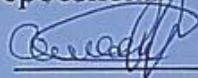


Міністерство освіти і науки України
Державний університет «Житомирська політехніка»
Факультет інформаційно-комп'ютерних технологій
Кафедра комп'ютерної інженерії та кібербезпеки

ПОЯСНЮВАЛЬНА ЗАПИСКА

до випускної кваліфікаційної роботи бакалавра
на тему: «Проект підсистеми захисту веб-сервера
організації»

Виконав студент 4-го курсу групи КБ-20-2
спеціальності 125 «Кібербезпека»



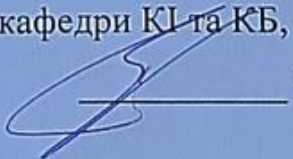
Дмитро Самчук

Керівник доцент кафедри КІ та КБ, к.пед.н., доцент



Олена Головня

Рецензент Доцент кафедри КІ та КБ, к.т.н.,



Олександр Пірог

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

5 березня 2024 р.

ЗАВДАННЯ

на випускню кваліфікаційну роботу бакалавра

Студент: Самчук Дмитро Олегович

Тема роботи: «Проект підсистеми захисту веб-сервера організації»

Затверджена Наказом університету від 5 березня 2024 р. №98/с

Термін здачі студентом закінченої роботи 9 червня 2024 р.

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

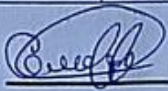
Об'єктом дослідження є процес розроблення захисту веб-сервера, а також предметом дослідження є метод

Обладнання та програмне забезпечення:

1. Серверний комп'ютер
2. Операційна система Windows 10
3. СУБД MongoDB

Консультанти з випускної кваліфікаційної роботи із зазначенням розділів, що їх стосуються

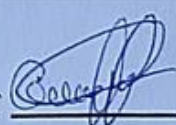
Розділ	Консультант	Підпис, дата	
		Завдання видав	Завдання прийняв
1	Головня О.С	05.03.2024	05.03.2024
2	Головня О.С	05.03.2024	05.03.2024
3	Головня О.С	05.03.2024	05.03.2024

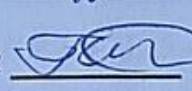
Студент  Дмитро Самчук

Керівник  Олена Головня

Календарний план

№ з/п	Назви етапів випускної кваліфікаційної роботи	Термін виконання етапів роботи	Примітка
1	Видача завдання	05.03.2024	виконано
2	Розробка плану роботи	13.03.2024	виконано
3	Розробка розгорнутого плану	19.03.2024	виконано
4	Подача чернетки керівнику для перевірки	07.05.2024	виконано
5	Доопрацювання роботи та перевірка керівником	09.05.2024	виконано
6	Затвердження графічних матеріалів у керівника	12.05.2024	виконано
7	Отримання відгуку керівника	28.06.2024	виконано
8	Допуск завідувача кафедри	28.06.2024	виконано
9	Подання роботи на рецензування	28.06.2024	виконано

Студент  Дмитро Самчук

Керівник  Олена Головня

АНОТАЦІЯ

Кваліфікаційна робота бакалавра присвячена розробці та впровадженню методів захисту веб-серверів для забезпечення конфіденційності, цілісності та доступності даних.

Метою роботи є розроблення та впровадження підсистеми захисту веб-сервера організації. Об'єктом дослідження є процес розроблення захисту веб-сервера, а також предметом дослідження є методи та технології, які використовуються при розробці та налаштуванні захисних механізмів для веб-сервера.

В першому розділі ми розібрали та проаналізували принципи роботи веб-серверів та їх методи роботи, також оглянули всі типи мережових атак. Надалі було важливим оглянути основні засоби захисту від кібератак.

В другому розділі було описано середовище для розробки для сервера та клієнта. Спроековано архітектуру підсистеми захисту, реалізовано функціонал підсистеми захисту.

В останньому заключному третьому розділі було оглянуто процес налагодження та було протестовано захист веб-серверу.

Пояснювальна записка до випускної кваліфікаційної роботи містить в собі 77 сторінок, 6 таблиць, 15 рисунків та 2 додатки. Список використаних джерел містить 25 найменування і займає 2 сторінки.

Ключові слова: ВЕБ-СЕРВЕР, ЗАХИСТ, БЕЗПЕКА, ВРАЗЛИВІСТЬ, ЗАХИСТ ВІД АТАК, НАЛАШТУВАННЯ, КІНОТЕАТР.

				КІК.КР.Б – 125 – 04 – ПЗ				
Змн.	Арк.	№ докум.	Підпис	Дата	Проект підсистеми захисту веб-сервера організації	Літ.	Арк.	Аркуші
Розроб.		Самчук Д.О.		28.06				
Керівник		Головня О.С.		28.06			4	36
Реценз.		Пірог О.В.		28.06		ФІКТ, гр. КБ-20-2		
Н. контр.		Головня О.С.		28.06				
Затверд.		Сфіменто А.А.		28.06				

ABSTRACT

The bachelor's qualification work is devoted to the development and implementation of web server protection methods to ensure confidentiality, integrity and availability of data.

The purpose of the work is the development and implementation of the protection subsystem of the organization's web server. The object of research is the process of developing web server protection, as well as the methods and technologies used in the development and configuration of security mechanisms for web server.

In the first chapter, we analyzed and analyzed the principles of web servers and their methods of operation, as well as reviewed all types of network attacks. In the future, it was important to review the main means of protection against cyber attacks.

The second chapter described the development environment for the server and the client. The architecture of the protection subsystem has been designed, the functionality of the protection subsystem has been implemented.

The final third chapter reviewed the debugging process and tested the security of the web server.

The explanatory note to the graduation thesis contains 77 pages, 6 tables, 15 figures and 2 appendices. The list of used sources contains 25 names and occupies 2 pages.

Keywords: WEB SERVER, PROTECTION, SECURITY, VULNERABILITY, PROTECTION AGAINST ATTACKS, CONFIGURATION, CINEMA.

		Самчук Д.О.			КІК.КР.Б – ПЗ	Арк.
		Головня О.С				4
Змн.	Арк.	№ докум.	Підпис	Дата		

ЗМІСТ

ВСТУП	7
РОЗДІЛ 1 АНАЛІЗ ЗАГРОЗ ТА ВРАЗЛИВОСТЕЙ ВЕБ-СЕРВЕРА ОРГАНІЗАЦІЇ.	9
1.1. Аналіз принципів функціонування веб-серверів та технологій їх будування	9
1.2. Основні типи мережових атак.....	16
1.3. Статистика і приклади випадків порушення кібербезпеки.....	25
1.4. Способи захисту веб-сервера від атак	30
1.5. Постановка технічного завдання	35
РОЗДІЛ 2 ПРОЕКТУВАННЯ ТА РОЗРОБКА ПІДСИСТЕМИ ЗАХИСТУ ВЕБ-СЕРВЕРА.....	38
2.1. Опис середовища розробки для сервера та клієнта.....	38
2.2. Проектування архітектури підсистеми захисту	40
2.3. Реалізація функціоналу підсистеми захисту	48
РОЗДІЛ 3 НАЛАГОДЖЕННЯ, ТЕСТУВАННЯ ТА ОЦІНКА ЕФЕКТИВНОСТІ ЗАХИСТУ ВЕБ-СЕРВЕРА	62
3.1. Огляд процесу налагодження та тестування.....	62
3.2. Тестування функціональності та захисту веб-сервера.....	63
3.3 Тестування веб-сервера для виявлення SQL ін'єкцій	71
ВИСНОВКИ.....	74
ДОДАТКИ.....	78

		Самчук Д.О.			КІК.КР.Б – ПЗ	Арк.
		Головня О.С				
Змн.	Арк.	№ докум.	Підпис	Дата		5

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

WAF – Web Application Firewall

DDoS – Distributed Denial of Service

TLS – Transport Layer Security

SSL – Secure Sockets Layer

CSRF – Cross-Site Request Forgery

XSS - Cross-Site Scripting

SQLi – SQL Injection

SIEM – Security Information and Event Management

IAM – Identity and Access Management

RBAC – Role-Based Access Control

SSO – Single Sign-On

CSP – Content Security Policy

HIDS – Host-Based Intrusion Detection System

NIDS – Network-Based Intrusion Detection System

API – Application Programming Interface

OTP – One-Time Password

DMZ – Demilitarized Zone

DNS – Domain Name System

		Самчук Д.О.			КІК.КР.Б – ПЗ	Арк.
		Головня О.С.				6
Змн.	Арк.	№ докум.	Підпис	Дата		

ВСТУП

Актуальність теми. Тема дослідження безпеки веб-серверів в сучасному інформаційному середовищі надзвичайно актуальна та важлива. З поширенням Інтернету та зростанням кількості онлайн-сервісів значно збільшилася також загроза для конфіденційності, цілісності та доступності даних, які зберігаються на веб-серверах. Зловмисники постійно шукають нові способи атаки на веб-додатки та сервери з метою отримання несанкціонованого доступу до конфіденційної інформації, зміни функціоналу системи або навіть знищення даних.

На фоні цього виникає потреба у розробці та вдосконаленні методів захисту веб-серверів від різноманітних кіберзагроз.

Мета і завдання роботи. Метою роботи є розроблення та впровадження підсистеми захисту веб-сервера організації.

Досягнення поставленої мети передбачає розв'язання наступних завдань:

- Аналіз принципів функціонування веб-серверів та технологій їх будування.
- Вивчення основних типів мережових атак та їх впливу на безпеку веб-сервера.
- Аналіз статистики та прикладів випадків порушення кібербезпеки в контексті веб-серверів.
- Розгляд наявних засобів захисту веб-сервера від потенційних атак.
- Реалізація функціоналу захисту веб-сервера на основі розроблених концепцій та архітектурних рішень.
- Тестування функціональності та захисту веб-сервера для підтвердження його ефективності

Об'єктом дослідження є процес розроблення захисту веб-сервера.

Предметом дослідження є методи та технології, які використовуються при розробці та налаштуванні захисних механізмів для веб-сервера.

Методи дослідження включають аналіз наукової літератури та публікацій з питань безпеки веб-серверів, проведення емпіричних досліджень шляхом аналізу вразливостей і атак на реальних веб-серверах або відтворення атак в лабораторних умовах, а також розробка та експериментальне випробування нових методів та інструментів захисту.

		Самчук Д.О.			КІК.КР.Б – ПЗ	Арк.
		Головня О.С.				7
Змн.	Арк.	№ докум.	Підпис	Дата		

Практичне значення роботи проявляється у **можливості застосування розробленої підсистеми захисту веб-сервера** у реальних умовах. Результати цього дослідження можуть бути використані розробниками веб-додатків, системними адміністраторами та інженерами з безпеки для забезпечення високого рівня безпеки та захисту даних на їхніх серверах.

Імплементація розробленого рішення дозволить запобігти різного роду кібератакам, забезпечуючи конфіденційність, цілісність та доступність інформації. Це дозволить підвищити рівень довіри користувачів до веб-сервісів та уникнути негативних наслідків, пов'язаних з можливими порушеннями безпеки.

		Самчук Д.О.			КІК.КР.Б – ПЗ	Арк.
		Головня О.С				8
Змн.	Арк.	№ докум.	Підпис	Дата		

РОЗДІЛ 1 АНАЛІЗ ЗАГРОЗ ТА ВРАЗЛИВОСТЕЙ ВЕБ-СЕРВЕРА ОРГАНІЗАЦІЇ

1.1. Аналіз принципів функціонування веб-серверів та технологій їх будування

В сучасному цифровому світі, де доступ до інформації та взаємодія з веб-ресурсами стали необхідністю, роль веб-серверів виявляється надзвичайною. Веб-сервери виступають як ключові компоненти інтернет-інфраструктури, забезпечуючи обробку запитів від клієнтських пристроїв та надання відповідної інформації. Отже, аналіз принципів їх функціонування та технологій, використовуваних у їх будуванні, виявляється ключовим завданням для розуміння суті інтернет-технологій та оптимізації процесів веб-розробки. У цьому пункті ми проведемо аналіз принципів роботи веб-серверів та розглянемо сучасні технології, які використовуються у їх конструюванні.

Веб-сервер є програмним засобом або комп'ютерною системою, яка надає інформаційні ресурси (веб-сторінки, файли, мультимедійний контент тощо) через Інтернет або локальну мережу користувачам, що використовують веб-браузери чи інші клієнтські програми. Основною функцією веб-сервера є обробка та відповідь на запити від клієнтських пристроїв, які надходять до нього через мережу.

Веб-сервери працюють на протоколі HTTP (Hypertext Transfer Protocol) або HTTPS (HTTP Secure), які використовуються для передачі гіпертекстових документів у форматі HTML (Hypertext Markup Language), а також інших ресурсів, таких як зображення, таблиці стилів CSS (Cascading Style Sheets), скрипти JavaScript тощо. Коли користувач відкриває веб-сторінку у браузері, браузер формує запит до веб-сервера, в якому вказується URL (Uniform Resource Locator) сторінки чи ресурсу, який користувач бажає отримати. Веб-сервер отримує цей запит, обробляє його і, в разі успішного виконання, надсилає клієнтові запрошений вміст у вигляді веб-сторінки чи іншого ресурсу.

		Самчук Д.О.			КІК.КР.Б – ПЗ	Арк.
		Головня О.С.				9
Змн.	Арк.	№ докум.	Підпис	Дата		

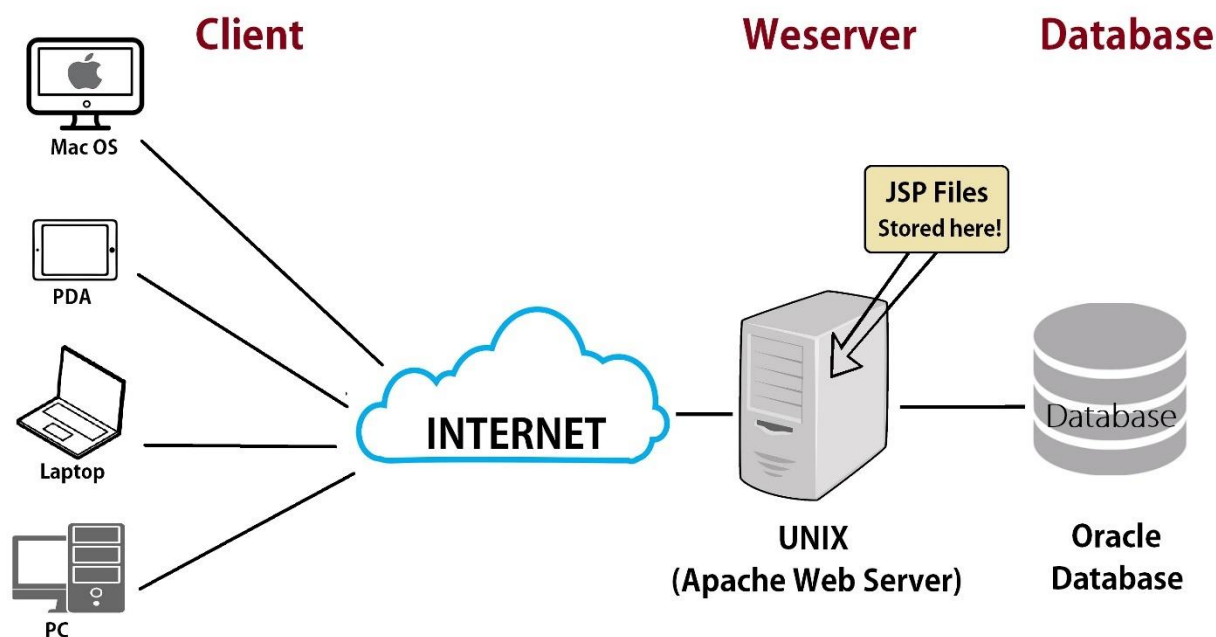


Рисунок 1.1– Схема взаємодії клієнт-сервер-база даних

Веб-сервер допомагає надавати веб-контент, до якого можна отримати доступ через Інтернет. Найпоширенішим використанням веб-серверів є розміщення веб-сайтів, оскільки Інтернет використовується не лише для отримання інформації, але й для інших цілей, таких як ігри, зберігання даних або запуск бізнес-додатків.

Веб-сервер може обробляти різноманітні типи запитів, включаючи запити на статичний контент (наприклад, HTML-файли або зображення), а також динамічний контент, який генерується на льоту за допомогою виконання серверних скриптів або застосунків. Одні з найпоширеніших веб-серверів включають Apache HTTP Server, Nginx, Microsoft Internet Information Services (IIS), а також вбудовані веб-сервери, які використовуються в різних мовах програмування та фреймворках, такі як Flask для Python або Express для Node.js.

Веб-сервери виконують різноманітні функції, які спрямовані на обробку та надання веб-ресурсів користувачам через Інтернет або локальні мережі. Розглянемо більш детально основні функції веб-серверів (табл.1.1).

		Самчук Д.О.			КІК.КР.Б – ПЗ	Арк.
		Головня О.С.				10
Змн.	Арк.	№ докум.	Підпис	Дата		

Основні функції веб-серверів

Функція	Характеристика
Прийом та обробка запитів	Одна з основних функцій веб-сервера полягає в прийомі запитів від клієнтських пристроїв через мережу. Це може бути запит на веб-сторінку, зображення, файл або інший ресурс. Веб-сервер обробляє ці запити, ідентифікує необхідний ресурс та генерує відповідь для клієнта
Надання вмісту	Однією з основних функцій веб-сервера є надання вмісту клієнтам у відповідь на їх запити. Це може включати відправку HTML-сторінок, зображень, аудіо- або відеофайлів, документів тощо
Обробка статичних та динамічних вмісту	Веб-сервер може обробляти як статичний, так і динамічний контент. Статичний контент зберігається на сервері у вигляді файлів і відправляється без будь-яких змін. Динамічний контент генерується на льоту за допомогою виконання серверних скриптів чи застосунків, які взаємодіють з базами даних або іншими джерелами даних
Керування сесіями та станом клієнта	Веб-сервер може керувати сесіями користувача, зберігаючи інформацію про стан сесії та ідентифікатори сесій для кожного клієнта. Це дозволяє зберігати стан між запитами, такі як дані кошика покупок або інформація про авторизацію
Забезпечення безпеки	Веб-сервери забезпечують безпеку обміну даними між сервером і клієнтом за допомогою шифрування та інших механізмів захисту, таких як HTTPS та SSL/TLS
Журналювання та моніторинг	Веб-сервер може вести журнали запитів, щоб забезпечити відстеження дій користувачів та виявлення проблем. Також він може забезпечувати моніторинг роботи сервера для виявлення навантаження та інших параметрів продуктивності

Ця таблиця надає детальний огляд основних функцій веб-серверів та їх характеристик. Прийом та обробка запитів, надання вмісту, обробка статичних та динамічних вмісту, керування сесіями та станом клієнта, забезпечення безпеки, а також журналювання та моніторинг – усі ці функції є ключовими складовими веб-серверів. Ці функції допомагають веб-серверам ефективно керувати веб-трафіком та надавати користувачам необхідні ресурси з високою швидкістю та надійністю.

Веб-сервери можуть бути класифіковані за різними критеріями, такими як функціональність, архітектура, операційна система, мова програмування та інші. Розглянемо основні види веб-серверів (рис.1.2).

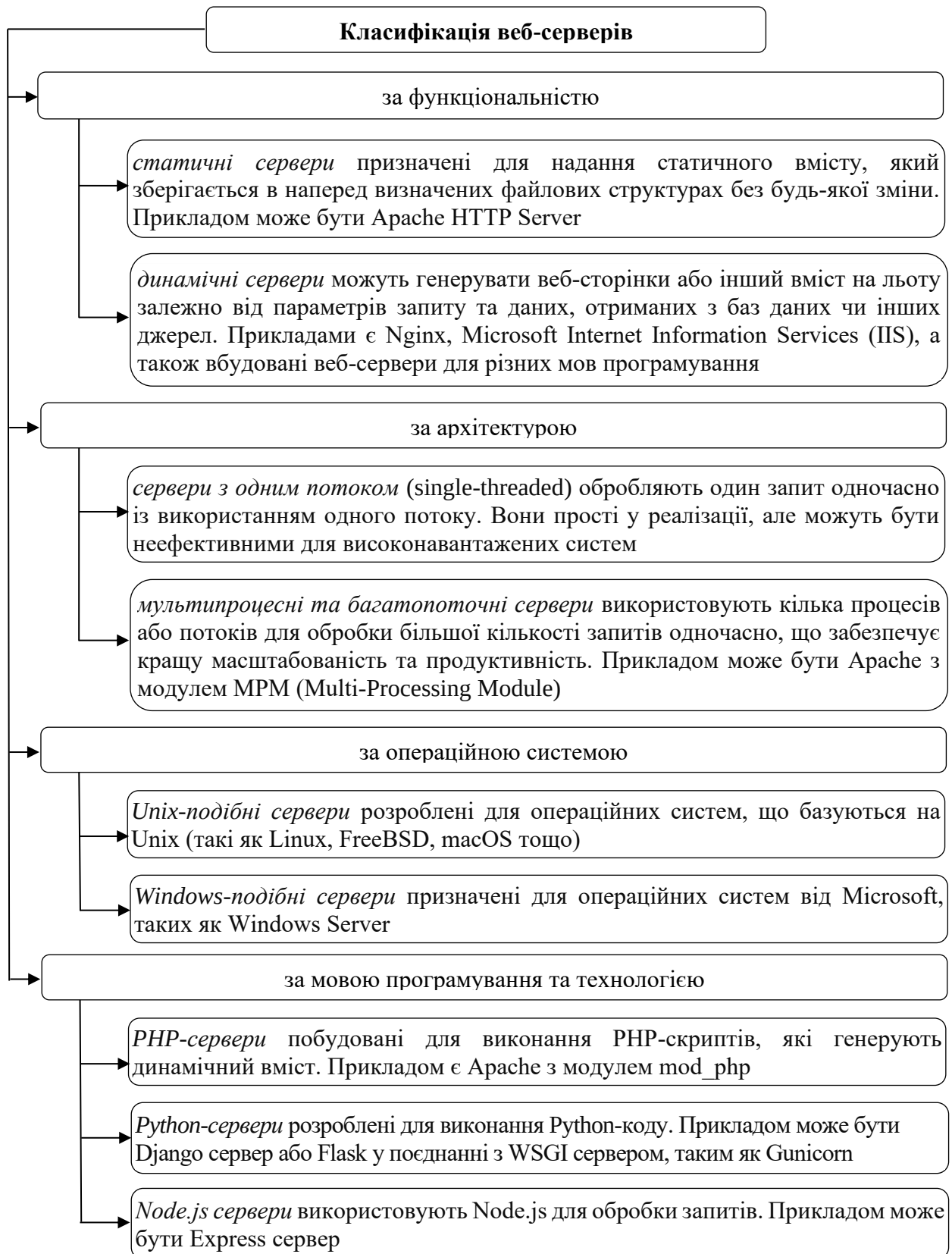


Рисунок 1.2 – Класифікація веб-серверів

Ця класифікація веб-серверів допомагає розуміти їхню роль та можливості у сучасній веб-розробці та інтернет-інфраструктурі. Кожен тип веб-сервера має свої

		Самчук Д.О.			КІК.КР.Б – ПЗ	Арк.
		Головня О.С.				12
Змн.	Арк.	№ докум.	Підпис	Дата		

переваги та недоліки, що може впливати на вибір конкретного рішення для певного проекту.

Принципи функціонування веб-серверів в основному базуються на обробці та відповіді на запити від клієнтських пристроїв. Основні принципи:

1. Прийом та аналіз запитів: веб-сервер постійно очікує на нові запити від клієнтів через мережу. Коли запит надходить, веб-сервер приймає його і аналізує, щоб визначити, який ресурс або послуга була запитана.

