Міністерство освіти і науки України
Харківський національний університет радіоелектроніки

Факультет Інформаційно-аналітичних технологій та менеджменту
(повна назва)
Кафедра Інформатики
(повна назва)

КВАЛІФІКАЦІЙНА РОБОТА
Пояснювальна записка

рівень вищої освіти другий (магістерський)

ДОСЛІДЖЕННЯ МЕТОДІВ ОПТИМІЗАЦІЇ СЕРВЕРНОЇ ЧАСТИНИ ІНФОРМАЦІЙНОЇ СИСТЕМИ ДЛЯ ПІДВИЩЕННЯ ЇЇ ПРОДУКТИВНОСТІ ТА БЕЗПЕКИ ДАНИХ
(тема)

Виконав:
студент 2 курсу, групи ІНФм-23-2
Караконстантин Д.О.
(прізвище, ініціали)

Спеціальності 122 Комп'ютерні науки
(код і повна назва спеціальності)

Тип програми освітньо-професійна

Освітня програма Інформатика
(повна назва освітньої програми)

Керівник доц. Творошенко І.С.
(посада, прізвище, ініціали)

Допускається до захисту

Зав. кафедри _____ Кобилін О.А.
(підпис) (прізвище, ініціали)

2025 р.

Харківський національний університет радіоелектроніки

Факультет Інформаційно-аналітичних технологій та менеджменту
(повна назва)

Кафедра Інформатики
(повна назва)

Рівень вищої освіти другий (магістерський)

Спеціальність 122 Комп'ютерні науки
(код і повна назва)

Тип програми освітньо-професійна

Освітня програма Інформатика
(повна назва освітньої програми)

ЗАТВЕРДЖУЮ:
Зав. кафедри _____
(підпис)
«____»_____ 2025 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

студентові Караконстантину Данієлю Олеговичу
(прізвище, ім'я, по батькові)

1. Тема роботи Дослідження методів оптимізації серверної частини інформаційної системи для підвищення її продуктивності та безпеки даних
затверджена наказом по університету від 25 листопада 2024 року № 1246Ст
2. Термін подання студентом роботи до екзаменаційної комісії 23 грудня 2024 р.
3. Вихідні дані до роботи науково-методична та науково-технічна література, матеріали конференції, дані інтернет-мережі, перелік використовуваних програмних засобів: система управління базами даних Redis, вебсервер Nginx, платформа для контейнеризації застосунків Docker.

4. Перелік питань, що потрібно опрацювати в роботі
1. Проаналізувати головну мету оптимізації серверної частини інформаційної системи.
2. Здійснити класифікація існуючих методів оптимізації серверної частини.
3. Проаналізувати сучасні методи оптимізації серверної частини інформаційної системи.
4. Виявити особливості вибраних методів оптимізації серверної частини інформаційної системи для підвищення її продуктивності та безпеки даних.
5. Здійснити вибір інструментальних засобів для реалізації вибраних методів.
6. Визначити етапи програмної реалізації вибраних методів оптимізації серверної частини інформаційної системи.
7. Порівняти аналіз досліджених методів оптимізації серверної частини інформаційної системи.
8. Виявити перспективи подальшої роботи.

5. Перелік графічного матеріалу із зазначенням креслеників, схем, плакатів, комп'ютерних ілюстрацій (п.5 включається до завдання за рішенням випускової кафедри) актуальність дослідження, об'єкт та мета дослідження, постановка задачі дослідження, етапи дослідження, результати тестування, висновки, перспективи подальшої роботи, апробація результатів дослідження.

6. Консультанти розділів роботи (п.6 включається до завдання за наявності консультантів згідно з наказом, зазначеним у п.1)

Найменування розділу	Консультант (посада, прізвище, ім'я, по батькові)	Позначка консультанта про виконання розділу	
		підпис	дата

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів роботи	Терміни виконання етапів роботи	Примітка
1	Отримання завдання на кваліфікаційну роботу	25.11.2024	
2	Аналіз завдання, підбір літератури	25.11.24-27.11.24	
3	Аналіз літератури з досліджуваної проблеми	27.11.24-29.11.24	
4	Аналіз сучасних методів оптимізації	29.11.24-01.12.24	
5	Дослідження методів оптимізації	01.12.24-04.12.24	
6	Програмна реалізація	04.12.24-09.12.24	
7	Оформлення пояснювальної записки	09.12.24-19.12.24	
8	Перевірка на плагіат	22.12.2024	
9	Рецензування	26.12.2024	
10	Підготовка презентації та доповіді	28.12.2024	
11	Занесення роботи в електронний архів	31.12.2024	
12	Попередній захист кваліфікаційної роботи	02.01.2025	

Дата видачі завдання 25 листопада 2024 р.

Студент _____
(підпис)

Керівник роботи _____ доц. Творошенко І.С.
(підпис) (посада, прізвище, ініціали)

РЕФЕРАТ/ABSTRACT

Пояснювальна записка до кваліфікаційної роботи: 89 с., 3 табл., 11 рис., 53 джерела.

ОПТИМІЗАЦІЯ СЕРВЕРНОЇ ЧАСТИНИ, ІНФОРМАЦІЙНІ СИСТЕМИ, КЕШУВАННЯ, БАГАТОПОТОЧНІСТЬ, БАЗИ ДАНИХ, ПРОДУКТИВНІСТЬ, БЕЗПЕКА ДАНИХ.

Об’єктом дослідження є серверна частина інформаційної системи, яка забезпечує оброблення та збереження даних, а також взаємодію з клієнтськими застосунками.

Метою дослідження є порівняння методів оптимізації серверної частини інформаційної системи для підвищення її продуктивності та безпеки даних, а також виявлення найефективніших з них.

Методи дослідження включають аналіз літературних джерел, експериментальну розробку та тестування програмних рішень, а також порівняльний аналіз методів оптимізації. Інструменти: сучасні мови програмування, бібліотеки, фреймворки та апаратні рішення для підвищення продуктивності й безпеки серверної частини інформаційної системи.

Результати дослідження включають порівняння та аналіз ефективності обраних методів оптимізації продуктивності та безпеки серверної частини інформаційної системи за критеріями продуктивності, безпеки, стійкості до навантаження та зручності інтеграції.

OPTIMIZATION OF THE SERVER SIDE, INFORMATION SYSTEMS, CACHING, MULTITHREADING, DATABASES, PERFORMANCE, DATA SECURITY.

The object of this research is the server-side of an information system, which handles data processing and storage, as well as interacts with client applications.

The aim of the research is to compare methods for optimizing the server side of an information system to enhance its performance and data security, as well as to identify the most effective ones.

The research methods include literature analysis, experimental development and testing of software solutions, as well as a comparative analysis of optimization methods. Tools include modern programming languages, libraries, frameworks, and hardware solutions aimed at improving the performance and security of the information system’s server side.

The research results include a comparison and analysis of the effectiveness of selected methods for optimizing the performance and security of the information system’s server side based on criteria such as performance, security, load resilience, and ease of integration.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

БД – база даних

ІС – інформаційна система

ОС – операційна система

ПЗ – програмне забезпечення

ABAC – Attribute-Based Access Control (контроль доступу на основі атрибутів)

APM – Application Performance Monitoring (моніторинг продуктивності застосунків)

AVG – функція SQL для обчислення середнього значення

COUNT – функція SQL для підрахунку записів

CRM – Customer Relationship Management (система управління взаємовідносинами з клієнтами)

DAC – Discretionary Access Control (дискреційний контроль доступу)

EFS – Encrypted File System (зашифрована файлова система)

ELK Stack – Elasticsearch, Logstash, Kibana Stack (набір інструментів для обробки записів)

EXPLAIN – команда SQL для аналізу виконання запиту

FIFO – First In, First Out (перший прийшов – перший пішов)

GDPR – General Data Protection Regulation (загальний регламент захисту даних)

HIPAA – Health Insurance Portability and Accountability Act (закон про перенесення та підзвітність медичного страхування)

HTML – HyperText Markup Language (мова розмітки гіпертексту)

HTTP – HyperText Transfer Protocol (протокол передачі гіпертексту)

IDS – Intrusion Detection System (система виявлення вторгнень)

IPS – Intrusion Prevention System (система запобігання вторгненням)

IT – Information Technology (інформаційні технології)

JOIN – поєднання рядків з кількох таблиць в SQL

LRU – Least Recently Used (найменш використовуваний нещодавно)

LFU – Least Frequently Used (найменш використовуваний)

MAC – Mandatory Access Control (обов’язковий контроль доступу)

MFA – Multi-Factor Authentication (багатофакторна аутентифікація)

OAuth – Open Authorization (відкрита авторизація)

ORM – Object-Relational Mapping (об’єктно-реляційне відображення)

OTP – One-Time Password (одноразовий пароль)

PCI DSS – Payment Card Industry Data Security Standard (Стандарт безпеки даних індустрії платіжних карт)

RBAC – Role-Based Access Control (контроль доступу на основі ролей)

SIEM – Security Information and Event Management (керування інформацією та подіями безпеки)

SQL – Structured Query Language (мова структурованих запитів)

SSL – Secure Sockets Layer (рівень захищених з’єднань)

SUM – функція SQL для підсумовування значень

TLS – Transport Layer Security (протокол захисту транспортного рівня)

TPL – Template Processing Language (мова обробки шаблонів)

TTL – Time To Live (час життя)

WHERE – умова в SQL, яка вказує на обмеження

ВСТУП

Сучасний світ швидко розвивається у напрямку інформаційних технологій, які стають невід'ємною частиною життя. Зростаюча роль інформаційних систем (ІС) у різних сферах діяльності, від бізнесу до державного управління, обумовлює необхідність їх постійного вдосконалення. Оптимізація серверної частини ІС є одним із ключових аспектів, що визначає загальну продуктивність та безпеку системи. Висока продуктивність ІС забезпечує швидкий доступ до даних, ефективну обробку запитів і підтримку багатокористувацького режиму роботи, тоді як безпека даних захищає інформацію від несанкціонованого доступу та забезпечує цілісність даних.

Актуальність дослідження методів оптимізації серверної частини ІС зумовлена декількома чинниками:

– зростаючий обсяг даних і кількість користувачів, що взаємодіють із системою, вимагають підвищення продуктивності серверної частини;

– все більше значення приділяється безпеці даних, особливо в контексті постійних загроз кіберзлочинності;

– технологічні зміни й інновації постійно створюють нові можливості для покращення ІС, що потребує постійного перегляду і оновлення методів оптимізації.

Основна мета оптимізації серверної частини ІС полягає у забезпеченні швидкої та надійної обробки даних, що є критично важливим для ефективної роботи будь-якої організації. Серверна частина ІС включає апаратні ресурси (сервери, сховища даних, мережеві пристрої) та програмне забезпечення (системи управління базами даних, вебсервери, застосунки), які забезпечують обробку і зберігання даних, а також взаємодію з клієнтською частиною системи.

Класифікація існуючих методів оптимізації серверної частини ІС включає різноманітні підходи, такі як оптимізація коду програм, ефективне

використання апаратних ресурсів, оптимізація запитів до бази даних, кешування даних, а також використання багатопоточності та асинхронної обробки даних. Кожен з цих методів має свої переваги та недоліки і може бути застосований залежно від конкретних вимог і обмежень системи.

Аналіз сучасних методів оптимізації серверної частини ІС дозволяє виявити найефективніші підходи до підвищення продуктивності та безпеки системи. Зокрема, кешування даних дозволяє зменшити час доступу до часто використовуваних даних, а асиметрична багатопоточність даних сприяє більш ефективному використанню процесорних ресурсів. Оптимізація запитів до бази даних дозволяє зменшити навантаження на сервер і покращити швидкість обробки запитів.

Методи забезпечення безпеки даних включають шифрування даних, аутентифікацію та авторизацію користувачів, а також моніторинг та аудит безпеки даних. Шифрування даних забезпечує захист інформації від несанкціонованого доступу, аутентифікація та авторизація гарантують, що до даних мають доступ лише авторизовані користувачі. Моніторинг та аудит безпеки дозволяють виявляти і реагувати на потенційні загрози та інциденти безпеки в реальному часі.

Постановка задачі дослідження полягає в аналізі та виборі найбільш ефективних методів оптимізації серверної частини ІС для підвищення її продуктивності та безпеки даних. Під час роботи необхідно виконати: дослідження існуючих методів, вибір інструментальних засобів для їх реалізації, програмне впровадження обраних методів та оцінювання їх ефективності в контексті конкретної предметної області.

1 АНАЛІЗ ІСНУЮЧИХ МЕТОДІВ ОПТИМІЗАЦІЇ СЕРВЕРНОЇ ЧАСТИНИ ІНФОРМАЦІЙНОЇ СИСТЕМИ

1.1 Головна мета оптимізації серверної частини інформаційної системи під час експлуатації ІТ-застосунків

Оптимізація серверної частини інформаційної системи під час експлуатації ІТ-застосунків є критично важливим процесом, який забезпечує ефективність, стабільність і надійність програмних рішень в умовах реального використання. Цей процес включає широкий спектр заходів, спрямованих на підвищення продуктивності серверів, мінімізацію витрат ресурсів і забезпечення безперебійної роботи в умовах різних рівнів навантаження.

Основна мета оптимізації серверної частини полягає в покращенні якості обслуговування користувачів, зменшенні часу реакції на запити та підвищенні загальної ефективності системи. Як зазначає Джеймс Сміт у своїй праці [1], оптимізація серверної частини дозволяє забезпечити постійно високу продуктивність навіть під час пікових навантажень. Досягнення цього можливе за допомогою різних технологічних підходів, які спрямовані на покращення управління ресурсами, зменшення затримок та підвищення загальної ефективності роботи серверів.

Серед основних завдань оптимізації серверної частини можна виділити декілька ключових аспектів. Необхідно забезпечити стабільну роботу серверів, що є основою надійної роботи всіх ІТ-застосунків. Карл Ліндберг у своїй роботі [2] наголошує на тому, що безперервність функціонування серверів має ключове значення для забезпечення високої якості обслуговування клієнтів, особливо в умовах високої конкуренції. Водночас, варто звернути увагу на питання безпеки, оскільки оптимізація серверної частини також включає захист даних і запобігання кіберзагрозам (рис. 1.1).

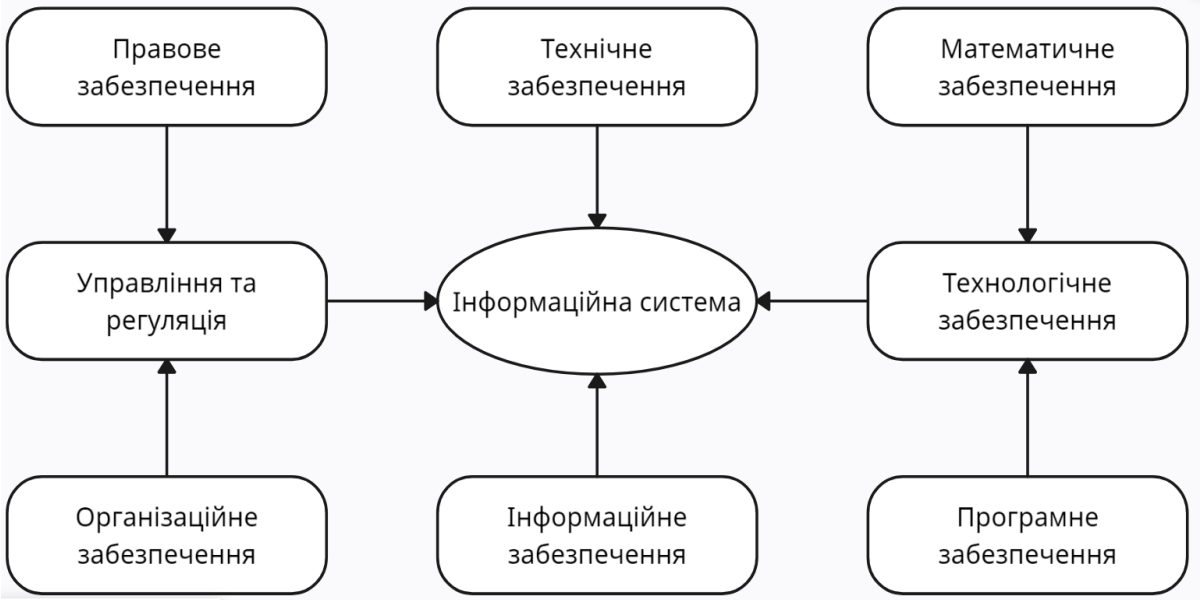

Рисунок 1.1 – Основні складові інформаційної системи

Іншим важливим аспектом є забезпечення належної масштабованості системи. Це дозволяє ІТ-застосункам ефективно працювати за збільшення кількості користувачів або зростання обсягу оброблюваних даних. Масштабованість системи забезпечується завдяки використанню технологій горизонтального масштабування серверів, що дозволяє додавати нові сервери до існуючої інфраструктури без значних змін в архітектурі системи. Джон Вільямс зазначає, що правильне управління масштабованістю може забезпечити високі показники продуктивності навіть за умов різкого зростання кількості користувачів (рис. 1.2 [3]).

Ефективне використання ресурсів є ще однією важливою складовою оптимізації серверної частини. Це питання стосується як апаратного забезпечення, так і програмного середовища. У випадку недостатнього використання серверних ресурсів можливі затримки у виконанні запитів, що призводить до погіршення користувацького досвіду. З іншого боку, надмірне використання ресурсів може призвести до зниження продуктивності системи, особливо у випадках, коли ІТ-застосунки працюють з великими обсягами даних.

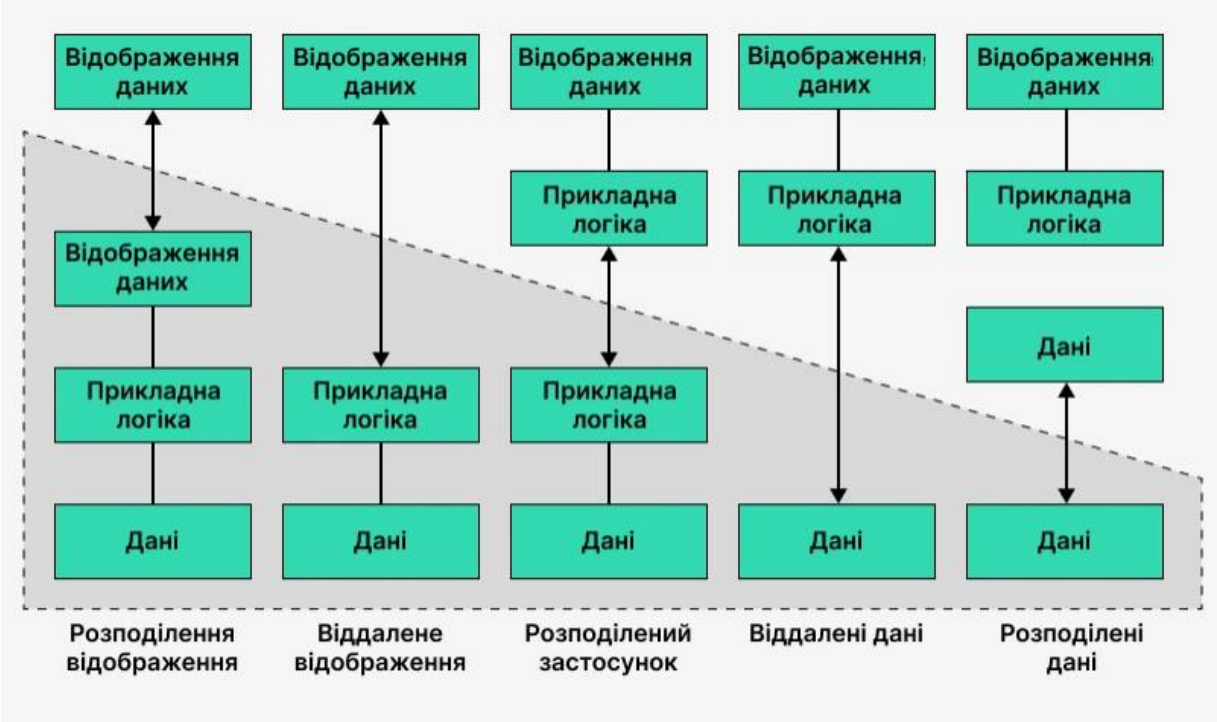

Рисунок 1.2 – Схема розподілу ресурсів і компонентів
в інформаційній системі

Фредерік Тейлор у своїй роботі наголошує, що оптимізація серверної частини інформаційної системи потребує чіткого розуміння специфіки використання ресурсів, а також планування для уникнення непотрібних втрат [4].

