

Міністерство освіти і науки України
Державний заклад
«Луганський національний університет імені Тараса Шевченка»

Факультет технологій та інформаційних систем

Кафедра інформаційних технологій та систем

Діденко Максим Вячеславович

РОЗРОБКА СИСТЕМИ КОНТРОЛЮ ДОСТУПУ ДО КОРПОРАТИВНИХ РЕСУРСІВ

кваліфікаційна робота

здобувача вищої освіти першого (бакалаврського) рівня
освітньої програми «Інженерія програмного забезпечення»
за спеціальністю 121 Інженерія програмного забезпечення

Особистий підпис _____  _____ Максим ДІДЕНКО

Науковий керівник _____  Світлана ПЕРЕЯСЛАВСЬКА,
кандидат педагогічних наук, доцент
кафедри інформаційних технологій
та систем

Завідувач кафедри _____  Микола СЕМЕНОВ,
кандидат педагогічних наук, доцент
кафедри інформаційних технологій
та систем

Лубни – 2026

Міністерство освіти і науки України	
Державний заклад «Луганський національний університет імені Тараса Шевченка»	
Факультет (інститут)	Навчально-науковий інститут математики та інформаційних технологій
	(повна назва)
Кафедра	Інформаційних технологій та систем
	(повна назва)
Освітній ступень	Бакалавр
	(код, назва)
Напрямок підготовки	Інженерія програмного забезпечення
	(код, назва)

ЗАТВЕРДЖУЮ

Завідувач кафедри ІТС
М.А. Семенов

(підпис) (ініціали, прізвище)

“ ” 2025 р.

**ЗАВДАННЯ
НА ДИПЛОМНИЙ ПРОЕКТ (РОБОТУ) СТУДЕНТУ**

Діденку Максиму Вячеславовичу

(прізвище, ім'я, по батькові)

1. Тема проекту (роботи) **Розробка системи контролю доступу до корпоративних ресурсів**

Керівник кваліфікаційної роботи

Переяславська С.О. к.п.н., доцент

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

затверджена наказом по університету

Від

2. Строк подання студентом проекту (роботи)

3. Вихідні дані до роботи (проекту)

Розроблено систему контролю доступу до корпоративних ресурсів

на основі рольової моделі
авторизації.

(визначаються кількісні або (та) якісні показники, яким повинен відповідати об'єкт розробки)

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити) **НАУКОВО-ДОСЛІДНА ЧАСТИНА.**

РОЗРОБКА СИСТЕМИ КОНТРОЛЮ ДОСТУПУ ДО КОРПОРАТИВНИХ РЕСУРСІВ

(визначаються назви розділів або (та) перелік питань, які повинні увійти до тексту ПЗ)

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень)

6. Консультанти розділів проекту/роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв

7. Дата видачі завдання „_____” _____ 2025р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломного проекту (роботи)	Строк виконання етапів проекту (роботи)	Примітка
	Вибір теми роботи, вивчення наукової літератури, затвердження теми та керівника.	До 15 жовтня	
	Аналіз літературних джерел за темою роботи. Розробка та апробація методики дослідно-експериментальної роботи. Подання структури теоретичної частини роботи та плану експериментальних досліджень.	Другий тиждень листопада (10 листопада)	
	Робота над теоретичною частиною. Подання теоретичної частини роботи для першого читання науковим керівником.	До 15 грудня	
	Усунення зауважень, урахування рекомендацій наукового керівника. Подання теоретичної частини роботи на друге читання.	До 28 січня	
	Проведення експериментальної роботи. Поетапний аналіз та обговорення її результатів. Перевірка стану виконання роботи.	Перший тиждень березня	
	Урахування рекомендацій наукового керівника, усунення недоліків, підготовка варіанта роботи до передзахисту. Розробка презентації.	До 31 березня	
	Попередній захист роботи на кафедрі	квітень	
	Доопрацювання роботи з урахуванням рекомендацій після передзахисту. Подання роботи науковому керівникові та рецензентові на підготовку відгуку та рецензії	За 10 днів до державної атестації	
	Подання на кафедру остаточного варіанта роботи, переплетеного та підписаного автором, науковим керівником і рецензентом.	За 5 днів до державної атестації	

Студент


підпис

Керівник проекту (роботи)


підпис

М. В. Діденко

(ініціали, прізвище)

С. О. Переяславська

(ініціали, прізвище)

АНОТАЦІЯ

Діденко М. В.

Тема: Розробка системи контролю доступу до корпоративних ресурсів на основі рольової моделі авторизації.

Спеціальність: 121 «Інженерія програмного забезпечення»

Установа: ЛНУ імені Тараса Шевченка, 2026 р.

Бакалаврська робота містить: 61 с., 15 рис., 12 табл., 22 джерел.

Об'єкт дослідження - процес управління доступом користувачів до корпоративних інформаційних ресурсів.

Предмет дослідження - методи, моделі та програмні засоби реалізації системи контролю доступу на основі рольової моделі авторизації

Мета роботи - розробка та програмна реалізація системи контролю доступу до корпоративних ресурсів із використанням сучасних методів автентифікації та рольової моделі авторизації.

Результати роботи.

У результаті виконання роботи спроектовано трирівневу клієнт-серверну архітектуру системи, розроблено рольову модель RBAC та структуру реляційної бази даних PostgreSQL. Реалізовано серверну частину на Node.js з фреймворком Express і клієнтський інтерфейс на React.

Забезпечено механізми безпечної автентифікації з bcrypt та JWT-токенами, авторизації на основі RBAC, журналювання дій користувачів та захисту REST API від несанкціонованого доступу.

Проведено комплексне тестування системи. Оцінка продуктивності підтвердила стабільну роботу при навантаженні до 100 одночасних запитів без виникнення помилок.

Ключові слова. система контролю доступу, RBAC, JWT, Node.js, React, PostgreSQL, автентифікація, авторизація, корпоративна безпека.

ABSTRACT

Didenko Maksym

Theme: Development of an Access Control System for Corporate Resources Based on a Role-Based Authorization Model.

Speciality: 123 "Software Engineering"

Institution: Luhansk Taras Shevchenko National University (LTSNU), 2026.

Diploma work contains: 61 pages, 15 Fig., 12 Table, 22 source.

A research object is the process of managing user access to corporate information resources.

An aim of work is the development and software implementation of an access control system for corporate resources using modern authentication methods and a role-based authorization model.

Job performances. As a result of the work, a three-tier client-server architecture was designed, an RBAC role model and a PostgreSQL relational database structure were developed. The server side was implemented using Node.js with the Express framework, and the client interface using React.

Secure authentication mechanisms using bcrypt and JWT tokens, RBAC-based authorization, user action logging, and REST API protection against unauthorized access were implemented.

Comprehensive system testing was conducted. Performance evaluation confirmed stable operation under a load of up to 100 concurrent requests without errors.

Keywords. Access control system, RBAC, JWT, Node.js, React, PostgreSQL, authentication, authorization, corporate security.

Міністерство освіти і науки України
Державний заклад «Луганський національний університет
імені Тараса Шевченка»

Факультет (інститут)

Технологій та інформаційних систем

(повна назва)

Кафедра

Інформаційних технологій та систем

(повна назва)

ТЕХНІЧНЕ ЗАВДАННЯ

на виконання програмної розробки (ПР):

**«РОЗРОБКА СИСТЕМИ КОНТРОЛЮ ДОСТУПУ
ДО КОРПОРАТИВНИХ РЕСУРСІВ»**

ІТС. ІПЗ4.2026-02-ТЗ

ПОГОДЖЕНО

Керівник кваліфікаційної роботи

Переяславська С.О.

“ ”  2026р.

ВИКОНАВЕЦЬ

Студент групи 4 ІПЗ

Діденко М.В.

“ ”  2026р.

Лубни – 2026

ЗМІСТ

ВСТУП	3
1. ХАРАКТЕРИСТИКА ОБ'ЄКТА	3
2. ПРИЗНАЧЕННЯ РОЗРОБКИ	3
3. ОСНОВНІ ВИМОГИ ДО ПРОГРАМНОГО КОМПЛЕКСУ	4
4. ТЕХНІКО-ЕКОНОМІЧНІ ВИМОГИ ДО КІНЦЕВОГО ПРОДУКТУ	5
5. ВИМОГИ ДО МАТЕРІАЛІВ І КОМПЛЕКТУЮЧИХ	5
6. ЕТАПИ ВИКОНАННЯ ПР	5
7. ПРИЙОМ	6
8. ПОРЯДОК ВНЕСЕННЯ ЗМІН ДО ТЕХНІЧНОГО ЗАВДАННЯ, ЩО ЗАТВЕРДЖЕНО	7

ВСТУП

- 1.1 Найменування: Система контролю доступу до корпоративних ресурсів.
- 1.2 Шифр ПР: АДР-2
- 1.3 Підстава до виконання ПР: Підставою для виконання даної розробки є необхідність забезпечення безпечного та керованого доступу користувачів до корпоративних інформаційних ресурсів організації на основі рольової моделі авторизації (RBAC).
- 1.4 Терміни розробки:
 - 1.4.1 Початок: 30 жовтня 2025 р.
 - 1.4.2 Закінчення: 20 березня 2026 р.
- 1.5 Фінансується за рахунок коштів замовника.

1. ХАРАКТЕРИСТИКА ОБ'ЄКТА

- 1.1. Розроблювана система повинна забезпечувати функції контролю та управління доступом. До складу об'єкту, що створюється, повинно входити:
 - 1.1.1. Серверна частина (back-end) на базі Node.js з фреймворком Express.
 - 1.1.2. Клієнтський інтерфейс (front-end) на базі React.
 - 1.1.3. Реляційна база даних PostgreSQL.
- 1.2. До вхідної інформації належать вимоги замовника щодо розмежування прав доступу, переліків ролей та ресурсів.