2. Визначення типу запиту: веб-сервер аналізує отриманий запит, щоб визначити, чи є це запит на статичний контент (наприклад, HTML-сторінку або зображення) чи на динамічний контент, який потребує обробки перед відправленням відповіді.

3. Обробка статичного контенту: якщо запит відноситься до статичного контенту, веб-сервер просто надсилає відповідний файл або ресурс назад клієнтові без будь-яких змін. Це може бути простий процес, оскільки статичні ресурси зберігаються на сервері у вигляді файлів.

4. Обробка динамічного контенту: у випадку динамічного контенту, як правило, виконується додатковий процес обробки. Це може включати виконання серверних скриптів або запуск веб-застосунків для генерації вмісту на льоту відповідно до параметрів запиту та даних, що можуть бути отримані з баз даних або інших джерел.

5. Формування відповіді: після обробки запиту веб-сервер формує відповідь, яка містить необхідний контент або інформацію, і відправляє її назад клієнтові через мережу.

6. Відправлення відповіді: коли відповідь сформована, веб-сервер відправляє її назад клієнтові через мережу. Вона може бути закодована у відповідному форматі, такому як HTML, XML, JSON тощо, залежно від типу запиту та призначення ресурсу.

7. Керування сесіями та безпекою: під час цього процесу веб-сервер може вести контроль сесій користувачів, забезпечуючи унікальні ідентифікатори сесій та зберігаючи стан між запитами. Веб-сервер також відповідає за забезпечення безпеки

		Самчук Д.О.			КІК.КР.Б – ПЗ	Арк.
		Головня О.С.				13
Змн.	Арк.	№ докум.	Підпис	Дата		

обміну даними між сервером і клієнтом, використовуючи різноманітні механізми шифрування та інші заходи безпеки, такі як HTTPS та SSL/TLS [17].

Ці принципи функціонування веб-серверів є основою для розуміння їхньої ролі в інтернет-інфраструктурі та веб-розробці. Вони дозволяють веб-серверам ефективно обробляти запити користувачів та надавати необхідні ресурси з високою швидкістю та надійністю.

Технології для будування веб-серверів на сьогоднішній день дуже різноманітні і можуть бути вибрані залежно від потреб проекту, вимог до продуктивності, масштабованості, зручності розробки та інших факторів. Ось детальний огляд деяких з найпопулярніших технологій для будування веб-серверів.

1. Серверні мови програмування:

– PHP є однією з найпоширеніших мов програмування для веб-розробки. Для побудови веб-серверів на PHP можна використовувати веб-сервер Apache з модулем `mod_php` або Nginx з підтримкою PHP-FPM.

– Python також широко використовується для створення веб-серверів. Для цього можна використовувати веб-фреймворки, такі як Django або Flask, у поєднанні з WSGI-серверами, такими як Gunicorn або uWSGI.

– Node.js дозволяє розробникам використовувати JavaScript для побудови веб-серверів. Для цього можна використовувати фреймворки, такі як Express, у поєднанні з серверами HTTP, які надає сам Node.js.

2. Системи управління контентом (CMS):

– WordPress – це одна з найпопулярніших платформ для створення веб-сайтів із власним веб-сервером, що базується на PHP та MySQL. Він має велику кількість розширень та тем, що робить його дуже гнучким.

– Drupal – це ще одна популярна CMS, що використовується для побудови веб-серверів. Вона надає більше можливостей для складних веб-проектів та має потужну систему модулів.

– Joomla – це CMS, яка також дозволяє швидко створювати веб-сайти з власним веб-сервером. Вона має добре розроблену систему розширень та шаблонів.

3. Фреймворки та бібліотеки:

		Самчук Д.О.			КІК.КР.Б – ПЗ	Арк.
		Головня О.С.				14
Змн.	Арк.	№ докум.	Підпис	Дата		

– Ruby on Rails – це фреймворк, який дозволяє швидко розробляти веб-сервери на мові програмування Ruby. Він має вбудовану підтримку для багатьох стандартних функцій, таких як автентифікація, маршрутизація та валідація.

– ASP.NET – це фреймворк для веб-розробки від Microsoft, який дозволяє розробникам будувати високопродуктивні веб-сервери на мові програмування C# або Visual Basic.NET.

4. Вбудовані веб-сервери:

– Express – це легкий фреймворк для побудови веб-серверів на Node.js. Він дозволяє швидко створювати API та додатки з використанням мінімального обсягу коду.

– Flask – це легкий мікрофреймворк для побудови веб-серверів на Python. Він дозволяє швидко створювати веб-сайти з використанням простого та інтуїтивно зрозумілого синтаксису.

Ці технології представляють лише деякі з можливих варіантів для будування веб-серверів, і вибір конкретної технології залежить від потреб проекту, експертизи розробника та інших факторів [21].

Таким чином, аналіз принципів функціонування веб-серверів та технологій їх будування свідчить про різноманітність і важливість цих компонентів у сучасній веб-розробці. Розуміння принципів роботи веб-серверів дозволяє розробникам ефективно планувати та реалізовувати веб-проекти з урахуванням вимог до продуктивності, безпеки та масштабованості. Вибір технологій для будування веб-серверів залежить від конкретних потреб проекту та експертизи розробників. Враховуючи широкий спектр доступних інструментів та фреймворків, розробники можуть забезпечити оптимальне рішення для реалізації веб-проектів будь-якої складності. Таким чином, аналіз принципів функціонування веб-серверів та огляд доступних технологій дозволяє підвищити якість, ефективність та надійність веб-додатків та сервісів.

		Самчук Д.О.			КІК.КР.Б – ПЗ	Арк.
		Головня О.С.				15
Змн.	Арк.	№ докум.	Підпис	Дата		

1.2. Основні типи мережевих атак

В сучасному цифровому світі загрози кібербезпеці стають невід’ємною складовою будь-якої мережевої інфраструктури. Одним із ключових аспектів забезпечення безпеки є розуміння основних типів мережевих атак, які можуть вплинути на інформаційну систему. Термін “мережеві атаки” відноситься до різноманітних стратегій та технік, які зловмисники використовують для незаконного доступу до мережевих ресурсів, порушення конфіденційності, цілісності чи доступності даних. У цьому пункті ми розглянемо основні типи таких атак та їх характеристики.

Розглянемо найбільш поширені загрози та атаки.

Комп’ютерний вірус, розроблений для зараження файлів та програмного забезпечення на комп’ютері, вважається одним із найпоширеніших видів комп’ютерних загроз. Ось детальна характеристика комп’ютерних вірусів та їхніх особливостей:

- комп’ютерний вірус вбудовує свій код в інші файли або програми, щоб розповсюджувати інфекцію;
- самореплікація: вірус може автоматично копіювати себе та вбудовувати свій код в інші файли на іншій машині або в мережі.

Шляхи розповсюдження:

- електронна пошта: віруси можуть поширюватися через електронну пошту шляхом використання відомих адрес або використання шкідливих вкладень;
- підозрілі веб-сайти: зловмисники можуть розміщувати вірусні файли на підозрілих веб-сайтах, які використовуються для інсталяції шкідливого ПЗ без відома користувача.

Шкідливі наслідки:

- видалення або пошкодження файлів: вірус може видаляти або пошкоджувати файли на комп’ютері, що може призвести до втрати даних або некоректної роботи системи;
- збір особистих даних: деякі віруси можуть збирати особисті дані користувачів, такі як паролі або інша конфіденційна інформація.

		Самчук Д.О.			КІК.КР.Б – ПЗ	Арк.
		Головня О.С.				16
Змн.	Арк.	№ докум.	Підпис	Дата		

Комп'ютерні віруси є серйозною загрозою для безпеки інформації та можуть спричинити значні шкоди користувачам. Розуміння їхньої природи та вжиття відповідних заходів захисту є важливими для збереження цілісності та безпеки комп'ютерної системи [14].

Шкідливе програмне забезпечення (ШПЗ), відоме також як malware, є загальним терміном, який охоплює широкий спектр комп'ютерних програм, призначених для нанесення шкоди комп'ютерній системі, користувачам або мережі. Ось детальніша характеристика шкідливого програмного забезпечення та його особливостей:

Типи шкідливого програмного забезпечення:

– віруси: шкідливі програми, які вбудовуються в інші файли або програми для їх зараження та розповсюдження;

– шпигунське програмне забезпечення: програми, які збирають конфіденційну інформацію про користувача без його знання або згоди;

– рекламне програмне забезпечення (adware): програми, які відображають нав'язливу рекламу або перенаправляють користувача на рекламні веб-сайти.

– рансомваре (ransomware): шкідливе програмне забезпечення, яке блокує доступ користувача до його файлів або системи та вимагає викуп за їх розблокування;

– троянські коні (Trojans): програми, які приховано встановлюються на комп'ютері користувача та намагаються отримати несанкціонований доступ до системи або даних.

Методи розповсюдження:

– фішинг та соціальна інженерія: використання маніпуляції та обману користувачів для отримання доступу до їх систем або конфіденційних даних;

– використання вразливостей програмного забезпечення: зловмисники використовують вразливості у програмах або операційних системах для введення шкідливого програмного забезпечення на комп'ютер користувача;

		Самчук Д.О.			КІК.КР.Б – ПЗ	Арк.
		Головня О.С.				17
Змн.	Арк.	№ докум.	Підпис	Дата		

– використання невідомих джерел: завантаження файлів або програм з ненадійних джерел, таких як торрент-сайти або неперевірені електронні листи, може призвести до зараження системи.

Шкідливі наслідки:

– втрата даних: ШПЗ може видаляти або пошкоджувати файли та дані на комп'ютері користувача;

– крадіжка особистої інформації: деяке ШПЗ збирає конфіденційну інформацію, таку як паролі, банківські дані або особисті дані користувачів;

– замедлення роботи системи: велика кількість шкідливих програм, що працюють одночасно, може спричинити збій роботи системи або замедлення її продуктивності [19].

Мережевий хробак (англ. “worm”) є одним із видів шкідливого програмного забезпечення, яке самостійно розповсюджується через комп'ютерні мережі. Ось детальна характеристика мережевих хробаків та їхніх особливостей:

Спосіб розповсюдження:

– самореплікація: мережевий хробак може автоматично копіювати свій код та поширювати себе до інших комп'ютерів у мережі без взаємодії користувача;

– використання вразливостей: хробаки можуть використовувати відомі вразливості в операційних системах або програмному забезпеченні для зараження комп'ютерів.

Шкідливі дії:

– переповнення мережі: масове поширення хробаків може призвести до переповнення мережевого трафіку, що може призвести до відмови в обслуговуванні (DoS) або перевантаження мережевих ресурсів;

– пошкодження даних: хробаки можуть вносити зміни до файлів або програм, що може призвести до втрати даних або некоректної роботи системи;

– утилізація ресурсів: хробаки можуть використовувати ресурси комп'ютера, такі як процесор та пам'ять, для виконання своїх завдань, що може спричинити сповільнення системи.

		Самчук Д.О.			КІК.КР.Б – ПЗ	Арк.
		Головня О.С.				18
Змн.	Арк.	№ докум.	Підпис	Дата		

Мережеві хробаки є серйозною загрозою для безпеки комп'ютерних мереж та можуть призвести до серйозних наслідків, таких як втрата даних, сповільнення роботи системи або навіть відмова в обслуговуванні. Ефективні заходи захисту та своєчасне реагування на загрози можуть допомогти зменшити ризики від хробаків у мережі [19].

Фішинг – це вид мережевої атаки, яка полягає в тому, що зловмисники намагаються обманом отримати конфіденційну інформацію, таку як паролі, номери кредитних карток чи інші особисті дані, шляхом імітації довіреного джерела. Ось детальніша характеристика фішингу та його особливості.

Способи атаки:

– електронна пошта: зловмисники надсилають електронні листи, що виглядають як листи від легітимних компаній чи організацій, які просять користувачів надати конфіденційну інформацію або перейти за посиланням на фішинговий сайт;

– соціальні мережі: зловмисники можуть створювати фішингові повідомлення або сторінки в соціальних мережах, що імітують відомі компанії або особи, щоб викликати довіру та отримати особисту інформацію;

– фішингові сайти: це сторінки, які імітують легітимні веб-сайти компаній, банків чи інших організацій, де користувачі ненавмисно вводять свої особисті дані.

Шкідливі наслідки:

– крадіжка особистих даних: зловмисники використовують отриману інформацію для крадіжки грошей, особистої інформації чи використання її в інших атаках;

– ідентифікаційний шахрайство: інформація, яка була отримана за допомогою фішингу, може бути використана для виконання ідентифікаційного шахрайства, такого як відкриття кредитних ліній чи укладення контрактів.

Фішинг є серйозною загрозою для безпеки користувачів мережі, але з уважністю та належними заходами захисту можна запобігти його наслідкам [21].

		Самчук Д.О.			КІК.КР.Б – ПЗ	Арк.
		Головня О.С.				19
Змн.	Арк.	№ докум.	Підпис	Дата		

Ботнет – це мережа комп’ютерів або інших пристроїв, які підконтрольні зловмисникам і використовуються для виконання різноманітних завдань без відома власників. Ось детальніша характеристика ботнетів та їх особливостей:

Створення і контроль:

– інфікування: зловмисники здійснюють інфікування пристроїв, включаючи комп’ютери, смартфони, планшети та підключені до Інтернету пристрої Інтернету речей (IoT), за допомогою шкідливого програмного забезпечення, такого як віруси, троянці або черв’яки.

– контроль: після інфікування зловмисник встановлює зв’язок з інфікованими пристроями через командно-контрольний сервер (C&C server), що дозволяє віддалено керувати ними.

Використання:

– DDoS атаки: ботнети часто використовуються для здійснення розподіленого атаки на замовлення (DDoS), коли велика кількість запитів надсилається до одного сервера одночасно, переповнюючи його трафіком та призводячи до недоступності.

– розсилка спаму: зловмисники можуть використовувати ботнети для масової розсилки спаму, включаючи шахрайські листи, рекламні повідомлення або шкідливі вкладення.

– крадіжка інформації: деякі ботнети спеціалізуються на крадіжці особистої інформації, такої як паролі, номери кредитних карток або інші конфіденційні дані.

Ботнети становлять серйозну загрозу для безпеки Інтернету, оскільки вони можуть використовуватися для виконання різних видів кібератак або незаконної діяльності. Захист від ботнетів вимагає ретельного моніторингу мережі та вжиття заходів для попередження інфікування пристроїв шкідливим програмним забезпеченням [19].

MIM або “Man-in-the-Middle” (людина посередині) – це тип атаки в комп’ютерних мережах, коли зловмисник вставляється між комунікуючими сторонами з метою перехоплення, зміни або перехватування комунікації між ними. Ось детальна характеристика атаки MIM:

Методи атаки:

		Самчук Д.О.			КІК.КР.Б – ПЗ	Арк.
		Головня О.С.				20
Змн.	Арк.	№ докум.	Підпис	Дата		

– перехоплення: зловмисник перехоплює передачу даних між двома сторонами, не дозволяючи їм знаходитися у прямому зв'язку.

– зміна: зловмисник може змінювати передані дані, щоб викликати неправильну поведінку або впливати на результати комунікації.

– перехват: зловмисник може перехоплювати дані для збору конфіденційної інформації, такої як паролі, особисті дані чи фінансова інформація.

Методи атаки:

– Wi-Fi атаки: зловмисники можуть використовувати нешифровані або слабо захищені Wi-Fi мережі для впровадження атак MIM та перехоплення трафіку між пристроями та мережевими точками доступу.

– ARP спуфінг: атакувач може використовувати ARP спуфінг для отримання контролю над маршрутизацією мережного трафіку та перенаправлення трафіку через свій проміжний сервер.

– SSL атаки: зловмисники можуть використовувати підроблені сертифікати SSL для перехоплення безпечних з'єднань та дешифрування зашифрованих даних.

Атака MIM є серйозною загрозою для безпеки мережі, оскільки вона дозволяє зловмисникам звертатися до чутливої інформації та діяти в ролі посередника між легітимними користувачами. Використання ефективних методів захисту та ретельний моніторинг мережі може допомогти попередити атаки MIM та забезпечити безпеку комунікації [14].

DoS (Denial of Service) і DDoS (Distributed Denial of Service) атаки – це типи кібератак, спрямованих на затримку чи припинення нормального функціонування веб-сервера, сервісу чи мережі, роблячи їх недоступними для законних користувачів. Розглянемо більш детально кожен тип атаки:

DoS атаки спрямовані на перевантаження або вичерпання ресурсів сервера чи мережі, змушуючи їх припинити обслуговування законних запитів.

Характеристики:

– надмірне навантаження: зловмисники намагаються надіслати велику кількість запитів до сервера або сервісу, перевантажуючи його та призводячи до відмови у обслуговуванні.

		Самчук Д.О.			КІК.КР.Б – ПЗ	Арк.
		Головня О.С.				21
Змн.	Арк.	№ докум.	Підпис	Дата		

– використання слабкостей: DoS атаки можуть використовувати вразливості у програмному забезпеченні чи мережевому обладнанні для досягнення своєї мети.

– приклади: SYN flood атаки, атаки з використанням ICMP echo-запитів, атаки на службу DNS, HTTP flood атаки тощо.

DDoS атаки використовують багато різних пристроїв, що входять у ботнет, для одночасної надсилання великої кількості запитів до цільового сервера чи мережі.

Характеристики:

– розподілені ресурси: зловмисники використовують багато різних комп'ютерів або пристроїв, щоб здійснити атаку, роблячи важчим виявлення та блокування джерел атаки.

– скоординованість: DDoS атаки зазвичай координуються через командно-контрольні сервери ботнета, щоб забезпечити синхронізацію атакуваних пристроїв.

– приклади: Mirai botnet, атаки на великі корпоративні мережі, атаки на онлайн-ігрові сервери тощо.

DoS та DDoS атаки можуть завдати серйозних шкоди компаніям та організаціям, що вони ціляться. Ефективний захист від таких атак вимагає ретельного моніторингу мережі та вжиття заходів захисту на всіх рівнях інфраструктури [21].

Програми-вимагачі (ransomware) представляють собою тип шкідливого програмного забезпечення, яке захоплює доступ до файлів або системи користувача та шифрує їх, після чого вимагає викуп за їх розшифрування. Розглянемо більш детально характеристику програм-вимагачів та їх особливостей:

– шифрування даних: програми-вимагачі шифрують файли на комп'ютері потерпілого користувача, що робить їх недоступними без ключа шифрування.

– ключ шифрування: зловмисники утримують ключ шифрування, який потрібен для розшифрування файлів, і вимагають викуп за його отримання.

Методи поширення:

– електронна пошта: програми-вимагачі можуть поширюватися через шкідливі електронні листи, вкладені файли або посилання на інфіковані веб-сайти.

		Самчук Д.О.			КІК.КР.Б – ПЗ	Арк.
		Головня О.С.				22
Змн.	Арк.	№ докум.	Підпис	Дата		

– збройні вразливості: зловмисники можуть використовувати вразливості в операційних системах або програмному забезпеченні для впровадження програм-вимагачів.

Вимоги викупу:

– вимоги до викупу: після шифрування файлів програми-вимагачі вимагають від користувача викуп за отримання ключа для розшифрування даних;

– оплата в криптовалюті: зловмисники зазвичай вимагають викуп у вигляді криптовалюти, такої як Bitcoin, для унеможливлення відстеження операцій.

Програми-вимагачі є серйозною загрозою для безпеки даних та функціонування організацій та індивідуальних користувачів. Розуміння їхньої природи та вжиття відповідних заходів захисту може допомогти запобігти втраті даних та уникнути викупу [19].

Атаки на мережі п'ятого покоління (5G) представляють собою потенційні загрози для безпеки та конфіденційності даних, пов'язаних з цими мережами. Вони можуть включати різні види атак, спрямованих на вразливості та слабкі місця в інфраструктурі 5G. Ось деякі з них.

Відмова в обслуговуванні (DoS) та розподілені атаки на замовлення (DDoS):

– ціль: переповнення мережі запитами, які призводять до відмови в обслуговуванні або недоступності послуг для легітимних користувачів;

– методи: використання ботнетів, флудінг мережі запитами або атаки на протоколи і маршрутизацію.

Атаки на протоколи мережі:

– ціль: використання вразливостей у протоколах 5G для виконання атак на аутентифікацію, авторизацію та обмін даними;

– методи: використання специфічних вразливостей у протоколах сигналізації, ідентифікації та криптографії.

Перехоплення та підробка трафіку:

– ціль: перехоплення конфіденційної інформації або підробка трафіку для зловживання.

		Самчук Д.О.			КІК.КР.Б – ПЗ	Арк.
		Головня О.С.				23
Змн.	Арк.	№ докум.	Підпис	Дата		

– методи: використання атаки MITM (людина посередині), вражений маніпулюванням або витягом даних, а також атаки на аутентифікацію.

Фізичні атаки на інфраструктуру:

– ціль: пошкодження чи знищення обладнання, що може призвести до відключення або недоступності послуг.

– методи: саботаж, крадіжка або фізичні втручання в інфраструктуру 5G, такі як атаки на антени або обладнання передачі.

Атаки на додаткові сервіси та функціональність:

– ціль: використання слабких місць у додаткових сервісах, таких як мережі IoT, мобільний доступ або віртуальні приватні мережі (VPN).

– методи: використання вразливостей у додаткових сервісах або пристроях для виконання атак на основну інфраструктуру 5G [20].

SQL Injection - це вид атаки на веб-додатки, яка полягає в тому, що зломисники використовують недостатньо захищені веб-форми або URL-адреси для внесення SQL-коду у базу даних, що використовується застосунком. Розглянемо більш детально характеристику SQL Injection та її особливостей:

Вразливість веб-додатка:

– неперевірені вхідні дані: зломисники можуть використовувати недостатньо перевірені форми введення на веб-сайті для вставки SQL-коду безпосередньо в запит до бази даних.

– URL-параметри: вразливості можуть бути також у URL-параметрах, де зломисники можуть впроваджувати SQL-код безпосередньо в URL-адресу.

Негативні наслідки:

– отримання доступу до даних: зломисники можуть отримати доступ до конфіденційної інформації, яка зберігається в базі даних, такої як ім'я користувача, паролі, особисті дані тощо.

– пошкодження або втрата даних: SQL Injection може призвести до видалення, модифікації або пошкодження даних у базі даних.

		Самчук Д.О.			КІК.КР.Б – ПЗ	Арк.
		Головня О.С.				
Змн.	Арк.	№ докум.	Підпис	Дата		24

– впровадження інших атак: зломисники можуть використовувати SQL Injection для впровадження інших атак, таких як видалення файлів, виконання коду або отримання контролю над сервером.

SQL Injection є серйозною загрозою для безпеки веб-додатків та може призвести до значних втрат даних або порушення конфіденційності користувачів. Застосування належних заходів захисту та ретельний моніторинг вразливостей допоможе запобігти успішним атакам SQL Injection.

Отже, на основі розглянутої інформації про основні типи мережових атак, можна зазначити, що ці атаки становлять серйозну загрозу для інформаційної безпеки користувачів та організацій у всьому світі. Кожен тип атаки має свої характеристики та методи поширення, а також може призвести до різних наслідків, включаючи втрату даних, порушення конфіденційності, перешкоджання нормальному функціонуванню системи та фінансові втрати.

