МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ПОЛІСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

Факультет інформаційних технологій, обліку та фінансів
Кафедра комп'ютерних технологій
і моделювання систем

Кваліфікаційна робота
на правах рукопису

Гінчук В'ячеслав Олександрович
(прізвище, ім'я, по батькові здобувача освіти)

УДК 004.056:004.438

КВАЛІФІКАЦІЙНА РОБОТА

Методи забезпечення кібербезпеки API серверного застосунку
(тема роботи)

125 «Кібербезпека та захист інформації»
(шифр і назва спеціальності)

Подається на здобуття освітнього ступеня магістр

кваліфікаційна робота містить результати власних досліджень. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело

_____
(підпис, ініціали та прізвище здобувача вищої освіти)

Керівник роботи
Николюк Ольга Миколаївна
(прізвище, ім'я, по батькові)
д.н економічних наук, професор
(науковий ступінь, вчене звання)

Житомир – 2024

Висновок кафедри _____
за результатами попереднього захисту: _____

Протокол засідання кафедри _____
№ ____ від «_____» _____ 20____ р.

Завідувач кафедри _____

_____ _____ _____
(науковий ступінь, вчене звання) (підпис) (прізвище, ім'я, по батькові)
«_____» _____ 20____ р.

Результати захисту кваліфікаційної роботи

Здобувач вищої освіти _____ захистив (ла)
(прізвище ,ім'я, по батькові)
кваліфікаційну роботу з оцінкою:

сума балів за 100-бальною шкалою _____
за шкалою ECTS _____
за національною шкалою _____

Секретар ЕК

_____ _____ _____
(науковий ступінь, вчене звання) (підпис) (прізвище, ім'я, по батькові)

АНОТАЦІЯ

Гінчук В. О. Методи забезпечення кібербезпеки API серверного застосунку. – Кваліфікаційна робота на правах рукопису.

Кваліфікаційна робота на здобуття освітнього ступеня магістр за спеціальністю 125 «Кібербезпека та захист інформації» – Поліський національний університет, Житомир, 2024.

У кваліфікаційній роботі досліджено методи захисту API для забезпечення безпеки серверних застосунків. Метою роботи є дослідження загроз API та визначення ефективних методів захисту серверних застосунків, таких як валідація даних, хешування паролів, використання HTTPS і обмеження частоти запитів.

Ключові слова: API, Кібербезпека, Методи захисту.

SUMMARY

Hinchuk V.O. Methods for Ensuring Cybersecurity of API in Server Applications. – Qualification Thesis Manuscript.

Qualification thesis for obtaining a Master's degree in the specialty 125 "Cybersecurity and Information Protection" – Polissia National University, Zhytomyr, 2024.

The qualification work explores methods for protecting APIs to ensure the security of server applications. The goal was to study API threats and identify effective methods for securing server applications, such as data validation, password hashing, the use of HTTPS, and rate limiting.

Keywords: API, Cybersecurity, Security methods.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

API – Application Programming Interface

REST – Representational State Transfer

SQL – Structured Query Language

BOLA – Broken Object Level Authorization

DoS – Denial of Service

DDoS – Distributed Denial of Service

MITM – Man in the Middle

2FA – Two factor authentication

JWT – Json Web Token

HTTP – HyperText Transfer Protocol

HTTPS – HyperText Transfer Protocol Secure

SSL – Secure Sockets Layer

TLS – Transport Layer Security

DTO – Data Transfer Object

ВСТУП

В умовах стрімкого розвитку інформаційних технологій, API (інтерфейс програмування застосунків) стала ключовим елементом для взаємодії між різними компонентами програмного забезпечення, дозволяючи ефективний обмін даними і забезпечуючи роботу численних сервісів. Зі збільшенням використання API зростають і ризики безпеки, оскільки саме через API часто здійснюються спроби атак на сервери та доступ до конфіденційної інформації [1].

У зв’язку з цим питання захисту API набуває особливої актуальності. Використання належних методів безпеки дозволяє мінімізувати ризики несанкціонованого доступу, зловживань, атак на систему та крадіжки даних. У даній роботі основна увага приділена опису різних методів захисту API, які можуть бути застосовані для підвищення рівня безпеки серверних застосунків.

Метою роботи є дослідження загроз API та визначення ефективних методів захисту серверних застосунків, таких як валідація даних, хешування паролів, використання HTTPS і обмеження частоти запитів.

Предмет дослідження – методи забезпечення кібербезпеки API, їх реалізація за допомогою популярних бібліотек.

Об'єкт дослідження – процесс забезпечення кібербезпеки API.

Наукова новизна. Дістала подальшого розвитку технологія імплементації методів забезпечення кібербезпеки та тестування безпеки API із кількісним оцінюванням рівня захисту API за умови використання окремих методів захисту.

Завдання роботи:

- Дослідити API та її основні типи (REST, GraphQL, WebSocket).
- Проаналізувати основні вразливості API,
- Дослідити методи забезпечення захисту API.
- Розробити практичні рекомендації щодо комплексного використання методів забезпечення кібербезпеки з метою підвищення рівня захищеності API

Дана робота покликана систематизувати існуючі методи захисту API та запропонувати рекомендації щодо їх застосування, з метою ефективної протидії

потенційним загрозам, виявленим вразливостям та пом’якшенню наслідків кіберінциндентів.

За результатами дослідження було прийнято участь у трьох наукових конференціях:

- Здобутки та досягнення прикладних та фундаментальних наук XXI століття (22 листопада 2024) – “ Аналіз сутності прикладних програмних інтерфейсів ” [2]
- Науковий простір: актуальні питання, досягнення та інновації (29 листопада 2024) – “Основні види прикладних програмних інтерфейсів” [3]
- Інноваційна наука: пошук відповідей на виклики сучасності (6 грудня 2025) – “ Загрози безпеці прикладних програмних інтерфейсів” [4]

РОЗДІЛ 1. АНАЛІЗ API ТА НЕОБХІДНІСТЬ ЇХ ЗАХИСТУ

1.1 Визначення API та їх роль у серверних застосунках

API (Application Programming Interface) – це інтерфейс програмування додатків, що визначає правила взаємодії між різними програмними компонентами [5]. API виступає посередником між клієнтськими запитами та серверною логікою, дозволяючи здійснювати обмін даними та виклик функцій без необхідності знання внутрішньої реалізації серверного застосунку. API є особливо важливим у сучасних веб-застосунках, де сервери обробляють величезні обсяги запитів від різних клієнтів, зокрема мобільних додатків, веб-браузерів і інших серверних систем.