2. ПРИЗНАЧЕННЯ РОЗРОБКИ

- 2.1. Призначення: веб-система контролю доступу до корпоративних ресурсів на основі рольової моделі авторизації.
- 2.2. Основні критерії ефективності:
 - 2.2.1. Зручний та інтуїтивний інтерфейс:

- 2.2.1.1. Адміністратор повинен мати можливість управляти ролями та правами доступу через веб-інтерфейс;
- 2.2.1.2. Користувач повинен мати доступ лише до ресурсів, що відповідають його ролі.
- 2.2.2. Надійна автентифікація:
 - 2.2.2.1. Використання JWT-токенів для підтримки сесій;
 - 2.2.2.2. Хешування паролів з використанням bcrypt.

3. ОСНОВНІ ВИМОГИ ДО ПРОГРАМНОГО КОМПЛЕКСУ

3.1. Загальні вимоги

- 3.1.1. Програмний комплекс функціонує на сучасних операційних системах (Windows 10/11, Linux, macOS), що підтримують Node.js версії 18 або вище;
- 3.1.2. Клієнтська частина повинна коректно відображатися в сучасних браузерях (Chrome, Firefox, Edge, Safari);
- 3.1.3. Система повинна забезпечувати стабільну роботу при навантаженні до 100 одночасних запитів без виникнення помилок.

3.2. Додаткові вимоги

- 3.2.1. Мова програмування: JavaScript (Node.js, React). База даних: PostgreSQL.
- 3.2.2. Вимоги до ліцензійного ПЗ не передбачаються і вирішуються замовником.

3.3. Вимоги до складу і архітектури

- 3.3.1. Трирівнева клієнт-серверна архітектура: клієнт (React), сервер (Node.js/Express), база даних (PostgreSQL);
- 3.3.2. Реалізація рольової моделі доступу RBAC з підтримкою ролей: адміністратор, менеджер, користувач.

3.4. Вимоги до якості і надійності

- 3.4.1. Програмний комплекс повинен надійно працювати без збоїв.

3.4.2. Розробник гарантує захист REST API від несанкціонованого доступу та журналювання дій користувачів.

3.4.3. Всі паролі повинні зберігатися виключно у хешованому вигляді (bcrypt).

3.5. Вимоги до експлуатації

3.5.1. Для розгортання системи необхідний сервер з встановленими Node.js, PostgreSQL та доступом до мережі.

4. ТЕХНІКО-ЕКОНОМІЧНІ ВИМОГИ ДО КІНЦЕВОГО ПРОДУКТУ

Вартість робіт по розробці даної ПР визначається згідно з договором на розробку. Вартість пропонувананих аналогів повинна забезпечити економічну доцільність їх застосування.

5. ВИМОГИ ДО МАТЕРІАЛІВ І КОМПЛЕКТУЮЧИХ

5.1. Вимоги до екологічної безпеки при експлуатації.

Не пред'являються.

5.2. Спеціальні вимоги до кінцевого продукту.

Не пред'являються.

5.3. Вимоги до безпеки для населення при експлуатації продукції.

Не пред'являються.

6. ЕТАПИ ВИКОНАННЯ ПР

Етапи виконання ПР можуть уточнюватися згідно календарного плану робіт за погодженням між замовником і виконавцем.

№	Етапи виконання роботи	Термін виконання і обсяг робіт	Звітні матеріали
1	Аналіз предметної галузі. Проектування архітектури системи та моделі бази даних. Розробка першої версії серверної частини. Попереднє тестування.	Фрагмент програмного комплексу на ЕОМ замовника, який виконує основні функції і звітна документація п.8.2	
2	Розробка клієнтського інтерфейсу на React. Реалізація RBAC-авторизації та JWT-автентифікації. Інтеграція клієнта і сервера. Тестування.	Готовий програмний комплекс на ЕОМ замовника і звітна документація п.8.2	
3	Доопрацювання окремих модулів. Комплексне тестування продуктивності. Розробка звітних матеріалів.	Звітні матеріали згідно з пунктом 8.	

7. ПРИЙОМ

7.1. Необхідні вимоги для впровадження ПР і завершення робіт.

Оцінка результатів розробки і доцільність її продовження здійснюється замовником за поданням наступних матеріалів:

- встановлений програмний комплекс на ЕОМ замовника;
- перелік файлів на резервному носії;
- короткий опис роботи ПР та опис усіх файлів, необхідних для роботи ПР.
- перелік документів:
- Технічне завдання

- Пояснювальна записка
- Програма і методика тестування
- Керівництво користувача

7.2. Перелік звітних документів, необхідних для прийняття етапів роботи:

- короткий опис результатів етапу у вигляді анотованого звіту (для 1 та 2 етапів);
- частковий програмний комплекс на ЕОМ замовника згідно календарного плану робіт;
- акт приймання продукції.

Звітні матеріали подаються у вигляді звітів на папері по ГОСТ 7.32-91.

7.3. Загальний перелік до прийому звітних документів, макетів, експериментальних зразків.

До прийому пред'являються: акт здачі-приймання продукції, акт впровадження ПР.

7.4. Тестування ПР.

Тестування виконується за «Програмою і методикою тестування», яка розробляється виконавцем і затверджується замовником.

8. ПОРЯДОК ВНЕСЕННЯ ЗМІН ДО ТЕХНІЧНОГО ЗАВДАННЯ, ЩО ЗАТВЕРДЖЕНО

Дане технічне завдання може уточнюватися в процесі розробки ПР при узгодженні сторін з оформленням доповнень до ТЗ.

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

**ДЕРЖАВНИЙ ЗАКЛАД „ЛУГАНСЬКИЙ НАЦІОНАЛЬНИЙ
УНІВЕРСИТЕТ ІМЕНІ ТАРАСА ШЕВЧЕНКА”**

Факультет технологій та інформаційних систем

Кафедра інформаційних технологій та систем

Пояснювальна записка
до кваліфікаційної роботи
за першим (бакалаврським) рівнем освіти

на тему: **РОЗРОБКА СИСТЕМИ КОНТРОЛЮ ДОСТУПУ ДО
КОРПОРАТИВНИХ РЕСУРСІВ**

Виконав: здобувач вищої освіти
4 курсу спеціальності
121 «Інженерія програмного
забезпечення»

(шифр і назва напрямку підготовки, спеціальності)

Діденко М. В.

(прізвище та ініціали)

Керівник Перейславська С.О.

(прізвище та ініціали)

Рецензент Козуб Ю.Г.

(прізвище та ініціали)

Лубни – 2026

ЗМІСТ

ВСТУП	3
РОЗДІЛ 1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ	6
1.1 Корпоративні інформаційні системи та їх особливості	6
1.2 Основні загрози інформаційній безпеці	6
1.3 Методи аутентифікації користувачів.....	7
1.4 Методи авторизації	8
1.5 Аналіз існуючих систем контролю доступу.....	8
1.6 Постановка задачі дослідження.....	10
РОЗДІЛ 2 ПРОЄКТУВАННЯ СИСТЕМИ КОНТРОЛЮ ДОСТУПУ	12
2.1 Архітектура системи контролю доступу	12
2.2 Модель ролей та прав доступу (RBAC).....	14
2.3 Проєктування бази даних системи	17
2.4 Опис алгоритмів аутентифікації та авторизації	20
2.5 Моделювання бізнес-процесів.....	24
РОЗДІЛ 3. ПРОГРАМНА РЕАЛІЗАЦІЯ	29
3.1 Обґрунтування вибору технологій розробки системи	29
3.2 Реалізація серверної частини системи контролю доступу	31
3.6 Реалізація клієнтської частини системи	43
3.7 Забезпечення безпеки системи контролю доступу.....	46
РОЗДІЛ 4. ТЕСТУВАННЯ СИСТЕМИ.....	51
4.1 Мета та задачі тестування	51
4.2 Види тестування системи.....	52
4.3 Розробка тестових сценаріїв	54
4.5 Оцінка продуктивності та надійності системи	58
ВИСНОВКИ.....	62
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ	64

ВСТУП

У сучасних умовах цифрової трансформації підприємств інформаційні системи стали критично важливим елементом функціонування бізнесу. Корпоративні ресурси – бази даних, файлові сховища, внутрішні веб-портали, CRM- та ERP-системи – містять конфіденційну інформацію, втрати або компрометація якої можуть призвести до фінансових збитків, репутаційних ризиків та порушення безперервності бізнес-процесів. У зв'язку з цим особливої актуальності набуває забезпечення ефективного контролю доступу до корпоративних ресурсів.

Зростання кількості кіберзагроз, поширення віддаленого формату роботи, використання хмарних сервісів та мобільних пристроїв суттєво ускладнюють завдання управління доступом користувачів до інформаційних систем. Несанкціонований доступ, компрометація облікових записів, використання вразливостей прикладного програмного забезпечення є одними з найпоширеніших інцидентів інформаційної безпеки. Традиційні підходи до автентифікації та авторизації часто не забезпечують достатнього рівня гнучкості, масштабованості та захищеності.

Система контролю доступу є ключовим компонентом архітектури безпеки корпоративної інформаційної системи. Вона повинна забезпечувати надійну автентифікацію користувачів, коректну авторизацію відповідно до їх ролей і повноважень, ведення журналів подій, а також можливість централізованого управління правами доступу. Однією з найбільш поширених і практично доцільних моделей є рольова модель контролю доступу (RBAC), яка дозволяє формалізувати політику безпеки через систему ролей та дозволів.

Актуальність теми бакалаврської роботи зумовлена необхідністю розробки програмного рішення, яке поєднує сучасні механізми автентифікації, ефективну модель авторизації та забезпечує високий рівень захисту корпоративних ресурсів. Створення власної системи контролю доступу дозволяє глибше дослідити архітектурні підходи, алгоритми перевірки прав доступу та механізми забезпечення інформаційної безпеки на рівні програмної реалізації.

Метою роботи є розробка та програмна реалізація системи контролю доступу до корпоративних ресурсів із використанням сучасних методів автентифікації та рольової моделі авторизації.

