e21bbe823:/app# bundle install
Fetching gem metadata from https://rubygems.org/.........
Resolving dependencies...
Fetching ttfunk 1.8.0
Fetching pdf-core 0.10.0
Installing pdf-core 0.10.0
Installing ttfunk 1.8.0
Fetching prawn 2.5.0
Installing prawn 2.5.0
Bundle complete! 18 Gemfile dependencies, 113 gems now installed.
Use `bundle info [gemname]` to see where a bundled gem is installed.
1 installed gem you directly depend on is looking for funding.
  Run `bundle fund` for details
root@070e21bbe823:/app#
```

Рисунок 2.23 – Інсталяція бібліотеки “prawn”

```
vlad@vlad-Lenovo-Legion-Y530-15ICH:~/khnu/scanner_app$ bundle install
Fetching gem metadata from https://rubygems.org/.........
Fetching byebug 11.1.3
Installing byebug 11.1.3 with native extensions
```

Рисунок 2.24 – Інсталяція бібліотеки “byebug”

Тепер коли все готово до розробки проекта варто розглянути яку архітектуру ми будемо використовувати в нашому проекті. За замовчуванням фреймворк базується на архітектурі MVC, яка базується на взаємодії трьох ключових компонентів це Controller, Model та View. В ній ключову роль відіграє база даних, яка не буде використовуватись в нашому застосунку. Тому нам варто задуматись над іншою архітектурою більш зручною та сучасною. В голову приходить кілька найбільш підходящих варіантів це VCS (View-Controller-Service), Action-Oriented Architecture (Command/Interactor Pattern), Lambda/Functional-style Architecture.

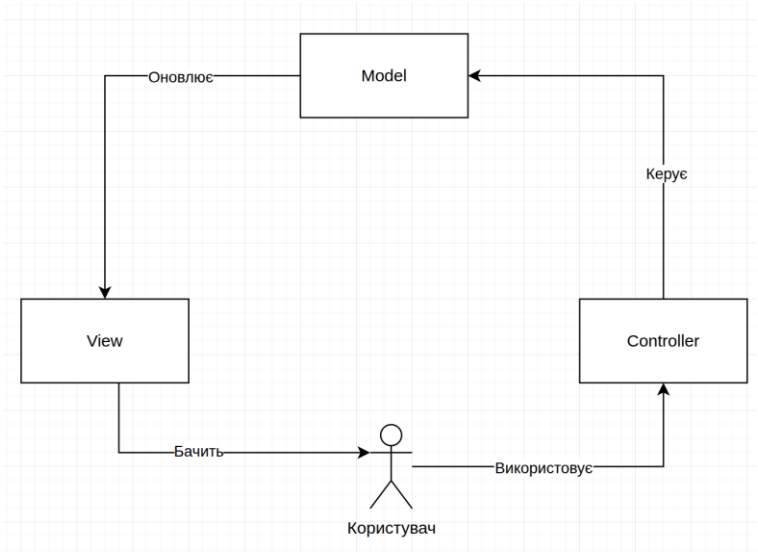


Рисунок 2.25 – Архітектура MVC

Архітектура Action-Oriented базується на ідеї, що кожна бізнес-операція або дія оформляється як окремий клас. Такий клас виконує одну конкретну задачу, наприклад, обробку тексту, перевірку даних, чи відправку листа. Перевагою даного підхода являється чітка інкапсуляєція логіки, простота у використанні, зручність у тестуванні. Він ідеально підходить, коли кожен запит у застосунку відповідаю одному сценарію або операції.

Рисунок 2.26 – Архітектура Action-Oriented

Архітектура Lambda/Functional-style підходить для маленьких проектів де потрібно писати мінімум коду, замість сервісів в use case класів можна використовувати прості Ruby-модулі з методами. Це максимально простий, але ефективний підхід, коли логіка дуже легка. Проблемою являється що даний архітектура ламає майбутні можливості по оновленню проекту оскільки при розширенні втрачається читабельність та логіка ускладниться.

В свою чергу архітектура VCS вирішує проблему громіздких контролерів. Це коли метод контроллера містить дуже багато бізнес-логіки застосунку, який в результаті стає важко читабельним навіть для людини яка його створювала і повернулась до нього через деякий час. Контроллер буде викликати потрібний сервіс який буде реалізовувати бізнес-логіку. Тобто застосунок буде працювати наступним чином якщо ми будемо використовувати дану архітектуру. Controller приймає HTTP-запити, викликає потрібні сервіси, потім сервіс виконує поставлену на нього бізнес-логіку, валідацію і тд і в кінцевому view відображає результат. Даний підхід зручний якщо не використовується база даних, виконує бізнес-логіку на основі вхідних параметрів або запитів.

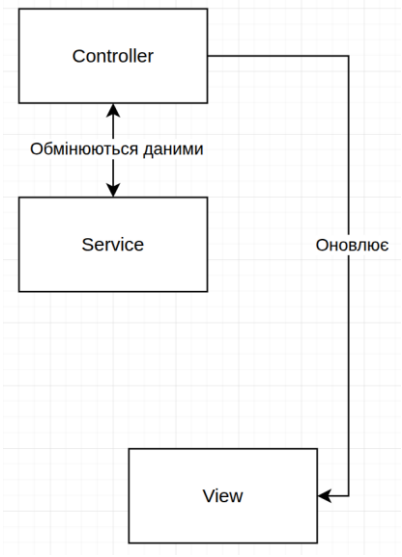


Рисунок 2.27 – Архітектура VCS

Отже, найбільш підходящою у нашій ситуації виявляється архітектура VCS (View-Controller-Service), оскільки вона надає змогу в подальшому розширювати наш аналізатор без значних змін у структурі коду. Цей підхід чітко розділяє логіку представлення, обробки даних та бізнес-логіку, що дозволяє підтримувати проект більш гнучким та зрозумілим для розробників. Завдяки поділу на окремі шари, кожен компонент може розвиватися незалежно, зберігаючи цілісність загальної архітектури. Наприклад, будь-які зміни у відображенні інтерфейсу користувача не зачіпають логіку контролерів або бізнес-операцій, що значно спрощує процес внесення модифікацій.

Окрім того, така структура сприяє більш ефективному тестуванню, оскільки кожен з шарів можна перевіряти окремо. Це дозволяє швидко виявляти помилки та ізолювати проблеми без необхідності повного перегляду всієї системи. Підхід VCS також сприяє кращому управлінню залежностями та полегшує інтеграцію нових сервісів або розширення існуючих. У майбутньому, коли виникне необхідність додавання нових функціональних можливостей до системи, це можна буде зробити без ризику порушити роботу інших компонентів.

Саме завдяки цій архітектурі ми отримуємо можливість не лише розвивати функціонал, але й підтримувати високу якість коду, що забезпечує його

стабільність та легкість у підтримці. VCS стає основою для побудови надійної та масштабованої системи, здатної адаптуватись до нових вимог і технологічних змін, зберігаючи при цьому логічну структуру та зрозумілість для розробників.

Тепер можна приступити до розробки front-end частини. Для того щоб зменшити час верстання нашого застосунку, тобто перенесення нашого макету який ми намалювали вручну через код, ми будемо використовувати вже готові бібліотеки стилів що збереже нам багато часу при адаптовані сторінок під різні екрани та зменшить громісткість нашого проекту. В нашому випадку ми використаємо бібліотеку bootstrap – це один з найбільш популярних фреймворків css, який призначений для адаптивної розробки, містить багато вже готових стилів що дає змогу розробнику зберегти багато часу під час розробки. Для того щоб почати верстати сторінку нам потрібно.

Тепер перейдемо до реалізації back-end частини. Для цього, на початковому етапі, необхідно розробити алгоритм, за яким буде функціонувати уся серверна логіка. Алгоритм є основою, на якій будується робота всієї системи, він визначає послідовність дій, правила обробки запитів, управління даними та забезпечення взаємодії між компонентами. Це свого роду "архітектурний план", що допомагає не лише структуровано підходити до реалізації коду, але й робить процес підтримки та масштабування значно простішим і зрозумілішим.

На етапі розробки алгоритму важливо продумати, як саме відбуватиметься обмін інформацією між клієнтом і сервером. Це включає визначення типів запитів (GET, POST, PUT, DELETE), їх маршрутизацію, обробку отриманих даних та формування відповіді для клієнта. Наприклад, при запиті на отримання списку користувачів сервер має виконати певну послідовність дій: отримати запит від клієнта, звернутися до бази даних для вибірки даних, обробити отримані результати (наприклад, відсортувати або профільтрувати), а потім відправити відповідь назад клієнту у вигляді структури даних (наприклад, JSON).

Крім того, алгоритм має передбачати логіку валідації даних. На цьому етапі відбувається перевірка коректності отриманих даних перед їхньою обробкою. Це дозволяє уникнути можливих помилок на рівні програми.