1.3. Статистика і приклади випадків порушення кібербезпеки

Протягом останніх двадцяти років світ неперервно стикається з кібератаками різного масштабу та наслідків, що призводять до збитків як матеріального, так і морального характеру. Кожен з нас стає все менш захищеним у кіберпросторі через постійну еволюцію методів атак та зростаючу складність захисних заходів. Україна, в свою чергу, відзначається серед країн, що найбільше страждають від кібератак, і ця тенденція посилюється з плином часу. Зокрема, з початком повномасштабної війни, спровокованої діями Російської Федерації, Україна стала ще більш вразливою до атак у мережі. Зломисники, керовані найвищою владою країни-агресора, активно використовують кіберзброю з метою завдання шкоди та дестабілізації ситуації в країні. Об'єктами нападів стають веб-ресурси урядових установ, міністерств, Збройних Сил України, банків, торговельних мереж. Основними акторами цих кібератак, згідно з аналізами, найчастіше виявляються групи хакерів з Росії, Білорусії, Китаю та Північної Кореї. Проте, незважаючи на ці загрози, захист від кібератак можливий і вкрай важливий.

		Самчук Д.О.			КІК.КР.Б – ПЗ	Арк.
		Головня О.С.				25
Змн.	Арк.	№ докум.	Підпис	Дата		

Кіберпростір, як віртуальне середовище комунікації в мережі Інтернет, оперує згідно загальних правил інформаційних та телекомунікаційних систем. Він стає основною платформою для обміну даними, інформацією та комунікацією між користувачами. Однак кібербезпека, що є станом захищеності інтересів користувачів, суспільства та держави у кіберпросторі, стає надзвичайно актуальною у зв'язку з розвитком технологій та зростанням кількості кібератак. Кібератаки, навмисні дії у кіберпросторі, спрямовані на порушення конфіденційності, цілісності та доступності електронних інформаційних ресурсів, можуть мати різні масштаби та наслідки. Вони поділяються на глобальні, спрямовані на системи банків, державні ресурси та ключові інфраструктурні об'єкти, а також на локальні, які спрямовані на конкретні комп'ютери, користувачів або організації з метою викрадення персональних даних, шантажу чи завдання іншої шкоди [18].

У світі протягом останніх кількох років державний сектор, неурядові організації та галузі освіти стали найбільш постраждалими від кібератак, що відображається у відсотковому співвідношенні кількості кібератак. Зокрема, дослідження, проведені компанією Microsoft, підтверджують, що США та Україна зазнали найбільшого обсягу кібератак, причому Україна займає друге місце після США за кількістю атак. Російські хакерські групи відіграють ключову роль у світовому кіберпросторі, здійснюючи майже половину всіх атак, в той час як іранські, китайські та північнокорейські хакери також виявляють активність у мережі.

Атаки на кіберпростір у XXI столітті є складними та різноманітними, виявляючи нові методи та стратегії злому інформаційних систем. Однією з найбільш відомих та значущих кібератак цього періоду є атака на ядерні розробки, відома як Stuxnet. В 2009 році Stuxnet зламав іранські центрифуги для збагачення урану, викликаючи серйозні технічні відмови та затримки в ядерній програмі країни. Ця атака, вважається спільним проектом США та Ізраїлю, підкреслила нові можливості кібервійськової стратегії.

Інший важливий випадок порушення кібербезпеки стосується компанії Yahoo, яка зазнала вторгнення у 2013, 2014 та 2016 роках, призвівши до витоку мільярдів

		Самчук Д.О.			КІК.КР.Б – ПЗ	Арк.
		Головня О.С.				26
Змн.	Арк.	№ докум.	Підпис	Дата		

облікових записів користувачів. Ці випадки порушення даних стали найбільшими в історії Інтернету та мають значний вплив на сферу кібербезпеки та взаємодії в мережі.

DDoS-атака на компанію Dyn у 2016 році також відображає нові загрози в кіберпросторі. Вона призвела до тимчасової недоступності численних популярних веб-сервісів і показала вразливість інфраструктури Інтернету перед масштабними атаками. Атака виявила нові можливості зламу та маніпулювання підключеними пристроями, що викликає підвищений інтерес до безпеки Інтернету речей [24].

Злам сайту знайомств Ashley Madison у 2015 році відображає вразливість особистих даних у цифровому столітті. Цей випадок показує, як кібератаки можуть порушувати приватність користувачів та мати серйозні особисті наслідки для тих, чий інформація стала доступною зламникам.

Атака, відома як ShadyRAT, розпочалася у 2006 році та є тривалою загрозою в кіберпросторі. Цей випадок вважається одним із найбільш відомих та продуктивних, і її джерелом вважається Китай. У рамках цієї атаки заражений вірусом «троjanом» електронні листи поширювалися серед співробітників різних організацій, що відкривали шлях для установки вірусу на комп'ютери. Жертвами атаки стали такі організації, як Олімпійський комітет ООН, Асоціація держав Південно-Східної Азії, а також компанії з Японії, Швейцарії, Великої Британії, Індонезії, Данії, Сінгапуру, Гонконгу, Німеччини, Індії та уряди Сполучених Штатів, Тайваню, Південної Кореї, В'єтнаму.

Атака на Sony PlayStation Network, що сталася 20 квітня 2011 року, представляє собою ще один значний випадок порушення кібербезпеки. У результаті злому сервісу PlayStation Network, який відповідає за всі онлайн-функції консолей PlayStation, компанія зазнала значних фінансових втрат і втрати довіри споживачів. За проведення атаки відповідальність взяла хакерська група Anonymous.

Інший приклад кібератаки – це атака на нафтовий трубопровід у травні 2021 року. Хакери вплинули на роботу трубопроводу, який постачав 45% нафти для східного узбережжя США, призводячи до серйозних збоїв у транспортуванні. Відповідальність за атаку взяло на себе російське хакерське угруповання DarkSide.

		Самчук Д.О.			КІК.КР.Б – ПЗ	Арк.
		Головня О.С.				27
Змн.	Арк.	№ докум.	Підпис	Дата		

Інший вірус-вимагач, відомий як Petya, був виявлений у 2016 році. Він шифрував файли на жорсткому диску комп'ютера та вимагав викуп за їх розшифрування. Ця атака мала серйозний вплив на безпеку даних та функціонування комп'ютерних систем.

Крім цього, варто згадати троянський вірус Godfather, що став відомим 22 грудня 2022 року. Цей вірус направлений на банки та криптогаманці по всьому світу та представляє собою серйозну загрозу для фінансової безпеки.

Атака, відома як ShadyRAT, почалася у 2006 році і продовжується й донині. Її походженням вважається Китай. Вірус розсилає заражені "троянів" електронні листи співробітникам певних організацій, після відкриття яких "троян" встановлюється на комп'ютер. Серед постраждалих організацій були Олімпійський комітет ООН, Асоціація держав Південно-Східної Азії, компанії Японії, Швейцарії, Великої Британії, Індонезії, Данії, Сінгапуру, Гонконгу, Німеччини, Індії та уряди Сполучених Штатів, Тайваню, Південної Кореї, В'єтнаму.

Злам Sony PlayStation Network. 20 квітня 2011 року сервіс PlayStation Network, відповідальний за всі онлайн-функції консолей PlayStation, зазнав злому. Компанія почала розслідування, під час якого її акції впали на 20%, а вартість компанії втратила значну частину своєї вартості. У результаті перерви в роботі PlayStation Network компанія понесла фінансові втрати у розмірі 171 мільйона доларів. За атаку відповідальність взяла хакерська група Anonymous.

Атака на нафтовий трубопровід. У травні 2021 року хакери здійснили атаку на компанію Pipeline, через що нафтовий трубопровід, який постачав 45% нафти для східного узбережжя США, припинив роботу на кілька днів. Майже всі заправки у Вашингтоні залишилися без палива, а авіасполучення було призупинено. Відповідальність за атаку взяло на себе російське хакерське угруповання DarkSide.

Вірус-вимагач Petya. Програму виявили навесні 2016 року. Вона шифрувала файли на жорсткому диску комп'ютера користувача і перезаписувала та шифрувала головний завантажувальний запис, необхідний для завантаження операційної системи. Усі файли, що зберігалися на комп'ютері, ставали недоступними. За

		Самчук Д.О.			КІК.КР.Б – ПЗ	Арк.
		Головня О.С.				28
Змн.	Арк.	№ докум.	Підпис	Дата		

відновлення доступу до файлів вона вимагала від користувача 0,9 біткоїна – близько 380 доларів за тодішнім курсом.

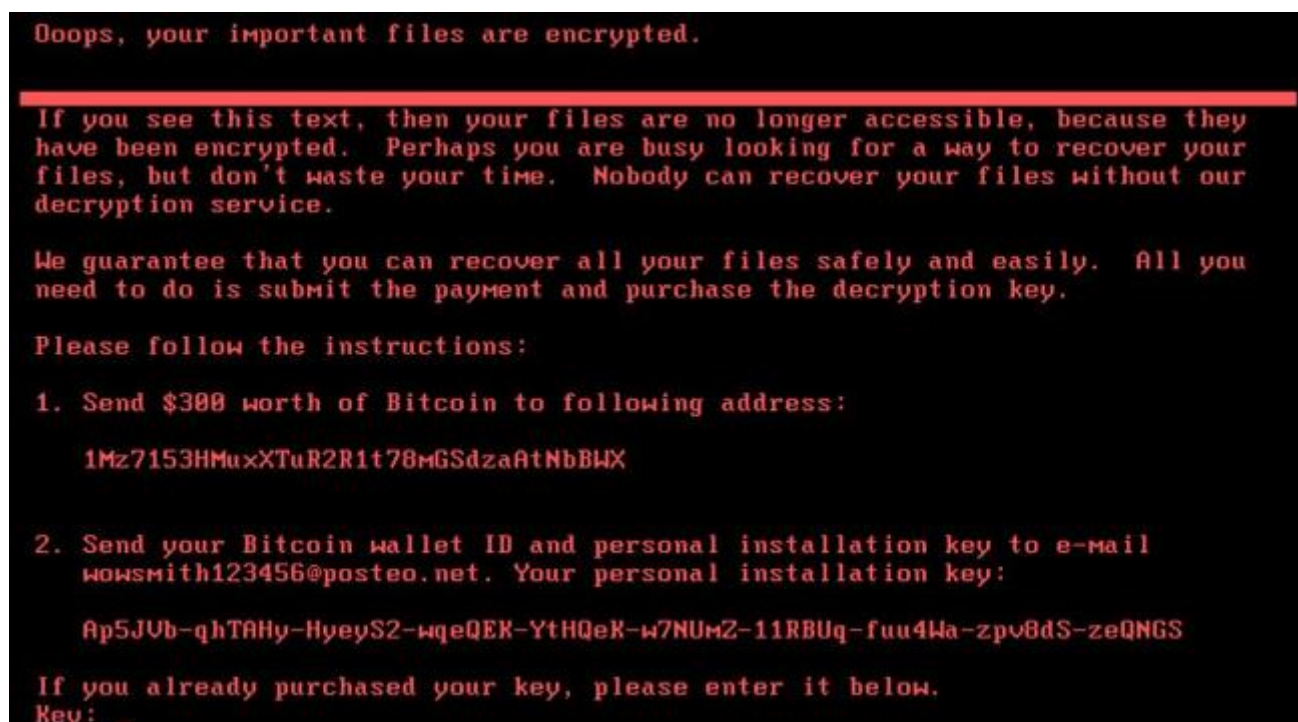


Рисунок 1.3 – Лист, якого розсилав вірус-вимагач

Україна стала свідком надзвичайної активності кібератак протягом повномасштабної війни, що відбулася в 2020–2022 роках. Кількість таких атак раптово зросла: у 2020 році їх було зафіксовано 800, у 2021 році – 1400, а у 2022 році ця цифра перевищила 4000. До того ж, у 2021 році кількість підозрілих подій в сфері інформаційної безпеки постійно зростала, і за цей період Служба безпеки України виявила 19 мільйонів подібних подій. Однак 2022 рік відзначився особливою інтенсивністю кібератак на різноманітні державні установи, системи та служби в Україні.

14 січня 2022 року відбулася одна з наймасштабніших кібератак на Україну за останні чотири роки, яка спрямовувалася на урядові веб-сайти. Деякі з сайтів були повідомлені про те, що всі їхні особисті дані стали доступними в мережі, а також про загрозу знищення інформації на комп'ютерах. Серед атакованих сайтів були Урядовий портал, веб-ресурси Міністерства освіти, закордонних справ, енергетики, молоді та спорту, екології, агрополітики, а також Державної служби з надзвичайних ситуацій та інші [18].

		Самчук Д.О.			КІК.КР.Б – ПЗ	Арк.
		Головня О.С.				29
Змн.	Арк.	№ докум.	Підпис	Дата		

15 лютого 2022 року були об'єктом DDoS-атак 15 банківських сайтів, а також сайти з доменом gov.ua, включаючи веб-ресурси Міноборони та інших державних установ. Через кілька днів, 23 лютого, атаки були повторено на державні та банківські сайти України. За цей період також стало відомо про атаки на системи Укренерго та супутникову мережу КА-SAT, що спричинили перебої у роботі кількох державних органів та підприємств.

Початок повномасштабної війни відкрив широкі можливості для проведення щоденних кібератак на різні об'єкти в Україні. Спеціалісти Департаменту кібербезпеки СБУ за цей період заблокували понад 4,5 тисячі таких атак. За інформацією Державної служби спеціального зв'язку та захисту інформації, протягом 2022 року було зафіксовано понад дві тисячі кібератак з боку Росії на державні органи України [18].

Основні цілі російських кібератак включають шпигунство, знищення критичної інфраструктури, інформаційно-психологічний вплив, а також отримання даних про громадян і системи переміщення всередині країни. Для посилення кіберзахисту Україна встановлює співпрацю з рядом країн, включаючи США, Велику Британію, Польщу, Німеччину, Швецію та Японію.

В цілому, ці атаки відображають складність та ризики, пов'язані з кібербезпекою в сучасному світі. Інциденти такого роду підкреслюють необхідність посилення заходів з кіберзахисту та розвитку відповідних стратегій, які б забезпечували надійну захищеність інформаційних систем в умовах постійно зростаючих загроз.

1.4. Способи захисту веб-сервера від атак

У сучасному цифровому середовищі захист веб-серверів від різноманітних кібератак стає дедалі більшою проблемою для організацій та підприємств. З огляду на постійно зростаючу кількість загроз інформаційній безпеці, розробники та адміністратори веб-серверів шукають ефективні та надійні методи захисту своїх систем від потенційних атак.

В сучасному світі існує велика кількість засобів для забезпечення кібербезпеки веб-серверів, і кожен з них має свої власні функції та особливості. Ось кілька

		Самчук Д.О.			КІК.КР.Б – ПЗ	Арк.
		Головня О.С.				
Змн.	Арк.	№ докум.	Підпис	Дата		30

наявних аналогів, які можна порівняти для вибору найбільш підходящого заходу захисту.

1. Akamai. Це глобальна мережева платформа, яка надає послуги кібербезпеки для веб-серверів. Вона пропонує рішення для захисту від DDoS-атак, WAF, CDN для швидкого розповсюдження веб-ресурсів, аналітику безпеки та інші функціональності. Akamai володіє широким мережевим покриттям та має розширені можливості для захисту веб-серверів від різних загроз.

2. Barracuda Web Application Firewall. Це апаратне або програмне забезпечення, розроблене для захисту веб-застосунків від атак і уразливостей. Воно включає в себе WAF, захист від DDoS, перевірку безпеки, фільтрацію вхідних даних та інші функції безпеки. Barracuda WAF може працювати в режимі активного моніторингу та блокування потенційно шкідливого трафіку.

3. Cloudflare. Це хмарна платформа, що пропонує широкий спектр послуг кібербезпеки для веб-серверів. Сервіси Cloudflare включають фаєрволи, DDoS-захист, SSL-шифрування, WAF, CDN та інші. Вони діють як посередники між клієнтами і веб-сервером, фільтруючи та обробляючи трафік для забезпечення безпеки.

4. Fortinet FortiWeb. Це рішення веб-безпеки, яке надає WAF, захист від DDoS, виявлення та захист від загроз на рівні протоколу, фільтрацію вхідних даних та інші функції. FortiWeb є інтегрованою частиною платформи безпеки Fortinet, що дозволяє забезпечити комплексний захист для веб-серверів.

5. ModSecurity. Цей веб-застосунок діє як модуль для веб-серверів Apache, Nginx та IIS. Його головна мета - захист веб-застосунків від різних видів атак, включаючи SQL Injection, XSS та інші. ModSecurity відзначається гнучкими налаштуваннями, правилами безпеки та можливістю моніторингу і реагування на інциденти безпеки.

6. Sucuri. Це інтернет-платформа для безпеки веб-сайтів, що пропонує послуги моніторингу, виявлення та виправлення вразливостей, захист від DDoS-атак та інших загроз. Sucuri надає WAF, мережеві файрволи, аналітику безпеки та інші інструменти для захисту веб-серверів [12].

		Самчук Д.О.			КІК.КР.Б – ПЗ	Арк.
		Головня О.С.				31
Змн.	Арк.	№ докум.	Підпис	Дата		

Кожен з цих аналогів має свої переваги і характеристики, які можуть відрізнятися залежно від потреб і вимог конкретної організації. Більш детальна характеристика, опис переваг та недоліків наведено в Додатку Б.

При виборі рішення для забезпечення кібербезпеки веб-серверів, необхідно враховувати специфічні потреби конкретної організації. Деякі ключові аспекти, на які варто звернути увагу, включають наступні:

1. Легкість використання: необхідно обирати рішення, яке легко налаштовується та керується. Інтерфейс користувача повинен бути зрозумілим та інтуїтивно зрозумілим, щоб адміністраторам було зручно працювати з ним.

2. Масштабованість: Слід переконатися, що рішення може масштабуватися разом із зростанням обсягу трафіку та потреб вашої організації. Воно повинне бути готове до збільшення обсягів роботи без втрати ефективності.

3. Підтримка та оновлення: Необхідно переконатися в наявності доступу до достатньої підтримки та регулярних оновлень від виробника рішення. Важливо, щоб постачальник готовий був надавати допомогу в разі виникнення проблем та швидко реагувати на нові загрози.

4. Продуктивність: бажано враховувати вплив обраного рішення на продуктивність веб-серверів. Важливо, щоб захист не зменшував швидкість роботи системи та не призводив до затримок у обробці запитів користувачів.

5. Рівень захисту: необхідно оцінити, наскільки ефективно рішення захищає від різних видів атак, таких як SQL Injection, XSS, DDoS тощо. Важливо вибрати рішення, яке найбільш оптимально відповідає потребам організації з точки зору рівня захисту.

6. Сумісність: необхідно переконатися, що обране рішення сумісне з веб-серверами та іншими системами, що використовуються в інфраструктурі. Інтеграція нового рішення повинна бути безперешодною і не викликати конфліктів з існуючими компонентами.

7. Цінова політика: слід оцінити вартість рішення, включаючи як витрати на впровадження, так і подальшу підтримку. Слід обирати рішення, яке відповідає бюджету, але при цьому забезпечує необхідний рівень захисту.

		Самчук Д.О.			КІК.КР.Б – ПЗ	Арк.
		Головня О.С.				
Змн.	Арк.	№ докум.	Підпис	Дата		32

Врахування цих ключових аспектів допоможе знайти оптимальне рішення для захисту веб-серверів організації.

У сучасному світі кібербезпека стає однією з найбільш актуальних проблем для бізнесу. Інвестування у заходи забезпечення кібербезпеки доцільно до виникнення кіберінциденту, адже наслідки таких інцидентів можуть суттєво перевищити витрати на захист. Проте, за даними Олесі Данильченко (керівниця Форуму Безпеки Розрахунків та Кредитів Української Міжбанківської Асоціації членів платіжних систем ЄМА; членкиня групи радників з фінансових сервісів Європейського Центру боротьби з кіберзлочинністю Європолу,) багато компаній недостатньо усвідомлюють це правило і часто чекають на кібератаку, перш ніж реалізувати необхідні заходи [18].

Ще одним ключовим аспектом для бізнесу є розуміння того, що просте придбання дорогого обладнання та програмного забезпечення не гарантує повноцінного захисту. За словами експертки, важливо усвідомлювати, що ефективний кіберзахист вимагає комплексного підходу, який включає як технічні рішення, так і належну експертизу фахівців.

У рамках наукового підходу до проблеми кібербезпеки, важливо враховувати рекомендації фахівців. Крім того, експертка надає кілька порад для бізнесу і громадян стосовно захисту у кіберпросторі:

- 1) потрібно налаштувати обов'язкову двофакторну аутентифікацію;
- 2) не можна відкривати листи з вкладеннями від невідомих відправників;
- 3) не можна переходити за посиланнями, отриманими у месенджерах від невідомих відправників;
- 4) треба ігнорувати усі повідомлення в усіх каналах комунікацій, які вимагають від вас швидких необдуманих дій – справа у тому, що слова і фрази на кшталт “негайно”, “терміново”, “потрібна допомога”, “для вашої безпеки і безпеки ваших рідних”, “акція”, “вигідна пропозиція”, “залишилось 2 дні”, “тільки ви” тощо є улюбленими для зловмисників. Тож перш ніж ухвалити будь-яке рішення, потрібно спокійно порадити до десяти і перевірити інформацію по інших каналах;

		Самчук Д.О.			КІК.КР.Б – ПЗ	Арк.
		Головня О.С.				
Змн.	Арк.	№ докум.	Підпис	Дата		33

5) треба перевіряти сайти, яким плануєте довірити персональні, ідентифікаційні та/або карткові дані через сервіс Check My Link Асоціації ЄМА [18].

Ми вважаємо, що для підвищення безпеки та запобігання зловмисним атакам необхідно використовувати декілька варіантів захисту.

1. Багатоетапна перевірка достовірності: рекомендується впровадження системи багатоетапної перевірки достовірності, яка охоплює перевірку даних, введених через інтерфейс користувача, а також на всіх наступних етапах перетину кордонів довіреної зони. Це дозволить ефективно відсікати як випадкових, так і цілеспрямованих зловмисників.

2. Перевірка та обробка введених даних: рекомендується не поєднувати введені користувачем дані без перевірки, оскільки об'єднання рядків може стати основною точкою входу для впровадження вразливих сценаріїв.

3. Уникання небезпечних імен файлів: важливо уникати використання певних небезпечних імен файлів у полях, які можуть бути використані для створення файлів. Наприклад, слід уникати використання імен файлів, які можуть викликати системні або безпекові проблеми, такі як AUX, CON, LPT1-LPT8, NUL і інші.

4. Обробка помилок після збереження запису користувача: створення механізму обробки помилок після збереження запису користувача в базі даних. Це включає реагування на можливі помилки, що можуть виникнути під час збереження даних.

5. Захист від атак на перебір паролів: впровадження заходів для запобігання атакам на перебір паролів, таких як обмеження кількості спроб введення пароля, застосування капчі, блокування облікового запису після кількох невдалих спроб тощо.

6. Використання сильних паролів та заборона використання слабких паролів: навчання користувачів використовувати складні паролі, а також встановлення правил для заборони використання слабких паролів.

7. Використання параметризованих запитів до бази даних: використання параметризованих запитів до бази даних для запобігання SQL Injection атак.

8. Реалізація журналювання подій і помилок за допомогою бібліотеки Winston: використання бібліотеки Winston для журналювання подій і помилок, що

		Самчук Д.О.			КІК.КР.Б – ПЗ	Арк.
		Головня О.С.				
Змн.	Арк.	№ докум.	Підпис	Дата		34

відбуваються на веб-сервері, для забезпечення детального моніторингу системи та виявлення можливих загроз.

9. Використання моніторингу систем для виявлення незвичайної активності та потенційних загроз: використання спеціалізованих систем моніторингу для виявлення незвичайної активності або потенційних загроз на веб-сервері для оперативної реакції та запобігання інцидентам.

Ці заходи допоможуть забезпечити ефективний рівень захисту веб-сервера та зменшити ймовірність успішних атак з боку злоумисників.

Таким чином, способи захисту веб-сервера від атак є критично важливими для забезпечення безпеки та недопущення негативних наслідків, таких як втрата даних, порушення конфіденційності або доступ до важливих ресурсів. Використання комплексного підходу, що охоплює різноманітні методи захисту, від валідації даних до моніторингу систем, дозволяє створити ефективну систему захисту, яка зменшує ризик вразливостей та підвищує рівень безпеки.