Для досягнення поставленої мети необхідно вирішити такі **завдання**:

1. Проаналізувати особливості корпоративних інформаційних систем та основні загрози інформаційній безпеці.
2. Дослідити існуючі моделі контролю доступу та методи автентифікації користувачів.
3. Обґрунтувати вибір архітектури системи та технологічного стеку.
4. Розробити модель ролей і прав доступу до корпоративних ресурсів.
5. Спроекувати структуру бази даних системи контролю доступу.
6. Реалізувати програмні модулі автентифікації та авторизації користувачів.
7. Забезпечити механізми захисту системи від типових атак.
8. Провести тестування розробленого програмного продукту та проаналізувати результати.

Об'єктом дослідження є процес управління доступом користувачів до корпоративних інформаційних ресурсів.

Предметом дослідження є методи, моделі та програмні засоби реалізації системи контролю доступу на основі рольової моделі авторизації.

У процесі виконання роботи застосовуються такі методи дослідження: аналіз наукових і технічних джерел, системний аналіз, об'єктно-орієнтоване проєктування, моделювання бізнес-процесів засобами UML, проєктування реляційних баз даних, методи тестування програмного забезпечення.

Практичне значення отриманих результатів полягає у створенні працездатної системи контролю доступу, яка може бути використана як основа для впровадження в корпоративному середовищі або інтеграції з іншими інформаційними системами. Розроблене програмне рішення забезпечує централізоване управління користувачами, ролями та правами доступу, а також підвищує рівень інформаційної безпеки організації.

Структура роботи складається зі вступу, чотирьох розділів, висновків, списку використаних джерел та додатків. У першому розділі здійснюється аналіз предметної області, у другому – проектування системи, у третьому – програмна реалізація, у четвертому – тестування та оцінка ефективності розробленого рішення.

РОЗДІЛ 1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Корпоративні інформаційні системи та їх особливості

Корпоративна інформаційна система (KIC) – це сукупність програмних, апаратних та організаційних засобів, що забезпечують автоматизацію бізнес-процесів підприємства, зберігання та обробку даних, а також підтримку управлінських рішень.

До основних типів корпоративних ресурсів належать:

- системи управління ресурсами підприємства (ERP);
- системи управління взаємовідносинами з клієнтами (CRM);
- внутрішні портали та документообіг;
- файлові сервери;
- хмарні сервіси;
- бази даних;
- REST API для інтеграції з іншими системами.

Особливостями корпоративних систем є:

1. Багаторівнева архітектура (клієнт – сервер – база даних).
2. Велика кількість користувачів з різними ролями.
3. Розподілений доступ (локальна мережа, VPN, Інтернет).
4. Високі вимоги до безпеки та відмовостійкості.
5. Необхідність аудиту дій користувачів.

З огляду на це система контролю доступу повинна бути масштабованою, централізованою та інтегрованою з іншими компонентами інфраструктури.

1.2 Основні загрози інформаційній безпеці

Контроль доступу безпосередньо пов'язаний із протидією загрозам інформаційної безпеки. Найпоширенішими з них є:

1. Несанкціонований доступ

Отримання доступу до ресурсів без відповідних прав через вразливості системи або компрометацію облікових даних.

2. Атаки типу brute-force

Автоматизований підбір паролів.

3. SQL-ін'єкції

Впровадження шкідливих SQL-запитів через форми введення.

4. Викрадення токенів доступу

Перехоплення або підробка JWT-токенів.

5. Ескалація привілеїв

Отримання користувачем прав, що перевищують його повноваження.

6. Внутрішні загрози

Зловживання доступом співробітниками.

1.3 Методи аутентифікації користувачів

Аутентифікація – це процес підтвердження особи користувача. Сучасні системи використовують різні механізми:

Парольна аутентифікація

Найпоширеніший метод, що базується на перевірці логіна та пароля.

Недоліки:

- низька стійкість до підбору;
- залежність від складності пароля.

Двофакторна аутентифікація (2FA)

Поєднання двох факторів:

- знання (пароль),
- володіння (SMS-код, додаток),
- біометрія.

Token-based аутентифікація

Використання JWT-токенів для підтвердження сесії.

Single Sign-On (SSO)

Єдина точка входу до декількох систем.

1.4 Методи авторизації

Авторизація – це визначення дозволених дій користувача після успішної аутентифікації.

DAC (Discretionary Access Control)

Доступ визначається власником ресурсу.

MAC (Mandatory Access Control)

Доступ регламентується централізованими політиками безпеки.

RBAC (Role-Based Access Control)

Доступ надається на основі ролей користувачів.

Переваги RBAC:

- простота адміністрування;
- масштабованість;
- логічна структура прав.

ABAC (Attribute-Based Access Control)

Рішення приймається на основі атрибутів користувача, ресурсу та середовища.

У рамках цієї роботи обрано модель RBAC як оптимальну для корпоративного середовища середнього масштабу.

1.5 Аналіз існуючих систем контролю доступу

Сучасний ринок пропонує різноманітні рішення для управління доступом.

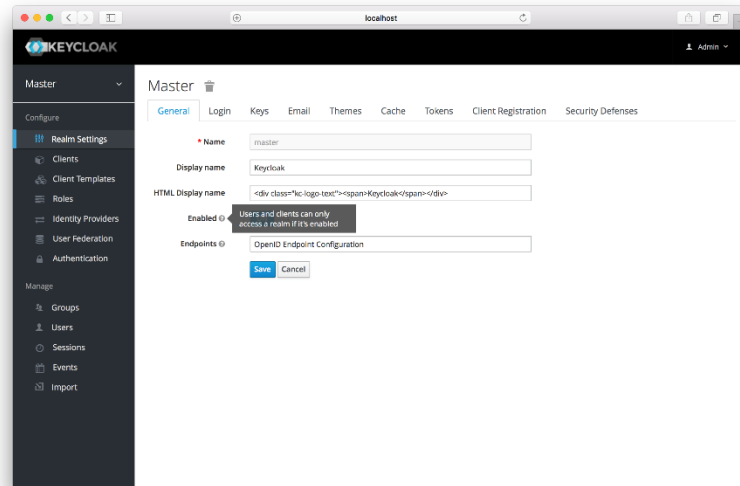


Рис 1.1. Keycloak

Open-source платформа для управління ідентифікацією та доступом. Підтримує OAuth2, OpenID Connect, SSO. Недолік – складність розгортання та адміністрування.

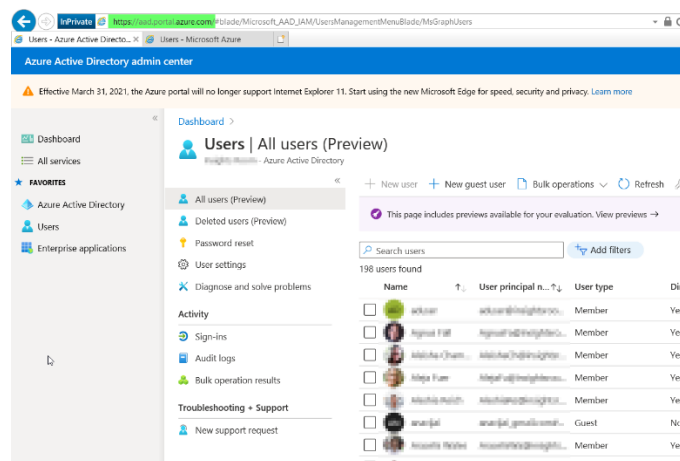


Рис 1.2. Microsoft Azure Active Directory

Хмарна система управління ідентифікацією. Переваги – масштабованість, інтеграція з екосистемою Microsoft. Недолік – комерційна ліцензія та залежність від хмарної інфраструктури.

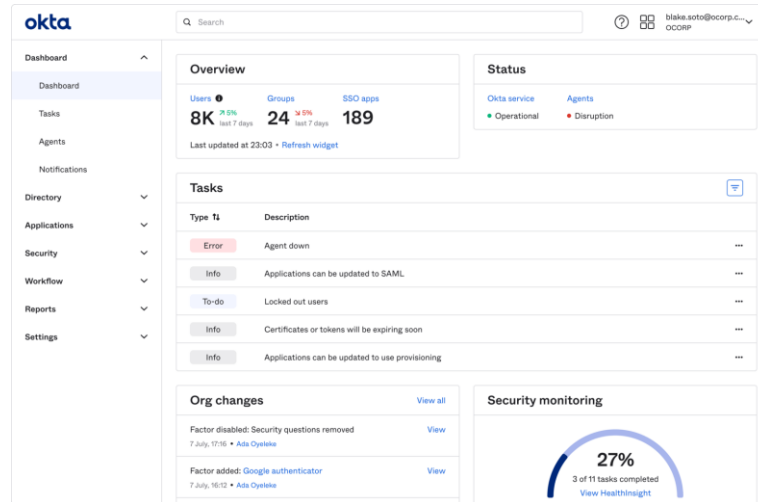


Рис 1.2. Okta

Комерційна SaaS-платформа для управління доступом. Переваги – висока надійність та готові інтеграції. Недоліки – висока вартість для середнього бізнесу.

1.6 Постановка задачі дослідження

На основі проведеного аналізу формулюється задача:

Розробити систему контролю доступу до корпоративних ресурсів, яка: реалізує рольову модель доступу (RBAC);

- підтримує безпечну аутентифікацію користувачів;
- використовує токени доступу (JWT);
- забезпечує аудит дій користувачів;
- має зручний інтерфейс адміністрування;
- відповідає вимогам інформаційної безпеки.

Система повинна бути реалізована у вигляді клієнт-серверного застосунку з REST API та реляційною базою даних.

Висновки до розділу 1

У першому розділі було здійснено комплексний аналіз предметної області дослідження. Розглянуто архітектурні особливості, класифікацію та специфіку функціонування сучасних корпоративних інформаційних систем, а також систематизовано ключові загрози інформаційній безпеці, зокрема несанкціонований доступ, SQL-ін'єкції, ескалацію привілеїв та компрометацію облікових даних.