Алгоритм повинен містити логіку обробки помилок. Помилки можуть виникати на різних етапах: під час надсилання некоректного формату url, в процесі пошуку потрібного url для сканування, при відправленні відповіді на клієнтську частину. Для забезпечення стабільної роботи застосунку необхідно передбачити всі вище перелічені помилки які можуть виникнути при використанні нашої системи. Оскільки якщо проігнорувати їх то в кінцевому варіанті вийде сирий недопрацьований застосунок з яким буде важко взаємодіяти бо не зрозуміло буде користувачу чому не видало в результаті файл з результатами тестування.

Завершальним етапом побудови алгоритму є визначення структури відповіді сервера. Це критично важливо, оскільки клієнтська частина має отримувати чітко структуровані дані, щоб коректно їх відображати. Найчастіше це реалізується через JSON або XML формати, які легко парсяться та обробляються на фронтенді.

Таким чином, правильно спроєктований алгоритм для back-end частини є основою надійності, безпеки та масштабованості всього застосунку. Це підвищує передбачуваність роботи системи, забезпечує її гнучкість та готовність до майбутніх змін чи розширення функціоналу.

Наш проєкт загально буде функціонувати по наступному алгоритму. Першою перевіркою буде чи введено в поле введення ціль сканування, якщо нічого не введено то видасть помилку “Будь ласка, вкажіть URL”. При введенні відбудеться перевірка чи даний ресурс активний в разі якщо ні то видасть помилку “Домен неактивний!”. Далі відбувається перевірка на те чи вибраний хоча би один з елементів сканування, якщо нічого не вибрано було то видасть про це помилку. При успішно пройдених попередніх перевірках відбудеться пошук шляху який веде на форму авторизації, якщо даний пошук не буде успішний то користувачу виводиться помилка “Не знайдено цілі для

сканування!”. Далі відбувається сканування на вказану або вказані вразливості. При закінченні сканування відбувається формування та надсилання звіту у форматі pdf та кодів для відкриття даного звіту.

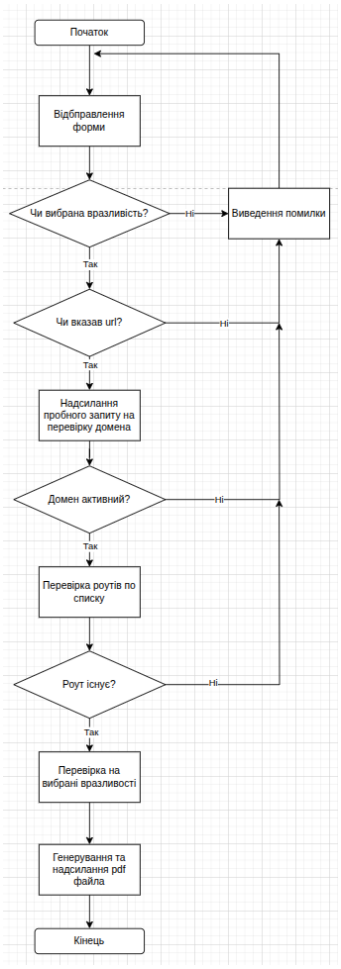


Рисунок 2.28 – Загальний алгоритм роботи системи

Тепер коли ми спроектували алгоритм за яким повністю буде працювати система ми можемо створювати потрібний нам контролер та сервіси які будуть виконувати кожен свою задачу. В загальному у нас буде чотири сервіси: “path_finder”, “pdf_creator”, “sql_scanner” та “xss_scanner”. У файлі “path_finder.rb” буде міститись реалізація пошуку потрібного маршруту до форми з авторизацією. “pdf_creator.rb” буде створювати відповідний звіт за результатами сканування які нададуть йому два наступних сервіси. В “sql_scanner” реалізується базова перевірка на sql-injection в разі успішного виявлення буде змінено значення атрибуту “result” на значення true. В

“xss_scanner” реалізований функціонал по виявленню межсайтового скрипту по аналогії з попереднім сервісом при успішному виявленні буде змінено значення атрибуту так само названого.

Також не варто забувати що нам потрібно десь зберігати список зі всіма можливими шляхами які ведуть до форми авторизації. Є два варіанти або на пряму зберігати в файлі “path_finder.rb” або ж створити окремий файл “.yml”. Другий варіант являється більш практичним з урахуванням того що даний сканер буде в подальшому модифіковувати. Оскільки якщо кількість можливих шляхів збільшиться до кількох сотень а то й тисяч то це погіршить читабельність написаного сервісу. Тож для реалізації даного варіанту нам потрібно створити два файли, один в каталозі config та інший в config/initializer. В першому каталозі потрібно створити файл з назвою “common_paths.yml”, в ньому будуть міститись всі можливі шляхи, у вигляді великого списку. Другий файл з назвою “common_paths.rb”. В результаті отримуємо рисунку 2.29 загальну картину структури нашого проєкту.

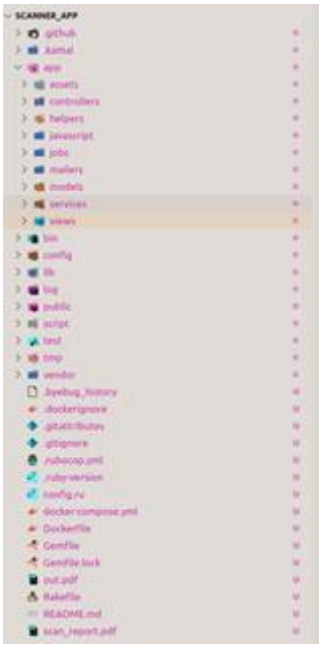


Рисунок 2.29 – Структура проєкта

Як і в будь-якому великому чи малому проєкті, розробники стикаються з різними проблемами, які можуть виникати на кожному етапі розробки. На

початковій стадії це може бути визначення правильної архітектури, вибір підходящого стеку технологій та планування структури бази даних. Під час імплементації виникають виклики, пов'язані з оптимізацією коду, забезпеченням продуктивності та сумісністю між різними модулями. Окрім цього, часто доводиться враховувати сумісність з різними операційними системами та версіями браузерів, що може ускладнити відлагодження. Важливою частиною розробки є тестування, оскільки без належної перевірки система може мати критичні вразливості або працювати нестабільно під навантаженням. Підтримка коду та розширення функціоналу після запуску також є не менш складними завданнями, адже зміни в одній частині програми можуть впливати на інші компоненти, що вимагає ретельного тестування та моніторингу. Додатково, інтеграція з іншими сервісами, робота з API та налаштування безпеки можуть стати причиною додаткових труднощів, які потребують не лише технічної компетенції, а й уміння швидко адаптуватися до нових умов. Таким чином, успішна розробка будь-якого програмного забезпечення – це не просто написання коду, а постійний процес вирішення проблем, пошуку оптимальних рішень та підтримки високої якості продукту.

На нашому шляху до розробки нашої системи ми також стикулись з певними проблемами, першою з них виявилась проблема того що базовий список символів наданий гемом “prawn” за замовчуванням з набором символів Windows-1252, який не містить кирилицю, на щастя нами вибраний фреймворк це все розписав і навіть запропонував рішення нашої проблеми, а саме додати в наш застосунок розширені шрифти. Знайти шрифт який підтримує кодування UTF-8 доволі легко та безкоштовно для цього потрібно їх скачати з офіційного сайту гугл по шрифтах у форматі архіву який в подальшому потрібно розпакувати та перенести та інтегрувати в наш застосунок.

Наступною проблемою у процесі розробки виявилася типовою помилкою Render and/or redirect were called multiple times in this action, яка виникає у випадку, коли в одному екшені контролера відбувається одночасне викликання render і redirect_to, або ці методи викликаються більше одного разу. У рамках

Ruby on Rails, кожна дія контролера має завершуватись лише одним із варіантів відповіді: або render, або redirect_to. Більше того, ці методи не припиняють виконання екшену, тому після їх виклику потрібно явно завершити метод, наприклад, через return, щоб уникнути конфлікту.

Для розв’язання цієї проблеми я застосував нестандартний, але ефективний підхід. Замість того, щоб відразу викликати redirect_to після встановлення повідомлення flash, я залишив відповідь у вигляді render, а на стороні клієнта реалізував таймер за допомогою JavaScript. Цей таймер очікує декілька секунд, аби дати користувачу змогу побачити повідомлення flash, після чого автоматично виконує перезавантаження сторінки або редірект. Такий підхід дозволяє уникнути конфлікту між render і redirect_to в межах одного екшену, а також забезпечує коректне відображення повідомлення flash, яке зазвичай втрачається при негайному перезавантаженні без редіректу.