1.5. Постановка технічного завдання

Дане завдання передбачає розробку та налаштування підсистеми захисту для веб-сервера організації з метою забезпечення надійного захисту від кібератак і гарантування доступності сервісів. В цьому розділі проведено детальний аналіз потенційних загроз та вразливостей, що можуть вплинути на безпеку веб-сервера, а також розглянуті технології захисту та їх переваги та недоліки. Ось деякі ключові загрози та вразливості, пов'язані з веб-серверами:

1. Крос-сайтовий скриптинг (XSS). Ця вразливість дозволяє злоумиснику впровадити зловисний скрипт у веб-сторінку, що може призвести до виконання шкідливих дій на стороні клієнта, таких як крадіжка облікових даних або перенаправлення на шкідливі сайти.

2. SQL-ін'єкція. Ця вразливість дозволяє злоумиснику впровадити шкідливий SQL-запит у веб-додаток, що може призвести до несанкціонованого доступу до бази даних або виконання небажаних операцій.

		Самчук Д.О.			КІК.КР.Б – ПЗ	Арк.
		Головня О.С.				
Змн.	Арк.	№ докум.	Підпис	Дата		35

3. Несанкціонований доступ. Ця загроза полягає в спробі зломисника проникнути в систему веб-сервера для отримання доступу до конфіденційних файлів або налаштувань сервера.

4. Відмова в обслуговуванні (DoS) і розподілена відмова в обслуговуванні (DDoS). Ці типи атак спрямовані на перевантаження веб-сервера або мережі, що може призвести до відмови в обслуговуванні для легітимних користувачів.

5. Вразливості в програмному забезпеченні. Ця загроза виникає через наявність вразливостей у веб-сервері або використаного програмного забезпечення, що може бути використане зломисниками для злому або компрометації сервера.

6. Недостатня автентифікація та авторизація. Якщо система автентифікації та авторизації не належним чином налаштована, це може дозволити зломисникам отримати несанкціонований доступ до захищених ресурсів або функціональностей веб-сервера.

Для успішного впровадження заходів захисту необхідно врахувати ці загрози та вразливості та приділити належну увагу впровадженню відповідних заходів безпеки для захисту веб-сервера та даних організації.

З проведеного аналізу випливає, що веб-сервери набувають все більшого значення в сучасному інформаційному середовищі, але водночас стають об'єктом зростаючого числа кібератак. Оскільки організації стають все більш залежними від інформаційних технологій, забезпечення ефективного захисту веб-серверів стає критичною необхідністю.

Основними завданнями, які стоять перед розробкою підсистеми захисту веб-сервера, є наявність

1. Впровадження захисту від атак, спрямованих на перебір паролів та підвищення рівня безпеки авторизаційних процедур.

2. Забезпечення використання сильних паролів та блокування можливості використання слабких або легко вгадуваних паролів.

3. Розробка та впровадження механізмів валідації введених користувачем даних для запобігання вразливостям типу XSS та SQL Injection.

		Самчук Д.О.			КІК.КР.Б – ПЗ	Арк.
		Головня О.С.				36
Змн.	Арк.	№ докум.	Підпис	Дата		

4. Встановлення індивідуальних правил валідації для різних типів запитів до веб-сервера.

5. Реалізація ефективної системи обробки помилок після збереження записів користувачів з метою уникнення потенційних загроз безпеці.

6. Розробка та впровадження заходів захисту від атак, спрямованих на введення некоректних або шкідливих даних в систему.

7. Використання параметризованих запитів до бази даних для запобігання SQL Injection та інших подібних атак.

8. Валідація формату даних та застосування механізмів обмеження обсягу введених даних для запобігання переповнення буфера або інших атак.

9. Реалізація системи журналювання подій і помилок з використанням бібліотеки Winston з метою вчасного виявлення незвичайної активності та потенційних загроз безпеці.

10. Використання моніторингу систем для виявлення незвичайної активності та потенційних загроз.

		Самчук Д.О.			КІК.КР.Б – ПЗ	Арк.
		Головня О.С.				37
Змн.	Арк.	№ докум.	Підпис	Дата		

РОЗДІЛ 2 ПРОЕКТУВАННЯ ТА РОЗРОБКА ПІДСИСТЕМИ ЗАХИСТУ ВЕБ-СЕРВЕРА

2.1. Опис середовища розробки для сервера та клієнта

Охорона веб-сервера від зловмисних атак та непередбачених подій є одним із найважливіших завдань у сфері кібербезпеки. У цьому розділі ми детально розглянемо методи, які використовуються для захисту веб-сервера, та процес їх проектування та розробки.

Почнемо з опису середовища розробки для сервера та клієнта. Вивчення та розуміння цих аспектів дозволить нам ефективно розробити та впровадити заходи захисту, необхідні для забезпечення безпеки веб-сервера та його користувачів.

Середовище розробки для серверу сайту кінотеатру включає ряд компонентів та інструментів, які сприяють ефективній розробці та забезпечують надійність програмного забезпечення.

1. Редактор коду: для створення серверної та клієнтської частини нашої додатку ми використовували Visual Studio Code, який надає зручне середовище розробки.

2. Node.js: використання Node.js дозволило нам створити серверну частину додатку. Ця відкрита платформа заснована на движку V8 JavaScript і дозволяє виконувати JavaScript-код на сервері.

3. Фреймворк Express: для створення серверної частини ми використали фреймворк Express, який надає зручні інструменти для роботи з HTTP-запитами, маршрутизації та обробки даних.

4. База даних: для зберігання даних ми використали MongoDB, що дозволяє ефективно керувати та зберігати інформацію.

5. React: використання бібліотеки React на фронтенді дозволило нам побудувати ефективний та реактивний інтерфейс для користувачів.

6. Тестування: для забезпечення якості та надійності ми використовували різноманітні техніки тестування, такі як модульні тести та інтеграційні тести, за допомогою інструменту Jest.

		Самчук Д.О.			КІК.КР.Б – ПЗ	Арк.
		Головня О.С.				
Змн.	Арк.	№ докум.	Підпис	Дата		38

7. Система контролю версій: ми використовували систему контролю версій Git для спільної роботи над проектом та відстеження змін в коді.

8. Пакетний менеджер: для управління залежностями ми використовували npm, що дозволяє легко встановлювати, оновлювати та керувати пакетами JavaScript.

Прийняття рішення використовувати Node.js та фреймворк Express для проектування сервера для веб-сайту кінотеатру має кілька переваг:

1. Швидкодія та ефективність: Node.js використовує подієвий та неблокуючий ввід/вивід, що дозволяє ефективно обробляти багато запитів одночасно без блокування потоків. Це особливо корисно для веб-сайтів з високим навантаженням, таких як кінотеатри, де потрібна швидка обробка багатьох запитів одночасно.

2. Єдина мова програмування: використання JavaScript як мови програмування для інтерфейсу серверу та клієнта спрощує розробку, тестування та підтримку проекту, оскільки розробники можуть працювати з однією мовою на обох сторонах додатку.

3. Велика спільнота та екосистема: Node.js має широку та активну спільноту розробників, яка підтримує велику кількість документації, модулів та інструментів. Це дозволяє швидко знаходити рішення для різних завдань та забезпечує надійність та стабільність проекту.

4. Простота використання та розширення: Express є легким та гнучким веб-фреймворком, який пропонує простий та зрозумілий інтерфейс для розробки веб-додатків. Він дозволяє розробникам швидко створювати та розширювати функціонал сервера за допомогою шаблонів маршрутів та розширень (middleware).

5. Розширюваність та підтримка RESTful архітектури: Express надає зручні засоби для реалізації RESTful API, що сприяє ефективній організації маршрутів та обробці HTTP-запитів. Це дозволяє створювати сучасні та ефективні веб-додатки.

В цілому, використання Node.js та Express надає потужну та ефективну платформу для розробки серверної частини веб-сайту кінотеатру з можливістю швидкого розширення та забезпечення високої продуктивності.

		Самчук Д.О.			КІК.КР.Б – ПЗ	Арк.
		Головня О.С.				
Змн.	Арк.	№ докум.	Підпис	Дата		39

2.2. Проектування архітектури підсистеми захисту

Проектування архітектури підсистеми захисту є ключовим етапом в розробці будь-якого веб-сервера, особливо в умовах постійних загроз та вразливостей в сучасному цифровому середовищі. Правильно спроектована архітектура захисту дозволяє ефективно захищати сервер від різноманітних видів атак, забезпечуючи високий рівень безпеки для користувачів та даних. Тому в цьому пункті ми розглянемо процес проектування архітектури підсистеми захисту для нашого веб-сервера кінотеатру, визначивши ключові етапи, методи та стратегії захисту, що допоможуть забезпечити стійку та надійну роботу сервера у вимогливих умовах онлайн середовища.

Аналіз вимог до безпеки є критичним етапом проектування будь-якої системи, зокрема веб-сервера кінотеатру. Він визначає потенційні загрози та вразливості системи, а також встановлює вимоги до безпеки, необхідні для захисту даних та забезпечення надійності та конфіденційності інформації. Вважаємо, що основними вимогами до безпеки веб-сервера кінотеатру є перелічені нижче

1. Ідентифікація загроз та вразливостей: перший крок у аналізі вимог до безпеки – це ідентифікація потенційних загроз та вразливостей системи. У випадку веб-сервера кінотеатру, це можуть бути атаки на основі введення шкідливих даних, атаки на перебір паролів, атаки DDoS, а також можливість витоку конфіденційної інформації про користувачів, таких як особисті дані чи фінансова інформація.

2. Оцінка потенційного впливу: після ідентифікації загроз та вразливостей, потрібно оцінити потенційний вплив цих загроз на систему. Наприклад, втрата конфіденційної інформації про користувачів може призвести до порушення їхньої приватності та психологічного дискомфорту, що може негативно позначитися на репутації кінотеатру та призвести до втрати клієнтів.

3. Встановлення вимог до безпеки: на основі ідентифікації загроз та оцінки їхнього потенційного впливу, встановлюються вимоги до безпеки. Це можуть бути вимоги до захисту конфіденційності, цілісності та доступності даних, вимоги до захисту від атак та забезпечення безпеки мережевого зв'язку.

		Самчук Д.О.			КІК.КР.Б – ПЗ	Арк.
		Головня О.С.				40
Змн.	Арк.	№ докум.	Підпис	Дата		

4. Розробка стратегій та методів захисту: на основі встановлених вимог до безпеки, розробляються стратегії та методи захисту системи. Це можуть бути заходи щодо валідації введених даних, захист від атак на перебір паролів, реалізація системи моніторингу для виявлення аномальної активності тощо;

5. Тестування та оновлення: не менш важливим є тестування розроблених стратегій та методів захисту. Після впровадження системи в експлуатацію варто регулярно проводити аудит безпеки, виявляти нові загрози та вразливості, а також вносити необхідні оновлення та покращення;

Аналіз вимог до безпеки є невід’ємною частиною проектування будь-якої системи, що допомагає забезпечити її надійність та захищеність від потенційних загроз.

Обробка CORS включає встановлення middleware для обробки запитів CORS у веб-додатку. Це важливий аспект з точки зору безпеки, оскільки він контролює доступ до ресурсів сервера з інших доменів. Встановлення відповідних заголовків дозволяє контролювати, які домени мають доступ до сервера, що сприяє запобіганню атакам, таким як CSRF та XSS, та підвищує безпеку додатку.

Middleware – auth використовується для автентифікації користувачів у веб-додатку. Він перевіряє наявність користувача у базі даних за їхнім електронним листом, створює JWT-токен для авторизації та додає його до запиту. Це дозволяє контролювати доступ до захищених маршрутів у додатку. Якщо користувач не автентифікується або надає невірні облікові дані, middleware повертає статус 401 та повідомлення про необхідність аутентифікації.

Ми використовуємо різні маршрути (routes), що обробляють різні типи запитів до сервера. Кожен з цих маршрутів відповідає за обробку запитів, пов’язаних із конкретними ресурсами або функціоналом додатку:

– userRouter – маршрут, який відповідає за обробку запитів, пов’язаних з користувачами, такими як реєстрація, вхід, отримання інформації про користувача тощо;

		Самчук Д.О.			КІК.КР.Б – ПЗ	Арк.
		Головня О.С.				41
Змн.	Арк.	№ докум.	Підпис	Дата		

– cinemaRouter – маршрут, що обробляє запити, пов’язані з кінотеатрами, такі як отримання списку кінотеатрів, додавання нового кінотеатру, редагування інформації про кінотеатр тощо;

– movieRouter – маршрут, який відповідає за обробку запитів, пов’язаних з фільмами, такими як отримання списку фільмів, додавання нового фільму, видалення фільму тощо;

– showtimeRouter – маршрут, який відповідає за обробку запитів, пов’язаних з графіком сеансів у кінотеатрі, такими як отримання розкладу сеансів, додавання нового сеансу, оновлення інформації про сеанс тощо;

– reservationRouter – маршрут, який відповідає за обробку запитів, пов’язаних з бронюванням квитків на сеанси у кінотеатрі, такими як створення нового бронювання, отримання списку заброньованих квитків, скасування бронювання тощо;

– invitationsRouter: маршрут, що обробляє запити, пов’язані зі запрошеннями на події або фільми, такі як надсилання запрошень, прийняття або відхилення запрошення тощо.

Кожен з цих маршрутів дозволяє взаємодіяти з відповідними ресурсами або функціоналом у веб-додатку через визначені HTTP-запити (GET, POST, PUT, DELETE).

Розглянути кожен маршрут у userRouter та його функціональність (табл.2.1).

Таблиця 2.1

Опис основних маршрутів userRouter

Метод	Маршрут	Опис	Дія
1	2	3	4
POST	/users	Створення нового користувача у системі	Перевірка та збереження нового користувача у базі даних
POST	/users/photo/:id	Завантаження фотографії профілю користувача	Збереження шляху до зображення в профіль користувача
POST	/users/login	Аутентифікація користувача в системі	Перевірка введеного логіну та пароля
POST	/users/logout	Вихід користувача із системи	Видалення токена доступу, пов'язаного із користувачем

		Самчук Д.О.			КІК.КР.Б – ПЗ	Арк.
		Головня О.С.				42
Змн.	Арк.	№ докум.	Підпис	Дата		

Продовження таблиці 2.1

1	2	3	4
POST	/users/logoutAll	Вихід користувача з усіх пристроїв	Видалення всіх токенів доступу користувача
GET	/users	Отримання списку всіх користувачів у системі	Повернення списку користувачів
GET	/users/me	Отримання власних даних користувача	Повернення даних профілю поточного користувача
GET	/users/:id	Отримання даних конкретного користувача	Повернення даних профілю користувача за його ідентифікатором
PATCH	/users/me	Редагування власного профілю користувача	Оновлення відповідних полів профілю користувача
PATCH	/users/:id	Редагування профілю конкретного користувача	Оновлення відповідних полів профілю користувача за його ідентифікатором
DELETE	/users/:id	Видалення користувача за його ідентифікатором	Видалення профілю користувача за його ідентифікатором
DELETE	/users/me	Видалення власного профілю користувача	Видалення профілю користувача та всіх пов'язаних даних

Таблиця 2.1 надає зручний та структурований огляд кожного маршруту в `userRouter`, його опису та відповідних дій.

Модель користувача включає наступні поля.

- `name`: рядок, що представляє ім'я користувача. Обов'язкове поле.
- `username`: рядок, що визначає унікальне ім'я користувача. Також обов'язкове поле.
- `email`: рядок, що визначає унікальну електронну адресу користувача. Обов'язкове поле та піддається валідації за допомогою перевірки формату електронної адреси.
- `password`: рядок, що містить пароль користувача. Пароль також піддається валідації, а саме: мінімальна довжина 7 символів та відсутність слова “password” у паролі.
- `role`: рядок, що визначає роль користувача (наприклад: гость, адміністратор, суперадміністратор). По замовчуванню встановлений рівень доступу “guest”.

– phone: рядок, що визначає номер телефону користувача. Унікальне поле, яке також піддається валідації за допомогою перевірки формату номеру телефону.

– imageUrl: рядок, що містить URL зображення користувача.

– tokens: масив, який містить токени доступу користувача для автентифікації. Кожен токен є об'єктом, що містить рядок токenu.

Додатково, модель має поля createdAt та updatedAt, які автоматично встановлюються для відстеження дати створення та останнього оновлення запису про користувача.

Розглянути кожен маршрут у cinemaRouter та його функціональність (табл.2.2).

Таблица 2.2

Опис основних маршрутів userRouter

Метод 1	Маршрут 2	Опис 3	Дія 4
POST	/cinemas	Даний роут призначений для створення нового кінотеатру в системі. Вимагає передачі в тілі запиту даних про кінотеатр (наприклад: назва, місто тощо).	Створення нового запису про кінотеатр у базі даних і відправлення успішного відповіді зі статусом 201 у разі успіху, або відправлення помилки зі статусом 400 у разі невдачі.
POST	/cinemas/photo/:id	Цей роут дозволяє завантажувати фото для певного кінотеатру за його ідентифікатором.	Збереження фото кінотеатру у системі і оновлення відповідного запису у базі даних.
GET	/cinemas	Роут призначений для отримання списку всіх кінотеатрів, збережених у системі.	Отримання списку кінотеатрів і відправлення їх у відповіді.
GET	/cinemas/:id	Даний роут дозволяє отримати інформацію про конкретний кінотеатр за його ідентифікатором.	Пошук кінотеатру за вказаним ідентифікатором і відправлення його у відповіді.
PATCH	/cinemas/:id	Цей роут дозволяє змінити інформацію про кінотеатр за його ідентифікатором.	Оновлення відповідного запису про кінотеатр у базі даних з новими даними, що передаються у тілі запиту.
DELETE	/cinemas/:id	Роут призначений для видалення кінотеатру за його ідентифікатором.	Видалення відповідного запису про кінотеатр з бази даних.

Модель кінотеатру включає наступні поля:

- name: рядок, який представляє назву кінотеатру. Обов’язкове поле.
 - ticketPrice: число, яке визначає вартість квитка у кінотеатрі. Обов’язкове поле.
 - city: рядок, що вказує на місто, де розташований кінотеатр. Обов’язкове поле.
 - seats: масив змішаних типів даних, який представляє доступні місця у кінозалі. Обов’язкове поле.
 - seatsAvailable: число, що показує загальну кількість доступних місць у кінозалі. Обов’язкове поле.
 - image: рядок, який містить URL-адресу зображення кінотеатру.
- Маршрути для фільмів включають наступні дії (табл.2.3).

Таблиця 2.3

Опис основних маршрутів userRouter

Метод	Маршрут	Опис	Дія
POST	/movies	Додає новий фільм за наданими даними	Створює новий запис фільму в базі даних
GET	/movies/photo/:id	Завантажує зображення для фільму за його ідентифікатором	Завантажує зображення та зберігає його шлях разом з інформацією про фільм
GET	/movies	Отримує список усіх фільмів	Повертає список усіх фільмів з бази даних
GET	/movies/:id	Оновлює інформацію про фільм за його ідентифікатором	Оновлює вказані поля для фільму зазначеного ідентифікатора
DELETE	/movies/:id	Видаляє фільм за його унікальним ідентифікатором	Видаляє запис про фільм з бази даних за його ідентифікатором
GET	/movies/usermodeling/:username	Отримує рекомендації щодо фільмів для конкретного користувача	Повертає список рекомендованих фільмів для користувача з вказаним ім'ям

Модель фільму включає наступні поля:

- title: рядок, що представляє назву фільму. Обов’язкове поле. Зберігається в нижньому регістрі.
- image: рядок, що містить шлях до зображення або постера фільму.
- language: рядок, що визначає мову фільму. Обов’язкове поле. Зберігається в нижньому регістрі.

– genre: рядок, що вказує на жанр фільму. Обов’язкове поле. Зберігається в нижньому регістрі.

– director: рядок, що містить ім’я режисера фільму. Обов’язкове поле. Зберігається в нижньому регістрі.

– cast: рядок, що містить ім’я акторів або актрис, що знялися у фільмі. Обов’язкове поле. Зберігається в нижньому регістрі.

– description: рядок, що містить опис або сюжет фільму. Обов’язкове поле. Зберігається в нижньому регістрі.

– duration: число, що вказує на тривалість фільму у хвилинах. Обов’язкове поле.

– releaseDate: дата, коли фільм був випущений. Обов’язкове поле.

– endDate: дата, коли показ фільму закінчився. Обов’язкове поле.

Ми ретельно розглянули основні маршрути та моделі, що відповідають за обробку запитів і управління даними у нашому веб-додатку кінотеатру. Основні моделі, такі як модель користувача, модель фільму та модель кінотеатру, були детально описані з вказанням полів та їх характеристик. Щодо маршрутів, таких як `showtimeRouter`, `reservationRouter` та `invitationsRouter`, ми також надали загальний опис їхнього призначення та функціональності, однак не деталізували кожен з них, оскільки вони вже відображають стандартний набір операцій, які виконуються над відповідними ресурсами.

Для зберігання даних у нашому веб-додатку ми використовуємо базу даних MongoDB, яка забезпечує надійність та швидкість доступу до інформації. Підключення до бази даних здійснюється за допомогою сервісу MongoDB Atlas, що забезпечує гнучкість та масштабованість в роботі з даними. Для забезпечення безпеки та контролю доступу до інформації, ми створили окремого користувача з правами адміністратора, що дозволяє ефективно керувати базою даних. Це забезпечує нам можливість ефективно працювати з даними користувачів, інформацією про кінотеатри, кінофільми та їх сеанси, забезпечуючи їхню конфіденційність та безпеку взаємодії.

Вибір методів та стратегій захисту веб-сервера – це критичний етап у процесі забезпечення безпеки системи. Для ефективного захисту необхідно враховувати

		Самчук Д.О.			КІК.КР.Б – ПЗ	Арк.
		Головня О.С.				
Змн.	Арк.	№ докум.	Підпис	Дата		46

потенційні загрози та вразливості, а також відповідати вимогам безпеки відповідно до контексту використання веб-сервера кінотеатру. Розглянемо методи та стратегії захисту, які можна використовувати:

1. Валідація введених даних користувачів:

- перевірка введених даних на відповідність формату, типу та обсягу.
- застосування механізмів фільтрації та санітаризації даних для запобігання введенню шкідливих або неочікуваних даних.

2. Захист від атак на перебір паролів:

- обмеження кількості спроб введення пароля за певний період часу.
- використання капчі або двофакторної аутентифікації для ускладнення процесу перебору паролів.

3. Використання сильних паролів та заборона використання слабких паролів:

- застосування політики паролів, яка вимагає від користувачів використовувати паролі з достатньою складністю.
- проведення навчання користувачів щодо важливості та методів створення сильних паролів.

4. Захист від атак на введення даних:

- використання механізмів фільтрації та валідації даних для запобігання введенню SQL-ін'єкцій, XSS-атак та інших шкідливих втручань.
- шифрування чутливих даних перед їх передачею на сервер та зберіганням у базі даних.

5. Використання параметризованих запитів до бази даних:

- використання параметризованих запитів для запобігання SQL-ін'єкціям, які можуть виникнути внаслідок використання несанкціонованих або некоректних запитів.

6. Валідація формату даних та застосування механізмів обмеження обсягу даних:

- перевірка формату та обсягу даних перед їх обробкою для запобігання переповненню буфера або інших атак на обробку даних.

7. Аудит подій та помилок:

		Самчук Д.О.			КІК.КР.Б – ПЗ	Арк.
		Головня О.С.				47
Змн.	Арк.	№ докум.	Підпис	Дата		

– реалізація системи журналювання, яка веде запис подій та помилок, що виникають під час роботи системи.