На основі порівняльного аналізу методів автентифікації та моделей розмежування прав (DAC, MAC, RBAC, ABAC) обґрунтовано доцільність застосування рольової моделі контролю доступу (RBAC) в межах сучасної концепції Zero Trust («Нульової довіри») як найбільш ефективного та збалансованого рішення для підприємств середнього масштабу. Дослідження існуючих IAM-платформ (Keycloak, Microsoft Azure AD, Okta) виявило їхні обмеження у вигляді високої вартості комерційного ліцензування чи значної складності адміністрування. Це дозволило чітко сформулювати технічні вимоги та поставити інженерну задачу на розробку власної гнучкої клієнт-серверної системи контролю доступу з REST API та реляційною базою даних.

РОЗДІЛ 2 ПРОЄКТУВАННЯ СИСТЕМИ КОНТРОЛЮ ДОСТУПУ

2.1 Архітектура системи контролю доступу

Проектування системи контролю доступу розпочинається з визначення її архітектурної моделі. Оскільки система передбачає програмну реалізацію та взаємодію між клієнтською частиною, сервером і базою даних, доцільним є використання трирівневої клієнт-серверної архітектури (Three-Tier Architecture).

Обґрунтування вибору архітектури

Трирівнева модель дозволяє:

- розділити відповідальність між компонентами;
- підвищити масштабованість;
- забезпечити ізоляцію бізнес-логіки;
- спростити тестування та модифікацію системи.

Архітектура складається з таких рівнів:

1. Presentation Layer (Клієнтський рівень)

Відповідає за взаємодію з користувачем.

Реалізується у вигляді веб-інтерфейсу.

2. Application Layer (Серверна логіка) Реалізує:

- аутентифікацію,
- авторизацію,
- управління ролями,
- логування подій,
- бізнес-правила.

3. Data Layer (Рівень даних) Реляційна база даних, що зберігає:

- користувачів,
- ролі,
- дозволи,
- ресурси,
- журнали доступу.

Архітектурна схема системи

Найбільш доречною для цієї роботи є схема, що відображає:

- клієнт (браузер),
- REST API сервер,
- модуль аутентифікації,
- модуль авторизації (RBAC),
- базу даних,
- журналювання подій.

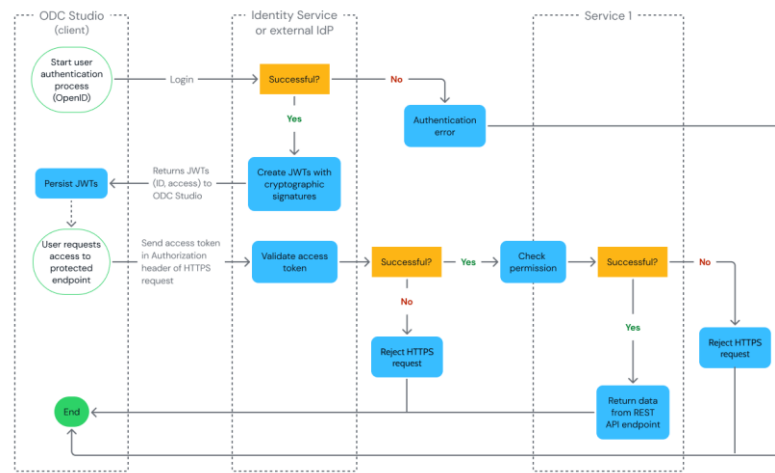


Рис. 2.1. Архітектурна схема системи

Логіка взаємодії компонентів

Алгоритм роботи системи при доступі до ресурсу:

1. Користувач надсилає запит на сервер через веб-інтерфейс.
2. Сервер перевіряє наявність JWT-токена.
3. Якщо токен валідний – виконується перевірка ролі користувача.
4. Модуль RBAC визначає, чи має користувач право доступу до запитуваного ресурсу.
5. У разі успіху ресурс повертається клієнту.
6. Подія записується в журнал доступу.

Компонентна структура серверної частини

Серверна частина системи логічно поділяється на такі модулі:

- AuthController – обробка логіну та видача токена;
- UserService – управління користувачами;
- RoleService – управління ролями;

- `PermissionService` – управління дозволами;
- `AccessMiddleware` – перевірка прав доступу;
- `AuditService` – логування дій.

Таке розділення відповідає принципам:

- SOLID;
- розділення відповідальності (Separation of Concerns);
- модульності.

Нефункціональні вимоги до архітектури

При проєктуванні враховуються:

- Безпека – захищені з'єднання (HTTPS), хешування паролів.
- Масштабованість – можливість розгортання на декількох серверах.
- Надійність – обробка помилок та журналювання.
- Продуктивність – мінімізація кількості запитів до БД.
- Розширюваність – можливість додавання 2FA або OAuth у майбутньому.

2.2 Модель ролей та прав доступу (RBAC)

Для реалізації системи контролю доступу в даній роботі використовується рольова модель керування доступом (Role-Based Access Control, RBAC). Вибір саме цієї моделі обумовлений її широким застосуванням у корпоративних інформаційних системах, простотою адміністрування та високою масштабованістю.

Загальна концепція RBAC

У моделі RBAC доступ до ресурсів надається не безпосередньо користувачам, а через ролі. Кожен користувач може мати одну або декілька ролей, а кожна роль – набір дозволів (permissions), які визначають перелік допустимих операцій над ресурсами.

Основні елементи моделі:

- User (Користувач) – суб'єкт доступу.
- Role (Роль) – логічна група повноважень.

- Permission (Дозвіл) – право виконання операції.
- Resource (Ресурс) – об’єкт, до якого здійснюється доступ.
- Session (Сесія) – активний контекст взаємодії користувача із системою.

Формальна модель

RBAC можна формалізувати у вигляді множин:

- U – множина користувачів
- R – множина ролей
- P – множина дозволів
- S – множина сесій

Визначаються відображення:

- $UA \subseteq U \times R$ – призначення ролей користувачам
- $PA \subseteq P \times R$ – призначення дозволів ролям

Таким чином, користувач отримує доступ до ресурсу лише тоді, коли існує роль, призначена йому, що містить відповідний дозвіл.

Ролі у розроблюваній системі

У рамках цієї роботи визначено базовий набір ролей:

1. Адміністратор (Administrator)

- створення та редагування користувачів;
- управління ролями;
- призначення дозволів;
- перегляд журналів доступу;
- повний доступ до ресурсів.

2. Менеджер (Manager)

- перегляд та редагування певних ресурсів;
- доступ до звітності;
- обмежене управління підлеглими користувачами.

3. Співробітник (Employee)

- перегляд власних даних;
- доступ лише до дозволених корпоративних ресурсів;

- відсутність адміністративних повноважень.

Типи дозволів

Дозволи визначають конкретну операцію над ресурсом. В системі використовуються такі типи:

- READ – перегляд;
- CREATE – створення;
- UPDATE – редагування;
- DELETE – видалення;
- MANAGE – повне управління.

Приклад дозволу:

READ:documents

UPDATE:employees

DELETE:reports

Структурна схема RBAC

- User ↔ Role (багато-до-багатьох)
- Role ↔ Permission (багато-до-багатьох)
- Permission ↔ Resource (багато-до-одного)

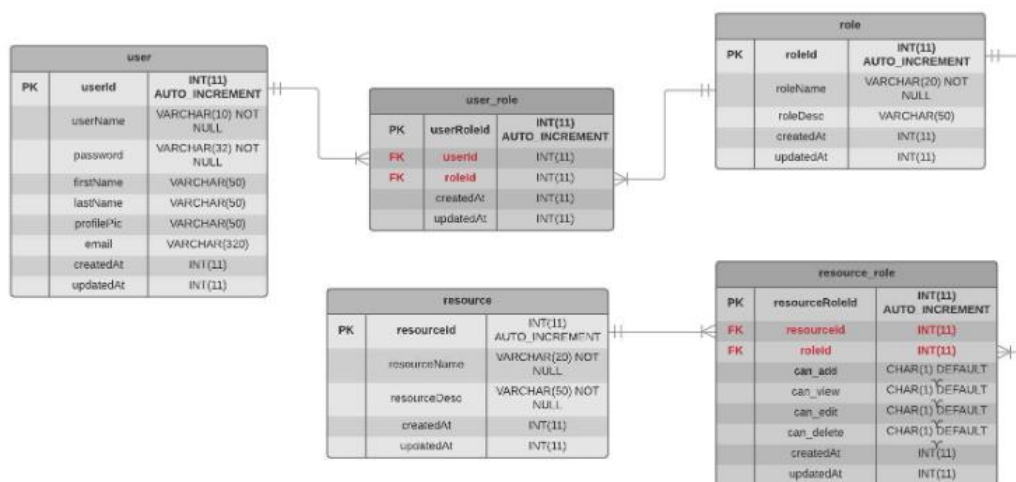


Рис. 2.2. Структурна схема RBAC

Алгоритм перевірки прав доступу

Процес авторизації в системі відбувається за таким алгоритмом:

1. Отримання ідентифікатора користувача з JWT-токена.

2. Отримання списку ролей користувача.
3. Отримання дозволів, пов'язаних з цими ролями.
4. Порівняння дозволів із запитуваною операцією.
5. У разі відповідності – надання доступу.
6. У разі відсутності дозволу – повернення HTTP 403 Forbidden.
7. Запис події у журнал доступу.

Переваги обраної моделі

- Централізоване управління правами.
- Мінімізація дублювання дозволів.
- Простота масштабування (додавання нових ролей).
- Гнучкість конфігурації.
- Відповідність принципу мінімальних привілеїв (Least Privilege).

Обмеження моделі

- Відсутність гнучкої логіки на основі контексту (час, IP, геолокація).
- Не враховує атрибутивні параметри (як в ABAC).
- Потребує чіткої структури ролей для уникнення надлишковості.