Таким чином, використання таймеру стало простим і надійним способом зберегти логіку повідомлення flash, не порушуючи правила відповідей у Rails, і водночас покращило досвід користувача завдяки зрозумілому візуальному зворотному зв’язку перед перезавантаженням.

3 ТЕСТУВАННЯ СТВОРЕНОЇ СИСТЕМИ

3.1 Перевірка роботоспроможності на різних ОС

Тестування застосунку на різних операційних системах, таких як Windows, Linux та macOS, є критично важливим етапом розробки програмного забезпечення, який дозволяє забезпечити стабільну роботу продукту в різноманітних середовищах. Це завдання передбачає вивчення поведінки програми в умовах різних архітектур, налаштувань та специфічних особливостей кожної з платформ. Оскільки операційні системи відрізняються за структурою ядра, файловою системою, принципами управління пам’яттю та мережею, важливо переконатися, що застосунок зберігає свою функціональність та коректність незалежно від того, під якою платформою він запускається.

Тестування застосунку на різних операційних системах є критично важливим етапом у забезпеченні його стабільності та сумісності. Кожна операційна система має свої унікальні особливості в обробці системних викликів, управлінні пам'яттю, роботі з файловою системою та мережевими протоколами. Те, що працює бездоганно на Windows, може викликати помилки або нестабільну поведінку на Linux чи macOS через різні версії бібліотек, різну структуру файлової системи або навіть інші налаштування безпеки. Наприклад, доступ до певних системних файлів або прав користувача може відрізнятися, що вимагає додаткової перевірки. Більше того, інструменти розробки та бібліотеки можуть вести себе по-різному на різних платформах, що може призвести до неочікуваних результатів під час виконання коду. Це особливо актуально для веб-застосунків, які часто взаємодіють із системними компонентами, такими як драйвери або засоби керування мережевими підключеннями. Відсутність тестування на різних ОС може спричинити непередбачувані помилки в роботі програмного забезпечення, що може негативно вплинути на користувацький досвід та завдати фінансових збитків через простої або втрату даних. Крім того, тестування на різних операційних системах дозволяє виявити потенційні проблеми з безпекою, оскільки кожна платформа має свої механізми захисту та

обробки викликів. Забезпечення коректної роботи застосунку в різних середовищах підвищує його надійність та розширює аудиторію користувачів, що позитивно впливає на репутацію продукту.

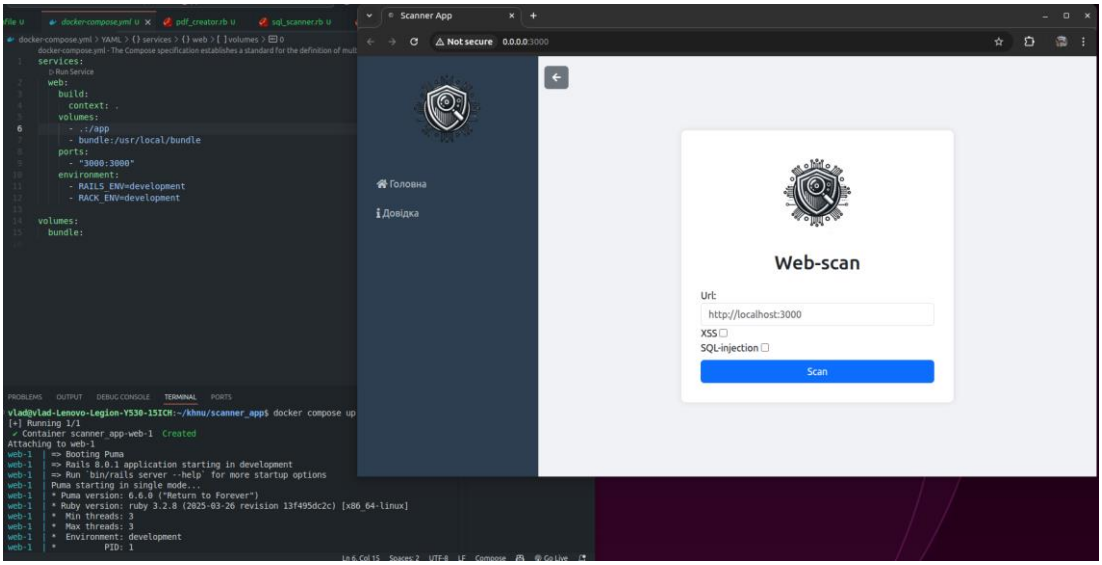


Рисунок 3.1 – Успішний запуск на Linux

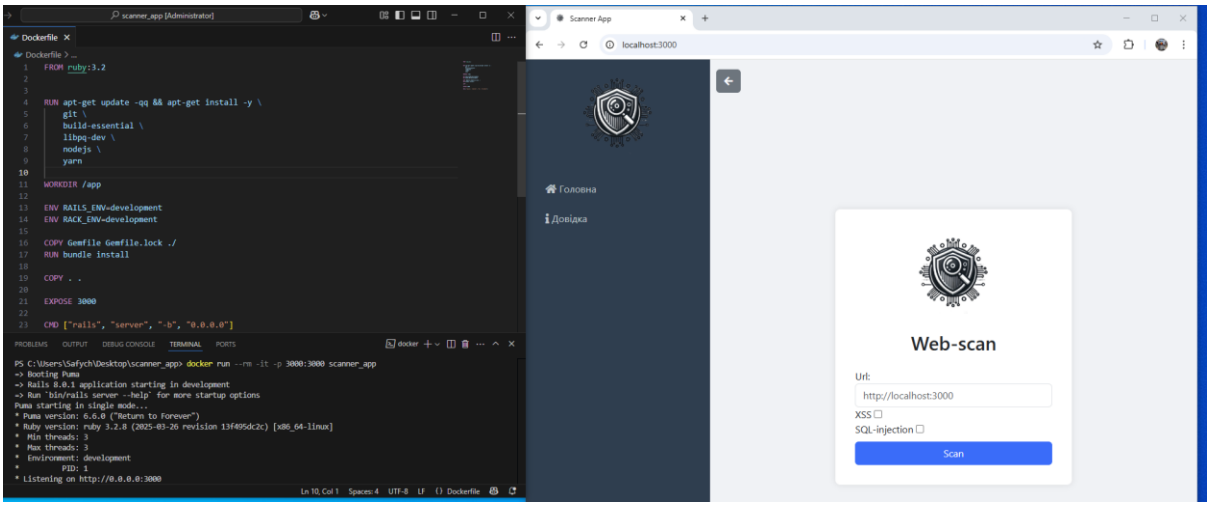


Рисунок 3.2 – Успішний запуск на Windows

Для спрощення процесу тестування на різних платформах, у цьому проєкті використовується Docker – інструмент контейнеризації, який дозволяє ізолювати середовище виконання. Docker забезпечує можливість створення уніфікованих контейнерів, що включають всі необхідні залежності та конфігурації для запуску додатку на будь-якій операційній системі. Це значно спрощує процес

розгортання та тестування, оскільки один і той самий контейнер може бути запущений на Windows рисунок 3.2, Linux рисунок 3.1 або macOS без необхідності налаштування оточення вручну. Завдяки цьому можна гарантувати, що код, який пройшов тестування в Docker-контейнері, буде працювати стабільно ідентично на різних операційних системах. Крім того, Docker дозволяє легко масштабувати процес тестування, піднімаючи одночасно декілька контейнерів з різними конфігураціями, що забезпечує швидкий зворотній зв'язок та виявлення помилок ще на ранніх етапах розробки.

3.2 Тестування функціональних спроможностей

Тестування в Ruby on Rails є невід'ємною частиною процесу розробки, що забезпечує якість, стабільність та безпеку додатку. Rails включає в себе три основних види тестування: Unit testing, Integration testing та System testing, кожен з яких має своє призначення та специфіку реалізації, рисунок 3.3 перелік усіх необхідний бібліотек для тестування. Крім того рисунок 3.4 потрібно виконати генерування конфігураційних файлів. Unit testing – це найнижчий рівень тестування, який орієнтований на перевірку окремих методів або моделей у відриві від інших частин додатку.

Unit testing дає змогу переконатися, що кожна функція або метод працює згідно з очікуваннями. Наприклад, якщо є модель User з методом full_name, тест має перевірити, чи повертає цей метод правильне об'єднання імені та прізвища. Unit тести виконуються ізольовано від бази даних або зовнішніх API, завдяки чому вони проходять швидко і дають чітке розуміння працездатності окремого компонента. Для написання таких тестів у Rails використовується RSpec або MiniTest. Також важливою практикою є використання моків та стабів для імітації поведінки залежностей.

В нашому випадку ми маємо два варіанти реалізації тестів це RSpec та Minitest. Кожен з них має свої плюси та мінуси. Minitest є вбудованим рішенням

у сам Ruby, тому він не потребує додаткової установки і є набагато ближчим до «стандартної бібліотеки». Він простий, мінімалістичний та дуже швидкий. Стиль написання тестів у Minitest нагадує класичні підходи, засновані на тестових класах та методах, що починаються з test_. Цей фреймворк добре підходить для розробників, які цінують простоту, лаконічність і хочуть бути максимально близькими до «рідного» Ruby. Minitest також добре інтегрується з Rails і дозволяє писати як юніт, так і інтеграційні або системні тести, але вимагає більше ручної організації тестових сценаріїв. Через свою простоту Minitest може виглядати менш виразно, особливо в великих проєктах, де важлива читаємість та підтримка складної тестової структури. RSpec – це потужніший фреймворк, створений спеціально для написання зрозумілих, описових тестів у стилі поведінкового програмування (BDD). Він надає розширену мову опису, яка дозволяє будувати дуже читабельні тести, що звучать майже як жива англійська мова. Наприклад, замість assert_equal, у RSpec пишеться expect(...).to eq(...), що краще передає логіку перевірки. У великих проєктах RSpec особливо цінується за свою гнучкість, підтримку модульності через describe, context, та можливість легко налаштовувати специфічні блоки підготовки та очищення даних. Завдяки великій екосистемі додаткових бібліотек, таких як Shoulda Matchers або FactoryBot, тестування з RSpec стає ще зручнішим і швидшим [37,38].

Виходячи з порівняння ми будемо використовувати RSpec через його більш зрозумілий та природний синтаксис, який легко читається навіть розробниками, що не мають глибоких знань у Ruby, для початку рисунок 3.3 додамо потрібні геми яких не вистачає, а далі рисунок 3.4 інсталюємо їх та згенеруємо конфігураційні файли. Читабельність даних тестів є особливо важливою для підтримки тестів, адже код, написаний мовою, наближеною до людської, легше зрозуміти та розширювати.

