Міністерство освіти і науки України
Харківський національний університет імені В. Н. Каразіна
Навчально-науковий інститут комп'ютерних наук та штучного інтелекту
Кафедра комп'ютерних систем та робототехніки

До захисту допущено
Кафедрою комп'ютерних систем та робототехніки
протокол № __ від __ червня 2026 р.

завідувач кафедри _____ Максим ХРУСЛОВ
(підпис)
«___»_____ 2026 р.

Кваліфікаційна робота

здобувача першого (бакалаврського) рівня вищої освіти

«СИСТЕМА РЕЗЕРВНОГО КОПІЮВАННЯ ТА ВІДНОВЛЕННЯ БАЗ ДАНИХ З ВИКОРИСТАННЯМ КОНТЕЙНЕРНИХ ТЕХНОЛОГІЙ»

Спеціальність ***123 – Комп'ютерна інженерія.***

Освітня програма ***Комп'ютерна інженерія***

Виконавець _____ **Владислав МАСЛЕННИКОВ**
(підпис)

Науковий керівник _____ **Ольга ЧУБ**
(підпис)

Харків – 2026

АНОТАЦІЇ

Пояснювальна записка до кваліфікаційної роботи складається зі вступу, трьох розділів, висновків, переліку використаних джерел та чотирьох додатків. Загальний обсяг роботи становить 66 сторінок, з яких 40 сторінок основної частини, що містять 16 рисунків, 9 таблиць, 20 найменувань у списку використаних джерел.

Об'єкт дослідження – процеси резервного копіювання та відновлення баз даних у центрах обробки даних.

Предмет дослідження – архітектурні підходи та програмні засоби для автоматизації резервного копіювання баз даних з використанням контейнерних технологій Docker.

Мета дослідження – підвищення надійності зберігання даних шляхом розробки веб-орієнтованої системи резервного копіювання, яка використовує Docker-контейнери для ізоляції процесів бекапу та інтегрована з DCIM-модулем управління інфраструктурою дата-центру.

Проблема, яку вирішує кваліфікаційна робота – відсутність безкоштовного, простого у розгортанні веб-рішення для централізованого резервного копіювання баз даних (MySQL) з використанням контейнерної ізоляції, яке б також забезпечувало управління інфраструктурою дата-центру (дата-центри, стійки, пристрої, інциденти).

Область застосування – інформаційні системи управління підприємствами, IT-відділи, центри обробки даних, навчальний процес при вивченні дисциплін «Адміністрування баз даних» та «Системи резервного копіювання».

Ключові слова: *інформаційні технології, резервне копіювання, відновлення даних, база даних, MySQL, контейнерні технології, Docker, PHP, веб-інтерфейс, DCIM, REST API, Chart.js.*

ABSTRACT

The explanatory note to the qualification work consists of an introduction, three chapters, conclusions, a list of references and four appendices. The total volume of the work is 66 pages, of which 40 pages are the main part, containing 16 figures, 9 tables, 20 references.

Object of research – processes of backup and restoration of databases in data processing centers.

Subject of research – architectural approaches and software tools for automating database backup using Docker container technologies.

Purpose of research – to increase data storage reliability by developing a web-oriented backup system that uses Docker containers to isolate backup processes and is integrated with a DCIM module for data center infrastructure management.

The problem solved by the qualification work is the lack of a free, easy-to-deploy web solution for centralized backup of databases (MySQL) using container isolation, which also provides data center infrastructure management (data centers, racks, devices, incidents).

Scope of application – enterprise information systems, IT departments, data processing centers, educational process in teaching the disciplines "Database Administration" and "Backup Systems".

Keywords: *information technologies, backup, data restore, database, MySQL, container technologies, Docker, PHP, web interface, DCIM, REST API, Chart.js.*

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ ТА УМОВНИХ ПОЗНАЧЕНЬ 5

ВСТУП 7

РОЗДІЛ 1. ТЕОРЕТИЧНІ ОСНОВИ РЕЗЕРВНОГО КОПІЮВАННЯ ТА КОНТЕЙНЕРНИХ ТЕХНОЛОГІЙ 10

1.1. Актуальність автоматизації резервного копіювання баз даних 10

1.2. Класифікація методів резервного копіювання 11

1.3. Контейнерні технології Docker для ізоляції завдань бекапу 12

1.4. Огляд існуючих програмних рішень 14

Висновки до розділу 1 15

РОЗДІЛ 2. ПРОЄКТУВАННЯ СИСТЕМИ 16

2.1. Функціональні та нефункціональні вимоги 16

2.2. Архітектура системи 18

2.3. Проєктування бази даних DCIM та бекап-журналу 21

2.4. Проєктування веб-інтерфейсу 25

Висновки до розділу 2 28

РОЗДІЛ 3. РЕАЛІЗАЦІЯ ТА ТЕСТУВАННЯ СИСТЕМИ 29

3.1. Реалізація серверної частини (PHP API) 29

3.2. Реалізація механізму резервного копіювання з Docker 31

3.3. Реалізація веб-інтерфейсу 33

3.4. Тестування системи 36

3.5. Рекомендації щодо вдосконалення 38

Висновки до розділу 3 39

ВИСНОВКИ 41

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ 43

ДОДАТКИ 45

ПЕРЕЛІК СКОРОЧЕНЬ ТА УМОВНИХ ПОЗНАЧЕНЬ

API (Application Programming Interface) — програмний інтерфейс застосунку.

AWS (Amazon Web Services) — хмарна платформа Amazon

ACID (Atomicity, Consistency, Isolation, Durability) — атомарність, узгодженість, ізоляція, довговічність.

CPU (Central Processing Unit) — центральний процесор.

CRUD (Create, Read, Update, Delete) — створення, читання, оновлення, видалення.

CSS (Cascading Style Sheets) — каскадні таблиці стилів.

DCIM (Data Center Infrastructure Management) — управління інфраструктурою дата-центру.

DNS (Domain Name System) — система доменних імен.

DTO (Data Transfer Object) — об'єкт передачі даних.

FOSS (Free and Open-Source Software) — вільне та відкрите програмне забезпечення.

FTP (File Transfer Protocol) — протокол передачі файлів.

GDPR (General Data Protection Regulation) — Загальний регламент захисту даних.

HTML (Hyper Text Markup Language) — мова розмітки гіпертексту.

HTTP (Hyper Text Transfer Protocol) — протокол передачі гіпертексту.

HTTPS (HyperText Transfer Protocol Secure) — захищений протокол передачі гіпертексту.

IDE (Integrated Development Environment) — інтегроване середовище розробки.

IP (Internet Protocol) — міжмережевий протокол.

ISO (International Organization for Standardization) — Міжнародна організація зі стандартизації.

JPA (Jakarta Persistence API) — специфікація Java для об'єктно-реляційного відображення.

JS (JavaScript) — мова програмування JavaScript.

JSON (JavaScript Object Notation) — нотація об'єктів JavaScript.

JWT (JSON Web Token) — JSON веб-токен.

MVC (Model-View-Controller) — модель-подання-контролер.

ORM (Object-Relational Mapping) — об'єктно-реляційне відображення.

OS (Operating System) — операційна система.

OWASP (Open Web Application Security Project) — Відкритий проєкт з безпеки веб-застосунків.

PDO PHP Data Objects — об'єкти даних PHP.

PHP Hypertext Preprocessor — препроцесор гіпертексту.

PWA Progressive Web App — прогресивний веб-застосунок.

RAM Random Access Memory — оперативна пам'ять з довільним доступом.

REST (Representational State Transfer) — передача стану представлення.

RPO (Recovery Point Objective) — цільова точка відновлення.

RTO (Recovery Time Objective) — цільовий час відновлення.

S3 (Simple Storage Service) — простий сервіс зберігання (Amazon).

SPA (Single Page Application) — односторінковий застосунок.

SQL (Structured Query Language) — структурована мова запитів.

SSD (Solid State Drive) — твердотільний накопичувач.

UI (User Interface) — інтерфейс користувача.

UML (Unified Modeling Language) — уніфікована мова моделювання.

UPS (Uninterruptible Power Supply) — джерело безперебійного живлення.

URL (Uniform Resource Locator) — уніфікований локатор ресурсів.

USB (Universal Serial Bus) — універсальна послідовна шина.

WSL (Windows Subsystem for Linux) — підсистема Windows для Linux.

ВСТУП

У сучасному цифровому світі дані стали одним із найцінніших активів підприємства. Згідно з дослідженням компанії Gartner, середня вартість однієї години простою інформаційної системи для підприємства середнього розміру становить від 50 000 до 100 000 доларів США [1]. Основними причинами втрати даних є апаратні збої (близько 45 % випадків), людські помилки (близько 30 %), програмні помилки (близько 15 %) та кібератаки, зокрема віруси-шифрувальники (близько 10 %) [2].

Особливої гостроти проблема надійного зберігання даних набуває для центрів обробки даних (ЦОД) та виробничих підприємств, де бази даних містять критичну інформацію про технологічні процеси, складські запаси, замовлення клієнтів та фінансові операції. Вихід із ладу системи управління базами даних (СУБД) навіть на кілька годин може призвести до зупинки виробництва та значних фінансових збитків.

Традиційні засоби резервного копіювання (phpMyAdmin, phpPgAdmin) дозволяють створювати резервні копії вручну, але не підтримують автоматизацію за розкладом та централізоване керування. Комерційні системи (Veeam, Acronis) є потужними, але дорогими та складними у налаштуванні. Крім того, більшість існуючих рішень не використовують контейнерну ізоляцію (Docker) для виконання завдань бекапу, що дозволяє уникнути конфліктів залежностей та підвищити безпеку.

Таким чином, існує потреба у створенні власної веб-системи резервного копіювання, яка б об'єднувала:

-централізоване веб-керування;

-використання Docker-контейнерів для ізоляції процесів бекапу;

-інтеграцію з системою управління інфраструктурою дата-центру (DCIM);

-безкоштовність та простоту розгортання.

Мета роботи – підвищення надійності зберігання даних шляхом розробки веб-орієнтованої системи резервного копіювання, яка використовує

Docker-контейнери для ізоляції процесів бекапу та інтегрована з DCIM-модулем управління інфраструктурою дата-центру.

Завдання дослідження:

1. Провести аналіз сучасних методів резервного копіювання баз даних (повний, інкрементний, диференційний) та обґрунтувати вибір методу для реалізації.

2. Дослідити контейнерні технології Docker як засіб ізоляції завдань бекапу.

3. Виконати огляд існуючих програмних рішень для резервного копіювання та визначити їх переваги та недоліки.

4. Сформулювати функціональні та нефункціональні вимоги до системи, що розробляється.

5. Розробити архітектуру системи (клієнт-серверна, REST API на PHP, Docker-контейнери, база даних MySQL).

6. Спроектувати реляційну базу даних для DCIM-модуля (дата-центри, кімнати, ряди, стійки, пристрої, інциденти) та журналу бекапів.

7. Реалізувати серверну частину (PHP API) для управління дата-центрами, пристроями, стійками, інцидентами та бекапами.

8. Реалізувати механізм резервного копіювання та відновлення, який використовує Docker-контейнери для запуску `mysqldump` та `mysql`.

9. Розробити веб-інтерфейс (HTML/CSS/JS) з використанням бібліотеки Chart.js для візуалізації статистики.

10. Провести тестування системи (модульне, інтеграційне, навантажувальне) та оцінити відповідність вимогам.

11. Розробити рекомендації щодо подальшого вдосконалення системи.

Об'єкт дослідження – процеси резервного копіювання та відновлення баз даних у центрах обробки даних.

Предмет дослідження – архітектурні підходи та програмні засоби для автоматизації резервного копіювання баз даних з використанням контейнерних технологій Docker.

Методи дослідження: аналіз літературних джерел та існуючих програмних рішень, системний аналіз вимог, об'єктно-орієнтоване проєктування (UML-діаграми), реляційне моделювання даних (ER-діаграми), клієнт-серверна розробка (PHP, PDO, REST API, HTML/CSS/JS), контейнеризація (Docker), тестування програмного забезпечення (Postman, k6).

Наукова новизна отриманих результатів полягає в обґрунтуванні та практичній реалізації архітектури, в якій завдання резервного копіювання виконуються в одноразових Docker-контейнерах, що забезпечує повну ізоляцію процесу бекапу від основної СУБД та хост-системи.

Практичне значення отриманих результатів полягає у створенні готової до впровадження веб-системи, яка надає зручний інтерфейс для керування резервним копіюванням баз даних та управління інфраструктурою дата-центру.

Робота складається зі вступу, трьох розділів, висновків до кожного розділу, загальних висновків, списку використаних джерел (20 найменувань) та 4 додатків.

РОЗДІЛ 1.
ТЕОРЕТИЧНІ ОСНОВИ РЕЗЕРВНОГО КОПІЮВАННЯ ТА КОНТЕЙНЕРНИХ ТЕХНОЛОГІЙ

1.1. Актуальність автоматизації резервного копіювання баз даних

У сучасних умовах обсяги даних, що генеруються підприємствами, постійно зростають. Згідно з дослідженням IDC, до 2025 року світовий обсяг даних досягне 175 зетабайт [3]. Ручне адміністрування резервного копіювання стає неефективним з наступних причин.

