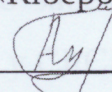


Міністерство освіти і науки України
Державний університет «Житомирська політехніка»
Факультет інформаційно-комп'ютерних технологій
Кафедра комп'ютерної інженерії та кібербезпеки

ПОЯСНЮВАЛЬНА ЗАПИСКА

до випускної кваліфікаційної роботи бакалавра
на тему: «Проект підсистеми безпеки веб-додатку на базі
FastAPI у середовищі Docker з використанням DevSecOps-
підходу»

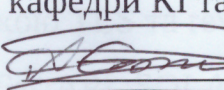
Виконав: студент 4-го курсу групи КБ-22-1,
спеціальності 125 «Кібербезпека»

 Андрій ЛАЩІЛІН

Керівник: професор кафедри КІ та КБ, д.т.н., доцент

 Володимир ВОРОТНІКОВ

Рецензент: зав. кафедри КІ та КБ, к.т.н, доцент

 Андрій ЄФІМЕНКО

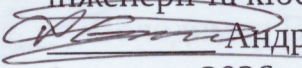
Житомир – 2026

Державний університет «Житомирська політехніка»
Факультет інформаційно-комп'ютерних технологій
Кафедра комп'ютерної інженерії та кібербезпеки
Спеціальність 125 «Кібербезпека»
Освітня програма «Кібербезпека»

ЗАТВЕРДЖУЮ

Завідувач

кафедри комп'ютерної
інженерії та кібербезпеки

 Андрій Єфіменко

2 лютого 2026 р.

ЗАВДАННЯ

на випускню кваліфікаційну роботу бакалавра

Студент: Лащілін Андрій Сергійович

Тема роботи: «Проект підсистеми безпеки веб-додатку на базі FastAPI у середовищі Docker з використанням DevSecOps-підходу»

затверджена Наказом університету від 2 лютого 2026 р. № 58/с

Термін здачі студентом закінченої роботи 10 червня 2026 р.

Вихідні дані роботи (зазначається предмет і об'єкт дослідження):

Об'єкт дослідження: процеси забезпечення безпеки веб-додатків у сучасних інформаційних системах.

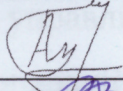
Предмет дослідження: методи та засоби комплексного забезпечення безпеки веб-додатку, включаючи захист на рівні мережі, серверної інфраструктури, прикладного програмного забезпечення, бази даних, а також інтеграцію засобів моніторингу та безпеки у CI/CD-процеси.

Обладнання та програмне забезпечення:

1. Сервер під керуванням ОС Linux.
2. Веб-сервер Nginx з підтримкою TLS та reverse proxy.
3. Інтерпретатор Python 3.13 (FastAPI, SQLAlchemy).
4. СУБД PostgreSQL 15.
5. Засоби контейнеризації Docker та Docker Compose.
6. Система контролю версій Git.
7. CI/CD-система GitLab CI.

Консультанти з випускної кваліфікаційної роботи

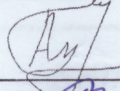
Розділ	Консультант	Підпис, дата	
		Завдання видав	Завдання прийняв
1	Покотило О.А.	02.02.2026	02.02.2026
2	Покотило О.А.	05.03.2026	05.03.2026
3	Покотило О.А.	12.04.2026	12.04.2026

Студент  Андрій ЛАЩІЛІН

Керівник  Володимир ВОРОТНІКОВ

Календарний план

№ з/п	Назви етапів кваліфікаційної роботи	Терміни	Примітка
1	Аналіз завдання та уточнення вимог до підсистеми безпеки	02.02.2026	Виконано
2	Аналіз літературних джерел та сучасних загроз (OWASP Top 10)	10.02.2026	Виконано
3	Дослідження механізмів захисту: JWT, bcrypt, ORM, TLS, Docker	20.02.2026	Виконано
4	Формування першого розділу (опис заходів безпеки)	05.03.2026	Виконано
5	Корегування першого розділу	15.03.2026	Виконано
6	Реалізація захисту: сервер, код, Nginx, PostgreSQL, Docker, CI/CD	10.04.2026	Виконано
7	Формування другого розділу (реалізація аспектів)	18.04.2026	Виконано
8	Налаштування моніторингу (Prometheus, Grafana, Loki)	25.04.2026	Виконано
9	Тестування безпеки: SQLi, CSRF, Brute-Force, IDOR	05.05.2026	Виконано
10	Формування третього розділу (спроби реалізації вразливостей)	15.05.2026	Виконано
11	Корегування третього розділу	25.05.2026	Виконано
12	Фінальні правки та оформлення пояснювальної записки	31.05.2026	Виконано

Студент  Андрій ЛАЩІЛІН

Керівник  Володимир ВОРОТНІКОВ

АНОТАЦІЯ

Кваліфікаційна робота бакалавра присвячена комплексному забезпеченню безпеки веб-додатку, розробленого з використанням сучасних серверних технологій. У межах роботи спроектовано та реалізовано захищений веб-додаток на базі фреймворку FastAPI та системи управління базами даних PostgreSQL із розгортанням на сервері під керуванням операційної системи Linux.

Пояснювальна записка містить 75 сторінок, 17 рисунків та 3 таблиці. Список використаних джерел налічує 20 найменувань і розміщений на 2 сторінках.

Метою випускної кваліфікаційної роботи є розробка та дослідження комплексної системи забезпечення безпеки веб-додатку з урахуванням захисту мережевого, серверного, прикладного рівнів, бази даних, а також інтеграції засобів автоматизованого моніторингу та контролю безпеки.

У роботі визначено основні загрози безпеці веб-додатків та вимоги до їх захисту. Проаналізовано сучасні методи та засоби забезпечення безпеки відповідно до рекомендацій OWASP. Розглянуто особливості реалізації механізмів аутентифікації та авторизації, захисту переданих даних і контролю доступу до бази даних. Показано розробку архітектури захищеного веб-додатку та інтеграцію засобів CI/CD, моніторингу та журналювання. Наведено результати тестування безпеки та аналіз ефективності впроваджених заходів захисту.

Ключові слова: ВЕБ-ДОДАТОК, ІНФОРМАЦІЙНА БЕЗПЕКА, FASTAPI, POSTGRESQL, CI/CD, МОНІТОРИНГ, GRAFANA, OWASP.

					KKIK.KP.B - 125 - 9 - ПЗ		
Змн.	Арк.	№ докум.	Підпис	Дата			
Розроб.		Лацілін А.С.		8.06.26	Літ.	Арк.	Архівів
Керівник		Воротніков В.В.		8.06.26		4	75
Реценз.		Єфіменко А.А.		8.06.26	ФІКТ, Гр. КБ-22-1		
Н. контр.		Воротніков В. В.		8.06.26			
Затверд.		Єфіменко А.А.		8.06.26			

ANNOTATION

The bachelor's qualification work is devoted to the comprehensive security of a web application developed using modern server technologies. As part of the work, a secure web application based on the FastAPI framework and the PostgreSQL database management system with deployment on a server running the Linux operating system was designed and implemented.

The explanatory note contains 72 pages, 17 figures and 3 tables. The list of sources used includes 20 names and is placed on 2 pages.

The purpose of the final qualification work is to develop and study a comprehensive system for ensuring the security of a web application, taking into account the protection of the network, server, application levels, database, as well as the integration of automated monitoring and security control tools.

The work identifies the main threats to the security of web applications and the requirements for their protection. Modern methods and means of ensuring security in accordance with OWASP recommendations are analyzed. The features of the implementation of authentication and authorization mechanisms, protection of transmitted data and control of access to the database are considered. The development of a secure web application architecture and integration of CI/CD, monitoring and logging tools are shown. The results of security testing and analysis of the effectiveness of the implemented security measures are presented.

Keywords: WEB APPLICATION, INFORMATION SECURITY, FASTAPI, POSTGRESQL, CI/CD, MONITORING, GRAFANA, OWASP.

		Лашілін А.С.			ЖИТОМИРСЬКА ПОЛІТЕХНІКА.26.125.9	Арк.
		Воротніков В. В				5
Змн.	Арк.	№ докум.	Підпис	Дата		

ЗМІСТ

ВСТУП.....	7
РОЗДІЛ 1 ОПИС ЗАХОДІВ БЕЗПЕКИ.....	9
1.1 Безпека Linux-сервера.....	9
1.2 Безпека коду веб-додатку.....	11
1.3 Захист від SQLi, XSS, CSRF.....	14
1.4 Nginx та безпека веб-сервера.....	17
1.5 PostgreSQL та безпека бази даних.....	19
1.6 Контейнеризація (Docker та Docker Compose).....	21
1.7 CI/CD та безпека розробки.....	23
1.8 Моніторинг та логування (Prometheus, Grafana, Loki).....	25
1.9 Захищені протоколи передачі даних (HTTPS, TLS).....	27
РОЗДІЛ 2 РЕАЛІЗАЦІЯ РОЗГЛЯНУТИХ АСПЕКТІВ.....	32
2.1 Захист сервера.....	32
2.2 Безпека коду.....	34
2.3 Nginx та безпека веб-сервера.....	38
2.4 Захист PostgreSQL.....	41
2.5 Docker та ізоляція сервісів.....	44
2.6 CI/CD та безпека розробки.....	47
2.7 Налаштування моніторингу та логування (Prometheus, Grafana, Loki).....	51
РОЗДІЛ 3 СПРОБИ РЕАЛІЗАЦІЇ ВРАЗЛИВОСТЕЙ.....	57
3.1 SQL-ін'єкції.....	57
3.2 CSRF (Cross-Site Request Forgery).....	59
3.3 Brute-Force.....	61
3.4 IDOR (Insecure direct object references).....	64
ВИСНОВКИ.....	67
СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ.....	69
ДОДАТКИ.....	70

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

ОС – операційна система

ПЗ – програмне забезпечення

ВЕБ – веб-додаток

БД – база даних

CI/CD – безперервна інтеграція та розгортання

TLS – Transport Layer Security

HTTPS – Hypertext Transfer Protocol Secure

RBAC – Role-Based Access Control

JWT – JSON Web Token

WAF – Web Application Firewall

API – Application Programming Interface

ORM – Object-Relational Mapping

SAST – Static Application Security Testing

DAST – Dynamic Application Security Testing

ELK – Elasticsearch, Logstash, Kibana

RLS – Row-Level Security

		Лашілін А.С.			ЖИТОМИРСЬКА ПОЛІТЕХНІКА.26.125.9	Арк.
		Воротніков В. В				7
Змн.	Арк.	№ докум.	Підпис	Дата		

ВСТУП

Актуальність теми. Сучасні веб-додатки широко використовуються у всіх сферах діяльності, що робить їх привабливими об'єктами для кібератак. Забезпечення комплексної безпеки веб-додатку є критично важливим завданням, оскільки вразливості можуть призвести до втрати даних, порушення конфіденційності користувачів та фінансових збитків. У межах даної роботи вирішується завдання створення захищеного веб-додатку та дослідження ефективності заходів безпеки на всіх рівнях – від серверної інфраструктури до протоколів передачі даних та додатку.

Метою кваліфікаційної роботи є розробка та дослідження комплексної системи забезпечення безпеки веб-додатку з урахуванням захисту серверної інфраструктури, протоколів передачі даних, прикладного рівня та бази даних, а також інтеграції засобів моніторингу та контролю безпеки.

Визначена мета обумовлює наступні завдання:

- аналіз сучасних загроз та вразливостей веб-додатків;
- дослідження та впровадження засобів захисту серверної інфраструктури та бази даних;
- забезпечення безпеки протоколів передачі даних (HTTPS, TLS) та веб-API;
- розробка механізмів аутентифікації та авторизації користувачів;
- інтеграція інструментів моніторингу, журналювання та CI/CD для контролю безпеки;
- тестування ефективності впроваджених заходів безпеки.

Об'єктом дослідження є процес забезпечення безпеки веб-додатків у сучасних інформаційних системах.

Предметом дослідження є методи, засоби та технології комплексного захисту веб-додатку, включаючи безпеку серверного середовища, протоколів передачі даних, бази даних, додатку та інтеграцію інструментів моніторингу і CI/CD.

У процесі виконання кваліфікаційної роботи було розроблено проект захищеного веб-додатку на базі FastAPI та PostgreSQL, що демонструє застосування заходів безпеки на всіх рівнях – від серверної інфраструктури до протоколів та прикладного коду.

		Лашілін А.С.			ЖИТОМИРСЬКА ПОЛІТЕХНІКА.26.125.9	Арк.
		Воротніков В. В				8
Змн.	Арк.	№ докум.	Підпис	Дата		

РОЗДІЛ 1 ОПИС ЗАХОДІВ БЕЗПЕКИ

Сучасний веб-додаток являє собою багаторівневу інформаційну систему, яка включає клієнтський інтерфейс, API, серверну частину, базу даних, контейнеризоване середовище виконання, засоби автоматизації розгортання та системи моніторингу. Кожен із цих компонентів може виступати потенційною точкою входу для зловмисника та потребує окремих механізмів захисту.

Основними загрозами для веб-додатків є несанкціонований доступ до ресурсів системи, SQL-ін'єкції, XSS та CSRF-атаки, brute-force атаки на механізми автентифікації, компрометація облікових даних і секретів, а також мережеві атаки на серверну інфраструктуру та API. Крім цього, сучасні підходи до розробки із використанням Docker та CI/CD створюють додаткові ризики, пов'язані з безпекою контейнерів, залежностей, конфігурацій та процесів автоматичного розгортання.

У зв'язку з цим забезпечення безпеки веб-додатку повинно реалізовуватись комплексно та охоплювати всі рівні системи: операційну систему сервера, reverse проху та мережеву взаємодію, прикладний код, базу даних, контейнеризоване середовище, процеси CI/CD, а також системи моніторингу та журналювання подій. Саме тому у даному розділі послідовно розглядаються основні механізми захисту кожного з перелічених компонентів веб-додатку, обґрунтовується вибір конкретних технологій та інструментів (FastAPI, JWT, Nginx, Docker, Grafana/Loki), а також проводиться порівняння з альтернативними підходами.

1.1 Безпека Linux-сервера

Безпека операційної системи Linux є однією з ключових складових комплексного захисту веб-додатків. Linux широко використовується для розгортання серверів завдяки своїй стабільності, гнучкості та відкритому коду, що дозволяє адміністраторам детально контролювати налаштування системи. Однак наявність великої кількості сервісів і відкритих портів створює потенційні вектори атак, які можуть бути використані зловмисниками для несанкціонованого доступу до даних або ресурсів сервера. Недостатньо захищена серверна інфраструктура може стати причиною компрометації

		Лашілін А.С.			ЖИТОМИРСЬКА ПОЛІТЕХНІКА.26.125.9	Арк.
		Воротніков В. В				9
Змн.	Арк.	№ докум.	Підпис	Дата		

веб-додатку, що особливо критично у випадках, коли сервер обробляє конфіденційну інформацію користувачів, фінансові дані або персональні дані.

Одним із базових принципів безпеки Linux є принцип найменших привілеїв (Least Privilege) [1]. Він полягає в тому, що кожен користувач і процес системи повинні мати лише ті права, які необхідні для виконання своїх функцій. Використання окремих облікових записів для різних сервісів, обмеження доступу до системних каталогів і файлів, а також контроль за використанням прав адміністратора (root) значно знижує ризик компрометації сервера. Крім того, важливо забезпечити правильну конфігурацію системних груп і прав доступу до виконуваних файлів, щоб унеможливити виконання несанкціонованих команд.

Ще одним важливим аспектом є регулярне оновлення системи та програмного забезпечення. Застосування оновлень і патчів дозволяє усувати відомі вразливості, які з'являються у зв'язку з розвитком програмних продуктів та виявленням нових видів атак. В сучасних умовах адміністрування серверів передбачає автоматизацію процесу оновлення ядра Linux та встановлених пакетів, що дозволяє оперативно закривати критичні дірки без ручного втручання адміністратора [4]. Недотримання цього правила може призвести до того, що відомі експлойти стануть доступними для злоумисників, що створює високий рівень ризику для безпеки веб-додатку.

Серверна безпека також включає захист мережевих з'єднань та сервісів. У Linux для цього використовуються брандмауери (iptables, ufw, nftables), які дозволяють контролювати трафік, обмежувати доступ до необхідних портів і блокувати підозрілі IP-адреси [2]. Важливою практикою є також обмеження віддаленого доступу до сервера, наприклад, шляхом конфігурації SSH для роботи тільки з ключами, відключення root-доступу та обмеження IP-адрес, з яких дозволено підключення. Ці заходи суттєво знижують ймовірність компрометації сервера через віддалені атаки [1, 8].

Ще один рівень захисту забезпечується системами виявлення та реагування на підозрілу активність, такими як Fail2ban. Вони дозволяють автоматично блокувати IP-адреси після декількох невдалих спроб входу або при виявленні підозрілих дій. Крім того, важливо впроваджувати системи моніторингу та журналювання, які фіксують події безпеки, дії користувачів і процесів [17, 18]. Аналіз логів дозволяє своєчасно виявляти

		Лашілін А.С.			ЖИТОМИРСЬКА ПОЛІТЕХНІКА.26.125.9	Арк.
		Воротніков В. В				10
Змн.	Арк.	№ докум.	Підпис	Дата		

аномалії, що можуть сигналізувати про спроби вторгнення або неправомірне використання ресурсів сервера.

Крім традиційних методів безпеки, практичною складовою є хардінг системи. Він включає видалення непотрібних сервісів, закриття незадіяних портів, налаштування безпечних конфігурацій ядра та служб, а також обмеження прав виконання скриптів і програм [3, 4]. Хардінг значно зменшує площу атаки та робить систему менш привабливою для зловмисників [5, 6].

Комплексний підхід до безпеки Linux-сервера передбачає не лише технічні засоби, але й належну організаційну практику [4]. Це включає розробку політик доступу, регулярний аудит системи, контроль змін у конфігурації та навчання адміністративного персоналу принципам кібербезпеки. Такий підхід дозволяє створити стійку до атак серверну інфраструктуру, яка є надійною основою для функціонування веб-додатку.

У підсумку, безпека Linux-сервера є фундаментальним елементом захисту веб-додатку. Тільки за умови впровадження комплексних заходів безпеки на рівні операційної системи можна забезпечити надійну роботу додатку, захист конфіденційних даних користувачів і стабільність функціонування всієї інформаційної системи [1, 6]. Важливо підкреслити, що безпека сервера повинна розглядатися як безперервний процес, який включає моніторинг, оновлення та вдосконалення заходів захисту у відповідь на нові загрози [4, 18].