2.3 Проєктування бази даних системи

Ефективність системи контролю доступу значною мірою залежить від коректно спроектованої структури бази даних. Оскільки система передбачає зберігання структурованих даних із чіткими зв'язками між сутностями, доцільним є використання реляційної моделі даних.

У роботі передбачається використання СУБД типу PostgreSQL або MySQL, що підтримують транзакційність, зовнішні ключі та механізми індексації.

Основні принципи проєктування

При розробці структури БД враховано такі принципи:

- нормалізація до 3НФ;
- забезпечення цілісності даних (PRIMARY KEY, FOREIGN KEY);
- уникнення дублювання інформації;
- оптимізація запитів через індекси;

- підтримка масштабованості.

Основні сутності бази даних

Відповідно до моделі RBAC, база даних містить такі таблиці:

1. users
2. roles
3. permissions
4. user_roles
5. role_permissions
6. resources
7. access_logs

Опис таблиць

1. Таблиця users

Призначена для зберігання облікових записів користувачів.

Таблиця 2.1

Users

Поле	Тип	Опис
id	SERIAL / INT	Первинний ключ
username	VARCHAR(100)	Унікальний логін
email	VARCHAR(150)	Електронна пошта
password_hash	VARCHAR(255)	Хеш пароля
is_active	BOOLEAN	Статус активності

Обмеження:

- UNIQUE(username)
- UNIQUE(email)

2. Таблиця roles

Таблиця 2.2

Roles

Поле	Тип	Опис
id	SERIAL	Первинний ключ
name	VARCHAR(100)	Назва ролі
description	TEXT	Опис ролі

3. Таблиця permissions

Таблиця 2.3

Permissions

Поле	Тип	Опис
id	SERIAL	Первинний ключ
name	VARCHAR(100)	Назва дозволу
resource_id	INT	Посилання на ресурс
action	VARCHAR(50)	Тип операції

4. Таблиця resources

Таблиця 2.4

Resources

Поле	Тип	Опис
id	SERIAL	Первинний ключ
name	VARCHAR(100)	Назва ресурсу
description	TEXT	Опис

5. Таблиця user_roles

Проміжна таблиця (many-to-many).

Таблиця 2.5

User_roles

Поле	Тип
user_id	INT (FK → users.id)
role_id	INT (FK → roles.id)

Первинний ключ: (user_id, role_id)

6. Таблиця role_permissions

Таблиця 2.6

Role_permissions

Поле	Тип
role_id	INT (FK → roles.id)
permission_id	INT (FK → permissions.id)

Первинний ключ: (role_id, permission_id)

7. Таблиця access_logs

Призначена для аудиту дій користувачів.

Таблиця 2.7

Access_logs

Поле	Тип	Опис
id	SERIAL	Первинний ключ
user_id	INT	Користувач
resource_id	INT	Ресурс
action	VARCHAR(50)	Виконана дія
timestamp	TIMESTAMP	Час події
status	VARCHAR(20)	SUCCESS / DENIED
ip_address	VARCHAR(45)	IP-адреса

Забезпечення цілісності даних

Передбачено використання:

- FOREIGN KEY з ON DELETE CASCADE;
- CHECK-обмежень для допустимих значень action;
- індексів для полів username, email;
- індексу на access_logs.timestamp для швидкого аналізу подій.

2.4 Опис алгоритмів аутентифікації та авторизації

У даному підрозділі описуються алгоритмічні механізми, що забезпечують перевірку особи користувача (аутентифікацію) та визначення його прав доступу (авторизацію). У межах розроблюваної системи використовується token-based аутентифікація з застосуванням JWT (JSON Web Token) у поєднанні з моделлю RBAC.

2.4.1 Алгоритм аутентифікації користувача

Аутентифікація реалізується за класичною схемою:

1. Користувач вводить логін та пароль.
2. Дані передаються на сервер через HTTPS.
3. Сервер перевіряє наявність користувача в базі.

4. Виконується перевірка хешу пароля (bcrypt).
5. У разі успіху генерується JWT-токен.
6. Токен повертається клієнту.
7. Клієнт зберігає токен та передає його в заголовок Authorization при кожному запиті.

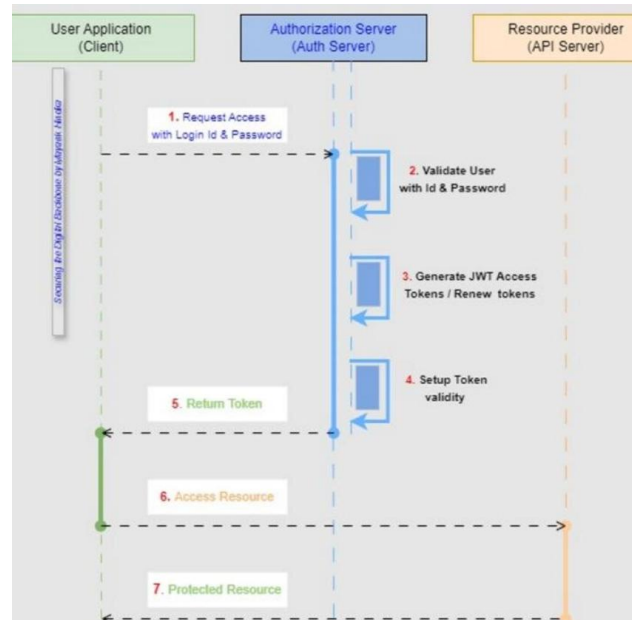


Рис. 2.3. Алгоритм аутентифікації користувача

2.4.2 Структура JWT-токена

JWT складається з трьох частин:

1. Header – тип токена та алгоритм підпису.
2. Payload – дані користувача (user_id, roles).
3. Signature – цифровий підпис.

Приклад payload:

```
{
  "user_id": 15,
  "roles": ["ADMIN"],
  "exp": 1710000000
}
```

Переваги використання JWT:

- відсутність серверної сесії;

- масштабованість;
- можливість передачі ролей у токени;
- швидка перевірка без звернення до БД.

2.4.3 Алгоритм авторизації (перевірка прав доступу)

Після аутентифікації кожен запит до захищеного ресурсу проходить перевірку авторизації.

Алгоритм:

1. Отримання токена із заголовка:

Authorization: Bearer <token>

2. Перевірка підпису токена.
3. Перевірка строку дії.
4. Отримання user_id та ролей.
5. Отримання дозволів для ролей.
6. Порівняння запитуваної операції з наявними дозволами.
7. Надання або відмова у доступі.

2.4.4 Middleware перевірки доступу

Авторизація реалізується через проміжний програмний шар (Middleware), який:

- перехоплює HTTP-запит;
- перевіряє токен;
- перевіряє роль;
- передає запит до контролера лише у разі успішної перевірки.

```
function authorize(request, required_permission):  
token = extract_token(request)  
  
if not validate(token):  
    return 401 Unauthorized  
  
user_roles = get_roles(token.user_id)  
permissions = get_permissions(user_roles)  
  
if required_permission in permissions:  
    return proceed()  
else:  
    return 403 Forbidden
```

2.4.5 Refresh-токени

Для підвищення безпеки використовується механізм короткоживучих access-токенів та довгоживучих refresh-токенів.

Access token:

- термін дії 15–30 хвилин.

Refresh token:

- зберігається окремо;
- використовується для генерації нового access-токена.

Алгоритм оновлення:

1. Access-токен закінчився.
2. Клієнт надсилає refresh-токен.
3. Сервер перевіряє його валідність.
4. Генерується новий access-токен.

2.4.6 Обробка помилок доступу

Система повинна повертати коректні HTTP-коди:

- 200 OK – успішний запит;
- 401 Unauthorized – невалідний токен;
- 403 Forbidden – відсутні права;
- 404 Not Found – ресурс відсутній.

2.4.7 Захист від типових атак

У рамках алгоритмів передбачено:

- хешування паролів (bcrypt);
- перевірка exp у токені;
- використання HTTPS;
- обмеження кількості спроб логіну (rate limiting);
- запис кожної спроби доступу у журнал.

2.5 Моделювання бізнес-процесів

Моделювання бізнес-процесів є важливим етапом проєктування системи контролю доступу. UML-діаграми дозволяють формалізувати функціональні вимоги, визначити взаємодію між користувачами та системою, а також описати внутрішню структуру програмних компонентів.

У межах даної роботи використано такі типи UML-діаграм:

- Use Case Diagram (діаграма варіантів використання);
- Class Diagram (діаграма класів);
- Activity Diagram (діаграма діяльності).

2.5.1 Діаграма варіантів використання (Use Case Diagram)

Use Case Diagram відображає взаємодію між акторами (ролями користувачів) та функціональними можливостями системи.

У розробленій системі визначено три основні актори:

- Administrator
- Manager
- Employee

Основні варіанти використання:

- Manage Users
- Assign Roles
- View Audit Logs
- Generate Reports
- Manage Team
- Access Resources
- Update Profile

Згенерована UML-діаграма варіантів використання демонструє, що:

- Адміністратор має повний доступ до управління користувачами та ролями.
- Менеджер може переглядати журнали та формувати звіти.
- Співробітник має обмежений доступ до ресурсів та власного профілю.

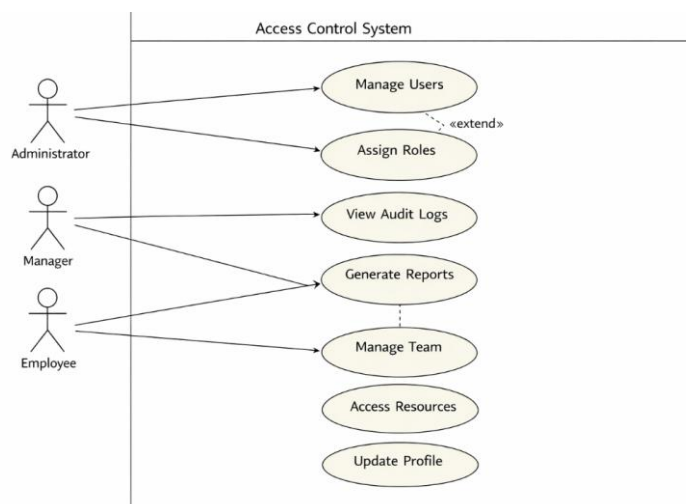


Рис. 2.4. Діаграма варіантів використання (Use Case Diagram)

Аналіз діаграми

З точки зору архітектури системи:

- кожен варіант використання відповідає окремому REST-endpoint;

- доступ до endpoint контролюється через RBAC-middleware;
- адміністративні функції ізольовані від звичайних користувачів.