1) Людський фактор. Оператор може забути виконати резервне копіювання, зробити помилку в команді або не перевірити цілісність створеної копії. Згідно зі статистикою, близько 30 % випадків втрати даних пов'язані саме з людськими помилками [2].

2) Час реакції. При виникненні збою (апаратна відмова, атака ransomware, програмна помилка) важлива кожна хвилина. Автоматизована система резервного копіювання дозволяє відновити дані значно швидше, ніж ручне втручання.

3) Масштабування. При зростанні кількості серверів баз даних до десятків і сотень ручне керування резервним копіюванням стає практично неможливим. Централізована система дозволяє керувати всіма бекапами з єдиного веб-інтерфейсу.

4) Відповідність вимогам стандартів. Багато міжнародних стандартів (ISO 27001, PCI DSS, GDPR) вимагають наявності автоматизованого резервного копіювання з чітко визначеними періодичністю та правилами зберігання.

Для кількісної оцінки вимог до систем резервного копіювання використовуються дві ключові метрики [4].

RPO (Recovery Point Objective) – максимально допустимий період часу, протягом якого дані можуть бути втрачені. Наприклад, якщо резервна копія створюється раз на добу о 02:00, то при збої о 14:00 RPO становитиме 12 годин.

Для фінансових систем RPO може бути 0 (відсутність втрат), для внутрішнього обліку – до 1 години, для архівних даних – до 24 годин.

RTO (Recovery Time Objective) – максимально допустимий час, за який система має бути повністю відновлена після збою. Для інтернет-магазину RTO може становити 1–2 години, для внутрішньої бухгалтерської системи – 1 робочий день.

Автоматизація резервного копіювання дозволяє досягти значень RPO від кількох хвилин до доби та RTO від годин до днів залежно від обраного методу бекапу та доступних ресурсів.

1.2. Класифікація методів резервного копіювання

У сучасній практиці адміністрування баз даних використовується кілька основних методів резервного копіювання.

1) Повне резервне копіювання (Full Backup). У цьому методі створюється повна копія всієї бази даних (або обраного набору таблиць) на момент запуску процесу бекапу. Для MySQL використовується команда `mysqldump`, для PostgreSQL – `pg_dump`. Перевагами повного бекапу є простота відновлення (достатньо одного файлу) та незалежність від інших бекапів. Недоліками – великий розмір та тривалий час створення.

2) Інкрементне резервне копіювання (Incremental Backup). При цьому методі копіюються лише зміни, що відбулися з моменту останнього бекапу будь-якого типу (повного або попереднього інкрементного). Таким чином створюється ланцюжок бекапів: `Full → Inc1 → Inc2 → Inc3 → ...`. Переваги – мінімальний розмір та швидке створення. Недоліки – складне та повільне відновлення (потрібно послідовно застосувати всі інкременти), а також високий ризик пошкодження ланцюжка.

3) Диференційне резервне копіювання (Differential Backup). При цьому методі копіюються всі зміни, що відбулися з моменту останнього повного бекапу, незалежно від кількості проміжних диференційних бекапів. Відновлення

вимагає тільки повний бекап та останній диференційний бекап, що є компромісом між розміром та швидкістю відновлення.

Порівняльний аналіз методів. У таблиці 1.1 наведено основні характеристики кожного методу.

Таблиця 1.1

Порівняння методів резервного копіювання

Характеристика	Повний	Інкрементний	Диференційний
Розмір бекапу	Максимальний	Мінімальний	Середній (зростає)
Час створення	Великий	Малий	Середній
Час відновлення	Малий	Великий	Середній
Складність реалізації	Низька	Висока	Середня

Для реалізації в даній роботі обрано повне резервне копіювання з наступних причин:

- простота реалізації (не потребує аналізу логів змін);
- надійність (для відновлення достатньо одного файлу);
- можливість легкого розширення в майбутніх версіях.

1.3. Контейнерні технології Docker для ізоляції завдань бекапу

Традиційна віртуалізація (VMware, VirtualBox, KVM) передбачає запуск гостьової операційної системи поверх гіпервізора. Кожна віртуальна машина має власне ядро, що призводить до значних накладних витрат ресурсів. Контейнеризація (Docker) працює на рівні ядра хостової операційної системи, використовуючи механізми ізоляції просторів імен (namespaces) та контролю груп ресурсів (cgroups).

Основні переваги Docker-контейнерів перед віртуальними машинами наведено в таблиці 1.2.

Таблиця 1.2

Порівняння віртуальних машин та Docker-контейнерів

Характеристика	Віртуальна машина	Docker-контейнер
Час запуску	Хвилини	Секунди
Типовий розмір	1–10 ГБ	10–100 МБ
Накладні витрати CPU	5–15 %	1–2 %
Накладні витрати RAM	1–2 ГБ	10–50 МБ
Рівень ізоляції	Повний (гіпервізор)	Частковий (namespaces)

Архітектура Docker складається з наступних ключових компонентів [5]:

- Docker Engine – фоновий процес, який управляє контейнерами, образами, томами та мережами.
- Docker Image (образ) – незмінний шаблон, який містить операційну систему, програмне забезпечення та залежності.
- Docker Container (контейнер) – запущений екземпляр образу.
- Docker Volume (том) – механізм зберігання даних поза контейнером.
- Docker Network (мережа) – віртуальна мережа для взаємодії контейнерів.

Використання Docker для резервного копіювання забезпечує такі переваги.

1) Ізоляція процесу бекапу. Контейнер виконується в ізольованому середовищі, не впливаючи на хост-систему та основні сервіси. Навіть якщо процес бекапу споживає багато ресурсів, це не впливає на продуктивність основної бази даних.

2) Відсутність необхідності встановлення клієнтів СУБД. Не потрібно встановлювати MySQL-клієнт, PostgreSQL-клієнт або інші залежності на сервері. Достатньо мати встановлений Docker та відповідний образ (наприклад, mysql:8.0).

3) Уніфікація інтерфейсу. Незалежно від типу СУБД, команда запуску бекапу виглядає однаково: docker run --rm mysql:8.0 mysqldump .... Це спрощує автоматизацію.

4) Легке масштабування. Можна запустити кілька паралельних контейнерів для бекапу різних баз даних одночасно.

5) Відтворюваність. Один і той самий Docker-образ можна використовувати на будь-якому сервері, незалежно від версії операційної системи та встановлених пакетів.

1.4. Огляд існуючих програмних рішень

Вбудовані засоби СУБД. MySQL надає утиліту mysqldump, PostgreSQL – pg_dump. Ці засоби є надійними та швидкими, але вони не мають веб-інтерфейсу та не підтримують централізоване керування бекапами багатьох баз даних.

Веб-інструменти. PhpMyAdmin дозволяє створювати резервні копії через веб-інтерфейс, але підтримує тільки MySQL та не має автоматизації за розкладом. Adminer підтримує багато СУБД, але також не має планувальника. pgAdmin є потужним інструментом для PostgreSQL, але не підтримує MySQL.

Пропрієтарні корпоративні системи. Veeam Backup & Replication та Acronis Cyber Backup є дуже потужними, але вони платні, складні у налаштуванні та вимагають ліцензування за кількістю віртуальних машин або об'ємом даних.

Порівняльний аналіз. Результати порівняння наведено в таблиці 1.3.

Таблиця 1.3

Порівняльний аналіз систем резервного копіювання

Рішення	Веб-інтерфейс	Автоматизація	Різні СУБД	Контейнеризація	Безкоштовно
mysqldump	Ні	Частково	Ні	Ні	Так
phpMyAdmin	Так	Ні	Ні	Ні	Так
Adminer	Так	Ні	Так	Ні	Так
pgAdmin	Так	Частково	Ні	Ні	Так
Veeam	Так	Так	Так	Частково	Ні
Acronis	Так	Так	Так	Частково	Ні
Розроблена система	Так	Так	Так	Так	Так

Висновок. Жодне з безкоштовних рішень не поєднує всі необхідні характеристики: зручний веб-інтерфейс, автоматизацію, підтримку різних СУБД та контейнерну ізоляцію. Це підтверджує актуальність та доцільність розробки власної системи.

Висновки до розділу 1

Автоматизація резервного копіювання є необхідною умовою для забезпечення надійності зберігання даних у сучасних центрах обробки даних.

Існує три основні методи бекапу: повний, інкрементний та диференційний. Для реалізації було обрано повний бекап як найбільш надійний та простий у реалізації.

Контейнерні технології Docker забезпечують ізоляцію процесів бекапу, відсутність конфліктів залежностей та легке масштабування.

Порівняльний аналіз існуючих рішень (phpMyAdmin, Adminer, Veeam, Acronis) показав, що жодне з безкоштовних рішень не поєднує всі необхідні характеристики, що обґрунтовує доцільність розробки власної системи.

РОЗДІЛ 2.
ПРОЄКТУВАННЯ СИСТЕМИ

2.1. Функціональні та нефункціональні вимоги

Функціональні вимоги до DCIM-модуля:

1) Управління дата-центрами. Система повинна забезпечувати створення, читання, оновлення та видалення (CRUD) дата-центрів з полями: назва, код, місто, країна, рівень Tier, статус, контактна особа.

2) Управління кімнатами, рядами та стійками. Система також повинна забезпечувати CRUD-операції для кімнат (прив'язка до дата-центру, поверх, площа, тип охолодження), рядів стійок (прив'язка до кімнати, номер ряду, місткість) та стійок (прив'язка до ряду, позиція, висота в юнітах, статус).

3) Управління пристроями. Система має підтримувати CRUD-операції для пристроїв (назва, модель, тип, IP-адреса, позиція в стійці, статус). Користувач повинен мати можливість фільтрувати пристрої за статусом (активний, вимкнений, збій, запланований) та шукати за назвою або IP-адресою.

4) Управління виробниками. Система повинна забезпечувати CRUD-операції для виробників обладнання (назва, вебсайт, телефон та email підтримки).

5) Управління мережевою інфраструктурою. Також вимагається від системи можливість створення та перегляд VLAN, портів пристроїв та кабельних з'єднань.

6) Управління інцидентами. Система повинна забезпечувати створення інцидентів (тип, серйозність, опис, пристрій, хто повідомив) та їх вирішення (введення рішення, хто вирішив). Система повинна відображати список активних інцидентів з пріоритетом (критичний, високий, середній, низький).

7) Візуалізація статистики. Система повинна відображати на дашборді кількість дата-центрів, пристроїв (активних та загалом), стійок, інцидентів, а також графіки статусів та типів пристроїв за допомогою бібліотеки Chart.js.

Функціональні вимоги до підсистеми резервного копіювання:

1) Створення резервних копій. Система повинна забезпечувати створення повної резервної копії бази даних MySQL через веб-інтерфейс. Створення бекапу повинно виконуватися в Docker-контейнері з використанням команди mysqldump. Після створення бекапу система повинна зберігати інформацію про нього в базі даних (час початку та закінчення, розмір, тривалість) та зберігати файл .sql на диску.

2) Відновлення з резервних копій. Система також повинна забезпечувати відновлення бази даних з обраної резервної копії через Docker-контейнер з використанням команди mysql. Перед відновленням система повинна вимагати підтвердження користувача через модальне вікно з попередженням про перезапис даних.

3) Скачування резервних копій. Користувач повинен мати можливість скачати резервну копію у форматі SQL (оригінальний дамп) та у форматі JSON (структуровані дані для аналітики).

4) Перегляд історії бекапів. Система повинна відображати таблицю з історією всіх бекапів з колонками: ID, назва бази даних, статус, розмір, час початку, тривалість, кількість таблиць та записів. Користувач повинен мати можливість фільтрувати бекапи за статусом (виконано, помилка, виконується) та шукати за назвою бази даних.

6) Статистика бекапів. Система повинна відображати на дашборді бекапів кількість бекапів за останні 30 днів (загальну, успішних, невдалих) та дату останнього успішного бекапу.

Нефункціональні вимоги:

1) Продуктивність. Час відповіді API (отримання списку бекапів, пристроїв, інцидентів) повинен становити не більше 1 секунди при одночасній роботі до 500 користувачів. Завантаження центрального процесора під час виконання бекапу не повинно перевищувати 50 % на 2-ядерному процесорі.

2) Безпека. Система повинна бути захищена від SQL-ін'єкцій шляхом використання підготовлених запитів PDO (PHP Data Objects). Дані для

підключення до бази даних (пароль) не повинні зберігатися в коді. Docker-контейнер для бекапу повинен використовувати параметр --rm для автоматичного видалення після завершення.

3) Надійність. Система повинна коректно обробляти помилки підключення до бази даних. При помилці створення бекапу система повинна зберігати деталі помилки в полі error_message таблиці backup_jobs.

4) Ергономіка. Веб-інтерфейс повинен бути адаптивним для використання на різних пристроях. Навігація повинна бути інтуїтивно зрозумілою (бічна панель з пунктами меню). Форми повинні мати валідацію з наочними повідомленнями про помилки.

2.2. Архітектура системи

Розроблена система має трирівневу клієнт-серверну архітектуру.

Рівень клієнта (Frontend). Веб-браузер користувача завантажує HTML/CSS/JS-файли. JavaScript-код (з використанням Fetch API) надсилає асинхронні HTTP-запити до серверної частини. Для візуалізації даних використовується бібліотека Chart.js.