У серверних застосунках API виконує ключову роль у забезпеченні взаємодії між клієнтською (Front-end) та серверною (Back-end) частинами. Основними функціями API є:

- Абстрагування бізнес-логіки: API приховує деталі реалізації серверної логіки, надаючи клієнтам лише необхідні методи для доступу до даних та функцій. Це дозволяє змінювати внутрішню структуру застосунку без порушення роботи клієнтів, що підвищує гнучкість системи.
- Модульність і повторне використання: За допомогою API можна створювати окремі модулі, які легко інтегруються у більші системи. Це сприяє повторному використанню компонентів, що скорочує час розробки та зменшує вартість підтримки.
- Забезпечення безпеки та контролю доступу: API часто використовують для реалізації механізмів аутентифікації та авторизації, що дозволяє контролювати доступ до ресурсів і захищати сервер від несанкціонованих запитів.

API також полегшує інтеграцію з зовнішніми сервісами та сторонніми бібліотеками [6]. Це особливо важливо у випадках, коли серверний застосунок потребує даних від інших систем, наприклад, для обробки платежів, відправлення електронної пошти чи взаємодії з хмарними сервісами. За допомогою API сервер

може отримувати необхідні дані та обробляти їх, використовуючи стандартизовані протоколи обміну, такі як HTTP або HTTPS.

Однією з ключових причин популярності API є стандартизація підходів до обміну даними [7]. Це забезпечує однаковий спосіб роботи з різними сервісами, що значно спрощує розробку клієнтських застосунків. Завдяки використанню API можна легко інтегрувати нові функції та сервіси без необхідності повної переробки існуючої інфраструктури.

Крім того, стандартизація дозволяє легко документувати API, що важливо для розробників, які використовують сторонні сервіси або працюють над великими проектами з багатьма командами. Наявність чіткої документації API спрощує процес ознайомлення з функціональністю сервісу та скорочує час на вирішення технічних питань.

API забезпечує стабільний зв’язок між різними компонентами застосунків, полегшує масштабування системи та підтримує можливість додавання нових функцій [8]. Це робить його невід'ємною частиною сучасних серверних архітектур, особливо у мікросервісних системах, де окремі компоненти часто взаємодіють через стандартизовані API.

З точки зору кібербезпеки API є критичним компонентом, який часто стає об'єктом атак, оскільки забезпечує доступ до важливих даних і функцій серверного застосунку. Кібервластивості API визначаються їх здатністю забезпечувати захист конфіденційності, цілісності та доступності даних у процесі взаємодії між клієнтами та сервером.

1.2 Основні види API

Існує кілька популярних типів API, які широко використовуються в серверних застосунках для забезпечення обміну даними між клієнтською та серверною частинами. Найбільш розповсюдженими є REST, GraphQL та WebSocket. Кожен з цих видів API має свої особливості, які визначають їхню область застосування та ефективність у різних ситуаціях.

1. REST (Representational State Transfer) – це архітектурний стиль для створення веб-сервісів, який базується на використанні HTTP-протоколу [9]. REST API забезпечує інтерфейс для обміну даними, де кожен ресурс ідентифікується унікальним URL. Основні характеристики REST API:

- Використання HTTP методів: REST API використовує стандартні HTTP методи, такі як GET (отримати дані), POST (створити дані), PUT (оновити дані), DELETE (видалити дані). Це забезпечує зрозумілий та стандартизований спосіб взаємодії з даними.
- Контекстність: REST API є контекстним, тобто кожен запит містить усю необхідну інформацію для його виконання, без збереження стану між запитами на сервері.
- Формати даних: Зазвичай REST API використовує JSON для передачі даних, що робить його зручним для роботи у веб-застосунках.
- Кешування: REST API дозволяє кешувати відповіді, що знижує навантаження на сервер та підвищує швидкість обробки запитів.

REST API підходить для побудови масштабованих та простих у використанні веб-сервісів, де необхідно працювати з ресурсами за допомогою стандартних HTTP методів. Однак при великій кількості запитів або складних запитах він може мати низьку продуктивність через необхідність надсилання множинних запитів.

2. GraphQL – це запитова мова для API, яка надає клієнтам можливість гнучко обирати, які саме дані їм потрібні від сервера [10]. Основні особливості GraphQL:

- Запити на основі потреб клієнта: Клієнт сам визначає структуру відповіді, запитуючи лише необхідні дані. Це зменшує обсяг переданих даних та підвищує ефективність.
- Єдиний ендпоінт: На відміну від REST API, де для різних ресурсів використовуються окремі URL, GraphQL API має один ендпоінт для всіх запитів. Це спрощує архітектуру API.

- Зниження кількості запитів: Оскільки клієнт може запитувати всі необхідні дані одним запитом, зменшується кількість викликів до сервера, що підвищує продуктивність.

GraphQL є хорошим вибором для складних застосунків з великою кількістю взаємозв'язаних даних, але він потребує ретельного проектування схеми API та додаткових заходів безпеки через можливість створення складних запитів.

3. WebSocket – це протокол для двонаправленого зв’язку між клієнтом і сервером у режимі реального часу [11]. На відміну від REST та GraphQL, які використовують HTTP, WebSocket встановлює постійне з’єднання між клієнтом і сервером. Основні особливості WebSocket:

- Двонаправлений зв’язок: Після встановлення з’єднання сервер та клієнт можуть обмінюватися даними в обидва боки без необхідності надсилати окремі запити.
- Низька затримка: Оскільки з'єднання залишається відкритим, відсутня потреба у повторних запитах на відкриття з’єднання, що знижує затримку передачі даних.
- Ефективність у режимі реального часу: WebSocket ідеально підходить для застосунків, де потрібен постійний обмін даними, таких як чати, онлайн-ігри та фінансові платформи.

WebSocket API забезпечує високу швидкість і ефективність у застосунках, де потрібна робота в реальному часі, проте він вимагає додаткового контролю з боку сервера та клієнта для обробки постійних з’єднань та забезпечення безпеки.

Таким чином, вибір виду API залежить від потреб застосунку, обсягу даних та вимог до продуктивності [12]. REST API підходить для простих сервісів, GraphQL – для складних запитів з великою кількістю взаємозв'язаних даних, а WebSocket – для обміну даними в реальному часі.

Таблиця 1 – Порівняння видів API

	REST	**GraphQL**	**WebSocket**
Протокол	HTTP	HTTP	WebSocket
Структура запитів	HTTP методи (GET, POST, PUT. DELETE, тощо)	Один ендпоінт, запит лише потрібних даних	Постійне двонаправлене з’єднання
Область застосування	Просте управління ресурсами	Складні запити з великим обсягом даних	Робота в реальному часі
Переваги	Простота налаштування	Гнучкість, зниження кількості запитів	Мінімальна затримка

1.3 Загрози безпеці API

Безпека API – це стан захищеності інтерфейсу програмування додатків, який забезпечує його коректне функціонування без деградації основних параметрів, таких як:

- Швидкість – час відповіді на запити.
- Пропускна здатність – обсяг запитів, які API може обробити за певний час.
- Конфіденційність – захист даних, які передаються через API, від несанкціонованого доступу.
- Цілісність – гарантія, що дані не змінюються чи пошкоджуються під час їхньої обробки.