1.2 Безпека коду веб-додатку

Безпека коду веб-додатку є критичною складовою загальної системи захисту інформаційних ресурсів. Навіть у випадку надійно захищеного сервера та мережевих протоколів, вразливості у коді можуть стати джерелом серйозних порушень конфіденційності, доступності та цілісності даних. Основна мета захисту коду полягає в мінімізації можливостей зловмисника впливати на роботу додатку, отримувати доступ до приватної інформації користувачів або порушувати логіку бізнес-процесів, які він реалізує.

Одним із ключових аспектів безпеки є аутентифікація та авторизація користувачів. Аутентифікація забезпечує перевірку достовірності користувача, а авторизація визначає

		Лашілін А.С.			ЖИТОМИРСЬКА ПОЛІТЕХНІКА.26.125.9	Арк.
		Воротніков В.В				11
Змн.	Арк.	№ докум.	Підпис	Дата		



Header

```
base64enc({
  "alg": "HS256",
  "typ": "JWT"
})
```

Payload

```
base64enc({
  "iss": "toptal.com",
  "exp": 1426420800,
  "company": "Toptal",
  "awesome": true
})
```

Signature

```
HMACSHA256(
  base64enc(header)
  + '.' +
  base64enc(payload)
  , secretKey)
```

Ще одним критичним аспектом є захист збережених даних, особливо паролів користувачів та конфіденційної інформації. Паролі повинні зберігатися лише у хешованому вигляді із застосуванням сучасних алгоритмів хешування, таких як Argon2 або bcrypt (рис. 1.2) [1]. Додатково рекомендується використання «солі» для унеможливлення атак типу rainbow table. Недотримання цих правил може призвести до швидкого розкриття паролів при несанкціонованому доступі до бази даних.



		Лашілін А.С.			ЖИТОМИРСЬКА ПОЛІТЕХНІКА.26.125.9	Арк.
		Воротніков В.В				12
Змн.	Арк.	№ докум.	Підпис	Дата		

Критично важливим є захист від найпоширеніших вразливостей веб-додатків, визначених у OWASP Top 10 [5]. До них належать SQL-ін'єкції, Cross-Site Scripting (XSS), Cross-Site Request Forgery (CSRF) та інші типи атак. Для запобігання SQL-ін'єкціям застосовується об'єктно-реляційне відображення (ORM) [11] та підготовлені запити, що виключають пряме впровадження користувацьких даних у SQL-код [10]. Для запобігання XSS та CSRF атакам реалізуються механізми валідації та очищення вхідних даних [6], а також генерація токенів безпеки для захисту запитів (рис. 1.3).

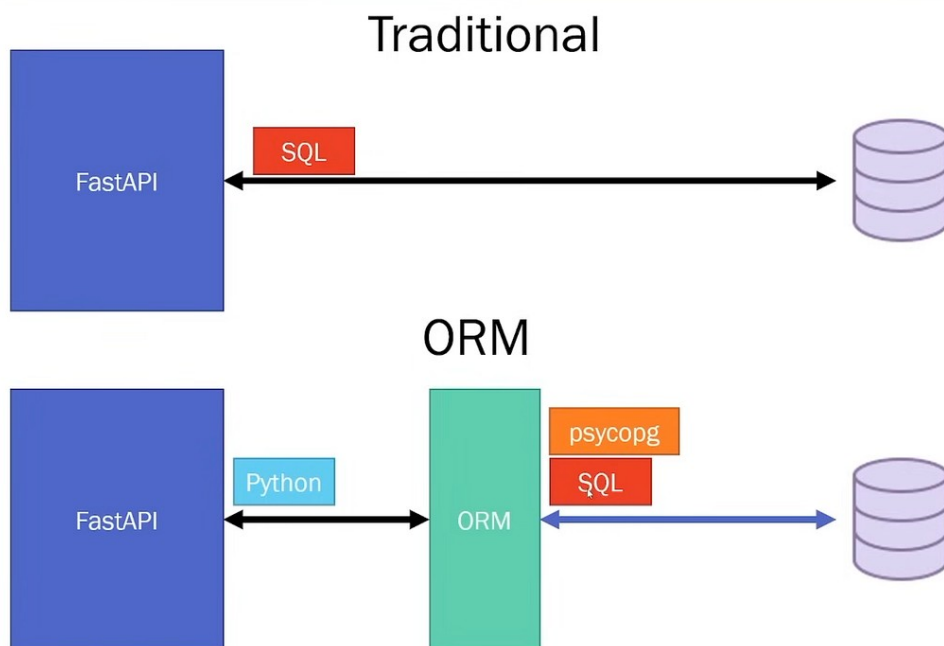


Рисунок 1.3 – Відмінності традиційного і ORM підходів

Крім того, безпека коду включає правильну обробку винятків та помилок. Неправильна обробка може призводити до витоку інформації про внутрішню структуру додатку або бази даних, що спрощує роботу зломисникам. Логування помилок має здійснюватися у захищеному середовищі, а повідомлення користувачу не повинні містити технічних деталей.

Ще однією важливою складовою є регулярне оновлення залежностей та бібліотек. Використання застарілих пакетів створює ризик появи відомих вразливостей, які можуть бути швидко використані у атаках. Інтеграція інструментів для перевірки безпеки залежностей, таких як `pip-audit`, дозволяє автоматично виявляти та усувати потенційні проблеми ще на етапі розробки та безперервної інтеграції (CI/CD) [16].

Комплексний підхід до безпеки коду веб-додатку передбачає не лише технічні заходи, але й дотримання правил безпечного програмування та архітектурного проектування. Це включає чітке розділення функціональних модулів, контроль доступу на рівні об'єктів [6], застосування принципів мінімізації поверхні атаки та регулярне проведення тестування й аудиту коду [20].

У результаті, дотримання всіх зазначених заходів забезпечує високий рівень безпеки веб-додатку, знижує ймовірність компрометації даних та забезпечує надійну роботу сервісу у різних умовах експлуатації. Безпека коду виступає фундаментом для захисту всього додатку, оскільки навіть надійно захищений сервер та мережеві протоколи не зможуть компенсувати наявність критичних помилок у самому коді.

1.3 Захист від SQLi, XSS, CSRF

Однією з найпоширеніших категорій вразливостей веб-додатків є атаки типу SQL-ін'єкцій (SQLi), Cross-Site Scripting (XSS) та Cross-Site Request Forgery (CSRF) [5]. Ці атаки дозволяють зловмисникам отримати несанкціонований доступ до даних, змінювати інформацію, виконувати довільні команди на сервері або від імені користувача виконувати небезпечні дії (рис. 1.4).

SQL-ін'єкції виникають, коли користувацькі дані безпосередньо вставляються в SQL-запити без належної обробки. В результаті зловмисник може модифікувати логіку запити, отримати доступ до конфіденційної інформації або навіть змінити дані в БД [5]. Однак сучасні веб-фреймворки, такі як FastAPI [9] у поєднанні з ORM-бібліотеками (наприклад, SQLAlchemy) [11], автоматично використовують підготовлені запити (prepared statements) та параметризовані запити. Це означає, що значення змінних відокремлюються від самої SQL-команди, що практично виключає можливість SQL-ін'єкції навіть без додаткових налаштувань.

		Лашілін А.С.			ЖИТОМИРСЬКА ПОЛІТЕХНІКА.26.125.9	Арк.
		Воротніков В. В				14
Змн.	Арк.	№ докум.	Підпис	Дата		

SQL Injection

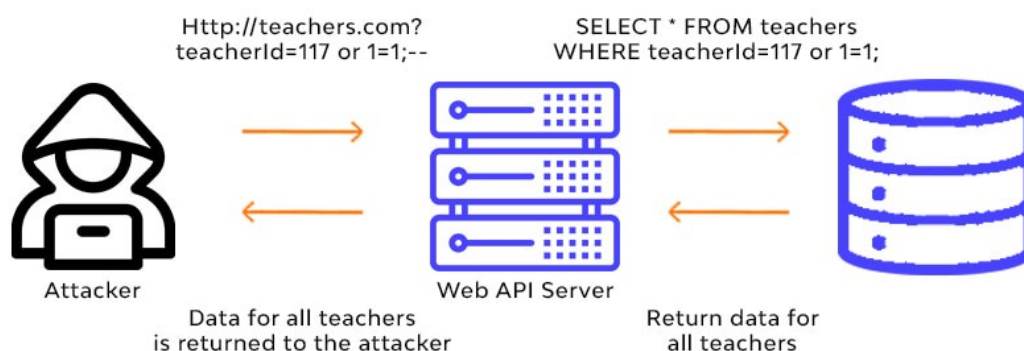


Рисунок 1.4 – Принцип SQL-ін'єкції

Cross-Site Scripting (XSS) є атакою, коли зловмисник вставляє шкідливий скрипт у веб-сторінку, що відображається іншому користувачу (рис. 1.5) [5]. Сучасні веб-фреймворки автоматично екранують або очищають вхідні дані перед відображенням на сторінках, що значно знижує ризик XSS [6]. Наприклад, FastAPI та інші фреймворки, що використовують шаблонізатори типу Jinja2, мають вбудовані механізми ескейпінгу HTML, що перетворюють спеціальні символи у безпечні представлення [9].

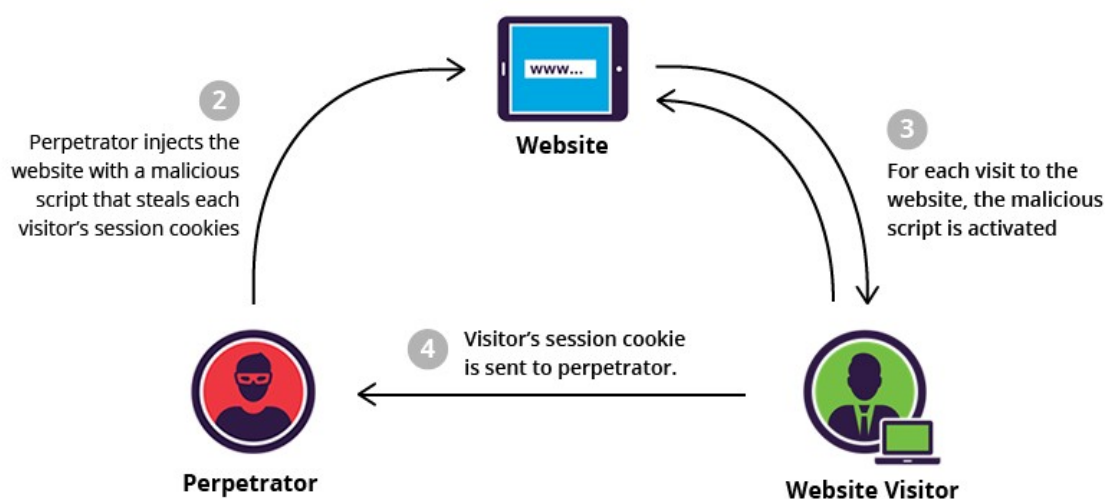


Рисунок 1.5 – Принцип атаки Cross-Site Scripting

Атака CSRF полягає у змушуванні авторизованого користувача виконати небажані дії на сервері (рис. 1.6) [5]. Сучасні фреймворки пропонують вбудовані механізми захисту від CSRF, такі як токени, що перевіряються на сервері під час обробки запитів, або автоматична перевірка заголовків “Origin” і “Referer” [2, 6]. Це дозволяє блокувати запити, які не походять з довіреного джерела, і практично виключає можливість виконання небажаних дій.

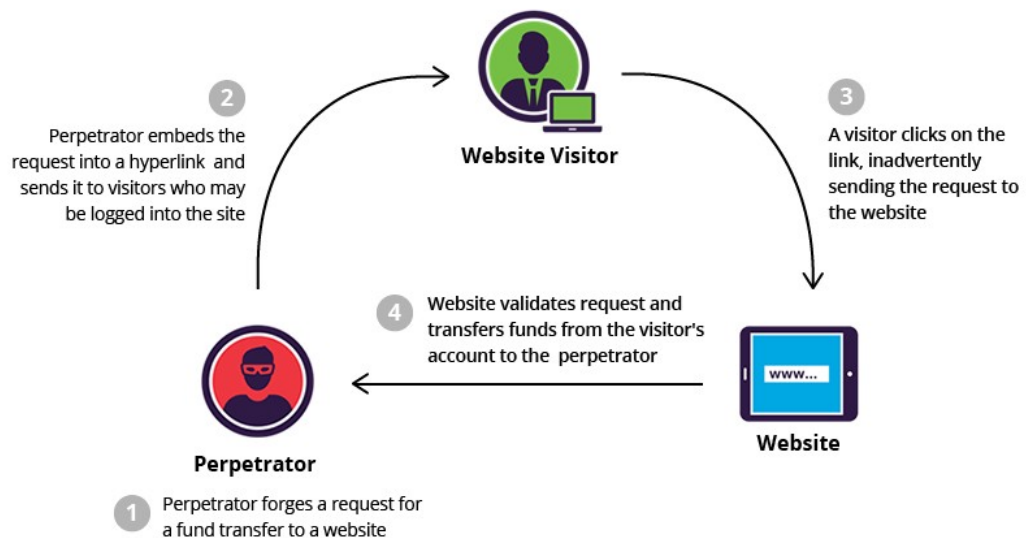


Рисунок 1.6 – Принцип атаки CSRF

Важливо розуміти, що хоча сучасні фреймворки забезпечують базовий захист від SQLi, XSS та CSRF, безпечний дизайн додатку та правильне використання механізмів валідації і обробки даних залишаються критично важливими. Розробник повинен усвідомлювати потенційні вразливості та додатково контролювати обробку даних, особливо при роботі з необробленими зовнішніми джерелами або нестандартними запитами.

У підсумку, застосування сучасних фреймворків у поєднанні з ORM та стандартними механізмами ескейпінгу та CSRF-токенів дозволяє значно підвищити безпеку веб-додатку і мінімізувати ризики від класичних атак на код, залишаючи можливість зосередитися на захисті бізнес-логіки та унікальних функціональних модулів.

1.4 Nginx та безпека веб-сервера

Веб-сервер є одним із ключових елементів інфраструктури веб-додатку, оскільки саме він виступає точкою входу для зовнішніх користувачів і обробляє всі HTTP- та HTTPS-запити. У сучасних архітектурах Nginx широко використовується як зворотний проксі-сервер для взаємодії між клієнтами та серверними застосунками, зокрема додатками, реалізованими за допомогою FastAPI [9, 12]. Правильна конфігурація та захист веб-сервера є критично важливими для забезпечення загальної безпеки системи.

Однією з основних функцій Nginx у контексті безпеки є ізоляція веб-додатку від прямого доступу з мережі Інтернет [3]. FastAPI зазвичай запускається на внутрішньому порту сервера і не приймає запити безпосередньо від користувачів [9]. Усі зовнішні HTTP-запити обробляються Nginx, який передає їх додатку вже у контрольованому вигляді [12]. Такий підхід дозволяє зменшити площу атаки та приховати внутрішню архітектуру серверного застосунку.

Важливою складовою захисту є реалізація захищеного з'єднання за допомогою HTTPS. Nginx підтримує сучасні криптографічні протоколи та алгоритми, що дозволяє забезпечити конфіденційність і цілісність переданих даних [12]. Налаштування TLS-сертифікатів, використання актуальних версій протоколу та відключення застарілих алгоритмів шифрування знижує ризик атак типу перехоплення трафіку або його модифікації [1, 8]. Таким чином, веб-сервер виступає ключовою ланкою в реалізації транспортної безпеки веб-додатку.

Nginx також виконує роль першого рівня захисту від мережесих атак, зокрема від надмірної кількості запитів або спроб перевантаження сервера [12]. Механізми обмеження швидкості запитів та кількості одночасних з'єднань дозволяють зменшити ефективність атак типу DoS і DDoS [2]. Крім того, веб-сервер може блокувати доступ до певних ресурсів або URL-шляхів, що не повинні бути доступними ззовні, наприклад службових або адміністративних інтерфейсів (рис. 1.7).

Ще одним важливим аспектом є захист конфігураційних файлів і статичних ресурсів. Nginx дозволяє обмежити доступ до внутрішніх файлів, виключити можливість їх завантаження або перегляду, а також заборонити виконання небезпечних типів файлів.

		Лашілін А.С.			ЖИТОМИРСЬКА ПОЛІТЕХНІКА.26.125.9	Арк.
		Воротніков В. В				17
Змн.	Арк.	№ докум.	Підпис	Дата		

Це запобігає витоку конфігураційної інформації та унеможливлює виконання шкідливого коду на сервері через веб-інтерфейс.

NGINX

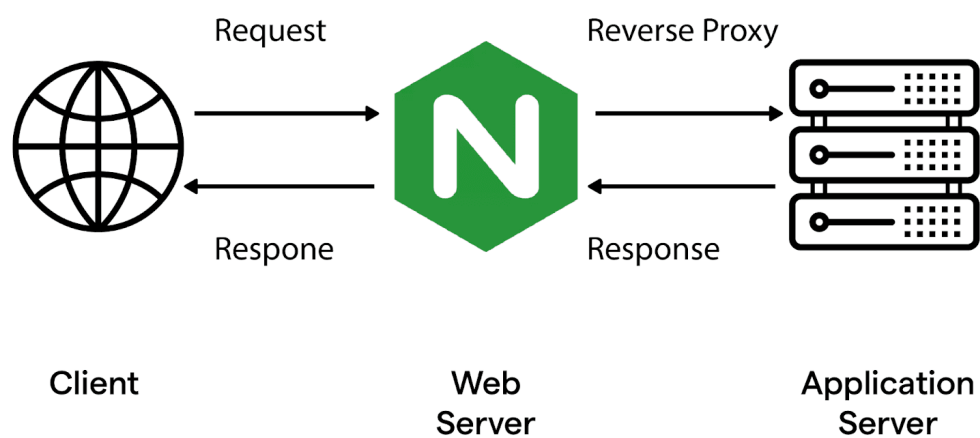


Рисунок 1.7 – Приклад роботи проксі

Значну роль у забезпеченні безпеки відіграє журналювання та аналіз запитів. Nginx фіксує всі вхідні запити та помилки, що дозволяє проводити подальший аналіз активності користувачів і виявляти підозрілі дії [12]. У поєднанні з централізованими системами збору логів, такими як Grafana Loki, журнали веб-сервера стають важливим джерелом інформації для моніторингу та реагування на інциденти безпеки.