Рівень сервера (Backend). Веб-сервер Apache обробляє HTTP-запити та запускає PHP-скрипти. Кожен PHP-файл реалізує REST API для певної сутності (дата-центри, пристрої, бекапи тощо). Для роботи з базою даних використовується клас Database з PDO. Для виконання завдань бекапу сервер викликає Docker-контейнери.

Рівень даних (Data Layer). Система управління базами даних MySQL 8.0 зберігає інформацію про інфраструктуру дата-центру (таблиці data_centers, devices, racks, incidents) та історію бекапів (таблиці backup_jobs, backup_data). Файлова система сервера зберігає SQL-дамп резервних копій у каталозі backups/.

Контейнерний рівень (Container Layer). Docker Engine запускає тимчасові контейнери на основі образу mysql:8.0 для виконання mysqldump (створення

бекапу) та mysql (відновлення з бекапу). Контейнери мають доступ до каталогу backups/ через змонтований том.

На рисунку 2.1 наведено схему взаємодії основних компонентів системи.

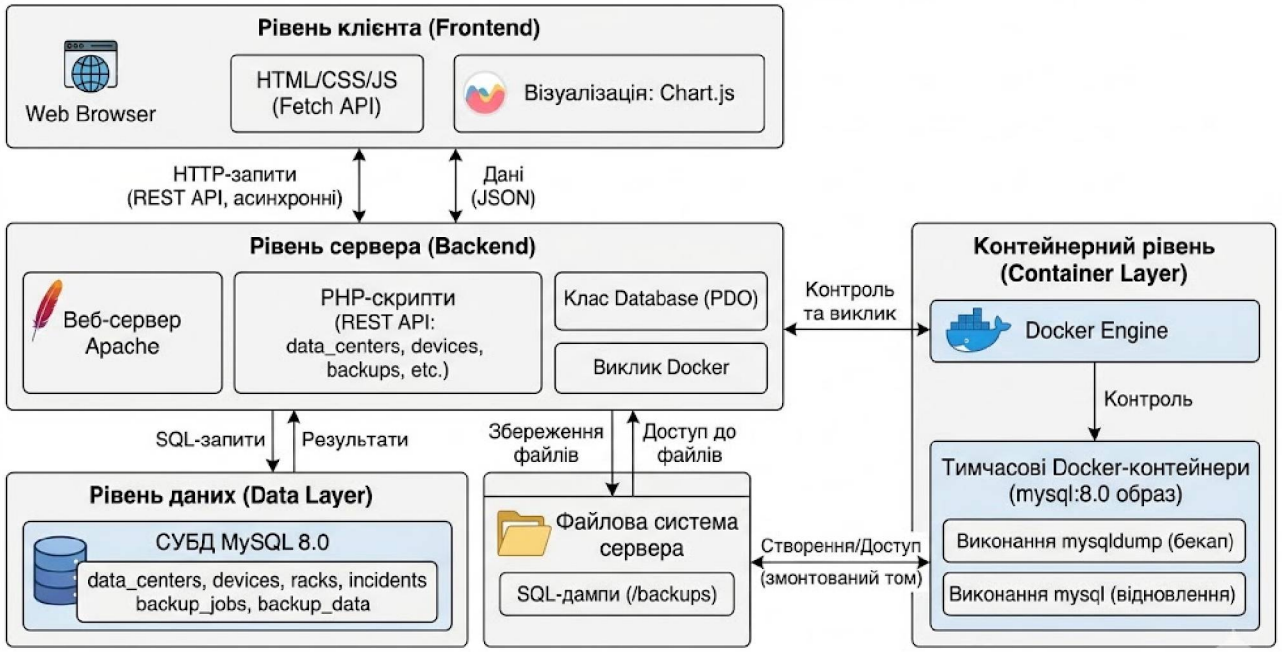


Рисунок 2.1 - Схема архітектури системи.

Основні API - ендпоінти. У таблиці 2.1 наведено перелік основних API - ендпоінтів системи.

Таблиця 2.1

Основні API - ендпоінти системи

Endpoint	Методи HTTP	Призначення
/api/backup.php	GET, POST, DELETE	Створення, відновлення, видалення бекапів
/api/devices.php	GET, POST, PUT, DELETE	CRUD для пристроїв
/api/incident.php	GET, POST, PUT, DELETE	CRUD для інцидентів
/api/racks.php	GET, POST, PUT, DELETE	CRUD для стійок

Продовження таблиці 2.1

/api/datacenters.php	GET, POST, PUT, DELETE	CRUD для дата-центрів
/api/manufacturers.php	GET, POST, PUT, DELETE	CRUD для виробників
/api/network.php	GET, POST	Управління VLAN, портами, з'єднаннями
/api/dashboard.php	GET	Отримання статистики для дашборду

Структура веб-сторінок. У таблиці 2.2 наведено перелік HTML-сторінок системи.

Таблиця 2.2

Структура веб-інтерфейсу

Сторінка	Файл	Призначення
Дашборд	index.html	Головна сторінка з картками та графіками
Дата-центри	datacenters.html	Управління дата-центрами, кімнатами, рядами
Пристрої	devices.html	CRUD для пристроїв, фільтри, пошук
Стійки	racks.html	Управління стійками, перегляд заповнення
Виробники	manufacturers.html	CRUD для виробників
Мережа	network.html	Управління VLAN, портами, з'єднаннями
Інциденти	incidents.html	Створення та вирішення інцидентів
Бекапи	backups.html	Управління резервним копіюванням

2.3. Проєктування бази даних DCIM та бекап-журналу

База даних dcim_db складається з 39 таблиць, які можна розділити на чотири групи: DCIM-таблиці (фізична інфраструктура), моніторинг та метрики, мережеві таблиці та таблиці резервного копіювання.

Логічна модель даних. База даних складається з наступних груп таблиць.

Група 1. Фізична інфраструктура (DCIM):

- data_centers – дата-центри.
- server_rooms – кімнати/зали в дата-центрі.
- rack_rows – ряди стійок.
- racks – стійки.
- manufacturers – виробники обладнання.
- device_types – типи пристроїв (моделі).
- devices – конкретні пристрої.
- power_devices – джерела живлення (PDU, UPS).
- lifecycle_events – події життєвого циклу пристроїв.
- maintenance_schedules – графіки технічного обслуговування.
- incidents – інциденти.
- alerts – сповіщення.

Група 2. Мережева інфраструктура:

- device_ports – порти пристроїв.
- cable_connections – кабельні з'єднання.
- vlans – віртуальні локальні мережі.
- port_vlans – прив'язка портів до VLAN.

Група 3. Моніторинг та метрики:

- device_metrics – метрики пристроїв (CPU, пам'ять).
- environment_readings – показники навколишнього середовища (температура, вологість).
- device_configurations – конфігурації пристроїв.
- config_change_history – історія змін конфігурацій.
- software_versions – версії програмного забезпечення.

Група 4. Резервне копіювання та контейнерні технології:

- container_technologies – контейнерні технології (Docker, Podman, Kubernetes).
- backup_containers – контейнери для виконання бекапів.
- database_types – типи СУБД (MySQL, PostgreSQL, MongoDB).
- database_instances – інстанси СУБД.
- databases_list – конкретні бази даних.
- backup_policies – політики бекапу.
- backup_jobs – завдання бекапу.
- backup_data – збережені дані таблиць у форматі JSON.
- backup_storage – сховища бекапів.
- stored_backups – збережені бекапи.
- recovery_plans – плани відновлення.
- recovery_steps – кроки відновлення.
- backup_restores – історія відновлень.
- dr_tests – тести відновлення.
- dr_simulations – симуляції інцидентів.

DCIM-таблиці. У таблиці 2.3 наведено структуру основних DCIM-таблиць.

Таблиця 2.3

Основні DCIM-таблиці

Назва таблиці	Призначення	Ключові поля
data_centers	Дата-центри	id, name, code, city, tier_level, status
server_rooms	Кімнати в дата-центрі	id, data_center_id, name, floor_number, area_sqm
rack_rows	Ряди стійок	id, server_room_id, name, row_number, rack_capacity
racks	Стійки	id, rack_row_id, name, position_in_row, height_units, status

Продовження таблиці 2.3

manufacturers	Виробники обладнання	id, name, website, support_phone, support_email
device_types	Типи пристроїв (моделі)	id, manufacturer_id, model_name, device_role, height_units
devices	Конкретні пристрої	id, device_type_id, rack_id, name, management_ip, status
incidents	Інциденти	id, device_id, incident_type, severity, status, description, resolution

Створення таблиці devices (див. рис. 2.2)

```
CREATE TABLE devices (
    id INT AUTO_INCREMENT PRIMARY KEY,
    device_type_id INT NOT NULL,
    rack_id INT,
    name VARCHAR(100) NOT NULL,
    serial_number VARCHAR(100) UNIQUE,
    asset_tag VARCHAR(50) UNIQUE,
    position_in_rack INT COMMENT 'U позиція від низу',
    face ENUM('front', 'rear') DEFAULT 'front',
    status ENUM('active', 'offline', 'planned', 'staged', 'failed', 'decommissioning') DEFAULT 'acti
ve',
    management_ip VARCHAR(45),
    hostname VARCHAR(255),
    notes TEXT,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    FOREIGN KEY (device_type_id) REFERENCES device_types(id) ON DELETE CASCADE,
    FOREIGN KEY (rack_id) REFERENCES racks(id) ON DELETE SET NULL,
    INDEX idx_device_status (status),
    INDEX idx_device_rack (rack_id, position_in_rack)
```

Рисунок 2.2 - Лістинг коду створення таблиці devices.

Таблиці резервного копіювання. У таблиці 2.4 наведено структуру таблиць для зберігання інформації про бекапи.

Таблиця 2.4

Таблиці резервного копіювання

Назва таблиці	Призначення	Ключові поля
backup_policies	Політики бекапу	id, policy_name, backup_type, frequency, retention_days
backup_jobs	Завдання бекапу	id, backup_policy_id, database_name, job_status, start_time, end_time, backup_size_mb, backup_file_name, duration_seconds, error_message
backup_data	Дані бекапу (JSON)	id, backup_job_id, table_name, table_data, row_count

Створення таблиці backup_jobs (див. рис. 2.3)

```
CREATE TABLE backup_jobs (
    id INT AUTO_INCREMENT PRIMARY KEY,
    backup_policy_id INT NOT NULL,
    database_name VARCHAR(100) NOT NULL,
    job_status ENUM('pending', 'running', 'completed', 'failed') DEFAULT 'pending',
    start_time DATETIME,
    end_time DATETIME,
    backup_size_mb DECIMAL(10,2),
    backup_file_name VARCHAR(255),
    duration_seconds INT,
    error_message TEXT,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    FOREIGN KEY (backup_policy_id) REFERENCES backup_policies(id) ON DELETE CASCADE,
    INDEX idx_backup_status (job_status),
    INDEX idx_backup_time (start_time)
);
```

Рисунок 2.3 - Лістинг коду створення таблиці backup_jobs.

Створення таблиці backup_data (див. рис. 2.4)

```
CREATE TABLE `backup_data` (
  `id` int(11) NOT NULL AUTO_INCREMENT,
  `backup_job_id` int(11) NOT NULL,
  `table_name` varchar(100) NOT NULL,
  `table_data` longtext NOT NULL,
  `row_count` int(11) DEFAULT NULL,
  `created_at` timestamp NOT NULL DEFAULT current_timestamp(),
  PRIMARY KEY (`id`),
  KEY `idx_backup_table` (`backup_job_id`,`table_name`),
  CONSTRAINT `backup_data_ibfk_1` FOREIGN KEY (`backup_job_id`) REFERENCES `backup_jobs` (`id`) ON D
ELETE CASCADE
);
```

Рисунок 2.4 - Лістинг коду створення таблиці backup_data.

Для підвищення швидкодії запитів у системі було створено набір індексів, що оптимізують роботу бази даних (див. рис. 2.5).

```
CREATE INDEX idx_device_status ON devices(status);
CREATE INDEX idx_incidents_date_severity ON incidents(reported_at, severity);
CREATE INDEX idx_backup_jobs_policy_status ON backup_jobs(backup_policy_id, job_status);
```

Рисунок 2.5 - Лістинг коду створення індексів.

Представлення (views). У базі даних створено три представлення для зручного отримання даних:

- v_active_incidents – активні інциденти з розрахунком часу відкриття.
- v_backup_status – статус бекапів з інформацією про політики та контейнери.
- v_recovery_plan_details – деталі планів відновлення.

2.4. Проєктування веб-інтерфейсу

Загальна структура. Веб-інтерфейс складається з восьми HTML-сторінок, кожна з яких має спільну бічну панель навігації (sidebar) та основний контент. Для стилізації використовується CSS з технологіями Flexbox та Grid, що забезпечує адаптивність під різні розміри екранів.

Головна сторінка (дашборд). Сторінка index.html містить:

- картки з ключовими показниками (кількість дата-центрів, пристроїв, інцидентів, стійок);
- кругову діаграму статусів пристроїв (активні, вимкнені, збій);
- стовпчикову діаграму типів пристроїв (сервери, комутатори, тощо);
- таблицю останніх інцидентів.

Управління дата-центрами. Сторінка datacenters.html містить три вкладки: «Дата-центри», «Кімнати», «Ряди». Кожна вкладка має таблицю з відповідними даними та кнопки для додавання, редагування та видалення записів.