– аналіз журналів для виявлення патернів атак та ненормальної активності.

8. Моніторинг системи:

– встановлення систем моніторингу, які перевіряють роботу сервера та виявляють незвичайну активність чи підозрілі події.

– реагування на виявлені загрози та вжиття заходів для їх ліквідації або запобігання.

Крім того, важливо забезпечити постійне оновлення та моніторинг безпекових патчів та застосунків, щоб забезпечити високий рівень безпеки системи впродовж її життєвого циклу.

2.3. Реалізація функціоналу підсистеми захисту

Підсистема захисту є однією з ключових складових веб-додатку, оскільки вона відповідає за забезпечення безпеки даних користувачів та захист їхніх особистих інформаційних ресурсів від незаконного доступу та зловживань. У цьому пункті ми розглянемо реалізацію функціоналу підсистеми захисту у нашому веб-додатку, включаючи використання методів аутентифікації та авторизації, застосування шифрування та інших методів безпеки для запобігання можливим загрозам та атакам. Наша головна мета полягає в створенні надійного та безпечного середовища для користувачів, щоб вони могли впевнено користуватися нашими сервісами та мати впевненість у захищеності своїх особистих даних.

Валідація введених даних користувачів

Для забезпечення впевненості в безпеці підсистеми веб-сервера організації, обов'язковою є валідація введених даних. Цей процес передбачає перевірку формату, типів даних, наявності обов'язкових полів і дотримання встановлених обмежень. Наприклад, важливо перевіряти правильність введених електронних адрес, відповідність даних певним регулярним виразам, а також відповідність числових значень заданим межах.

		Самчук Д.О.			КІК.КР.Б – ПЗ	Арк.
		Головня О.С.				48
Змн.	Арк.	№ докум.	Підпис	Дата		

Для досягнення цієї мети ми використовуємо бібліотеку Joi, яка надає зручний та потужний інтерфейс для визначення правил валідації.

Щоб забезпечити валідацію даних користувачів, ми використовуємо Joi для перевірки введених даних користувачів перед їх обробкою.

```
const registerSchema = Joi.object({
  name: Joi.string().required(),
  username: Joi.string().alphanum().min(3).max(30).required(),
  email: Joi.string().pattern(emailRegex).required(),
  password: Joi.string()
    .min(8)
    .pattern(/^(?=.*\d)(?=.*[a-z])(?=.*[A-Z])(?=.*[!@#$$%^&*]).{8,}$/)
    .required()
    .messages({
      "string.pattern.base":
        "Password must contain at least one lowercase letter, one uppercase
        letter, one digit, and one special character (!@#$$%^&*)",
    }),
  phone: Joi.string().required(),
  imageUrl: Joi.string(),
});

const loginSchema = Joi.object({
  username: Joi.string().required(),
  password: Joi.string().required(),
});

const verifyEmailSchema = Joi.object({
  email: Joi.string().pattern(emailRegex).required(),
});
```

registerSchema використовується для валідації даних, які користувач вводить під час реєстрації в системі. Вона перевіряє, чи всі необхідні поля заповнені, чи вони відповідають встановленим обмеженням (наприклад, мінімальна та максимальна довжина для імені користувача), а також чи відповідають формату (наприклад, чи правильно введена електронна адреса або чи містить пароль необхідні елементи - цифри, велику та малу літери, спеціальні символи).

loginSchema використовується для валідації даних, які користувач вводить під час процесу входу в систему. Зазвичай це ім'я користувача (або ім'я користувача або

		Самчук Д.О.			КІК.КР.Б – ПЗ	Арк.
		Головня О.С.				49
Змн.	Арк.	№ докум.	Підпис	Дата		

електронна адреса) та пароль. Вона перевіряє, чи обидва поля заповнені, і чи вони відповідають вимогам безпеки.

verifyEmailSchema використовується для валідації електронної адреси під час процесів, пов'язаних з підтвердженням електронної адреси (наприклад, підтвердження реєстрації або відновлення паролю). Вона перевіряє, чи введена електронна адреса має правильний формат і відповідає регулярному виразу, що використовується для перевірки правильності електронної адреси.

handleSaveErrors є допоміжною функцією, яка обробляє помилки після збереження запису користувача.

emailRegexp є регулярним виразом для перевірки формату електронної пошти.

Модель користувача User створюється на основі схеми userSchema.

Також, ми додали валідаційні схеми для інших моделей у нашій системі, таких як кінотеатри, фільми, резервації та сеанси.

```
const cinemaSchemaValidation = Joi.object({
  name: Joi.string()
    .min(2)
    .regex(/^[a-zA-Z0-9 ]+$/)
    .required(),
  ticketPrice: Joi.number().required(),
  city: Joi.string().required(),
  seats: Joi.array().required(),
  seatsAvailable: Joi.number().required(),
  image: Joi.string(),
});

const movieSchemaValidation = Joi.object({
  title: Joi.string().required(),
  image: Joi.string(),
  language: Joi.string().required(),
  genre: Joi.string().required(),
  director: Joi.string().required(),
  cast: Joi.string().required(),
  description: Joi.string().required(),
  duration: Joi.number().required(),
  releaseDate: Joi.date().required(),
  endDate: Joi.date().required(),
});
```

Ці схеми дозволяють перевірити правильність введених даних для кожного типу і забезпечують безпеку та надійність нашого веб-додатку.

		Самчук Д.О.			КІК.КР.Б – ПЗ	Арк.
		Головня О.С.				50
Змн.	Арк.	№ докум.	Підпис	Дата		

Загалом, застосування бібліотеки Joi для перевірки введених даних є ефективним і надійним способом забезпечення безпеки підсистеми веб-сервера. Використання валідаційних схем гарантує коректність та надійність обробки даних перед їх використанням у системі. Ці заходи забезпечують повну впевненість у валідності введених даних перед їх збереженням або подальшим використанням, що сприяє підвищенню безпеки та надійності підсистеми веб-сервера.

Захист від атак на авторизацію та ідентифікацію

Один із ключових аспектів полягає у захисті веб-сервера від атак, спрямованих на авторизацію та ідентифікацію користувачів. Одним з методів забезпечення безпеки є реалізація процесу верифікації, який призначений для підтвердження ідентичності користувача та його облікових даних.

Для забезпечення повної безпеки та відповідності стандартам, ми розробили та впровадили процес верифікації веб-сервера. Цей процес включає кілька етапів, які користувач повинен успішно пройти для підтвердження своєї ідентичності та отримання доступу до облікового запису.

Один із стандартних методів верифікації полягає у підтвердженні електронної пошти користувача. Після реєстрації, на вказану під час реєстрації пошту користувача автоматично відправляється лист з посиланням для підтвердження їхньої електронної адреси. Пройшовши за посиланням, користувач підтверджує свою електронну пошту та пройде верифікацію.

```
router.get("/verify/:verificationToken", async (req, res) => {
  const { verificationToken } = req.params;
  const user = await User.findOne({ verificationToken });
  if (!user) {
    throw HttpError(404, `User not found`);
  }
  await User.findByIdAndUpdate(user._id, {
    verify: true,
    verificationToken: "",
  });

  res.json({
    status: "success",
  });
});
```

		Самчук Д.О.			КІК.КР.Б – ПЗ	Арк.
		Головня О.С.				
Змн.	Арк.	№ докум.	Підпис	Дата		51

```

code: 200,
data: {
  message: "Verification successful",
},
});
});

```

Процес верифікації забезпечує безпеку шляхом перевірки унікального токену верифікації для кожного користувача та оновлення значень полів “перевірено” та “унікальний токен верифікації” у базі даних. Після успішної верифікації користувача його обліковий запис вважається підтвердженим, і веб-сервер надає доступ.

Використання процесу верифікації відповідає високим стандартам безпеки, що забезпечує захист веб-сервера від потенційних атак на авторизацію та ідентифікацію користувачів. Цей механізм гарантує, що тільки підтверджені користувачі мають доступ до системи, зменшуючи ризик несанкціонованого доступу та збільшуючи загальний рівень безпеки.

Захист від атак перебору паролів

Захист від атак перебору паролів - це важлива складова безпеки для будь-якої системи, особливо для веб-додатків, де перебір паролів може бути використаний для несанкціонованого доступу до облікових записів користувачів. Для захисту від цих атак, існують кілька ефективних практик.

1. Використання сильних паролів: Система повинна вимагати від користувачів використання сильних паролів, які складаються з комбінації великих і малих літер, цифр і спеціальних символів. Чим більше символів та різноманітних символів в паролі, тим важче його перебрати.

2. Обмеження спроб введення пароля: Система повинна обмежувати кількість спроб введення пароля, перед тим як заблокувати обліковий запис на певний період часу. Це ускладнює атакуючим перебір паролів шляхом спроби-помилки.

3. Використання двофакторної аутентифікації (2FA): 2FA додає додатковий шар безпеки, вимагаючи не лише пароль для входу, але й додатковий код, який генерується або надсилається на зареєстроване пристрій користувача.

		Самчук Д.О.			КІК.КР.Б – ПЗ	Арк.
		Головня О.С.				
Змн.	Арк.	№ докум.	Підпис	Дата		52

4. Моніторинг та журналювання подій: Важливо вести моніторинг спроб введення невірних паролів та інших підозрілих активностей для вчасного виявлення можливих атак та реагування на них.

5. Регулярна зміна паролів: Рекомендується змінювати паролі з регулярністю, щоб зменшити ризик в разі, якщо пароль став відомим зломисникам через несанкціонований доступ або іншими шляхами.

Загалом захист від атак на перебір паролів вимагає комплексного підходу, який включає в себе технічні, процесні та організаційні заходи для забезпечення високого рівня безпеки облікових записів користувачів.

Щоб реалізувати обмеження кількості спроб аутентифікації, ми можемо додати додаткові поля до моделі користувача User, такі як loginAttempts та lockUntil. Це дозволить нам вести облік невдалих спроб входу і встановлювати тимчасові блокування облікового запису користувача.

Додавши ці поля, ми зможемо зберігати кількість невдалих спроб аутентифікації для кожного користувача, а також встановлювати час, до якого обліковий запис буде заблоковано після досягнення певного порогу невдалих спроб.

Наприклад, якщо користувач здійснив надто багато невдалих спроб входу, його обліковий запис може бути тимчасово заблокований для захисту від потенційних атак на перебір паролів.

Такий підхід дозволяє нам активно моніторити та контролювати спроби несанкціонованого доступу до облікових записів користувачів, забезпечуючи високий рівень безпеки системи.

```
verify: {
  type: Boolean,
  default: false,
},
verificationToken: {
  type: String,
  required: [true, "Verify token is required"],
},
loginAttempts: {
  type: Number,
  default: 0,
},
lockUntil: {
```

		Самчук Д.О.			КІК.КР.Б – ПЗ	Арк.
		Головня О.С.				53
Змн.	Арк.	№ докум.	Підпис	Дата		

```

    type: Date,
    default: null,
  },

```

Включимо нову функціональність, яка автоматично збільшує лічильник невдалих спроб та блокує обліковий запис, коли досягнуто максимальної кількості спроб.

```

const { User } = require("../models/user");

const increaseLoginAttempts = async (email) => {
  try {
    const user = await User.findOne({ email });
    if (!user) {
      throw new Error("User not found");
    }

    user.loginAttempts += 1;
    if (user.loginAttempts >= 5) {
      user.lockUntil = Date.now() + 24 * 60 * 60 * 1000;
    }

    await user.save();
  } catch (error) {
    throw new Error("Failed to increase login attempts");
  }
};

module.exports = {
  increaseLoginAttempts,
};

```

Таким чином, ми додали функцію `increaseLoginAttempts`, яка використовується для збільшення лічильника невдалих спроб входу користувача та блокування його облікового запису у разі досягнення максимальної кількості спроб.

У функції спершу шукається користувач за вказаною електронною поштою. Якщо користувача не знайдено, викликається помилка “User not found”. Після цього лічильник невдалих спроб користувача збільшується на одиницю. Якщо кількість невдалих спроб перевищує п'ять, то поле `lockUntil` облікового запису встановлюється на 24 години в майбутньому. Після цього зберігаються зміни у базі даних.

		Самчук Д.О.		
		Головня О.С.		
Змн.	Арк.	№ докум.	Підпис	Дата

```

router.post("/users/login", auth.enhance, async (req, res) => {
  try {
    const user = await User.findByCredentials(
      req.body.username,
      req.body.password
    );

    if (!user || !passwordCompare) {
      await increaseLoginAttempts(email);
      throw HttpError(401, "Email or password is wrong");
    }

    if (!user.verify) {
      throw HttpError(401, `Verification not confirmed`);
    }

    const payload = {
      id: user._id,
    };

    const token = jwt.sign(payload, SECRET_KEY, { expiresIn: "23h" });
    await User.findByIdAndUpdate(user._id, { token });

    res.json({
      status: "success",
      code: 200,
      data: {
        token,
        user: {
          email: user.email,
          subscription: user.subscription,
        },
      },
    });
  } catch (e) {
    res.status(400).send({
      error: { message: "You have entered an invalid username or password" },
    });
  }
});

```

Ця функція може бути використана у маршруті для обробки спроби входу користувача. У разі невдачі при вході, функція викликається для збільшення лічильника невдалих спроб. Якщо кількість невдалих спроб перевищує максимальне значення, обліковий запис користувача буде заблоковано на 24 години.

		Самчук Д.О.			КІК.КР.Б – ПЗ	Арк.
		Головня О.С.				55
Змн.	Арк.	№ докум.	Підпис	Дата		

Крім того, для забезпечення високої безпеки облікових записів користувачів необхідно вимагати використання сильних паролів. Для цього ми додамо правила, які будуть перевіряти складність нових паролів та вимагатимуть використання різних типів символів, а також встановимо мінімальну довжину пароля.

```
const registerSchema = Joi.object({
  name: Joi.string().required(),
  username: Joi.string().alphanum().min(3).max(30).required(),
  email: Joi.string().pattern(emailRegex).required(),
  password: Joi.string()
    .min(8)
    .pattern(/^(?=.*\d)(?=.*[a-z])(?=.*[A-Z])(?=.*[!@#%&*]).{8,}$/)
    .required()
    .messages({
      "string.pattern.base":
        "Password must contain at least one lowercase letter, one uppercase
        letter, one digit, and one special character (!@#%&*)",
    }),
  phone: Joi.string().required(),
  imageurl: Joi.string(),
});
```

Складний пароль, що використовується в даній схемі, забезпечує високий рівень безпеки облікових записів користувачів. Значення складових частин:

$^(?=.*\d)$ – пароль повинен містити принаймні одну цифру.

$(?=.*[a-z])$ – пароль повинен містити принаймні одну маленьку літеру.

$(?=.*[A-Z])$ – пароль повинен містити принаймні одну велику літеру.

$(?=.*[!@#\$%^&*])$ – пароль повинен містити принаймні один спеціальний символ.

$.{8,}$ – Це квантифікатор $\{8,\}$, який вказує, що пароль повинен бути принаймні 8 символів у довжину.

Отже, складний пароль, заданий у вказаній схемі, забезпечує високий рівень безпеки, оскільки вимагає від користувачів використання різноманітних типів символів (цифри, великі та малі літери, спеціальні символи) та мінімальну довжину пароля. Це робить пароль більш стійким до атак перебору та підвищує безпеку облікових записів користувачів.

		Самчук Д.О.			КІК.КР.Б – ПЗ	Арк.
		Головня О.С.				56
Змн.	Арк.	№ докум.	Підпис	Дата		

Захист від атак на введення даних

Сервер повинен надійно захищатися від будь-яких атак, спрямованих на введення недійсних або шкідливих даних. Це включає в себе не лише фільтрацію та очищення вхідних даних, а також використання параметризованих запитів до бази даних, валідацію формату даних та застосування механізмів обмеження об'єму даних.

Для реалізації ефективного захисту від атак на введення даних на сервері кінотеатру, нам слід виконати наступні кроки:

1. Фільтрація та очищення вхідних даних: Перед обробкою будь-яких вхідних даних, що надходять на сервер, ми повинні застосувати механізми фільтрації для видалення небажаних символів, HTML-тегів або потенційно шкідливого коду. Для цього ми можемо використовувати потужні бібліотеки, такі як DOMPurify, щоб очистити HTML-дані перед їх обробкою або збереженням на сервері.

```
const window = new JSDOM("").window;
const DOMPurify = createDOMPurify(window);

app.use((req, res, next) => {
  winstonLogger.info("Отримано запит", { url: req.url, method: req.method });
  if (req.body.htmlData) {
    req.body.htmlData = DOMPurify.sanitize(req.body.htmlData);
  }
  next();
});
```

В даному коді застосовується бібліотека DOMPurify для очищення HTML-даних, що надходять у запитах до сервера. Це забезпечує захист від потенційно шкідливих вмісту, які можуть містити небезпечний або виконуючийся код, такий як скрипти або iframe-елементи.

Створення і використання об'єкта JSDOM дозволяє нам отримати об'єкт window, який потрібний для роботи DOMPurify. Після цього DOMPurify ініціалізується з використанням цього об'єкта window.

		Самчук Д.О.			КІК.КР.Б – ПЗ	Арк.
		Головня О.С.				57
Змн.	Арк.	№ докум.	Підпис	Дата		

При обробці кожного запиту до сервера, ми перевіряємо, чи містить тіло запиту поле `htmlData`. Якщо це поле присутнє, ми застосовуємо функцію `DOMPurify.sanitize()` для очищення вмісту цього поля від потенційно шкідливого HTML-коду. Це допомагає уникнути XSS-атак та інших безпекових проблем, пов'язаних з обробкою HTML-даних на сервері. Такий підхід забезпечує безпеку від шкідливих атак та зберігає цілісність даних у веб-додатку.

Застосування механізмів обмеження обсягу даних

Застосування механізмів обмеження обсягу даних - це важлива стратегія для забезпечення безпеки і ефективності веб-додатків. Ця стратегія полягає у встановленні максимально допустимого обсягу даних, які можуть бути передані або оброблені у системі. Такий підхід дозволяє запобігти витоку пам'яті, перевантаженню сервера і можливим атакам на переповнення буфера.

Основні підходи до обмеження обсягу даних включають:

1. Встановлення лімітів на розмір файлів: Для обмеження обсягу даних, що передаються через веб-додаток, можна встановити максимально припустимий розмір файлів для завантаження або відправки. Наприклад, максимальний розмір файлу може бути обмежений на рівні сервера або в конфігурації мережевого протоколу.

2. Обмеження кількості запитів або даних на запит: Для зменшення навантаження на сервер можна обмежити кількість запитів, що обробляються за один раз або кількість даних, які можуть бути передані у межах одного запиту. Це допоможе попередити перевантаження сервера і збільшити його ефективність.

3. Валідація та фільтрація вхідних даних: Валідація та фільтрація вхідних даних перед їх обробкою дозволяє відсіяти непотрібні або потенційно шкідливі дані ще до того, як вони будуть збережені або передані у систему. Це допомагає зменшити обсяг даних, які потрібно обробляти, і запобігти можливим атакам.

4. Кешування даних: Використання кешування даних дозволяє зберігати попередньо оброблені або запитані дані у пам'яті сервера або іншому сховищі, що дозволяє швидше відповідати на запити і зменшує навантаження на сервер.

		Самчук Д.О.			КІК.КР.Б – ПЗ	Арк.
		Головня О.С.				
Змн.	Арк.	№ докум.	Підпис	Дата		58

Загалом, застосування механізмів обмеження обсягу даних є важливою складовою безпеки та ефективності веб-додатків, яка допомагає забезпечити стабільну та безпечну роботу системи.

Необхідно установити обмеження на обсяг даних, які можуть бути передані на сервер, для запобігання надмірного завантаження сервера або атак типу DDoS. Це можна здійснити через обмеження розміру завантажуваних файлів або кількості запитів, які можна зробити за певний період часу.

DDoS-атаки є серйозною загрозою для користувачів та власників веб-ресурсів, оскільки призводять до перерв у роботі сервера. Захист від цього типу атак є важливою складовою безпеки і негайною необхідністю, оскільки відмова у обслуговуванні може негативно позначитися на репутації та фінансовому стані організації.

Щоб збільшити ефективність і безпеку нашого сервера, пропонується кілька можливих удосконалень. По-перше, ми можемо обмежити типи файлів, які можуть бути завантажені на сервер, для запобігання завантаженню небезпечних файлів. Наприклад, через опцію `fileFilter` в `multer` можна обмежити завантаження лише зображень.

```
const multer = require("multer");
const path = require("path");
const { nanoid } = require("nanoid");

const tmpDir = path.join(__dirname, "../", "tmp");

const multerConfig = multer.diskStorage({
  destination: tmpDir,
  filename: (req, file, cb) => {
    const uniqueFilename = nanoid() + path.extname(file.originalname);
    cb(null, uniqueFilename);
  },
});

const upload = multer({
  storage: multerConfig,
  fileFilter: (req, file, cb) => {
    if (file.mimetype.startsWith("image/")) {
      cb(null, true);
    } else {
      cb(new Error("Неприпустимий тип файлу"));
    }
  },
});

module.exports = upload;
```

		Самчук Д.О.			КІК.КР.Б – ПЗ	Арк.
		Головня О.С.				59
Змн.	Арк.	№ докум.	Підпис	Дата		

Цей код визначає middleware для завантаження файлів у веб-додатках за допомогою бібліотеки `multer` в середовищі `Node.js`. Параметризовані файли зберігаються в тимчасовій директорії, використовуючи унікальні ідентифікатори, що генеруються функцією `nanoid`. Механізм також включає фільтрацію файлів, дозволяючи лише файлам типу `"image/"`, і викидає помилку для неприпустимих типів файлів.

Унікальне ім'я файлу генерується для кожного завантаженого файлу замість використання його оригінального імені. Це запобігає можливості конфліктування імен файлів і забезпечує унікальність кожного файлу в системі.

Ця middleware може бути використана в маршрутах `Express` для обробки завантаження файлів з клієнта, забезпечуючи безпеку та контроль над типами файлів, які можна завантажувати. Такий підхід дозволяє створити ефективну та безпечну систему завантаження файлів для веб-додатка.

Системний аудит та моніторинг

Системний аудит та моніторинг - це важливі аспекти в управлінні і підтримці будь-якого веб-додатка або системи. Ці процеси дозволяють вам відстежувати події, помилки та виконання вашого додатка для забезпечення його ефективності, безпеки та доступності.

Системний аудит включає запис різних подій, таких як запити користувачів, виникнення помилок, зміни стану системи тощо. Ці журнали можуть бути використані для аналізу діяльності системи, виявлення проблем та відладки. Моніторинг, з іншого боку, передбачає постійне спостереження за показниками роботи системи, такими як завантаження сервера, використання ресурсів, час відгуку тощо. Це допомагає забезпечити вчасне виявлення проблем та прийняття необхідних заходів для їх вирішення.

У нашому проекті ми використовуємо бібліотеку `Winston` для реалізації журналювання. `Winston` - це потужна бібліотека для журналювання в середовищі `Node.js`, яка надає різноманітні можливості для ефективного збору та аналізу журнальних повідомлень.

		Самчук Д.О.			КІК.КР.Б – ПЗ	Арк.
		Головня О.С.				60
Змн.	Арк.	№ докум.	Підпис	Дата		

Основні переваги використання Winston включають гнучкість налаштувань, можливість вибору різних транспортів для збереження журнальних записів (таких як файлові, консольні, HTTP тощо), різні рівні журналювання для керування деталізацією виводу та можливість обробки та фільтрації повідомлень.

```
const winston = require("winston");
const winstonLogger = winston.createLogger({
  transports: [
    new winston.transports.Console(),
    new winston.transports.File({ filename: "logs/error.log", level: "error" }),
    new winston.transports.File({ filename: "logs/info.log", level: "info" }),
  ],
});
app.use((req, res, next) => {
  winstonLogger.info("Отримано запит", { url: req.url, method: req.method });
  if (req.body.htmlData) {
    req.body.htmlData = DOMPurify.sanitize(req.body.htmlData);
  }
  next();
});
```

У даному фрагменті коду ми використовуємо бібліотеку Winston для реалізації журналювання в нашому додатку. Ми створюємо екземпляр логгера за допомогою методу `createLogger` з налаштуваннями, включаючи різні транспорти для збереження журнальних записів. У цьому випадку ми використовуємо консольний транспорт для виводу журнальних повідомлень у консоль, а також два файлових транспорти для збереження повідомлень у відповідні файли “error.log” та “info.log”.