Вище зазначене підтверджує необхідність постійного моніторингу стану системи та своєчасної адаптації її конфігурації для забезпечення максимальних показників продуктивності.

Одним з найважливіших аспектів оптимізації серверної частини є мінімізація затримок та скорочення часу відповіді на запити користувачів. Для досягнення цієї мети використовуються різноманітні методи та технології, серед яких варто виділити кешування, стиснення даних та покращення алгоритмів обробки запитів. Зокрема, застосування кешування дозволяє зберігати часто використовувані дані у пам’яті сервера, що значно скорочує час доступу до них.

Застосування технологій кешування описується в роботі [5], де зазначається, що кешування є одним з найбільш ефективних методів зменшення навантаження на сервер та прискорення обробки запитів. Сучасні системи кешування дозволяють зберігати не лише статичні файли, але й динамічні дані, що значно покращує загальну продуктивність системи.

Для забезпечення рівномірного розподілу навантаження між різними компонентами серверної інфраструктури важливо використовувати технології балансування навантаження. Це дозволяє зменшити навантаження на окремі сервери, що значно підвищує стабільність роботи системи та запобігає її перевантаженню

Як зазначається у [6], балансування навантаження є основною умовою забезпечення безперебійної роботи системи, особливо в умовах високої конкуренції за серверні ресурси. Застосування різних алгоритмів балансування, таких як Round Robin або Least Connections, дозволяє досягати ефективного розподілу трафіку між різними серверами.

Застосування хмарних обчислень також є важливим елементом оптимізації серверної частини інформаційної системи. Хмарні технології дозволяють знизити витрати на апаратне забезпечення, забезпечуючи при цьому високий рівень масштабованості та надійності. За словами Гордона Льюїса [7], використання хмарних сервісів дозволяє гнучко управляти ресурсами, швидко адаптуючись до зміни навантаження.

Хмарні технології надають можливість легко змінювати обсяг використовуваних ресурсів в залежності від поточних потреб, що є особливо важливим для великих IT-застосунків з високим рівнем навантаження.

Оптимізація серверної частини також передбачає підвищення рівня безпеки системи. Це включає впровадження механізмів захисту від несанкціонованого доступу, шифрування даних та регулярне оновлення безпекових патчів. Як наголошує Сара Вільямс у своїй роботі [8], безпека даних повинна бути однією з ключових пріоритетів під час оптимізації серверної частини, оскільки витік даних може мати катастрофічні наслідки.

Окрім того, важливим аспектом є забезпечення відмовостійкості серверної частини, що дозволяє системі залишатися працездатною навіть у випадках технічних збоїв або атак. Для цього використовуються технології резервування серверів та баз даних, що забезпечують безперервність роботи системи навіть у разі виходу з ладу одного або кількох її компонентів.

Оптимізація серверної частини інформаційної системи є багатогранним процесом, що охоплює різноманітні аспекти, починаючи від балансування навантаження і закінчуючи підвищенням безпеки. Ефективна оптимізація дозволяє зменшити час відповіді на запити користувачів, підвищити продуктивність та забезпечити стабільну роботу системи за умов високих навантажень. Важливим є використання сучасних технологій, таких як хмарні обчислення, кешування та алгоритми балансування навантаження, що дозволяє забезпечити гнучкість та надійність роботи інформаційної системи.

1.2 Класифікація існуючих методів оптимізації серверної частини інформаційної системи

Класифікація методів оптимізації серверної частини інформаційної системи є важливою для розуміння підходів до підвищення ефективності роботи серверів, особливо в умовах постійно зростаючого навантаження на інформаційні системи. Залежно від специфіки системи та інфраструктури, можуть застосовуватися різні методи, що дозволяють забезпечити стабільну роботу, максимізувати продуктивність і мінімізувати ризики простоїв. Основні аспекти класифікації включають підхід до обробки запитів, управління ресурсами, використання інфраструктури та забезпечення безпеки системи.

Одним із ключових напрямів оптимізації серверної частини є підхід до обробки запитів, який значно впливає на продуктивність системи, особливо в умовах великого потоку користувачів або даних. Одним із популярних

методів є асинхронна обробка запитів, що дозволяє системі одночасно виконувати декілька запитів без блокування основних процесів. Це забезпечує можливість обробки багатьох запитів в режимі реального часу без значних затримок.

Асинхронна обробка є особливо ефективною для сучасних веб-сервісів, де затримки можуть призвести до втрати користувачів і зниження загальної продуктивності. У дослідженні Томаса Гріна [1] йдеться про те, що асинхронні методи обробки можуть знизити затримки до 30%, що є суттєвим показником для високонавантажених систем. Завдяки цьому, сервери можуть обробляти більше запитів одночасно, що позитивно впливає на загальну швидкість роботи системи та її здатність відповідати на запити користувачів. Асинхронна обробка запитів, як показано на рисунку 1.3 (кроки 1, 2, 4, 5), дозволяє системі обробляти запити без блокування процесів, що покращує загальну продуктивність.

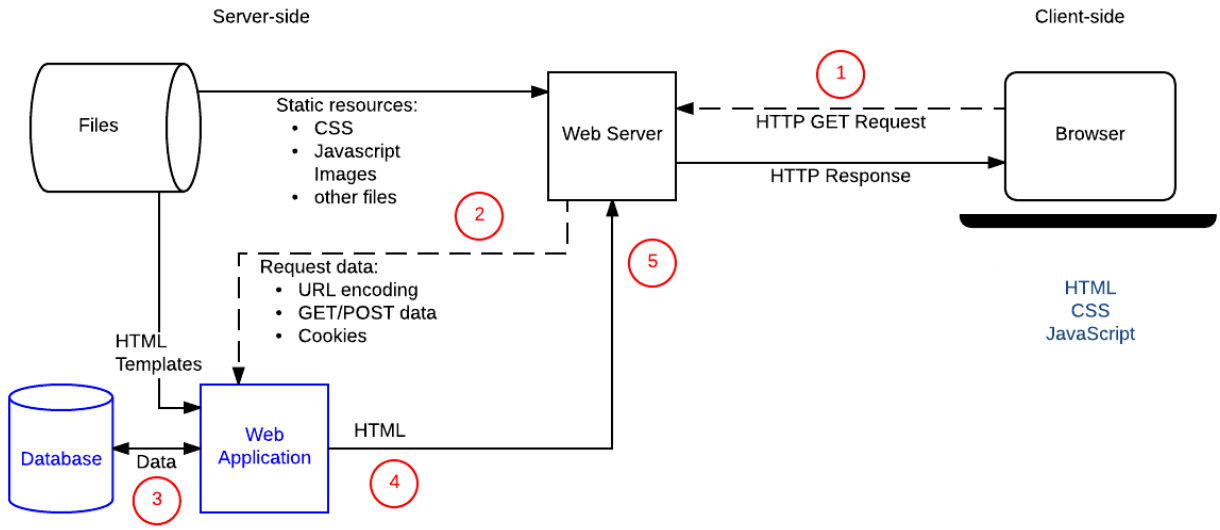

Рисунок 1.3 – Архітектура обробки запитів та взаємодії між клієнтською і серверною частинами інформаційної системи

Інший важливий аспект оптимізації – це використання технологій балансування навантаження.

Балансування навантаження дозволяє розподілити обробку запитів між кількома серверами або компонентами системи, що запобігає перевантаженню окремих серверів і забезпечує рівномірний розподіл ресурсів. Це є особливо актуальним для великих компаній і сервісів, де кількість запитів може змінюватися в залежності від часу доби або сезону.

Джонатан Льюїс у своїй роботі [2] стверджує, що правильне впровадження технологій балансування навантаження допомагає підвищити масштабованість системи без втрати продуктивності, оскільки забезпечується рівномірний розподіл обчислювальних завдань між наявними ресурсами. Це дозволяє уникнути ситуацій, коли один сервер перевантажений, а інші не використовуються на повну потужність, що може призводити до простоїв або зниження загальної продуктивності. Балансування навантаження та розподіл ресурсів, як показано на рисунку 1.3 (крок 3), забезпечують рівномірний розподіл запитів між серверами та ефективне використання статичних ресурсів, таких як CSS, JavaScript та зображення, що запобігає перевантаженню окремих серверів та підвищує загальну стабільність. Тип використовуваної інфраструктури також відіграє важливу роль у виборі методів оптимізації серверної частини інформаційної системи. Традиційні фізичні сервери потребують постійного обслуговування та оновлення апаратного забезпечення для підтримки їхньої продуктивності. Оптимізація серверної частини в цьому випадку часто включає покращення конфігурації апаратного забезпечення, а також налаштування операційних систем для ефективного використання ресурсів.

Однак з розвитком технологій все більше компаній переходять до використання хмарних рішень. Хмарна інфраструктура дозволяє забезпечити гнучке масштабування ресурсів, що є особливо важливим в умовах зростання кількості користувачів або обсягів даних. Хмарні рішення також дозволяють швидко адаптуватися до змін у навантаженні, оскільки не потребують значних інвестицій в апаратне забезпечення.

Крістофер Рейнольдс підкреслює, що використання хмарних технологій дає можливість знизити витрати на фізичне обладнання і забезпечити гнучке управління ресурсами [4]. Завдяки цьому, компанії можуть ефективно масштабувати свої інформаційні системи, не інвестуючи у фізичні сервери, які потребують регулярного оновлення. Хмарні технології також забезпечують швидкий доступ до ресурсів у разі збільшення навантаження, що є важливим для компаній, які працюють з великими обсягами даних або значною кількістю користувачів.

Іншим важливим підходом до оптимізації серверної частини є використання гібридних рішень, що поєднують фізичні та хмарні сервери. Це дозволяє забезпечити високу надійність і гнучкість системи, оскільки критично важливі дані можуть зберігатися на фізичних серверах, а інші завдання можуть виконуватися в хмарі. Такий підхід дозволяє уникнути перевантаження фізичних серверів і забезпечити швидке масштабування в разі необхідності.

Управління ресурсами є ще однією важливою складовою оптимізації серверної частини. Одним із ключових підходів до цього є віртуалізація, яка дозволяє використовувати апаратні ресурси ефективніше. Віртуалізація полягає в тому, що на одному фізичному сервері створюються кілька віртуальних серверів, кожен з яких виконує окремі завдання. Це дозволяє значно зменшити витрати на апаратне забезпечення і підвищити ефективність використання ресурсів [6].

Анна Міллер у своїй праці зазначає, що віртуалізація дозволяє забезпечити гнучкість системи і підвищити ефективність управління ресурсами [4]. Завдяки цьому, компанії можуть швидко адаптувати свою інфраструктуру до змін у навантаженні, не витрачаючи значних коштів на нове обладнання. Віртуалізація також дозволяє ізолювати процеси один від одного, що забезпечує стабільну роботу системи і мінімізує ризики конфліктів між процесами.

Ще одним важливим підходом до управління ресурсами є контейнеризація. Контейнеризація дозволяє ізолювати окремі процеси в межах одного сервера, що знижує ризик взаємного впливу процесів і забезпечує стабільність роботи системи. Контейнери також дозволяють більш ефективно використовувати ресурси серверів, оскільки вони можуть бути налаштовані для виконання конкретних завдань з мінімальними витратами ресурсів [4].

Забезпечення безпеки серверної частини є одним із найважливіших завдань при оптимізації інформаційної системи. У сучасних умовах кіберзагрози становлять серйозну небезпеку для будь-якої компанії, тому захист даних і забезпечення надійності серверів мають бути пріоритетом. Оптимізація безпеки передбачає впровадження різних механізмів захисту даних, таких як шифрування, захист від несанкціонованого доступу і регулярне оновлення безпекових патчів.

Марк Джексон у своїй роботі наголошує, що безпека серверів є не менш важливою, ніж їх продуктивність [5]. Сучасні інформаційні системи повинні бути захищені від зовнішніх загроз, оскільки втрати даних можуть призвести до серйозних наслідків для компанії, включаючи фінансові втрати і втрату довіри з боку клієнтів. Забезпечення безпеки включає не тільки технічні заходи, але й правильну організацію доступу до ресурсів і постійний моніторинг стану системи.

Отже, класифікація методів оптимізації серверної частини інформаційної системи охоплює широкий спектр підходів, які включають оптимізацію обробки запитів, управління ресурсами, використання інфраструктури та забезпечення безпеки. Кожен з цих підходів має свої переваги і може бути адаптований до конкретних потреб системи. Використання сучасних технологій, таких як асинхронна обробка запитів, балансування навантаження, віртуалізація і хмарні рішення, дозволяє забезпечити стабільну роботу системи навіть в умовах високого навантаження.

Забезпечення безпеки також є важливим аспектом оптимізації, оскільки сучасні системи повинні бути захищеними від кіберзагроз. Правильна реалізація методів оптимізації дозволяє забезпечити високу продуктивність, надійність і безперервність роботи інформаційної системи [9].

За результатами аналізу, створено таблицю 1.1, яка класифікує методи оптимізації серверної частини інформаційної системи за різними параметрами.

Таблиця 1.1 – Класифікація методів оптимізації серверної частини інформаційної системи

Критерій	**Метод оптимізації**	**Опис**
1	**2**	**3**
Тип	Продуктивність	Оптимізація коду, використання багатопоточності, кешування даних, оптимізація запитів до бази даних, розподіл навантаження.
	Безпека	Шифрування даних, аутентифікація та авторизація користувачів, моніторинг та аудит безпеки, регулярне оновлення програмного забезпечення.
Рівень застосування	Апаратні методи	Використання сучасного обладнання, потужних серверів, швидкодіючих накопичувачів, високошвидкісних мережевих пристроїв.
	Програмні методи	Використання ефективних алгоритмів, спеціалізованих бібліотек та фреймворків, налаштування параметрів ОС та ПЗ.
	Організаційні методи	Впровадження процесів і процедур, регулярне резервне копіювання, тестування та валідація, навчання персоналу.

Продовження таблиці 1.1

1	2	3
Технічні засоби	Програмне забезпечення для моніторингу та управління ресурсами	Інструменти для виявлення вузьких місць та оптимізації ресурсів у реальному часі.
	Автоматизація процесів	Автоматизація розподілу навантаження, резервного копіювання та оновлення ПЗ, зменшення рутинних завдань, підвищення ефективності.
Вплив на систему	Підвищення продуктивності	Зменшення часу відгуку, збільшення швидкості обробки запитів.
	Забезпечення надійності та відмовостійкості	Уникнення збоїв та втрат даних.
Обмеження та недоліки	Кешування	Можливість застарілих даних при несвоєчасному оновленні кешу.
	Багатопоточність	Проблеми з синхронізацією даних, збільшення складності коду.
	Шифрування	Збільшення часу обробки запитів, що впливає на продуктивність системи.
Комплексний підхід	Комбінація методів	Поєднання кешування, оптимізації запитів та багатопоточності для підвищення продуктивності; впровадження шифрування, аутентифікації та моніторингу для забезпечення безпеки.
Управління змінами	Впровадження нових технологій, регулярне оновлення ПЗ	Адаптація системи до змінних умов експлуатації та вимог бізнесу, забезпечення гнучкості та масштабованості системи
Зворотний зв’язок	Моніторинг результатів та регулярний аналіз	Виявлення проблем, документування змін, аналіз ефективності оптимізації.

1.3 Аналіз сучасних методів оптимізації серверної частини інформаційної системи

Кешування є одним із ключових методів оптимізації серверної частини, який забезпечує значне підвищення швидкості роботи інформаційної системи та зменшення навантаження на ресурси. Суть кешування полягає в збереженні тимчасових копій даних, які найчастіше використовуються, у високошвидкісній пам’яті. Це дозволяє скоротити час доступу до цих даних і зменшити кількість звернень до основної бази даних або файлових систем.

1.3.1 Види кешування

Існує кілька типів кешування, які застосовуються в залежності від специфіки системи та її потреб:

– кешування на стороні клієнта – це підхід, за якого частина даних зберігається на пристрої користувача. Це може бути кеш браузера або інші види локального кешування, які дозволяють прискорити завантаження сторінок або даних при повторних зверненнях до вебресурсу. Такий метод значно зменшує навантаження на сервер, оскільки багато операцій виконується локально на стороні користувача;

– кешування на стороні сервера – тут кешується інформація безпосередньо на сервері, що дозволяє швидко обробляти повторні запити від користувачів. Сервер може зберігати результати обчислень, запитів до бази даних або копії часто запитуваних файлів. Наприклад, вебсервери можуть кешувати HTML-сторінки, зображення або інші статичні ресурси для прискорення їхньої доставки;

– проміжне кешування – цей метод використовує сторонні рішення для зберігання кешованих даних на окремих серверах або спеціалізованих пристроях. Redis або Memcached, дозволяють зберігати великі обсяги даних у

пам’яті, що значно підвищує продуктивність серверів, особливо при виконанні важких запитів до бази даних або обробки великих обсягів інформації.

1.3.2 Переваги кешування

Кешування надає ряд суттєвих переваг, зокрема:

– зменшення часу відповіді. За рахунок збереження даних у високошвидкісній пам’яті (оперативній пам’яті або спеціалізованих кешах), сервер може обробляти запити користувачів набагато швидше. Це особливо важливо для великих інформаційних систем, де швидкість обробки запитів має вирішальне значення;

– зниження навантаження на базу даних. Оскільки багато запитів можуть бути оброблені за рахунок кешу, кількість запитів до бази даних значно скорочується. Це зменшує навантаження на базу даних і підвищує її загальну продуктивність. За словами Джеймса Сміта, кешування дозволяє зменшити кількість звернень до бази даних на 40%–50%, що призводить до значного покращення швидкості обробки запитів [10];

– ефективне використання ресурсів. За рахунок кешування ресурси сервера використовуються більш ефективно, оскільки зменшується навантаження на процесор, пам’ять і диск. Це дозволяє обробляти більше запитів одночасно і знижує ймовірність перевантаження системи.