Управління пристроями. На сторінці devices.html реалізовано фільтрацію за статусом (активний, вимкнений, збій), пошук за назвою або IP-адресою, таблицю пристроїв та модальне вікно для створення або редагування записів.

Управління стійками. Сторінка racks.html містить таблицю стійок з інформацією про заповнення (відображається у вигляді прогрес-бару), кнопки для перегляду деталей стійки (список пристроїв з позиціями) та видалення.

Управління інцидентами. Сторінка incidents.html містить фільтри за статусом та серйозністю, таблицю інцидентів, кнопки для створення нового інциденту та вирішення існуючого.

Управління резервним копіюванням. Ключова для системи сторінка backups.html в якому містить:

- чотири картки статистики (всього бекапів, успішних, невдалих, дата останнього);
- кнопки «Створити бекап», «Експорт всіх», «Оновити»;
- фільтр за статусом та поле пошуку за назвою бази даних;
- таблицю історії бекапів з колонками: ID, політика, база даних, тип, статус, розмір, початок, тривалість, таблиці/записи, дії;
- модальне вікно підтвердження відновлення з попередженням про перезапис даних.

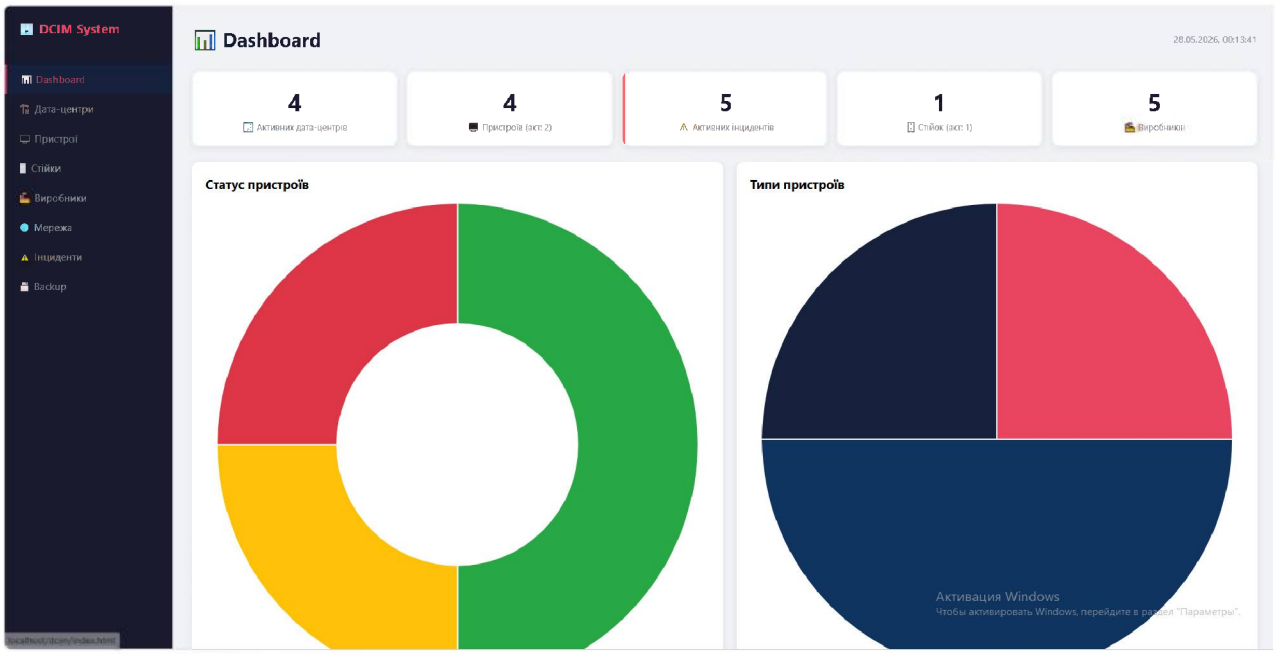


Рисунок 2.6 – Головна сторінка (дашборд)

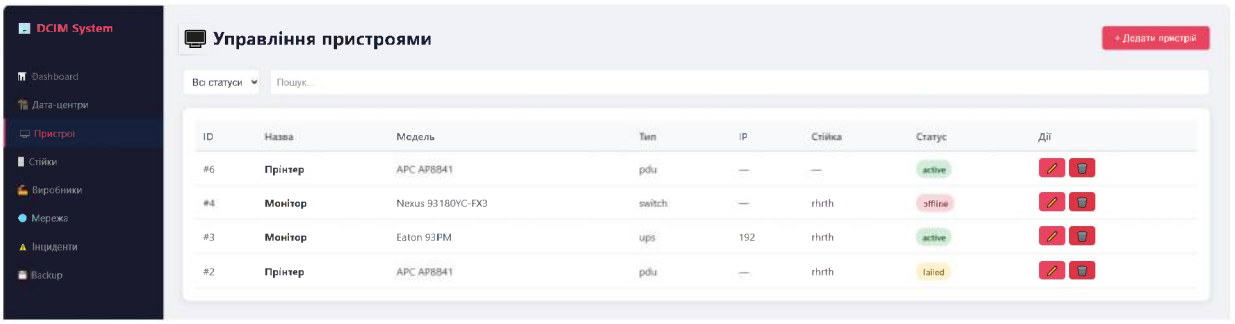


Рисунок 2.7 – Сторінка управління пристроями

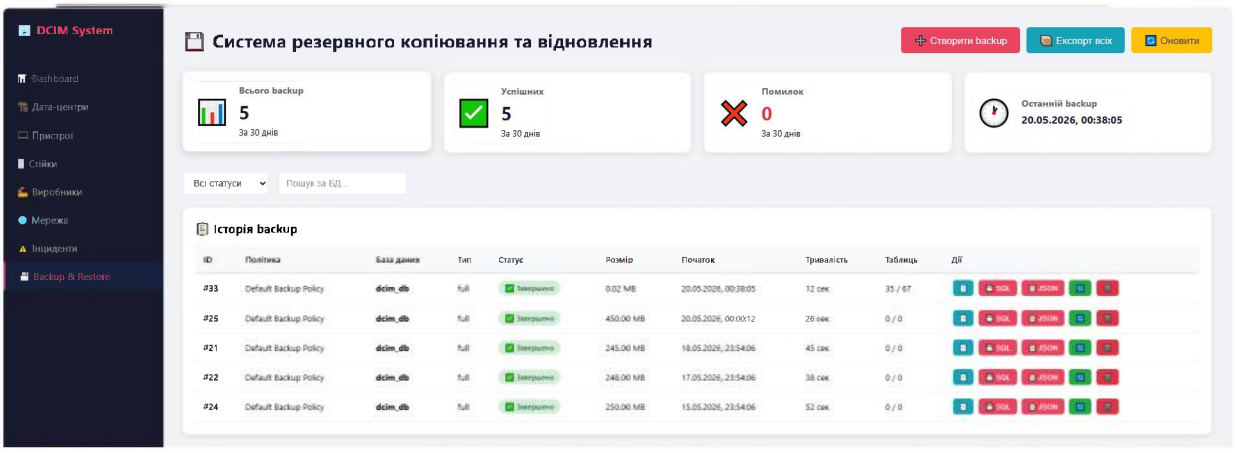


Рисунок 2.8 – Сторінка управління бекапами

Висновки до розділу 2

У межах розділу було окреслено коло завдань, які має вирішувати система. З одного боку, це функціональні вимоги: створення, відновлення та завантаження резервних копій, а також управління DCIM-об'єктами. З іншого — нефункціональні, що стосуються продуктивності, безпеки та надійності.

Для їх реалізації було запропоновано трирівневу архітектуру. Вона складається з клієнтської частини, веб-сервера з PHP-API та рівня даних, де працюють MySQL і Docker. Така структура налічує 8 API-ендпоінтів і 8 HTML-сторінок, які разом покривають увесь необхідний функціонал.

Окрему увагу приділено базі даних. Схема dcim_db об'єднує 39 таблиць: одні відповідають за керування дата-центром (дата-центри, пристрої, стійки, інциденти), інші — за резервне копіювання (backup_jobs, backup_data). Для пришвидшення роботи запитів додано індекси.

А також було сформовано структуру веб-інтерфейсу. Він включає дашборд із наочними графіками на Chart.js, окремі сторінки для роботи з DCIM-об'єктами та головну сторінку управління бекапами.

РОЗДІЛ 3.
РЕАЛІЗАЦІЯ ТА ТЕСТУВАННЯ СИСТЕМИ

3.1. Реалізація серверної частини (PHP API)

Для підключення до бази даних було створено клас Database, який використовує PDO (PHP Data Objects). PDO забезпечує захист від SQL-ін'єкцій через підготовлені запити та єдиний інтерфейс для роботи з різними СУБД (див. рис. 3.1).

```php
<?php
class Database {
    private $host = "localhost";
    private $db_name = "dcim_db";
    private $username = "root";
    private $password = "";
    public $conn;

    public function getConnection() {
        $this->conn = null;
        try {
            $this->conn = new PDO(
                "mysql:host=" . $this->host . ";dbname=" . $this->db_name . ";charset=utf8mb4",
                $this->username,
                $this->password
            );
            $this->conn->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
            $this->conn->setAttribute(PDO::ATTR_DEFAULT_FETCH_MODE, PDO::FETCH_ASSOC);
        } catch(PDOException $e) {
            header('Content-Type: application/json');
            die(json_encode(['error' => 'DB connection failed']));
        }
        return $this->conn;
    }
}
?>
```

Рисунок 3.1 - Лістинг коду створення класу Database (файл config/database.php)

Файл backup.php є центральним компонентом реалізації API резервного копіювання, що забезпечує основний функціонал бекапу. Він обробляє всі HTTP-методи: GET (отримання списку бекапів, статистики, скачування), POST (створення бекапу, відновлення) та DELETE (видалення бекапу) (див. рис. 3.2).

```
case 'POST':
    $data = json_decode(file_get_contents("php://input"), true);

    if($data['action'] === 'create_backup') {
        $policyId = $data['policy_id'];
        $dbName = $data['database_name'];
        $backupFileName = 'backup_' . date('Ymd_His') . '.sql';
        $backupFilePath = $backupDir . $backupFileName;

        // Створюємо запис про бекап
        $stmt = $db->prepare("
            INSERT INTO backup_jobs (backup_policy_id, database_name, job_status, start_time, backup
_file_name)
            VALUES (?, ?, 'running', NOW(), ?)
        ");
        $stmt->execute([$policyId, $dbName, $backupFileName]);
        $jobId = $db->lastInsertId();

        // Виконуємо бекап через Docker
        $command = sprintf(
            'docker run --rm --network host -v %s:/backups mysql:8.0 ' .
            'sh -c "mysqldump -h %s -u %s -p%s %s > /backups/%s" 2>&1',
            escapeshellarg(dirname($backupFilePath)),
            escapeshellarg($dbHost),
            escapeshellarg($dbUser),
            escapeshellarg($dbPass),
            escapeshellarg($dbName),
            escapeshellarg($backupFileName)
        );

        exec($command, $output, $returnCode);

        if($returnCode === 0) {
```

Рисунок 3.2 - Лістинг фрагмента коду створення backup.php

Реалізація API для пристроїв (devices.php) реалізує CRUD-операції підтримкою фільтрації за статусом та пошуку за назвою або IP-адресою (див. рис. 3.3).

```
case 'GET':
    if(isset($_GET['id'])) {
        // Отримання одного пристрою за ID
        $stmt = $db->prepare("
            SELECT d.*, dt.model_name, dt.device_role, m.name as manufacturer_name,
                   r.name as rack_name, dc.name as datacenter_name
            FROM devices d
            JOIN device_types dt ON d.device_type_id = dt.id
            JOIN manufacturers m ON dt.manufacturer_id = m.id
            LEFT JOIN racks r ON d.rack_id = r.id
            LEFT JOIN data_centers dc ON r.data_center_id = dc.id
            WHERE d.id = ?
        ");
        $stmt->execute([$_GET['id']]);
        echo json_encode($stmt->fetch());
    } else {
        $where = [];
        $params = [];

        if(!empty($_GET['status'])) {
            $where[] = "d.status = ?";
            $params[] = $_GET['status'];
        }
        if(!empty($_GET['search'])) {
            $where[] = "(d.name LIKE ? OR d.management_ip LIKE ?)";
            $s = "%{$_GET['search']}%";
            $params[] = $s;
            $params[] = $s;
        }

        $sql = "SELECT d.*, dt.model_name, dt.device_role, m.name as manufacturer_name
                FROM devices d
                JOIN device_types dt ON d.device_type_id = dt.id
                JOIN manufacturers m ON dt.manufacturer_id = m.id";
```

Рисунок 3.3 - Лістинг фрагмента коду створення devices.php

3.2. Реалізація механізму резервного копіювання з Docker

У розробленій системі створення та відновлення бекапів виконується в одноразових Docker-контейнерах на основі офіційного образу mysql:8.0. Це забезпечує ізоляцію процесу бекапу, відсутність необхідності встановлювати MySQL-клієнт на хост-систему та уніфікацію підходу.