Далі, ми використовуємо створений логгер для журналювання подій у нашому Express додатку. У міدلварі, що викликається перед обробкою запитів, ми реєструємо подію “Отримано запит” з відповідними даними про URL та метод запиту. Це дозволяє нам відстежувати та аналізувати запити до сервера для подальшого моніторингу та відладки додатку.

Таким чином, використання Winston дозволяє нам збирати та аналізувати журнальні записи з різних частин нашого додатку, що допомагає відстежувати помилки, виконання операцій та інші події для відладки, моніторингу та вдосконалення функціональності та продуктивності додатку.

		Самчук Д.О.			КІК.КР.Б – ПЗ	Арк.
		Головня О.С.				61
Змн.	Арк.	№ докум.	Підпис	Дата		

РОЗДІЛ 3 НАЛАГОДЖЕННЯ, ТЕСТУВАННЯ ТА ОЦІНКА ЕФЕКТИВНОСТІ ЗАХИСТУ ВЕБ-СЕРВЕРА

3.1. Огляд процесу налагодження та тестування

Процес налагодження та тестування є невід'ємною частиною розробки програмного забезпечення, спрямованою на забезпечення його якості та надійності. Від цих процесів залежить успішність впровадження програмних продуктів та задоволення потреб користувачів.

Основними цілями тестування веб-сервера кінотеатру є забезпечення його надійності, безпеки, продуктивності та відповідності функціональним та нефункціональним вимогам. Розглянемо більш детально цілі тестування.

1. Надійність: тестування спрямоване на переконання, що веб-сервер кінотеатру може працювати стабільно та безперебійно у різних умовах, включаючи високі навантаження та ситуації відмов. Це включає тестування на витривалість, стійкість до помилок та відновлення після збоїв.

2. Безпека: тестування має за мету виявлення та усунення потенційних загроз безпеці, таких як атаки на застосування SQL, перехоплення сесій, зловживання правами доступу та інші види кіберзлочинності.

3. Продуктивність: тестування спрямоване на визначення продуктивності веб-сервера та його здатності відповідати на запити користувачів у разі великого навантаження. Це включає вимірювання часу відповіді сервера на запити, оптимізацію ресурсів сервера та забезпечення ефективного використання мережевого трафіку.

4. Відповідність функціональним вимогам: тестування повинно переконатися, що веб-сервер кінотеатру відповідає всім функціональним вимогам, таким як можливість реєстрації користувачів, покупки квитків, перегляду розкладу сеансів, а також адміністрування контенту. Також важливо перевірити відповідність нормативним вимогам, які можуть стосуватися безпеки, доступності та конфіденційності даних.

		Самчук Д.О.			КІК.КР.Б – ПЗ	Арк.
		Головня О.С.				
Змн.	Арк.	№ докум.	Підпис	Дата		62

Процес тестування є важливою складовою забезпечення якості програмного забезпечення, оскільки він допомагає виявляти помилки та дефекти, що можуть виникнути під час розробки програми, та забезпечує їх виправлення перед випуском продукту на ринок.

3.2.Тестування функціональності та захисту веб-сервера

Тестування функціональності та захисту веб-сервера є критично важливою частиною процесу розробки програмного забезпечення, оскільки воно спрямоване на перевірку роботи веб-сервера з точки зору функціональних можливостей і забезпечення його захищеності від потенційних загроз та атак. В даному пункті ми розглянемо процес тестування функціональності, а також заходи захисту, які використовуються для забезпечення безпеки веб-сервера. Тестування функціональності включає в себе перевірку роботи різних функцій та можливостей веб-сервера згідно з вимогами та очікуваними результатами. У той же час, захист веб-сервера передбачає впровадження заходів безпеки для запобігання несанкціонованому доступу, а також виявлення та усунення потенційних загроз та вразливостей. Дослідження та розгляд цих аспектів допоможе забезпечити високу якість та надійність веб-сервера в роботі.

Щоб здійснити функціональне тестування за допомогою Postman, виконаємо наступні кроки:

1. Завантажимо та встановимо Postman з офіційного веб-сайту, а потім запустимо програму після встановлення.
2. Додамо проєкт на GitHub.
3. Розгорнемо сервер на Render.com.

Для цього спочатку потрібно налаштувати змінні середовища (див. рис. 3.1).

		Самчук Д.О.			KIK.KP.B – ПЗ	Арк.
		Головня О.С.				
Змн.	Арк.	№ докум.	Підпис	Дата		63

Environment Variables

Set environment-specific config and secrets (such as API keys), then read those values from your code. [Learn more.](#)

Key	Value	
GMAIL_PASSWORD		
GMAIL_USER		
MAIL_PASSWORD		
MONGODB_URI		
PORT		

[Create Environment Group](#)

[+ Add Environment Variable](#)[Add from .env](#)[Save Changes](#)

Рисунок 3.1 – Додаємо змінні оточення на Render.com.

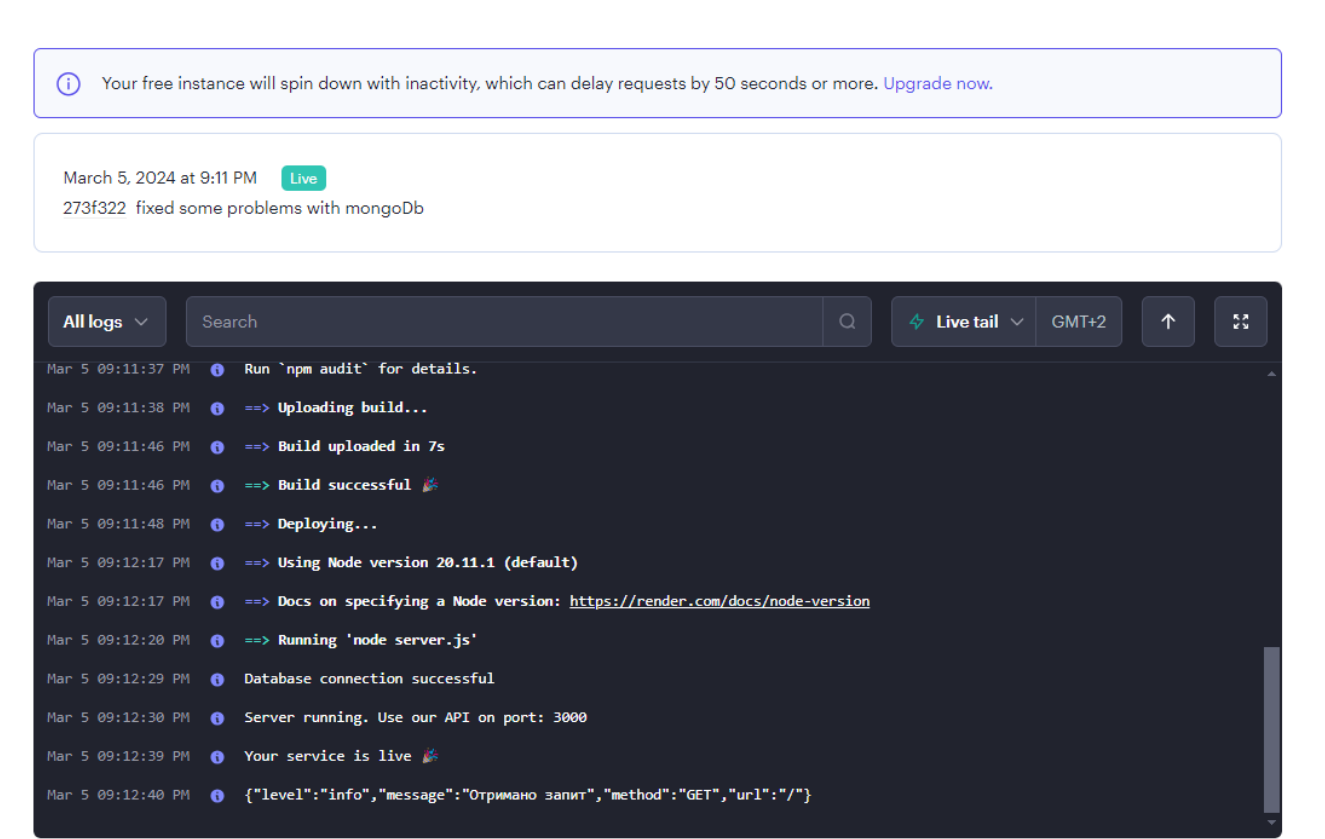


Рисунок 3.2 – Деплой проекту на <https://cyber-security-1.onrender.com>

3. Виконання запиту для перевірки доступу до серверу кінотеатру без авторизації.

- Створюємо новий запит у Postman.

- Встановлюємо метод запиту на GET.
- Встановлюємо URL-адресу запиту на адресу серверу кінотеатру.
- Натискаємо кнопку “Send” для відправлення запиту.
- Перевіряємо отриману відповідь. Очікується помилка 401 “Unauthorized”, якщо доступ без авторизації заборонено.

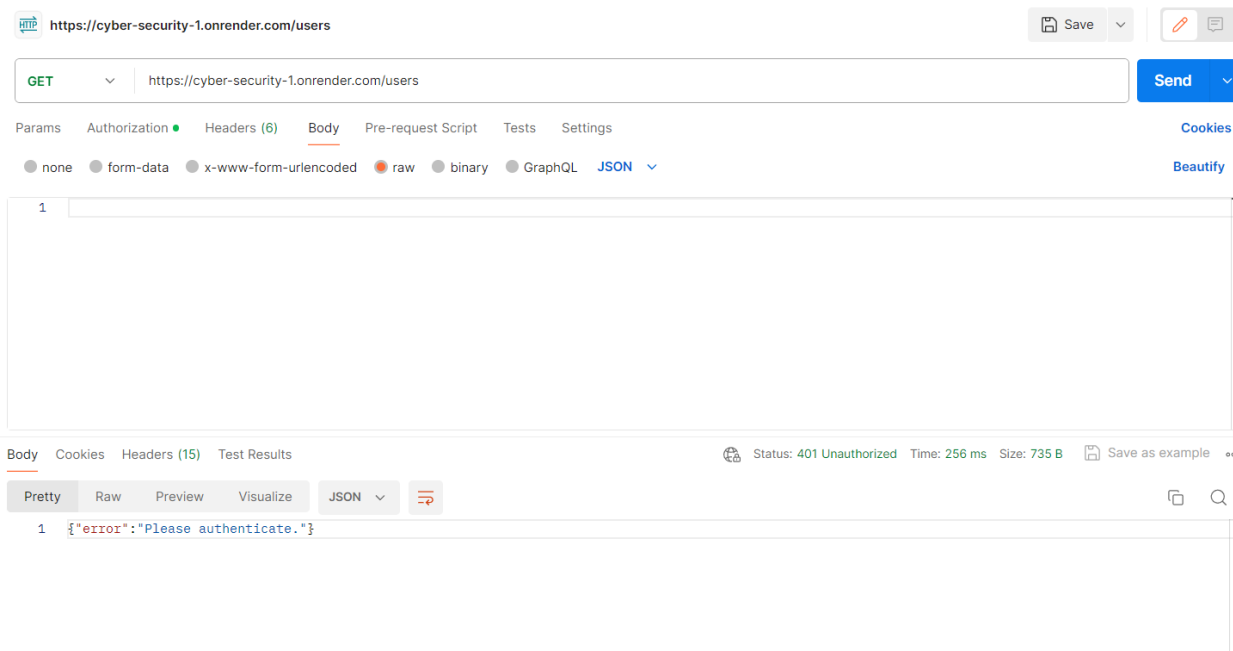


Рисунок 3.3 – Підтвердження неможливості отримання доступу до серверу кінотеатру без авторизації користувача

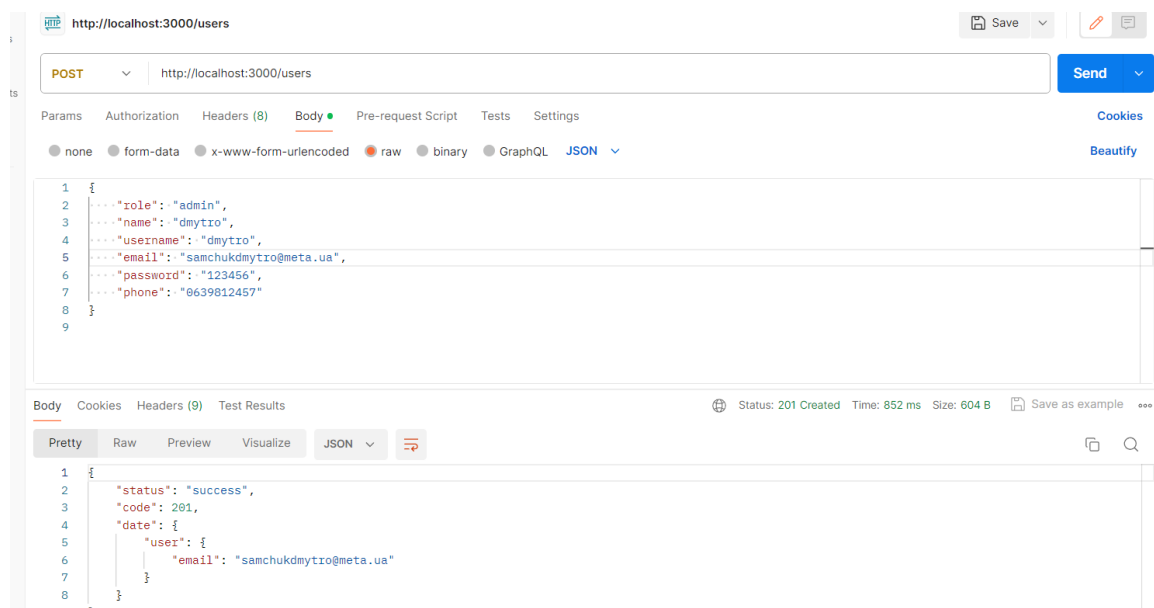


Рисунок 3.4 – Успішна реєстрація

		Самчук Д.О.			КІК.КР.Б – ПЗ	Арк.
		Головня О.С.				65
Змн.	Арк.	№ докум.	Підпис	Дата		

Крім того, ми реалізували функціонал двофакторної аутентифікації за допомогою перевірки пошти. Коли користувач реєструється або намагається увійти в систему, йому надсилається лист на пошту з унікальним посиланням або кодом підтвердження.

Підтвердження реєстрації

Дякуємо за реєстрацію на нашому сайті. Будь ласка, натисніть на кнопку нижче, щоб підтвердити свою адресу електронної пошти:

Підтвердити реєстрацію

Якщо ви не реєструвалися на нашому сайті, проігноруйте цей лист.

Рисунок 3.5 – Лист-підтвердження реєстрації

Після цього користувач переходить за посиланням або вводить код підтвердження в веб-інтерфейс.



Риунок 3.6 – Підтвердження успішної верифікації

Після успішної перевірки пошти та підтвердження ідентифікації в базі даних оновлюється поле `verify`, яке стає значенням `true`. Цей механізм дозволяє підвищити безпеку облікових записів користувачів, забезпечуючи додатковий шар захисту перед доступом до функцій системи.

		Самчук Д.О.			КІК.КР.Б – ПЗ	Арк.
		Головня О.С.				
Змн.	Арк.	№ докум.	Підпис	Дата		66

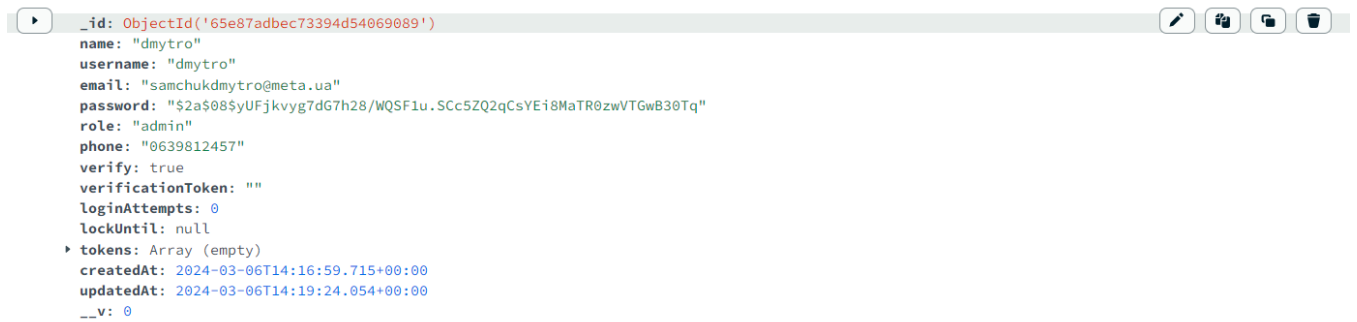


Рисунок 3.7 – Оновлення поля verificationToken в базі даних MongoDB

Ми протестували логування за допомогою бібліотеки Winston. Для цього ми створили і налаштували логгер з різними транспортами, які направляють логи до консолі та файлів. Наприклад, ми налаштували два файли для запису логів - “error.log” для рівня помилок і “info.log” для інформаційних повідомлень.

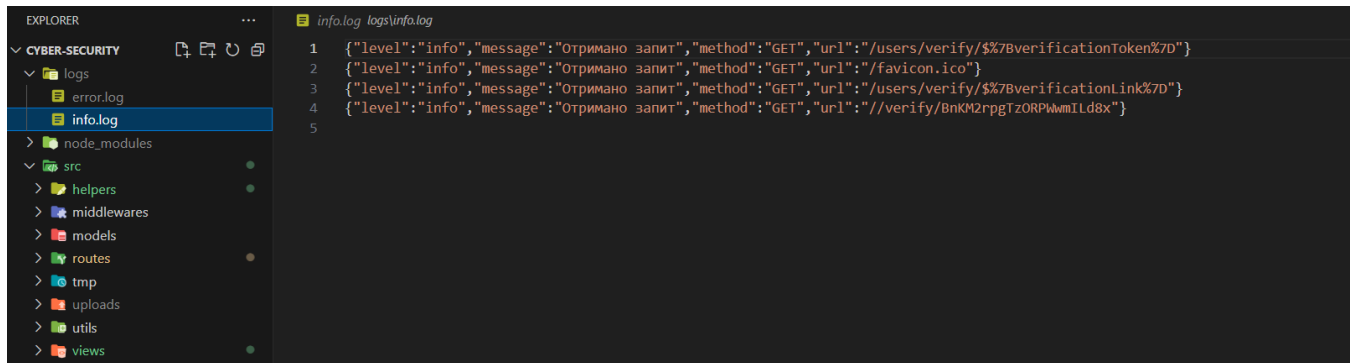


Рисунок 3.8 – Логування запитів

Ці записи логів допомагають нам відстежувати та аналізувати вхідні запити до веб-сервера, що дозволяє нам виявляти можливі проблеми та виконувати відповідні заходи щодо їх вирішення.

Post запит в Postman для додання кінотеатру до переліку.

		Самчук Д.О.			КІК.КР.Б – ПЗ	Арк.
		Головня О.С.				67
Змн.	Арк.	№ докум.	Підпис	Дата		

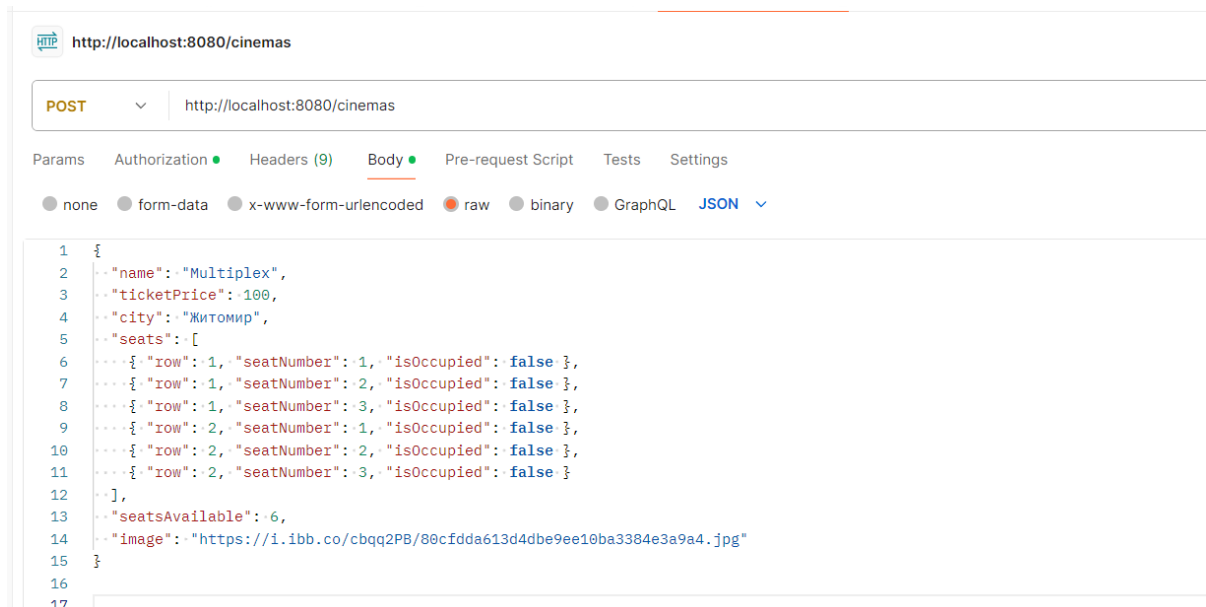


Рисунок 3.9 – Post запит для додавання кінотеатру до переліку

Кінотеатри відображаються на сторінці Our Cinema.

Route movies надає можливості роботи з фільмами у системі. Основні функції включають:

1. Створення фільму: користувач може додати новий фільм до системи, вказавши всі необхідні характеристики, такі як назва, мова, жанр, режисер, акторський склад, тривалість, дата виходу та інші.

2. Завантаження фото фільму: існує можливість додати фотографію фільму для подальшого відображення на сторінках користувача. Фотографії додають інформативність та привабливість до фільмів.

3. Отримання списку всіх фільмів: користувач може переглянути список всіх доступних фільмів разом з їхніми основними характеристиками.

4. Отримання інформації про конкретний фільм за його ідентифікатором: користувач може отримати детальну інформацію про певний фільм, включаючи його характеристики та статус.

5. Оновлення характеристик фільму: існують можливості редагування та оновлення інформації про фільм, такі як зміна назви, мови, жанру, режисера, акторського складу та інших характеристик.

6. Видалення фільму: користувач має можливість видалити фільм з системи у разі необхідності.

		Самчук Д.О.			КІК.КР.Б – ПЗ	Арк.
		Головня О.С.				68
Змн.	Арк.	№ докум.	Підпис	Дата		

Для прикладу представлено post запит для додання кінофільму.