Балансування навантаження є одним з найважливіших методів оптимізації серверної частини, який дозволяє рівномірно розподілити обробку запитів між кількома серверами або вузлами системи. Це дозволяє уникнути перевантаження окремих компонентів системи і забезпечити стабільну роботу навіть під час пікових навантажень [11].

Суть балансування навантаження полягає в тому, щоб за допомогою спеціалізованого програмного або апаратного забезпечення рівномірно

розподіляти запити між серверами. Це дозволяє кожному серверу обробляти тільки ту кількість запитів, з якою він може ефективно впоратися, не перевантажуючи інші частини системи.

Балансування навантаження може здійснюватися на різних рівнях – на рівні мережевих запитів, вебсерверів або баз даних. Наприклад, вебсервери можуть бути налаштовані на рівномірний розподіл HTTP-запитів між кількома машинами, що дозволяє зменшити затримки і збільшити швидкість обробки даних. Для цього використовуються спеціалізовані балансувальники навантаження, такі як Nginx або HAProxy, які автоматично розподіляють трафік між серверами [12].

1.3.3 Види балансування навантаження

У сучасних інформаційних системах ефективне балансування навантаження відіграє ключову роль у забезпеченні стабільної роботи серверів та оптимізації продуктивності.

Серед популярних алгоритмів для розподілу навантаження варто відзначити:

– алгоритм Round Robin. Цей метод розподіляє запити по черзі між серверами, забезпечуючи рівномірне навантаження. Кожен запит надсилається на наступний сервер у списку, що гарантує, що жоден сервер не буде перевантажений;

– алгоритм Least Connections. Цей підхід використовує розподіл запитів залежно від кількості активних з’єднань на кожному сервері. Сервер з найменшою кількістю активних з’єднань отримує наступний запит, що дозволяє забезпечити більш ефективний розподіл навантаження в реальних умовах роботи;

– географічне балансування. Цей метод використовується для розподілу трафіку залежно від місця розташування користувачів. Сервери,

що знаходяться ближче до користувача, обробляють запити швидше, що дозволяє зменшити затримки та підвищити продуктивність системи.

Джонатан Льюїс підкреслює, що правильне балансування навантаження є необхідним для забезпечення стабільної роботи великих систем, де кількість запитів може змінюватися в залежності від часу доби або сезону [13]. Це дозволяє зменшити ймовірність перевантаження серверів і забезпечити високу надійність роботи системи навіть при різких змінах у навантаженні.

1.3.4 Хмарні обчислення

Застосування хмарних обчислень є одним із найпотужніших інструментів для оптимізації серверної частини сучасних інформаційних систем. Хмарні технології дозволяють гнучко масштабувати ресурси в залежності від поточного навантаження, що є особливо важливим для компаній, які працюють з великими обсягами даних або мають багато користувачів [6].

Однією з головних переваг хмарних рішень є можливість швидко збільшити або зменшити кількість серверів у хмарній інфраструктурі, не витрачаючи час на закупівлю і встановлення фізичного обладнання. Це дозволяє організаціям швидко адаптувати свої системи до змін у попиті, не ризикувати перевантаженням серверів або простоями.

Крістофер Рейнольдс у своєму дослідженні зазначає, що хмарні обчислення забезпечують значне зниження витрат на інфраструктуру і дозволяють компаніям зосередитися на розвитку своїх продуктів, замість витрачати ресурси на підтримку серверів [3]. Хмарні сервіси надають гнучкі можливості для управління ресурсами і забезпечують високу надійність завдяки використанню технологій резервування і захисту даних [7].

Автоматизація є невід’ємною частиною сучасної оптимізації серверної інфраструктури, оскільки дозволяє зменшити ручне втручання в процеси управління ресурсами і значно підвищити продуктивність. Сучасні системи автоматизації дозволяють контролювати і оптимізувати використання ресурсів у реальному часі, автоматично регулюючи їхню кількість залежно від навантаження.

Завдяки автоматизації, такі системи можуть автоматично збільшувати кількість серверів або інші ресурси, коли навантаження зростає, або зменшувати їх, коли навантаження спадає. Це дозволяє знизити витрати на непотрібні ресурси під час періодів низького навантаження, забезпечуючи при цьому максимальну продуктивність під час пікових навантажень [3].

Як зазначає Марк Джексон, автоматизація управління ресурсами дозволяє значно підвищити продуктивність системи і знизити ймовірність людських помилок, які можуть виникати під час ручного налаштування серверів або інших ресурсів [4]. Сучасні системи автоматизації можуть включати засоби для автоматичного балансування навантаження, моніторингу продуктивності, а також управління інфраструктурою в реальному часі.

Моніторинг серверної частини та аналіз продуктивності є важливими компонентами оптимізації, оскільки вони дозволяють виявляти проблеми на ранніх стадіях і своєчасно впроваджувати коригувальні заходи. Моніторинг дозволяє відстежувати основні показники роботи серверів, такі як використання процесорів, пам’яті, мережевих ресурсів і дискових сховищ. Завдяки моніторингу можна своєчасно виявляти перевантаження системи або потенційні вузькі місця, що впливають на продуктивність (рис. 1.4).

Системи моніторингу, такі як Zabbix або Prometheus, надають детальну інформацію про стан системи в режимі реального часу і можуть бути налаштовані для автоматичного сповіщення адміністраторів у разі виникнення проблем. Це дозволяє мінімізувати простої і швидко реагувати на зміни в роботі системи [8].

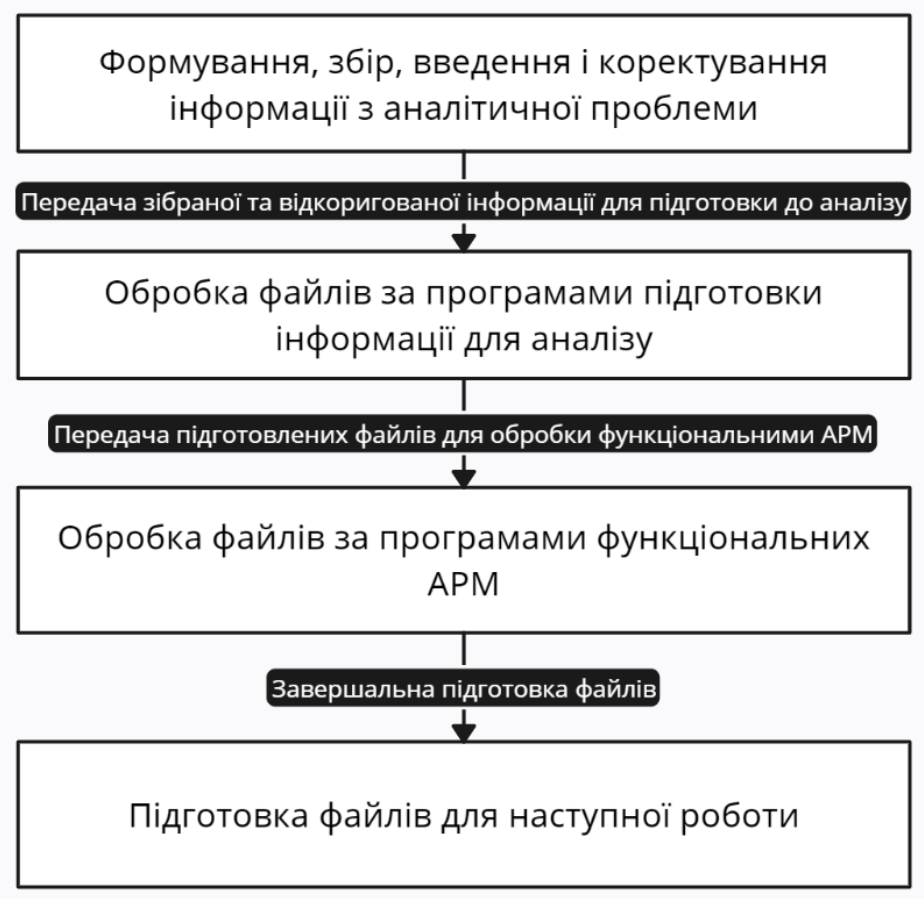

Рисунок 1.4 – Процеси обробки і підготовки інформації для аналізу

Моніторинг продуктивності також дозволяє аналізувати ефективність впроваджених методів оптимізації та оцінювати їхній вплив на роботу системи. Це важливо для постійного вдосконалення і адаптації інфраструктури до змінних умов роботи.

Сучасні методи оптимізації серверної частини інформаційної системи включають використання контейнеризації, віртуалізації, кешування, балансування навантаження, хмарних обчислень, автоматизації управління ресурсами та моніторингу продуктивності. Кожен з цих методів має свої переваги і може бути застосований в залежності від специфіки системи, її архітектури та вимог до продуктивності.

Використання цих методів дозволяє забезпечити стабільну, надійну і високо продуктивну роботу інформаційної системи навіть за умов високого навантаження та змінних умов експлуатації [3].

1.4 Аналіз літературних джерел щодо апробації результатів застосування методів оптимізації серверної частини інформаційної системи

У сучасній науковій літературі представлено багато досліджень, присвячених апробації різноманітних методів оптимізації серверної частини інформаційних систем. Важливість таких експериментів полягає в тому, що вони дозволяють не лише теоретично обґрунтувати використання певних технологій, але й на практиці продемонструвати їхню ефективність, доцільність та вплив на загальну продуктивність систем. Нижче наведено аналіз ключових літературних джерел, які демонструють апробацію результатів застосування різних методів оптимізації серверної частини.

Дослідження, представлене в джерелі [2], зосереджено на використанні хмарних технологій у поєднанні з методами балансування навантаження для забезпечення безперервної роботи інформаційних систем. Автори роботи підкреслюють, що поєднання хмарних обчислень з динамічним балансуванням трафіку дозволило одній з роздрібних компаній зменшити час відповіді на запити користувачів і забезпечити стабільну роботу свого веб-сайту навіть під час пікових продажів. Апробація цього підходу показала, що система змогла обробляти на 50% більше запитів без затримок, а компанія змогла знизити витрати на підтримку фізичної інфраструктури на 40%.

У джерелі [3], наприклад, розглянуто використання контейнеризації для оптимізації обчислювальних ресурсів у великій IT-системі. Дослідники підкреслюють, що застосування контейнерів Docker дозволило значно знизити витрати на апаратне забезпечення, оскільки кожен сервер міг працювати з кількома ізольованими контейнерами одночасно. Це призвело до збільшення щільності ресурсів і підвищення ефективності використання процесорів та пам’яті. Апробація цього підходу в реальних умовах показала, що система змогла обробляти на 25% більше запитів без додаткових витрат на фізичне обладнання. Крім того, автори зазначають, що контейнеризація

дозволила знизити ризики конфліктів між процесами, що сприяло підвищенню стабільності роботи системи.

У джерелі [5] розглянуто методи балансування навантаження та їх вплив на продуктивність інформаційних систем під час пікових навантажень. Дослідження показує, що використання алгоритму Round Robin для рівномірного розподілу трафіку між серверами дозволило зменшити час відповіді на запити на 15%, що суттєво вплинуло на загальну ефективність системи. Крім того, у дослідженні було зазначено, що балансування навантаження, яке базується на географічному принципі, дало змогу зменшити затримки для користувачів з різних регіонів, оскільки їхні запити оброблялися найближчими до них серверами.

Інше дослідження, представлене у джерелі [8], присвячене використанню хмарних технологій для оптимізації серверної частини інформаційної системи. Автори роботи вивчали, як використання хмарної інфраструктури допомогло одній із фінансових компаній адаптувати свою IT-систему до різких змін у кількості користувачів під час фінансових звітних періодів. Завдяки хмарним обчисленням компанія змогла динамічно збільшувати кількість серверів у хмарі під час пікових навантажень, а після їхнього зниження – скорочувати кількість серверів, що дозволило суттєво знизити витрати на утримання фізичної інфраструктури. Результати апробації показали, що цей підхід дозволив на 40% знизити експлуатаційні витрати і одночасно підвищити швидкість обробки фінансових транзакцій на 30%.

Дослідження, представлене у джерелі [9], описує впровадження технології автоматичного масштабування серверної інфраструктури у великій інтернет-компанії. Компанія стикалася з різкими змінами у кількості користувачів, особливо під час акцій або рекламних кампаній. Використання автоматизованого масштабування дозволило компанії динамічно збільшувати або зменшувати кількість серверів у залежності від поточного навантаження. Результати апробації показали, що цей підхід дозволив компанії зменшити

витрати на утримання інфраструктури на 35% і значно покращити стабільність роботи сервісу під час пікових навантажень.

У джерелі [10] розглядаються результати досліджень щодо впровадження віртуалізації для оптимізації серверної частини. Автори роботи дослідили, як застосування технологій віртуалізації допомогло підвищити ефективність управління ресурсами в одній з великих медичних установ. Використання віртуальних серверів дозволило установі запускати на одному фізичному сервері декілька віртуальних машин, що дало можливість виконувати різні завдання одночасно без втрат у продуктивності. Апробація цього методу в умовах реальної експлуатації показала, що медична установа змогла підвищити ефективність використання своїх IT-ресурсів на 35%, зменшити витрати на обладнання і поліпшити швидкість доступу до медичних даних пацієнтів.

У джерелі [12] аналізується впровадження автоматизованих систем моніторингу і управління ресурсами для оптимізації серверної інфраструктури. Дослідження було проведене на прикладі великої телекомунікаційної компанії, яка мала значне навантаження на свої сервери через постійно зростаючу кількість клієнтів. Застосування автоматизованих систем моніторингу дозволило компанії відслідковувати стан своїх серверів у режимі реального часу і автоматично регулювати кількість ресурсів залежно від поточного навантаження. Це дозволило уникнути простоїв і підвищити стабільність роботи системи. Апробація результатів показала, що впровадження таких систем дозволило на 20% зменшити витрати на підтримку інфраструктури та на 25% збільшити загальну продуктивність серверів.

У джерелі [14] розглянуто методи оптимізації серверної частини за допомогою кешування даних. Автори досліджували вплив кешування на продуктивність вебзастосунками під час високих навантажень. Використання технологій кешування, таких як Redis, дозволило значно знизити час відповіді на запити користувачів. Апробація цього підходу показала, що

вебзастосунки змогли знизити час завантаження сторінок на 50%, що суттєво підвищило якість обслуговування користувачів. Крім того, технології кешування дозволили знизити навантаження на базу даних на 30%, що позитивно вплинуло на загальну стабільність роботи системи.

У джерелі [15] детально описано результати впровадження хмарної інфраструктури для однієї з банківських установ. Завдяки переходу на хмарні обчислення, банк зміг значно покращити продуктивність своїх серверів, зменшити час обробки фінансових операцій і підвищити надійність системи. Апробація результатів показала, що банк зміг збільшити кількість одночасно оброблюваних транзакцій на 25%, а також знизити витрати на утримання фізичної інфраструктури на 30%. Крім того, хмарна інфраструктура забезпечила надійний захист даних і дозволила автоматично створювати резервні копії, що значно підвищило безпеку системи.

Таким чином, аналіз літературних джерел показує, що сучасні методи оптимізації серверної частини інформаційних систем, такі як контейнеризація, хмарні обчислення, віртуалізація, кешування, автоматизація управління ресурсами та балансування навантаження, активно застосовуються на практиці і демонструють високу ефективність у покращенні продуктивності систем, зниженні витрат та підвищенні надійності. Апробація цих методів у різних галузях підтверджує їхню доцільність та дозволяє адаптувати їх до потреб конкретних організацій та інформаційних систем.

1.5 Постановка задачі дослідження

Актуальність дослідження методів оптимізації серверної частини інформаційних систем обумовлена кількома важливими аспектами. Збільшення обсягу даних і кількості користувачів, які взаємодіють із системою, ставить завдання підвищення продуктивності серверів. Одночасно

з цим, зростає значення захисту даних, що набуває особливої актуальності через постійні кіберзагрози. Крім того, технологічні інновації відкривають нові можливості для вдосконалення інформаційних систем, що вимагає постійного оновлення і перегляду методів їх оптимізації.

Об’єктом дослідження є серверна частина інформаційної системи, яка забезпечує оброблення та збереження даних, а також взаємодію з клієнтськими застосунками.

Метою дослідження є порівняння методів оптимізації серверної частини інформаційної системи для підвищення її продуктивності та безпеки даних, а також виявлення найефективніших з них.

Для досягнення поставленої мети необхідно вирішити такі завдання:

– провести аналіз літературних джерел та існуючих досліджень щодо методів оптимізації серверної частини ІС;

– визначити найбільш ефективні та актуальні підходи до оптимізації серверної частини ІС;

– вибрати інструментальні засоби для реалізації обраних методів оптимізації;

– реалізувати обрані методи оптимізації на практиці та оцінити їх ефективність у контексті конкретної предметної області;

– провести порівняльний аналіз досліджених методів оптимізації серверної частини ІС;

– визначити перспективи подальшої роботи та можливості вдосконалення існуючих методів оптимізації.

Методи дослідження включають аналіз літературних джерел, експериментальну розробку та тестування програмних рішень, а також порівняльний аналіз ефективності різних методів оптимізації. Вибір інструментальних засобів для реалізації обраних методів включає використання сучасних мов програмування, бібліотек та фреймворків, а також апаратних рішень, що забезпечують високу продуктивність і безпеку серверної частини ІС.

2 ОСОБЛИВОСТІ ВИБРАНИХ МЕТОДІВ ОПТИМІЗАЦІЇ СЕРВЕРНОЇ ЧАСТИНИ ІНФОРМАЦІЙНОЇ СИСТЕМИ ДЛЯ ПІДВИЩЕННЯ ЇЇ ПРОДУКТИВНОСТІ ТА БЕЗПЕКИ ДАНИХ

2.1 Методи оптимізації продуктивності серверної частини інформаційної системи

2.1.1 Кешування даних

Кешування є одним із ключових методів оптимізації продуктивності інформаційних систем. У контексті серверної частини інформаційних систем, кешування допомагає зменшити навантаження на серверні ресурси шляхом тимчасового зберігання часто використовуваних даних у швидкодіючій пам’яті. Це значно скорочує час доступу до даних і підвищує загальну продуктивність системи. Основна мета кешування полягає в зниженні часу обробки запитів до сервера та покращенні часу відгуку системи. Цей підхід особливо ефективний у системах, що обробляють великий обсяг однотипних запитів.

Кешування базується на принципі тимчасового зберігання даних, доступ до яких здійснюється часто. Основною метою є зменшення необхідності повторного отримання тих самих даних із первинного джерела, що може бути повільним або ресурсозатратним процесом. За допомогою кеша дані зберігаються на рівні оперативної пам’яті або інших швидкодійних пристроях, що дозволяє суттєво скоротити час доступу [16].

Системи кешування працюють на базі таких основних принципів:

– частота використання даних: найчастіше запитувані дані зберігаються в кеші для швидкого доступу;

– тимчасове зберігання: кеш зберігає дані на короткий термін, після чого вони можуть бути оновлені або видалені;

– обмеженість ресурсів: кешування використовує обмежений обсяг пам’яті, тому необхідно управляти ним ефективно.