```
group :test do
  gem "capybara"
  gem "selenium-webdriver"
  gem "webdrivers"
  gem "rspec-rails"
end
```

Рисунок 3.3 – Перелік гемів для тестування

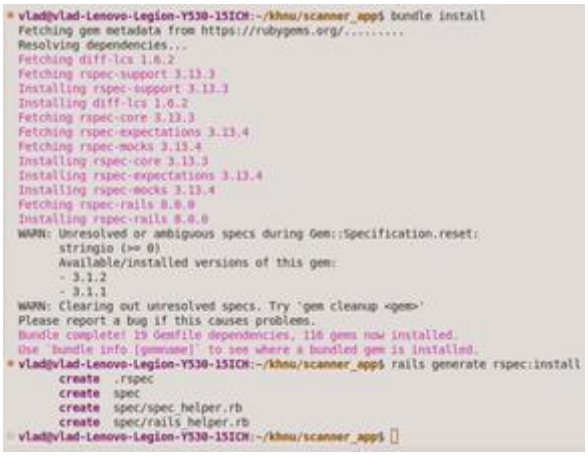


Рисунок 3.4 – Генерування конфігураційних файлів

RSpec дозволяє створювати чіткі описи поведінки моделей і контролерів, що робить процес тестування більш прозорим. Крім того, він підтримує можливість логічного групування тестів за допомогою конструкцій “describe” та “context”, що допомагає структурувати навіть великі набори тестів у зручну для навігації форму. Це спрощує аналіз тестових сценаріїв та дозволяє швидко знаходити необхідний блок коду. Важливим аспектом є те, що RSpec пропонує багатий набір інструментів для перевірки станів об'єктів, викликів методів, виключень та інших умов, що робить тестування більш точним та гнучким. Також RSpec має широку підтримку спільноти та велику кількість додаткових бібліотек, які інтегруються без зайвих труднощів. Наприклад, для тестування HTTP-запитів часто використовується бібліотека Capybara, яка бездоганно працює з RSpec, забезпечуючи повну емуляцію взаємодії користувача з додатком. Це дозволяє легко створювати інтеграційні та системні тести, покриваючи не лише логіку моделей, але й реальну поведінку у браузері. У той

час як MiniTest, безсумнівно, є потужним і швидким інструментом, RSpec виграє за рахунок своєї читабельності, чіткої структури та гнучкості в описі тестів, що значно спрощує підтримку та розширення тестового коду в процесі розвитку проєкту.

Юніт-тестування в контексті RSpec – це фундаментальний підхід до забезпечення якості коду, який полягає в перевірці найменших одиниць функціональності застосунку в повній ізоляції від інших компонентів. У світі Ruby on Rails такими одиницями найчастіше виступають методи моделей, окремі сервіси або модулі, які містять логіку, що не залежить від зовнішніх ресурсів, як-от база даних, веб-інтерфейс чи сторонні API. Основна мета юніт-тестування – переконатися, що кожна окрема одиниця виконує свої функції саме так, як це задумано, і реагує правильно як на типові вхідні дані, так і на граничні або некоректні ситуації.

Тестування кожного сервісу у програмному застосунку має критичне значення для забезпечення якості, надійності та стабільності всієї системи. У сучасних архітектурах, зокрема в Ruby on Rails, сервіси використовуються для винесення бізнес-логіки за межі моделей та контролерів, що дозволяє зберігати чисту структуру коду та спрощувати підтримку. Однак саме тому сервіси стають ключовими компонентами, від яких залежить правильність роботи багатьох процесів, і навіть незначна помилка в одному з них може призвести до серйозних збоїв на вищому рівні [39].

Тестування сервісів дозволяє переконатися, що кожен окремий елемент логіки функціонує відповідно до очікувань рисунок 3.5, навіть якщо його поведінка зміниться в майбутньому. Коли сервіси покриваються юніт-тестами, кожен сценарій, кожна умова та кожна гілка логіки може бути перевірена ізольовано, без впливу з боку інших частин системи. Це дає змогу відловлювати баги ще до того, як вони виявляться у взаємодії з інтерфейсом або базою даних, а також забезпечує впевненість у тому, що окрема функціональність працює саме так, як було задумано.

Ще одним аргументом на користь тестування сервісів є гнучкість при зміні реалізації. У процесі розробки часто виникає потреба у рефакторингу – заміні частини логіки, оптимізації алгоритмів або додаванні нових параметрів. Якщо для сервісу існує повноцінний набір тестів, розробник може вносити зміни без страху, що щось поламається. Усі критичні сценарії вже зафіксовані в тестах, і кожна помилка одразу стане помітною.

Рисунок 3.5 – Тестування pdf creator

Integration testing дозволяє протестувати взаємодію між різними компонентами системи. У Rails інтеграційні тести перевіряють роботу маршрутизації, запити до контролерів, взаємодію моделей між собою та процеси, що відбуваються під час виконання користувацьких сценаріїв. Це допомагає виявити проблеми, що можуть виникнути під час взаємодії кількох моделей чи контролерів одночасно. Наприклад, можна перевірити, чи коректно відправляються форми, чи правильно обробляються помилки під час запитів, та чи відображається потрібна інформація на сторінці після успішного виконання

операцій. У процесі інтеграційного тестування можуть бути використані такі бібліотеки, як Capybara для імітації дій користувача та WebMock для відсікання зовнішніх HTTP-запитів під час тестування.

Рисунок 3.6 – Успішно виконаний системний тест

System testing – це комплексний вид тестування, що дозволяє перевірити роботу додатку у веб-браузері. З допомогою інструментів, таких як Capybara та Selenium, можна емулювати дії користувача: заповнення форм, натискання кнопок, навігацію між сторінками. Це дозволяє протестувати повну взаємодію з додатком з точки зору кінцевого користувача, включаючи JavaScript і AJAX-запити. System тести є максимально наближеними до реальної взаємодії з додатком, оскільки вони проходять через весь цикл обробки запиту, включаючи рендеринг сторінок. Наприклад, можна автоматизувати процес реєстрації користувача, додавання товару в корзину або здійснення покупки, перевіряючи при цьому всі етапи взаємодії. В нашому випадку рисунок 3.6 ми ми перевіряємо чи нормально працює навігація в нашому застосунку [40].

ВИСНОВКИ