Крім того, Nginx може використовуватися як базовий інструмент для реалізації політик безпеки на рівні HTTP-заголовків [12]. Налаштування таких механізмів дозволяє зменшити ризик атак на клієнтські частини додатку, зокрема XSS або підміни контенту [5, 6]. Таким чином, веб-сервер виконує не лише роль маршрутизатора запитів, але й активного учасника системи захисту веб-додатку.

У підсумку, Nginx є потужним та гнучким інструментом для забезпечення безпеки веб-додатку. Його правильне використання дозволяє значно підвищити рівень захисту

		Лашілін А.С.			ЖИТОМИРСЬКА ПОЛІТЕХНІКА.26.125.9	Арк.
		Воротніков В. В				18
Змн.	Арк.	№ докум.	Підпис	Дата		

серверної частини, реалізувати контроль доступу, забезпечити захищене з'єднання та створити додатковий бар'єр між зовнішнім середом і внутрішньою логікою додатку. У комплексі з іншими заходами безпеки Nginx виступає важливою складовою багаторівневої системи захисту веб-додатку.

1.5 PostgreSQL та безпека бази даних

База даних є одним із найкритичніших компонентів веб-додатку, оскільки саме в ній зберігається основна частина інформації, включаючи облікові дані користувачів, персональні відомості, журнали подій та інші чутливі дані. У контексті комплексного забезпечення безпеки веб-додатку захист бази даних має першочергове значення, оскільки компрометація цього рівня може призвести до повної втрати конфіденційності та цілісності інформації [5, 6]. PostgreSQL є сучасною реляційною системою управління базами даних, яка надає широкий набір вбудованих механізмів безпеки та контролю доступу.

Одним із базових принципів безпеки PostgreSQL є чітке розмежування доступу користувачів. Кожен додаток або сервіс повинен працювати під окремим обліковим записом бази даних з обмеженими правами. Такий підхід дозволяє реалізувати принцип найменших привілеїв і зменшити ризик пошкодження або витоку даних у разі компрометації облікових даних додатку. Важливо, щоб сервер бази даних не використовував обліковий запис з адміністративними правами для повсякденної роботи веб-додатку.

PostgreSQL підтримує аутентифікацію та шифрування з'єднань, що дозволяє захистити дані під час передачі між додатком і сервером бази даних [10]. Використання захищених каналів зв'язку (наприклад, SSL/TLS) унеможливлює перехоплення облікових даних або інформації під час передачі мережею [2, 8]. Крім того, система підтримує різні методи автентифікації, що дозволяє інтегрувати базу даних у загальну політику безпеки серверної інфраструктури.

Важливу роль у захисті інформації відіграє контроль доступу на рівні об'єктів та рядків. PostgreSQL надає можливість обмежувати доступ не лише до таблиць або схем, але й до окремих рядків за допомогою механізму Row-Level Security (RLS) [10]. Це

		Лашілін А.С.			ЖИТОМИРСЬКА ПОЛІТЕХНІКА.26.125.9	Арк.
		Воротніков В. В				19
Змн.	Арк.	№ докум.	Підпис	Дата		

дозволяє реалізувати гнучкі політики доступу, коли користувачі або сервіси мають доступ лише до тієї частини даних, яка їм дозволена згідно з бізнес-логікою додатку [6]. Такий підхід значно підвищує рівень безпеки при роботі з багатокористувацькими системами.

Не менш важливим аспектом є захист від несанкціонованих змін та втрати даних. PostgreSQL підтримує механізми журналювання транзакцій, резервного копіювання та відновлення даних, що дозволяє зберегти цілісність інформації навіть у випадку помилок або збоїв [10]. Регулярне створення резервних копій та перевірка їх працездатності є важливою складовою загальної стратегії безпеки, оскільки дозволяє швидко відновити систему після інциденту без втрати критичних даних.

Значну роль у забезпеченні безпеки відіграє моніторинг та аудит доступу до бази даних. PostgreSQL дозволяє вести детальне журналювання операцій, що надає можливість аналізувати активність користувачів і додатків, виявляти підозрілі запити та спроби несанкціонованого доступу [10]. У поєднанні з централізованими системами логування (такими як Prometheus та Grafana Loki) ці дані можуть бути використані для виявлення інцидентів безпеки та реагування на них у реальному часі.

Окрему увагу слід приділяти оновленню системи управління базами даних. Як і будь-яке складне програмне забезпечення, PostgreSQL може містити вразливості, які з часом виявляються та усуваються розробниками [10]. Регулярне встановлення оновлень та патчів дозволяє мінімізувати ризик експлуатації відомих вразливостей та підтримувати актуальний рівень безпеки (рис. 1.8).

		Лашілін А.С.			ЖИТОМИРСЬКА ПОЛІТЕХНІКА.26.125.9	Арк.
		Воротніков В. В				20
Змн.	Арк.	№ докум.	Підпис	Дата		

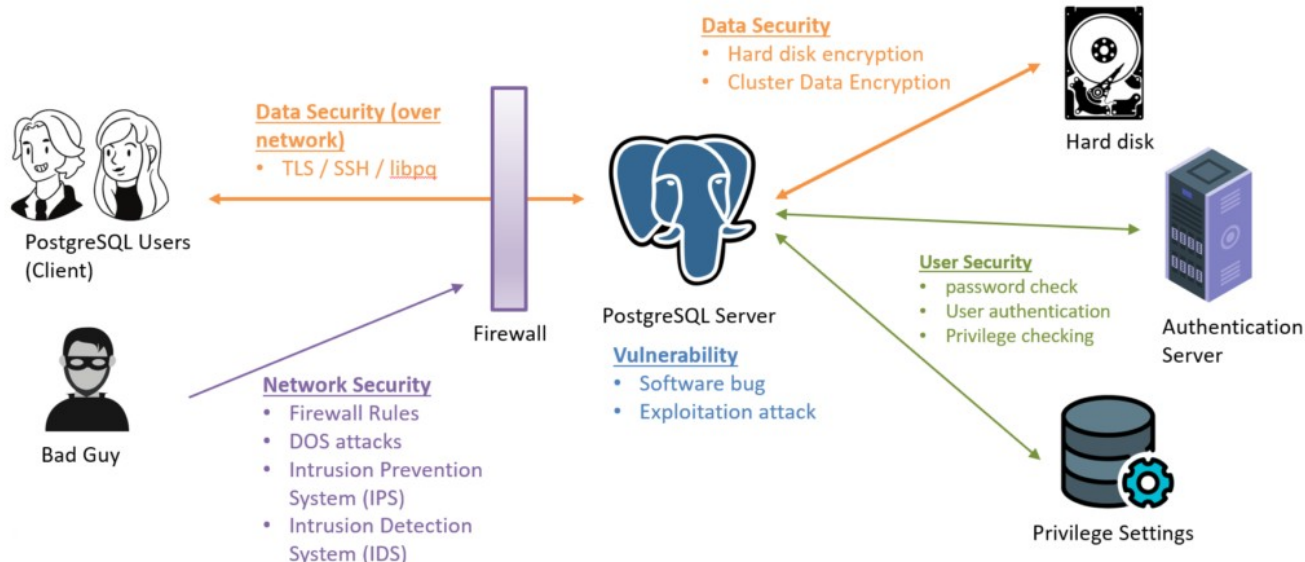


Рисунок 1.8 – Узагальнена схема вразливостей бази даних

У підсумку, PostgreSQL надає широкий набір інструментів для забезпечення безпеки бази даних, однак їх ефективність безпосередньо залежить від правильної конфігурації та інтеграції у загальну архітектуру веб-додатку. Комплексний підхід до захисту бази даних, який включає контроль доступу, шифрування з'єднань, аудит, резервне копіювання та оновлення, дозволяє значно підвищити рівень безпеки всієї системи та забезпечити надійне зберігання інформації.

1.6 Контейнеризація (Docker та Docker Compose)

Контейнеризація є сучасним підходом до розгортання та експлуатації веб-додатків, який дозволяє ізолювати програмні компоненти та забезпечити їх стабільну роботу незалежно від середовища виконання. Docker є одним із найпоширеніших інструментів контейнеризації, що широко використовується у серверних інфраструктурах для спрощення процесів розгортання, масштабування та обслуговування додатків [14]. Docker Compose, у свою чергу, є інструментом оркестрації, що дозволяє декларативно описати багатоконтейнерну інфраструктуру в єдиному конфігураційному файлі та керувати нею як єдиним цілим [15]. У контексті комплексного забезпечення безпеки веб-додатку контейнеризація відіграє важливу роль, оскільки дозволяє створювати додатковий рівень ізоляції між сервісами та операційною системою.

Однією з ключових переваг контейнеризації з точки зору безпеки є ізоляція процесів. Кожен контейнер виконується у власному середовищі з обмеженим доступом до ресурсів хост-системи [14]. Це означає, що у разі компрометації одного контейнера зломисник не отримує прямого доступу до всієї системи або інших сервісів. Таким чином, контейнеризація дозволяє зменшити наслідки потенційних атак та локалізувати інциденти безпеки.

Docker також сприяє контролю залежностей та середовища виконання [4]. Усі бібліотеки, конфігурації та версії програмного забезпечення фіксуються у Docker-образі, що забезпечує передбачуваність роботи додатку [14]. Це знижує ризик появи вразливостей, пов'язаних із несумісністю бібліотек або використанням застарілих компонентів у різних середовищах [5, 6]. З точки зору безпеки це дозволяє швидше виявляти та усувати проблеми, оскільки середовище є повністю відтворюваним.

Важливим аспектом є обмеження прав контейнерів. Контейнери не повинні виконуватися з правами адміністратора (root), а доступ до файлової системи та мережі має бути строго контрольованим [14]. Правильна конфігурація Docker дозволяє обмежити можливості контейнерів взаємодіяти з хост-системою, що суттєво підвищує загальний рівень захисту серверної інфраструктури.

Docker Compose забезпечує додатковий рівень безпеки через декларативне управління інфраструктурою. Увесь склад сервісів, їхні залежності, мережеві з'єднання та томи описуються у файлі docker-compose.yml, що робить конфігурацію прозорою та придатною для аудиту. Директива depends_on гарантує коректний порядок запуску сервісів, а механізм env_file дозволяє передавати конфіденційні параметри через зовнішній файл змінних середовища, не зберігаючи секрети безпосередньо у конфігураційному файлі чи Docker-образі [15]. Це є критично важливим при публікації коду у системах контролю версій.

Контейнеризація також спрощує оновлення та усунення вразливостей [4]. У разі виявлення проблеми безпеки створюється новий образ із виправленнями, який замінює попередню версію без втручання у хост-систему [14]. Такий підхід зменшує час простою та дозволяє швидко реагувати на інциденти. Крім того, Docker-образи можуть

		Лашілін А.С.			ЖИТОМИРСЬКА ПОЛІТЕХНІКА.26.125.9	Арк.
		Воротніков В. В				
Змн.	Арк.	№ докум.	Підпис	Дата		22

перевірятися на наявність відомих вразливостей за допомогою спеціалізованих інструментів, що підвищує рівень контролю безпеки ще до розгортання додатку.

Окрему увагу слід приділити мережевій безпеці контейнерів. Docker Compose автоматично створює окрему внутрішню мережу для всіх описаних сервісів, у межах якої вони взаємодіють між собою за DNS-іменами контейнерів [15]. Ця мережа є недоступною ззовні, а назовні публікуються лише явно визначені порти. Наприклад, база даних може бути доступна лише для сервісу додатку всередині мережі Docker, тоді як ззовні вона залишається повністю ізольованою. Такий підхід відповідає принципу мінімізації поверхні атаки та знижує ризик несанкціонованого доступу.

З точки зору архітектури, контейнеризація сприяє побудові багаторівневої системи безпеки, де кожен компонент має чітко визначену роль та обмежені права [3]. Docker Compose дозволяє також керувати постійним зберіганням даних через іменовані томи, відокремлюючи стан системи від самих контейнерів — це забезпечує збереження даних між перезапусками та спрощує резервне копіювання.

У підсумку, спільне використання Docker та Docker Compose у процесі розгортання веб-додатку підвищує рівень безпеки за рахунок ізоляції сервісів, декларативного управління інфраструктурою, захисту секретів через змінні середовища та контрольованої мережевої взаємодії між компонентами. Контейнеризація не є заміною традиційним механізмам захисту, однак у комплексі з іншими заходами виступає надійним базисом для захисту системи.

1.7 CI/CD та безпека розробки

У сучасній розробці веб-додатків значну роль відіграють процеси безперервної інтеграції та безперервного розгортання (CI/CD), які дозволяють автоматизувати етапи тестування, перевірки та впровадження програмного забезпечення (рис. 1.9) [4]. Використання CI/CD сприяє підвищенню якості коду, зменшенню кількості помилок та пришвидшенню процесу доставки оновлень. У контексті комплексного забезпечення безпеки веб-додатку CI/CD-процеси виступають важливим інструментом впровадження безпеки на ранніх етапах життєвого циклу розробки [16].

		Лашілін А.С.			ЖИТОМИРСЬКА ПОЛІТЕХНІКА.26.125.9	Арк.
		Воротніков В. В				23
Змн.	Арк.	№ докум.	Підпис	Дата		

Однією з ключових переваг CI/CD є можливість автоматичної перевірки коду на наявність помилок та вразливостей ще до його розгортання у продуктивному середовищі [4]. Інтеграція засобів статичного аналізу безпеки коду (SAST) та автоматизованого тестування дозволяє виявляти потенційні проблеми, пов'язані з небезпечними конструкціями, некоректною обробкою даних або неправильним використанням бібліотек [6, 20]. Таким чином, більшість критичних вразливостей можуть бути усунені ще на етапі розробки, що значно знижує ризики експлуатації додатку.

CI/CD також відіграє важливу роль у контролі залежностей та сторонніх бібліотек. Сучасні веб-додатки активно використовують зовнішні пакети, які можуть містити відомі вразливості [5]. Автоматичні перевірки залежностей (Dependency Scanning) у межах інструментів автоматизації дозволяють виявляти застарілі або небезпечні версії бібліотек і своєчасно оновлювати їх [16]. Це сприяє підтримці актуального рівня безпеки без необхідності ручного контролю з боку розробника.

Окрему увагу в CI/CD слід приділяти безпеці процесу розгортання [4]. Автоматизоване розгортання зменшує ймовірність людської помилки та унеможливорює несанкціоновані зміни у конфігурації системи [3]. Використання контрольованих середовищ, чітко визначених етапів деплою та обмеження доступу до конфігураційних файлів дозволяє забезпечити цілісність додатку під час оновлень.

Важливим аспектом безпеки розробки є захист облікових даних та секретів, які використовуються у процесі CI/CD. Паролі, токени доступу та ключі шифрування не повинні зберігатися безпосередньо у коді або репозиторіях [5, 6]. Натомість вони передаються через захищені змінні середовища (CI/CD Variables) або спеціалізовані сховища секретів [16]. Такий підхід дозволяє запобігти витоків чутливої інформації навіть у випадку компрометації репозиторію.

CI/CD сприяє впровадженню концепції DevSecOps, яка передбачає інтеграцію безпеки у всі етапи розробки та експлуатації програмного забезпечення [4]. У межах цього підходу безпека розглядається не як окремий етап, а як постійний процес, що супроводжує розробку від написання коду до його експлуатації. Це дозволяє підвищити загальний рівень захисту веб-додатку та забезпечити швидку реакцію на нові загрози.

		Лашілін А.С.			ЖИТОМИРСЬКА ПОЛІТЕХНІКА.26.125.9	Арк.
		Воротніков В.В				24
Змн.	Арк.	№ докум.	Підпис	Дата		



Рисунок 1.9 – Етапи розгортання

У підсумку, використання CI/CD у процесі розробки веб-додатку є важливим елементом комплексної системи безпеки. Автоматизація перевірок, контроль залежностей, захист процесу розгортання та управління секретами дозволяють зменшити кількість вразливостей і забезпечити стабільну та безпечну роботу додатку. Поєднання CI/CD з іншими заходами захисту створює цілісну та ефективну систему безпеки на всіх етапах життєвого циклу веб-додатку.

1.8 Моніторинг та логування (Prometheus, Grafana, Loki)

Моніторинг та логування є невід'ємними складовими комплексного забезпечення безпеки веб-додатку, оскільки дозволяють не лише відстежувати стабільність роботи системи, але й своєчасно виявляти інциденти безпеки, аномальну поведінку та потенційні загрози [4]. У сучасних розподілених і контейнеризованих системах кількість подій та обсяг даних значно зростає, що робить ручний контроль практично неможливим [3]. Саме тому автоматизовані системи моніторингу та централізованого збору логів відіграють ключову роль у забезпеченні надійної та безпечної роботи веб-додатків.

Моніторинг дозволяє здійснювати безперервний контроль стану серверів, сервісів і додатків. Prometheus є поширеним інструментом з відкритим кодом для збору та зберігання метрик за моделлю pull — система сама з визначеним інтервалом звертається до цільових сервісів через HTTP-ендпоінт /metrics і зберігає отримані дані у вигляді часових рядів [17]. До таких метрик належать використання процесорного часу, оперативної пам'яті, мережевої активності, а також специфічні показники роботи веб-додатку: кількість HTTP-запитів, розподіл кодів відповідей, час обробки запитів [9].

		Лашілін А.С.			ЖИТОМИРСЬКА ПОЛІТЕХНІКА.26.125.9	Арк.
		Воротніков В.В				25
Змн.	Арк.	№ докум.	Підпис	Дата		

Аналіз цих даних дозволяє виявляти перевантаження, збої або аномальну активність, які можуть бути наслідком атак або помилок конфігурації.

Grafana використовується для візуалізації даних моніторингу та надання наочного уявлення про стан системи в режимі реального часу [18]. Вона підтримує підключення різноманітних джерел даних і дозволяє будувати інтерактивні дашборди з графіками, таблицями та індикаторами стану. Завдяки механізму сповіщень Grafana може автоматично повідомляти адміністраторів при перевищенні порогових значень метрик. У контексті безпеки це дозволяє оперативно реагувати на різкі зміни навантаження або нестандартні шаблони активності, що можуть свідчити про спроби несанкціонованого доступу або атаки типу відмови в обслуговуванні.