Діаграма підтверджує відповідність системи принципу Separation of Concerns та принципу мінімальних привілеїв.

2.5.2 Діаграма класів (Class Diagram)

Діаграма класів описує статичну структуру системи та взаємозв'язки між основними сутностями.

Ключові класи:

User

- id
- username
- email
- passwordHash
- isActive

Role

- id
- name
- description

Permission

- id
- action
- resource

Resource

- id
- name
- description

AccessLog

- id

- userId
- resourceId
- action
- timestamp
- status

Зв'язки:

- User ↔ Role (many-to-many)
- Role ↔ Permission (many-to-many)
- Permission → Resource (many-to-one)
- AccessLog → User (many-to-one)

Діаграма класів відображає структуру бази даних та логіку серверної частини, що реалізує бізнес-правила.

2.5.3 Діаграма діяльності (Activity Diagram)

Activity Diagram використовується для моделювання процесу доступу до ресурсу.

Основні етапи:

1. Введення логіна та пароля.
2. Перевірка облікових даних.
3. Генерація JWT-токена.
4. Надсилання запиту до ресурсу.
5. Перевірка токена.
6. Перевірка ролі.
7. Надання або відмова у доступі.
8. Запис у журнал доступу.

Висновки до розділу 2

Другий розділ присвячено системному проєктуванню архітектури, алгоритмів та структур даних розроблюваної системи контролю доступу. Обґрунтовано вибір трирівневої клієнт-серверної архітектури (Presentation, Application, Data layers), яка забезпечує чітке розділення відповідальності між компонентами, підвищує масштабованість та ізолює безпосередню бізнес-логіку від рівня представлення даних.

Спроектовано формальну структуру рольової моделі із базовим набором сутностей (User, Role, Permission, Resource) та прав для базових ролей системи: адміністратора, менеджера та співробітника відповідно до принципу мінімальних привілеїв (Least Privilege). На основі реляційного підходу розроблено структуру бази даних, яка повністю відповідає вимогам третьої нормальної форми (3НФ), що гарантує цілісність інформації та швидку індексацію подій для системи журналювання дій. За допомогою інструментів UML-моделювання (діаграм варіантів використання, класів та діяльності) формалізовано логіку взаємодії компонентів та алгоритми token-based автентифікації й авторизації з використанням короткоживучих access-токенів та безпечних довгоживучих refresh-токенів.

РОЗДІЛ 3. ПРОГРАМНА РЕАЛІЗАЦІЯ

3.1 Обґрунтування вибору технологій розробки системи

Реалізація системи контролю доступу до корпоративних ресурсів потребує використання сучасних програмних технологій, що забезпечують масштабованість, безпечність та можливість інтеграції з іншими інформаційними системами підприємства.

При виборі технологічного стеку враховувалися такі критерії:

- підтримка клієнт-серверної архітектури;
- висока продуктивність обробки HTTP-запитів;
- можливість реалізації REST API;
- підтримка механізмів автентифікації JWT;
- зручність реалізації рольової моделі доступу RBAC;
- поширеність технології у корпоративній розробці.

Вибір серверної платформи

Для реалізації серверної частини системи обрано середовище Node.js із використанням фреймворку Express.js.

Node.js дозволяє виконувати JavaScript на стороні сервера та використовує неблокуючу модель введення-виведення, що забезпечує ефективну обробку великої кількості одночасних запитів.

Основні переваги Node.js:

- висока продуктивність;
- асинхронна обробка запитів;
- велика екосистема бібліотек;
- зручна реалізація REST-сервісів;
- підтримка мікросервісної архітектури.

Фреймворк Express забезпечує:

- маршрутизацію HTTP-запитів;
- підтримку middleware;

- просту інтеграцію механізмів авторизації.

Вибір системи керування базами даних

У якості системи керування базами даних використано PostgreSQL.

PostgreSQL є об'єктно-реляційною СУБД корпоративного рівня та забезпечує:

- підтримку транзакцій (ACID);
- механізми зовнішніх ключів;
- індексацію;
- високу надійність зберігання даних;
- масштабованість.

Реляційна модель добре відповідає структурі RBAC, що містить зв'язки типу many-to-many між користувачами, ролями та дозволами.

Технології забезпечення безпеки

Для реалізації механізмів безпеки використано:

JWT (JSON Web Token) – для token-based аутентифікації користувачів.

JWT дозволяє:

- уникнути серверних сесій;
- масштабувати систему;
- передавати інформацію про ролі користувача.

bcrypt – для хешування паролів користувачів.

Алгоритм bcrypt забезпечує:

- захист від brute-force атак;
- використання salt;
- неможливість відновлення початкового пароля.

Технології клієнтської частини

Клієнтська частина реалізується як веб-застосунок із використанням бібліотеки React.

React забезпечує:

- компонентний підхід;
- швидке оновлення інтерфейсу;
- зручну роботу з REST API;

- підтримку сучасних SPA-застосунків.

Взаємодія компонентів системи

Обмін даними між клієнтською та серверною частинами здійснюється через REST API з використанням формату JSON.

Загальна схема взаємодії:

1. Користувач виконує авторизацію.
2. Сервер генерує JWT-токен.
3. Клієнт зберігає токен.
4. Подальші запити виконуються з передачею токена.
5. Middleware перевіряє права доступу.

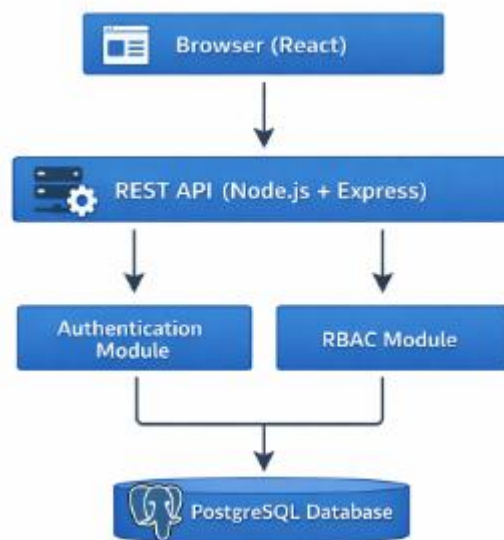


Рис. 3.1. Схема взаємодії

3.2 Реалізація серверної частини системи контролю доступу

Серверна частина системи контролю доступу реалізує основну бізнес-логіку застосунку та забезпечує виконання процесів автентифікації, авторизації, управління користувачами, ролями та контролю доступу до корпоративних ресурсів.

Розроблена система побудована відповідно до клієнт-серверної архітектури та реалізована у вигляді REST API із використанням середовища Node.js та фреймворку Express.js.

Структура серверного проєкту

Для забезпечення масштабованості та підтримуваності програмного коду використано модульну структуру проєкту.

Структура каталогів серверної частини має вигляд:

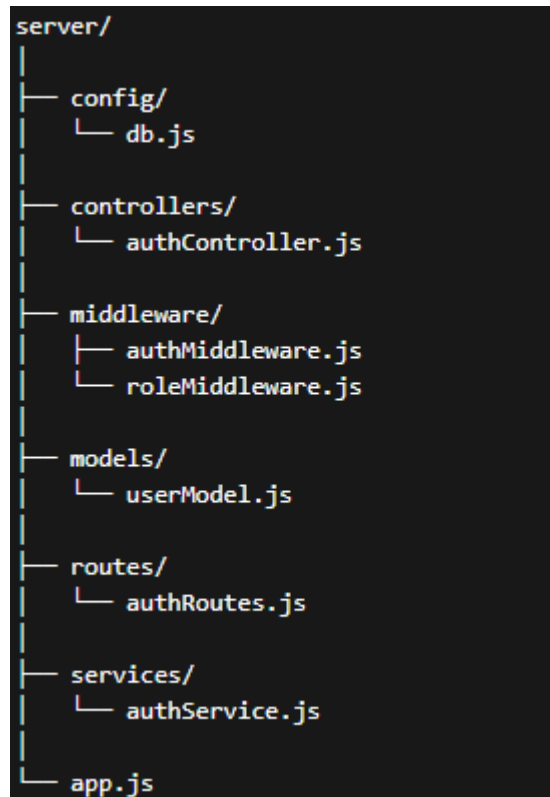


Рис. 3.2. Структура каталогів серверної частини

Підключення до бази даних

Для взаємодії із СУБД PostgreSQL використовується бібліотека pg.

Файл db.js

Лістинг 3.1

```
const { Pool } = require("pg");
```

```
const pool = new Pool({  
  user: "postgres",  
  host: "localhost",  
  database: "access_system",  
  password: "password",  
  port: 5432,  
});
```

```
module.exports = pool;
```

Даний модуль забезпечує централізоване підключення до бази даних та повторне використання з'єднання.

Реалізація автентифікації користувача

Процес автентифікації включає:

1. перевірку логіна;
2. перевірку пароля;
3. генерацію JWT-токена.