У процесі виконання кваліфікаційної роботи було розроблено прикладну систему аудиту та аналізу безпеки вебзастосунку для автоматизованого сканування вебресурсів на наявність критичних вразливостей, зокрема міжсайтового скриптування (XSS) та ін’єкцій SQL-запитів (SQL Injection). Обидва ці типи атак тривалий час залишаються в топі найбільш небезпечних загроз за версією OWASP, що підкреслює їхню актуальність у сучасному кіберпросторі. Реалізація системи велася з урахуванням практичних потреб спеціалістів з безпеки, тому акцент був зроблений не лише на виявленні загроз, а й на створенні зручного механізму звітування, що не потребує постійного збереження інформації.

Від самого початку було визначено цілі, які охоплювали створення інструменту з чіткою логікою сканування та безпекою обробки результатів. Ключовою особливістю реалізованої системи стало те, що результати сканування не зберігаються в базі даних, а видаються одразу у вигляді захищеного паролем звіту у форматі PDF. Такий підхід не лише підвищує рівень конфіденційності, але й мінімізує ризики витоку чутливої інформації про вразливості у разі компрометації системи зберігання даних. Створення таких звітів реалізовано за допомогою бібліотеки prawn, яка забезпечила гнучкі можливості оформлення, шифрування та стилізації документів. У кожному звіті надається перелік знайдених вразливостей, вказано дату та час виконання даного сканування.

У розробці було використано фреймворк Ruby on Rails, який дозволив швидко побудувати архітектуру яка базується на використанні сервісів. Це забезпечило гнучкість при розробці й можливість масштабування системи в майбутньому, наприклад, через додавання нових типів вразливостей або інтеграцію з системами CI/CD.

Для контролю якості коду та стабільності роботи застосунку активно використовувались бібліотека byebug для покрокового дебагінгу та RSpec – потужний фреймворк для модульного тестування. Завдяки цьому вдалось

виявити та усунути низку помилок ще на ранніх етапах розробки, що в перспективі суттєво зменшує витрати на підтримку та супровід коду. Тестування проводилось із різними сценаріями, включно з граничними значеннями параметрів, невалідними запитами, а також симуляцією реальних XSS та SQLi-атак.

Особливу увагу було приділено питанням кросплатформенності та простоти розгортання. Для досягнення цього метою система була контейнеризована за допомогою Docker. Це дозволило створити ізольоване середовище з усіма необхідними залежностями, що гарантує стабільну роботу незалежно від операційної системи користувача – будь то Windows, Linux чи macOS. Крім того, використання Docker значно спрощує процес розгортання системи як для локального використання, так і для інтеграції в хмарну інфраструктуру чи серверну платформу компанії.

Загалом створена система є прикладом ефективного використання сучасних засобів веброзробки для вирішення задач інформаційної безпеки. Вона демонструє можливість побудови потужного інструменту для первинного тестування безпеки без необхідності складних інсталяцій чи довготривалого навчання користувачів. Формування захищеного звіту дозволяє забезпечити збереження результатів навіть в умовах підвищених вимог до конфіденційності та відповідності внутрішнім політикам безпеки організацій.

У результаті реалізований продукт успішно виконує поставлені задачі, забезпечуючи виявлення XSS та SQL-ін’єкцій, генерування інформативного PDF-звіту, роботу на різних операційних системах та відповідність практичним потребам фахівців у галузі кібербезпеки. Отриманий результат може слугувати основою для подальшого розвитку системи – зокрема, через інтеграцію з базами знань вразливостей, розширення інтерфейсу для взаємодії через API або побудову графічного вебінтерфейсу.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАНЬ

1 What is cross-site scripting, URL: https://portswigger.net/web-security/cross-site-scripting (дата звернення 30.04.2025)

2 Grossman J., Hansen R., Petkov P., Rager A. XSS Attacks: Cross Site Scripting Exploits and Defense. Burlington : Syngress, 2007. 312 с.

3 Cross-Site Scripting (XSS), URL: https://medium.com/@yadav-ajay/cross-site-scripting-xss-eb66824493ae (дата звернення 30.04.2025)

4 Describing XSS: The story hidden in time, URL: https://medium.com/@ryoberfelder/describing-xss-the-story-hidden-in-time-80c3600ffe81 (дата звернення 03.05.2025)

5 What is SQL Injection, URL: https://portswigger.net/web-security/sql-injection (дата звернення 03.05.2025)

6 Clarke J. SQL Injection Attacks and Defense. Burlington : Syngress, 2012. 312 с.

7 What is path traversal, URL: https://portswigger.net/web-security/file-path-traversal (дата звернення 03.05.2025)

8 File Inclusion Path Traversal, URL: https://medium.com/@RosanaFS/tryhackme-file-inclusion-path-traversal-9f99395562c8 (дата звернення 03.05.2025)

9 OS command injection, URL: https://portswigger.net/web-security/os-command-injection (дата звернення 04.05.2025)

10 OS command injection, URL: https://medium.com/@nishadbabu1015/os-command-injection-8cd2a1a56743 (дата звернення 05.05.2025)

11 Discovered Sensitive Data, URL: https://medium.com/@anonymousshetty2003/discovered-sensitive-data-exposure-on-wati-com-a-case-study-using-sublist3r-and-dirbuster-409a03f8f75d (дата звернення 05.05.2025)

12 What is OWASP? WHAT is the OWASP Top 10, URL: https://www.cloudflare.com/learning/security/threats/owasp-top-10/ (дата звернення 07.05.2025)