Логування є не менш важливою складовою безпеки, оскільки забезпечує фіксацію подій та дій користувачів і системних компонентів [1]. Журнали веб-додатку, веб-сервера, бази даних та операційної системи містять цінну інформацію для аналізу інцидентів та розслідування порушень безпеки [10, 12]. Централізований збір логів дозволяє уникнути втрати даних у разі зупинки окремих сервісів та значно спрощує їх аналіз при виникненні інцидентів.

Loki — система агрегації та зберігання логів, розроблена Grafana Labs і орієнтована насамперед на контейнеризовані середовища [19]. На відміну від класичних систем повнотекстового пошуку, Loki індексує лише метадані у вигляді міток, а не вміст самих логів, що робить її значно економнішою за ресурсами при збереженні достатньої гнучкості для аналізу. Запити до логів виконуються мовою LogQL, яка дозволяє фільтрувати записи за мітками та здійснювати пошук за текстовим вмістом. Loki тісно інтегрується з Grafana, що дозволяє відображати логи та метрики в єдиному інтерфейсі та здійснювати кореляцію між ними.

Для збору логів з контейнерів Docker використовується агент Promtail, який читає журнали безпосередньо з файлової системи хоста, збагачує їх метаданими (іменами контейнерів, образів, міткою завдання) та надсилає до Loki [14, 19]. Такий підхід не вимагає змін у самих застосунках — агент збирає стандартний вивід контейнерів автоматично.

		Лашілін А.С.			ЖИТОМИРСЬКА ПОЛІТЕХНІКА.26.125.9	Арк.
		Воротніков В. В				26
Змн.	Арк.	№ докум.	Підпис	Дата		

Моніторинг і логування також відіграють важливу роль у реагуванні на інциденти безпеки [4]. Завдяки збереженим журналам та метрикам адміністратор може відновити послідовність подій, визначити джерело проблеми та оцінити масштаб інциденту. Зокрема, раптове зростання кількості відповідей з кодом 401 може свідчити про brute-force атаку на автентифікацію, а аномальне збільшення трафіку — про DoS-атаку [2, 5]. Це особливо важливо для аналізу атак, виявлення вразливостей та подальшого вдосконалення системи захисту (табл. 1.1).

Крім того, системи моніторингу та логування є важливими з точки зору відповідності вимогам безпеки та аудиту [6]. Наявність журналів подій та історії метрик дозволяє підтвердити дотримання політик безпеки, а також забезпечити прозорість роботи системи. Це має особливе значення для інформаційних систем, які обробляють персональні або конфіденційні дані.

Таблиця 1.1

Порівняння інструментів моніторингу

Інструмент	Роль	Тип даних
Prometheus	Збір метрик (pull-модель)	Числові часові ряди
Grafana	Візуалізація та сповіщення	Дашборди
Loki	Агрегація логів	Текстові журнали
Promtail	Агент збору логів з контейнерів	—

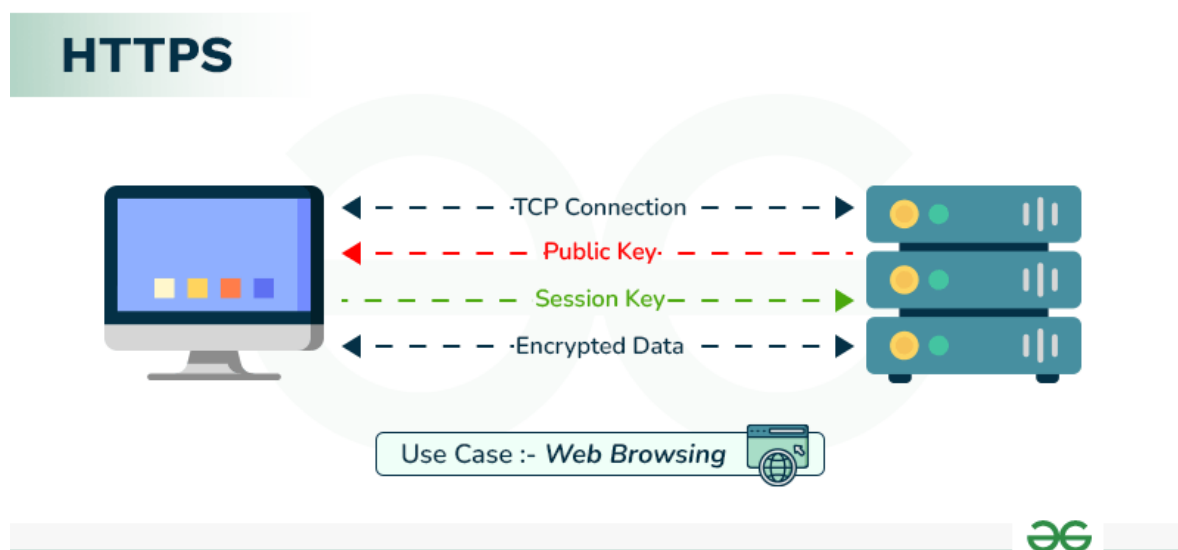
У підсумку, використання стеку спостережуваності на базі Prometheus, Grafana, Loki та Promtail дозволяє створити ефективну систему контролю за станом веб-додатку та всієї серверної інфраструктури. У поєднанні з іншими заходами безпеки ці інструменти забезпечують своєчасне виявлення загроз, швидке реагування на інциденти та підвищення загальної надійності веб-додатку.

1.9 Захищені протоколи передачі даних (HTTPS, TLS)

Захищені протоколи передачі даних є фундаментальною складовою безпеки веб-додатків, оскільки забезпечують конфіденційність, цілісність та автентичність інформації, що передається між клієнтом і сервером [1]. У сучасних інформаційних системах значна частина обміну даними відбувається через відкриті мережі, зокрема мережу Інтернет, що робить передачу інформації потенційно вразливою до перехоплення, підміни або модифікації [2]. Саме тому використання захищених протоколів є обов'язковою умовою побудови надійних веб-сервісів.

Протокол HTTPS є розширенням протоколу HTTP, яке використовує криптографічні механізми для захисту переданих даних [2]. Основою HTTPS є протокол TLS, який відповідає за шифрування з'єднання, перевірку автентичності сервера та забезпечення цілісності повідомлень [8]. Завдяки TLS клієнт може бути впевненим у тому, що взаємодіє саме з легітимним сервером, а передана інформація не може бути прочитана або змінена сторонніми особами (рис. 1.10).

Використання HTTPS дозволяє захистити облікові дані користувачів, токени доступу, персональну інформацію та інші чутливі дані від перехоплення під час передачі мережею [5, 7]. Без застосування TLS будь-який трафік може бути легко перехоплений за допомогою атак типу “людина посередині” (MitM), що створює серйозну загрозу для безпеки веб-додатку [1, 6]. Таким чином, HTTPS є необхідним елементом захисту навіть для простих веб-сервісів.



		Лашілін А.С.			ЖИТОМИРСЬКА ПОЛІТЕХНІКА.26.125.9	Арк.
		Воротніков В.В.				28
Змн.	Арк.	№ докум.	Підпис	Дата		

Рисунок 1.10 – Принцип роботи HTTPS

Протокол TLS також забезпечує цілісність даних, що передаються між клієнтом і сервером [8]. Це означає, що у разі спроби змінити інформацію під час передачі з'єднання буде автоматично розірване [2]. Такий механізм захищає веб-додаток від підміни контенту, ін'єкції шкідливого коду та інших атак, спрямованих на зміну переданих даних (рис. 1.11).

Важливим аспектом використання HTTPS є правильна конфігурація криптографічних параметрів [12]. Сучасні веб-сервери дозволяють відключати застарілі версії TLS та небезпечні алгоритми шифрування, що знижує ризик криптографічних атак [1, 8]. Регулярне оновлення сертифікатів за допомогою автоматизованих клієнтів та використання перевірених центрів сертифікації забезпечує довіру клієнтів до веб-додатку та підвищує загальний рівень безпеки.

Застосування HTTPS позитивно впливає не лише на безпеку, але й на функціональні можливості веб-додатку. Багато сучасних браузерів обмежують або блокують роботу веб-функцій у разі відсутності захищеного з'єднання, що робить HTTPS стандартом де-факто для сучасних веб-систем.

		Лашілін А.С.			ЖИТОМИРСЬКА ПОЛІТЕХНІКА.26.125.9	Арк.
		Воротніков В. В				29
Змн.	Арк.	№ докум.	Підпис	Дата		

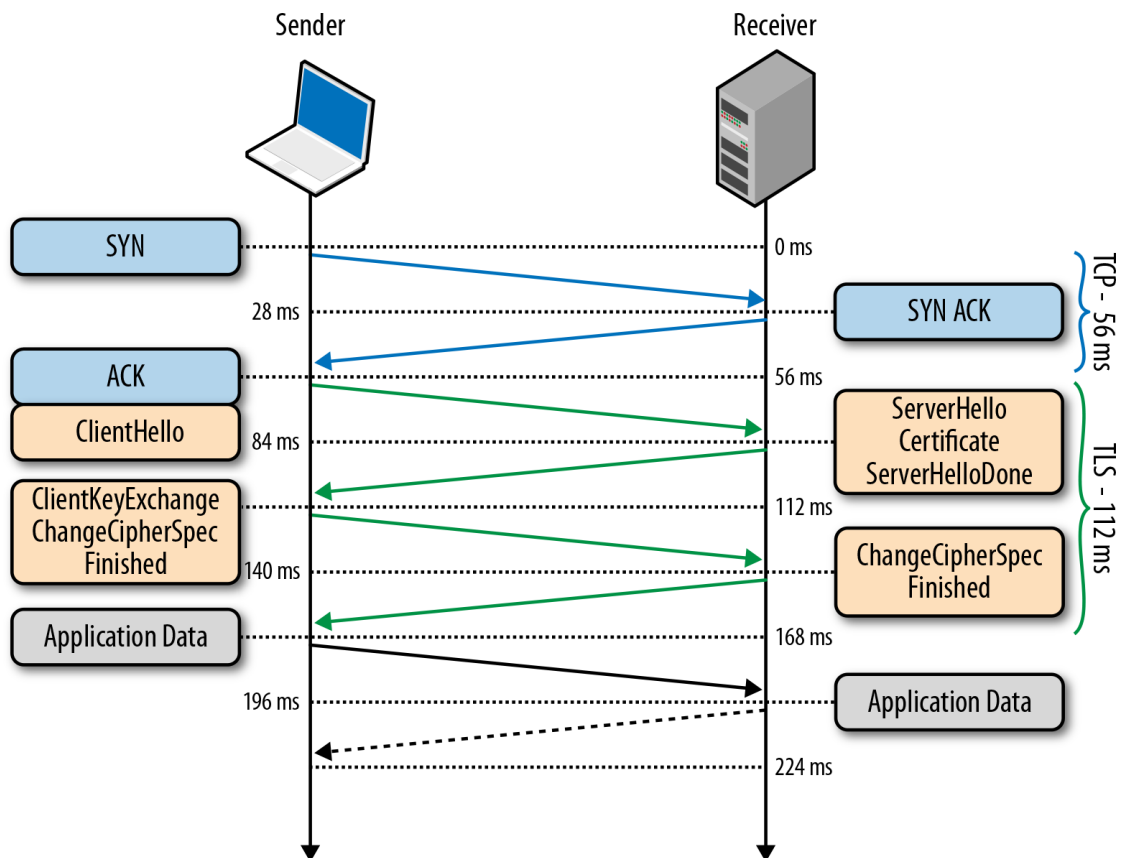


Рисунок 1.11 – Принцип роботи TLS

Крім того, використання захищених протоколів є необхідною умовою для реалізації багатьох механізмів безпеки, таких як захищені cookie або токени авторизації.

У контексті комплексного забезпечення безпеки веб-додатку HTTPS та TLS виступають першим рівнем захисту даних, який доповнює інші механізми безпеки, зокрема аутентифікацію, авторизацію та захист серверної інфраструктури [3, 4]. Без належного захисту каналу передачі даних навіть найнадійніші серверні та програмні механізми можуть бути скомпрометовані.

У підсумку, використання захищених протоколів передачі даних є обов'язковою умовою для створення сучасного та безпечного веб-додатку. HTTPS та TLS забезпечують конфіденційність, цілісність та автентичність інформації, що передається, і відіграють ключову роль у загальній системі захисту веб-додатку. Їх правильне налаштування та використання у поєднанні з іншими заходами безпеки дозволяє створити надійну та стійку до загроз інформаційну систему.

Висновки до першого розділу

		Лашілін А.С.			ЖИТОМИРСЬКА ПОЛІТЕХНІКА.26.125.9	Арк.
		Воротніков В.В.				30
Змн.	Арк.	№ докум.	Підпис	Дата		

У першому розділі було проведено детальний аналіз теоретичних основ безпеки веб-додатків та розглянуто ключові аспекти захисту на різних рівнях системи. Було досліджено безпеку Linux-сервера, включаючи принципи управління доступом, конфігурацію брандмауера, оновлення системи та захист від несанкціонованого доступу.

Особлива увага приділена безпеці коду веб-додатку: розглянуто механізми автентифікації, хешування паролів, обробку помилок, використання JWT та принципи контролю доступу на основі ролей. Також проаналізовано захист від типових веб-атак, таких як SQL-ін'єкції, XSS та CSRF, із урахуванням того, що сучасні фреймворки (FastAPI, SQLAlchemy) значною мірою мінімізують ці ризики.

У межах розділу детально розглянуто безпеку веб-сервера Nginx, включаючи налаштування reverse проху, HTTPS та TLS, захисні HTTP-заголовки, обмеження доступу та механізми rate limiting. Було описано принципи безпечної роботи з базою даних PostgreSQL, включно з автентифікацією SCRAM-SHA-256, розмежуванням ролей, Row-Level Security, шифруванням з'єднань та аудитом дій користувачів.

Також проаналізовано роль контейнеризації за допомогою Docker у забезпеченні ізоляції сервісів, а також інтеграцію практик CI/CD для підвищення безпеки процесу розробки та розгортання. Розглянуто моніторинг і логування (Prometheus, Grafana, ELK/Loki), що дозволяють відстежувати безпеку системи та оперативно реагувати на інциденти.

У результаті проведеного теоретичного дослідження встановлено, що комплексний підхід до безпеки веб-додатку повинен охоплювати всі рівні системи: серверну інфраструктуру, код, базу даних, мережу та процеси розробки. Такий підхід забезпечує мінімізацію ризиків компрометації та відповідає сучасним стандартам безпеки веб-додатків.

		Лашілін А.С.			ЖИТОМИРСЬКА ПОЛІТЕХНІКА.26.125.9	Арк.
		Воротніков В. В				31
Змн.	Арк.	№ докум.	Підпис	Дата		

РОЗДІЛ 2 РЕАЛІЗАЦІЯ РОЗГЛЯНУТИХ АСПЕКТІВ

У другому розділі реалізовано комплексну багаторівневу систему захисту веб-додатку, яка охоплює серверну безпеку, налаштування reverse проху, контейнеризацію, процеси CI/CD, механізми автентифікації, а також систему моніторингу та журналювання. Кожен підрозділ є невід’ємною частиною єдиної архітектури безпеки, де заходи захисту взаємодоповнюють та підсилюють один одного (рис. 2.1).

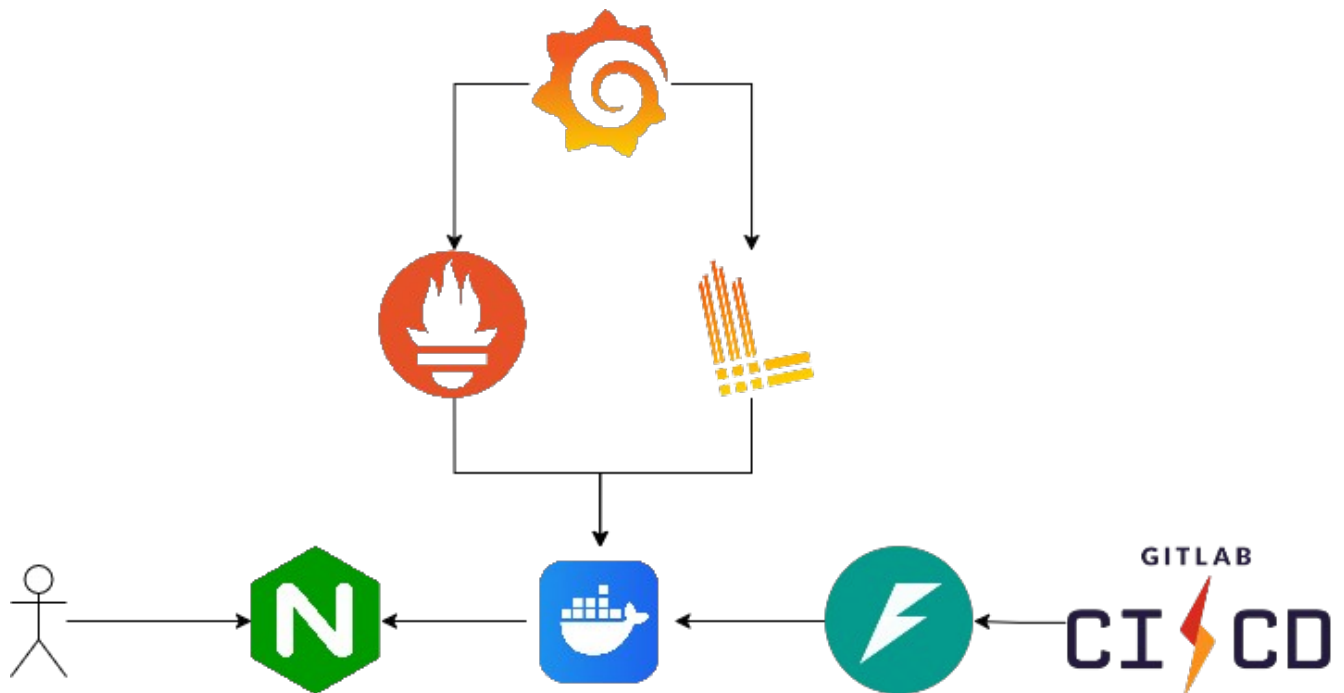


Рисунок 2.1 – Схема роботи додатку

Варто зазначити, що окремі налаштування в межах цього розділу є демонстраційними і відповідають вимогам тестового середовища. Зокрема, спрощені облікові дані та деякі параметри конфігурацій є прийнятними для локальної розробки та навчального проєкту, однак у production-середовищі вони мають бути замінені на захищені значення: паролі повинні зберігатися у захищених змінних середовища та відповідати вимогам складності.