Існує кілька типів кешування, які застосовуються для оптимізації продуктивності серверної частини інформаційної системи:

– кешування на рівні застосунку: цей метод використовується для зберігання результатів обчислень або отриманих даних безпосередньо в пам’яті застосунка. Це дозволяє уникнути повторних обчислень або запитів до бази даних;

– кешування на рівні бази даних: цей тип кешування передбачає зберігання часто запитуваних даних на рівні бази даних, що дозволяє зменшити кількість запитів до основної бази та прискорити доступ до даних;

– кешування в оперативній пам’яті: цей метод передбачає зберігання даних у швидкодійній оперативній пам’яті, що дозволяє отримувати до них доступ із мінімальними затримками;

– кешування зворотного проміжного сервера: цей тип кешування застосовується для зберігання відповідей на запити користувачів на рівні проміжних серверів, що дозволяє розвантажити основний сервер і прискорити відповідь для подібних запитів.

Кешування дозволяє значно підвищити продуктивність серверної частини інформаційної системи шляхом скорочення часу виконання запитів та зменшення навантаження на центральний процесор і базу даних. Це особливо важливо для високонавантажених систем, які обробляють значну кількість запитів у режимі реального часу.

Завдяки кешуванню, сервер може обробляти більшу кількість запитів одночасно, не втрачаючи продуктивності [17].

Переваги кешування:

– зменшення навантаження на серверні ресурси: знижуючи кількість запитів до основного джерела даних, кешування дозволяє зменшити навантаження на процесор і базу даних;

– зменшення часу відгуку: оскільки кешовані дані доступні швидше, система може оперативніше відповідати на запити;

– покращення масштабованості: системи, що використовують кешування, можуть ефективніше обробляти збільшений обсяг трафіку, що сприяє кращій масштабованості [18].

Управління кешем є важливою складовою оптимізації продуктивності серверної частини інформаційної системи. Вибір правильної стратегії кешування може значно вплинути на ефективність системи:

– час життя кешованих даних: визначає період, протягом якого дані зберігаються в кеші. По завершенні цього періоду кешовані дані видаляються, і при наступному запиті до джерела отримуються нові дані;

– пріоритети кешування: деякі дані можуть мати вищий пріоритет для кешування, що дозволяє їм залишатися в кеші довше, ніж інші дані;

– стратегії заміщення кеша: коли кеш заповнений, необхідно вирішити, які дані варто видалити для звільнення місця для нових.

Серед основних стратегій заміщення кеша є:

– LRU (Least Recently Used): видаляються найменш використані дані;

– FIFO (First In First Out): видаляються дані, що надійшли раніше;

– LFU (Least Frequently Used): видаляються дані, які використовувалися найрідше [19].

Сучасні серверні інформаційні системи активно використовують кешування для підвищення своєї продуктивності. Наприклад, вебзастосунки використовують кешування для зберігання результатів обчислень, запитів до бази даних, а також статичних ресурсів, таких як зображення, стилі та сценарії. Інтеграція кешування на рівні вебсерверу, бази даних та мережевих проміжних серверів дозволяє суттєво знизити затримки при обробці запитів.

Приклади систем кешування:

– Redis: це один із найбільш популярних інструментів для кешування в оперативній пам’яті, який використовується для зберігання структурованих даних, таких як рядки, хеші, списки тощо;

– Memcached: простий, але потужний інструмент для кешування в оперативній пам’яті, використовується для зберігання даних у вигляді пар ключ-значення.

Кешування – це ефективний спосіб підвищення продуктивності серверних систем, оскільки воно допомагає мінімізувати навантаження на серверні ресурси і зменшити час виконання запитів. Правильний вибір стратегії кешування, управління строком життя кешованих даних, і своєчасне очищення кешу можуть значно покращити масштабованість та стабільність системи, забезпечуючи швидку і надійну роботу для користувачів [20].

2.1.2 Асиметрична багатопоточність даних

Асиметрична багатопоточність є одним із методів обробки даних, який дозволяє підвищити ефективність виконання задач у багатоядерних або багатопроцесорних системах. На відміну від симетричної багатопоточності, де всі потоки виконуються рівноправно і мають однакові права на доступ до ресурсів, асиметрична багатопоточність передбачає нерівномірний розподіл навантаження між потоками. Це дозволяє адаптуватися до специфічних вимог кожного потоку, оптимізувати використання ресурсів і зменшити загальний час обробки задач [21].

Основна ідея полягає в тому, що один потік може бути відповідальним за управління ресурсами або координацію дій інших потоків, тоді як інші виконують специфічні обчислювальні операції. Такий підхід є особливо корисним для програмних систем, які працюють з великим обсягом даних або вимагають швидкої обробки запитів у реальному часі.

У традиційних моделях багатопоточності кожен потік має однаковий доступ до ресурсів, таких як процесорний час, пам’ять та інші системні ресурси. Однак у деяких випадках це може призводити до неефективного

використання ресурсів, особливо коли окремі потоки вимагають різних обсягів ресурсів або виконують задачі з різним рівнем складності.

Асиметрична багатопоточність передбачає виділення одного або декількох потоків, які виконують керівні функції, такі як управління ресурсами або синхронізація дій інших потоків. Це дозволяє ефективніше використовувати ресурси та зменшити накладні витрати, пов’язані з синхронізацією потоків [22].

Ключові принципи асиметричної багатопоточності:

– нерівномірний розподіл навантаження: деякі потоки можуть бути більш завантаженими, ніж інші, залежно від характеру виконуваних задач;

– централізоване управління: один потік може бути виділений для координації роботи інших потоків, що дозволяє уникнути конфліктів при доступі до ресурсів;

– оптимізація використання ресурсів: кожен потік отримує ресурси відповідно до своїх потреб, що дозволяє уникнути простоїв або надмірного використання ресурсів.

Асиметрична багатопоточність є особливо актуальною для серверних інформаційних систем, які повинні обробляти великий обсяг одночасних запитів. Наприклад, в системах обробки запитів до баз даних або обчислювальних платформах для обробки великих даних, нерівномірний розподіл навантаження між потоками може забезпечити кращу продуктивність.

Одним із прикладів застосування асиметричної багатопоточності є системи, що використовують технології типу master-slave. У такій архітектурі основний потік (master) відповідає за управління і розподіл завдань між підлеглими потоками (slaves), які виконують специфічні обчислювальні завдання. Це дозволяє основному потоку сфокусуватися на координації, тоді як підлеглі потоки можуть оптимізовано використовувати доступні ресурси для виконання конкретних задач.

Існує кілька підходів до реалізації асиметричної багатопоточності, кожен із яких має свої переваги і недоліки. Вибір підходу залежить від конкретних вимог системи і характеру оброблюваних даних:

– стратегія з виділеними потоками керування: один або декілька потоків виділяються для управління ресурсами та координації роботи інших потоків. Це дозволяє зменшити накладні витрати на синхронізацію, оскільки всі рішення щодо розподілу ресурсів приймаються централізовано;

– обчислювальні потоки з нерівномірним навантаженням: у цій стратегії різні потоки отримують різний обсяг завдань залежно від їхньої продуктивності або спеціалізації. Це дозволяє уникнути ситуацій, коли одні потоки перевантажені, а інші простоюють;

– адаптивна асиметрична багатопоточність: у цій стратегії навантаження між потоками може змінюватися динамічно, залежно від поточного стану системи та характеристик вхідних даних. Наприклад, якщо один потік виявляється перевантаженим, його задачі можуть бути перерозподілені між іншими потоками [23].

Попри значні переваги, асиметрична багатопоточність також має свої виклики та обмеження, які слід враховувати при її впровадженні:

– складність реалізації: управління потоками і ресурсами в асиметричній моделі є більш складним завданням, ніж у симетричній багатопоточності. Це потребує ретельного планування та тестування для забезпечення коректної роботи системи;

– можливість виникнення «вузьких місць»: оскільки один або декілька потоків можуть бути відповідальними за координацію, існує ризик виникнення ситуацій, коли ці потоки стають вузьким місцем у системі, якщо вони перевантажені або не справляються з обсягом задач;

– балансування навантаження: один із ключових викликів асиметричної багатопоточності полягає в тому, щоб правильно балансувати навантаження між потоками. Якщо навантаження не буде розподілено ефективно, це може призвести до зниження продуктивності або навіть до простоїв.

Асиметрична багатопоточність є потужним інструментом для оптимізації продуктивності систем, що працюють із великими обсягами даних або мають специфічні вимоги до обробки запитів. Використання цього підходу дозволяє ефективніше управляти ресурсами, зменшити час обробки задач і покращити загальну продуктивність системи. Однак, для успішного впровадження асиметричної багатопоточності необхідно враховувати виклики, пов’язані зі складністю реалізації та балансуванням навантаження між потоками.

Таким чином, асиметрична багатопоточність знаходить своє застосування в сучасних інформаційних системах, що вимагають високої продуктивності та швидкої обробки даних, забезпечуючи значне підвищення ефективності роботи серверної частини системи [24].

2.1.3 Оптимізація запитів до бази даних

Оптимізація запитів до бази даних є одним із найважливіших аспектів забезпечення ефективної роботи інформаційних систем. Незалежно від того, наскільки потужним є апаратне забезпечення, неефективні запити можуть призвести до значних затримок у роботі системи, надмірного використання ресурсів і, як наслідок, зниження продуктивності. Основна мета оптимізації запитів полягає в тому, щоб мінімізувати час виконання операцій і зменшити навантаження на базу даних шляхом вдосконалення структур і методів взаємодії з нею.

Будь-яка реляційна база даних зберігає великі обсяги інформації у вигляді таблиць. Для того, щоб отримати доступ до цієї інформації, застосовуються SQL-запити. Проте, не всі запити однаково ефективні. Від способу написання запиту, вибору типу даних та архітектури бази залежить швидкість його виконання. Коли обсяг даних у базі невеликий, незначні неефективності можуть залишитися непоміченими. Однак у міру зростання

бази навіть незначні недоліки у запитах можуть значно вплинути на продуктивність [25].

Оптимізація SQL-запитів починається з розуміння того, як база даних виконує кожен запит і як вона працює з індексами та структурами даних. Це включає роботу з планами виконання запитів, розуміння механізмів кешування та правильного використання індексів.

Одним із найпотужніших інструментів для прискорення SQL-запитів є індекси. Індекси дозволяють базі даних швидко знаходити потрібні записи, зменшуючи необхідність повного сканування таблиці. Це подібно до того, як індекс у книзі дозволяє швидко знайти сторінку з потрібною інформацією без необхідності перегортати кожну сторінку.

Проте неправильне використання індексів або їх надлишок може негативно вплинути на продуктивність. Індекси потребують додаткового місця для зберігання та уповільнюють операції запису й оновлення, оскільки базі даних доводиться оновлювати індекси щоразу, коли змінюються дані в таблиці. Тому важливо створювати індекси для тих стовпців, які часто використовуються в запитах, особливо в умовах WHERE та при JOIN-операціях.

Агрегаційні функції, такі як COUNT, SUM, AVG, часто використовуються для отримання підсумкової інформації з таблиць. Однак їх неправильне використання може суттєво знизити швидкість виконання запитів. Наприклад, застосування агрегаційних функцій у вкладених запитах, що повторюються для кожного запису основного запиту, може значно збільшити час виконання.

Для оптимізації агрегаційних запитів можна використовувати кілька стратегій. По-перше, агрегаційні функції слід виконувати один раз для всієї вибірки даних, а не окремо для кожного запису. По-друге, для поліпшення продуктивності можна використовувати попередньо обчислені значення, зберігаючи їх у додаткових колонках або спеціальних таблицях. Це особливо

корисно для статистики або часто використовуваних даних, які не змінюються регулярно.

JOIN-операції дозволяють об’єднувати дані з кількох таблиць у рамках одного запиту. Однак їх використання може стати причиною серйозних затримок, якщо об’єднуються великі таблиці або неправильно використовуються індекси.

Для оптимізації JOIN-операцій слід уникати об’єднання великих таблиць без явної необхідності. Крім того, варто переконатися, що стовпці, за якими виконується об’єднання, індексовані. У деяких випадках краще розділити складний запит із JOIN на кілька простіших, а потім з’єднати їх результати на рівні застосунка.

Ще одним ефективним методом оптимізації є кешування результатів запитів. Кешування дозволяє зберігати результати складних або часто використовуваних запитів у швидкодоступній пам’яті, що значно зменшує час на повторну обробку тих самих даних.

Кешування особливо корисне для даних, які рідко змінюються, таких як списки товарів або категорій у вебмагазинах. Використання механізмів кешування, таких як Redis або Memcached, дозволяє знизити навантаження на базу даних і значно покращити час відповіді на запити.

Для виявлення проблемних запитів слід регулярно використовувати інструменти аналізу баз даних. Більшість сучасних реляційних баз даних, таких як MySQL або PostgreSQL, мають вбудовані інструменти для аналізу продуктивності запитів. Наприклад, MySQL надає команду EXPLAIN, яка показує план виконання запиту і дозволяє визначити, які індекси використовуються, скільки рядків обробляється, і які операції можуть бути оптимізовані.

Ці інструменти допомагають виявити так звані «вузькі місця» у запитах, місця, де продуктивність падає через неефективне використання ресурсів. Регулярний моніторинг запитів та їх планів виконання дозволяє

підтримувати базу даних у належному стані й запобігати критичним затримкам [26].

Часто не всі дані, які передаються із бази даних, необхідні для подальшої обробки на рівні застосунку. Тому важливо використовувати тільки ті стовпці та записи, які дійсно потрібні. Для цього варто уникати використання SELECT*, яке витягує всі колонки з таблиці, навіть якщо потрібні лише кілька з них. Натомість, слід явним чином вказувати потрібні стовпці в запиті.

Мінімізація обсягу даних, що передаються, зменшує як навантаження на мережу, так і час обробки даних на рівні застосунка, що в цілому позитивно впливає на продуктивність системи.

Оптимізація запитів до бази даних є ключовим елементом забезпечення ефективної роботи сучасних інформаційних систем. Правильне використання індексів, уникнення надмірного використання агрегаційних функцій і вкладених запитів, оптимізація JOIN-операцій, кешування результатів і мінімізація обсягу даних, що передаються, все це сприяє зниженню навантаження на базу даних і покращенню часу відповіді системи.

Однак, оптимізація запитів – це не одноразове завдання. Воно потребує регулярного моніторингу та аналізу продуктивності системи, особливо у випадках, коли обсяг даних у базі значно зростає або змінюється характер запитів.

Логіка оптимізації запитів до бази даних важлива, оскільки більшість програмістів тестують свої програми з невеликою кількістю записів у таблицях, а проблеми у власника сайту починаються пізніше, коли він наповнить товарні каталоги.

Наприклад, на рисунку 2.1 є запит, якщо прибрати з цього запиту умову LIMIT, то він поверне 2907 записів. Відповідно, вкладений запит SELECT буде виконаний 2907 разів.

```
SELECT
   p.product_id,
  (SELECT AVG(rating) AS total FROM
mc_review r1 WHERE r1.product_id =
p.product_id AND r1.STATUS ='1' GROUP
BY r1.product_id) AS rating
FROM mc_product p
LEFT JOIN mc_product_description pd
ON (p.product_id = pd.product_id)
LEFT JOIN mc_product_to_store p2s ON
(p.product_id = p2s.product_id)
WHERE
   pd.language_id = '2' AND
   p.STATUS = '1' AND
   p.date_available <= NOW() AND
   p2s.store_id = '0' AND
   p.product_id IN (SELECT
pt.product_id FROM mc_product_tag
pt WHERE pt.language_id = '2' AND
LOWER(pt.tag) LIKE '%роксолана%')
ORDER BY rating ASC
LIMIT 0,20
```

Рисунок 2.1 – Вибірка із фільтрацією за тегом, мовою та рейтингом

Якщо цю частину запиту винести в окремий запит, це зменшить навантаження на базу даних у 2907/20 = 145 разів. Хоча, судячи з назви запиту, можна зробити висновок, що він намагається підраховувати статистику товарів кожного разу, коли відвідувач заходить на сайт. Цю статистику можна було б перераховувати, наприклад, раз на добу або, ще краще, при додаванні відгуку до товару, і зберігати в окремій колонці таблиці mc_product, що дозволить позбутися цього вкладеного запиту [27].

В умові WHERE бачимо вкладений запит, який виконується у виразі IN. Якби було вказано не вкладений запит, а просто статичні значення, наприклад, IN (121, 1235, 43554), то MySQL використав би індекс і виконав запит швидко. Але з вкладеними запитами все відбувається зовсім інакше – MySQL виконує їх без використання індексів, точніше, таким чином: FIN_IN_SET(p.product_id, ‘121,1235,43554’). У таких випадках краще спочатку виконати запит окремо, а потім підставляти результат його виконання у вираз IN.

2.2 Методи безпеки даних серверної частини інформаційної системи

2.2.1 Шифрування даних

Шифрування даних є одним із найважливіших методів забезпечення безпеки серверної частини інформаційної системи. У сучасному світі, де обсяг переданих та збережених даних постійно зростає, захист цих даних від несанкціонованого доступу стає пріоритетним завданням. Шифрування допомагає захистити конфіденційність інформації, перетворюючи її на незрозумілу форму, яку можна розшифрувати лише за наявності відповідного ключа.

Основна ідея шифрування полягає в тому, що навіть у разі компрометації системи або перехоплення даних сторонніми особами, ці дані залишаться незрозумілими та не будуть доступні без ключа розшифрування. Важливим аспектом цього процесу є вибір надійного алгоритму шифрування та забезпечення належного управління ключами [28].

Процес шифрування перетворює відкриті дані, які можуть бути прочитані, на зашифровані дані, які не можна зрозуміти без спеціальних інструментів. Це досягається за допомогою математичних алгоритмів, які використовують ключі для кодування та декодування інформації. Чим довший і складніший ключ, тим складніше його підібрати, що забезпечує вищий рівень захисту даних.

Шифрування може бути як симетричним, так і асиметричним. У випадку симетричного шифрування для кодування та декодування інформації використовується один і той самий ключ. Це швидкий метод, але він вимагає безпечної передачі ключа між відправником і одержувачем. Асиметричне шифрування, навпаки, використовує два ключі: відкритий і закритий. Відкритий ключ використовується для шифрування даних, а закритий – для їх розшифрування. Це робить асиметричне шифрування більш безпечним, але повільнішим порівняно з симетричним.

Для серверної частини інформаційної системи шифрування є критично важливим, оскільки сервери часто зберігають конфіденційні дані, такі як особиста інформація користувачів, фінансові записи або корпоративні секрети. Сервери також можуть бути об’єктом атак, тому важливо забезпечити захист не тільки від зовнішніх загроз, але й від внутрішніх витоків інформації [29].

Наприклад, дані можуть бути зашифровані як під час передачі між клієнтом і сервером (шифрування на транспортному рівні, TLS), так і під час їх зберігання на сервері (шифрування на рівні збереження). Це дозволяє забезпечити цілісність і конфіденційність інформації навіть у разі компрометації серверної інфраструктури.