13 What is OWASP, URL: https://medium.com/@prathapkbutta/what-is-owasp-e5962241f2d2 (дата звернення 07.05.2025)

14 Burp Suite: Overview, URL: https://medium.com/@dancovic/burp-suite-overview-3401280d05a5 (дата звернення 07.05.2025)

15 Mastering Burp Suite Vulnerability Scanner, URL: https://medium.com/@1200km/mastering-burp-suites-vulnerability-scanner-019ed82c8bac (дата звернення 07.05.2025)

16 RubyGems.org your community gem host, URL: https://rubygems.org/ (дата звернення 08.05.2025)

17 Introduction to Ruby on Rails, URL: https://medium.com/@zachlandis91/introduction-to-ruby-on-rails-6cafde5b5373 (дата звернення 08.05.2025)

18 Ruby on Rails fundamental concepts, URL: https://medium.com/@bhuvaneshganesan/ruby-on-rails-fundamental-concepts-a3f6b92b0127 (дата звернення 10.05.2025)

19 UML діграми, їхні основні типи та процес розроблення, URL: https://dou.ua/forums/topic/40575/ (дата звернення 11.05.2025)

20 System design UML diagram, URL: https://medium.com/@wainaina.pierre/system-design-uml-diagrams-a-summary-39cf6c9ab63c (дата звернення 12.05.2025)

21 Class diagram, URL: https://www.geeksforgeeks.org/unified-modeling-language-uml-class-diagrams/ (дата звернення 11.05.2025)

22 UML Class Diagrams Tutorial, URL: https://medium.com/@smagid_allThings/uml-class-diagrams-tutorial-step-by-step-520fd83b300b (дата звернення 11.05.2025)

23 Component Based Diagram, URL: https://www.geeksforgeeks.org/component-based-diagram/ (дата звернення 11.05.2025)

24 How to Implement UML Component Diagrams, URL: https://medium.com/initial-main/how-to-implement-uml-component-diagrams-in-systemverilog-3bcaf4b6b856 (дата звернення 11.05.2025)

25 State Machine Diagrams, URL: https://www.geeksforgeeks.org/unified-modeling-language-uml-state-diagrams/ (дата звернення 11.05.2025)

26 How to Create a State Diagram, URL: https://medium.com/geekculture/how-to-create-a-state-diagram-110d709c2fc8 (дата звернення 11.05.2025)

27 UX/UI дизайн. Інструменти та навички дизайнера, URL: https://medium.com/@vitalka057/ux-ui-3c0f9d931697 (дата звернення 12.05.2025)

28 UI/UX Fundamentals: Design Process for Beginners, URL: https://fr0st2o11.medium.com/ui-ux-fundamentals-design-process-for-beginners-b422bd3c441f (дата звернення 12.05.2025)

29 What is Figma, URL: https://www.geeksforgeeks.org/what-is-figma/ (дата звернення 12.05.2025)

30 Figma 101: A Beginner’s Guide, URL: https://medium.com/@farahqadeer30/title-figma-101-a-beginners-guide-to-getting-started-with-figma-be8d0303e54a (дата звернення 12.05.2025)

31 What Makes Visual Studio Code So Popular, URL: https://medium.com/adl-blog/what-makes-visual-studio-code-so-popular-54206c386503 (дата звернення 14.05.2025)

32 What is RubyMine, URL: https://medium.com/@michaelwnek/what-is-rubymine-c5a968c761c0 (дата звернення 14.05.2025)

33 Sublime Text, URL: https://mivocloud.com/blog/Sublime-Text-Why-Its-the-Perfect-Text-Editor-for-Your-Business (дата звернення 14.05.2025)

34 Introduction to Docker, URL: https://medium.com/@senali/introduction-to-docker-c159e5b102c4 (дата звернення 15.05.2025)

35 Репозиторій до гему prawn, URL: https://github.com/prawnpdf/prawn (дата звернення 15.05.2025)

36 Репозиторій до гему byebug, URL: https://github.com/deivid-rodriguez/byebug (дата звернення 15.05.2025)

37 Minitest Tutorial, URL: https://launchschool.medium.com/assert-yourself-a-detailed-minitest-tutorial-f186acf50960 (дата звернення 20.05.2025)

38 RSpec for Beginners, URL: https://medium.com/@radadiyahardik355/rspec-for-beginners-a-step-by-step-guide-to-testing-in-rails-a7d4de40401b (дата звернення 20.05.2025)

39 What is unit testing in rails, URL: https://imvishalpandey.medium.com/what-is-unit-testing-in-rails-9d56db36d59a (дата звернення 20.05.2025)

40 Difference Between System Testing and Integration Testing, URL: https://www.geeksforgeeks.org/difference-between-system-testing-and-integration-testing/ (дата звернення 21.05.2025)

41 OWASP ZAP, URL: https://medium.com/@1200km/owasp-zap-a-comprehensive-guide-to-web-application-security-testing-6c247f4be39b (дата звернення 21.05.2025)

42 What's new in Burp Suite Professional, URL: https://portswigger.net/blog/whats-new-in-burp-suite-professional-a-year-of-innovation (дата звернення 21.05.2025)

43 Visual Studio Code, URL: https://code.visualstudio.com/ (дата звернення 21.05.2025)

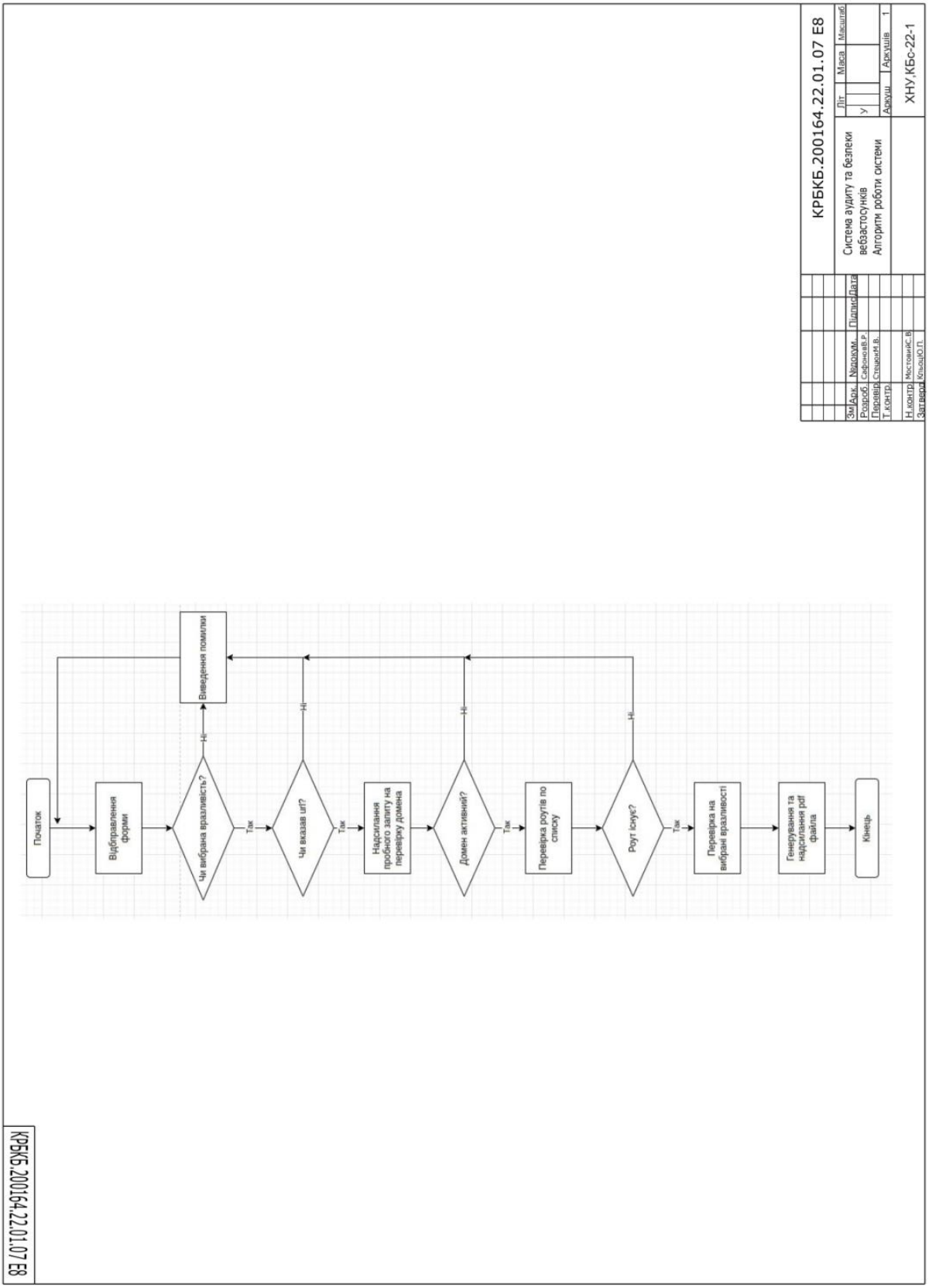
44 RubyMine, URL: https://www.jetbrains.com/ruby/ (дата звернення 21.05.2025)

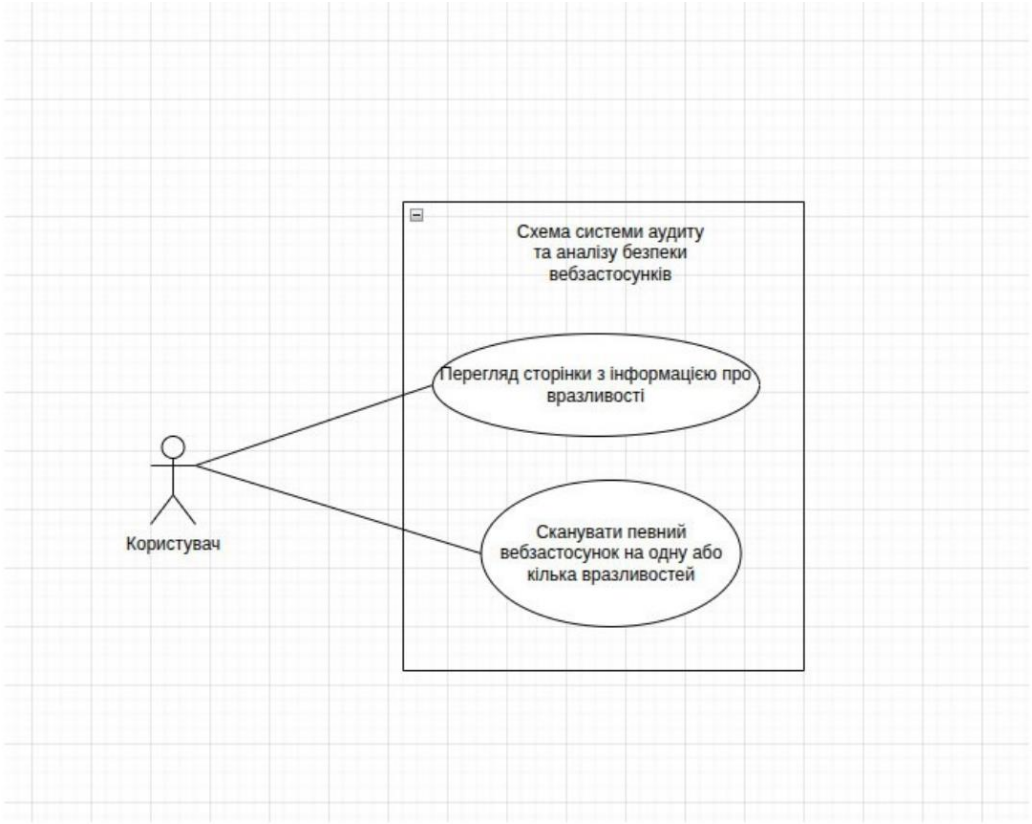
45 Sublime Text, URL: https://www.sublimetext.com/blog/articles/sublime-text-3-point-0 (дата звернення 21.05.2025)

46Docker, URL: https://www.docker.com/ (дата звернення 21.05.2025)

ДОДАТОК А

Копія графічної частини





						КРБКБ.200164.22.01.07 Е8		
Зм.	Арк.	№докум.	Підпис	Дата	Система аудиту та безпеки вебзастосунків Діаграма прецедентів	Літ	Маса	Масштаб
Розроб.		СафоновВ.Р.				У		
Перевір.		СтецюкМ.В.				Аркуш	Аркушів 1	
Т.контр.								
Н.контр.		МостовийС.В.				ХНУ,КБс-22-1		
Затверд.		КльоцЮ.П.						

КРБКБ.200164.22.01.07 Е8

Зм	Арк	Докум.	Підпис	Дата
Розроб.		Сафонов В.Р.		
Перевір.		Стешок М.В.		
Т. контр.				
Н. контр.		Мостовий С.В.		
Затверд.		Юнь Ю.П.		

Система аудиту та безпеки вебзастосунків
Діаграма класів

Літ	Маса	Масштаб
У		
Аркуш	Аркушів	1

ХНУ, КБс-22-1

КРБКБ.200164.22.01.07 Е8

ДОДАТОК Б

Програмний код сервісів сканера

pdf_creator.rb