2.1 Захист сервера

Захист серверного середовища реалізовано на базі операційної системи Linux (Ubuntu Server), оскільки саме сервер є фундаментом для розгортання всіх компонентів

		Лашілін А.С.			ЖИТОМИРСЬКА ПОЛІТЕХНІКА.26.125.9	Арк.
		Воротніков В. В				32
Змн.	Арк.	№ докум.	Підпис	Дата		

веб-додатку. Практична реалізація безпеки починається з базового налаштування системи, обмеження доступу та зменшення поверхні атаки [2].

Після розгортання сервера виконано оновлення системних пакетів з метою усунення відомих вразливостей:

```
sudo apt update && sudo apt upgrade -y
```

Це дозволяє підтримувати актуальний стан ядра та системного програмного забезпечення, що є базовою вимогою безпеки [4].

Для роботи з сервером створено окремого користувача без прав суперкористувача, що відповідає принципу найменших привілеїв [1, 6]:

```
sudo adduser appuser  
sudo usermod -aG sudo appuser
```

Доступ під обліковим записом `root` по SSH було вимкнено, що зменшує ризик атак перебором [2]. Віддалений доступ налаштовано виключно через ключову автентифікацію. У конфігурації SSH-сервера змінено відповідні параметри в `/etc/ssh/sshd_config`:

```
PermitRootLogin no  
PasswordAuthentication no
```

Після внесення змін служба SSH була перезапущена:

```
sudo systemctl restart ssh
```

Для обмеження мережевого доступу використано вбудований брандмауер `ufw` [2]. Доступ дозволено лише до необхідних сервісів — SSH та HTTPS:

```
sudo ufw allow OpenSSH  
sudo ufw allow 443  
sudo ufw enable
```

Журналювання системних подій використовується для подальшого аналізу безпеки. Логи доступні через стандартні механізми Linux:

```
journalctl -xe
```

Ці журнали надалі передаються до централізованої системи логування, що дозволяє здійснювати аналіз подій у масштабах всієї інфраструктури [19].

		Лашілін А.С.			ЖИТОМИРСЬКА ПОЛІТЕХНІКА.26.125.9	Арк.
		Воротніков В. В				33
Змн.	Арк.	№ докум.	Підпис	Дата		

Також було перевірено список активних мережевих сервісів з метою виявлення зайвих відкритих портів [2]:

```
ss -tuln
```

Усі служби, які не використовуються для роботи веб-додатку, були вимкнені або видалені.

У результаті виконаних дій сервер було підготовлено до безпечного розгортання веб-додатку. Реалізовані заходи забезпечують базовий рівень захисту операційної системи, контроль доступу, мережеву ізоляцію та механізми реагування на підозрілу активність. Це створює надійну основу для подальшої реалізації безпеки на рівні веб-додатку, бази даних та інфраструктурних сервісів.

2.2 Безпека коду

Безпека коду веб-додатку реалізується на рівні автентифікації, авторизації, зберігання облікових даних та роботи з конфігурацією. У даному проєкті веб-додаток розроблений із використанням фреймворку FastAPI [9], що дозволяє застосовувати сучасні механізми захисту на рівні коду та архітектури.

Захист конфігураційних даних:

Чутливі параметри, такі як секретні ключі, алгоритми підпису токенів та параметри доступу до бази даних, винесені у файл .env [15]. Це дозволяє уникнути жорсткого кодування секретів у вихідному коді та підвищує безпеку при роботі з репозиторіями.

```
SECRET_KEY=f3b3cc4586e4d7a177623f5603ad6f54
ALGORITHM=HS256
ACCESS_TOKEN_EXPIRE_MINUTES=30
```

Зчитування конфігурації реалізовано через окремий клас AuthConfig, що централізує роботу з параметрами безпеки [9]:

```
class AuthConfig(BaseConfig):
    SECRET_KEY: str
    ALGORITHM: str
    ACCESS_TOKEN_EXPIRE_MINUTES: int
```

Хешування та зберігання паролів:

		Лашілін А.С.			ЖИТОМИРСЬКА ПОЛІТЕХНІКА.26.125.9	Арк.
		Воротніков В. В				
Змн.	Арк.	№ докум.	Підпис	Дата		34

Для захисту облікових даних користувачів у додатку використовується хешування паролів із застосуванням алгоритму bcrypt [1]. Це виключає можливість зберігання паролів у відкритому вигляді та значно ускладнює їх відновлення у разі витоку бази даних [5, 6].

```
pwd_context = CryptContext(schemes=["bcrypt"],
deprecatеd="auto")
oauth2_scheme = OAuth2PasswordBearer(tokenUrl="/users/login")
```

Перед збереженням користувача пароль хешується:

```
def _get_password_hash(self, password: str) -> str:
return pwd_context.hash(password)
```

Під час автентифікації виконується перевірка відповідності введеного пароля збереженому хешу:

```
def _verify_password(
self,
plain_password: str,
hashed_password: str,
) -> bool:
return pwd_context.verify(plain_password, hashed_password)
```

Таким чином, реальні паролі користувачів ніколи не передаються і не зберігаються у відкритому вигляді.

Автентифікація та авторизація користувачів:

У веб-додатку реалізовано токенну автентифікацію з використанням стандарту OAuth2 та JSON Web Token (JWT) [7]. FastAPI надає вбудований механізм OAuth2PasswordBearer [9], який використовується для отримання токена доступу:

```
pwd_context = CryptContext(schemes=["bcrypt"],
deprecatеd="auto")
oauth2_scheme = OAuth2PasswordBearer(tokenUrl="/users/login")
```

Після успішної автентифікації користувача формується JWT-токен із заданим терміном дії [7]:

```
def create_access_token(
self,
data: dict,
expires_delta: timedelta | None = None,
) -> str:
to_encode = data.copy()
if expires_delta:
expire = datetime.now(UTC) + expires_delta
```

```

else:
    expire = datetime.now(UTC) + timedelta(minutes=15)
    to_encode.update({"exp": expire})
    return jwt.encode(
        to_encode,
        AuthConfig().SECRET_KEY,
        algorithm=AuthConfig().ALGORITHM,
    )

```

Використання поля exp дозволяє автоматично обмежувати час дії токена та зменшувати ризик його використання у разі компрометації [6, 7].

Перевірка токена та захист доступу до ресурсів:

Для отримання поточного авторизованого користувача реалізовано окремий метод, який декодує та перевіряє JWT-токен [7]:

```

payload = jwt.decode(
    token,
    AuthConfig().SECRET_KEY,
    algorithms=[AuthConfig().ALGORITHM],
)

```

У разі використання простроченого або некоректного токена генерується відповідна помилка авторизації [6]:

```

except ExpiredSignatureError as jwt_expired_e:
    raise http_exceptions.Unauthorized401 from jwt_expired_e
except JWTError as jwt_e:
    raise http_exceptions.Unauthorized401 from jwt_e

```

Після перевірки токена виконується пошук користувача у базі даних, що унеможлиблює доступ до ресурсів для неіснуючих або видалених облікових записів [10, 11].

Обробка помилок та захист логіки додатку:

У коді реалізована централізована обробка помилок, зокрема при створенні користувачів та авторизації. Наприклад, у разі спроби створити користувача з уже існуючим логіном перехоплюється помилка бази даних:

```

except IntegrityError as e:
    raise http_exceptions.Conflict409

```

		Лашілін А.С.			ЖИТОМИРСЬКА ПОЛІТЕХНІКА.26.125.9	Арк.
		Воротніков В.В.				36
Змн.	Арк.	№ докум.	Підпис	Дата		

Такий підхід дозволяє не розкривати внутрішню структуру бази даних та забезпечує контрольований вивід помилок для клієнта.

Висновок: у результаті реалізації наведених механізмів безпеки код веб-додатку забезпечує захищене зберігання облікових даних, надійну автентифікацію та контроль доступу до ресурсів. Використання сучасних криптографічних алгоритмів, JWT-токенів, хешування паролів і винесення конфігурацій у змінні середовища дозволяє зменшити ризики компрометації та відповідає сучасним вимогам безпеки веб-додатків.

2.3 Nginx та безпека веб-сервера

Веб-сервер Nginx у проєкті використовується як зворотний проксі (reverse proxy) для FastAPI-додатку та як основна точка входу клієнтських HTTP/HTTPS-запитів [3, 12]. Правильна конфігурація Nginx відіграє ключову роль у забезпеченні безпеки веб-додатку, оскільки саме на цьому рівні реалізуються механізми шифрування трафіку, обмеження доступу, фільтрації запитів та захисту від типових атак [5, 6].

На Linux-сервері Nginx встановлюється з офіційних репозиторіїв [12]:

```
sudo apt update
sudo apt install nginx -y
```

Після встановлення сервіс запускається та додається до автозапуску:

```
sudo systemctl start nginx
sudo systemctl enable nginx
```

Використання системного сервісу дозволяє централізовано контролювати стан веб-сервера та оперативно реагувати на збої.

Використання Nginx як reverse proxy:

Для ізоляції бекенд-додатку FastAPI від прямого доступу з мережі Інтернет Nginx налаштований як reverse proxy [9, 12]. FastAPI працює на локальному порту, доступному лише з сервера, наприклад 127.0.0.1:8000.

Приклад конфігурації серверного блоку:

```
server {
listen 80;
server_name example.com;

location / {
```

		Лашілін А.С.			ЖИТОМИРСЬКА ПОЛІТЕХНІКА.26.125.9	Арк.
		Воротніков В. В				
Змн.	Арк.	№ докум.	Підпис	Дата		37

```

proxy_pass http://127.0.0.1:8000;
proxy_set_header Host $host;
proxy_set_header X-Real-IP $remote_addr;
proxy_set_header X-Forwarded-For
$proxy_add_x_forwarded_for;
proxy_set_header X-Forwarded-Proto $scheme;
}
}

```

Такий підхід приховує внутрішню структуру додатку та зменшує площу атаки.

Захист через HTTPS та TLS:

Одним з ключових аспектів безпеки веб-сервера є шифрування переданих даних [1]. У проєкті використовується HTTPS із застосуванням TLS-сертифікатів [8], отриманих за допомогою Let's Encrypt.

Встановлення Certbot [13]:

```
sudo apt install certbot python3-certbot-nginx -y
```

```
sudo certbot -nginx -d example.com
```

Після цього Nginx автоматично налаштовується на роботу через HTTPS, що забезпечує конфіденційність та цілісність даних, включно з JWT-токенами та обліковими даними користувачів [7, 8].

Обмеження доступу та приховування інформації:

Для зменшення ризику розкриття службової інформації в Nginx вимикається відображення версії сервера [12]:

```
server_tokens off;
```

Також можливе обмеження доступу до адміністративних або службових маршрутів за IP-адресами:

```

Location /admin {
allow 127.0.0.1;
deny all;
}

```

Це унеможливлює несанкціонований доступ до критичних частин додатку [6].

Захист від надмірних запитів та DDOS-атак:

		Лашілін А.С.			ЖИТОМИРСЬКА ПОЛІТЕХНІКА.26.125.9	Арк.
		Воротніков В.В.				38
Змн.	Арк.	№ докум.	Підпис	Дата		

Nginx дозволяє реалізувати базовий захист від атак типу brute force та DDoS за допомогою обмеження кількості запитів [12].

```
limit_req_zone $binary_remote_addr zone=one: 10m rate=10r/s;

server {
    location / {
        limit_req zone=one burst=20 nodelay;
        proxy_pass http://127.0.0.1:8000;
    }
}
```

Таке обмеження дозволяє зменшити навантаження на бекенд та запобігти зловживанню API.

Для підвищення рівня безпеки у відповідях сервера додаються спеціальні HTTP-заголовки [6, 11]:

```
add_header X-Frame-Options "DENY";
add_header X-Content-Type-Options "nosniff";
add_header X-XSS-Protection "1; mode=block";
add_header Referrer-Policy "no-referrer";
```

Використання цих заголовків зменшує ризик XSS-атак, clickjacking та витоку інформації через заголовки.

Логування та інтеграція з моніторингом:

Nginx веде журнали доступу та помилок, які можуть бути інтегровані з системами централізованого логування, такими як ELK або Grafana Loki [12, 19]:

```
access_log /var/log/nginx/access.log;
error_log /var/log/nginx/error.log;
```

Це дозволяє аналізувати підозрілу активність, відстежувати спроби атак та реагувати на інциденти безпеки [4].

Висновок: у рамках проєкту Nginx виконує не лише роль веб-сервера, але й важливий компонент системи безпеки. Використання reverse проху, HTTPS, обмеження доступу, захисних заголовків та механізмів rate limiting дозволяє значно підвищити рівень захисту веб-додатку та зменшити ризики атак на серверну інфраструктуру.

		Лашілін А.С.			ЖИТОМИРСЬКА ПОЛІТЕХНІКА.26.125.9	Арк.
		Воротніков В. В				39
Змн.	Арк.	№ докум.	Підпис	Дата		

2.4 Захист PostgreSQL

База даних PostgreSQL є одним з найбільш критичних компонентів веб-додатку, оскільки саме в ній зберігаються облікові дані користувачів, службова інформація та результати роботи системи. Тому захист бази даних реалізується на декількох рівнях: ізоляція доступу, автентифікація, керування правами, шифрування з'єднань та аудит дій [10].

PostgreSQL встановлюється з офіційних репозиторіїв:

```
sudo apt update  
sudo apt install postgresql postgresql-contrib -y
```

Після встановлення сервіс працює під окремим системним користувачем postgres, що забезпечує початкову ізоляцію від інших процесів у системі.

Для підвищення безпеки доступ до сервера PostgreSQL обмежується локальним інтерфейсом [10]. У файлі конфігурації postgresql.conf задається:

```
listen_address = 'localhost'
```

Це запобігає прямому підключенню до бази даних ззовні мережі Інтернет.

Автентифікація та керування доступом:

PostgreSQL використовує файл pg_hba.conf для контролю автентифікації клієнтів.

Для локальних з'єднань застосовується сучасний метод хешування:

```
Localallallscram-sha-256  
host allall127.0.0.1/32    scram-sha-256
```

Метод scram-sha-256 забезпечує захищену передачу облікових даних і вважається рекомендованим для сучасних інсталяцій PostgreSQL [1, 10].

Розмежування ролей та мінімальні привілеї:

Для веб-додатку створюється окремий користувач бази даних з обмеженими правами [10]:

```
CREATE ROLE app_user WITH LOGIN PASSWORD 'strong_password';  
CREATE DATABASE books_db OWNER app_user;
```

Додатку не надаються адміністративні привілеї, такі як SUPERUSER, що мінімізує наслідки можливої компрометації облікових даних.

Доступ до таблиць обмежується лише необхідними операціями:

		Лашілін А.С.			ЖИТОМИРСЬКА ПОЛІТЕХНІКА.26.125.9	Арк.
		Воротніков В. В				40
Змн.	Арк.	№ докум.	Підпис	Дата		

```
GRANT SELECT, INSERT, UPDATE, DELETE ON ALL TABLES IN SCHEMA
public To app_user;
```

Таким чином реалізується принцип найменших привілеїв.

Захист від SQL-ін'єкцій на рівні БД:

Хоча основний захист від SQL-ін'єкцій реалізується на рівні коду за допомогою ORM [11], PostgreSQL додатково обмежує виконання небезпечних операцій шляхом заборони динамічного виконання SQL для ролей додатку [10]. Веб-додаток працює виключно з підготовленими запитами, що значно знижує ризик атак.

Row-Level Security (RLS):

Для додаткового контролю доступу до даних використовується механізм Row-Level Security [10]. Він дозволяє обмежувати доступ до рядків таблиць залежно від поточного користувача.

```
ALTER TABLE users ENABLE ROW LEVEL SECURITY;

CREATE POLICY user_policy
ON users
FOR SELECT
USING (username = current_user);
```

RLS дозволяє реалізувати логіку доступу безпосередньо на рівні бази даних, що зменшує залежність від коректності реалізації бізнес-логіки в коді [6].

Шифрування з'єднань

Для захисту даних під час передачі між додатком та базою даних використовується SSL. У конфігурації PostgreSQL вмикається підтримка SSL [10]:

```
ssl = on
```

У клієнтському підключенні (FastAPI) вказується обов'язкове використання захищеного з'єднання. Це особливо важливо при розгортанні бази даних на окремому сервері або у контейнерному середовищі [15].

Логування та аудит

PostgreSQL підтримує детальне логування дій користувачів, що дозволяє виявляти підозрілу активність [10]:

```
log_connections = on
log_disconnections = on
log_statement = 'ddl'
```

		Лашілін А.С.			ЖИТОМИРСЬКА ПОЛІТЕХНІКА.26.125.9	Арк.
		Воротніков В. В				41
Змн.	Арк.	№ докум.	Підпис	Дата		

Отримані логи можуть бути інтегровані з централізованими системами моніторингу, такими як ELK або Grafana Loki, для подальшого аналізу інцидентів безпеки.

Резервне копіювання та відновлення

Для захисту від втрати даних унаслідок атак або збоїв реалізується регулярне резервне копіювання бази даних:

```
pg_dump books_db > backup.sql
```

Резервні копії зберігаються в захищеному середовищі та не доступні з веб-сервера.

Висновок: у межах проєкту захист PostgreSQL реалізовано шляхом ізоляції доступу, використання сучасних методів автентифікації, розмежування прав, шифрування з'єднань та аудиту дій. Такий підхід дозволяє мінімізувати ризики несанкціонованого доступу до даних та забезпечити цілісність і конфіденційність інформації, що зберігається у базі даних.

2.5 Docker та ізоляція сервісів

Контейнеризація є одним із ключових елементів сучасної безпечної інфраструктури веб-додатків. У межах даного проєкту Docker та Docker Compose використовуються для ізоляції та оркестрації сервісів веб-додатку, бази даних і систем моніторингу, що дозволяє зменшити ризики компрометації системи та спростити керування середовищем розгортання.

Роль контейнеризації у безпеці

Docker забезпечує логічну ізоляцію процесів, файлової системи та мережі кожного сервісу. Навіть у разі компрометації одного контейнера зломисник не отримує прямого доступу до хост-системи або інших сервісів. Такий підхід значно зменшує наслідки потенційних атак та дозволяє чітко розмежувати відповідальність між компонентами системи.