Найпоширеніші загрози безпеці API:

1. Атаки, які впливають на пропускну здатність та швидкість (DoS/DDoS)

DoS (Denial of Service) та DDoS (Distributed Denial of Service) атаки націлені на те, щоб перевантажити API великою кількістю запитів, що робить сервіс недоступним для легітимних користувачів [13]. Зловмисник може використовувати

ботнети або інші засоби для генерації масивного потоку трафіку, що призводить до виснаження ресурсів сервера.

Наслідки:

- Недоступність сервісу для користувачів.
- Фінансові втрати через простої системи
- Погіршення репутації сервісу через низьку надійність.

2. Атаки, які впливають на цілісність даних (SQL/NoSQL Injection):

SQL Injection та NoSQL Injection – це атаки, при яких зловмисник вводить шкідливі SQL/NoSQL-запити в параметри API [14]. Це можливо, якщо серверна частина застосунку не здійснює належної валідації вхідних даних. Зловмисник може отримати доступ до бази даних, викрасти або змінити дані, а в деяких випадках навіть виконати шкідливий код на сервері.

Наслідки:

- Неавторизований доступ до бази даних та її вмісту.
- Зміна або видалення даних, що може порушити роботу сервісу.
- Виконання довільного коду на сервері, що веде до повного компрометування системи.

3. Атаки, які впливають на конфіденційність (BOLA та MITM)

BOLA (Broken Object Level Authorization) – це вразливість, яка виникає, коли API недостатньо перевіряє права доступу до конкретних об'єктів, таких як записи бази даних, файли або інші ресурси [15].

Наслідки:

- Неконтрольований витік даних.
- Витік конфіденційної інформації, яка може бути використана для подальших атак.
- Перевантаження сервера та зниження продуктивності системи.

Атака типу Man-in-the-Middle (MITM) відбувається, коли зловмисник перехоплює або змінює комунікацію між двома сторонами (клієнтом та сервером)

[16]. Оскільки API зазвичай передають дані через мережу, атаки MITM можуть стати серйозною загрозою, якщо з'єднання не захищене належним чином. Зловмисник може виступати як посередник, фальсифікуючи з'єднання між клієнтом і сервером, що дозволяє йому отримати доступ до чутливої інформації та маніпулювати даними.

Наслідки:

- Перехоплення конфіденційної інформації
- Модифікація даних

Висновок до першого розділу

У першому розділі було проведено аналіз інтерфейсів програмування застосунків (API), їх значення та основні типи (REST, GraphQL, WebSocket). Також було розглянуто ключові загрози безпеці, з якими стикаються API, серед яких brute-force атаки, SQL/NoSQL ін’єкції, атаки типу BOLA, DoS/DDoS атаки та атаки "людина посередині". Проведений аналіз підтвердив важливість захисту API та підкреслив необхідність застосування комплексних заходів безпеки для забезпечення цілісності, конфіденційності та доступності даних, що передаються через API.

РОЗДІЛ 2. МЕТОДИ ЗАХИСТУ API ВІД КІБЕРЗАГРОЗ

2.1 Валідація даних

Валідація даних є ключовим методом забезпечення безпеки API, особливо в контексті захисту від різноманітних атак, таких як SQL Injection. Це процес перевірки та фільтрації вхідних даних, що надходять від користувачів або інших систем, для запобігання внесенню шкідливих або некоректних даних в систему. Коректно реалізована валідація може значно знизити ризики зловживань та атак на серверні застосунки.

Підходи до валідації даних:

1. Перевірка типів даних. Це дозволяє уникнути ін'єкцій, а також некоректного поводження з даними на сервері.
2. Використання регулярних виразів. Це корисно, коли потрібно перевірити, чи відповідає введене значення певному шаблону, наприклад, для електронної пошти, телефонних номерів чи URL-адрес. Це дає змогу запобігти потраплянню даних, що можуть бути використані для атак.
3. Чорні списки та білі списки символів. Чорні списки використовують для блокування певних символів або конструкцій, які можуть бути використані в атаках (наприклад, SQL-оператори чи шкідливі скрипти). У той час як білі списки – це дозволені набори значень або символів. знаки, що виключає можливість ввести небажані або шкідливі символи.
4. Нормалізація даних. Це процес очищення вхідних даних перед їх використанням у системі. Наприклад, видалення зайвих пробілів або переведення тексту в нижній регістр для порівняння.
5. Ескейпінг даних. Для деяких типів даних, таких як текстові рядки, варто використовувати ескейпінг, тобто екранування спеціальних символів, що дозволяє уникнути виконання шкідливих інструкцій. Це важливо, коли дані передаються в запити, особливо в контексті баз даних або запитів до інших зовнішніх систем.

Рекомендації:

- Використовувати готові бібліотеки для валідації. Багато бібліотек для валідації даних, таких як joi [17], zod [18], class-validator [19] та інші, можуть значно полегшити процес перевірки введених даних і забезпечити їх коректність.
- Забезпечити валідацію на всіх рівнях. Валідація повинна проводитися не тільки на сервері, а й на стороні клієнта для покращення користувацького досвіду. Однак серверна валідація є обов'язковою, оскільки клієнтські перевірки можна обійти.
- Логування помилок валідації. Важливо вести журнал всіх помилок валідації, щоб аналізувати спроби зловживань або аномального введення даних.

2.2 Хешування паролів та керування аутентифікацією

Забезпечення безпеки паролів є одним із найважливіших аспектів захисту інформації в серверних застосунках. Хешування паролів та належне управління процесами аутентифікації дозволяють зменшити ризики витоків чутливої інформації та підвищити загальний рівень безпеки API.

Хешування є одностороннім процесом, при якому паролі користувачів перетворюються в рядки фіксованої довжини, що не можуть бути легко відновлені до вихідного значення (наприклад рядок 1234 може бути перетвореним на e2e610d6ac2aba290766c2ec2c68a74dfac776). Це дозволяє зберігати паролі в захищеному вигляді, навіть якщо база даних буде скомпрометована. Хешування є надійною технікою для захисту паролів, оскільки навіть з доступом до хешованих значень неможливо однозначно відновити сам пароль.

Основні кроки хешування паролів:

1. Використання криптографічних хеш-функцій. Сучасні алгоритми хешування, такі як bcrypt [20] або argon2 [21], забезпечують високу стійкість до атак на хешовані паролі.
2. Сольова обробка (salting) [22]. Для підвищення безпеки, кожен пароль слід хешувати з унікальним значенням солі, що запобігає атакам з використанням

заздалегідь підготовлених таблиць. Кожен користувач повинен мати свою сіль, яка генерується випадковим чином і зберігається разом із хешованим паролем.

3. Множинне хешування. Алгоритми дозволяють налаштовувати кількість ітерацій хешування, що підвищує складність атаки.