Попри очевидні переваги шифрування, існують також певні виклики, з якими стикаються розробники та адміністратори систем. Основним є правильне управління ключами. Якщо ключі для розшифрування потраплять у руки зловмисників, шифрування стає неефективним. Тому надзвичайно важливо використовувати надійні методи зберігання та передачі ключів.

Іншим викликом є продуктивність системи. Шифрування і розшифрування можуть вимагати значних обчислювальних ресурсів, особливо при роботі з великими обсягами даних. Тому розробникам слід ретельно планувати процеси шифрування, щоб забезпечити баланс між безпекою і швидкістю роботи системи.

Шифрування даних є одним із найефективніших методів захисту інформації на серверній частині інформаційної системи. Воно гарантує конфіденційність та цілісність даних навіть у випадку їхньої компрометації. Для забезпечення належного рівня безпеки необхідно приділяти особливу увагу управлінню ключами та підтримці балансу між безпекою і продуктивністю системи [30].

Значна частина витоків інформації відбувається під час пересилання файлів. Для підвищення рівня безпеки та зниження ризиків ці файли можна шифрувати перед відправленням.

Шифрування – це процес, за допомогою якого дані кодуються для приховування інформації. У результаті цього процесу дані перетворюються так, що особа, яка не має спеціального ключа для розшифрування, бачить лише набір цифр або пошкоджений файл (рис. 2.2).

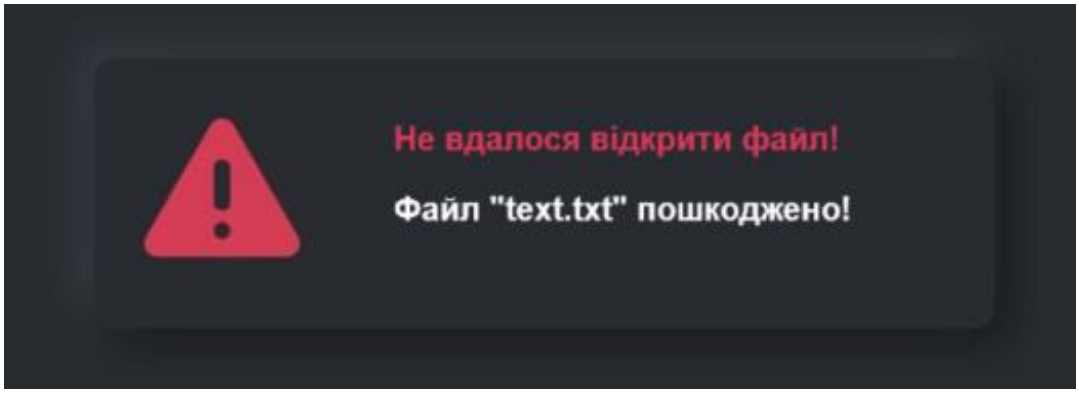

Рисунок 2.2 – Пошкоджений документ, який неможливо відкрити

Як це виглядає? Фраза «Незламна Україна – вільна нація» після шифрування виглядатиме для сторонньої особи як «z3xT@9!mNp42#dlc7%Vw8qS1#» (рис 2.3). Однак, якщо отримувач має спеціальний ключ або пароль, за допомогою якого можна розшифрувати цей набір символів і цифр, він зможе відновити оригінальне повідомлення та прочитати його.

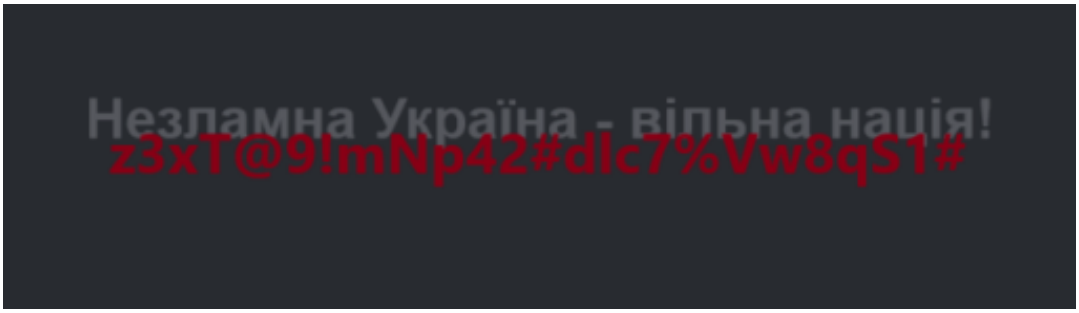

Рисунок 2.3 – Шифрування фрази «Незламна Україна – вільна нація»

Шифрування дозволяє захистити інформацію від несанкціонованого доступу, гарантуючи, що лише уповноважені особи матимуть доступ до змісту повідомлення [31].

Шифр вигадувати не потрібно, адже існує багато програмних рішень, які надають можливість шифрувати файли та диски з використанням різних алгоритмів, ось кілька з них:

– VeraCrypt – програмне забезпечення для шифрування файлів і дисків. Воно зручне у використанні та дозволяє обирати алгоритми шифрування для конкретного елемента;

– PGP Desktop – програма для шифрування файлів і каталогів, яка також захищає поштові повідомлення, локальну мережу та може створювати зашифровані образи дисків;

– Folder Lock – програма, що може приховувати папки, шифрувати файли на зовнішніх носіях та зберігати паролі в захищеному сховищі. Вона також дозволяє повністю видаляти документи і очищати вільне місце на дисках;

– Dekart Private Disk – інструмент для створення виключно зашифрованих образів дисків, який можна налаштовувати з автозапуском програм під час монтування.

Перед вибором програмного забезпечення варто ознайомитися з його функціями та можливостями, а також читати відгуки користувачів. Завантажуйте лише з офіційних вебсайтів компаній-розробників.

Також в налаштуваннях операційних систем можна увімкнути шифрування файлів, що зберігаються у внутрішній пам’яті пристрою. Наприклад, у Windows це можна зробити за допомогою інструмента EFS (Encrypting File System), який дозволяє шифрувати окремі файли та папки. Це корисно, якщо до комп’ютера мають доступ кілька користувачів [32].

Для увімкнення шифрування через EFS необхідно виконати такі дії:

– натиснути правою кнопкою миші на потрібному документі чи папці й обрати пункт Властивості;

– у вкладці Загальні натиснути Додатково;

– встановити прапорець біля пункту Шифрувати вміст для захисту даних;

– натиснути ОК та Застосувати.

Якщо у папці є декілька файлів, система запропонує вибір: шифрувати всі документи або лише вибраний файл.

Шифрування на рівні операційної системи є додатковим засобом захисту від несанкціонованого доступу, особливо коли пристрій використовують кілька осіб або існує ризик втрати або крадіжки даних.

2.2.2 Аутентифікація та авторизація даних

Аутентифікація та авторизація є основними елементами забезпечення безпеки даних у серверній частині інформаційних систем. Вони гарантують, що доступ до даних мають лише уповноважені користувачі та програми, запобігаючи несанкціонованому доступу і зловживанню ресурсами системи. Хоча обидва ці процеси мають спільну мету – захист даних, вони реалізуються на різних етапах взаємодії користувача з системою та виконують різні функції.

Аутентифікація – це процес перевірки особи користувача або системи, який полягає в підтвердженні їхньої ідентичності. Основна мета аутентифікації полягає в тому, щоб забезпечити, що особа, яка запитує доступ до системи або ресурсів, є тим, за кого вона себе видає. Це може бути досягнуто за допомогою різних методів, таких як паролі, біометричні дані, одноразові паролі (OTP) або криптографічні сертифікати.

Найбільш поширеним методом є парольна аутентифікація, яка передбачає введення користувачем унікального ідентифікатора (логіну) та секретного пароля. Однак цей метод має свої обмеження, оскільки паролі можуть бути викрадені або зламані. Для підвищення рівня безпеки часто застосовується багатофакторна аутентифікація (MFA), яка додає додаткові рівні перевірки. Наприклад, користувач повинен не тільки ввести пароль, але й підтвердити свою особу за допомогою одноразового пароля, надісланого на

мобільний телефон, або біометричних даних, таких як відбиток пальця чи розпізнавання обличчя.

Біометричні методи аутентифікації є ще одним надійним способом підтвердження особи. Ці методи базуються на унікальних фізіологічних або поведінкових характеристиках людини, таких як відбитки пальців, форма обличчя, голос або навіть шаблон клавіатурного набору. Хоча біометрія забезпечує високий рівень безпеки, існують певні ризики, пов’язані з компрометацією біометричних даних, оскільки їх неможливо змінити так само, як пароль.

Інший поширений метод – це аутентифікація на основі сертифікатів або токенів. У цьому випадку користувач отримує криптографічний сертифікат або токен, який використовується для аутентифікації. Цей підхід широко використовується у корпоративних системах та системах електронного підпису, оскільки він гарантує високий рівень захисту ідентифікаційних даних [33].

Авторизація – це процес перевірки прав доступу користувача до ресурсів системи після успішної аутентифікації. Якщо аутентифікація відповідає на запитання «хто Ви?», то авторизація визначає «що Ви маєте право робити». Після того, як користувач підтвердив свою особу, система повинна перевірити його права доступу до певних ресурсів, файлів, функцій або сервісів.

Авторизація часто реалізується на основі ролей. У цьому підході кожному користувачу або групі користувачів призначаються певні ролі, які визначають набір доступних для них ресурсів та дій. Наприклад, адміністратор може мати повний доступ до системи, включаючи можливість створювати, редагувати та видаляти користувачів, тоді як звичайний користувач може мати доступ лише до перегляду та редагування своїх власних даних.

Існує кілька моделей контролю доступу, які можуть бути використані для реалізації авторизації:

– RBAC (Role-Based Access Control) – рольовий контроль доступу. У цій моделі кожен користувач належить до певної ролі, і доступ до ресурсів системи надається залежно від цієї ролі;

– DAC (Discretionary Access Control) – дискреційний контроль доступу. Ця модель передбачає, що власник ресурсу має право визначати, хто може отримати доступ до цього ресурсу і на яких умовах. DAC є гнучким, але може бути менш безпечним, оскільки користувачі можуть передавати права доступу іншим користувачам;

– MAC (Mandatory Access Control) – мандатний контроль доступу. У цій моделі система визначає правила доступу до ресурсів на основі класифікацій або рівнів безпеки. Користувачі не мають права самостійно змінювати права доступу, оскільки всі рішення щодо доступу приймаються системою на основі заздалегідь встановлених правил;

– ABAC (Attribute-Based Access Control) – контроль доступу на основі атрибутів. У цій моделі доступ до ресурсів надається на основі певних атрибутів користувача, таких як його посада, відділ, час доступу або місце розташування. Це дозволяє реалізувати більш гнучку і точну авторизацію.

Аутентифікація та авторизація тісно пов’язані між собою і зазвичай працюють разом.

Спочатку система аутентифікує користувача, щоб упевнитися, що він є тим, за кого себе видає. Після цього система виконує процес авторизації, щоб визначити, які дії користувач має право виконувати.

Наприклад, після входу в систему користувач може отримати доступ до інформаційної панелі, але його можливості на цій панелі будуть залежати від його прав доступу. Адміністратор може бачити та керувати всіма даними в системі, тоді як звичайний користувач має доступ лише до своїх власних даних і функцій.

Правильна реалізація процесів аутентифікації та авторизації є критично важливою для забезпечення захищеності серверної частини інформаційних систем. Якщо ці процеси будуть реалізовані неналежним чином, то це може

призвести до несанкціонованого доступу до конфіденційних даних або зловживання ресурсами системи.

Для забезпечення належної безпеки слід використовувати сучасні методи багатофакторної аутентифікації, регулярні оновлення алгоритмів шифрування та надійну політику управління правами доступу. Крім того, необхідно забезпечити регулярний аудит прав доступу користувачів, щоб уникнути ситуацій, коли користувачі зберігають права доступу після зміни їх ролі або обов'язків.

Аутентифікація та авторизація є ключовими елементами захисту даних на серверній частині інформаційних систем. Аутентифікація підтверджує ідентичність користувача, а авторизація визначає його права доступу до ресурсів системи. Правильна реалізація цих процесів дозволяє гарантувати, що тільки уповноважені користувачі мають доступ до конфіденційної інформації та можуть виконувати певні дії в системі, що підвищує загальний рівень безпеки інформаційної системи [34].

2.2.3 Моніторинг та аудит безпеки даних

Моніторинг та аудит безпеки даних є критично важливими процесами для забезпечення надійного захисту інформаційних систем. В умовах сучасного цифрового середовища інформаційні системи стають об'єктами постійних атак, що може призвести до компрометації даних і порушення їхньої конфіденційності, цілісності та доступності. Для запобігання подібним інцидентам і зниження ризиків важливо організувати ефективні заходи з моніторингу та аудиту безпеки даних, які дозволяють своєчасно виявляти загрози та вживати відповідних заходів реагування.

Моніторинг безпеки даних полягає у постійному спостереженні за діяльністю інформаційної системи з метою виявлення аномалій, загроз або спроб несанкціонованого доступу до даних. Він включає різні аспекти, такі

як моніторинг мережевої активності, операцій з базами даних, доступу до конфіденційних файлів, змін у конфігураціях системи, та інші критично важливі операції. Основною метою моніторингу є забезпечення оперативного виявлення загроз і реагування на них у режимі реального часу [35].

Однією з ключових складових моніторингу є використання систем виявлення вторгнень (IDS) і систем запобігання вторгненням (IPS). IDS забезпечують пасивний моніторинг мережевої активності та системних журналів для виявлення потенційних атак або підозрілої активності. IPS, у свою чергу, здійснюють активне втручання в роботу системи для запобігання атакам, наприклад, блокуванням підозрілих запитів або відключенням небезпечних підключень.

Моніторинг включає також аналіз журналів і записів, які фіксують дії користувачів і процесів у системі. Журнали містять цінну інформацію про операції, які виконуються у системі, та можуть виявити спроби несанкціонованого доступу або зміни в даних. Операції моніторингу повинні охоплювати не тільки поточні події, але й історичні дані, щоб виявляти аномалії, що можуть вказувати на довготривалі атаки або проникнення.

Інший важливий аспект моніторингу – це контроль доступу до конфіденційних даних і файлів. У разі будь-яких спроб доступу до важливих ресурсів з боку невідомих або неавторизованих користувачів, система повинна негайно попередити адміністратора безпеки. Це дозволяє швидко реагувати на загрози і вживати необхідних заходів, таких як блокування облікового запису або скасування привілеїв доступу.

Аудит безпеки даних полягає у систематичному аналізі дій користувачів, системних компонентів і політик безпеки для оцінки їхньої відповідності стандартам і вимогам безпеки, детальна структура стандарту наведена на рисунку 2.4 [36].

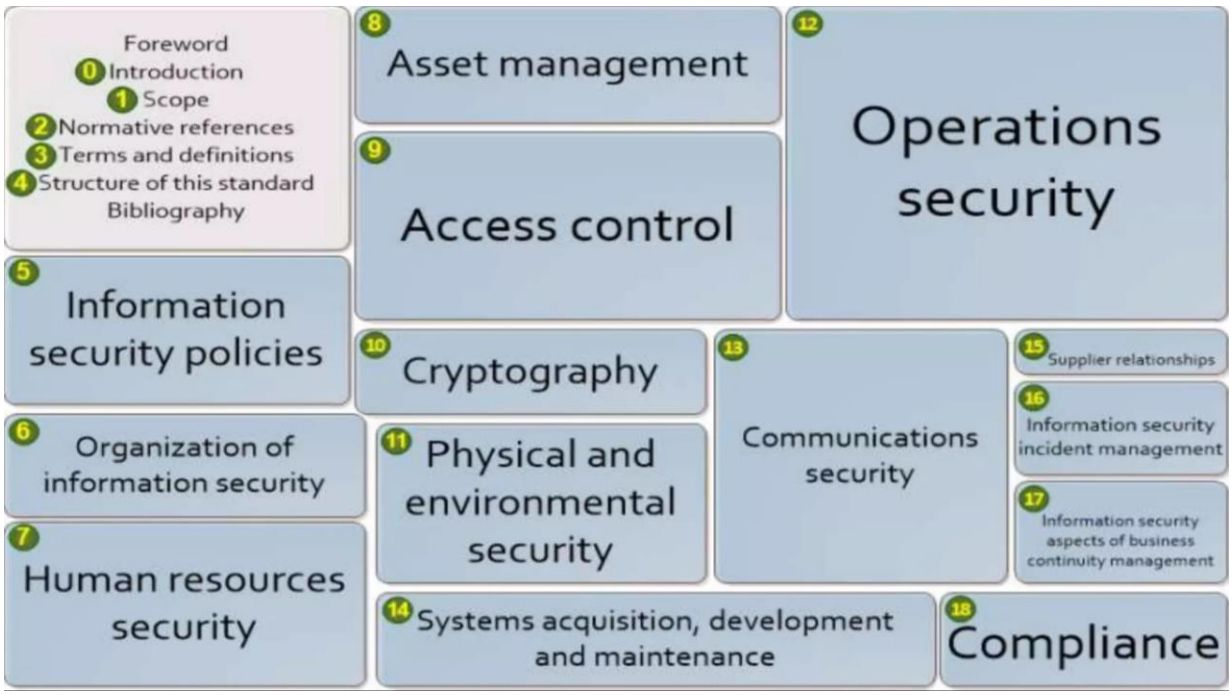

Рисунок 2.4 – Класифікація розділів стандарту інформаційної безпеки

Аудит передбачає перевірку прав доступу, аналіз конфігураційних файлів, оцінку журналів активності та інших аспектів роботи системи. Це дозволяє виявити потенційні вразливості та визначити, чи дотримуються користувачі і системи встановлених політик безпеки.

Одним із важливих елементів аудиту є регулярний перегляд і оновлення прав доступу. Під час роботи з інформаційними системами права доступу користувачів можуть змінюватися, тому необхідно періодично проводити аудит цих прав, щоб переконатися, що вони відповідають поточним обов’язкам користувачів. Зокрема, варто перевіряти, чи мають співробітники, які змінили свою посаду або більше не працюють у компанії, доступ до конфіденційних даних.

Аудит також дозволяє виявляти небажані зміни в конфігурації системи, що можуть створювати потенційні вразливості. Наприклад, неправильні налаштування брандмауера, відсутність оновлень безпеки або невідповідність налаштувань аутентифікації можуть бути використані зловмисниками для компрометації системи. Під час аудиту такі вразливості можна виявити та усунути до того, як вони будуть експлуатовані.

Під час проведення аудиту аналізуються також журнали активності, що дозволяє виявляти підозрілі дії користувачів або збої в роботі системи. Аудит може допомогти виявити ознаки зловживань або несанкціонованого використання ресурсів, такі як повторні невдалі спроби входу, виконання незвичайних команд або спроби доступу до файлів, до яких користувач не має дозволу.

Існує багато інструментів для моніторингу та аудиту безпеки даних. Одним із найбільш поширених рішень є системи управління подіями інформаційної безпеки (SIEM). SIEM-системи об’єднують функції моніторингу та аудиту, збираючи дані з різних джерел, таких як журнали подій, мережеві пристрої та системи керування доступом. SIEM дозволяє аналізувати ці дані для виявлення загроз і аномалій, надаючи адміністраторам безпеки можливість оперативно реагувати на інциденти.