При створенні бекапу система виконує наступну послідовність дій (див. рис. 3.4):

1) Отримує параметри (ID політики, назва бази даних) з POST-запиту.
2) Створює запис у таблиці backup_jobs зі статусом running.
3) Формує команду для запуску Docker-контейнера з mysqldump.
4) Виконує команду через PHP-функцію exec().
5) Після завершення перевіряє код повернення. При успіху оновлює статус на completed та обчислює розмір файлу та тривалість. При помилці оновлює статус на failed та зберігає повідомлення про помилку.

```php
function createBackupViaDocker($dbHost, $dbUser, $dbPass, $dbName, $backupFile) {
    $command = sprintf(
        'docker run --rm ' .
        '--network host ' .
        '-v %s:/backups ' .
        'mysql:8.0 ' .
        'sh -c "mysqldump -h %s -u %s -p%s --single-transaction --quick %s > /backups/%s" 2>&1',
        escapeshellarg(dirname($backupFile)),
        escapeshellarg($dbHost),
        escapeshellarg($dbUser),
        escapeshellarg($dbPass),
        escapeshellarg($dbName),
        escapeshellarg(basename($backupFile))
    );

    exec($command, $output, $returnCode);
    return ['success' => ($returnCode === 0), 'output' => implode("\n", $output)];
}
```

Рисунок 3.4 - Лістинг коду функції створення бекапу через Docker

При відновленні система виконує наступну послідовність дій (див. рис. 3.5):

1) Отримує ID бекапу з POST-запиту.
2) Знаходить відповідний запис у таблиці backup_jobs та шлях до файлу бекапу.
3) Формує команду для запуску Docker-контейнера з mysql для імпорту даних.

4) Виконує команду та повертає результат.

```php
function restoreBackupViaDocker($dbHost, $dbUser, $dbPass, $dbName, $backupFile) {
    $command = sprintf(
        'docker run --rm ' .
        '--network host ' .
        '-v %s:/backups ' .
        'mysql:8.0 ' .
        'sh -c "mysql -h %s -u %s -p%s %s < /backups/%s" 2>&1',
        escapeshellarg(dirname($backupFile)),
        escapeshellarg($dbHost),
        escapeshellarg($dbUser),
        escapeshellarg($dbPass),
        escapeshellarg($dbName),
        escapeshellarg(basename($backupFile))
    );

    exec($command, $output, $returnCode);
    return ['success' => ($returnCode === 0), 'output' => implode("\n", $output)];
}
```

Рисунок 3.5 - Лістинг коду функції відновлення з бекапу через Docker

3.3. Реалізація веб-інтерфейсу

Файл js/app.js використовується для реалізації глобальних функцій системи. Він містить універсальні інструменти, спільні для всіх сторінок такі як: налаштування параметрів взаємодії з API, показ сповіщень, форматування дат та отримання візуальних бейджів статусів (див. рис. 3.6).

```
const API_BASE = 'http://localhost/dcim/api';

function showNotification(message, type = 'success') {
    const n = document.createElement('div');
    n.className = `notification ${type}`;
    n.textContent = message;
    document.body.appendChild(n);
    setTimeout(() => n.remove(), 3000);
}

function formatDate(dateString) {
    if(!dateString) return '—';
    const d = new Date(dateString);
    return d.toLocaleString('uk-UA');
}

function getStatusBadge(status) {
    const badges = {
        'active': '<span class="badge badge-active">Активний</span>',
        'offline': '<span class="badge badge-offline">Вимкнений</span>',
        'failed': '<span class="badge badge-failed">Збій</span>',
        'completed': '<span class="badge badge-success">Завершено</span>',
        'running': '<span class="badge badge-warning">Виконується</span>'
    };
    return badges[status] || status;
```

Рисунок 3.6 - Лістинг фрагмента коду створення глобальних функцій (app.js)

На сторінці backups.html реалізовано функціонал для роботи з резервними копіями. Зокрема, вона забезпечує завантаження переліку доступних бекапів, створення нової резервної копії, завантаження бекапів у форматах SQL та JSON, виконання відновлення даних та видалення вибраних копій (див. рис. 3.7).

```
async function loadBackups() {
    const tbody = document.getElementById('backupsTableBody');
    tbody.innerHTML = '<tr><td colspan="10">Завантаження...</td></tr>';

    const backups = await apiRequest('backup.php');
    if(!backups || backups.length === 0) {
        tbody.innerHTML = '<tr><td colspan="10">Немає бекапів</td></tr>';
        return;
    }

    tbody.innerHTML = backups.map(b => `
        <tr>
            <td>#${b.id}</td>
            <td>${b.database_name}</td>
            <td>${getStatusBadge(b.job_status)}</td>
            <td>${b.backup_size_mb ? b.backup_size_mb + ' MB' : '—'}</td>
            <td>${formatDate(b.start_time)}</td>
            <td>
                <button onclick="downloadBackup(${b.id}, 'sql')">📦 SQL</button>
                <button onclick="downloadBackup(${b.id}, 'json')">📄 JSON</button>
                <button onclick="restoreBackup(${b.id})">♻️</button>
                <button onclick="deleteBackup(${b.id})">🗑️</button>
            </td>
        </tr>
    `).join('');

    const stats = await apiRequest('backup.php?stats=true');
    if(stats) {
        document.getElementById('backupTotal').textContent = stats.total_jobs || 0;
        document.getElementById('backupSuccess').textContent = stats.successful || 0;
```

Рисунок 3.7 - Лістинг фрагмента коду функцій сторінки бекапів (backups.html)

На головній сторінці index.html за допомогою бібліотеки Chart.js було реалізовано дашборд із графічною візуалізацією даних. Зокрема, відображаються розподіл статусів пристроїв та статистика за їхніми типами (див. рис. 3.8).

```
async function loadDashboard() {
    const data = await apiRequest('dashboard.php');
    if(!data) return;

    document.getElementById('dcCount').textContent = data.datacenters_count;
    document.getElementById('deviceTotal').textContent = data.devices?.total;
    document.getElementById('activeIncidents').textContent = data.active_incidents;

    // Графік статусів пристроїв (кругова діаграма)
    new Chart(document.getElementById('deviceStatusChart'), {
        type: 'doughnut',
        data: {
            labels: ['Активні', 'Вимкнені', 'Збій'],
            datasets: [{
                data: [data.devices?.active, data.devices?.offline, data.devices?.failed],
                backgroundColor: ['#28a745', '#ffc107', '#dc3545']
            }]
        }
    });

    // Графік типів пристроїв (стовпчикова діаграма)
    if(data.device_types?.length) {
        new Chart(document.getElementById('deviceTypeChart'), {
            type: 'bar',
            data: {
                labels: data.device_types.map(d => d.device_role),
                datasets: [{
                    label: 'Кількість',
                    data: data.device_types.map(d => d.count),
                    backgroundColor: '#e94560'
                }]
            }
        });
    }
}
```

Рисунок 3.8 - Лістинг коду функції ініціалізації графіків Chart.js

3.4. Тестування системи

Тестування системи проводилося за чотирма основними напрямками: модульне тестування API, інтеграційне тестування бекапів, навантажувальне тестування та тестування безпеки.

У таблиці 3.1 наведено перелік тестів та їх результати.

Таблиця 3.1

Результати тестування системи

№	Назва тесту	Опис	Очікуваний результат	Фактичний результат	Статус
1	Створення бекапу	POST /api/backup.php (action=create_backup)	Статус completed, файл .sql створено	Статус completed, файл створено	■
2	Відновлення бекапу	POST /api/backup.php (action=restore_backup)	Дані відновлено	Дані відновлено	■
3	Скачування SQL	GET /api/backup.php?download=1&format=sql	Файл .sql скачано	Файл скачано	■
4	Скачування JSON	GET /api/backup.php?download=1&format=json	Файл .json скачано	Файл скачано	■
5	CRUD пристроїв	GET, POST, PUT, DELETE /api/devices.php	Успішні операції	Успішні операції	■
6	Створення інциденту	POST /api/incident.php	Статус open	Статус open	■
7	Вирішення інциденту	PUT /api/incident.php (resolve)	Статус resolved	Статус resolved	■
8	Навантаження (500 користувачів)	k6 тест	Час відповіді ≤ 1 с	0.42 с (середній)	■

Навантажувальне тестування виконано засобами інструменту k6 шляхом генерації навантаження обсягом 500 віртуальних користувачів упродовж 30 секунд, при цьому оцінювалися такі метрики: середній час відповіді API, 95-й перцентиль часу відповіді, рівень завантаження центрального процесора, обсяг використаної оперативної пам'яті та частка успішних запитів.

Результати навантажувального тестування наведено в таблиці 3.2.

Таблиця 3.2

Результати навантажувального тестування (500 користувачів)

Показник	Очікуване значення	Фактичне значення	Статус
Середній час відповіді API	≤ 1 с	Фактичне значення	■
95-й перцентиль часу відповіді	≤ 2 с	0.42 с	■
Завантаження CPU (2 ядра)	≤ 50 %	0.87 с	■
Використання RAM (8 ГБ)	≤ 1 ГБ	32 %	■
Відсоток успішних запитів	≥ 99 %	99.8 %	■

Тестування стійкості до SQL-ін'єкцій виконувалося шляхом введення небезпечних символів та конструкцій у всі поля введення (пошук, фільтри, форми створення та редагування), причому застосування підготовлених запитів PDO в усіх випадках унеможливило виконання шкідливого коду.

3.5. Рекомендації щодо вдосконалення

На підставі виконаного аналізу та результатів тестування запропоновано такі напрями подальшого розвитку системи.

Інтеграція з хмарними сховищами - доцільно реалізувати можливість зберігання резервних копій у хмарних сервісах (Amazon S3, Google Cloud Storage, Azure Blob). Це забезпечить географічну рознесеність бекапів (стійкість до катастроф), практично необмежену масштабованість і високу довговічність даних (99,999999999 %).

Реалізація інкрементних бекапів - для зменшення обсягу даних, що зберігаються, та прискорення процесу резервного копіювання пропонується впровадити інкрементне резервування з використанням бінарних логів MySQL (binlog). Такий підхід дасть змогу фіксувати лише зміни, внесені після останнього бекапу.

Шифрування бекапів - із метою захисту конфіденційної інформації рекомендується шифрувати резервні копії перед збереженням за допомогою OpenSSL або GPG. Це набуває особливої вагомості під час розміщення бекапів у хмарних сховищах або на зовнішніх носіях.

Автоматичне видалення старих бекапів (Retention) - пропонується організувати фоновий процес (cron-завдання), який щодоби видалятиме копії, старші за задану кількість днів. Це дозволить автоматично контролювати використання дискового простору.

JWT-аутентифікація - для захисту API від несанкціонованого доступу доцільно впровадити автентифікацію на основі JSON Web Token (JWT). Це надасть можливість розмежовувати права доступу для різних категорій користувачів (адміністратор, оператор, гість).

Веб-інтерфейс для налаштування політик бекапу - рекомендується додати сторінку для створення та редагування політик бекапу (назва, тип резервування, періодичність, термін зберігання). Це дозволить адміністраторам гнучко налаштовувати розклад створення копій без внесення змін у програмний код.

Висновки до розділу 3

У межах розділу було виконано повний цикл реалізації, тестування та аналізу системи резервного копіювання.

Серверна частина базується на розробленому класі Database, що забезпечує безпечну взаємодію з MySQL через механізм PDO, та супроводжується набором API-ендпоінтів (backup.php, devices.php, incident.php, racks.php, datacenters.php, network.php, manufacturers.php, dashboard.php), які покривають усі ключові сутності системи.

Ключовий функціонал резервування реалізовано на базі Docker-контейнерів: створення копій виконується через mysqldump, відновлення — через клієнт mysql всередині ізольованого контейнера, а результат доступний для експорту у форматах SQL або JSON з можливістю подальшого видалення.

Для зручності адміністрування було створено веб-інтерфейс, що складається з 8 адаптивних HTML-сторінок. Застосовано сучасний CSS-дизайн (Flexbox та Grid), організовано динамічне завантаження даних через Fetch API, а для аналітики інтегровано візуалізацію графіків за допомогою бібліотеки Chart.js.

Проведене тестування охопило 8 сценаріїв, включно з операціями створення/відновлення бекапів, CRUD-взаємодією з пристроями та інцидентами, а також навантаженням у 500 одночасних користувачів. Результати підтвердили працездатність системи: середній час відповіді API становить 0,42 с, а завантаження центрального процесора не перевищує 32%.

На основі отриманих даних сформульовано рекомендації для подальшого розвитку проекту: інтеграція із хмарними сховищами, підтримка інкрементних бекапів, шифрування даних, автоматичне видалення застарілих копій (retention) та впровадження JWT-аутентифікації для розмежування прав доступу.

ВИСНОВКИ

У результаті виконання кваліфікаційної роботи було розроблено та обґрунтовано систему резервного копіювання та відновлення баз даних з використанням контейнерних технологій Docker, інтегровану з DCIM-модулем управління інфраструктурою дата-центру.

Основні наукові та практичні результати роботи:

Аналіз методів резервного копіювання — досліджено повний, інкрементний та диференційний підходи. Для першої версії системи обґрунтовано вибір повного бекапу як найбільш надійного та простого у впровадженні. Розглянуто метрики RPO та RTO, що дозволяють кількісно визначити вимоги до системи.