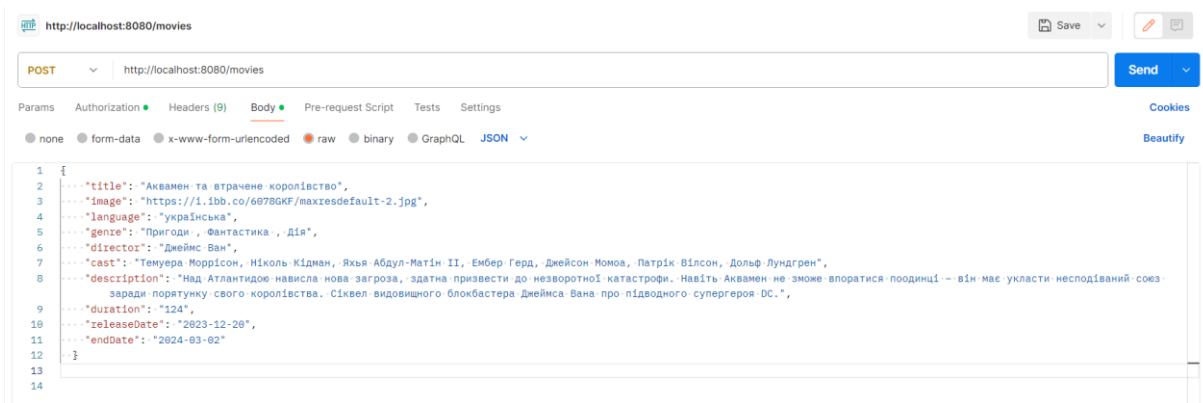


Рисунок 3.10 – Post запит для додавання фільму

При цьому отримана відповідь зі статусом 201 та присвоєним id фільму.

```
{
  "_id": "65c5f42f69033848c080747d",
  "title": "аквамен та втрачене королівство",
  "image": "https://i.ibb.co/6078GKF/maxresdefault-2.jpg",
  "language": "українська",
  "genre": "пригоди , фантастика , дія",
  "director": "джеймс ван",
  "cast": "темуера моррісон, ніколь кідман, яхья абдул-матін ii, ембер герд, джейсон момоа, патрік вілсон, дольф лундгрєн",
  "description": "над атлантидою нависла нова загроза, здатна призвести до незворотної катастрофи. навіть аквамен не зможе впоратися поодиночці – він має укласти несподіваний союз заради порятунку свого королівства. сіквел видовищного блокбастера джеймса вана про підводного супергероя dc.",
  "duration": 124,
  "releaseDate": "2023-12-20T00:00:00.000Z",
  "endDate": "2024-03-02T00:00:00.000Z",
  "__v": 0
}
```

Залежно від дати початку трансляції фільму, відображаються на сайті або на сторінці “Now Showing”, або на “Coming Soon” (якщо дата початку трансляції ще не настала).

Для додавання сеансів до фільмів вам потрібно виконати POST-запит за такою адресою: <http://localhost:8080/showtimes>. В тілі запиту передайте дані про сеанс, які включають час початку (startAt), дату початку (startDate), дату закінчення (endDate), ідентифікатор фільму (movieId) та ідентифікатор кінотеатру (cinemaId). Нижче наведений приклад:

		Самчук Д.О.			КІК.КР.Б – ПЗ	Арк.
		Головня О.С.				69
Змн.	Арк.	№ докум.	Підпис	Дата		

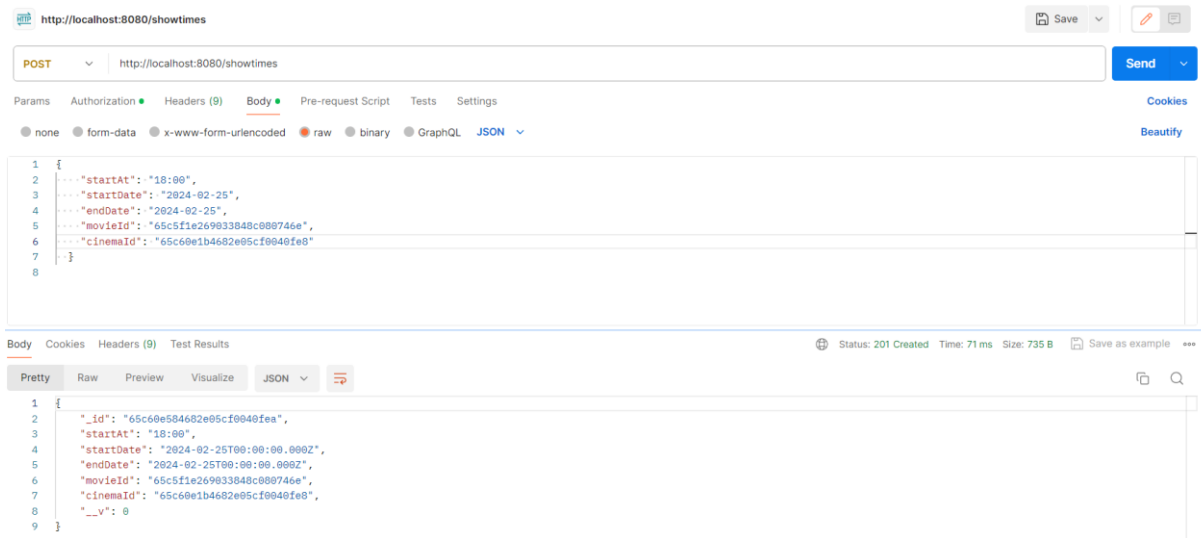


Рисунок 3.11 – Post запит для додавання сеансів

Процес резервування місць на сеанс включає такі кроки.

1. Виконуємо POST-запит за адресою `http://localhost:8080/reservations`.
2. У тілі запиту передаємо дані для резерву, такі як дата (date), час початку (startAt), інформація про місця (seats), вартість квитка (ticketPrice), загальна сума (total), ідентифікатор фільму (movieId), ідентифікатор кінотеатру (cinemaId), ім'я користувача (username) і телефон (phone). Деталі місць передаються у форматі масиву, де кожен елемент масиву - це масив з трьох елементів: номер ряду, номер місця і статус (чи вільне місце).

```
{
  "date": "2024-02-25",
  "startAt": "18:00",
  "seats": [
    [10, 5, true],
    [10, 6, true],
    [10, 7, true]
  ],
  "ticketPrice": 400,
  "total": 1200,
  "movieId": "65c5f1e269033848c080746e",
  "cinemaId": "65c60e1b4682e05cf0040fe8",
  "username": "daria",
  "phone": "+3806345678978"
}
```

3. Після успішного виконання запиту місця будуть зарезервовані, отримуємо підтвердження резерву та QR код.

		Самчук Д.О.			КІК.КР.Б – ПЗ	Арк.
		Головня О.С.				70
Змн.	Арк.	№ докум.	Підпис	Дата		

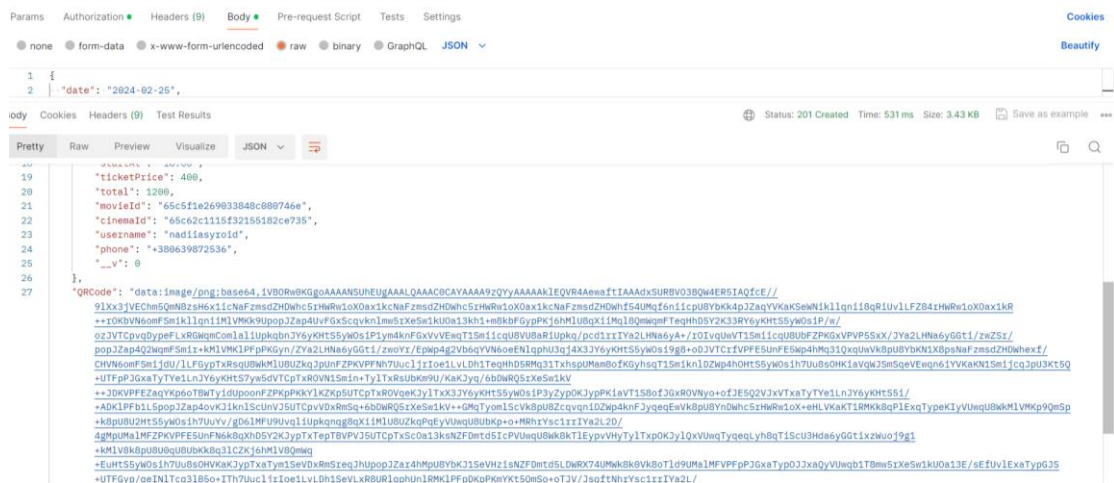


Рисунок 3.12 – Відповідь із успішним виконанням резервування

Таким чином, у ході роботи з програмним продуктом було описано основні функціональні можливості, такі як додавання кінотеатрів, фільмів, сеансів та резервування квитків. Для кожного з цих етапів були розроблені відповідні ендпоінти API, які дозволяють взаємодіяти з програмним продуктом через HTTP-запити.

Також було надано приклади використання цих ендпоінтів шляхом надсилання POST-запитів для створення об'єктів кінотеатрів, фільмів, сеансів та резервів.

Ці кроки допоможуть переконатися у коректному функціонуванні програмного продукту та його здатності задовольняти вимоги користувачів у плані роботи з кінотеатрами, фільмами та бронюванням квитків.

3.3 Тестування веб-сервера для виявлення SQL ін'єкцій

Веб-серверне тестування визначає відповідність функціональності сервера вимогам і її реалізацію шляхом спостереження за роботою програми в різних умовах і на обмеженому наборі тестів. Цей процес контролює якість сервера та виявляє розбіжності між очікуваною та фактичною функціональністю [10]. Функціональне тестування є одним з видів, що перевіряє відповідність функцій програмного забезпечення встановленим вимогам [11].

При виявленні SQL-ін'єкцій важливо перевірити наступне:

- Збереження інформації про SQL-ін'єкції в лог-файл.
- Запис інформації про SQL-ін'єкції у базу даних.
- Надсилання інформації про SQL-ін'єкції адміністратору.

Тести будуть успішними, якщо:

- Результати відповідають очікуваному.
- Помилки вводу даних виявляються та повідомляються.
- Дані у файлі та повідомленнях коректні.

Додаткові дані про тести використання можна знайти у таблицях 3.1-3.3 [10, 13].

Таблиця 3.1 – Перевірка запису інформації про SQL-ін'єкцію у лог-файл

Мета тесту	Перевірка можливості запису інформації про SQL-ін'єкції у лог-файл
Початковий стан	Відкрите програмне забезпечення виявлення SQL-ін'єкцій
Вхідні дані	Дані про злоумисника, що намагається впровадити SQL-код: "Код події", "Текст події", "Користувач", "Дата", "Час", "Шлях запити"
Схема проведення тесту	Перехід на веб-застосунок та виконання HTTP-запиту з впровадженням SQL-коду
Очікуваний результат	Дані записані до лог-файлу
Стан програмного продукту після проведення випробувань	Дані записані до лог-файлу

Таблиця 3.2 – Перевірка запису інформації про SQL-ін'єкцію до бази даних

Мета тесту	Перевірка запису інформації про SQL-ін'єкцію до бази даних
Початковий стан	Відкритий програмне забезпечення виявлення SQL-ін'єкцій
Вхідні дані	Дані про злоумисника, який намагається виконати впровадження SQL-коду: «Код події», «Текст події», «Користувач», «Дата», «Час», «Шлях запити»
Схема проведення тесту	Перейти на веб-застосунок та виконати HTTP-запит з впровадженням SQL-коду
Очікуваний результат	Дані записані до бази даних
Стан програмного продукту після проведення випробувань	Дані записані до бази даних

Після проведення перевірки контрольного прикладу “Перевірка запису інформації про SQL-ін'єкцію до бази даних” у програмному засобі виявлення SQL-ін'єкцій було отримано наступний результат: інформація про SQL-ін'єкції успішно занесена до бази даних.

		Самчук Д.О.			КІК.КР.Б – ПЗ	Арк.
		Головня О.С.				72
Змн.	Арк.	№ докум.	Підпис	Дата		

Таким чином, під час тестування було оцінено ефективність заходів безпеки, вжитих для запобігання SQL ін'єкціям. Це може включати використання параметризованих запитів, валідацію введених даних та обмеження прав доступу до бази даних.

На основі результатів тестування можна розробити рекомендації щодо вдосконалення безпеки веб-сервера. Це може включати в себе впровадження додаткових заходів безпеки, проведення навчання персоналу з питань безпеки або вдосконалення політик безпеки даних.

Тестування веб-сервера для виявлення SQL ін'єкцій є важливим етапом у забезпеченні безпеки веб-додатків. Відповідно до результатів тестування, необхідно прийняти відповідні заходи для запобігання вразливостям і підвищення рівня безпеки веб-сервера.

		Самчук Д.О.			КІК.КР.Б – ПЗ	Арк.
		Головня О.С.				73
Змн.	Арк.	№ докум.	Підпис	Дата		

ВИСНОВКИ

Захист веб-серверів є критично важливим аспектом для забезпечення безпеки і конфіденційності інформації, яка обробляється та зберігається на цих серверах. В сучасному цифровому світі веб-сервери є ключовою складовою інфраструктури для багатьох організацій та підприємств, в тому числі і в кінотеатральній галузі.

Розробка веб-сервера кінотеатру підвищує важливість його захисту, оскільки такий сервер містить конфіденційну інформацію про користувачів, таку як особисті дані, платіжні дані, інформація про персональні вподобання та інше. В разі порушення безпеки цієї інформації можуть виникнути серйозні наслідки, такі як виток особистих даних, фінансові втрати, порушення конфіденційності та пошкодження репутації.

Розроблений веб-сервер має структуру доступу з різними рівнями привілеїв для користувачів, гарантуючи захист даних від неправомірної зміни. Механізм аутентифікації та авторизації, разом із встановленням прав доступу на рівні маршрутів, дозволяє забезпечити контроль над даними та запобігти їх неправомірній модифікації. Модульний підхід бекенду розділяє логіку додатку на окремі компоненти, що сприяє підтримці та розширенню системи, забезпечуючи високу продуктивність та надійність. Використання фреймворку Express.js, розділення маршрутів на окремі файли та зберігання даних в MongoDB з Mongoose допомагає забезпечити стабільність та безпеку роботи системи, зберігаючи конфіденційні дані в безпечному форматі та забезпечуючи швидку та масштабовану роботу.

Для забезпечення високого рівня безпеки веб-сервера кінотеатру ми розробили та впровадили комплекс заходів захисту, які мають на меті уникнути потенційних загроз та захистити конфіденційні дані користувачів:

1. Проведення ін'єкції коду або XSS атак становить серйозну загрозу для безпеки нашого веб-сервера. Для забезпечення високого рівня безпеки ми рекомендуємо розробити та впровадити систему валідації введених даних користувачів. Встановлення правил валідації для різних типів запитів та застосування механізмів фільтрації та очищення вхідних даних перед обробкою може запобігти успішній атаці.

		Самчук Д.О.			КІК.КР.Б – ПЗ	Арк.
		Головня О.С.				74
Змн.	Арк.	№ докум.	Підпис	Дата		

2. Для захисту від атак на перебір паролів ми рекомендуємо обмежити кількість спроб введення паролів та тимчасово заблокувати обліковий запис після досягнення певної межі. Це дозволить уникнути несанкціонованого доступу до системи та зменшити ризик компрометації облікових записів користувачів.

3. Щоб уникнути використання слабких паролів, важливо вимагати від користувачів створення сильних паролів, які містять комбінацію букв, цифр та спеціальних символів. Відхилення використання слабких паролів може підвищити безпеку облікових записів користувачів.

4. Щоб захиститися від ін'єкції SQL або XSS атак, слід розробити та впровадити заходи захисту від атак на введення даних. Це може включати фільтрацію та очищення вхідних даних перед обробкою та використання параметризованих запитів до бази даних.

5. Для запобігання переповненню буфера або іншим атакам на обсяг даних, рекомендуємо використовувати валідацію формату даних та застосовувати механізми обмеження обсягу даних. Це дозволить уникнути можливих атак на обсяг даних та зберегти безпеку системи.

6. Для запобігання несанкціонованого доступу та зловживання привілеями, слід реалізувати обробку помилок після збереження запису користувача, включаючи перевірку результатів операцій збереження та прийняття відповідних заходів, таких як відмова у доступі або повідомлення про помилку.

7. Для виявлення невідомих або непередбачених помилок та подій рекомендується реалізувати журналювання подій і помилок. Збереження записів про події та помилки, що виникають під час роботи системи, дозволить вчасно виявляти потенційні проблеми та забезпечити аналіз безпеки.

8. Для виявлення аномальної активності та потенційних загроз рекомендується використовувати систему моніторингу. Вона дозволить виявляти незвичайну активність та надсилати сповіщення для подальшого аналізу та реагування на можливі проблеми безпеки.

Ці пропозиції допомагають підвищити рівень безпеки веб-сервера, зменшуючи ризики вразливостей та забезпечуючи захист інформації користувачів.

		Самчук Д.О.			КІК.КР.Б – ПЗ	Арк.
		Головня О.С.				75
Змн.	Арк.	№ докум.	Підпис	Дата		

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. CERT Coordination Center. Software Engineering Institute, Carnegie Mellon University. URL: <https://www.cert.org/>
2. Cybersecurity threatscape: Q3 2019. URL: <https://www.ptsecurity.com/ww-en/analytics/cybersecurity-threatscape-2019-q3/>
3. Krebs, B. Krebs on Security. URL: <https://krebsonsecurity.com/>
4. Mell, P. Information Security Handbook: A Guide for Managers. National Institute of Standards and Technology (NIST). URL: <https://www.nist.gov/>
5. OWASP Top 10 Security Risks & Vulnerabilities URL: <https://sucuri.net/guides/owasp-top-10-security-vulnerabilities-2020/>
6. Rich Cannings, Himanshu Dwivedi, Zane Lackey., Hacking Exposed Web 2.0: Web 2.0 Security Secrets and Solutions/ Rich Cannings, Himanshu Dwivedi, Zane Lackey // McGraw-Hill Education; December 2007
7. Ross, R. S. Computer Security Resource Center. National Institute of Standards and Technology (NIST). URL: <https://csrc.nist.gov/>
8. Schneier, B. Schneier on Security. URL: <https://www.schneier.com/blog/>
9. Securing web applications: top OWASP threats and what to do about them URL: <https://www.ptsecurity.com/ww-en/analytics/knowledge-base/securing-webapplications-top-owasp-threats-and-what-to-do-about-them/>
10. Ukrainian Cyber Alliance. URL: <https://uca.org.ua/>
11. Voitovych, O. P., O. S. Yuvkovetskyi, and L. M. Kupershtein. "SQL injection prevention system." 2016 International Conference Radio Electronics & Info Communications (UkrMiCo). IEEE, 2016 URL: <https://ir.lib.vntu.edu.ua/bitstream/handle/123456789/27968/45nigxju2tpe63tv5iy6aa8jor5pguaiic.pdf?sequence=1&isAllowed=y>
12. Web Applications vulnerabilities and threats: statistics for 2019 URL: <https://www.ptsecurity.com/ww-en/analytics/web-vulnerabilities-2020/>
13. What is a web server? URL: <https://informationq.com/what-is-a-web-server/>
14. Березін О.Є., Гордієнко Л.Є., Іванченко Є.О. Комп'ютерні мережі: захист інформації: навчальний посібник. Київ: Видавництво "Кондор", 2017.

		Самчук Д.О.			КІК.КР.Б – ПЗ	Арк.
		Головня О.С.				76
Змн.	Арк.	№ докум.	Підпис	Дата		

15. Болехівський Н., Полотай О. Класифікація мережевих атак та методи протидії і захисту. URL: <https://sci.ldubgd.edu.ua/bitstream/123456789/6737/1/1.pdf>
16. Борисенко В.О., Бондаренко О.І., Железняк І.О. Захист інформації на комп'ютерах і мережах: підручник. Київ: Видавництво “Наукова думка”, 2019.
17. Грінченко С.В., Мартинюк Ю. В., Щербак С. О. Методи та засоби захисту інформації в комп'ютерних системах: навчальний посібник. Київ: Видавничий центр КНЕУ, 2018.
18. Кібервійна – це сучасна реальність: Найгучніші кібератаки в Україні та світі і способи захисту від них. URL: <https://psm7.com/uk/security/kibervijna-ce-suchasna-realnist-najguchnishi-kiberataki-v-ukra%D1%97ni-ta-sviti-i-sposobi-zaxistu-vid-nix.html>
19. Колощук М.С., Вакалюк Т.А. Поширені типи мережевих атак. URL: <https://conf.ztu.edu.ua/wp-content/uploads/2022/04/58.pdf>
20. Мачида Л., Еванс Р. “Проблеми захисту сайтів” DiaSoft, Київ. 2008. 101 с.
21. Молчанов В.П., Пандорін О.К. Технології розробки WEB-ресурсів: навчальний посібник. Харків : ХНЕУ ім. С. Кузнеця, 2019. 130 с.
22. Томсон Лаура и Люк Веллинг “Хитрощі PHP” DiaSoft Киев 2006. 78 с.
23. Томсон Лаура, Люк Веллинг “Розробка Web додатків на PHP та MySQL” DiaSoft Киев 2006. с. 34-52.
24. ТОП 10 загроз кібербезпеці бізнесу у 2023 році. URL: <https://www.bdo.ua/uk-ua/insights-2/information-materials/2023/top-10-cybersecurity-threats-to-businesses-in-2023>
25. Шилова, Ю. А. Захист інформації в комп'ютерних системах: підручник. Київ: Видавничо-поліграфічний центр “Київський університет”, 2016.