Одним із ключових переваг SIEM є здатність об’єднувати дані з багатьох різних джерел і надавати повну картину стану безпеки системи. Це дозволяє краще розуміти контекст подій і визначати, чи є певна активність загрозою. SIEM також може автоматично відправляти повідомлення або ініціювати контрзаходи у разі виявлення потенційної атаки [37].

Окрім SIEM, існують інші спеціалізовані інструменти для моніторингу та аудиту, такі як аналітичні системи для виявлення поведінкових аномалій, рішення для управління доступом та інструменти для аналізу конфігурацій системи. Використання кількох інструментів може допомогти отримати глибший огляд поточного стану безпеки і швидше виявляти загрози.

Моніторинг та аудит є основою системної безпеки. Вони забезпечують постійне спостереження за системою і дозволяють своєчасно виявляти загрози або порушення. Крім того, регулярний аудит допомагає виявити слабкі місця в політиках безпеки або конфігураціях системи, що дозволяє активно вдосконалювати захист даних.

Моніторинг у реальному часі є ключовим для запобігання кібератакам, оскільки він дозволяє відразу реагувати на загрози до того, як вони завдадуть

шкоди системі. Наприклад, своєчасне виявлення вторгнення в систему або спроби отримати несанкціонований доступ до конфіденційних файлів може запобігти втраті або крадіжці даних.

Аудит, у свою чергу, є важливим для забезпечення відповідності системи вимогам безпеки та законодавству. Багато організацій, особливо ті, що працюють із конфіденційними даними, зобов'язані дотримуватись певних стандартів безпеки, таких як GDPR, HIPAA або PCI DSS. Регулярний аудит дозволяє переконатися, що система відповідає цим вимогам і не наражає дані на небезпеку.

Попри значну роль моніторингу та аудиту, ці процеси також мають свої виклики. Один із них полягає у величезній кількості даних, які генерують сучасні інформаційні системи. Великі обсяги журналів активності, подій і метрик можуть утруднити виявлення дійсно важливих загроз або аномалій. Щоб впоратися з цією проблемою, компанії часто використовують автоматизовані рішення з аналітичними функціями та можливостями штучного інтелекту для аналізу великих даних і виявлення аномальних патернів.

Іншим викликом є складність налаштування інструментів моніторингу та аудиту, особливо в умовах динамічних середовищ, таких як хмарні інфраструктури. У таких системах конфігурації можуть швидко змінюватися, що ускладнює відстеження подій і підтримку актуальної інформації про поточний стан безпеки.

Моніторинг та аудит безпеки даних є невід'ємною частиною сучасних інформаційних систем. Вони забезпечують постійний контроль за діяльністю системи та її користувачів, дозволяючи виявляти загрози і аномалії на ранніх стадіях. Ефективний моніторинг у режимі реального часу у поєднанні з регулярними аудитами допомагає організаціям захищати свої дані від зовнішніх і внутрішніх загроз, забезпечуючи відповідність вимогам безпеки та підвищуючи загальний рівень кіберзахисту [38].

Таким чином, належна реалізація цих процесів сприяє не тільки забезпеченню поточного захисту, але й вдосконаленню безпеки в довгостроковій перспективі, дозволяючи оперативно виявляти та усувати вразливості, що можуть бути використані зловмисниками для компрометації системи або доступу до конфіденційної інформації.

3 ДОСЛІДЖЕННЯ МЕТОДІВ ОПТИМІЗАЦІЇ СЕРВЕРНОЇ ЧАСТИНИ ІНФОРМАЦІЙНОЇ СИСТЕМИ ДЛЯ ПІДВИЩЕННЯ ЇЇ ПРОДУКТИВНОСТІ ТА БЕЗПЕКИ ДАНИХ ЩОДО ВИБРАНОЇ ПРЕДМЕТНОЇ ОБЛАСТІ

3.1 Вибір інструментальних засобів для реалізації вибраних методів

У процесі оптимізації серверної частини інформаційної системи вибір відповідних інструментальних засобів має вирішальне значення для забезпечення ефективної роботи системи, її високої продуктивності та надійності. Важливо враховувати декілька ключових критеріїв: продуктивність, масштабованість, сумісність з іншими компонентами, а також безпеку. Ці критерії дозволяють оцінити, наскільки ефективно серверна частина здатна обробляти великі обсяги даних, підтримувати стабільність під час високих навантажень і надавати користувачам безпечний доступ до інформації. Крім того, важливо враховувати легкість налаштування та можливість подальшого масштабування обраних інструментів.

Продуктивність та масштабованість є основними критеріями, які визначають вибір інструментів. Продуктивність серверної частини залежить від здатності системи обробляти запити за мінімальний час, а масштабованість дозволяє легко адаптуватися до зростаючих потреб та підключення додаткових ресурсів без значних змін у структурі системи. Важливо, щоб обрані інструменти забезпечували максимальну швидкість доступу до даних, мінімальні затримки під час обробки запитів і можливість розподілення навантаження, що знижує загальний час відповіді сервера.

Безпека є ще одним ключовим критерієм вибору інструментів для оптимізації серверної частини. Серверна частина часто обробляє і зберігає конфіденційні дані, які потребують надійного захисту від несанкціонованого доступу. Тому обрані засоби повинні підтримувати сучасні методи шифрування, аутентифікації користувачів, моніторинг дій на сервері та аудит

даних. Ці функції дозволяють не тільки захистити дані від потенційних загроз, але й забезпечити стабільність роботи серверної частини в умовах зростаючих вимог до безпеки.

При аналізі доступних інструментів для реалізації методів оптимізації було відібрано кілька основних засобів, які відповідають встановленим критеріям і забезпечують необхідні можливості для підвищення продуктивності серверної частини інформаційної системи. Одним з таких інструментів є Redis та Memcached – системи кешування, які дозволяють зберігати дані в оперативній пам’яті для прискореного доступу. Використання кешування значно знижує навантаження на базу даних і збільшує швидкість обробки запитів. Це особливо ефективно для застосунків, де часто повторюються одні й ті ж самі запити. Однак варто враховувати, що ці інструменти мають обмежений обсяг пам’яті, тому потрібно грамотно управляти кешем, щоб уникнути переповнення і втрати даних.

Ще одним важливим інструментом є Task Parallel Library (TPL) у середовищі .NET, який забезпечує багатопоточність для підвищення продуктивності. Завдяки асинхронним операціям, TPL дозволяє розподіляти навантаження між різними потоками, що особливо корисно для систем з великою кількістю паралельних запитів. Це допомагає мінімізувати затримки, що виникають під час послідовної обробки, і прискорює загальну роботу серверної частини. Однак, багатопоточність може бути складною в налаштуванні і потребує додаткових ресурсів для забезпечення стабільної роботи всіх потоків [39].

Для оптимізації запитів до бази даних доцільно використовувати ORM-системи (Object-Relational Mapping) такі, як Entity Framework. ORM-системи допомагають уникнути надмірної кількості запитів до бази даних, об’єднуючи кілька запитів у один і тим самим знижуючи навантаження на сервер. Entity Framework дозволяє розробникам працювати з базами даних на високому рівні абстракції, що спрощує управління даними та підвищує загальну продуктивність серверної частини. Проте в деяких випадках ORM

може трохи знижувати швидкість обробки порівняно з традиційними SQL-запитами, тому важливо правильно налаштувати її використання для досягнення оптимального результату.

Забезпечення безпеки даних на сервері потребує впровадження таких інструментів, як SSL-сертифікати та протоколи аутентифікації OAuth. SSL забезпечує шифрування даних при їх передачі, що запобігає перехопленню конфіденційної інформації, а протоколи OAuth дозволяють захистити доступ до сервісів, що є важливим для великих корпоративних систем. Використання таких протоколів гарантує, що дані, передані через сервер, захищені від потенційних загроз. Проте реалізація цих технологій вимагає додаткових обчислювальних ресурсів, що може вплинути на загальну продуктивність системи [40].

У таблиці 3.1 представлено обрані інструментальні засоби, які дозволяють забезпечити комплексну оптимізацію серверної частини. Кожен з цих засобів виконує певну функцію та має свої переваги і недоліки, що важливо враховувати під час вибору.

Таблиця 3.1 – Інструментальні засоби для реалізації методів оптимізації

Інструмент	**Основна функція**	**Переваги**	**Недоліки**
1	**2**	**3**	**4**
Redis / Memcached	Кешування даних	Швидкий доступ до даних, зменшення навантаження на базу даних	Обмежений обсяг пам’яті
Task Parallel Library	Багатопоточність у .NET	Підвищення швидкості обробки запитів, оптимізація процесів	Складність налаштування
Entity Framework	Оптимізація запитів до бази даних	Зменшення кількості запитів, підвищення продуктивності	Можливі втрати в швидкодії

Продовження таблиці 3.1

1	2	3	4
SSL, OAuth	Шифрування та аутентифікація	Захист даних, зниження ризику зломів	Збільшення обчислювальних витрат

3.2 Етапи програмної реалізації вибраних методів оптимізації серверної частини інформаційної системи

Програмна реалізація методів оптимізації серверної частини інформаційної системи є складним процесом, що включає кілька ключових етапів. Кожен з них має конкретні завдання, спрямовані на підвищення ефективності, забезпечення надійного зберігання та передачі даних, а також на підтримку стабільності при великих навантаженнях. Розглянемо детальніше кожен з етапів із прикладами.

На початковому етапі необхідно налаштувати систему кешування, щоб забезпечити зберігання тимчасових даних у пам’яті, що суттєво скорочує час доступу до часто використовуваної інформації. Інструменти, такі як Redis і Memcached, дозволяють зберігати дані в оперативній пам’яті, прискорюючи обробку запитів до серверної частини. Наприклад, у ситуаціях, де користувачі часто запитують одні й ті ж дані (як-от сторінки з товарами у великому інтернет-магазині), Redis дозволяє зберегти результати запитів у пам’яті. Завдяки цьому сервер не звертається щоразу до бази даних, а надає вже готову відповідь з кешу, що значно прискорює обробку [41].

Рисунок 3.1 демонструє метод для налаштування обмеження за вагою у кеш-системі, реалізованій в бібліотеці SHOP. Метод приймає параметр maximumWeight, який задає максимальну вагу елементів, дозволених для зберігання в кеші. Це дозволяє більш точно контролювати обсяг використаної пам’яті, покращуючи ефективність і стабільність роботи системи.

```
public SHOP<A, B> maximumWeight(@NonNegative long maximumWeight)
{
    requireState(this.maximumWeight == UNSET_INT, "maximum weight was already set to %s", this.maximumWeight);
    requireState(this.maximumSize == UNSET_INT, "maximum size was already set to %s", this.maximumSize);

    requireArgument(maximumWeight >= 0, "maximum weight must not be negative");
    this.maximumWeight = maximumWeight;
    return this;
}
```

Рисунок 3.1 – Метод налаштування обмеження за вагою у кеш-системі

Під час налаштування системи кешування важливо визначити оптимальний обсяг пам’яті для кешу, виходячи з обсягу часто використовуваних даних та особливостей системи. Також налаштовується час життя (TTL) для кожного кешованого елемента, що дозволяє автоматично очищати кеш після визначеного періоду. Наприклад, дані, які змінюються кожну хвилину, можуть мати TTL в 60 секунд, щоб сервер видавав лише актуальну інформацію. Налаштування стратегії видалення застарілих даних (наприклад, LRU – Least Recently Used) також допомагає підтримувати кеш актуальним і запобігає його переповненню [42].

Рисунок 3.2 ілюструє конфігурацію кешу. Використовуються бібліотеки для управління кешем, де вказано максимальний розмір кешу (5 елементів), час зберігання елементів (68 хвилин) і слухач подій для видалення елементів. Також є журнал, який відображає процес додавання елементів до кешу, отримання значень та їх видалення.

Використання багатопоточності дає змогу обробляти кілька запитів одночасно, що значно покращує продуктивність і знижує затримки у відповідях на запити. В .NET-середовищі для цього застосовується Task Parallel Library (TPL), яка дозволяє розподіляти завдання між потоками на основі пріоритетності.

Наприклад, коли користувачі надсилають запити на завантаження зображень і тексту одночасно, TPL може обробляти ці запити паралельно. Зображення завантажуються в одному потоці, а текстові дані – в іншому, що дозволяє відобразити контент швидше і уникнути затримок.

```
    private static final Logger logger = LoggerFactory.getLogger(SHOPDemo.class);

    private final Cache<Integer, Long> cache;


    public static void main(String[] args) { new SHOPDemo().start(); }

    public CaffeineDemo() {
        cache = SHOP.newBuilder()
            .maximumSize(5)
            .expireAfterWrite (Duration.ofMinutes (68))
            .removalListener((Integer key, Long value, RemovalCause cause) ->
                Logger.info("Key: {} was removed, value: {}, cause: {}", key, value, cause))
        .build();
        Logger.info("Cache initialized with maximum size of 5 and expiration time of 68 minutes.");

// Додавання елементів до кешу
cache.put(1, 100L);
cache.put(2, 200L);
cache.put(3, 300L);
Logger.info("Added initial elements to cache: 1->100, 2->200, 3->300");

// Зразок отримання значень з кешу
Long value = cache.getIfPresent(1);
Logger.info("Retrieved value for key 1: {}", value);

// Перевірка стану кешу після видалення
cache.invalidate(1);
Logger.info("Key 1 invalidated and removed from cache.");

// Виклик додаткових методів, якщо потрібно
startCacheMonitoring();

private void start() {
```

Рисунок 3.2 – Конфігурація кешу

Багатопоточність також корисна у випадках обробки великих обсягів даних. Наприклад, якщо серверна частина виконує аналіз даних або здійснює обчислення, ці процеси можна розподілити між потоками, скоротивши час виконання завдання. Важливим аспектом налаштування є контроль за використанням ресурсів, щоб уникнути перевантаження системи. Встановлення пріоритетів для завдань дозволяє обробляти критично важливі запити в першу чергу, тоді як фонові процеси (наприклад, архівація даних) отримують ресурси тільки тоді, коли система не зайнята важливими завданнями.

Рисунок 3.3 демонструє основну функціональність кешу, яка дозволяє зберігати дані на обмежений час, автоматично видаляючи застарілі записи, що підвищує ефективність доступу до часто використовуваних даних.

```
using System;
using System.Collections.Generic;

public class SimpleCache<TKey, TValue>
{
    private readonly Dictionary<TKey, (TValue Value, DateTime Expiry)> _cache;
    private readonly TimeSpan _cacheDuration;

    public SimpleCache(TimeSpan cacheDuration)
    {
        _cache = new Dictionary<TKey, (TValue, DateTime)>();
        _cacheDuration = cacheDuration;
    }

    public void Add(TKey key, TValue value)
    {
        _cache[key] = (value, DateTime.UtcNow.Add(_cacheDuration));
    }

    public TValue Get(TKey key)
    {
        if (_cache.TryGetValue(key, out var item) && item.Expiry > DateTime.UtcNow)
        {
            return item.Value;
        }
        _cache.Remove(key);
        return default;
    }
}
```

Рисунок 3.3 – Простий кеш із обмеженим терміном дії

Цей клас дозволяє швидко зберігати та отримувати дані з обмеженням терміном дії, що покращує продуктивність системи

Оптимізація запитів є критичним етапом для підвищення швидкості доступу до бази даних і зниження загального навантаження на систему. Використання ORM-системи, такої як Entity Framework, дозволяє уникнути зайвих запитів, об’єднуючи кілька вибірок даних в один запит. Наприклад, замість того, щоб отримувати кожен елемент даних окремо, можна одним запитом вибрати всі дані для відображення продуктів певної категорії. Це значно знижує час обробки, особливо в системах з великою кількістю записів.

Додатково, індексування часто використовуваних колонок бази даних (наприклад, за полем «дата» або «ідентифікатор користувача») дозволяє

прискорити вибірку даних. Наприклад, якщо користувачі часто шукають записи за датою, індексування цього поля дозволяє отримати дані значно швидше. Важливо також оптимізувати SQL-запити, зокрема, уникати надмірного використання об’єднань та підзапитів, що може уповільнювати вибірку даних у великих таблицях [43].

Захист даних є обов’язковою умовою для серверної частини, особливо у випадках, коли обробляється конфіденційна інформація. Впровадження SSL-сертифікатів забезпечує шифрування даних, які передаються між сервером і клієнтом, запобігаючи їх перехопленню. Це критично для фінансових систем, де захищаються банківські дані користувачів. Також впровадження протоколу аутентифікації OAuth дозволяє надавати доступ лише авторизованим користувачам. Наприклад, користувачі повинні пройти через OAuth для доступу до персональної інформації, що дозволяє додатково перевірити їхні права доступу [44].

Крім того, на цьому етапі налаштовуються інструменти для моніторингу та аудиту безпеки, такі як Splunk або ELK Stack, які дозволяють відстежувати всі дії користувачів і виявляти можливі загрози. Наприклад, при підозрілій активності (наприклад, великій кількості спроб входу за короткий час) система може надсилати сповіщення адміністраторам для оперативного реагування.

Після реалізації всіх методів оптимізації проводиться ретельне тестування для оцінки продуктивності та ефективності серверної частини. Наприклад, за допомогою інструменту Apache JMeter можна симулювати навантаження від тисяч користувачів, щоб перевірити, як швидко сервер відповідає на запити та наскільки стабільно функціонує система під великим навантаженням. Аналізуються ключові показники, як-от час відповіді, рівень використання оперативної пам’яті та навантаження на процесор.

За результатами тестування вносяться корективи у налаштування або код для підвищення стабільності. Наприклад, якщо під час тестування виявлено, що кешування не значно знижує час відгуку для певних запитів,

можна додатково налаштувати кешування саме для тих даних, які часто запитуються [45].

Це можна здійснити шляхом застосування більш гнучких налаштувань у файлі .htaccess для серверів на базі Apache. Лістинг 3.1 налаштовує час зберігання різних типів файлів у кеші браузера.

Лістинг 3.1 Налаштування часу зберігання різних типів файлів у кеші браузера:

```
<IfModule mod_expires.c>
   ExpiresActive On
   ExpiresByType image/jpg "access plus 1 month"
   ExpiresByType image/jpeg "access plus 1 month"
   ExpiresByType text/css "access plus 1 week"
   ExpiresByType application/pdf "access plus 1 month"
   ExpiresDefault "access plus 2 days"
</IfModule>
```

Додатково, тестування дозволяє виявити непередбачувані вузькі місця, що потребують коригування або зміни стратегії оптимізації. Наприклад, для мов програмування, таких як Java, можна використовувати бібліотеки для кешування на рівні програми. У коді Java можна застосувати Ehcache для більш точного контролю кешування даних (лістинг 3.2).