```
class PdfCreator < Prawn::Document
 attr_reader :target, :vulnerabilities,
:owner_password, :user_password

 def initialize(target, vulnerabilities)
  super(page_size: 'A4', margin: 40)

  @target = target
  @vulnerabilities = vulnerabilities

  encrypt_document(
   user_password: generate_pass(12),
   owner_password: generate_pass(32),
   permissions: {
    print_document: false,
    modify_contents: false,
    copy_contents: false,
    modify_annotations: false
   }
  )

  self.font_families.update("OpenSans" => {
   :normal =>
Rails.root.join("vendor/assets/fonts/Open_San
s/static/OpenSans_Condensed-Regular.ttf"),
   :bold =>
Rails.root.join("vendor/assets/fonts/Open_San
s/static/OpenSans_Condensed-Bold.ttf")
  })

  font "OpenSans"
 end

 def render_report
  text "Звіт про безпековий аналіз", size: 24,
style: :bold, align: :center
  move_down 20

  text "Ціль сканування:", size: 16, style:
:bold
  text @target, size: 14
  move_down 10

  text "Результати перевірки:", size: 16,
style: :bold
  move_down 10

  text "• Вразливість XSS:
#{@vulnerabilities[:xss_found] ? 'Виявлено' :
'Не виявлено'}", size: 14, color:
@vulnerabilities[:xss_found] ? "ff0000" :
"008800"
  text "• Вразливість SQL Injection:
#{@vulnerabilities[:sql_found] ? 'Виявлено' :
'Не виявлено'}", size: 14, color:
@vulnerabilities[:sql_found] ? "ff0000" :
"008800"
   move_down 20
  text "Дата сканування:
#{Time.now.strftime('%d.%m.%Y
%H:%M')}", size: 10, align: :right
 end

 private

 def generate_pass(length)
  SecureRandom.hex(length)
 end
end
```

path_finder.rb

```
require 'net/http'

class PathFinder
 attr_reader :domain, :found_url

 def initialize(domain)
  @domain = domain
  @found_url = ""
 end

 def call
  getPath
 end

 private

 def getPath
  COMMON_PATHS.each do |path|
   url = "#{@domain}#{path}"
```

```
    if page_exists?(url)
      @found_url = url
      return
    end
  end
 end

 def page_exists?(url)
  response =
Net::HTTP.get_response(URI(url))
  return url if response.code.to_i == 200
 rescue StandardError => e
  false
 end
end
```

sql_scanner.rb

```
class SqlScanner
 attr_reader :login_url, :result

 def initialize(login_url)
  @login_url = login_url
  @result = false
 end

 def call
  scan
 end

 private

 def scan
  uri = URI.parse(@login_url)

  params = { username: "admin' --", password:
"random_password" }

  response = Net::HTTP.post_form(uri,
params)

  if response.code == "200"
    @result = true
  end
 end
end
```

xss_scanner.rb

```
class XssScanner
 attr_reader :login_url, :result, :payloads

 def initialize(login_url)
  @login_url = login_url
  @result = false
  @payloads = [
    "<script>alert('XSS')</script>",
    "<img src=x onerror=alert('XSS')>",
    "javascript:alert('XSS')",
    "<svg/onload=alert('XSS')>",
    "'\"><script>alert('XSS')</script>"
  ]
 end

 def call
  scan
 end

 private

 def scan
  uri = URI.parse(@login_url)