Аутентифікація є процесом перевірки особи користувача для доступу до захищених ресурсів API [23]. Усі механізми аутентифікації мають бути розроблені таким чином, щоб забезпечити максимальний рівень безпеки для користувачів та запобігти несанкціонованому доступу.

Основні підходи до аутентифікації:

1. Аутентифікація за паролем. Це найбільш традиційний метод аутентифікації, при якому користувач вводить свій пароль. Однак важливо, щоб сервер не зберігав паролі у відкритому вигляді, а лише хешовані значення. Також важливо обмежити кількість спроб введення неправильного пароля (механізм захисту від брутфорс-атак [24]).
2. Двофакторна аутентифікація (2FA) [25]. Для підвищення рівня безпеки часто використовують двоетапну аутентифікацію. Це додатковий рівень перевірки, де після введення пароля користувач отримує одноразовий код на свою електронну пошту або мобільний телефон. Він повинен ввести цей код для завершення процесу аутентифікації.
3. JSON Web Tokens (JWT) [26]. Це стандарт, що використовується для обміну інформацією між клієнтом і сервером. Після успішної аутентифікації користувача, сервер генерує токен (JWT), який містить закодовану інформацію про користувача. Цей токен можна передавати в заголовках HTTP-запитів для подальшої аутентифікації без необхідності повторного введення пароля.
4. OAuth 2.0 [27]. Цей протокол дозволяє авторизувати доступ до ресурсів користувача без необхідності передавати пароль стороннім сервісам. Це особливо корисно для інтеграцій між різними додатками (наприклад, при авторизації через Google або Facebook).

Популярні практики:

- Зберігання паролів у захешованому вигляді з унікальною сіллю.
- Використання сучасних та безпечних алгоритмів хешування.
- Обмеження кількості спроб аутентифікації.
- Забезпечення відправки токенів через HTTPS, щоб уникнути можливості перехоплення даних під час передачі.
- Введення політики використання складних паролів з регулярною зміною, або ж використання двофакторної аутентифікації.

2.3 Використання HTTPS

HTTPS (HyperText Transfer Protocol Secure) є розширенням стандартного HTTP-протоколу, що забезпечує захищену передачу даних через зашифроване з'єднання [28]. Використання HTTPS є обов'язковим для всіх сучасних веб-додатків і серверних API, оскільки він забезпечує конфіденційність і цілісність переданих даних, що є критичним для запобігання атакам типу "людина посередині" (MITM).

Популярні практики:

- Шифрування з'єднання через SSL/TLS [29]: Шифрування за допомогою протоколу SSL/TLS дозволяє захистити з'єднання, гарантуючи, що жодна стороння особа не зможе перехопити або змінити передані дані. SSL сертифікати повинні бути видані надійними сертифікаційними центрами (CA) і використовувати сучасні методи шифрування.
- Перевірка сертифікатів: Всі з'єднання через HTTPS повинні бути перевірені на відповідність серверу через валідацію SSL-сертифікатів. Якщо сертифікат серверу не відповідає вимогам або є недійсним, клієнт повинен бути попереджений про потенційну небезпеку.
- HSTS (HTTP Strict Transport Security): Цей заголовок HTTP змушує браузери завжди використовувати HTTPS для доступу до сервера. Якщо сервер підтримує HSTS, браузер автоматично перенаправляє всі HTTP-запити на HTTPS, запобігаючи MITM-атакам, що можуть виникнути через змішане з'єднання або зловмисний редирект.

- Автоматичне оновлення сертифікатів та алгоритмів шифрування: Постійне оновлення сертифікатів і використання новітніх алгоритмів шифрування є важливою частиною стратегії захисту від MITM-атак.

2.4 Обмеження частоти запитів

Rate Limiting – це техніка, яка обмежує кількість запитів, які користувач або клієнт може зробити до API за певний проміжок часу [30]. Вона є важливим інструментом захисту від декількох типів атак, таких як brute-force атаки, DoS (Denial of Service) та DDoS (Distributed Denial of Service) атаки. Оскільки API зазвичай відкриті для доступу з інтернету, зловмисники можуть спробувати перевантажити сервер, надсилаючи надмірну кількість запитів. Це може призвести до порушення роботи сервісу, зниження продуктивності або навіть до повної його недоступності.

Види обмежень частоти запитів:

- Ліміти на користувача. Обмеження кількості запитів на одиничного користувача або IP-адресу. Це допомагає уникнути атак на підбір паролів чи інших спроб автоматизованого доступу.
- Ліміти на API ключі. API може надавати ключі доступу, які пов'язані з певним користувачем або клієнтом. Ліміти можуть бути встановлені на кожен ключ доступу, що дозволяє управляти обсягом запитів для кожного користувача.
- Ліміти на рівень аккаунту або роль. Якщо API підтримує різні рівні доступу або ролі (наприклад, адміністратора, звичайного користувача), різним рівням доступу можна встановлювати різні обмеження на запити.

Популярні практики:

- Встановлення відкладеного блокування після досягнення ліміту.
- Моніторинг та логування запитів для виявлення аномальних активностей.
- Повідомлення користувачів при досягнення ліміту запитів.
- Використання кешування для зменшення навантаження на API.

Висновок до другого розділу

У другому розділі було розглянуто основні методи захисту API, зокрема валідація даних, хешування паролів, використання HTTPS та обмеження частоти запитів. Кожен з цих методів є ключовим для забезпечення безпеки API, мінімізації ризиків несанкціонованого доступу та зловживань. Детально описані популярні практики для впровадження цих методів, що дозволяє ефективно захищати серверні застосунки від потенційних загроз. Використання таких методів є важливим кроком для забезпечення надійності та безпеки систем, що використовують API.

РОЗДІЛ 3. ВПРОВАДЖЕННЯ МЕТОДІВ ЗАХИСТУ API

3.1 Реалізація методів захисту API

Усі нижче наведені приклади реалізації методів захисту API виконані з використанням Node.js – популярної серверної платформи, що дозволяє створювати швидкі та масштабовані веб-додатки. Використання Node.js забезпечує високу продуктивність та гнучкість при розробці API, а також підтримує велику кількість бібліотек і фреймворків, що спрощують інтеграцію різних методів безпеки.

Усі приклади коду демонструють, як за допомогою популярних бібліотек для Node.js можна ефективно реалізувати заходи захисту API від різноманітних атак та забезпечити безпеку даних користувачів.

1. Валідація даних з використанням бібліотеки class-transformer.

Одним із основних методів захисту API є валідація вхідних даних, яка дозволяє перевіряти коректність і безпечність даних, що надходять від користувачів або сторонніх сервісів. Це допомагає запобігти введенню некоректних або потенційно небезпечних даних, таких як SQL-ін'єкції, XSS-атаки чи інші типи зловживань.