		Самчук Д.О.			КІК.КР.Б – ПЗ	Арк.
		Головня О.С.				77
Змн.	Арк.	№ докум.	Підпис	Дата		

ДОДАТКИ

		Самчук Д.О.			КІК.КР.Б – ПЗ	Арк.
		Головня О.С				78
Змн.	Арк.	№ докум.	Підпис	Дата		

```

{"level":"info","message":"Отримано
запит","method":"GET","url":"/users/verify/%7BverificationToken%7D"}
{"level":"info","message":"Отримано запит","method":"GET","url":"/favicon.ico"}
{"level":"info","message":"Отримано
запит","method":"GET","url":"/users/verify/%7BverificationLink%7D"}
{"level":"info","message":"Отримано
запит","method":"GET","url":"/verify/BnKM2rpgTzORPWwmILd8x"}
{"level":"info","message":"Отримано запит","method":"GET","url":"/favicon.ico"}

```

```

const fs = require("fs");
const path = require("path");
const handlebars = require("handlebars");

```

```

const { BASE_URL } = process.env;

```

```

const createVerifyEmail = (verificationToken, email) => {
  const templatePath = path.join(
    __dirname,
    "..",
    "views",
    "verifyEmailTemplate.hbs"
  );
  const emailTemplate = fs.readFileSync(templatePath, "utf-8");
  const compiledTemplate = handlebars.compile(emailTemplate);
  const verificationLink = `${BASE_URL}/verify/${verificationToken}`;
  const emailContent = compiledTemplate({ verificationLink });

```

```

  return {
    to: email,
    subject: "Verify email",
    html: emailContent,
  };
};

```

```

module.exports = createVerifyEmail;

```

```

const ctrlWrapper = (ctrl) => {
  const func = async (req, res, next) => {
    try {
      await ctrl(req, res, next);
    } catch (error) {
      next(error);
    }
  };

```

```

  return func;
};

```

```

module.exports = ctrlWrapper;

```

		Самчук Д.О.			КІК.КР.Б – ПЗ	Арк.
		Головня О.С.				
Змн.	Арк.	№ докум.	Підпис	Дата		79

```

const handleSaveErrors = (error, data, next) => {
  const { name, code } = error;
  error.status = name === "MongoServerError" && code === 11000 ? 409 : 400;
  next();
};

module.exports = handleSaveErrors;

const errorMessageList = {
  400: "Bad Request",
  401: "Unauthorized",
  403: "Forbidden",
  404: "Not found",
  409: "Conflict",
};

const HttpError = (status, message = errorMessageList[status]) => {
  const error = new Error(message);
  error.status = status;
  return error;
};

module.exports = HttpError;

const createVerifyEmail = require("./createVerifyEmail");
const ctrlWrapper = require("./ctrlWrapper");
const handleSaveErrors = require("./handelSaveErrors");
const HttpError = require("./HttpError");
const sendEmail = require("./sendEmail");

module.exports = {
  handleSaveErrors,
  ctrlWrapper,
  HttpError,
  sendEmail,
  createVerifyEmail,
};

const nodemailer = require("nodemailer");

const { MAIL_PASSWORD } = process.env;

const nodemailerConfig = {
  host: "smtp.meta.ua",
  port: 465,
  secure: true,
  auth: {
    user: "samchukdmytro@meta.ua",
    pass: MAIL_PASSWORD,
  },
};

```

		Самчук Д.О.			КІК.КР.Б – ПЗ	Арк.
		Головня О.С.				80
Змн.	Арк.	№ докум.	Підпис	Дата		

```

const transport = nodemailer.createTransport(nodemailerConfig);

const sendEmail = async (data) => {
  const email = { ...data, from: "samchukdmytro@meta.ua" };
  await transport.sendMail(email);
  return true;
};

module.exports = sendEmail;

const jwt = require('jsonwebtoken');
const User = require('../models/user');

const simple = async (req, res, next) => {
  try {
    const { email } = req.body;
    const user = await User.findOne({ email });

    const { SECRET_KEY } = process.env;
    console.log(email);

    const payload = {
      id: user._id,
    };

    const token = jwt.sign(payload, SECRET_KEY, { expiresIn: '23h' });
    if (!user) throw new Error();
    req.token = token;
    req.user = user;
    next();
  } catch (e) {
    res.status(401).send({ error: 'Please authenticate.' });
  }
};

const enhance = async (req, res, next) => {
  try {
    const { email } = req.body;
    const user = await User.findOne({ email });
    console.log(email);

    const { SECRET_KEY } = process.env;

    const payload = {
      id: user._id,
    };

    const token = jwt.sign(payload, SECRET_KEY, { expiresIn: '23h' });

    if (!user) throw new Error();

    req.token = token;

```

		Самчук Д.О.			КІК.КР.Б – ПЗ	Арк.
		Головня О.С.				81
Змн.	Арк.	№ докум.	Підпис	Дата		

```

    req.user = user;
    next();
  } catch (e) {
    res.status(401).send({ error: 'Please authenticate.' });
  }
};

```

```

module.exports = { simple, enhance };

```

```

const jwt = require("jsonwebtoken");

```

```

const { User } = require("../models/user");

```

```

const { HttpError } = require("../helpers");

```

```

const { SECRET_KEY } = process.env;

```

```

const authenticate = async (req, res, next) => {
  const { authorization = "" } = req.headers;
  const [bearer, token] = authorization.split(" ");

```

```

  if (bearer !== "Bearer") {
    next(HttpError(401, "Not authorized"));
  }

```

```

  if (!token) {
    next(HttpError(401, "Not authorized"));
  }

```

```

  try {
    const { id } = jwt.verify(token, SECRET_KEY);

```

```

    const user = await User.findById(id);

```

```

    if (!user || !user.token || user.token !== token) {
      next(HttpError(401, "Not authorized"));
    }

```

```

    req.user = user;
    next();
  } catch {
    next(HttpError(401));
  }
};

```

```

module.exports = authenticate;

```

```

const validation = require("../validation");
const authenticate = require("../authenticate");

```

```

module.exports = { validation, authenticate };

```

```

const { HttpError } = require("../helpers");

```

		Самчук Д.О.			КІК.КР.Б – ПЗ	Арк.
		Головня О.С.				82
Змн.	Арк.	№ докум.	Підпис	Дата		

```

const validation = (schema) => {
  const func = (req, res, next) => {
    const { error } = schema.validate(req.body);
    if (error) {
      next(HttpError(400, error.message));
    }
    next();
  };

  return func;
};

module.exports = validation;

const mongoose = require('mongoose');
const Joi = require("joi");
const handleSaveErrors = require("../helpers/handelSaveErrors");

const { Schema } = mongoose;

const cinemaSchema = new Schema({
  name: {
    type: String,
    required: true,
    trim: true,
  },
  ticketPrice: {
    type: Number,
    required: true,
  },
  city: {
    type: String,
    required: true,
    trim: true,
    lowercase: true,
  },
  seats: {
    type: [Schema.Types.Mixed],
    required: true,
  },
  seatsAvailable: {
    type: Number,
    required: true,
  },
  image: {
    type: String,
  },
});

const Cinema = mongoose.model('Cinema', cinemaSchema);

module.exports = Cinema;

```

		Самчук Д.О.			КІК.КР.Б – ПЗ	Арк.
		Головня О.С.				
Змн.	Арк.	№ докум.	Підпис	Дата		83

```

const emailRegexp =
  /^[a-zA-Z0-9.!#$%&'*/+=?^_`{|}~-]+@[a-zA-Z0-9-]+(?:\.[a-zA-Z0-9-]+)*$/;

module.exports = emailRegexp;

const mongoose = require("mongoose");
const Joi = require("joi");

const handleSaveErrors = require("../helpers/handelSaveErrors");
const { Schema } = mongoose;

// Logout all
router.post("/users/logoutAll", auth.enhance, async (req, res) => {
  try {
    req.user.tokens = [];
    await req.user.save();
    res.send();
  } catch (e) {
    res.status(400).send(e);
  }
});

router.post("/verify", async (req, res) => {
  const { email } = req.body;
  const user = await User.findOne({ email });
  if (!user) {
    throw HttpError(404, `User not found`);
  }
  if (user.verify) {
    throw HttpError(400, "Verification has already been passed");
  }

  const newVerificationToken = nanoid();
  user.verificationToken = newVerificationToken;
  await user.save();

  const verifyEmail = createVerifyEmail(newVerificationToken, email);
  await sendEmail(verifyEmail);

  res.json({
    status: "success",
    code: 200,
    data: {
      message: "Verification email sent",
    },
  });
});

router.get("/verify/:verificationToken", async (req, res) => {
  const { verificationToken } = req.params;
  const user = await User.findOne({ verificationToken });

```

		Самчук Д.О.			КІК.КР.Б – ПЗ	Арк.
		Головня О.С.				
Змн.	Арк.	№ докум.	Підпис	Дата		84

```

if (!user) {
  throw HttpError(404, `User not found`);
}
await User.findByIdAndUpdate(user._id, {
  verify: true,
  verificationToken: "",
});

res.json({
  status: "success",
  code: 200,
  data: {
    message: "Verification successful",
  },
});
});

// Get all users
// eslint-disable-next-line consistent-return
router.get("/users", auth.enhance, async (req, res) => {
  if (req.user.role !== "superadmin")
    return res.status(400).send({
      error: "Only the god can see all the users!",
    });
  try {
    const users = await User.find({});
    res.send(users);
  } catch (e) {
    res.status(400).send(e);
  }
});

// User infos
router.get("/users/me", auth.simple, async (req, res) => {
  try {
    res.send(req.user);
  } catch (e) {
    res.status(400).send(e);
  }
});

// Get user by id only for admin
// eslint-disable-next-line consistent-return
router.get("/users/:id", auth.enhance, async (req, res) => {
  if (req.user.role !== "superadmin")
    return res.status(400).send({
      error: "Only the god can see the user!",
    });
  const _id = req.params.id;
  try {
    const user = await User.findById(_id);
    if (!user) return res.sendStatus(404);
  }
});

```

		Самчук Д.О.			КІК.КР.Б – ПЗ	Арк.
		Головня О.С.				
Змн.	Арк.	№ докум.	Підпис	Дата		85

```

    res.send(user);
  } catch (e) {
    res.sendStatus(400);
  }
});

// Edit/Update user
// eslint-disable-next-line consistent-return
router.patch("/users/me", auth.simple, async (req, res) => {
  console.log(req.body);
  const updates = Object.keys(req.body);
  const allowedUpdates = ["name", "phone", "username", "email", "password"];
  const isValidOperation = updates.every((update) =>
    allowedUpdates.includes(update)
  );
  if (!isValidOperation)
    return res.status(400).send({ error: "Invalid updates!" });

  try {
    const { user } = req;
    // eslint-disable-next-line no-return-assign
    updates.forEach((update) => (user[update] = req.body[update]));
    await user.save();
    res.send(user);
  } catch (e) {
    res.status(400).send(e);
  }
});

// Admin can update user by id
// eslint-disable-next-line consistent-return
router.patch("/users/:id", auth.enhance, async (req, res) => {
  if (req.user.role !== "superadmin")
    return res.status(400).send({
      error: "Only the god can update the user!",
    });
  const _id = req.params.id;

  const updates = Object.keys(req.body);
  const allowedUpdates = [
    "name",
    "phone",
    "username",
    "email",
    "password",
    "role",
  ];
  const isValidOperation = updates.every((update) =>
    allowedUpdates.includes(update)
  );

  if (!isValidOperation)

```

		Самчук Д.О.			КІК.КР.Б – ПЗ	Арк.
		Головня О.С.				86
Змн.	Арк.	№ докум.	Підпис	Дата		

```

    return res.status(400).send({ error: "Invalid updates!" });

    try {
        const user = await User.findById(_id);
        // eslint-disable-next-line no-return-assign
        updates.forEach((update) => (user[update] = req.body[update]));
        await user.save();

        if (!user) return res.sendStatus(404);
        res.send(user);
    } catch (e) {
        res.status(400).send(e);
    }
});

// Delete by id
// eslint-disable-next-line consistent-return
router.delete("/users/:id", auth.enhance, async (req, res) => {
    if (req.user.role !== "superadmin")
        return res.status(400).send({
            error: "Only the god can delete the user!",
        });
    const _id = req.params.id;

    try {
        const user = await User.findByIdAndDelete(_id);
        if (!user) return res.sendStatus(404);

        res.send({ message: "User Deleted" });
    } catch (e) {
        res.sendStatus(400);
    }
});

// eslint-disable-next-line consistent-return
router.delete("/users/me", auth.simple, async (req, res) => {
    if (req.user.role !== "superadmin")
        return res.status(400).send({
            error: "You cannot delete yourself!",
        });
    try {
        await req.user.remove();
        res.send(req.user);
    } catch (e) {
        res.sendStatus(400);
    }
});

router.get("/verify/:verificationToken", async (req, res) => {
    const { verificationToken } = req.params;
    const user = await User.findOne({ verificationToken });
    if (!user) {

```

		Самчук Д.О.			КІК.КР.Б – ПЗ	Арк.
		Головня О.С.				
Змн.	Арк.	№ докум.	Підпис	Дата		87

```

    throw HttpError(404, `User not found`);
  }
  await User.findByIdAndUpdate(user._id, {
    verify: true,
    verificationToken: "",
  });

  res.json({
    status: "success",
    code: 200,
    data: {
      message: "Verification successful",
    },
  });
});

module.exports = router;

const { User } = require("../models/user");

const increaseLoginAttempts = async (email) => {
  try {
    const user = await User.findOne({ email })
      .select("+email +loginAttempts +lockUntil")
      .lean();
    if (!user) {
      throw new Error("User not found");
    }

    user.loginAttempts += 1;
    if (user.loginAttempts >= 5) {
      user.lockUntil = Date.now() + 24 * 60 * 60 * 1000;
    }

    // Update user without validation
    await User.updateOne(
      { _id: user._id },
      { $set: { loginAttempts: user.loginAttempts, lockUntil: user.lockUntil } }
    );
  } catch (error) {
    console.error("Failed to increase login attempts:", error);
    throw new Error("Failed to increase login attempts: " + error.message);
  }
};

module.exports = increaseLoginAttempts;

const QRCode = require('qrcode');

const generateQR = async text => {
  try {
    return await QRCode.toDataURL(text);
  }

```

		Самчук Д.О.			КІК.КР.Б – ПЗ	Арк.
		Головня О.С.				88
Змн.	Арк.	№ докум.	Підпис	Дата		

```

    } catch (err) {
      return console.error(err);
    }
  };

module.exports = generateQR;

const nodemailer = require('nodemailer');

const transporter = nodemailer.createTransport({
  host: 'smtp.meta.ua',
  secure: true,
  port: 465,
  auth: {
    user: process.env.GMAIL_USER,
    pass: process.env.GMAIL_PASSWORD,
  },
  tls: {
    rejectUnauthorized: false,
  },
});

transporter.sendEmail = function(mailRequest) {
  return new Promise(function(resolve, reject) {
    transporter.sendMail(mailRequest, error => {
      if (error) {
        reject(error);
      } else {
        resolve("The message was sent!");
      }
    });
  });
};

module.exports = transporter;

const multer = require("multer");
const path = require("path");
const { nanoid } = require("nanoid");

const tmpDir = path.join(__dirname, "..", "tmp");

const multerConfig = multer.diskStorage({
  destination: tmpDir,
  filename: (req, file, cb) => {
    const uniqueFilename = nanoid() + path.extname(file.originalname);
    cb(null, uniqueFilename);
  },
});

const upload = multer({
  storage: multerConfig,

```

		Самчук Д.О.			КІК.КР.Б – ПЗ	Арк.
		Головня О.С.				89
Змн.	Арк.	№ докум.	Підпис	Дата		

```

fileFilter: (req, file, cb) => {
  if (file.mimetype.startsWith("image/")) {
    cb(null, true);
  } else {
    cb(new Error("Неприпустимий тип файлу"));
  }
},
});

module.exports = upload;

const Reservation = require('../models/reservation');
const Movie = require('../models/movie');
const Cinema = require('../models/cinema');

// Cinema User modeling (GET ALL CINEMAS)
// Get all cinemas based on the user's past reservations
// @return a sorted cinema list
const cinemaUserModeling = async (cinemas, username) => {
  const userReservations = await Reservation.find({ username: username });

  if (userReservations.length) {
    const cinemaResult = {};
    // eslint-disable-next-line array-callback-return
    userReservations.map(userReservation => {
      const id = userReservation.cinemaId;
      // eslint-disable-next-line no-unused-expressions, no-prototype-builtins, no-plusplus
      cinemaResult.hasOwnProperty(id) ? ++cinemaResult[id] : (cinemaResult[id] = 1);
    });
    const sortedCinemaResult = [];
    // eslint-disable-next-line no-restricted-syntax, guard-for-in
    for (const cinema in cinemaResult) {
      sortedCinemaResult.push([cinema, cinemaResult[cinema]]);
    }

    sortedCinemaResult.sort((a, b) => {
      return b[1] - a[1];
    });
    console.log(sortedCinemaResult);

    const newCinemas = JSON.parse(JSON.stringify(cinemas));
    let i = 0;
    let extractedObj;
    // eslint-disable-next-line no-restricted-syntax
    for (const sortedCinema of sortedCinemaResult) {
      // eslint-disable-next-line no-loop-func
      newCinemas.forEach((cinema, index) => {
        // eslint-disable-next-line eqeqeq
        if (cinema._id == sortedCinema[0]) {
          console.log('FOUND');
          extractedObj = newCinemas.splice(index, 1);
        }
      })
    }
  }
}

```

		Самчук Д.О.			КІК.КР.Б – ПЗ	Арк.
		Головня О.С.				90
Змн.	Арк.	№ докум.	Підпис	Дата		

```

    });
    newCinemas.splice(i, 0, extractedObj[0]);
    // eslint-disable-next-line no-plusplus
    i++;
  }

  console.log(newCinemas);

  return newCinemas;
}
return cinemas;
};

const moviesUserModeling = async username => {
  // eslint-disable-next-line no-undef
  userPreference = {
    genre: {},
    director: {},
    cast: {},
  };

  const userReservations = JSON.parse(
    JSON.stringify(await Reservation.find({ username: username })))
  );
  const Allmovies = JSON.parse(JSON.stringify(await Movie.find({})));

  // eslint-disable-next-line array-callback-return, consistent-return
  const moviesWatched = userReservations.map(reservation => {
    // eslint-disable-next-line no-restricted-syntax
    for (const movie of Allmovies) {
      // eslint-disable-next-line eqeqeq
      if (movie._id === reservation.movieId) {
        return movie;
      }
    }
  });

  // console.log(moviesWatched);

  // eslint-disable-next-line array-callback-return
  moviesWatched.map(movie => {
    const genres = movie.genre.replace(/\s*,\s*/g, ',').split(',');
    const directors = movie.director.replace(/\s*,\s*/g, ',').split(',');
    const casts = movie.cast.replace(/\s*,\s*/g, ',').split(',');
    // eslint-disable-next-line no-restricted-syntax
    for (const genre of genres) {
      // eslint-disable-next-line no-unused-expressions, no-undef
      if (userPreference.genre[genre]) {
        // eslint-disable-next-line no-plusplus, no-undef
        ++userPreference.genre[genre];
      } else {
        // eslint-disable-next-line no-undef

```

		Самчук Д.О.			КІК.КР.Б – ПЗ	Арк.
		Головня О.С.				91
Змн.	Арк.	№ докум.	Підпис	Дата		

```

        userPreference.genre[genre] = 1;
    }
}
// eslint-disable-next-line no-restricted-syntax
for (const director of directors) {
    // eslint-disable-next-line no-unused-expressions, no-undef
    userPreference.director[director]
        ? // eslint-disable-next-line no-plusplus, no-undef
          ++userPreference.director[director]
        : // eslint-disable-next-line no-undef
          (userPreference.director[director] = 1);
}
// eslint-disable-next-line no-restricted-syntax
for (const cast of casts) {
    // eslint-disable-next-line no-unused-expressions, no-undef, no-plusplus
    userPreference.cast[cast] ? ++userPreference.cast[cast] : (userPreference.cast[cast] = 1);
}
});

// console.log(userPreference)

// find movies that are available for booking
// eslint-disable-next-line no-use-before-define
const availableMovies = availableMoviesFilter(Allmovies);
// console.log(availableMovies)
// eslint-disable-next-line no-use-before-define
const moviesNotWatched = moviesNotWatchedFilter(availableMovies, userReservations);
// console.log(moviesNotWatched)

// eslint-disable-next-line no-use-before-define, no-undef
const moviesRated = findRates(moviesNotWatched, userPreference);

moviesRated.sort((a, b) => {
    return b[1] - a[1];
});
// console.log(moviesRated)

const moviesToObject = moviesRated.map(array => {
    return array[0];
});
return moviesToObject;
};

const findRates = (moviesNotWatched, userPreference) => {
    const result = [];
    let rate;
    // eslint-disable-next-line no-restricted-syntax
    for (const movie of moviesNotWatched) {
        rate = 0;
        // eslint-disable-next-line no-restricted-syntax, guard-for-in
        for (const pref in userPreference) {
            // eslint-disable-next-line no-use-before-define

```

		Самчук Д.О.			КІК.КР.Б – ПЗ	Арк.
		Головня О.С.				92
Змн.	Арк.	№ докум.	Підпис	Дата		

```

    rate += getRateOfProperty(pref, userPreference, movie);
    // TODO we can use weights here
    console.log(rate, pref);
  }
  if (rate !== 0) result.push([movie, rate]);
}
// console.log(result);
return result;
};

const getRateOfProperty = (pref, userPreference, movie) => {
  let rate = 0;
  const values = Object.keys(userPreference[pref]).map(key => {
    return [key, userPreference[pref][key]];
  });
  const movieValues = movie[pref].replace(/\s*,\s*/g, ',').split(',');
  // eslint-disable-next-line no-restricted-syntax, no-const-assign
  for (values of values) {
    if (movieValues.length) {
      // eslint-disable-next-line no-restricted-syntax, no-const-assign
      for (movieValues of movieValues) {
        // eslint-disable-next-line eqeqeq
        if (movieValues == values[0]) {
          rate += values[1];
        }
      }
    }
  }
}

return rate;
};

const availableMoviesFilter = Allmovies => {
  const today = new Date();
  const returnMovies = [];
  // eslint-disable-next-line array-callback-return
  Allmovies.map(movie => {
    const releaseDate = new Date(movie.releaseDate);
    const endDate = new Date(movie.endDate);
    if (today >= releaseDate && today <= endDate) {
      returnMovies.push(movie);
    }
  });
  return returnMovies;
};

const moviesNotWatchedFilter = (availableMovies, userReservations) => {
  const returnMovies = [];
  // eslint-disable-next-line array-callback-return
  availableMovies.map(movie => {
    const isFirst = [];
    // eslint-disable-next-line no-restricted-syntax

```

		Самчук Д.О.			КІК.КР.Б – ПЗ	Арк.
		Головня О.С.				93
Змн.	Арк.	№ докум.	Підпис	Дата		

```

for (const reservation of userReservations) {
  // eslint-disable-next-line eqeqeq
  if (reservation.movieId == movie._id) {
    isFirst.push(false);
  } else {
    isFirst.push(true);
  }
}

if (isFirst.every(Boolean)) {
  returnMovies.push(movie);
}
});
return returnMovies;
};

const reservationSeatsUserModeling = async username => {
  const numberOfTicketsArray = [];
  let numberOfTickets = 1;
  const positions = {
    front: 0,
    center: 0,
    back: 0,
  };
  const cinemas = JSON.parse(JSON.stringify(await Cinema.find({})));
  const userReservations = JSON.parse(
    JSON.stringify(await Reservation.find({ username: username })))
  );

  // eslint-disable-next-line array-callback-return
  userReservations.map(reservation => {
    // eslint-disable-next-line no-restricted-syntax
    for (const cinema of cinemas) {
      // eslint-disable-next-line eqeqeq
      if (cinema._id == reservation.cinemaId) {
        // find how many rows the cinema has
        // eslint-disable-next-line no-use-before-define
        const position = getPosition(cinema.seats.length, reservation.seats);
        // eslint-disable-next-line no-plusplus
        ++positions[position];
        numberOfTicketsArray.push(reservation.seats.length);
      }
    }
  });
  numberOfTickets = Math.round(
    numberOfTicketsArray.reduce((a, b) => a + b, 0) / numberOfTicketsArray.length
  );

  return {
    numberOfTickets,
    positions,
  };
};

```

		Самчук Д.О.			КІК.КР.Б – ПЗ	Арк.
		Головня О.С.				94
Змн.	Арк.	№ докум.	Підпис	Дата		

```

};

// eslint-disable-next-line consistent-return
const getPosition = (cinemaRows, seats) => {
  const rowNum = seats[0][0];
  const step = cinemaRows / 3;
  let pos = 1;
  for (let i = step; i <= cinemaRows; i += step) {
    if (rowNum < i) {
      // eslint-disable-next-line default-case
      switch (pos) {
        case 1:
          return 'front';
        case 2:
          return 'center';
        case 3:
          return 'back';
      }
    }
    // eslint-disable-next-line no-plusplus
    pos++;
  }
};

const userModeling = {
  cinemaUserModeling,
  moviesUserModeling,
  reservationSeatsUserModeling,
};

module.exports = userModeling;

<html lang="uk">

<head>
  <meta charset="UTF-8" />
  <meta name="viewport" content="width=device-width, initial-scale=1.0" />
  <title>Підтвердження реєстрації</title>
  <style>
    body { font-family: Arial, sans-serif; background-color: #f4f4f4; margin:
    0; padding: 0; } .container { max-width: 600px; margin: 50px auto;
    padding: 20px; background-color: #ffffff; border-radius: 10px; box-shadow:
    0 0 10px rgba(0, 0, 0, 0.1); } h1 { color: #333333; text-align: center;
    margin-bottom: 30px; } p { color: #666666; font-size: 16px; line-height:
    1.5; margin-bottom: 20px; } .btn { display: inline-block;
    background-color: #007bff; color: #ffffff; text-decoration: none; padding:
    10px 20px; border-radius: 5px; } .btn:hover { background-color: #0056b3; }
  </style>
</head>

<body>
  <div class="container">

```

		Самчук Д.О.			КІК.КР.Б – ПЗ	Арк.
		Головня О.С.				
Змн.	Арк.	№ докум.	Підпис	Дата		95