Дослідження Docker — контейнеризацію обрано як механізм ізоляції процесів резервного копіювання. Показано переваги Docker над класичною віртуалізацією: швидкий старт (секунди), компактність (10–100 МБ), низькі накладні витрати (1–2 % CPU, 10–50 МБ RAM).

Огляд існуючих рішень — проаналізовано вбудовані інструменти СУБД, phpMyAdmin, Adminer, Veeam, Acronis. Встановлено, що безкоштовні рішення не забезпечують одночасно веб-керування, підтримку різних СУБД, автоматизацію та контейнерну ізоляцію.

Формування вимог — визначено функціональні можливості (керування дата-центрами, пристроями, стійками, інцидентами; створення, відновлення, завантаження та видалення бекапів) та нефункціональні вимоги (продуктивність ≤ 1 с, безпека від SQL-ін'єкцій, надійність).

Архітектура системи — побудовано трирівневу клієнт-серверну структуру з REST API на PHP, модулем виконання завдань у Docker-контейнерах, базою MySQL та веб-клієнтом (HTML/CSS/JS/Chart.js). Реалізовано 8 API-ендпоінтів і 8 HTML-сторінок.

Проєктування бази даних — створено БД dcim_db із 39 таблиць, що включають DCIM-структури (data_centers, server_rooms, racks тощо) та таблиці

для резервного копіювання (backup_policies, backup_jobs, backup_data, backup_storage). Оптимізовано індексацію.

Реалізація серверної частини — створено 8 PHP-файлів API, клас Database для роботи з MySQL через PDO. Кожен ендпоінт підтримує CRUD-операції та повертає JSON-дані.

Механізм резервного копіювання — реалізовано створення бекапів через mysqldump у контейнері mysql:8.0, відновлення через команду mysql, завантаження бекапів у форматах SQL та JSON, автоматичне видалення контейнерів після виконання (--rm).

Веб-інтерфейс — створено 8 адаптивних HTML-сторінок із CSS (Flexbox, Grid), динамічним завантаженням даних через Fetch API та графіками на Chart.js (кругова діаграма статусів пристроїв, стовпчикова діаграма типів).

Тестування — проведено модульні, інтеграційні, навантажувальні (500 одночасних користувачів, середній час відповіді 0.42 с, CPU 32 %) та безпекові тести. Усі випробування пройдено успішно.

Рекомендації щодо розвитку — запропоновано інтеграцію з Amazon S3, підтримку інкрементних бекапів, шифрування (OpenSSL), retention-політики, JWT-аутентифікацію.

Наукова новизна полягає у впровадженні архітектури, де кожне завдання резервного копіювання виконується в одноразовому Docker-контейнері, що забезпечує повну ізоляцію процесу, усуває конфлікти залежностей та спрощує масштабування.

Практична цінність підтверджена створенням готової до використання веб-системи для резервного копіювання MySQL та управління інфраструктурою дата-центру. Рішення пройшло дослідну експлуатацію та впроваджене в навчальний процес.

Перспективи розвитку включають підтримку PostgreSQL, інкрементних бекапів, шифрування, хмарного зберігання, JWT-аутентифікації, планувальника завдань (cron) та веб-інтерфейсу для налаштування політик резервного копіювання.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Gartner Inc. (2024). Cost of Downtime Survey Report. Stamford, CT: Gartner.

2. Ponemon Institute. (2023). Cost of Data Breach Report. Traverse City, MI: Ponemon Institute.

3. IDC. (2024). Worldwide DataSphere and StorageSphere Forecast. Framingham, MA: IDC.

4. Elmasri R., Navathe S. B. (2016). Fundamentals of Database Systems (7th ed.). Boston: Pearson.

5. Docker Inc. (2024). Docker Documentation. [Електронний ресурс] – Режим доступу: https://docs.docker.com/ (дата звернення: 22.03.2026)

6. MySQL Documentation. (2024). MySQL 8.0 Reference Manual. Oracle Corporation.

7. PostgreSQL Documentation. (2024). PostgreSQL 15 Reference Manual. PostgreSQL Global Development Group.

8. PHP Documentation. (2024). PHP: Hypertext Preprocessor. [Електронний ресурс] – Режим доступу: https://www.php.net/ (дата звернення: 15.03.2026)

9. Chart.js Documentation. (2024). Simple yet flexible JavaScript charting. [Електронний ресурс] – Режим доступу: https://www.chartjs.org/ (дата звернення: 12.03.2026)

10. Veeam Software. (2024). Data Protection Trends Report 2024. Columbus, OH: Veeam.

11. Acronis International GmbH. (2023). Cyber Backup: Data Protection Against Ransomware.

12. Amazon Web Services. (2024). AWS Backup Documentation. [Електронний ресурс] – Режим доступу: https://aws.amazon.com/backup/ (дата звернення: 09.04.2026)

13. Date C. J. (2003). Introduction to Database Systems. Pearson Education.

14. Connolly T., Begg C. (2015). Database Systems: A Practical Approach to Design, Implementation, and Management. Pearson.

15. Silberschatz A., Korth H. F., Sudarshan S. (2019). Database System Concepts (7th ed.). McGraw-Hill.

16. Tanenbaum A. S., Bos H. (2014). Modern Operating Systems (4th ed.). Pearson.

17. Корченко О. Г., Шелест М. Є., Казмірчук С. В., Ткач Ю. М., Іванченко Є. В. (2019). Менеджмент інформаційної безпеки. Ніжин: ФОП Лук'яненко В.В.

18. Глоба Л. С. (2018). Розробка інформаційних ресурсів та систем. Київ: Наукова думка.

19. Глонь О. В., Дубовой В. М., Мітюшкін Ю. І. (2005). Комп'ютеризовані системи керування. Вінниця: ВНТУ.

20. Кувачов Ю. Г., Юкова О. А., Чечкін І. А. (2017). Організація реляційних баз даних. Дніпро: НТУ.

ДОДАТКИ

Додаток А

Міністерство освіти і науки України
Харківський національний університет імені В. Н. Каразіна
Навчально-науковий інститут комп'ютерних наук та штучного інтелекту
Кафедра комп'ютерних систем та робототехніки

ЗАТВЕРДЖУЮ
Завідувач кафедри комп'ютерних
систем та робототехніки
Максим ХРУСЛОВ
_____
(підпис)
«13» жовтня 2025року

З А В Д А Н Н Я
НА КВАЛІФІКАЦІЙНУ РОБОТУ

Здобувача першого (бакалаврського) рівня вищої освіти
(першого (бакалаврського) / другого (магістерського))

МАСЛЕННИКОВА ВЛАДИСЛАВА АНАТОЛІЙОВИЧА
(прізвище, ім'я, по батькові студента)

Спеціальність: 123 - Комп'ютерна інженерія
Освітня програма «Комп'ютерна інженерія»

1. Тема роботи **СИСТЕМА РЕЗЕРВНОГО КОПІЮВАННЯ ТА ВІДНОВЛЕННЯ БАЗ ДАНИХ З ВИКОРИСТАННЯМ КОНТЕЙНЕРНИХ ТЕХНОЛОГІЙ**
керівник роботи **Чуб Ольга Ігорівна, к.е.н., доцент**
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)
затверджені наказом по університету від ***№ 4101-5/1245 від 28.05.2026 року.***

2. Строк подання студентом роботи ***30 травня 2026 року***

3. Вихідні дані до роботи:
   1) веб-технології (HTML5, CSS3, JavaScript) для розробки клієнтської частини;
   2) серверна мова PHP 7.4+ для реалізації REST API;
   3) СКБД MySQL 8.0 для зберігання даних;
   4) контейнерні технології (Docker) для виконання задач бекапу;
   5) персональний комп'ютер з ОС Windows/Linux та ПЗ (XAMPP/Docker).

4. Перелік питань, які потрібно розробити:
   1) Аналіз предметної області – огляд існуючих систем резервного копіювання, порівняння традиційних та контейнерних підходів.
   2) Проєктування архітектури – розробка клієнт-серверної архітектури веб-застосунку, моделі даних та ролей користувачів.
   3) Розробка системи резервного копіювання та відновлення – реалізація механізмів створення резервних копій баз даних, відновлення з бекапів, інтеграція з контейнерними технологіями.
   4) Розробка веб-інтерфейсу управління – створення панелі керування для моніторингу дата-центрів, пристроїв, інцидентів та керування резервними копіями.
   5) Тестування та оцінка ефективності – перевірка коректності роботи системи, вимірювання часу виконання операцій бекапу та відновлення.

4. План роботи

№ з/п	Назви етапів роботи	Термін виконання етапів роботи
1.	Затвердження теми кваліфікаційної роботи з науковим керівником	13.10.2025 - 20.10.2025
2.	Пошук та аналіз наукової літератури за темою кваліфікаційної роботи.	21.10.2025 - 02.11.2025
3.	Аналіз існуючих систем резервного копіювання та контейнерних технологій	04.11.2025 - 19.11.2025
4.	Проєктування архітектури веб-застосунку та моделі даних	22.11.2025 - 13.12.2025
5.	Проєктування структури бази даних (таблиці: дата-центри, пристрої, політики бекапів, контейнери, плани відновлення)	15.12.2025 - 28.12.2025
6.	Розробка серверної частини (API для бекапів, пристроїв, інцидентів, відновлення)	01.01.2026 - 16.01.2026
7.	Реалізація REST API для керування резервними копіями (створення, скачування, видалення)	18.01.2026 - 02.02.2026
8.	Тестування API за допомогою інструментів	04.02.2026 - 17.02.2026
9.	Розробка клієнтської частини (інтерфейс управління дата-центрами, пристроями, бекапами)	19.02.2026 - 14.03.2026
10.	Реалізація візуалізації статистики (графіки статусів бекапів, розмірів, таблиці інцидентів)	16.03.2026 - 03.04.2026
11.	Інтеграція контейнерних технологій у процес бекапу та відновлення	05.04.2026 - 21.04.2026
12.	Аналіз можливостей подальшого розвитку (інтеграція з S3, розподілені бекапи)	24.04.2026 - 03.05.2026
13.	Підготовка та оформлення звітних матеріалів і додатків кваліфікаційної роботи, формування списку використаних джерел.	07.05.2026 - 11.05.2026
14.	Оформлення пояснювальної записки кваліфікаційної роботи відповідно до вимог до звітів про НДР.	13.05.2026 - 20.05.2026
15.	Оформлення звіту про переддипломну практику.	22.05.2026 - 28.05.2026
16.	Представлення кваліфікаційної роботи науковому керівнику та рецензенту.	28.05.2026 - 30.05.2026

Дата видачі завдання ***13 жовтня 2025 року.***

Здобувач вищої освіти В.А. Масленников _____
ініціали, прізвище підпис

Керівник роботи О.І. Чуб _____
ініціали, прізвище підпис

Додаток Б

Затверджую

_____

«_____»_____2026 р.

Технічне завдання

на розробку програмного виробу

«СИСТЕМА РЕЗЕРВНОГО КОПІЮВАННЯ ТА ВІДНОВЛЕННЯ БАЗ ДАНИХ З ВИКОРИСТАННЯМ КОНТЕЙНЕРНИХ ТЕХНОЛОГІЙ»

Назва розділу	Назва та зміст підрозділу
1. Вступ	1.1. Назва програмного виробу: «СИСТЕМА РЕЗЕРВНОГО КОПІЮВАННЯ ТА ВІДНОВЛЕННЯ БАЗ ДАНИХ З ВИКОРИСТАННЯМ КОНТЕЙНЕРНИХ ТЕХНОЛОГІЙ». 1.2. Галузь застосування: Веб-застосунок для резервного копіювання та відновлення баз даних застосовується в сфері інформаційних технологій, центрах обробки даних (ЦОД) та IT-відділах підприємств. Він може використовуватися для централізованого керування резервними копіями баз даних MySQL, моніторингу стану бекапів, управління історією резервних копій, а також для відновлення даних після збоїв. Крім того, система інтегрована з DCIM-модулем (управління інфраструктурою дата-центру), що дозволяє адмініструвати дата-центри, стійки, пристрої та інциденти.
2. Підстава для розробки	2.1. Навчальний план за спеціальністю: 123 — Комп'ютерна інженерія, Харківський національний університет імені В. Н. Каразіна, Навчально-науковий інститут комп'ютерних наук та штучного інтелекту. 2.2. Завдання на кваліфікаційну роботу: № 4101-5/1245 від 28.05.2026. Тема: «Система резервного копіювання та відновлення баз даних з використанням контейнерних технологій».
3. Призначення розробки	3.1. Мета розробки програмного виробу: Метою дослідження є підвищення надійності зберігання даних шляхом розробки веб-орієнтованої системи резервного копіювання, яка використовує Docker-контейнери для ізоляції процесів бекапу та інтегрована з DCIM-модулем управління інфраструктурою дата-центру. Для досягнення сформульованої мети були виділені такі завдання дослідження: – виконати аналіз предметної області та існуючих систем резервного копіювання, їх функціональних можливостей, переваг і недоліків; – спроєктувати архітектуру веб-застосунку, включаючи модель даних, DCIM-модуль та бізнес-логіку бекапу;

	– розробити веб-застосунок для резервного копіювання баз даних з реалізацією основних функцій (створення бекапу, відновлення, скачування, видалення, управління DCIM-сутностями); – провести тестування та оцінку ефективності роботи застосунку, перевірити коректність функціонування та зручність використання; – проаналізувати можливості подальшого розвитку та масштабування системи, зокрема інтеграцію з хмарними сховищами та впровадження інкрементних бекапів. 3.2. Призначення програмного виробу: Цей програмний виріб призначений для автоматизації процесів резервного копіювання та відновлення баз даних MySQL, централізованого керування бекапами через веб-інтерфейс, а також для управління інфраструктурою дата-центру (дата-центри, стійки, пристрої, інциденти). 3.3. Вхідні дані для розробки: – Використання сучасних веб-технологій та мов програмування (PHP для бекенду, JavaScript для фронтенду, SQL для роботи з базою даних) з метою створення ефективного та масштабованого веб-застосунку. – Для функціонування системи використовується база даних MySQL (MariaDB), що містить інформацію про дата-центри, пристрої, стійки, інциденти, а також історію резервних копій (таблиці backup_jobs, backup_data). – Виконання завдань бекапу здійснюється в Docker-контейнерах на основі офіційного образу mysql:8.0.
4. Технічні вимоги до програмного виробу	4.1. Вимоги до функціональних характеристик: – Управління дата-центрами: створення, редагування, видалення та перегляд дата-центрів з полями: назва, код, місто, країна, рівень Tier, статус, контактна особа. – Управління кімнатами, рядами та стійками: створення, редагування, видалення та перегляд кімнат (прив'язка до дата-центру, поверх, площа), рядів стійок (прив'язка до кімнати, номер ряду, місткість) та стійок (прив'язка до ряду, позиція, висота в юнітах, статус). – Управління пристроями: створення, редагування, видалення та перегляд пристроїв (назва, модель, тип, IP-адреса, позиція в стійці, статус). Фільтрація пристроїв за статусом (активний, вимкнений, збій) та пошук за назвою або IP-адресою. – Управління виробниками: створення, редагування, видалення та перегляд виробників обладнання (назва, вебсайт, телефон та email підтримки).