В Node.js для валідації даних часто використовують бібліотеки, такі як joi, zod, class-validator, що дозволяють автоматично перевіряти й трансформувати дані. Приклад реалізації валідації даних за допомогою npm пакета class-validator:

1) Створення класу, який буде описувати DTO (Data Transfer Object) користувача. У class-validator використовуються декоратори для валідації параметрів класу.

```
import { IsString, MinLength, MaxLength, IsEmail } from 'class-validator';

class UserDTO {
  @IsString()
  @MinLength(3)
  @MaxLength(50)
  username: string;

  @IsEmail()
  email: string;
}
```

2) Створення функції для валідації даних користувача. За допомогою class-transformer ми перетворюємо простий об'єкт в екземпляр класу, що дозволяє використовувати всі переваги валідації класів. Функція validate з бібліотеки class-validator виконує перевірку об'єкта за правилами, визначеними в DTO.

```
import { plainToClass } from 'class-transformer';
import { validate } from 'class-validator';

async function validateUserData(userData) {
  const user = plainToClass(UserDTO, userData);

  const errors = await validate(user);

  if (errors.length > 0) {
    console.log("Validation failed. Errors: ", errors);
    return false;
  } else {
    console.log("Validation succeeded");
    return true;
  }
}
```

2. Хешування паролів з використанням бібліотеки crypto.

Бібліотека crypto є вбудованою в Node.js і надає функціональність для криптографічних операцій, зокрема для хешування паролів. Вона підтримує різноманітні алгоритми хешування, такі як SHA-256, SHA-512, і HMAC, що дозволяє ефективно реалізувати захист паролів у вашому API. Для більшої безпеки також рекомендується використовувати "сіль" – випадкову строку, яка додається до пароля перед його хешуванням. В даному прикладі використовується алгоритм pbkdf2 з бібліотеки crypto.

1) Створення функції для хешування пароля:

```
import * as crypto from 'crypto';

async function hashPassword(password) {
  const salt = crypto.randomBytes(16).toString('hex');

  return new Promise((resolve, reject) => {
    crypto.pbkdf2(password, salt, 100000, 64, 'sha512', (err, derivedKey) => {
      if (err) reject(err);
      resolve(`${salt}:${derivedKey.toString('hex')}`);
    });
  });
}
```

2) Створення функції для перевірки пароля:

```
async function comparePassword(plainPassword, hashedPassword) {
  const [salt, password] = storedHash.split(':');

  return new Promise((resolve, reject) => {
    crypto.pbkdf2(plainPassword, salt, 100000, 64, 'sha512', (err, hashedPlainPassword) => {
      if (err) reject(err);
      resolve(password === hashedPlainPassword.toString('hex'));
    });
  });
}
```

3. Генерація JWT токенів за допомогою бібліотеки jsonwebtoken

JWT (JSON Web Token) є стандартом, який дозволяє безпечно передавати інформацію між клієнтом і сервером у вигляді компактного, підписаного токена. Він зазвичай використовується для аутентифікації та авторизації в API. У Node.js можна ефективно реалізувати захист API за допомогою бібліотеки jsonwebtoken, яка надає прості функції для генерації, перевірки та декодування JWT.

1) Створення функції для генерації JWT токену.

```
function generateToken(userId) {
  const payload = { userId };
  const secretKey = 'SecretKey';
  const options = { expiresIn: '1h' };

  return jwt.sign(payload, secretKey, options);
}
```

2) Створення функції для перевірки JWT токену:

```
function verifyToken(token) {
  const secretKey = 'SecretKey';

  try {
    return jwt.verify(token, secretKey);
  } catch (error) {
    throw new Error('Invalid or expired token');
  }
}
```

4. Rate limiting за допомогою бібліотеки express-rate-limit

Rate Limiting – це метод обмеження кількості запитів, які клієнт може надіслати до API за певний проміжок часу. Це ефективний спосіб захисту від атак типу DDoS та надмірного використання API. У Node.js з використанням

фреймворку express цей підхід можна реалізувати за допомогою бібліотеки express-rate-limit, яка дозволяє легко налаштувати обмеження швидкості запитів для API.

1) Створення middleware для обмеження кількості запитів по IP.

```
import rateLimit from 'express-rate-limit';

const limiter = rateLimit({
  windowMs: 15 * 60 * 1000, // 15 хвилин
  max: 100, // Максимум 100 запитів з однієї IP-адреси за 15 хвилин
  standardHeaders: true,
  legacyHeaders: false,
  message: 'Rate limit',
});
```

2) Підключення middleware до express api.

```
import express from 'express';

const app = express();
app.use(limiter);

app.listen(3000, () => {
  console.log('Server is running on port 3000');
});
```

3.2 Тестування безпеки API

Тестування безпеки API – це процес, спрямований на перевірку рівня захищеності API від можливих атак і вразливостей. Забезпечення безпеки API є критично важливим для збереження цілісності та конфіденційності даних, а також для запобігання несанкціонованому доступу до ресурсів системи. Цей процес включає декілька різних видів тестування, кожен з яких перевіряє специфічні аспекти функціонування API.

Основні види тестування API:

1) Функціональне тестування.

Функціональне тестування перевіряє відповідність API заявленим вимогам і правильність його функціонування. Це базовий вид тестування, який забезпечує коректну роботу основних функцій API.

У середовищі Node.js одним з найпопулярніших інструментів для функціонального тестування є Jest. Jest забезпечує зручну платформу для

написання тестів, надаючи функції для асинхронного тестування та зручний формат звітів. Приклад функціонального тесту з використанням Jest:

```
import request from 'supertest';
import app from '../app.js';

describe('Функціональне тестування API користувачів', () => {
  it('повинен повернути список користувачів', async () => {
    const response = await request(app).get('/api/users');
    expect(response.statusCode).toBe(200);
    expect(Array.isArray(response.body)).toBe(true);
  });


  it('повинен створити нового користувача', async () => {
    const newUser = { name: TestUser, email: testUser@example.com' };
    const response = await request(app).post('/api/users').send(newUser);
    expect(response.statusCode).toBe(201);
    expect(response.body).toHaveProperty('id');
  });

});
```

2) Тестування безпеки

Тестування безпеки включає перевірку захищеності API від можливих атак і вразливостей.

Основні цілі тестування безпеки:

- Виявлення вразливостей, таких як SQL Injection, Cross-Site Scripting (XSS), Cross-Site Request Forgery (CSRF).
- Перевірка правильності реалізації механізмів аутентифікації та авторизації.
- Перевірка відповідності стандартам безпеки, наприклад OWASP API Security Top 10 [31].

Інструменти для тестування безпеки:

- OWASP ZAP: автоматизований сканер вразливостей, який дозволяє виконувати перевірку безпеки API та виявляти поширені атаки.
- Burp Suite: професійний інструмент для аналізу безпеки веб-додатків, що включає функціонал для тестування API [32].

3) Тестування продуктивності.

Цей вид тестування перевіряє, як API поводиться під навантаженням і чи здатен витримувати великий обсяг одночасних запитів без зниження продуктивності.