Структура контейнеризованого середовища

		Лашілін А.С.			ЖИТОМИРСЬКА ПОЛІТЕХНІКА.26.125.9	Арк.
		Воротніков В. В				42
Змн.	Арк.	№ докум.	Підпис	Дата		

Уся інфраструктура проєкту описана в єдиному файлі `docker-compose.yml`, що дозволяє запускати та зупиняти всі сервіси однією командою [15]. У проєкті кожен компонент розміщується в окремому контейнері (табл. 2.1).

Таблиця 2.1

Перелік сервісів Docker Compose та їх ролі

Контейнер	Образ	Роль
api	python:3.13-slim	FastAPI-додаток
db	postgres:16	База даних
prometheus	prom/prometheus	Збір метрик
grafana	grafana/grafana	Візуалізація
loki	grafana/loki	Зберігання логів
promtail	grafana/promtail	Агент логів

Контейнери взаємодіють між собою через внутрішню мережу Docker Compose, яка за замовчуванням недоступна ззовні [15].

Dockerfile та мінімізація поверхні атаки

Для веб-додатку використовується мінімальний базовий образ `python:3.11-slim`, що зменшує кількість встановлених пакетів і, відповідно, потенційних вразливостей [5, 14]:

```
FROM python:3.13-slim

WORKDIR /app

COPY requirements.txt .
RUN pip install --no-cache-dir -r requirements.txt

COPY . .

CMD ["fastapi", "dev", "backend/main.py", "--host", "0.0.0.0"]
```

Використання образу `slim` суттєво зменшує розмір контейнера порівняно з повним образом, скорочуючи кількість компонентів, які можуть містити відомі вразливості. Прапор `--no-cache-dir` при встановленні залежностей додатково зменшує розмір фінального образу та виключає збереження зайвих файлів усередині контейнера [14].

Оркестрація сервісів через Docker Compose

Docker Compose дозволяє декларативно описати всю інфраструктуру проєкту, включаючи залежності між сервісами, мережеву взаємодію та монтування томів [15]. Директива `depends_on` гарантує, що контейнер `арі` запускається лише після того, як буде запущено контейнер бази даних `db`, що виключає помилки підключення при старті системи.

Захист змінних середовища

Конфіденційні дані застосунку, такі як `SECRET_KEY`, облікові дані бази даних та параметри токенів, передаються у контейнер `арі` через окремий файл `.env` за допомогою директиви `env_file` [15]:

```
env_file:  
- ./ .env
```

Це дозволяє уникнути збереження секретних даних безпосередньо у `docker-compose.yml` або всередині Docker-образу, що є критично важливим при публікації коду у відкритих репозиторіях [5, 6]. Файл `.env` додається до `.gitignore` і не потрапляє до системи контролю версій.

Для сервісу Grafana облікові дані адміністратора передаються через змінні середовища безпосередньо у `docker-compose.yml`:

```
- GF_SECURITY_ADMIN_USER=admin  
- GF_SECURITY_ADMIN_PASSWORD=admin
```

У продуктивному середовищі ці значення також мають виноситися у захищений `.env`-файл або менеджер секретів.

Ізоляція мережі

Docker Compose автоматично створює окрему внутрішню мережу для всіх сервісів, описаних у файлі. Сервіси взаємодіють між собою за іменами контейнерів як DNS-іменами (наприклад, `арі` звертається до бази даних за адресою `db`), але ці імена недоступні ззовні мережі Docker. Назовні публікуються лише необхідні порти — зокрема, порт 8000 для API та порт 3000 для Grafana — що мінімізує зовнішню поверхню атаки [2].

Постійне зберігання даних через томи

Для збереження даних між перезапусками контейнерів використовуються іменовані томи Docker [15]:

		Лашілін А.С.			ЖИТОМИРСЬКА ПОЛІТЕХНІКА.26.125.9	Арк.
		Воротніков В.В				44
Змн.	Арк.	№ докум.	Підпис	Дата		

```
volumes:
  postgres_data:
  grafanadata:
  prometheusdata:
```

Дані PostgreSQL, Grafana та Prometheus зберігаються у відповідних томах і не втрачаються при зупинці або оновленні контейнерів. Такий підхід відокремлює стан системи від самих контейнерів, що відповідає принципу незмінності інфраструктури.

Моніторинг та логування як елементи безпеки

Стек моніторингу, що складається з Promtail, Loki, Prometheus та Grafana, дозволяє в реальному часі відстежувати стан сервісів і виявляти аномалії. Promtail збирає логи з усіх запущених контейнерів, читаючи їх безпосередньо з Docker-сокету та директорії /var/lib/docker/containers, і передає до Loki для агрегації. Prometheus збирає числові метрики з додатку, а Grafana надає зручний інтерфейс для їх аналізу та налаштування сповіщень [17, 18]. З точки зору кібербезпеки, централізоване логування є важливим інструментом виявлення інцидентів та проведення їх розслідування.

Висновок: використання Docker та Docker Compose у проєкті дозволило реалізувати ізольовану, керовану та безпечну інфраструктуру веб-додатку. Декларативний опис усіх сервісів в єдиному файлі спрощує розгортання та аудит конфігурації. Контейнеризація мінімізує поверхню атаки, забезпечує чітке розмежування сервісів та підвищує загальний рівень безпеки системи, що є критично важливим для сучасних веб-додатків.

2.6 CI/CD та безпека розробки

Для автоматизації процесу розробки, тестування та розгортання веб-додатку у межах проєкту використовується GitLab CI/CD [16]. Пайплайн складається з трьох основних етапів, які наведені в табл.2.2.

Файл .gitlab-ci.yml конфігурує ці етапи та визначає глобальні параметри виконання [16].

Таблиця 2.2

Етапи CI/CD пайплайну

Етап				Образ		Дія	
		Лашілін А.С.					Арк.
		Воротніков В. В					45
Змн.	Арк.	№ докум.	Підпис	Дата	ЖИТОМИРСЬКА ПОЛІТЕХНІКА.26.125.9		

build	docker:25-cli	Збірка Docker-образу
test	python:3.11	pytest + PostgreSQL сервіс
deploy	docker:25-cli	Білд сервісів та деплой

Глобальна конфігурація пайплайну

Файл `.gitlab-ci.yml` конфігурує ці етапи та визначає глобальні параметри виконання [16].

Глобальна конфігурація пайплайну

На рівні файлу визначається базовий образ та сервіс для всіх етапів:

```
default:
  image: docker:25-cli
  services:
  - name: docker:25-dind
    alias:
```

Використання Docker-in-Docker (`docker:25-dind`) дозволяє виконувати команди Docker безпосередньо всередині CI/CD-раннера. Змінна `DOCKER_TLS_CERTDIR: "/certs"` вмикає TLS-шифрування між клієнтом Docker та демоном, що забезпечує захист комунікації всередині пайплайну [8, 14].

Опис етапів

1. Build

На цьому етапі перевіряється доступність Docker та збирається образ застосунку:

```
build-job:
  stage: build
  script:
  - docker info
  - docker build -t api:latest .
```

Команда `docker info` підтверджує коректну роботу Docker-демона перед збіркою. Збірка образу з тегом `api:latest` гарантує наявність актуальної версії застосунку для подальших етапів [16].

2. Test

		Лашілін А.С.			ЖИТОМИРСЬКА ПОЛІТЕХНІКА.26.125.9	Арк.
		Воротніков В.В.				46
Змн.	Арк.	№ докум.	Підпис	Дата		

Тестування виконується в окремому середовищі на базі образу python:3.11 з підключеним сервісом PostgreSQL 16:

```
test-job:
  stage: test
  image: python:3.11
  services:
  - name: postgres:16
    alias: db
```

Це забезпечує повну ізоляцію тестового середовища від середовища збірки та розгортання. База даних запускається як окремий сервіс з параметрами, що передаються через змінні CI/CD. Перед запуском тестів встановлюються залежності та виконується перехід до директорії backend:

```
before_script:
  - pip install -r requirements.txt
  - cd backend
script:
  - pytest -s -v
```

Тести перевіряють автентифікацію, роботу з JWT-токенами та взаємодію з базою даних [9, 20]. Код, що не пройшов тестування, автоматично блокується і не потрапляє на етап розгортання.

3. Deploy

Після успішного проходження тестів відбувається автоматичне розгортання додатку. На цьому етапі динамічно формується файл .env зі змінних CI/CD, після чого запускається Docker Compose:

```
script:
  - echo "SECRET_KEY=$SECRET_KEY" >> .env
  - echo "ALGORITHM=$ALGORITHM" >> .env
  - echo
  "ACCESS_TOKEN_EXPIRE_MINUTES=$ACCESS_TOKEN_EXPIRE_MINUTES"
  >> .env
  - echo "LOGIN_USER=$LOGIN_USER" >> .env
  - echo "PASSWORD=$PASSWORD" >> .env
  - echo "SERVERNAME=$SERVERNAME" >> .env
  - echo "DBNAME=$DBNAME" >> .env
  - echo "DRIVER=$DRIVER" >> .env
  - docker compose up -d --build
```

Такий підхід дозволяє уникнути зберігання секретів у репозиторії — файл .env генерується безпосередньо під час виконання пайплайну зі змінних, що зберігаються у захищеному сховищі GitLab CI/CD Variables. Прапор --build

		Лашілін А.С.			ЖИТОМИРСЬКА ПОЛІТЕХНІКА.26.125.9	Арк.
		Воротніков В.В				47
Змн.	Арк.	№ докум.	Підпис	Дата		

гарантує, що при кожному розгортанні буде зібрано актуальний образ, а прапор `-d` запускає сервіси у фоновому режимі [15].

На цьому ж етапі явно вимикається TLS для Docker-демона (`DOCKER_TLS_CERTDIR: ""`), оскільки з'єднання відбувається по внутрішній мережі раннера через `tcp://docker:2375`, що є прийнятним у межах ізольованого CI/CD-середовища [14, 16].

З точки зору кібербезпеки пайплайн реалізує декілька важливих принципів:

Ізоляція середовищ — кожен етап виконується у власному контейнері з окремим набором сервісів, що обмежує можливі наслідки компрометації будь-якого з етапів.

Захист секретів — усі чутливі дані (`SECRET_KEY`, паролі, параметри підключення) зберігаються у GitLab CI/CD Variables і ніколи не з'являються у репозиторії у відкритому вигляді.

Динамічне формування конфігурації — файл `.env` генерується виключно під час виконання пайплайну і не зберігається між запусками.

Автоматичний контроль якості — код, що не пройшов тести, блокується на етапі `test` і не може бути розгорнутий, що зменшує ризик потрапляння вразливого коду у `production`.

Аудит змін — GitLab CI/CD реєструє всі коміти, запуски пайплайнів та їх результати, що забезпечує повну прозорість процесу розробки та можливість розслідування інцидентів.

Висновок: реалізований GitLab CI/CD пайплайн забезпечує безпечну та контрольовану розробку і розгортання веб-додатку. Чіткий поділ на етапи збірки, тестування та деплою, у поєднанні із захистом секретів через змінні середовища та ізоляцією контейнерів, дозволяє мінімізувати людський фактор, підвищити стабільність системи та знизити ризики несанкціонованих змін у `production-середовищі`.

2.7 Налаштування моніторингу та логування (Prometheus, Grafana, Loki)

Prometheus

		Лашілін А.С.			ЖИТОМИРСЬКА ПОЛІТЕХНІКА.26.125.9	Арк.
		Воротніков В. В				48
Змн.	Арк.	№ докум.	Підпис	Дата		

Файл конфігурації prometheus.yml визначає глобальний інтервал збору метрик та цілі для моніторингу:

```
global:
  scrape_interval: 3s
scrape_configs:
  - job_name: "fastapi"
metrics_path: /metrics
static_configs:
  - targets: ["api:8000"]
```

Інтервал збору у 3 секунди забезпечує високу гранулярність даних, що дозволяє оперативно виявляти короточасні аномалії в навантаженні або кількості помилок. Prometheus звертається до FastAPI-додатку за адресою api:8000/metrics — це внутрішня DNS-адреса у мережі Docker Compose, недоступна ззовні [9, 15]. Таким чином, ендпоінт метрик не є публічно відкритим, що запобігає витоку внутрішньої інформації про стан системи.

Loki

Конфігурація loki-config.yaml визначає параметри зберігання та обробки логів:

```
auth_enabled: false
server:
  http_listen_port: 3100
schema_config:
  configs:
  - from: 2020-10-24
    store: boltdb-shipper
    object_store: filesystem
    schema: v11
    index:
      prefix: index_
      period: 24h
limits_config:
  retention_period: 168h # 7 days
  allow_structured_metadata: false
table_manager:
  retention_deletes_enabled: true
  retention_period: 168h # 7 days
```

Loki використовує локальне файлове сховище (boltdb-shipper + filesystem) для індексів та чанків логів відповідно. Термін зберігання логів обмежено 7 днями (168 годин) — після цього вони автоматично видаляються, що дозволяє контролювати обсяг займаного дискового простору. Параметр allow_structured_metadata: false

		Лашілін А.С.			ЖИТОМИРСЬКА ПОЛІТЕХНІКА.26.125.9	Арк.
		Воротніков В. В				49
Змн.	Арк.	№ докум.	Підпис	Дата		

вимикає структуровані метадані, спрощуючи схему індексування. Ingester налаштований з коротким часом утримання чанків (chunk_idle_period: 5m, max_chunk_age: 1h), що забезпечує своєчасний запис логів на диск [19].

Promtail

Конфігурація promtail-config.yaml визначає джерела збору логів та правила їх обробки:

```
clients:
  - url: http://loki:3100/loki/api/v1/push
scrape_configs:
  - job_name: docker-logs
    static_configs:
      - targets:
        - localhost
        labels:
          job: docker-logs
          __path__: /var/lib/docker/containers/*/*log
          pipeline_stages:
            - json:
                expressions:
                  stream: stream
                  attrs: attrs
                  tag: attrs.tag
            - regex:
                expression: (?P<image_name>(?:[^\s]*[^\s]))\.(?P<container_name>(?:[^\s]*[^\s]))\.(?P<image_id>(?:[^\s]*[^\s]))\.(?P<container_id>(?:[^\s]*[^\s]))
                source: "tag"
            - labels:
                tag:
                  image_name:
                  container_name:
```

Promtail читає логи з директорії /var/lib/docker/containers/*/*log, яка монтується у контейнер у режимі лише для читання (ro) [14, 15]. Це важлива деталь з точки зору безпеки — агент має доступ до логів, але не може модифікувати їх. Конвеєр обробки (pipeline_stages) парсить JSON-формат логів Docker, витягує з поля tag імена образів та контейнерів за допомогою регулярного виразу і додає їх як мітки Loki [14, 19]. Завдяки цьому в Grafana можна фільтрувати логи за конкретним контейнером або образом.

Grafana — підключення джерел даних

Файл grafana/datasources.yaml описує автоматичне провізування джерел даних при першому запуску Grafana:

		Лашілін А.С.			ЖИТОМИРСЬКА ПОЛІТЕХНІКА.26.125.9	Арк.
		Воротніков В. В				50
Змн.	Арк.	№ докум.	Підпис	Дата		

```

apiVersion: 1
datasources:
  - name: Prometheus
    type: prometheus
    access: proxy
    url: http://prometheus:9090
    isDefault: true
    editable: true
  - name: Loki
    type: loki
    access: proxy
    url: http://loki:3100
    editable: true

```

Обидва джерела підключаються у режимі access: proxy, тобто всі запити до Prometheus та Loki виконує сам сервер Grafana, а не браузер користувача. Це означає, що порти Prometheus та Loki не обов'язково повинні бути відкриті назовні — Grafana звертається до них через внутрішню мережу Docker Compose. Grafana виступає єдиною точкою доступу до всіх даних моніторингу для кінцевого користувача.

Реалізований стек спостережуваності виконує кілька важливих функцій з позиції інформаційної безпеки.

Виявлення аномалій та атак. Метрики FastAPI, що збираються Prometheus, дозволяють відстежувати різке зростання кількості HTTP-запитів (потенційна DoS-атака), збільшення частки відповідей з кодами 401 та 403 (спроби несанкціонованого доступу або brute-force автентифікації), а також аномальний час відповіді сервісів [5, 6]. Grafana дозволяє налаштувати автоматичні сповіщення при перевищенні порогових значень цих метрик.

Централізований аудит подій. Loki агрегує логи з усіх контейнерів в єдиному місці. Це дозволяє проводити кореляцію подій між різними сервісами, що є критично важливим при розслідуванні інцидентів — наприклад, зіставити підозрілий запит до API з відповідним записом у логах бази даних.

Незмінність логів. Promtail монтує директорію з логами Docker у режимі лише для читання, що унеможливорює модифікацію або видалення журналів з боку агента [14, 15]. Логи, що надійшли до Loki, зберігаються незалежно від стану контейнерів

		Лашілін А.С.			ЖИТОМИРСЬКА ПОЛІТЕХНІКА.26.125.9	Арк.
		Воротніков В. В				51
Змн.	Арк.	№ докум.	Підпис	Дата		

застосунку — навіть якщо контейнер буде зупинено або видалено, його журнали залишаться доступними протягом встановленого терміну зберігання.

Мінімізація зовнішньої поверхні атаки. Сервіси Prometheus та Loki не є обов'язково публічно доступними — у поточній конфігурації їх порти відкриті переважно для розробки та налагодження. У продуктивному середовищі рекомендується закрити ці порти для зовнішніх підключень, залишивши доступ лише через Grafana та внутрішню мережу Docker [2, 15].

Контроль терміну зберігання даних. Обмеження зберігання логів сімома днями відповідає принципу мінімізації даних — система не накопичує надмірний обсяг чутливої інформації, що зменшує потенційний збиток у разі несанкціонованого доступу до сховища логів [6].

Висновок: реалізований стек моніторингу та логування на базі Prometheus, Grafana, Loki та Promtail забезпечує повну спостережуваність інфраструктури веб-додатку. З точки зору кібербезпеки він виконує функції системи виявлення аномалій, централізованого аудиту подій та інструменту розслідування інцидентів. Автоматичне провізування конфігурацій через файли `datasources.yaml` та ізоляція сервісів у внутрішній мережі Docker Compose мінімізують поверхню атаки та забезпечують цілісність зібраних даних.