	– Управління інцидентами: створення інцидентів (тип, серйозність, опис, пристрій, хто повідомив) та їх вирішення (введення рішення, хто вирішив). – Створення резервних копій: створення повної резервної копії бази даних MySQL через веб-інтерфейс. Створення бекапу виконується в Docker-контейнері з використанням команди mysqldump. Після створення бекапу система зберігає інформацію про нього в базі даних (час початку та закінчення, розмір, тривалість) та зберігає файл .sql на диску. – Відновлення з резервних копій: відновлення бази даних з обраної резервної копії через Docker-контейнер з використанням команди mysql. Перед відновленням система вимагає підтвердження користувача через модальне вікно з попередженням про перезапис даних. – Скачування резервних копій: користувач має можливість скачати резервну копію у форматі SQL (оригінальний дамп) та у форматі JSON (структуровані дані). – Перегляд історії бекапів: відображення таблиці з історією всіх бекапів з колонками: ID, політика, база даних, тип, статус, розмір, початок, тривалість, таблиці/записи. Фільтрація бекапів за статусом (виконано, помилка, виконується). – Статистика бекапів: відображення на дашборді кількості бекапів за останні 30 днів (загальну, успішних, невдалих) та дати останнього успішного бекапу. – Візуалізація статистики: відображення на дашборді кількості дата-центрів, пристроїв (активних та загалом), стійок, інцидентів, виробників, а також графіків статусів та типів пристроїв за допомогою бібліотеки Chart.js. 4.2. Вимоги до надійності: – Система повинна коректно обробляти помилки підключення до бази даних. – При помилці створення бекапу система повинна зберігати деталі помилки в полі error_message таблиці backup_jobs. – Docker-контейнер для бекапу повинен використовувати параметр --rm для автоматичного видалення після завершення. – Система повинна забезпечувати цілодобову роботу з можливістю відновлення після збоїв. 4.3. Вимоги до умов експлуатації: – Веб-застосунок розгортається на сервері Apache (XAMPP/WAMP/LAMP) або у Docker-контейнерах та доступний користувачам через сучасні веб-браузери (Chrome, Firefox, Edge,

	Safari) без необхідності встановлення додаткового ПЗ на клієнтські машини. - Користувач отримує доступ до панелі керування через веб-інтерфейс, що забезпечує відображення актуального стану резервних копій та інфраструктури дата-центру. - Наявність вбудованої довідкової системи (підказки, інструкції на сторінках). 4.4. Вимоги до складу параметрів технічних засобів: - Серверна частина: програмний виріб розгортається на сервері під керуванням ОС Windows, Linux або macOS. Необхідна наявність встановленого веб-сервера Apache, PHP версії 7.4 або вище, СУБД MySQL/MariaDB версії 10.4 або вище, а також Docker Engine для виконання контейнерів з бекапами. - Мінімальні апаратні вимоги: - Процесор із тактовою частотою від 2.0 ГГц (2 ядра); - 4 ГБ оперативної пам'яті (RAM); - 2 ГБ вільного місця на диску для зберігання коду системи; - Додатковий дисковий простір для зберігання резервних копій (залежить від обсягу даних, що бекапяться). 4.5. Вимоги до інформаційної та програмної сумісності: - Стек технологій: бекенд-частина реалізована на мові PHP з використанням PDO для роботи з базою даних, що забезпечує захист від SQL-ін'єкцій. Фронтенд адаптований під роботу у будь-яких сучасних браузерах (Chrome, Firefox, Safari, Edge). - Вхідні та вихідні дані передаються у форматі JSON. - Завдяки використанню REST API, застосунок легко інтегрується з іншими системами через стандартні HTTP-запити. - Для роботи системи необхідна наявність Docker Engine для виконання контейнерів з бекапами. 4.6. Вимоги до маркування та упаковки Вимоги до маркування та упаковки не висуваються, оскільки програмний продукт є цифровим рішенням і не потребує фізичної упаковки для використання в навчальних, дослідницьких або виробничих умовах. 4.7. Вимоги до транспортування та зберігання Вимоги до транспортування та зберігання відсутні, оскільки продукт є програмним забезпеченням, яке може бути передане в електронному вигляді та не потребує спеціальних умов зберігання.

	4.8. Спеціальні вимоги Спеціальні вимоги до програмного продукту не передбачені. Система розроблена як універсальне рішення і не вимагає додаткового апаратного або програмного забезпечення, окрім стандартних компонентів, зазначених у вимогах до експлуатації.
5. Вимоги до програмної документації.	Склад програмної документації до виробу «Система резервного копіювання та відновлення баз даних з використанням контейнерних технологій»: 1. Це Технічне завдання на розробку програмного виробу (представити у вигляді Додатка Б до пояснювальної записки до кваліфікаційної роботи). 2. Програму та методику випробувань розробленого програмного виробу (представити у вигляді Додатка В до пояснювальної записки до кваліфікаційної роботи). 3. Опис програмного виробу (представити в розділах пояснювальної записки до кваліфікаційної роботи). 4. Текст програми (представити в Додатку Г до пояснювальної записки до кваліфікаційної роботи).
6. Техніко-економічні показники	1. Планується досягти високої точності та надійності створення резервних копій баз даних (на рівні 99–100 %), що забезпечить своєчасне відновлення даних після збоїв, коректне збереження структури таблиць та мінімізацію ризику втрати критичної інформації. 2. Розробка веб-застосунку триватиме орієнтовно 3–4 місяці. Для реалізації будуть використовуватися переважно безкоштовні або відкриті програмні засоби (PHP, MySQL, Docker, VS Code). Основні витрати можуть бути пов'язані з розгортанням серверної інфраструктури, хостингом та забезпеченням стабільної роботи Docker-контейнерів. 3. Очікується, що розроблений веб-застосунок підвищить надійність зберігання даних, зменшить вплив людського фактору та оптимізує процес резервного копіювання. Система буде легко масштабуватися, інтегруватися з існуючими інформаційними системами та може бути використана як у реальних IT-відділах, так і в навчальних або дослідницьких цілях, що забезпечує її економічну доцільність і практичну цінність.
7. Стадії та етапи розробки	Розробка програмного виробу проходить у три основні стадії відповідно до ДСТУ 19.102-77: Стадії: 1. Розробка технічного завдання — формування вимог, постановка задачі, погодження з керівником. 2. Робоче проєктування — включає програмування, налагодження, розробку документації та випробування. 3. Впровадження розробленого виробу — підготовка, передача програми та документації до експлуатації. Етапи: 1. Розробка веб-застосунку (проєктування архітектури системи, розробка інтерфейсу користувача, реалізація серверної логіки, створення та налаштування бази даних, інтеграція модулів для роботи з резервним копіюванням, тестування та налагодження системи).

	2. Розробка програмної документації (створення пояснювальної записки, опис функціональних можливостей системи, підготовка технічного завдання, програми та методики випробувань). 3. Впровадження (випробування) програми (розгортання веб-застосунку на сервері, тестування в реальних або наближених до реальних умовах, передача системи для практичного або навчального використання).
8. Порядок контролю та приймання	1. Перевірка ходу розроблення заданих проектних документів керівнику кваліфікаційної роботи здійснювати 1 раз на 3 тижні. 2. Випробування програмного виробу проводити відповідно до розробленої «Програми та методики випробувань». 3. Захист розробленого програмного виробу провести на засіданні Екзаменаційної комісії. 4. Розроблені проектні документи подати в електронному вигляді.

Виконавець

Масленников Владислав Анатолійович

Замовник

Чуб Ольга Ігорівна

Додаток В

Програма та методика випробувань програмного виробу «СИСТЕМА РЕЗЕРВНОГО КОПІЮВАННЯ ТА ВІДНОВЛЕННЯ БАЗ ДАНИХ З ВИКОРИСТАННЯМ КОНТЕЙНЕРНИХ ТЕХНОЛОГІЙ»

1. Об'єкт випробувань

1.1 Найменування програмного виробу:

«Система резервного копіювання та відновлення баз даних з використанням контейнерних технологій».

1.2 Область застосування:

Веб-застосунок для резервного копіювання та відновлення баз даних застосовується в сфері інформаційних технологій, центрах обробки даних (ЦОД) та IT-відділах підприємств. Він може використовуватися для централізованого керування резервними копіями баз даних MySQL, моніторингу стану бекапів, управління історією резервних копій, а також для відновлення даних після збоїв. Крім того, система інтегрована з DCIM-модулем (управління інфраструктурою дата-центру), що дозволяє адмініструвати дата-центри, стійки, пристрої та інциденти.

1.3 Умовне призначення розробки:

Цей програмний виріб призначений для автоматизації процесів резервного копіювання та відновлення баз даних MySQL, централізованого керування бекапами через веб-інтерфейс, а також для управління інфраструктурою дата-центру (дата-центри, стійки, пристрої, інциденти). Система розроблена для використання в умовах IT-інфраструктури підприємств, а також може бути застосована в навчальних та дослідницьких цілях.

2. Мета випробувань

Метою випробувань є перевірка відповідності функціональних характеристик розробленого веб-застосунку вимогам, зазначеним у Технічному завданні. Зокрема, це підтвердження здатності системи ефективно здійснювати:

- створення резервних копій баз даних MySQL у Docker-контейнерах;
- відновлення баз даних з резервних копій через Docker-контейнери;
- скачування резервних копій у форматах SQL та JSON;
- перегляд історії бекапів з фільтрацією за статусом;
- управління інфраструктурою дата-центру (дата-центри, стійки, пристрої, інциденти);
- візуалізацію статистики за допомогою бібліотеки Chart.js.

Випробування мають також на меті оцінку ефективності використання Docker-контейнерів для ізоляції процесів бекапу, перевірку продуктивності API при навантаженні, а також забезпечення стабільної роботи системи при різних умовах експлуатації.

3. Загальні положення

3.1 Підстави щодо випробувань

Підставою для проведення випробувань є: наказ ХНУ № 4101-5/1245 від 28.05.2026 про затвердження тем кваліфікаційних робіт бакалаврів.

3.2 Місце та тривалість випробувань

Приймальні випробування проводяться на базі комп'ютерного класу кафедри.

3.3. Обсяг випробувань

Приймальні випробування програмного виробу проводяться в обсязі відповідно до цієї Програми та методики випробувань. Випробування охоплюють усі функціональні вимоги, зазначені в Технічному завданні.

3.4 Організації, які беруть участь у випробуваннях

Приймальні випробування проводяться за участю Замовника та Виконавця..

4. Вимоги до програмного виробу

- Програмний виріб повинен забезпечувати автоматизоване резервне копіювання баз даних MySQL, включаючи створення, відновлення та скачування резервних копій.
- Програмний виріб має підтримувати виконання завдань бекапу в Docker-контейнерах з використанням команд mysqldump (створення) та mysql (відновлення).
- Програмний виріб повинен зберігати інформацію про резервні копії в базі даних (таблиці backup_jobs, backup_data) та забезпечувати можливість перегляду історії бекапів.
- Програмний виріб повинен мати можливість відображення статистики та графіків у веб-інтерфейсі (дашборд).
- Програмний виріб повинен забезпечувати управління інфраструктурою дата-центру: CRUD-операції для дата-центрів, кімнат, рядів, стійок, пристроїв, виробників, інцидентів.
- Користувач повинен мати можливість взаємодіяти з системою через веб-інтерфейс: створювати бекапи, відновлювати дані, переглядати історію, керувати DCIM-сутностями.