Основні аспекти тестування продуктивності:

- Навантажувальне тестування (Load Testing): перевіряє, як API обробляє певний обсяг запитів у визначений період часу [33].
- Стрес-тестування (Stress Testing): визначає, як API поводиться при максимальному навантаженні та коли система перевантажена.
- Тестування стабільності (Stability Testing): оцінює, наскільки стабільною є робота API під час тривалого навантаження [34].

Інструменти для тестування продуктивності:

- Apache JMeter – один із найбільш популярних інструментів для створення сценаріїв тестування навантаження [35].
- k6 – сучасний інструмент для тестування навантаженнями, який дозволяє створювати сценарії за допомогою JavaScript [36].

4) Інтеграційне тестування

Інтеграційне тестування перевіряє коректну взаємодію між різними компонентами API, включаючи базу даних, зовнішні сервіси та інші модулі [37]. Приклад інтеграційного тесту з використанням Jest [38]:

```
import request from 'supertest';
import app from '../app.js';

describe('Інтеграційне тестування аутентифікації', () => {
  it('повинен успішно аутентифікувати користувача з правильними даними', async () => {
    const response = await request(app)
      .post('/api/login')
      .send({ email: 'user@example.com', password: 'password123' });
    expect(response.statusCode).toBe(200);
    expect(response.body).toHaveProperty('token');
  });

  it('повинен повернути помилку з некоректними даними', async () => {
    const response = await request(app)
      .post('/api/login')
      .send({ email: 'user@example.com', password: 'wrongpassword' });
    expect(response.statusCode).toBe(401);
    expect(response.body.message).toBe('Неправильний пароль');
  });
});
```

5) Ручне тестування

Хоча автоматизовані тести можуть охопити більшість сценаріїв та допомогти при регресивному тестуванні, ручне тестування також є важливим етапом, особливо при перевірці складних бізнес-процесів або нестандартних випадків [39].

Для ручного тестування часто використовується Postman [40] – інструмент, що дозволяє зручно створювати та відправляти HTTP-запити до API, а також аналізувати відповіді. Він дає змогу тестувальникам без необхідності писати код перевіряти різні ендпойнти API, перевіряти статус-коди, вміст відповідей та налаштовувати параметри запитів. Postman також підтримує різні методи аутентифікації, що дозволяє перевіряти захищені ресурси API, та дозволяє автоматизувати деякі процеси тестування.

3.3 Технологія імплементації методів забезпечення кібербезпеки та тестування безпеки API

Розглянувши основні методи забезпечення кібербезпеки API та тестування безпеки API було розроблено технологію імплементації методів кібербезпеки та тестування безпеки API. Процес впровадження методів базується на принципах паралельного виконання незалежних етапів, що дозволяє оптимізувати ресурси та скоротити час розробки.

Опис кроків алгоритму технології імплементації методів забезпечення кібербезпеки та тестування безпеки API:

1) Ініціалізація проекту – вибір архітектурного стилю API та написання базової файлової структури проекту.
2) Додавання SSL сертифікатів (HTTPS) – отримання від замовника або покупка SSL сертефікатів з їх подальшим налаштуванням на сервері.
3) Імплементація Rate Limiting – вибір технології та налаштування Rate Limiting на рівні серверу або на рівні API.

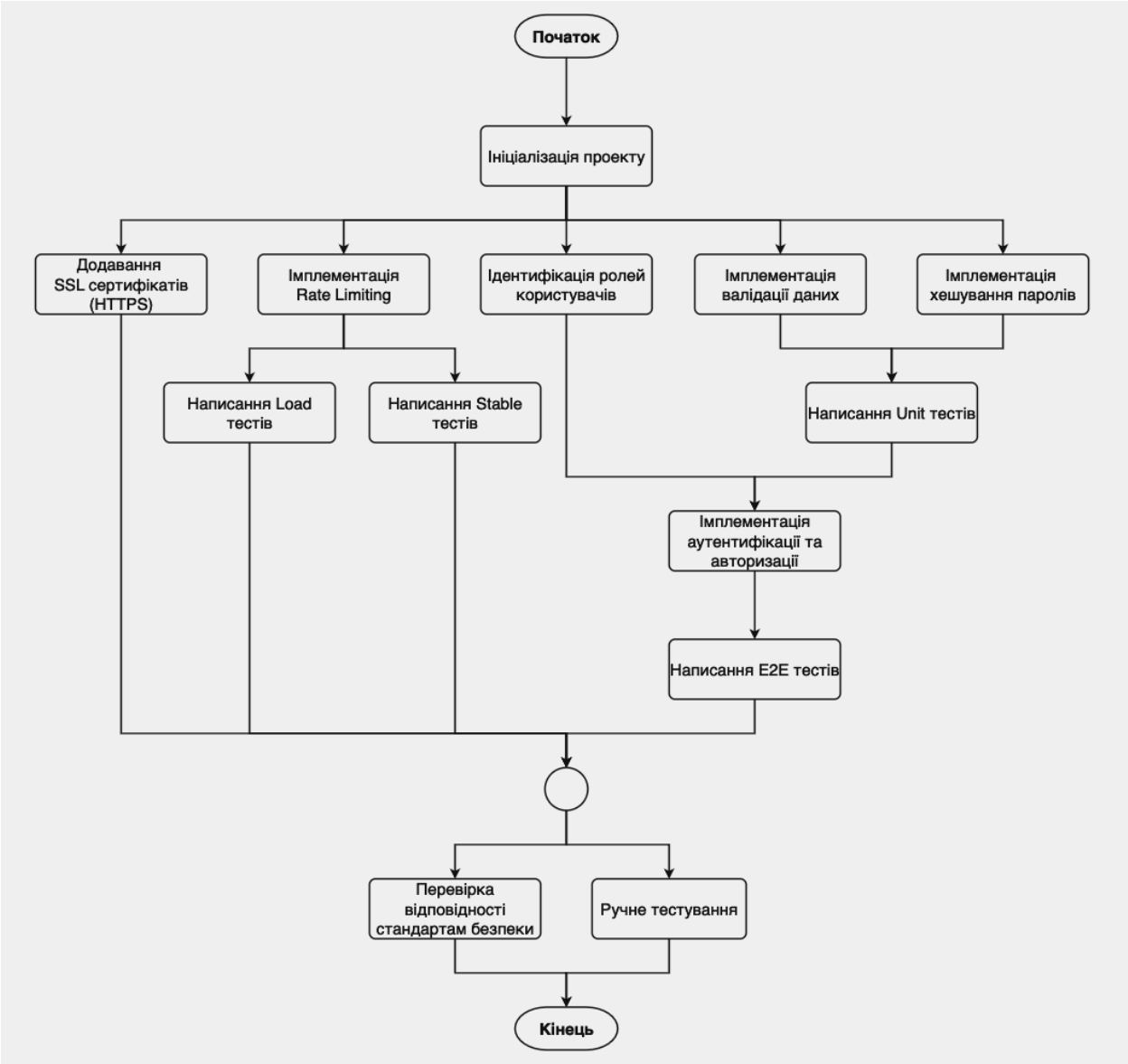

Рисунок 1 – Алгоритм технології імплементації методів забезпечення кібербезпеки та тестування безпеки API

4) Написання Load тестів та написання Stable тестів – написання тестів для перевірки коректності роботи rate limiting та для регресивного тестування в подальшому.
5) Ідентифікація ролей користувачів – комунікація з замовником для визначення ролей користувачів, які будуть використовуватись в продукті.
6) Імплементація валідації даних – вибір бібліотеки для валідації даних та побудова інфраструктури для валідації даних.