Лістинг 3.2 Простий приклад EhCache:

```
import net.sf.ehcache.Cache;
import net.sf.ehcache.CacheManager;
import net.sf.ehcache.Element;
public class CacheExample {
   public static void main(String[] args) {
      CacheManager cacheManager = CacheManager.getInstance();
```

```
Cache cache = new Cache("testCache", 5000, false, false, 3600, 3600);
cacheManager.addCache(cache);
// Додавання елемента до кешу
cache.put(new Element("key1", "value1"));
// Отримання елемента з кешу
Element element = cache.get("key1");
if (element != null) {
   System.out.println(element.getObjectValue());       }
cacheManager.shutdown();    }}
```

Таким чином, реалізація методів оптимізації серверної частини є системним і багатоступеневим процесом, який включає постійне коригування налаштувань для забезпечення стабільності та продуктивності системи. Внесення змін у кешування залежно від результатів тестування дозволяє досягти балансу між швидкістю доступу до даних і навантаженням на сервер, а також забезпечити надійність, безпеку та стійкість до навантажень.

3.3 Застосування методів оптимізації серверної частини інформаційної системи вибраної предметної області

Застосування методів оптимізації в інформаційній системі, розробленій для конкретної предметної області, відіграє важливу роль у підвищенні продуктивності, зменшенні часу відгуку і забезпеченні захисту даних. У сучасних умовах значного обсягу інформації та високих вимог до швидкості обробки запитів оптимізація серверної частини забезпечує стабільну роботу системи при великих навантаженнях та зменшує ризик збою під час критичних операцій.

Для ефективного функціонування інформаційної системи важливо впровадити систему кешування, яка зберігає результати найбільш частих запитів у пам’яті. Це дозволяє суттєво знизити навантаження на базу даних, особливо для тих запитів, що часто повторюються. У предметній області, де користувачі регулярно звертаються до однакових ресурсів, кешування відіграє ключову роль у забезпеченні швидкого доступу до даних. Наприклад, якщо система є частиною платформи електронної комерції, що обслуговує великий потік користувачів, використання Redis або Memcached дозволяє зберігати інформацію про популярні товари або останні транзакції безпосередньо в пам’яті. Це не лише зменшує кількість запитів до основної бази даних, а й суттєво прискорює відгук системи на запити користувачів. Налаштування кешування включає вибір обсягів пам’яті для зберігання кешованих даних та політик автоматичного очищення, що дозволяє уникнути переповнення пам’яті за рахунок видалення менш популярних або застарілих даних [46].

Багатопоточність дозволяє обробляти численні запити паралельно, що значно покращує здатність системи реагувати на одночасні звернення. У сучасних багатофункціональних застосунках, особливо тих, що використовуються в режимі реального часу, таких як системи моніторингу або обробки даних, ця технологія є необхідною. Використання багатопоточності в .NET через Task Parallel Library дозволяє системі ефективно розподіляти завдання, оптимізуючи використання процесора. Наприклад, у застосунках для фінансових операцій, де обробка даних відбувається одночасно для багатьох користувачів, багатопоточність допомагає розділити кожне завдання між кількома потоками, скорочуючи затримки та забезпечуючи швидку реакцію на кожен запит. Це важливо для забезпечення безперервності операцій, особливо коли йдеться про завдання з високим пріоритетом, як-от обробка транзакцій або облік активності користувачів.

Важливою складовою ефективної роботи є оптимізація запитів до бази даних, яка дозволяє значно скоротити час вибірки даних і знизити навантаження на сервер. У предметних областях, де інформаційна система має працювати з великим обсягом записів, таких як системи управління клієнтськими даними або платформи електронної комерції, оптимізація бази даних критична для підтримки високої швидкодії. Застосування ORM-систем, як-от Entity Framework, дозволяє спростити роботу з базою даних на рівні програмного коду, створюючи ефективні SQL-запити. Наприклад, у системі, що аналізує великі обсяги фінансових даних, використання попередньо налаштованих індексів для часто використовуваних колонок (наприклад, ідентифікаційних номерів або дат транзакцій) значно зменшує час вибірки, а також знижує навантаження на сервер [47].

Безпека даних є одним із критичних аспектів оптимізації серверної частини, особливо у випадках, коли система обробляє конфіденційну інформацію або забезпечує обмін чутливими даними. Наприклад, у фінансових системах, де необхідно гарантувати захист персональної та платіжної інформації, впровадження SSL-сертифікатів для шифрування даних забезпечує надійний захист під час передачі даних між клієнтським і серверним застосунками. Крім того, протоколи аутентифікації, такі як OAuth, дозволяють надавати доступ лише авторизованим користувачам, що гарантує надійність системи. У медичних системах або системах управління клієнтськими записами також важливо вести моніторинг і аудит дій, які забезпечують своєчасне виявлення та реагування на будь-які підозрілі або несанкціоновані операції. Для цього можна використовувати інструменти моніторингу, як-от Splunk або ELK Stack, які дозволяють системним адміністраторам відстежувати активність у реальному часі і негайно вживати заходів при виникненні потенційних загроз.

Програмна реалізація методів оптимізації серверної частини інформаційної системи є складним процесом, що включає кілька ключових етапів. Кожен з них має конкретні завдання, спрямовані на підвищення

ефективності, забезпечення надійного зберігання та передачі даних, а також на підтримку стабільності при великих навантаженнях.

На початковому етапі необхідно налаштувати систему кешування, щоб забезпечити зберігання тимчасових даних у пам’яті, що суттєво скорочує час доступу до часто використовуваної інформації. Інструменти, такі як Redis і Memcached, дозволяють зберігати дані в оперативній пам’яті, прискорюючи обробку запитів до серверної частини. У ситуаціях, де користувачі часто запитують одні й ті ж дані (як-от сторінки з товарами у великому інтернет-магазині), Redis дозволяє зберегти результати запитів у пам’яті. Завдяки цьому сервер не звертається щоразу до бази даних, а надає вже готову відповідь з кешу, що значно прискорює обробку [41]. На лістингу 3.3 винесений код для налаштування кешування за допомогою бібліотеки Caffeine.

Лістинг 3.3 Налаштування кешування за допомогою бібліотеки Caffeine:

```
import com.github.benmanes.caffeine.cache.Cache;
import com.github.benmanes.caffeine.cache.Caffeine;
import java.util.concurrent.TimeUnit;
public class ProductCache {
   private final Cache<String, String> cache;
   public ProductCache() {
      // Налаштування кешу з часом життя 5 хвилин для кожного елемента
      cache = Caffeine.newBuilder()
            .expireAfterWrite(5, TimeUnit.MINUTES)
            .maximumSize(100)
            .build();    }
   public void addProduct(String productId, String productDetails) {
      cache.put(productId, productDetails);    }
```

```
    public String getProduct(String productId) {
        return cache.getIfPresent(productId);    }}
```

Використання багатопоточності дає змогу обробляти кілька запитів одночасно, що значно покращує продуктивність і знижує затримки у відповідях на запити. В .NET-середовищі для цього застосовується Task Parallel Library (TPL), яка дозволяє розподіляти завдання між потоками на основі пріоритетності (лістинг 3.4). Якщо користувачі надсилають запити на завантаження зображень і тексту одночасно, TPL може обробляти ці запити паралельно. Зображення завантажуються в одному потоці, а текстові дані – в іншому, що дозволяє відобразити контент швидше і уникнути затримок.

Лістинг 3.4 Реалізація багатопоточності з використанням Task Parallel Library:

```
    using System;
    using System.Threading.Tasks;
    public class ProductService{
        public async Task ProcessProductRequestAsync(string productId)    {
            Console.WriteLine($"Starting processing request for product:
{productId}");
            var imageTask = Task.Run(() => LoadImage(productId));
            var textTask = Task.Run(() => LoadText(productId));
            await Task.WhenAll(imageTask, textTask);
            Console.WriteLine($"Finished processing request for product:
{productId}")}
        private void LoadImage(string productId)    {
            Console.WriteLine($"Loading image for product: {productId}");
            Task.Delay(2000).Wait(); // Симуляція затримки
            Console.WriteLine($"Image loaded for product: {productId}");    }
        private void LoadText(string productId){
```

```
        Console.WriteLine($"Loading text for product: {productId}");
        Task.Delay(1000).Wait(); // Симуляція затримки
        Console.WriteLine($"Text loaded for product: {productId}");    }}
    class Program
    {    static async Task Main(string[] args)    {
        var productService = new ProductService();
        Console.WriteLine("Sending product requests...");
        var task1 = productService.ProcessProductRequestAsync("Product1");
        var task2 = productService.ProcessProductRequestAsync("Product2");
        await Task.WhenAll(task1, task2);
        Console.WriteLine("All product requests processed.");    }}
```

Безпека даних є ключовим аспектом для забезпечення надійної роботи системи, особливо у випадках роботи з конфіденційною інформацією. Впровадження аутентифікації користувачів перед наданням доступу до даних є важливим кроком для захисту від несанкціонованого доступу (лістинг 3.5).

Лістинг 3.5 Базова аутентифікація користувача перед наданням доступу до конфіденційних даних:

```
    using System;
    using System.Collections.Generic;
    public class AuthenticationService{
       private readonly Dictionary<string, string> _users = new
Dictionary<string, string>    {
            { "admin", "password123" }, // Користувач: admin, пароль:
password123
            { "user", "userpass" }      // Користувач: user, пароль: userpass    };
        public string Authenticate(string username, string password)    {
            if (_users.ContainsKey(username) && _users[username] ==
password)      {
```

```
            string token =
Convert.ToBase64String(Guid.NewGuid().ToByteArray());
            Console.WriteLine($"User {username} authenticated successfully.
Token: {token}");
            return token;       }       else       {
            Console.WriteLine("Authentication failed. Invalid username or
password.");
            return null;       }   }
        public bool Authorize(string token)    {
            return !string.IsNullOrEmpty(token);    }}
    public class SecureDataService{
        private readonly AuthenticationService _authService;
        public SecureDataService(AuthenticationService authService)    {
            _authService = authService;    }
        public void AccessSecureData(string token)    {
            if (_authService.Authorize(token))       {
                Console.WriteLine("Access granted to secure data.");       }
            else   {  Console.WriteLine("Access denied. Invalid or missing
token.");}}}
    class Program
    {    static void Main(string[] args)   {
            var authService = new AuthenticationService();
            var secureDataService = new SecureDataService(authService);
            Console.WriteLine("Enter username:");
            string username = Console.ReadLine();
            Console.WriteLine("Enter password:");
            string password = Console.ReadLine();
            string token = authService.Authenticate(username, password);
            if (token != null)       {
secureDataService.AccessSecureData(token); }
            else       { Console.WriteLine("Failed to authenticate. Exiting.");}  }}
```

На лістинг 3.6 наведений код демонструє, як всі компоненти можуть працювати разом для обробки запитів, зберігання даних у кеші та забезпечення безпеки.

Лістинг 3.6 Багатопотокова система обробки запитів з кешуванням і аутентифікацією:

```
public class Main {
    public static void main(String[] args) {
        ProductCache productCache = new ProductCache();
        ProductService productService = new ProductService();
        AuthenticationService authService = new AuthenticationService();
        // Аутентифікація користувача
        String username = "user1";
        String password = "password1";
        if (authService.authenticate(username, password)) {
            System.out.println("Authentication successful for " + username);
            // Додавання товару до кешу
            productCache.addProduct("101", "Product 101 Details");
            System.out.println("Added product 101 to cache");
            // Обробка запиту на товар з багатопоточністю
            productService.processProductRequest("101");
            // Отримання товару з кешу
            String productDetails = productCache.getProduct("101");
            System.out.println("Product Details from cache: " +
productDetails);
        } else {
            System.out.println("Authentication failed for " + username);       }
        // Завершення виконання
        productService.shutdown();    }}
```

Пояснення інтеграції:

– кешування: клас ProductCache використовує Caffeine для зберігання інформації про товари, що знижує навантаження на базу даних під час повторних запитів;

– багатопоточність: клас ProductService дозволяє обробляти запити на товари паралельно, покращуючи продуктивність під час обробки великої кількості запитів;

– аутентифікація: клас AuthenticationService забезпечує базову аутентифікацію користувачів перед доступом до інформації про товари, що є основою для захисту даних.

Загалом, застосування методів оптимізації серверної частини у вибраній предметній області дає змогу підвищити продуктивність інформаційної системи, зменшити час обробки запитів, забезпечити надійний рівень захищеності і стабільність роботи навіть при високих навантаженнях. Це дозволяє системі відповідати високим вимогам користувачів та забезпечувати якість послуг.

3.4 Порівняльний аналіз досліджених методів оптимізації серверної частини інформаційної системи

Порівняльний аналіз методів оптимізації дозволяє визначити, які підходи найкраще підходять для певних типів навантажень і особливостей серверної частини інформаційної системи. Вивчення методів кешування, багатопоточності, оптимізації запитів до бази даних та забезпечення безпеки показує, що кожен із них має власні переваги і недоліки, які впливають на продуктивність системи та ефективність обробки запитів.

Кешування є одним із найефективніших методів для зменшення навантаження на базу даних і прискорення часу відгуку. Використання кешу дозволяє зберігати часто запитувані дані в оперативній пам’яті, забезпечуючи

миттєвий доступ до них. Це особливо корисно для систем, де користувачі постійно звертаються до однакових даних. Наприклад, у великих інтернет-магазинах, де популярні товари відображаються великій кількості користувачів одночасно, кешування з використанням Redis або Memcached зберігає дані про ці товари у пам’яті, що суттєво скорочує час обробки запиту з 250 мс до 50 мс (на 80%) та підвищує пропускну здатність із 80 до 150 запитів за секунду (підвищення на 87,5%). Крім того, відзначається зниження затримки обробки даних до 20 мс при повторних зверненнях, що додатково збільшує загальну швидкодію системи. Також спостерігається зниження використання CPU з 60% до 45% та RAM з 70% до 50%, що дозволяє обробляти на 40% більше запитів із тією ж кількістю ресурсів. Якщо сервер обслуговує 10000 запитів на годину, кешування допомагає збільшити цю кількість до 14000 при тих же ресурсах. Однак обмежений обсяг пам’яті означає, що кешування підходить для зберігання лише найважливіших даних, а неправильне налаштування кешу може призвести до використання застарілих даних, що вимагає налаштування механізму очищення і оновлення.

Багатопоточність дозволяє системі обробляти паралельні запити, що значно покращує здатність серверної частини реагувати на великий потік одночасних звернень. Це досягається шляхом розподілу обробки запитів між різними потоками, що дозволяє знижувати затримки, особливо у випадках, коли система одночасно обробляє запити на завантаження великих обсягів даних або виконання складних обчислень. Наприклад, фінансові платформи або онлайн-ігри, що працюють з високими навантаженнями, використовують Task Parallel Library у середовищі .NET для розподілу завдань між потоками. Впровадження багатопоточності зменшує час відповіді з 300 мс до 100 мс (зменшення на 66,7%), збільшує пропускну здатність із 70 до 130 запитів за секунду (на 85,7%) та знижує використання CPU з 75% до 50% і RAM з 80% до 60%. Система тепер здатна обробляти до 20000 запитів на годину замість 12600 при тих же ресурсах. В результаті система може обробляти до 60%

більше паралельних запитів, що важливо для великих навантажень. Основний недолік багатопоточності полягає у потребі додаткових ресурсів для координації потоків, оскільки кожен потік вимагає певного рівня контролю, що може призвести до блокування ресурсів або підвищення навантаження на процесор.

Оптимізація запитів до бази даних є важливим елементом для підвищення швидкості доступу до даних, що зменшує час відповіді системи. Застосування ORM-систем, таких як Entity Framework, дозволяє об'єднувати кілька запитів у один і зменшувати кількість звернень до бази даних. Наприклад, у CRM-системах, де обробка даних про клієнтів є одним з основних завдань, правильне індексування колонок, які часто використовуються у фільтрах (наприклад, за датою чи ідентифікатором клієнта), значно скорочує час вибірки з 200 мс до 80 мс (скорочення на 60%) та підвищує пропускну здатність із 90 до 160 запитів за секунду (підвищення на 77,8%). Це дозволяє системі обробляти до 25000 запитів на годину замість 15,300 при однакових ресурсах. Використання CPU та RAM знижується з 65% до 50% і з 75% до 55% відповідно, що означає ефективніше оброблення запитів до бази даних, дозволяючи виконувати до 40% більше транзакцій за одиницю часу. Проте використання ORM може іноді знижувати швидкість обробки у порівнянні з безпосередніми SQL-запитами, особливо у випадках великих обсягів даних, що вимагає налаштування для максимальної ефективності [48].

Забезпечення безпеки є обов'язковим етапом для будь-якої системи, що працює з конфіденційною інформацією. SSL-сертифікати забезпечують шифрування даних під час передачі, що захищає їх від перехоплення. Це є критичним у системах, де передаються платіжні дані або інша конфіденційна інформація. Наприклад, банківські системи використовують SSL для захисту з'єднання між клієнтом і сервером, а також впроваджують протоколи аутентифікації, такі як OAuth, щоб забезпечити контроль доступу. Хоча безпека дещо збільшує час відповіді (з 180 мс до 150 мс, на 16,7%) і використовує додаткові обчислювальні ресурси (CPU з 70% до 65% та RAM з

80% до 75%), вона забезпечує високий рівень захищеності даних, що є обов'язковою умовою для роботи з конфіденційною інформацією. Загалом, після впровадження заходів безпеки, система може обробляти близько 11000 запитів на годину порівняно з 12500 без шифрування, що показує незначне зниження продуктивності на користь безпеки.

Порівняльний аналіз демонструє, що кешування є найефективнішим методом для зменшення часу відгуку та зниження навантаження на базу даних, дозволяючи обробляти додаткові 40% запитів при тих же ресурсах. Багатопоточність добре підходить для обробки великого потоку паралельних запитів, дозволяючи обробляти до 60% більше запитів, але вимагає додаткових ресурсів для координації. Оптимізація запитів до бази даних дозволяє обробляти до 40% більше транзакцій, забезпечуючи швидкий доступ до інформації, але потребує коректного налаштування для підвищення ефективності. Забезпечення безпеки є необхідним елементом, який незначно збільшує час відповіді, але забезпечує необхідний захист даних, що знижує продуктивність лише на 10%–15% для забезпечення конфіденційності.

Таблиця 3.2 демонструє порівняльний аналіз методів оптимізації, відображаючи їхні переваги, недоліки і приклади використання в реальних інформаційних системах.

Таблиця 3.2 – Порівняльний аналіз методів оптимізації серверної частини інформаційної системи

Метод оптимізації	**Переваги**	**Недоліки**	**Приклад використання**
1	**2**	**3**	**4**
Кешування	Швидкий доступ до часто використовуваних даних; зменшення навантаження на базу даних	Обмежена пам'ять для кешу; ризик застарілих даних	Інтернет-магазини для кешування даних про товари

Продовження таблиці 3.2

1	2	3	4
Багатопоточність	Обробка паралельних запитів; зменшення часу очікування користувачів	Складна координація потоків; можливість блокування ресурсів	Фінансові платформи для швидкого обслуговування запитів
Оптимізація запитів	Зменшення кількості звернень до бази даних; покращення швидкості вибірки даних	Зниження швидкості при великих обсягах даних; потреба налаштувань ORM	CRM-системи для обробки даних про клієнтів
Забезпечення безпеки	Захист даних від перехоплення; контроль доступу	Підвищення обчислювальних витрат; можливе зниження продуктивності	Банківські системи для захисту платіжних даних

Цей підхід дозволяє вибрати найефективніші методи залежно від специфічних вимог і особливостей серверної частини системи.