Контролер автентифікації

Лістинг 3.2

```
const bcrypt = require("bcrypt");
const jwt = require("jsonwebtoken");
const pool = require("../config/db");

exports.login = async (req, res) => {

  const { username, password } = req.body;

  const user = await pool.query(
    "SELECT * FROM users WHERE username=$1",
    [username]
  );

  if (!user.rows.length)
    return res.status(401).json({ message: "User not found" });

  const validPassword = await bcrypt.compare(
    password,
    user.rows[0].password_hash
  );

  if (!validPassword)
    return res.status(401).json({ message: "Invalid password" });

  const token = jwt.sign(
    {
      user_id: user.rows[0].id,
      role: user.rows[0].role
```

```

    },
    "SECRET_KEY",
    { expiresIn: "30m" }
  );

  res.json({ token });
};

```

У результаті успішної автентифікації користувач отримує JWT-токен, який використовується при подальших запитах.

Middleware перевірки токена

Для захисту REST-endpoint реалізовано проміжний програмний шар – middleware.

Лістинг 3.3

```

const jwt = require("jsonwebtoken");

module.exports = function (req, res, next) {

  const authHeader = req.headers.authorization;

  if (!authHeader)
    return res.status(401).json({ message: "No token" });

  const token = authHeader.split(" ")[1];

  try {
    const decoded = jwt.verify(token, "SECRET_KEY");
    req.user = decoded;
    next();
  } catch (err) {
    return res.status(401).json({ message: "Invalid token" });
  }
}

```

```
}  
};
```

Middleware перехоплює кожен HTTP-запит та перевіряє валідність токена.

Реалізація RBAC-перевірки доступу

Лістинг 3.4

```
module.exports = function (requiredRole) {  
  
  return (req, res, next) => {  
  
    if (req.user.role !== requiredRole) {  
      return res.status(403).json({  
        message: "Access denied"  
      });  
    }  
  
    next();  
  };  
};
```

Таким чином забезпечується доступ до ресурсів відповідно до ролі користувача.

Приклад захищеного маршруту

Лістинг 3.5

```
const router = require("express").Router();  
const auth = require("../middleware/authMiddleware");  
const role = require("../middleware/roleMiddleware");  
  
router.get(  
  "/admin",  
  auth,  
  role("ADMIN"),  
  (req, res) => {  
    res.json({ message: "Admin resource" });  
  }  
);
```

```
module.exports = router;
```

Доступ до даного endpoint отримує лише користувач із роллю адміністратора.

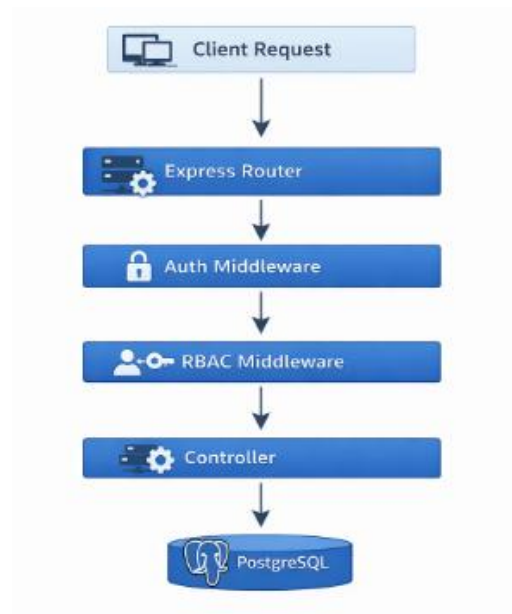


Рис. 3.3. Приклад захищеного маршруту

Реалізація серверної логіки управління користувачами та ролями

Після реалізації механізму автентифікації наступним етапом є створення функціоналу управління користувачами та їх правами доступу. Даний модуль забезпечує адміністрування системи та реалізацію рольової моделі доступу RBAC.

Реалізація створення користувача

Адміністратор системи має можливість створювати нових користувачів із визначеною роллю доступу.

Перед збереженням у базі даних пароль користувача проходить процедуру хешування.

Контролер створення користувача

Лістинг 3.6

```
const bcrypt = require("bcrypt");  
const pool = require("../config/db");
```

```

exports.createUser = async (req, res) => {

  const { username, email, password, role } = req.body;

  const hash = await bcrypt.hash(password, 10);

  const newUser = await pool.query(
    `INSERT INTO users
      (username, email, password_hash, role)
      VALUES ($1,$2,$3,$4)
      RETURNING *`,
    [username, email, hash, role]
  );

  res.json(newUser.rows[0]);
};

```

Хешування паролів дозволяє виключити зберігання конфіденційної інформації у відкритому вигляді.

Отримання списку користувачів

Лістинг 3.7

```

exports.getUsers = async (req, res) => {

  const users = await pool.query(
    "SELECT id, username, email, role FROM users"
  );

  res.json(users.rows);
};

```

Даний endpoint використовується адміністративною панеллю системи.

Призначення ролі користувачу

RBAC передбачає можливість зміни ролі без модифікації бізнес-логіки системи.

Лістинг 3.8

```
exports.updateRole = async (req, res) => {  
  
  const { id } = req.params;  
  const { role } = req.body;  
  
  await pool.query(  
    "UPDATE users SET role=$1 WHERE id=$2",  
    [role, id]  
  );  
  
  res.json({ message: "Role updated" });  
};
```

Централізоване управління ролями дозволяє швидко змінювати політику доступу підприємства.

Реалізація журналювання доступу (Audit Log)

Однією з ключових вимог корпоративних систем є аудит дій користувачів. Для цього реалізовано модуль логуювання.

Лістинг 3.9

```
exports.logAccess = async (  
  userId,  
  resource,  
  action,  
  status,  
  ip  
) => {  
  
  await pool.query(  
    `INSERT INTO access_logs  
    (user_id, resource_id, action, status, ip_address)  
    VALUES ($1,$2,$3,$4,$5)`,  
    [userId, resource, action, status, ip]  
  );  
}
```

);
};

Журнал доступу дозволяє:

- відстежувати дії користувачів;
- виявляти спроби несанкціонованого доступу;
- проводити аналіз інцидентів безпеки.

Таблиця 3.1

Приклад REST API системи

Метод	Endpoint	Опис
POST	/auth/login	Авторизація
POST	/users	Створення користувача
GET	/users	Список користувачів
PUT	/users/:id/role	Зміна ролі
GET	/admin	Захищений ресурс

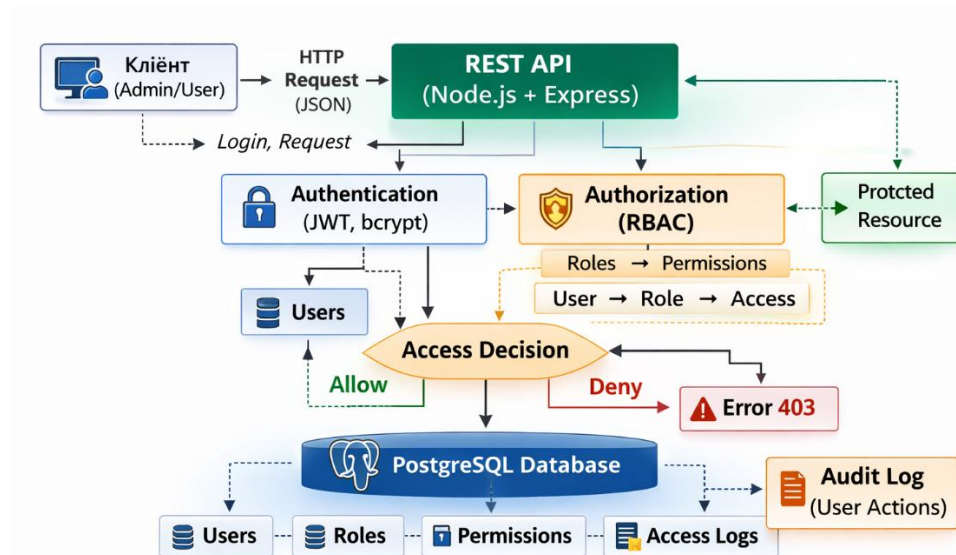


Рис. 3.4. Взаємодія REST API із системою контролю доступу (RBAC)

На рисунку 3.4. представлено загальну схему взаємодії клієнтської частини системи із серверною частиною контролю доступу до корпоративних ресурсів.

Користувач системи (адміністратор або звичайний співробітник) надсилає HTTP-запит до REST API, реалізованого із використанням середовища Node.js та фреймворку Express. Запит передається у форматі JSON та може містити дані автентифікації або запит доступу до захищеного ресурсу.

На першому етапі відбувається автентифікація користувача, що включає:

- перевірку логіну та пароля;

- хешування пароля за допомогою bcrypt;
- формування JWT-токена доступу.

Після підтвердження особи система переходить до етапу авторизації, реалізованого за моделлю RBAC (Role-Based Access Control). Користувачу призначається роль, яка визначає набір дозволених операцій.

Модуль прийняття рішення щодо доступу аналізує:

- роль користувача;
- дозволи;
- тип ресурсу;
- запитувану дію.

У випадку відповідності прав доступ надається, інакше система повертає помилку 403 (Forbidden). Усі дії користувачів фіксуються в журналі аудиту, що зберігається у базі даних PostgreSQL.

3.4 Реалізація механізму авторизації користувачів

Одним із ключових компонентів системи контролю доступу є механізм авторизації, який визначає можливість виконання користувачем певних операцій у корпоративній інформаційній системі.

У розробленій системі використано модель RBAC, що дозволяє централізовано керувати правами доступу через ролі.

Принцип роботи авторизації

Алгоритм перевірки доступу складається з таких етапів:

1. Отримання JWT-токена із запиту.
2. Перевірка валідності токена.
3. Визначення користувача.
4. Отримання ролі користувача.
5. Перевірка дозволів ролі.
6. Надання або заборона доступу.