7) Імплементація хешування паролів – вибір бібліотеки та алгоритмів для хешування та побудова інфраструктури для хешування паролів.
8) Написання Unit тестів – написання тестів для перевірки коректності роботи валідації та хешування паролів.
9) Імплементація аутентифікації та авторизації – вибір підходів для авторизації та аутентифікації та їх імплементація у систему.
10) Написання E2E тестів – написання тестів для перевірки коректності роботи функціоналу авторизації та аутентифікації.
11) Перевірка відповідності стандартам безпеки – перевірка відповідності впровадженого захисту стандартам безпеки (SOC 2, HIPAA, GDPR, ISO 27001).
12) Ручне тестування – тестування впровадженого захисту в ручному режимі.

Розподіл кроків алгоитму технології імплементації методів забезпечення кібербезпеки та тестування безпеки API по виконавцям:

Таблиця 2 – Виконавці кроків алгоритму

Виконавці	**Кроки Алгоритму**
Backend Developer та DevOps	Ініціалізація проекту
	Імплементація Rate Limiting
	Написання Load тестів
	Написання Stable тестів
DevOps	Додавання SSL сертифікатів (HTTPS)
Backend Developer	Імплементація валідації даних
	Імплементація хешування паролів
	Написання Unit тестів
	Імплементація аутентифікації та авторизації
Business Analyst	Ідентифікація ролей користувачів

Automatic QA	Написання E2E тестів
Cybersecurity Specialist	Перевірка відповідності стандартам безпеки
Manual QA	Ручне тестування

Розподіл кроків алгоитму технології імплементації методів забезпечення кібербезпеки та тестування безпеки API на рівні захисту (кожен наступний рівень включає в себе методи з попереднього):

Таблиця 3 – Рівні захисту алгоритму

Рівень захисту	**Кроки Алгоритму**	**Опис рівня захисту**	**Призначення**
1	Ініціалізація проекту	Рівень захисту 1 спрямований на запобігання основним загрозам, таким як перехоплення даних (MITM-атаки), перевантаження системи (DoS/DDoS-атаки), а також забезпечення стабільної роботи під високими навантаженнями.	Для публічних API, які лише відображають дані.
	Додавання SSL сертифікатів (HTTPS)		
	Імплементація Rate Limiting		
	Написання Load тестів		
	Написання Stable тестів		
	Ручне тестування		
2	Імплементація валідації даних	Рівень захисту 2 розширює базові заходи, додаючи запобігання загрозам,	Для публічних API, які приймають дані від користувачів або
	Написання Unit тестів		

	Перевірка відповідності стандартам безпеки	пов’язаним із введенням некоректних даних (SQL/NoSQL ін’єкції, XSS), і гарантує, що система відповідає сучасним стандартам безпеки.	інтегруються з іншими системами.
3	Імплементація хешування паролів	Рівень захисту 3 забезпечує захист чутливих даних, таких як паролі, реалізує механізми аутентифікації та авторизації для захисту ресурсів.	Для вебсайтів, які працюють із зареєстрованими користувачами, вимагають контролю доступу та захисту персональних даних.
	Ідентифікація ролей користувачів		
	Імплементація аутентифікації та авторизації		
	Написання E2E тестів		

Висновок до третього розділу

У третьому розділі було розглянуто практичну реалізацію методів захисту API, описаних у попередньому розділі. Зокрема, було наведено приклади впровадження валідації даних, хешування паролів, використання HTTPS та обмеження частоти запитів на практиці. Також було сформовано блок-схему технології імплементації методів забезпечення кібербезпеки та тестування безпеки API. Такі практичні рекомендації дозволяють забезпечити надійний захист API та значно знизити ризики безпеки.

ВИСНОВОК

Під час виконання курсової роботи було здійснено глибоке дослідження методів захисту API, їх значення в умовах сучасного розвитку інформаційних технологій, а також розглянуті основні загрози безпеці, з якими можуть стикатися серверні застосунки. API є важливими інструментами для забезпечення інтеграції та взаємодії між різними програмними компонентами, однак саме через них часто реалізуються атаки, що ставлять під загрозу безпеку даних і функціонування системи. Тому належний захист API є важливим аспектом в контексті забезпечення інформаційної безпеки.

У першому розділі роботи було здійснено детальний аналіз API, їхніх основних типів, таких як REST, GraphQL та WebSocket, зокрема їхніх особливостей і функціональних можливостей. Це дозволило отримати чітке уявлення про роль API в сучасних інформаційних системах та їхню значущість у забезпеченні комунікації між різними сервісами. Визначення основних типів API дозволяє зрозуміти, які методи захисту підходять для кожного з них, оскільки їхня структура і характер взаємодії можуть впливати на вибір конкретних інструментів і підходів для забезпечення безпеки. У результаті аналізу було розкрито, що кожен тип API має свої специфічні вимоги до захисту, і тому універсальних рішень бути не може.

Другий розділ був присвячений практичним методам захисту API, що можуть бути застосовані для мінімізації ризиків, зазначених у першому розділі. Описані методи включають валідацію даних, хешування паролів, використання HTTPS та впровадження обмежень на частоту запитів. Валідація даних дозволяє запобігти введенню небезпечних або неправомірних даних, хешування паролів — захищає облікові дані користувачів від зловмисників, використання HTTPS забезпечує шифрування інформації під час її передачі, а обмеження частоти запитів (rate limiting) запобігає атакам на основі перевантаження серверу. Всі ці методи є ефективними і простими для впровадження в реальні проекти.

У третьому розділі було проаналізовано реалізацію методів захисту за допомогою популярних бібліотек і фреймворків. Зокрема, було розглянуто бібліотеки для валідації даних, інструменти для хешування паролів, а також

бібліотеки для обмеження частоти запитів. Практичний аналіз дозволив переконатися в ефективності застосування цих методів у реальних умовах і показав, що їх реалізація є достатньо простою і доступною для впровадження в різні типи проектів. Дослідження цих методів дозволило сформувати блок-схему технології імплементації методів забезпечення кібербезпеки та тестування безпеки API.