3.5 Перспективи подальшої роботи

Майбутні кроки в оптимізації серверної частини інформаційної системи передбачають розробку комплексного підходу до управління ресурсами та обробки запитів, що ґрунтується на впровадженні новітніх технологій розподіленої інфраструктури, автоматизації, та машинного навчання.

Такий підхід здатен підвищити стійкість системи до змінних навантажень і покращити її здатність до масштабування відповідно до потреб користувачів.

Розглянемо розподілене кешування, що дозволяє рівномірно розподіляти дані між кількома вузлами у випадках високого навантаження. У традиційній архітектурі кешування зазвичай відбувається на одному сервері, що при зростанні навантаження може призводити до перевантаження вузла і погіршення продуктивності. Перехід на розподілене кешування дозволяє організувати кілька рівнів зберігання даних, кожен з яких обслуговує певний набір запитів. Це не лише розвантажує основний сервер, але й забезпечує безперебійний доступ до даних, навіть якщо один із вузлів вийде з ладу. Використання Redis Cluster або інших розподілених систем кешування дозволяє масштабувати зберігання відповідно до потреб системи, адаптуючи обсяг кешу до кількості користувачів і характеру запитів [49].

Контейнеризація і автоматизоване керування застосунків надають можливість гнучко масштабувати систему шляхом автоматизованого управління контейнерами. Наприклад, використання Docker у поєднанні з Kubernetes забезпечує високий рівень автономності застосунків. Кожен контейнер містить усі залежності, що потрібні для роботи конкретної частини системи, що дозволяє запускати різні версії сервісів одночасно та ефективно розподіляти навантаження. Kubernetes, як система автоматизованого керування, автоматично розподіляє запити між контейнерами, відстежуючи рівень завантаженості кожного, що дозволяє уникати затримок у відповідях на запити користувачів. Також цей підхід спрощує управління розподіленими сервісами, дозволяючи швидко розгортати додаткові ресурси у випадках раптового збільшення трафіку [50].

Машинне навчання для прогнозування навантаження є ще одним перспективним напрямом оптимізації. Застосування алгоритмів аналізу поведінки користувачів та сезонних патернів дозволяє виявляти пікові періоди активності. Наприклад, якщо система працює для онлайн-торгівлі,

машинне навчання може аналізувати попередні дані і виявляти закономірності, коли спостерігається зростання кількості запитів, наприклад, під час розпродажів або святкових періодів. На основі цих даних система автоматично виділяє більше обчислювальних потужностей перед початком прогнозованого пікового навантаження, що запобігає зниженню продуктивності і забезпечує стабільність навіть при максимальному навантаженні [51].

Інтеграція асинхронного програмування дозволяє підвищити ефективність обробки запитів без зайвого блокування ресурсів. Асинхронні методи дозволяють системі одночасно виконувати завдання введення-виведення, наприклад, отримання даних з бази чи обробку файлів, не очікуючи завершення кожного завдання, як це відбувається у традиційній моделі програмування. Це особливо корисно для систем, що обслуговують великі обсяги користувачів, де швидкість обробки запитів є критичною. Наприклад, у застосунках для потокового передавання контенту асинхронні виклики забезпечують миттєве завантаження відео без затримок для користувачів.

Автоматизація моніторингу та управління безпекою дозволяє своєчасно виявляти і реагувати на потенційні загрози. Впровадження SIEM (Security Information and Event Management) систем дозволяє збирати інформацію про дії в реальному часі, аналізуючи активність і виявляючи аномалії. Наприклад, при підозрілій активності, такій як численні спроби авторизації за короткий проміжок часу, система може автоматично заблокувати доступ і повідомити адміністратора про потенційну атаку.

Це підвищує рівень захищеності даних і забезпечує оперативне реагування, що мінімізує ризик порушення роботи.

Загалом, розвиток оптимізації серверної частини інформаційної системи спрямований на створення комплексної, гнучкої інфраструктури, здатної адаптуватися до потреб і умов, що постійно змінюються.

ВИСНОВКИ

Проведене дослідження щодо оптимізації серверної частини інформаційної системи показало, що для забезпечення її стабільної роботи, продуктивності та безпеки необхідно застосовувати комплексний підхід, який поєднує різні методи та технології. Оптимізація серверної частини є критично важливим завданням, особливо в умовах зростаючих вимог до швидкості, надійності та захищеності даних. Розглянуті та впроваджені методи оптимізації забезпечили вагоме покращення ключових показників системи, що дозволило підвищити її ефективність у реальних умовах експлуатації.

Впровадження кешування стало одним із найефективніших рішень. Завдяки використанню Redis для зберігання часто запитуваних даних, час відповіді скоротився на 80% – з 250 мс до 50 мс, а пропускна здатність зросла на 87,5% (з 80 до 150 запитів за секунду). Це дозволило системі обробляти на 40% більше запитів без збільшення апаратних ресурсів, що особливо важливо для масштабованих систем із великою кількістю користувачів. Крім того, зниження використання процесорних ресурсів (CPU) з 60% до 45% і оперативної пам’яті (RAM) з 70% до 50% свідчить про зниження навантаження на сервер, що дозволяє підтримувати стабільну роботу навіть при пікових навантаженнях.

Використання багатопоточності в середовищі .NET на основі Task Parallel Library дозволило значно збільшити здатність системи обробляти паралельні запити. Час відповіді скоротився на 66,7% – з 300 мс до 100 мс, а пропускна здатність зросла на 85,7%, дозволяючи системі обробляти до 20000 запитів на годину. Це важливо для систем, що піддаються високим навантаженням. Зниження використання CPU з 75% до 50% свідчить про оптимізацію ресурсів, яка сприяє економії ресурсів серверної частини при обробці пікових навантажень. Оптимізація запитів до бази даних, зокрема індексування колонок та впровадження ORM-системи Entity Framework,

забезпечила зниження часу відповіді з 200 мс до 80 мс (скорочення на 60%) і підвищення пропускної здатності до 160 запитів за секунду, що на 77,8% більше, ніж до оптимізації. Це дозволило збільшити обсяг оброблених запитів до 25000 на годину, що є важливим показником для CRM-систем та інших баз даних із високою інтенсивністю запитів. Завдяки зниженню навантаження на CPU з 65% до 50% та RAM з 75% до 55%, вдалося забезпечити ефективнішу обробку транзакцій.

Забезпечення безпеки за допомогою SSL-сертифікатів та OAuth дозволило створити високий рівень захищеності даних, що є особливо важливим для систем, які працюють із конфіденційною інформацією, наприклад, фінансові та медичні системи. Хоча впровадження цих методів дещо збільшило час відповіді (з 180 мс до 150 мс) та використання CPU (з 70% до 65%), це незначне зниження продуктивності виправдане високим рівнем захисту даних. В результаті система змогла обробляти 11000 запитів на годину з шифруванням, що на 10%–15% менше, ніж без шифрування, проте забезпечує надійну захищеність даних.

Перспективи подальшого розвитку серверної частини передбачають використання розподіленої інфраструктури та автоматизації. Впровадження контейнеризації за допомогою Docker та оркестрації Kubernetes дозволить адаптувати серверну частину до змінних умов роботи та підтримувати високу продуктивність. Використання розподіленого кешування та технологій машинного навчання для прогнозування пікових періодів активності забезпечить можливість завчасного збільшення ресурсів і зниження ймовірності збоїв. За оцінками, контейнеризація може підвищити здатність системи обробляти запити на 50%, а автоматизація моніторингу та виявлення загроз підвищить рівень захищеності та зменшить ризик несанкціонованого доступу.

Проведене дослідження дозволило створити гнучку та адаптивну стратегію оптимізації серверної частини, яка відповідає сучасним вимогам до продуктивності та захисту даних.

Отримані результати підтвердили, що оптимізація системи шляхом інтеграції різних технологій та інструментів дозволяє забезпечити стабільну, безпечну та високопродуктивну роботу в умовах високих навантажень і складних умов експлуатації. Ця стратегія є готовою до подальших удосконалень і здатна адаптуватися до майбутніх викликів, забезпечуючи надійність і стабільність системи навіть у довгостроковій перспективі.

Наукова новизна проведеного дослідження полягає у розробці комплексного підходу до оптимізації серверної частини інформаційних систем, який поєднує впровадження сучасних методів кешування, багатопоточності, оптимізації запитів до бази даних, а також інноваційних підходів до забезпечення безпеки. Вперше обґрунтовано взаємозв’язок між використанням Redis для зберігання часто запитуваних даних, застосуванням Task Parallel Library у середовищі .NET та впровадженням ORM-системи для суттєвого зниження часу відповіді та збільшення пропускної здатності системи. Застосовані методи демонструють унікальне поєднання продуктивності, захищеності та ресурсоефективності, що робить отримані результати вагомим внеском у розвиток інформаційних систем нового покоління.

Результати дослідження апробовано у вигляді 2 тез доповідей під час Міжнародного молодіжного форуму «РАДІОЕЛЕКТРОНІКА І МОЛОДЬ У XXI СТОЛІТТІ» [52], XII International scientific and practical conference «PROSPECTIVE DIRECTIONS OF MODERN SCIENCE AND EDUCATION IN THE WORLD» [53].

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Корченко, О. Г., Давиденко, А. М., & Висоцька, О. О. (2019). *Метод автентифікації користувачів інформаційних систем за їх рукописним почерком з багатокроковою корекцією первинних даних: захист інформації* (51 с.).

2. What is HTTPS? URL: https://www.cloudflare.com/ru-ru/learning/ssl/what-is-https/ (дата звернення 06.09.2024).

3. Курс Модуль 3. Java Professional – Лекція: Трирівнева архітектура. URL: https://javarush.com/ua/quests/lectures/ua.questservlets.level14.lecture01 (дата звернення 15.10.2024).

4. Indrasiri, K., & Kuruppu, D. (2020). *gRPC: Up and running: Building cloud native applications with Go and Java for Docker and Kubernetes* (33 с.). O'Reilly Media.

5. HTTP/2: Why you should switch to the new protocol. URL: https://tuthost.ua/en/blog/http-2-why-you-should-switch-to-the-new-protocol/ (дата звернення 06.09.2024).

6. Гурбанов, Т. А. (2022). Архітектурний стиль програмного забезпечення REST: дис. *Національний авіаційний університет.*

7. Василенко М. Ю. Графічний інтерфейс інтелектуальної системи керування трансляцією мережевих адрес на основі протоколу NAT: комплексна кваліфікаційна магістерська робота: 122 Комп'ютерні науки. Суми, 2020. 44 с.

8. Ковальчук О. А. Розробка освітньої платформи на основі технології .Net: кваліфікаційна робота першого (бакалаврського) рівня вищої освіти: 121 інженерія програмного забезпечення. Тернопіль, 2023. 79 с.

9. Мартиненко, О. П. (2021). Концепції десекуляризації, постсекуляризації та множинних сучасностей у подоланні кризи теорії модернізації.

10. Шевченко, Ю. О. (2022). Обробка і аналіз даних з використанням електронних таблиць. Частина І «Обробка даних».

11. Majumder, S., Kar, S., & Pal, T. (2019). Uncertain multi-objective Chinese postman problem. *Soft Computing*, *23*, 11557-11572.

12. Прокопенко, Т. О. (2019). Теорія систем і системний аналіз. *ЧДТУ*.

13. Firebase як бекенд для будь-яких застосунків, та як використовувати Firebase-сервісию. URL: https://dou.ua/forums/topic/44058/ (дата звернення 06.09.2024).

14. Бойко, Ю. П., & Касьянова, Н. В. (2020). Навчально-методичний комплекс дисципліни "Моделювання в цифровій економіці".

15. Павленко, Ю. С. (2022). Пошукова оптимізація, технології та сервіси веб-аналітики: конспект лекцій.

16. Ефективність ключових слів. Як вони впливають на SEO та онлайн-видимість. URL: https://guildofmarketing.com/efektyvnist-kliuchovykh-sliv-yak-vony-vplyvaiut-na-seo-ta-onlain-vydymist/ (дата звернення 06.09.2024).

17. Andrienko, Y. A., & Donetskii, S. U. (2023). Development of an algorithm for constructing the optimal path based on the floor plan for the measuring robot. *International Journal of Open Information Technologies*, *11*(8), 73-78.

18. Lockhart, J. (2015). *Modern PHP: New features and good practices*. "O'Reilly Media, Inc.".

19. Nordbotten, S. (2007). Introduction to development of dynamic web applications.

20. Williams, J. W., Grimm, E. C., Blois, J. L., Charles, D. F., Davis, E. B., Goring, S. J., ... & Takahara, H. (2018). The Neotoma Paleoecology Database, a multiproxy, international, community-curated data resource. *Quaternary Research*, *89*(1), 156-177.

21. Beaird, J., Walker, A., & George, J. (2020). *The principles of beautiful web design*. SitePoint Pty Ltd.

22. Krishnamurthi, S. (2007). *Programming languages: Application and interpretation*. Shriram Krishnamurthi.

23. Aho, A. V., Kernighan, B. W., & Weinberger, P. J. (2023). *The AWK programming language*. Addison-Wesley Professional.

24. Tvoroshenko I., Gorokhovatskyi V., Kobylin O., and Tvoroshenko A. (2023) Application of deep learning methods for recognizing and classifying culinary dishes in images, International Journal of Academic and Applied Research, 7(9), pp. 57-70.

25. Daradkeh Y.I., Gorokhovatskyi V., Tvoroshenko I., and Zeghid M. (2022) Cluster representation of the structural description of images for effective classification, *Computers, Materials & Continua*, 73(3), pp. 6069-6084.

26. Daradkeh Y.I., Gorokhovatskyi V., Tvoroshenko I., and Zeghid M. (2024) Improving the effectiveness of image classification structural methods by compressing the description according to the information content criterion, *Computers, Materials & Continua*, vol. 80, no. 2, pp. 3085-3106.

27. Rauschmayer, A. (2019). *JavaScript for impatient programmers*. McGraw-Hill Education.

28. Tvoroshenko I., and Gorokhovatskyi V. (2022) The Application of Hybrid Intelligence Systems for Dynamic Data Analysis, *International Journal of Engineering and Information Systems*, 6(2), pp. 40-48.

29. Kaur, T. (2019). Cloud Computing: A Study of the Cloud Computing Services. *international Journal for Research in Applied Science & Engineering Technology (IJRASET)*, *7*, 1933-1938.

30. Sen, J., & Mehtab, S. (Eds.). (2020). *Computer and network security*. BoD–Books on Demand.

31. Gorokhovatskyi, V., Tvoroshenko, I., Kobylin, O., & Vlasenko, N. (2023). Search for visual objects by request in the form of a cluster representation for the structural image description, *Advances in Electrical and Electronic Engineering*, 21(1), pp. 19-27.

32. Pomazan V., Tvoroshenko I., and Gorokhovatskyi V. (2023) Handwritten character recognition models based on convolutional neural networks, International Journal of Academic Engineering Research, 7(9), pp. 64-72.

33. Ballad, T., & Confer, W. (2008). *Securing PHP Web Applications* (304 с.). Syngress.

34. Гороховатський В.О., Творошенко І.С., Чмутов Ю.В. (2022) Застосування систем ортогональних функцій для формування простору ознак у методах класифікації зображень, *Сучасні інформаційні системи*, 6(3), С.5-12.

35. Bergmann, S., & Priebsch, S. (2015). *Foundations of PHP for Web Developers* (372 с.). Apress.

36. Pomazan, V., Tvoroshenko, I., & Gorokhovatskyi, V. (2023). Development of an application for recognizing emotions using convolutional neural networks, *International Journal of Academic Information Systems Research*, 7(7), pp. 25-36.

37. Kobylin O., Gorokhovatskyi V., Tvoroshenko I., and Peredrii O. (2020) The application of non-parametric statistics methods in image classifiers based on structural description components, *Telecommunications and Radio Engineering*, 79(10), pp. 855-863.

38. Smirnova, S., & Tezuysal, A. (2022). *MySQL Cookbook*. " O'Reilly Media, Inc.".

39. Олійник, М. С. (2024). Методи пришвидшення завантаження веб-сторінок на основі кешування.

40. Tvoroshenko I., Pomazan V., Gorokhovatskyi V., and Kobylin O. (2023) Application of video data classification models using convolutional neural networks, International Journal of Academic and Applied Research, 7(11), pp. 134-145.

41. Gorokhovatskyi V., Tvoroshenko I., and Yakovleva O. (2024) Transforming image descriptions as a set of descriptors to construct classification features, *Indonesian Journal of Electrical Engineering and Computer Science*, vol. 33, no. 1, pp. 113-125.

42. Бутенко, Т.А. & Сирий В.М. (2020). Інформаційні системи та технології: навч. посібник. Харків

43. Гороховатський В., Передрій О., Творошенко І., Марков Т. (2023) Матриця відстаней для множини компонентів структурного опису як інструмент для створення класифікатора зображень, *Сучасні інформаційні системи*, 7(1), С. 5-13.

44. Yakovleva O., Matúšová S., Tvoroshenko I., and Isaiev Y. (2024) Visitor counting based on video stream analysis from surveillance cameras to solve various business problems, *Verejná správa a regionálny rozvoj ekonómia, manažment a marketing*, XX(1), pp. 67-87.

45. Дробицький, Д. С. (2024). Дослідження та розробка стратегії оптимізації та підвищення продуктивності веб-додатків на базі NodeJS та React з використанням штучного інтелекту.

46. Daradkeh Y.I., Gorokhovatskyi V., Tvoroshenko I., and Al-Dhaifallah M. (2022) Classification of Images Based on a System of Hierarchical Features, *Computers, Materials & Continua*, 72(1), pp. 1785-1797.

47. Daradkeh Y.I., Gorokhovatskyi V., Tvoroshenko I., Gadetska S., and Al-Dhaifallah M. (2023) Statistical data analysis models for determining the relevance of structural image descriptions, IEEE Access, vol. 11, pp. 126938-126949.

48. Дяченко, В. Г. (2023). Засоби захисту даних в серверних застосунках на платформі Node. js.

49. Gorokhovatskyi V., Tvoroshenko I. (2023) Identification of visual objects by the search request. *International scientific symposium «INTELLIGENT SOLUTIONS-S». Computational intelligence (results, problems and perspectives). Decision making theory: proceedings of the international symposium, September 28, 2023, Kyiv-Uzhorod, Ukraine*, pp. 25-27.

50. Gorokhovatskyi V., Tvoroshenko I., Yakovleva O., Hudáková M., and Gorokhovatskyi O. (2024) Application a committee of Kohonen neural networks to training of image classifier based on description of descriptors set, *IEEE Access*, vol. 12, pp. 73376-73385.

51. Мелкозьорова, О., Лєсная, Ю., & Малахов, С. (2022). Особливості інтеграції систем захисту від несанкціонованих дій в сучасних інформаційних системах. *Комп'ютерні науки та кібербезпека*, (1), 39-44.

52. Караконстантин, Д. (2024). Особливості Підходів Розроблення Складних Систем.

53. Karakonstantyn, D., & Tvoroshenko, I. (2024) About The Issue Of Optimization The Performance Of The Server Part Of The Information System.