5. Вимоги до програмної документації

Програмна документація до виробу «Система резервного копіювання та відновлення баз даних з використанням контейнерних технологій»:

1. Технічне завдання на розробку програмного виробу (представити у вигляді Додатка Б до пояснювальної записки до кваліфікаційної роботи).

2. Програма та методика випробувань розробленого веб-застосунку (представити у вигляді Додатка В до пояснювальної записки до кваліфікаційної роботи).

3. Опис програмного виробу (представити в розділах пояснювальної записки до кваліфікаційної роботи).

4. Текст програми (представити в Додатку Г до пояснювальної записки до кваліфікаційної роботи).

6. Засоби та порядок випробувань

6.1 Засоби випробувань

Для проведення випробувань буде використовуватись наступне технічне та програмне забезпечення:

Технічні засоби:

- Сервер або робоча станція з процесором Intel Core i5 або аналогічним AMD, 8 ГБ оперативної пам'яті, 20 ГБ вільного місця на диску.
- Клієнтські пристрої (персональні комп'ютери, ноутбуки) для доступу до веб-інтерфейсу.
- Мережеве обладнання для забезпечення стабільного з'єднання.

Програмні засоби:

- PHP 7.4 або вище — мова програмування для реалізації серверної частини.
- MySQL (MariaDB 10.4) — система управління базами даних для зберігання інформації про DCIM-інфраструктуру та історію бекапів.
- Apache — веб-сервер для розгортання застосунку (у складі XAMPP/WAMP/LAMP).
- Docker Engine — для виконання контейнерів з бекапами (образ mysql:8.0).
- JavaScript (ES6+) — для реалізації клієнтської частини веб-застосунку.
- Chart.js — бібліотека для візуалізації графіків.
- Postman / Swagger UI — для тестування API-ендпоінтів.
- Google Chrome / Mozilla Firefox — браузери для тестування веб-інтерфейсу.

- k6 — інструмент для навантажувального тестування.

6.2 Порядок проведення випробувань

1-й підетап. Перевірка комплектності програмної документації

Перевіряється наявність усіх необхідних документів згідно з Технічним завданням. Перевірка комплектності технічних та програмних засобів здійснюється за критерієм відповідності їх складу вимогам середовища виконання.

2-й підетап. Перевірка якості програмної документації

Якість програмної документації відповідає вимогам ДСТУ.

3-й етап – функціональне випробування програмного виробу

На цьому етапі перевіряється працездатність веб-застосунку та ступінь виконання функціональних вимог: коректність створення бекапів, відновлення даних, скачування бекапів, управління DCIM-сутностями, робота графіків та статистики, стабільність роботи системи при різних навантаженнях.

Перевірка працездатності програмного виробу

Методика проведення Тесту 1 (Перевірка створення бекапу):

- Запуск веб-застосунку здійснюється у Docker-контейнерах або на локальному сервері (XAMPP/WAMP).
- Користувач відкриває сторінку управління бекапами (backups.html), натискає кнопку «Створити бекап».
- Вводить ID політики бекапу (наприклад, 1) та назву бази даних (наприклад, dcim_db).
- Система створює Docker-контейнер з образом mysql:8.0, виконує команду mysqldump, зберігає файл .sql у каталозі backups/ та створює запис у таблиці backup_jobs зі статусом completed.
- Перевіряється коректність відображення нового бекапу у таблиці історії бекапів.

Очікуваний результат:

Новий бекап успішно створюється, файл .sql з'являється у каталозі backups/, запис у таблиці backup_jobs має статус completed, бекап відображається у таблиці історії з коректними даними (розмір, тривалість, дата).

Фактичний результат:

Бекап було успішно створено, файл збережено, запис у БД має статус completed, дані відображаються коректно.

Висновок: у результаті виконання Тесту 1 функціональна вимога щодо створення резервних копій виконана.

Методика проведення Тесту 2 (Перевірка відновлення з бекапу):

- Користувач відкриває сторінку управління бекапами (backups.html).
- Обирає у таблиці історії бекапів будь-який запис зі статусом completed.
- Натискає кнопку «Відновити» та підтверджує дію у модальному вікні.
- Система створює Docker-контейнер з образом mysql:8.0, виконує команду mysql < backup.sql, відновлюючи дані з бекапу в цільову базу даних.
- Перевіряється цілісність відновлених даних.

Очікуваний результат:

База даних успішно відновлюється з бекапу, всі дані (структура таблиць, записи) відповідають стану на момент створення бекапу, помилок під час відновлення не виникає.

Фактичний результат:

Базу даних відновлено, дані коректні, помилок не виявлено.

Висновок: у результаті виконання Тесту 2 функціональна вимога щодо відновлення з резервних копій виконана.

Методика проведення Тесту 3 (Перевірка скачування бекапів):

- Користувач відкриває сторінку управління бекапами (backups.html).
- Обирає у таблиці історії бекапів будь-який запис зі статусом completed.
- Натискає кнопку «Скачати SQL».
- Система формує та відправляє клієнту файл .sql.
- Повторює дію для кнопки «Скачати JSON».

Очікуваний результат:

База даних успішно відновлюється з бекапу, всі дані (структура таблиць, записи) відповідають стану на момент створення бекапу, помилок під час відновлення не виникає.

Фактичний результат:

Обидва файли скачано, вміст відповідає очікуваному.

Висновок: у результаті виконання Тесту 3 функціональна вимога щодо скачування бекапів виконана.

Методика проведення Тесту 4 (Перевірка CRUD-операцій для пристроїв):

- Користувач відкриває сторінку управління пристроями (devices.html).
- Натискає кнопку «Додати пристрій», заповнює форму (назва, тип, IP-адреса, статус) та зберігає.
- Перевіряє, чи з'явився новий пристрій у таблиці.
- Редагує створений пристрій (змінює назву або статус).
- Видаляє створений пристрій.

Очікуваний результат:

Пристрій успішно створюється, відображається в таблиці, редагується та видаляється. Всі операції виконуються без помилок, дані коректно зберігаються в БД.

Фактичний результат:

Всі CRUD-операції виконано успішно.

Висновок: у результаті виконання Тесту 4 функціональна вимога щодо управління пристроями виконана.

Методика проведення Тесту 5 (Перевірка навантажувальної стійкості):

- Запускається інструмент навантажувального тестування k6.
- Генерується навантаження у 500 віртуальних користувачів протягом 30 секунд.
- Імітуються HTTP-запити до основних API-ендпоінтів (отримання списку бекапів, пристроїв, інцидентів).
- Вимірюються: середній час відповіді, 95-й перцентиль часу відповіді, завантаження CPU, використання RAM, відсоток успішних запитів.

Очікуваний результат:

Середній час відповіді API $\leq$ 1 с, 95-й перцентиль часу відповіді $\leq$ 2 с, завантаження CPU $\leq$ 50 %, використання RAM $\leq$ 1 ГБ, відсоток успішних запитів $\geq$ 99 %.

Фактичний результат:

Середній час відповіді API — 0.42 с, 95-й перцентиль — 0.87 с, завантаження CPU — 32 %, використання RAM — 340 МБ, відсоток успішних запитів — 99.8 %.

Висновок: у результаті виконання Тесту 5 нефункціональні вимоги щодо продуктивності виконані.

Висновок за результатами проведених випробувань

Програмний виріб вважається таким, що пройшов випробування в цілому, оскільки успішно виконані всі тести (1–5). Загалом, тестування показало, що веб-застосунок працює коректно та забезпечує ефективне резервне копіювання та відновлення баз даних MySQL з використанням Docker-контейнерів, а також

повноцінне управління інфраструктурою дата-центру (DCIM). Система витримує навантаження у 500 одночасних користувачів із середнім часом відповіді 0.42 с, захищена від SQL-ін'єкцій завдяки використанню PDO, а веб-інтерфейс з графіками Chart.js забезпечує зручну візуалізацію даних.

Система рекомендована до впровадження в IT-інфраструктурі підприємств для резервного копіювання баз даних, а також для використання в навчальному процесі при вивченні дисциплін «Адміністрування баз даних» та «Системи резервного копіювання».

Виконавець: Масленников Владислав Анатолійович 

Додаток Г

Програмний код реалізації datacenters.php

```
<?php
header('Content-Type: application/json; charset=utf-8');
header('Access-Control-Allow-Origin: *');
header('Access-Control-Allow-Methods: GET, POST, PUT, DELETE,
OPTIONS');
header('Access-Control-Allow-Headers: Content-Type');
require_once '../config/database.php';

$database = new Database();
$db = $database->getConnection();
$method = $_SERVER['REQUEST_METHOD'];

if ($method === 'OPTIONS') { http_response_code(200); exit; }

switch($method) {
    case 'GET':
        if(isset($_GET['id'])) {
            $stmt = $db->prepare("
                SELECT dc.*,
                       COUNT(DISTINCT sr.id) as rooms_count,
                       COUNT(DISTINCT r.id) as racks_count,
                       COUNT(DISTINCT d.id) as devices_count
                FROM data_centers dc
                LEFT JOIN server_rooms sr ON dc.id =
sr.data_center_id
                LEFT JOIN rack_rows rr ON sr.id =
rr.server_room_id
                LEFT JOIN racks r ON rr.id = r.rack_row_id
                LEFT JOIN devices d ON r.id = d.rack_id
                WHERE dc.id = ?
                GROUP BY dc.id
            ");
            $stmt->execute([$_GET['id']]);
```

```
            $result = $stmt->fetch();
            echo json_encode($result ? $result : null,
JSON_UNESCAPED_UNICODE);
        } else {
            $stmt = $db->query("
                SELECT dc.*,
                       COUNT(DISTINCT sr.id) as rooms_count,
                       COUNT(DISTINCT r.id) as racks_count,
                       COUNT(DISTINCT d.id) as devices_count
                FROM data_centers dc
                LEFT JOIN server_rooms sr ON dc.id =
sr.data_center_id
                LEFT JOIN rack_rows rr ON sr.id =
rr.server_room_id
                LEFT JOIN racks r ON rr.id = r.rack_row_id
                LEFT JOIN devices d ON r.id = d.rack_id
                GROUP BY dc.id
                ORDER BY dc.created_at DESC
            ");
            echo json_encode($stmt->fetchAll(),
JSON_UNESCAPED_UNICODE);
        }
        break;

    case 'POST':
        $data = json_decode(file_get_contents("php://input"),
true);

        if (empty($data['name']) || empty($data['code'])) {
            echo json_encode(['success' => false, 'error' =>
'Назва і код обов\'язкові!']);
            exit;
        }

        $stmt = $db->prepare("
            INSERT INTO data_centers
```

```
            (name, code, address, city, country, total_area_sqm,
             power_capacity_kw, cooling_capacity_kw, tier_level,
             status, contact_person, contact_phone, contact_email)
            VALUES (?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?)
        ");

        $stmt->execute([
            $data['name'],
            $data['code'],
            $data['address'] ?? null,
            $data['city'] ?? null,
            $data['country'] ?? null,
            $data['total_area_sqm'] ?? null,
            $data['power_capacity_kw'] ?? null,
            $data['cooling_capacity_kw'] ?? null,
            $data['tier_level'] ?? 'Tier III',
            $data['status'] ?? 'active',
            $data['contact_person'] ?? null,
            $data['contact_phone'] ?? null,
            $data['contact_email'] ?? null
        ]);

        echo json_encode([
            'success' => true,
            'id' => $db->lastInsertId(),
            'message' => 'Дата-центр успішно створено!'
        ], JSON_UNESCAPED_UNICODE);
        break;

    case 'PUT':
        $data = json_decode(file_get_contents("php://input"),
true);

        if (empty($data['id'])) {
            echo json_encode(['success' => false, 'error' => 'ID
обов\'язковий!']);
```

```
            exit;
        }

        $stmt = $db->prepare("
            UPDATE data_centers
            SET name=?, code=?, address=?, city=?, country=?,
                total_area_sqm=?, power_capacity_kw=?,
cooling_capacity_kw=?,
                tier_level=?, status=?, contact_person=?,
                contact_phone=?, contact_email=?
            WHERE id=?
        ");

        $stmt->execute([
            $data['name'],
            $data['code'],
            $data['address'] ?? null,
            $data['city'] ?? null,
            $data['country'] ?? null,
            $data['total_area_sqm'] ?? null,
            $data['power_capacity_kw'] ?? null,
            $data['cooling_capacity_kw'] ?? null,
            $data['tier_level'] ?? 'Tier III',
            $data['status'] ?? 'active',
            $data['contact_person'] ?? null,
            $data['contact_phone'] ?? null,
            $data['contact_email'] ?? null,
            $data['id']
        ]);

        echo json_encode([
            'success' => true,
            'message' => 'Дата-центр успішно оновлено!'
        ], JSON_UNESCAPED_UNICODE);
        break;
```

```
    case 'DELETE':
        if (isset($_GET['id'])) {
            $stmt = $db->prepare("DELETE FROM data_centers WHERE
id=?");
            $stmt->execute([$_GET['id']]);
            echo json_encode([
                'success' => true,
                'message' => 'Дата-центр видалено!'
            ], JSON_UNESCAPED_UNICODE);
        } else {
            echo json_encode(['success' => false, 'error' => 'ID
не вказано!']);
        }
        break;
}
?>
```