		Лашілін А.С.			ЖИТОМИРСЬКА ПОЛІТЕХНІКА.26.125.9	Арк.
		Воротніков В. В				52
Змн.	Арк.	№ докум.	Підпис	Дата		

Висновки до другого розділу

У другому розділі роботи було реалізовано практичні аспекти комплексного забезпечення безпеки веб-додатку, включно із захистом серверної інфраструктури, бази даних, коду додатку, протоколів передачі даних та застосуванням сучасних практик DevSecOps.

Зокрема, було проведено захист Linux-сервера: встановлено та налаштовано мінімально необхідні сервіси, обмежено доступ через користувачів та групи, налаштовано firewall і регулярне оновлення системи.

У межах безпеки коду веб-додатку реалізовано автентифікацію користувачів із використанням JWT-токенів, хешування паролів за алгоритмом bcrypt, винесення конфіденційних параметрів у змінні середовища та централізовану обробку помилок.

Було забезпечено захист від типових веб-атак: використання сучасного ORM і FastAPI мінімізує ризик SQL-ін'єкцій, XSS та CSRF, а у PostgreSQL застосовано RLS для контролю доступу на рівні рядків.

Nginx налаштовано як reverse proxy із підтримкою HTTPS та TLS, захисними HTTP-заголовками, обмеженням доступу та механізмами rate limiting, що зменшує ймовірність DoS-атак.

У PostgreSQL забезпечено автентифікацію SCRAM-SHA-256, мінімальні привілеї користувачів, SSL-з'єднання та аудит дій користувачів.

Контейнеризація через Docker та оркестрація за допомогою Docker Compose дозволили ізолювати сервіси додатку у внутрішній мережі, декларативно описати всю інфраструктуру в єдиному файлі та захистити секрети через динамічне формування env-файлу, що підвищує безпеку та керованість системи.

CI/CD на GitLab забезпечує автоматичну збірку, тестування та розгортання з чітким розподілом на ізольовані етапи. Секрети передаються виключно через захищені змінні GitLab CI/CD Variables і динамічно підставляються під час виконання пайплайну, що унеможливорює їх потрапляння до репозиторію. Код, що не пройшов автоматичне тестування, блокується і не може бути розгорнутий у production.

Розгорнутий стек моніторингу та логування на базі Prometheus, Grafana, Loki та Promtail забезпечує повну спостережуваність інфраструктури. Prometheus збирає

		Лашілін А.С.			ЖИТОМИРСЬКА ПОЛІТЕХНІКА.26.125.9	Арк.
		Воротніков В.В				
Змн.	Арк.	№ докум.	Підпис	Дата		53

метрики FastAPI-додатку з інтервалом у три секунди, що дозволяє оперативно виявляти аномалії у навантаженні та кількості помилок. Promtail агрегує логи з усіх контейнерів і передає їх до Loki для централізованого зберігання та аналізу, а Grafana надає єдину точку доступу до всіх даних спостережуваності. З точки зору кібербезпеки цей стек виконує функції системи виявлення аномалій, централізованого аудиту подій та інструменту розслідування інцидентів.

У результаті всіх реалізованих заходів досягнуто багаторівневий захист веб-додатку, який охоплює серверну інфраструктуру, код, базу даних, мережу, процеси розробки та спостережуваність системи. Такий комплексний підхід мінімізує ймовірність компрометації системи та відповідає сучасним стандартам безпеки веб-додатків.

		Лашілін А.С.			ЖИТОМИРСЬКА ПОЛІТЕХНІКА.26.125.9	Арк.
		Воротніков В. В				54
Змн.	Арк.	№ докум.	Підпис	Дата		

РОЗДІЛ 3 СПРОБИ РЕАЛІЗАЦІЇ ВРАЗЛИВОСТЕЙ

3.1 SQL-ін'єкції

SQL-ін'єкція (SQLi) — одна з найпоширеніших вразливостей веб-додатків, що посідає перше місце в рейтингу OWASP Top 10. Суть атаки полягає у вставці довільного SQL-коду в параметри запиту з метою маніпуляції логікою бази даних: обходу аутентифікації, витоку даних або їх знищення.

У даному проєкті основним засобом захисту є SQLAlchemy ORM — бібліотека об'єктно-реляційного відображення, яка автоматично параметризує всі запити до бази даних. Параметризація означає, що вхідні дані користувача ніколи не конкатенуються безпосередньо з SQL-рядком, а передаються як окремі параметри — драйвер бази даних обробляє їх як літеральні значення, а не як частину SQL-синтаксису.

Тест 1. Класична обхідна ін'єкція в полі логіну

Мета: перевірити, чи можна обійти перевірку пароля, підставивши завжди-істинну умову.

Якби запит формувався конкатенацією рядків, наступний payload змусив би БД повернути першого користувача незалежно від пароля:

```
username: ' OR 1=1 --  
password: anything
```

Username
' OR 1=1 --

Password
•••••

SUBMIT

☐ Register

User not found. Please register.

Рисунок 3.1 – Спроба реалізації загрози

		Лашілін А.С.			ЖИТОМИРСЬКА ПОЛІТЕХНІКА.26.125.9	Арк.
		Воротніков В. В				55
Змн.	Арк.	№ докум.	Підпис	Дата		

Пояснення результату: SQLAlchemy сформував параметризований запит:

```
"""Read one entry of a specified model by a specified field
from db."""
async with session_local() as session:
    query = select(self.alchemy_model).where(
        getattr(self.alchemy_model, property_name) ==
        property_value,
    )
    result = await session.execute(query)
    return result.scalars().first()
```

Рядок 'OR 1=1 -- був переданий у базу як звичайне значення поля username. Жоден користувач із таким іменем не існує, тому ORM повернув порожній результат, і FastAPI відповів стандартною помилкою аутентифікації.

Тест 2. Ін'єкція DROP TABLE

Мета: перевірити, чи можна виконати DDL-команду через поле вводу.

```
> curl -s -X POST 'http://localhost:8000/users/login' \
-H 'accept: application/json' \
-H 'Content-Type: application/x-www-form-urlencoded' \
-d "grant_type=password&username=%27%3B+DROP+TABLE+users%3B+--&password=x&scope=&client_id=string&client_secret=string"
{"detail": "Resource(s) not found"}
~ ~ |
```

Рисунок 3.2 – Спроба реалізації загрози

Пояснення результату: SQLAlchemy використовує підготовлені вирази (prepared statements). Команда DROP TABLE не потрапила до виконання — весь рядок був інтерпретований виключно як значення відповідного параметра.

Висновок: жодна зі спроб SQL-ін'єкції не призвела до несанкціонованого доступу або модифікації бази даних. Двошаровий захист — валідація типів Pydantic на вході та параметризовані запити SQLAlchemy — суттєво знижує ризик реалізації цього класу вразливостей. Слід зазначити, що захист від SQL-ін'єкцій не є абсолютним і залежить від коректного використання ORM у всіх частинах коду: застосування сирих SQL-запитів без параметризації може відновити вразливість.

3.2 CSRF (Cross-Site Request Forgery)

CSRF (Cross-Site Request Forgery) — атака, за якої зловмисник змушує браузер жертви надіслати HTTP-запит до цільового сервера від її імені, використовуючи автоматично прикріплені облікові дані — як правило, сесійні cookie. Класична схема

		Лашілін А.С.			ЖИТОМИРСЬКА ПОЛІТЕХНІКА.26.125.9	Арк.
		Воротніков В.В				
Змн.	Арк.	№ докум.	Підпис	Дата		56

атаки виглядає наступним чином: жертва авторизується на легітимному сайті, браузер зберігає сесійну cookie; жертва переходить на шкідливий сайт зловмисника; сторінка у фоні надсилає запит до цільового API; браузер автоматично додає cookie до запиту, і сервер сприймає його як легітимний.

У даному проєкті аутентифікація реалізована на основі JWT Bearer-токена, який передається як query-параметр `jwt_token` у URL запиту. Це архітектурне рішення принципово унеможливлює CSRF, оскільки браузер ніколи не додає query-параметри до запитів автоматично — на відміну від cookie.

Тест 1. Запит до захищеного ендпоінту без токена

Імітується ситуація, коли шкідливий сайт намагається звернутись до API без токена жертви.

```
curl -si "http://localhost:8000/users/me"
```

```
> curl -si 'http://localhost:8000/users/me'
HTTP/1.1 422 Unprocessable Entity
date: Thu, 07 May 2026 09:09:31 GMT
server: uvicorn
content-length: 95
content-type: application/json

{"detail":[{"type":"missing","loc":["query","jwt_token"],"msg":"Field required","input":null}]}%
~>|
```

Рисунок 3.3 – Спроба реалізації загрози

Pydantic відхилив запит ще до реалізації будь-якої бізнес-логіки — поле `jwt_token` є обов'язковим query-параметром, і без нього запит не проходить валідацію схеми.

Тест 2. Запит з підробленим токеном

JWT має унікальну структуру що визначається у коді додатку. Якщо зловмиснику, якимось чином, вдалось викрасти структуру токена — потенційну атаку унеможливлює секретний ключ, прив'язаний до нього.

		Лашілін А.С.			ЖИТОМИРСЬКА ПОЛІТЕХНІКА.26.125.9	Арк.
		Воротніков В. В				
Змн.	Арк.	№ докум.	Підпис	Дата		57

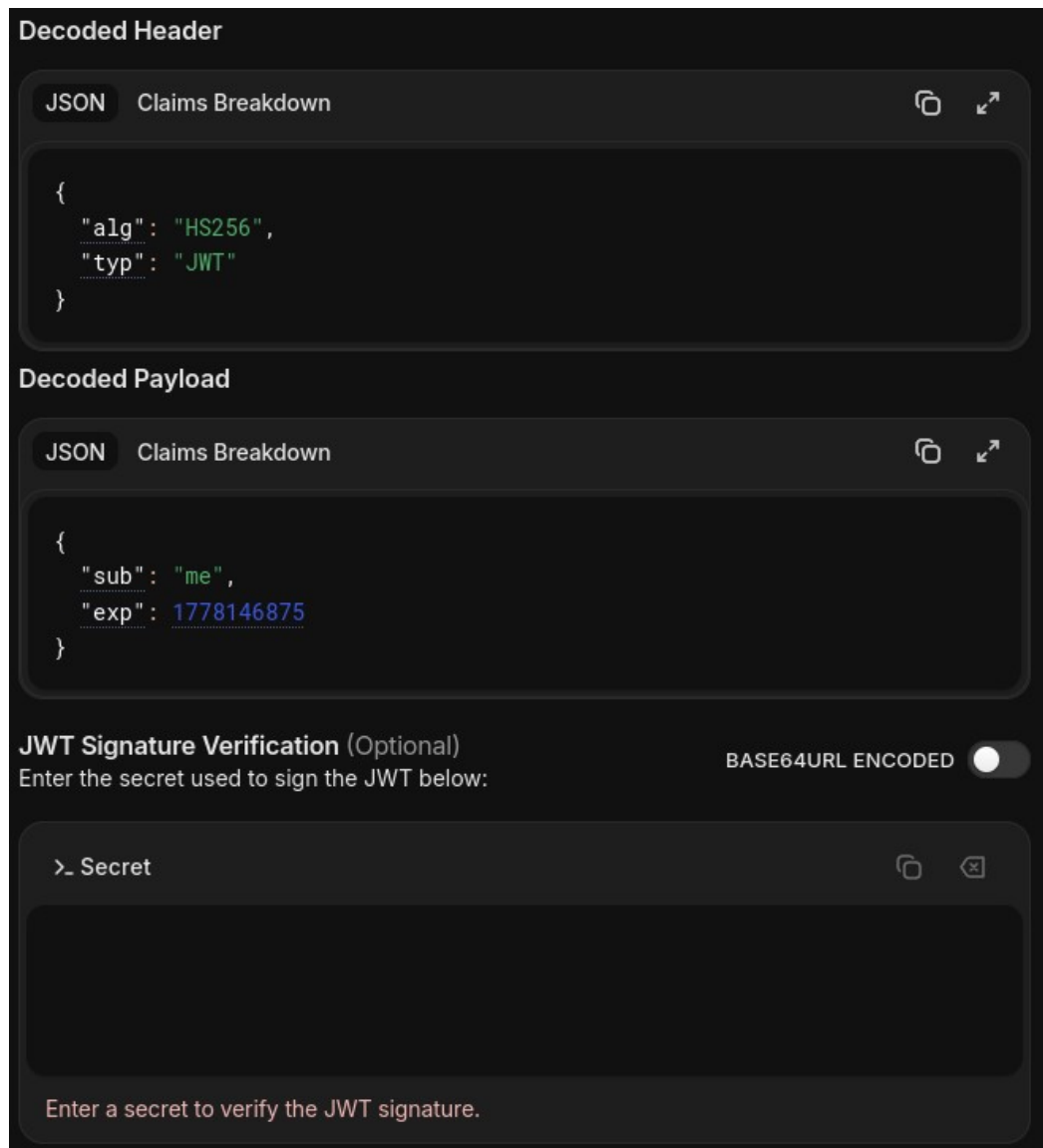


Рисунок 3.4 – Невалідний підпис

Баз валідного secret в зломисника жодних шансів отримати доступ до акаунту користувача.

3.3 Brute-Force

Брутфорс-атака на ендпоінт аутентифікації полягає у автоматизованому переборі паролів з метою отримання несанкціонованого доступу до облікового запису. За відсутності обмежень на кількість запитів зломисник може надсилати тисячі спроб входу за короткий проміжок часу, використовуючи словникові атаки або перебір за маскою. У даному проєкті захист реалізований на двох рівнях. На рівні застосунку — алгоритм хешування вступ природно уповільнює перевірку кожного пароля. На рівні

		Лашілін А.С.			ЖИТОМИРСЬКА ПОЛІТЕХНІКА.26.125.9	Арк.
		Воротніков В. В				58
Змн.	Арк.	№ докум.	Підпис	Дата		

Nginx — директива `limit_req` обмежує кількість запитів з однієї IP-адреси до 5 на хвилину для ендпоінту `/users/login`.

```
limit_req_zone $binary_remote_addr zone=login:10m rate=5r/m;

location /users/login {
    limit_req zone=login burst=3 nodelay;
    limit_req_status 429;
    proxy_pass http://api:8000;
}
```

Директива `limit_req_zone` створює зону пам'яті розміром 10 МБ для відстеження IP-адрес з лімітом 5 запитів на хвилину. Параметр `burst=3` дозволяє короточасний сплеск до 3 додаткових запитів, після чого всі наступні запити блокуються з кодом 429.

Тест 1. Порівняння пропускної здатності захищеного і незахищеного ендпоінту

Для встановлення базового показника спочатку виміряно пропускну здатність публічного ендпоінту `/books/` без rate limiting за допомогою інструменту `wrk`:

```
wrk -t4 -c50 -d15s http://localhost:80/books/
```

```
> wrk -t4 -c50 -d15s http://localhost:8000/books/
Running 15s test @ http://localhost:8000/books/
 4 threads and 50 connections
   Thread Stats   Avg      Stdev     Max    +/-  Stdev
   Latency    91.05ms   51.08ms  563.23ms   78.93%
   Req/Sec   139.95    30.67   210.00    75.64%
 8220 requests in 15.01s, 3.03MB read
Requests/sec:   547.53
Transfer/sec:   206.93KB
~ 15s ) |
```

Рисунок 3.5 – Спроба реалізації загрози

Ендпоінт без обмежень обробив 547 запитів на секунду.

Тест 2. Брутфорс-атака на ендпоінт логіну

Надіслано 10 послідовних запитів з невірним паролем на ендпоінт `/users/login`:

```
for i in {1..10}; do
    curl -s -o /dev/null -w "%{http_code}\n" \
-X POST http://localhost:80/users/login \
-H "Content-Type: application/x-www-form-urlencoded" \
```

		Лашілін А.С.			ЖИТОМИРСЬКА ПОЛІТЕХНІКА.26.125.9	Арк.
		Воротніков В.В				59
Змн.	Арк.	№ докум.	Підпис	Дата		

```
-d "grant_type=password&username=me&password=wrong&scope=&
client_id=string&client_secret=string"
done
```

```
) for i in {1..10}; do
  curl -s -o /dev/null -w "%{http_code}\n" \
    -X POST http://localhost:80/users/login \
    -H "Content-Type: application/x-www-form-urlencoded" \
    -d "grant_type=password&username=me&password=wrong&scope="
done
401
401
401
401
429
429
429
429
429
429
~ ) |
```

Рисунок 3.6 – Спроба реалізації загрози

Перші 4 запити досягли FastAPI і отримали відповідь 401 Unauthorized — невірний пароль, але запит оброблено. Починаючи з 5-го запиту Nginx заблокував усі наступні спроби з кодом 429 Too Many Requests, навіть не передаючи їх до застосунку.

Висновки: налаштування rate limiting на рівні Nginx ефективно нейтралізує брутфорс-атаки на ендпоінт аутентифікації. При перевищенні ліміту в 5 запитів на хвилину Nginx блокує подальші спроби з кодом 429, не передаючи їх до застосунку — це знижує навантаження на сервер і унеможливорює автоматизований перебір паролів. У поєднанні з bcrypt-хешуванням, яке природно обмежує швидкість перевірки до 2-3 запитів на секунду навіть без Nginx, система забезпечує надійний дворівневий захист від атак на автентифікацію.

3.4 IDOR (Insecure direct object references)

IDOR — це вразливість класу «Broken Access Control», що виникає, коли додаток надає прямий доступ до об'єктів (записів БД, файлів) на основі вхідних даних

		Лашілін А.С.			ЖИТОМИРСЬКА ПОЛІТЕХНІКА.26.125.9	Арк.
		Воротніков В. В				60
Змн.	Арк.	№ докум.	Підпис	Дата		

користувача без належної перевірки прав доступу. Класичний сценарій атаки передбачає зміну числового ідентифікатора в URL (наприклад, з /users/10 на /users/11) для несанкціонованого перегляду чужого профілю.

У даному проєкті архітектура ендпоінту GET /users/me побудована таким чином, що цей клас атак є апіорі неможливим.

```
"/users/me": {
  "get": {
    "tags": [
      "users"
    ],
    "summary": "Get current user.",
    "description": "Get current user as a schema.",
    "operationId": "get_me_users_me_get",
    "parameters": [
      {
        "name": "jwt_token",
        "in": "query",
        "required": true,
        "schema": {
          "type": "string",
          "title": "Jwt Token"
        }
      }
    ]
  }
}
```

Відсутність вхідного вектора маніпуляції:

На відміну від традиційних систем, де ідентифікатор користувача (user_id) передається відкрито у шляху запиту, ендпоінт /users/me взагалі не приймає ID як параметр. Клієнт не має технічної можливості «вказати» серверу, чий профіль він хоче переглянути.