  @payloads.each do |payload|
    params = { username: payload, password:
"random_password" }

    response = Net::HTTP.post_form(uri,
params)

    if response.body.include?(payload)
      @result = true
    end
  end
 end
end
```

web_scan_controller.rb

```
require 'net/http'

class WebScanController <
ApplicationController
 before_action :checked_url

 def scan
  path_finder = PathFinder.new(params['url'])
  path_finder.call

  if path_finder.found_url.empty?
    flash[:error] = "Не знайдено цілі для
сканування!"
    return redirect_to root_path
```

```
  end

  sql_scanner =
SqlScanner.new(path_finder.found_url)
  xss_scanner =
XssScanner.new(path_finder.found_url)

  sql_scanner.call if params[:sql_injection]
  xss_scanner.call if params[:xss]

  pdf_creator = PdfCreator.new(params['url'],
{ xss_found: xss_scanner.result, sql_found:
sql_scanner.result })
  pdf_creator.render_report

  flash[:pdf_message] = "Будь ласка
випишіть дані паролі собі на листочок.
                Тільки для перегляду:
#{pdf_creator.user_password}
                Для усього:
#{pdf_creator.owner_password}"

  send_data(
    pdf_creator.render,
    filename: 'result.pdf',
    type: 'application/pdf',
    disposition: "attachment"
  )
 end

 def checked_url
  if(!params[:sql_injection] && !params[:xss])
    flash[:error] = "Ви не обрали
вразливість!"
    return redirect_to root_path
  end

  unless params['url'].present?
    flash[:error] = "Будь ласка, вкажіть URL"
    return redirect_to root_path
  end

  response =
Net::HTTP.get_response(URI(params['url']))
  response
 rescue
  flash[:error] = "Домен неактивний!"
  return redirect_to root_path
 end
end
```

Завідувачу кафедри кібербезпеки
к.т.н., доц. Кльоцу Ю.П.
Сафонова Владислава Руслановича
ПІБ здобувача вищої освіти
Студента ФІТ, 3 курсу, групи КБс-22-1

ЗАЯВА

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

Також надаю університету право на передачу моєї роботи для обробки та збереження в базах даних програмно-обчислювального комплексу StrikePlagiarism та/або програмнотехнічного засобу Anti-Plagiarism і використання роботи для виявлення академічного плагіату в інших роботах, які перевіряються програмно-технічними засобами та користувачами, що мають доступ до цих програмно-технічних засобів, виключно в обмежених цілях для виявлення текстових збігів в роботах.

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

27.05.2025
дата

В. Саф
підпис

Протокол аналізу звіту подібності науковим керівником

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

Автор: Сафонов Владислав Русланович

Співавтор:

Назва: Система аудиту та аналізу безпеки вебдодатків

Науковий керівник:

Підрозділ: Кафедра кібербезпеки

Коефіцієнт подібності 1:0.3%

Коефіцієнт подібності 2:0%

Мікропробіли: 8

Заміна букв: 0

Інтервали: 0

Білі знаки: 0

Дата створення звіту: 2025-05-28 18:20:15.0

Після аналізу Звіту подібності констатую наступне:

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

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

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

Обгрунтування:

29.05.2025р.

Anti-Plagiarism v-15.258 (global version)

The maximum coincidence with one document 0.0%

Dictionaries check: en_US, ru_RU, ua_UA. **Errors in the documents: 8%**

ID: 242313 Title: Система аудиту та аналізу безпеки вебдодатків Added in a DB: 2025-05-28 Authors: Сафонов Владислав Русланович Heads: Стецюк М.В. Consultants: Opponents:	Document		Sum coincidence on the DB	
	Symbols	Lexemes	Symbols	Lexemes
	89237	627	496 (1%)	6 (1%)

Plagiarism sources

ID	Description	Plagiarism presence in the document	
		Symbols	Lexemes

РІШЕННЯ ЕКСПЕРНОЇ КОМІСІЇ

КАФЕДРИ КІБЕРБЕЗПЕКИ

ПРО ДОПУСК КВАЛІФІКАЦІЙНОЇ РОБОТИ ДО ЗАХИСТУ

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

Назва: Система аудиту та аналізу безпеки вебдодатків

Автор: Сафонов Владислав Русланович

Спеціальність: 125 – Кібербезпека та захист інформації

Освітня програма: Кібербезпека та захист інформації

Науковий керівник: Микола СТЕЦЮК др. філософії, ст. викладач

Після аналізу звіту подібності зроблено такий висновок:

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

Підтвердження:

Оригінальність тексту роботи за результатами перевірки системою StrikePlagiarism складає 99.9%, оригінальність тексту роботи за результатами перевірки системою Anti-Plagiarism складає 99.9%.

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

Виявлені модифікації стосуються математичних формул і не є порушенням академічної доброчесності.

Керівник роботи _____ Микола СТЕЦЮК

Гарант ОП _____ Віктор ЧЕШУН

Завідувач кафедри кібербезпеки _____ Юрій КЛЬОЦ

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ХМЕЛЬНИЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

РЕЦЕНЗІЯ НА КВАЛІФІКАЦІЙНУ РОБОТУ

освітнього ступеня «бакалавр»

Студент Сафонов Владислав Русланович

Тема Система аудиту та анлізу безпеки вебдодатків

Галузь знань 12 – Інформаційні технології

Спеціальність 125 – Кібербезпека та захист інформації

Обсяг кваліфікаційної роботи освітнього ступеня «бакалавр»:

кількість листів креслень 3 ; кількість сторінок записки 69

1. Кваліфікаційну роботу присвячено розробці вебзастосунку для аудиту та аналізу безпеки вебресурсів, що спеціалізується на виявленні вразливостей типу SQL-ін'єкцій та XSS у формах авторизації. Користувач має змогу налаштувати процес сканування, обравши типи вразливостей, які потрібно перевірити. За підсумками аналізу формується захищений паролем звіт у форматі PDF, що забезпечує конфіденційність отриманих результатів.

2. Висновок про відповідність кваліфікаційної роботи завданню Кваліфікаційна робота відповідає поставленому завданню як в теоретичній, так і в практичній частині.

3. Характеристика виконання кожного розділу роботи, ступінь використання останніх досягнень науки і техніки і передових методів роботи: У роботі застосовано сучасні методи та здобутки в галузі аналізу безпеки вебзастосунків. Перший розділ присвячено огляду найпоширеніших вразливостей, аналізу наявних рішень і плануванню розробки власної системи. У другому розділі розглянуто етапи проектування — зокрема побудову UML-діаграм, створення користувацького інтерфейсу та безпосередню реалізацію функціоналу. Завершальний розділ описує процес тестування системи: від запуску на різних операційних системах до проведення автоматизованих перевірок.

4. Позитивні сторони роботи Робота добре структурована, чітко висвітлено усі етапи розробки системи від аналізу предметної області і закінчуючи тестуванням яке включає в себе ручні та автоматизовані методи перевірки.

5. Негативні сторони роботи Звіт системи недостатньо інформативний та кількість реалізованих методів перевірки на вразливості є обмеженою, що знижує загальну ефективність аналізу безпеки.

6. Оцінка графічного оформлення та пояснювальної записки роботи У цілому кваліфікаційна робота відзначається високим рівнем виконання. Матеріал викладено послідовно, ясно та структуровано, а зміст розділів добре відображає всі ключові етапи створення системи.

7. Відгук про роботу в цілому Кваліфікаційна робота присвячена розробці вебзастосунку для аналізу безпеки з виявленням SQL та XSS вразливостей. Робота добре структурована, охоплює всі етапи — від аналізу предметної області до тестування системи. Застосовано сучасні методи розробки, проте звітність потребує певного вдосконалення. Загалом, робота заслуговує на високу оцінку.

8. Інші зауваження

9. Оцінка дипломної роботи Розглянувши позитивні та негативні сторони представленої кваліфікаційної роботи, можна зробити висновок, що вона заслуговує оцінку «відмінно».

РЕЦЕНЗЕНТ (прізвище, ім'я, по батькові, посада, місце роботи)
Підченко Сергій Констянтинович, завідувач ТМІТ

«02» 06 2025 р. (підпис)