Враховуючи, що API є важливим компонентом сучасних веб-застосунків, їхній захист є ключовим елементом забезпечення загальної безпеки інформаційних систем. У результаті дослідження були виявлені основні загрози безпеці API та представлені ефективні методи захисту, здатні мінімізувати ці загрози. Описані методи є необхідними елементами для забезпечення належної безпеки API.

ПЕРЕЛІК ВИКОРИСТАНИХ ДЖЕРЕЛ

1) Top 10 API Breaches in 2024: веб-сайт. 09.09.2024. URL: https://equixly.com/blog/2024/09/06/top-10-api-breaches-in-2024/ (дата звернення 11.10.2024)
2) Гінчук В. Аналіз сутності прикладних програмних інтерфейсів : матеріали міжнар. наук. конф. Здобутки та досягнення прикладних та фундаментальних наук XXI століття, м. Біла Церква, 22.11.2024 р. Біла Церква, 2024, с. 309-310.
3) Гінчук В. Основні види прикладних програмних інтерфейсів: матеріали міжнар. наук. конф. Науковий простір: актуальні питання , досягнення та інновації, м. Житомир, 29.11.2024 р, Житомир, 2024, с. 336-337.
4) Гінчук В. Загрози безпеці прикладних програмних інтерфейсів : матеріали міжнар. наук. конф. Інноваційна наука : пошук відповідей на виклики сучасності, м. Могилів-Подільський, 6.12.2024 р., Могилів-Подільський, 2024, с. 256-257.
5) API: веб-сайт. 15.10.2024. URL: https://en.wikipedia.org/wiki/API (дата звернення 21.10.2024)
6) What is API Integration?: веб-сайт. URL: https://www.cleo.com/blog/what-is-api-integration#:~:text=API%20integration%20refers%20to%20the,exchange%20data%20and%20perform%20actions (дата звернення 22.10.2024)
7) The Importance of Standardized API Design: веб-сайт. 26.05.2017. URL: https://swagger.io/blog/api-design/the-importance-of-standardized-api-design/ (дата звернення 27.10.2024)
8) What is an API (Application Programming Interface)?: веб-сайт. URL: https://aws.amazon.com/what-is/api/?nc1=h_ls (дата звернення 23.10.2024)
9) What is a RESTful API?: веб-сайт. URL: https://aws.amazon.com/what-is/restful-api/?nc1=h_ls (дата звернення 24.10.2024)
10) GraphQL: веб-сайт. URL: https://graphql.org/ (дата звернення 24.10.2024)

11) WebSocket: веб-сайт. 10.10.2024. URL: https://en.wikipedia.org/wiki/WebSocket#:~:text=WebSocket%20is%20a%20computer%20communications,as%20RFC%206455%20in%202011 (дата звернення 24.10.2024)

12) API Types and Protocols: веб-сайт. URL: https://stoplight.io/api-types (дата звернення 01.11.2024)

13) Що таке DDoS-атака?: веб-сайт. URL: https://www.microsoft.com/uk-ua/security/business/security-101/what-is-a-ddos-attack (дата звернення 01.11.2024)

14) SQL Injection: веб-сайт. URL: https://www.w3schools.com/sql/sql_injection.asp (дата звернення 02.11.2024)

15) What Is BOLA?: веб-сайт. URL: https://www.akamai.com/glossary/what-is-bola (дата звернення 02.11.2024)

16) man-in-the-middle attack (MitM): веб-сайт. URL: https://www.techtarget.com/iotagenda/definition/man-in-the-middle-attack-MitM (дата звернення 02.11.2024)

17) joi: веб-сайт. URL: https://www.npmjs.com/package/joi (дата звернення 03.12.2024)

18) Zod: веб-сайт. URL: https://zod.dev/ (дата звернення 03.12.2024)

19) class-validator: веб-сайт. URL: https://github.com/typestack/class-validator (дата звернення 03.12.2024)

20) bcrypt: веб-сайт. 09.10.2024. URL: https://en.wikipedia.org/wiki/Bcrypt (дата звернення 03.12.2024)

21) Argon2: веб-сайт. 28.11.2024. URL: https://en.wikipedia.org/wiki/Argon2 (дата звернення 04.12.2024)

22) Salt (cryptography): веб-сайт. 23.10.2024. URL: https://en.wikipedia.org/wiki/Salt_(cryptography) (дата звернення 04.11.2024)

23) Brute-force search: веб-сайт. 15.11.2024. URL: https://en.wikipedia.org/wiki/Brute-force_search (дата звернення 20.11.2024)

24) API Authentication and Authorization: веб-сайт. 05.06.2023. URL: https://frontegg.com/guides/api-authentication-api-authorization (дата звернення 23.10.2024)

25) What is two-factor authentication or 2FA?: веб-сайт. URL: https://frontegg.com/guides/api-authentication-api-authorization (дата звернення 23.10.2024)

26) JSON Web Tokens: веб-сайт. URL: https://jwt.io/ (дата звернення 14.11.2024)

27) OAuth 2.0: веб-сайт. URL: https://oauth.net/2/ (дата звернення 14.11.2024)

28) HTTPS: веб-сайт. 04.11.2024. URL: https://en.wikipedia.org/wiki/HTTPS (дата звернення 16.11.2024)

29) Transport Layer Security: веб-сайт. 15.12.2024. URL: https://en.wikipedia.org/wiki/Transport_Layer_Security#SSL_1.0,_2.0,_and_3.0 (дата звернення 18.11.2024)

30) What Is Rate Limiting?: веб-сайт. URL: https://www.imperva.com/learn/application-security/rate-limiting/ (дата звернення 18.11.2024)

31) OWASP API Security Top 10: веб-сайт. URL: https://owasp.org/API-Security/editions/2023/en/0x11-t10/ (дата звернення 20.10.2024)

32) Burp Suite: веб-сайт. URL: https://portswigger.net/burp (дата звернення 06.12.2024)

33) Software load testing: веб-сайт. 13.08.2024 URL: https://en.wikipedia.org/wiki/Software_load_testing (дата звернення 01.12.2024)

34) Beyond The Breaking Point: How Stress Testing Ensures Software Stability: веб-сайт. 13.09.2024 URL: https://www.qatouch.com/blog/stress-testing/ (дата звернення 01.12.2024)

35) Jest : веб-сайт. URL: https://jestjs.io/ (дата звернення 05.12.2024)

36) Integration testing: веб-сайт. 19.09.2024 URL: https://en.wikipedia.org/wiki/Integration_testing (дата звернення 05.12.2024)

37) Apache JMeter: веб-сайт. URL: https://jmeter.apache.org/ (дата звернення 06.12.2024)

38) k6: веб-сайт. URL: https://k6.io/ (дата звернення 07.12.2024)

39) Manual testing: веб-сайт. 15.10.2024 URL: https://en.wikipedia.org/wiki/Manual_testing (дата звернення 19.10.2024)

40) Postman: веб-сайт. URL: https://www.postman.com/ (дата звернення 08.12.2024)