Ідентифікація через JWT:

Єдиним джерелом ідентифікації є поле sub (subject) всередині JWT-токена. Оскільки ідентифікатор витягується сервером безпосередньо з розшифрованого та верифікованого корисного навантаження (payload), користувач обмежений виключно власним обліковим записом.

		Лашілін А.С.			ЖИТОМИРСЬКА ПОЛІТЕХНІКА.26.125.9	Арк.
		Воротніков В.В.				61
Змн.	Арк.	№ докум.	Підпис	Дата		

Спроба підмінити ID всередині токена (наприклад, змінити "sub": "5" на "sub": "1") призводить до негайного порушення цілісності криптографічного підпису. Як було доведено у розділі 3.2 (Тест 2), сервер миттєво відхиляє такий токен із кодом 401 Unauthorized, оскільки зломисник не володіє секретним ключем сервера для створення валідного підпису під зміненими даними.

Уникнення ризику викрадення токена:

Аргумент про можливість використання чужого токена для доступу до даних не стосується вразливості IDOR, а належить до компрометації облікових даних. Проте, у даній системі викрадення токена є надскладним завданням:

1. Токен передається через зашифроване TLS-з'єднання (HTTPS).
2. Токен має обмежений термін дії, що мінімізує вікно можливостей для атаки.
3. Відсутність збереження токена в автоматично передаваних Cookie (як описано в розділі 3.2) виключає можливість його перехоплення через CSRF-сценарії.

Висновок: завдяки використанню абстрактного ендпоінту /users/me замість параметризованих шляхів типу /users/{id}, ризик реалізації вразливості IDOR у системі суттєво знижено на рівні архітектури. Процес ідентифікації є ізольованим і захищеним криптографічним підписом HS256, що робить спроби несанкціонованого доступу через підміну ідентифікаторів вкрай складними для реалізації.

Висновки до третього розділу

У третьому розділі проведено практичне тестування реалізованих механізмів безпеки шляхом спроб відтворення типових атак на веб-додаток.

Спроби SQL-ін'єкцій не призвели до несанкціонованого доступу або модифікації бази даних. Використання SQLAlchemy ORM та валідація вхідних даних засобами Pydantic суттєво знижують ризик реалізації атак цього класу. При тестуванні CSRF було підтверджено, що архітектурне рішення на базі JWT Bearer-токенів, які передаються як query-параметр, унеможливорює класичну схему CSRF-атаки. Спроба brute-force на ендпоінт автентифікації була заблокована Nginx починаючи з п'ятого запиту (код 429 Too Many Requests), що підтверджує ефективність налаштованого rate limiting. Ризик реалізації вразливості IDOR суттєво знижений завдяки архітектурному рішенню —

		Лашілін А.С.			ЖИТОМИРСЬКА ПОЛІТЕХНІКА.26.125.9	Арк.
		Воротніков В.В.				
Змн.	Арк.	№ докум.	Підпис	Дата		62

використанню ендпоінту /users/me з ідентифікацією виключно через верифікований JWT-токен.

Усі тестування проводились переважно вручну з використанням інструментів curl та wrk. Для посилення практичної частини роботи перспективним є застосування спеціалізованих інструментів безпекового тестування, таких як OWASP ZAP, Nmap або Trivy, що дозволило б провести більш глибокий автоматизований аналіз захищеності системи.

		Лашілін А.С.			ЖИТОМИРСЬКА ПОЛІТЕХНІКА.26.125.9	Арк.
		Воротніков В. В				63
Змн.	Арк.	№ докум.	Підпис	Дата		

ВИСНОВКИ

У ході виконання випускної кваліфікаційної роботи було розроблено та досліджено систему безпеки вебдодатку на базі фреймворку FastAPI у середовищі Docker з використанням сучасного DevSecOps-підходу. За результатами проведеної роботи можна зробити наступні висновки:

Обраний стек технологій (FastAPI, SQLAlchemy, PostgreSQL, Nginx) дозволив створити високоефективну та масштабовану систему. Використання Docker та Docker Compose забезпечило повну ізоляцію компонентів додатка, що є критично важливим для мінімізації поверхні атаки та спрощення процесу розгортання.

Впроваджена система автентифікації на основі JWT-токенів з використанням алгоритму HS256 забезпечила надійний контроль доступу. Використання ORM SQLAlchemy дозволило автоматично параметризувати SQL-запити, що на рівні архітектури нівелювало ризики SQL-ін'єкцій.

Налаштування зворотного проксі-сервера Nginx дозволило реалізувати додаткові ешелони захисту, зокрема обмеження інтенсивності запитів (Rate Limiting) для протидії брутфорс-атакам та впровадження HTTP-заголовків безпеки (X-Frame-Options, X-Content-Type-Options, CSP). Це дозволило захистити клієнтську частину додатка від атак класу Clickjacking та XSS.

Проведене практичне тестування (Penetration Testing) підтвердило стійкість системи до найбільш розповсюджених загроз з рейтингу OWASP Top 10. Експериментально доведено ефективність захисту від SQLi, CSRF та IDOR. Зокрема, архітектурне рішення використання ендпоінту /users/me у поєднанні з криптографічним підписом JWT зробило маніпуляцію ідентифікаторами неможливою.

Інтеграція перевірок безпеки на кожному етапі розробки та розгортання дозволила створити систему, що відповідає принципу «Secure by Design». Автоматизація контролю трафіку та приховування технічної інформації про сервер (server tokens) значно підвищили загальний рівень захищеності інфраструктури.

		Лашілін А.С.			ЖИТОМИРСЬКА ПОЛІТЕХНІКА.26.125.9	Арк.
		Воротніков В.В.				
Змн.	Арк.	№ докум.	Підпис	Дата		64

Таким чином, розроблений проєкт демонструє комплексний підхід до кібербезпеки сучасних вебдодатків, поєднуючи надійні програмні практики з грамотним адмініструванням серверної інфраструктури, що дозволяє мінімізувати ризики витоку даних та порушення працездатності системи в умовах реальних кіберзагроз.

Разом з тим, реалізована система має певні обмеження та напрями для подальшого вдосконалення. Зокрема, у поточній версії відсутня інтеграція спеціалізованих інструментів автоматизованого аналізу безпеки (SAST/DAST) в CI/CD-пайплайн, а також не реалізовано Web Application Firewall (WAF) та механізми виявлення вторгнень (IDS/IPS). Тестування проводилось переважно вручну без застосування таких інструментів, як OWASP ZAP, Trivy чи Nmap. Крім того, конфігурація деяких сервісів є демонстраційною і потребує адаптації для production-середовища. Подальша робота може бути спрямована на інтеграцію SAST-сканування залежностей у пайплайн, налаштування WAF на рівні Nginx, а також розширення покриття тестами безпеки з використанням спеціалізованих інструментів.

		Лашілін А.С.			ЖИТОМИРСЬКА ПОЛІТЕХНІКА.26.125.9	Арк.
		Воротніков В. В				65
Змн.	Арк.	№ докум.	Підпис	Дата		

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. Столлінгс В. Криптографія та безпека мереж. Принципи і практика. – Київ: Видавнича група BHV, 2019. – 784 с.
2. Таненбаум Е., Везерол Д. Комп'ютерні мережі. – 5-те вид. – Київ: Діалектика, 2018. – 960 с.
3. Burns B., Grant B., Oppenheimer D. Designing Distributed Systems. – O'Reilly Media, 2018. – 198 p.
4. Kim G., Humble J., Debois P., Willis J. The DevOps Handbook. – IT Revolution Press, 2016. – 480 p.
5. OWASP Foundation. OWASP Top 10 Web Application Security Risks. – OWASP, 2023.
6. OWASP Foundation. OWASP Application Security Verification Standard (ASVS). – OWASP, 2023.
7. Jose J. JSON Web Token (JWT) Specification (RFC 7519). – IETF, 2015.
8. Rescorla E. The Transport Layer Security (TLS) Protocol (RFC 8446). – IETF, 2018.
9. FastAPI Documentation. FastAPI – Modern, Fast Web Framework for Python. – Official documentation.
10. PostgreSQL Global Development Group. PostgreSQL Documentation. – Official documentation.
11. SQLAlchemy Documentation. SQLAlchemy 2.0 Documentation. – Official documentation. URL: <https://docs.sqlalchemy.org/en/20/>
12. Nginx, Inc. NGINX Documentation. – Official documentation.
13. Certbot Documentation. Certbot: EFF's ACME client. – Read the Docs. URL: <https://eff-certbot.readthedocs.io/en/stable/>
14. Docker Inc. Docker Documentation. – Official documentation.
15. Docker Inc. Docker Compose Documentation. – Official documentation. URL: <https://docs.docker.com/compose/>
16. GitLab Inc. GitLab CI/CD Documentation. – Official documentation.

		Лашілін А.С.			ЖИТОМИРСЬКА ПОЛІТЕХНІКА.26.125.9	Арк.
		Воротніков В. В				66
Змн.	Арк.	№ докум.	Підпис	Дата		

17. Prometheus Authors. Prometheus: Overview and Introduction. – Official documentation. URL: <https://prometheus.io/docs/introduction/overview/>
18. Grafana Labs. Grafana Documentation. – Official documentation.
19. Grafana Labs. Grafana Loki Documentation. – Official documentation. URL: <https://grafana.com/docs/loki/latest/>
20. Pytest Development Team. Pytest: helps you write better programs. – Official documentation. URL: <https://docs.pytest.org/en/stable/>

		Лашілін А.С.			ЖИТОМИРСЬКА ПОЛІТЕХНІКА.26.125.9	Арк.
		Воротніков В. В				67
Змн.	Арк.	№ докум.	Підпис	Дата		

ДОДАТКИ

		Лашілін А.С.			ЖИТОМИРСЬКА ПОЛІТЕХНІКА.26.125.9	Арк.
		Воротніков В. В				68
Змн.	Арк.	№ докум.	Підпис	Дата		

Grafana, Loki, Prometheus, Promtail

grafana/datasources.yaml

```

apiVersion: 1

datasources:
  - name: Prometheus
    type: prometheus
    access: proxy
    url: http://prometheus:9090
    isDefault: true
    editable: true

  - name: Loki
    type: loki
    access: proxy
    url: http://loki:3100
    editable: true

```

loki-config.yaml

```

auth_enabled: false

server:
  http_listen_port: 3100

ingester:
  lifecycler:
    ring:
      kvstore:
        store: inmemory
      replication_factor: 1
  chunk_idle_period: 5m
  chunk_retain_period: 30s
  max_chunk_age: 1h

schema_config:
  configs:
    - from: 2020-10-24
      store: boltdb-shipper
      object_store: filesystem
      schema: v11
      index:
        prefix: index_
        period: 24h

storage_config:
  boltdb_shipper:
    active_index_directory: /loki/index
    cache_location: /loki/boltdb-cache
  filesystem:
    directory: /loki/chunks

limits_config:

```

		Лашілін А.С.			ЖИТОМИРСЬКА ПОЛІТЕХНІКА.26.125.9	Арк.
		Воротніков В.В.				69
Змн.	Арк.	№ докум.	Підпис	Дата		

```

retention_period: 168h # 7 days
allow_structured_metadata: false

table_manager:
  retention_deletes_enabled: true
  retention_period: 168h # 7 days

query_scheduler:
  max_outstanding_requests_per_tenant: 2048

common:
  path_prefix: /tmp/loki

```

prometheus.yaml

```

global:
  scrape_interval: 3s

scrape_configs:
  - job_name: "fastapi"
    metrics_path: /metrics
    static_configs:
      - targets: ["api:8000"]

```

promtail-config.yaml

```

server:
  http_listen_port: 9080
  grpc_listen_port: 0

positions:
  filename: /tmp/positions.yaml

clients:
  - url: http://loki:3100/loki/api/v1/push

scrape_configs:
  - job_name: docker-logs
    static_configs:
      - targets:
          - localhost
        labels:
          job: docker-logs
          __path__: /var/lib/docker/containers/*/*log

  pipeline_stages:
    - json:
        expressions:
          stream: stream
          attrs: attrs
          tag: attrs.tag

    - regex:
        expression: (?P<image_name>(?:[^\s]*[^\s]))\.(?P<container_name>(?:[^\s]*[^\s]))\.(?P<image_id>(?:[^\s]*[^\s]))\.(?P<container_id>(?:[^\s]*[^\s]))

```

		Лашілін А.С.			ЖИТОМИРСЬКА ПОЛІТЕХНІКА.26.125.9	Арк.
		Воротніков В.В				70
Змн.	Арк.	№ докум.	Підпис	Дата		

source: "tag"

- labels:
tag:
image_name:
container_name:

		Лашілін А.С.			ЖИТОМИРСЬКА ПОЛІТЕХНІКА.26.125.9	Арк.
		Воротніков В. В				71
Змн.	Арк.	№ докум.	Підпис	Дата		

Docker, Docker Compose

Dockerfile

```
FROM python:3.13-slim

WORKDIR /app

COPY requirements.txt .
RUN pip install --no-cache-dir -r requirements.txt

COPY . .

CMD ["fastapi", "dev", "backend/main.py", "--host", "0.0.0.0"]
```

docker-compose.yaml

```
services:
  api:
    build: .
    volumes:
      - ./app
    #ports:
    # - "8000:8000"
    env_file:
      - ./env
    depends_on:
      - db

  db:
    image: postgres:16
    environment:
      POSTGRES_USER: postgres
      POSTGRES_PASSWORD: root
      POSTGRES_DB: books_db
    #ports:
    # - "5432:5432"
    volumes:
      - postgres_data:/var/lib/postgresql/data

  promtail:
    image: grafana/promtail:latest
    container_name: promtail
    #ports:
    # - "9080:9080"
    volumes:
      - /var/lib/docker/containers:/var/lib/docker/containers:ro
      - /var/run/docker.sock:/var/run/docker.sock
      - ./promtail-config.yaml:/etc/promtail/config.yaml
    command: -config.file=/etc/promtail/config.yaml
    privileged: true

  loki:
    image: grafana/loki:latest
    container_name: loki
```

		Лашілін А.С.			ЖИТОМИРСЬКА ПОЛІТЕХНІКА.26.125.9	Арк.
		Воротніков В.В				72
Змн.	Арк.	№ докум.	Підпис	Дата		

```

#ports:
# - "3100:3100"
volumes:
  - ./loki-config.yaml:/etc/loki/local-config.yaml

prometheus:
  image: prom/prometheus:latest
  container_name: prometheus
  #ports:
  # - "9090:9090"
  volumes:
    - ./prometheus.yml:/etc/prometheus/prometheus.yml
    - prometheusdata:/prometheus

grafana:
  image: grafana/grafana:latest
  container_name: grafana
  ports:
    - "3000:3000"
  environment:
    - GF_SECURITY_ADMIN_USER=admin
    - GF_SECURITY_ADMIN_PASSWORD=admin
  volumes:
    -
./grafana/datasources.yaml:/etc/grafana/provisioning/datasources/datasources.yaml
    - grafanadata:/var/lib/grafana

nginx:
  image: nginx:latest
  container_name: nginx
  ports:
    - "80:80"
  volumes:
    - ./nginx.conf:/etc/nginx/nginx.conf:ro
  depends_on:
    - api

volumes:
  postgres_data:
  grafanadata:
  prometheusdata:

```

		Лашілін А.С.			ЖИТОМИРСЬКА ПОЛІТЕХНІКА.26.125.9	Арк.
		Воротніков В. В				73
Змн.	Арк.	№ докум.	Підпис	Дата		

Nginx

nginx.conf

```

events {
    worker_connections 1024;
}

http {
    # Hide Nginx version number in error pages and Server header
    server_tokens off;

    # Rate limiting configuration: 5 requests per minute per IP
    limit_req_zone $binary_remote_addr zone=login:10m rate=5r/m;

    server {
        listen 80;

        # Prevent Clickjacking by forbidding the page from being
        rendered in an iframe
        add_header X-Frame-Options "DENY" always;
        # Prevent the browser from interpreting files as a different
        MIME type
        add_header X-Content-Type-Options "nosniff" always;
        # Enable XSS filtering in modern web browsers
        add_header X-XSS-Protection "1; mode=block" always;
        # Control how much referrer information is passed during
        navigation
        add_header Referrer-Policy "strict-origin-when-cross-origin"
        always;
        # Content Security Policy: allows scripts/styles from self
        and handles NiceGUI requirements
        add_header Content-Security-Policy "default-src 'self';
        script-src 'self' 'unsafe-inline'; style-src 'self' 'unsafe-inline';
        img-src 'self' data:; connect-src 'self' ws: wss:;" always;

        # Deny access to sensitive metrics endpoints
        location /metrics {
            deny all;
            return 403;
        }

        # Brute-force protection for the login endpoint
        location /users/login {
            limit_req zone=login burst=3 nodelay;
            limit_req_status 429;

            proxy_pass http://api:8000;
            proxy_set_header Host $host;
            proxy_set_header X-Real-IP $remote_addr;
            proxy_set_header X-Forwarded-For
            $proxy_add_x_forwarded_for;
        }

        # WebSocket support for NiceGUI interaction
    }
}

```

		Лашілін А.С.			ЖИТОМИРСЬКА ПОЛІТЕХНІКА.26.125.9	Арк.
		Воротніков В.В				74
Змн.	Арк.	№ докум.	Підпис	Дата		

```

location /v1/_nicegui_ws/ {
    proxy_pass http://api:8000;
    proxy_http_version 1.1;
    proxy_set_header Upgrade $http_upgrade;
    proxy_set_header Connection "upgrade";
    proxy_set_header Host $host;
    proxy_read_timeout 3600s;
}

# Default proxy location for the application
location / {
    proxy_pass http://api:8000;
    proxy_set_header Host $host;
    proxy_set_header X-Real-IP $remote_addr;
    proxy_set_header X-Forwarded-For
$proxy_add_x_forwarded_for;
    }
}

```

		Лашілін А.С.			ЖИТОМИРСЬКА ПОЛІТЕХНІКА.26.125.9	Арк.
		Воротніков В. В				75
Змн.	Арк.	№ докум.	Підпис	Дата		