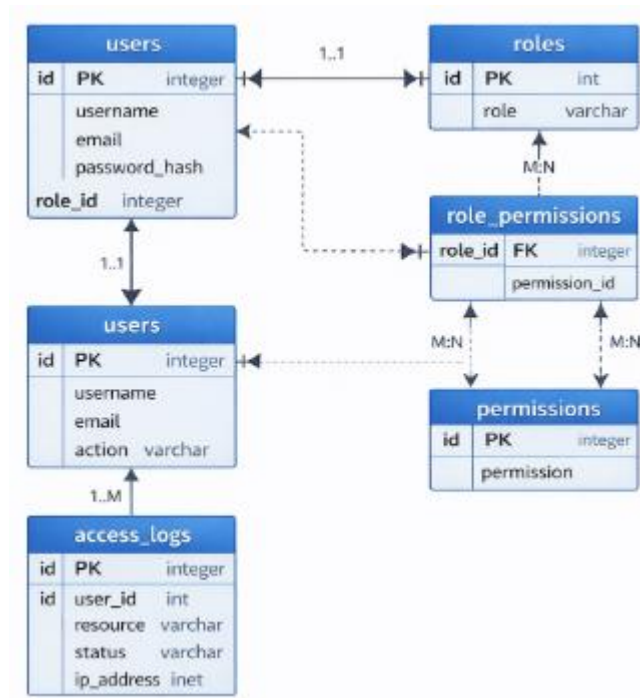


Рис. 3.5. Структура бази даних системи контролю доступу

На рисунку 3.5. представлено структуру реляційної бази даних системи контролю доступу, розробленої відповідно до моделі RBAC (Role-Based Access Control).

База даних побудована з урахуванням принципів нормалізації та забезпечення цілісності даних. Основними сутностями системи є таблиці users, roles, permissions, role_permissions та access_logs.

Таблиця users

Таблиця users зберігає облікові записи користувачів системи. Вона містить такі основні поля:

- id – первинний ключ (Primary Key);
- username – унікальний логін користувача;
- email – електронна адреса;
- password_hash – хешований пароль;
- role_id – зовнішній ключ, що визначає роль користувача.

Зв'язок між таблицями users та roles є типу 1:N, оскільки одна роль може бути призначена багатьом користувачам.

Таблиця roles

Таблиця roles містить перелік ролей системи (наприклад, Administrator, Manager, User). Основні поля:

- id – первинний ключ;
- role – назва ролі.

Кожна роль визначає набір дозволів, що реалізується через проміжну таблицю.

Таблиця permissions

Таблиця permissions зберігає дозволені дії в системі. Вона містить:

- id – первинний ключ;
- permission – назва дозволу (READ, CREATE, UPDATE, DELETE тощо).

Таблиця role_permissions

Оскільки між ролями та дозволами існує зв'язок типу багато-до-багатьох (M:N), використовується проміжна таблиця role_permissions, яка містить:

- role_id – зовнішній ключ на таблицю roles;
- permission_id – зовнішній ключ на таблицю permissions.

Даний механізм дозволяє гнучко конфігурувати права доступу без зміни структури системи.

Таблиця access_logs

Таблиця access_logs реалізує механізм аудиту дій користувачів. Вона містить:

- id – первинний ключ;
- user_id – посилання на користувача;
- resource – назва ресурсу;
- status – результат доступу (Allowed або Denied);
- ip_address – IP-адреса клієнта.

Зв'язок між таблицями users та access_logs є типу 1:M, оскільки один користувач може здійснити багато дій.

Загальна характеристика моделі

Запропонована структура бази даних:

- відповідає принципам третьої нормальної форми (3НФ);

- забезпечує цілісність через використання первинних та зовнішніх ключів;
- підтримує масштабування системи;
- дозволяє централізовано керувати правами доступу;
- забезпечує повний аудит дій користувачів.

Розроблена модель є оптимальною для корпоративної інформаційної системи середнього масштабу та відповідає сучасним вимогам інформаційної безпеки.

3.6 Реалізація клієнтської частини системи

Клієнтська частина системи контролю доступу реалізована у вигляді односторінкового веб-застосунку (Single Page Application, SPA) із використанням бібліотеки React.

Використання React дозволяє реалізувати компонентний підхід до розробки інтерфейсу, забезпечити швидке оновлення даних без перезавантаження сторінки та зручно інтегрувати REST API серверної частини.

The screenshot displays a web application interface for user management. At the top, a header bar contains the title 'Панель користувача' and a 'Вийти' button. The main content area is divided into several sections. On the left, there is a sidebar with a dark blue background. The first section, 'Доступні ресурси', lists four resources: 'Employee Resource', 'Manager Resource', 'Admin Resource', and 'Усі ресурси'. Below this is a section titled 'Адміністративні функції' which includes a 'Завантажити користувачів' button and a large dark blue rectangular area. To the right of the sidebar, there is a section for 'Створити користувача' with input fields for 'Логін', 'Email', 'Пароль', and a dropdown menu for 'EMPLOYEE', followed by a 'Створити' button. Below this is a 'Змінити роль користувача' section with an input field for 'ID користувача', a dropdown menu for 'EMPLOYEE', and an 'Оновити роль' button. At the bottom right, there is a 'Завантажити audit logs' button and another large dark blue rectangular area.

Рис. 3.6. Клієнтська частина

Архітектура клієнтського застосунку

Клієнтська частина складається з таких основних компонентів:

- LoginPage – сторінка авторизації користувача;

- Dashboard – головна сторінка після входу;
- AdminPanel – управління користувачами та ролями;
- ProtectedRoute – механізм захисту маршрутів;
- API Service – модуль взаємодії з сервером.

JWT-токен зберігається у localStorage браузера та передається в заголовок:

Authorization: Bearer <token>

3.6.1 Інтерфейс адміністративної панелі

Реалізація сторінки авторизації

Лістинг 3.10

```
import React, { useState } from "react";
import axios from "axios";

function LoginPage() {

  const [username, setUsername] = useState("");
  const [password, setPassword] = useState("");

  const handleLogin = async () => {
    const response = await axios.post(
      "http://localhost:5000/auth/login",
      { username, password }
    );

    localStorage.setItem("token", response.data.token);
    window.location.href = "/dashboard";
  };

  return (
    <div>
      <h2>Авторизація</h2>
      <input
        type="text"
        placeholder="Логін"
        onChange={(e) => setUsername(e.target.value)}
      />
      <input
        type="password"
        placeholder="Пароль"
      />
    </div>
  );
}
```

```

    onChange={e => setPassword(e.target.value)}
  />
  <button onClick={handleLogin}>Увійти</button>
</div>
);
}

export default LoginPage;

```

3.6.2 Захист клієнтських маршрутів

Для запобігання несанкціонованому доступу до сторінок застосунку реалізовано компонент ProtectedRoute.

Лістинг 3.11

```

import { Navigate } from "react-router-dom";

function ProtectedRoute({ children }) {

  const token = localStorage.getItem("token");

  if (!token) {
    return <Navigate to="/" />;
  }

  return children;
}

export default ProtectedRoute;

```

Таким чином забезпечується контроль доступу до інтерфейсу навіть на клієнтському рівні.

3.6.3 Взаємодія з REST API

Усі запити до серверної частини виконуються через Axios із автоматичним додаванням токена:

Лістинг 3.12

```
import axios from "axios";

const api = axios.create({
  baseURL: "http://localhost:5000"
});

api.interceptors.request.use((config) => {
  const token = localStorage.getItem("token");

  if (token) {
    config.headers.Authorization = `Bearer ${token}`;
  }

  return config;
});

export default api;
```

3.7 Забезпечення безпеки системи контролю доступу

Оскільки розроблена система призначена для контролю доступу до корпоративних ресурсів, питання забезпечення інформаційної безпеки є одним із ключових аспектів її реалізації. У даному підрозділі розглянуто основні механізми захисту, впроваджені в системі.

3.7.1 Захист паролів користувачів

Паролі користувачів не зберігаються у відкритому вигляді. Перед записом у базу даних вони проходять процедуру хешування з використанням алгоритму bcrypt.

Алгоритм bcrypt забезпечує:

- використання сольового значення (salt);
- багаторазове ітеративне хешування;
- стійкість до атак типу brute-force;
- неможливість відновлення початкового пароля.

Завдяки цьому навіть у разі компрометації бази даних зломисник не зможе отримати реальні паролі користувачів.

3.7.2 Захист від SQL Injection

Для взаємодії з базою даних використовуються параметризовані SQL-запити:

```
pool.query(  
  "SELECT * FROM users WHERE username=$1",  
  [username]  
);
```

Рис. 3.7. Взаємодії з базою даних

3.7.3 Захист від XSS та CSRF атак

Для запобігання XSS-атакам:

- всі дані, що відображаються в інтерфейсі, проходять санітизацію;
- React автоматично екранує HTML-вміст;
- заборонено використання небезпечних вставок типу dangerouslySetInnerHTML.

Для мінімізації ризику CSRF:

- використовується token-based автентифікація (JWT);
- сервер не використовує cookie-сесії;
- перевіряється наявність Authorization-заголовка.

3.7.4 Захист від brute-force атак

Для запобігання багаторазовим спробам підбору пароля реалізовано обмеження кількості запитів (rate limiting), логування невдалих спроб входу, можливість блокування облікового запису після визначеної кількості помилок.

Механізм rate limiting може бути реалізований із використанням middleware:

```
const rateLimit = require("express-rate-limit");

const limiter = rateLimit({
  windowMs: 15 * 60 * 1000,
  max: 100
});

app.use(limiter);
```

Рис. 3.8. Механізм rate limiting

3.7.5 Використання HTTPS

Передбачено використання протоколу HTTPS для:

- шифрування переданих даних;
- захисту токенів доступу;
- запобігання перехопленню мережевого трафіку.

SSL/TLS забезпечує цілісність та конфіденційність переданих даних.

3.7.6 Принцип мінімальних привілеїв

У системі реалізовано принцип least privilege, згідно з яким:

- кожен користувач отримує лише ті права, які необхідні для виконання службових обов'язків;
- ролі чітко структуровані;
- адміністративні функції доступні лише ролі ADMIN.

Це мінімізує потенційні наслідки компрометації облікового запису.

3.7.7 Механізм оновлення токенів (Refresh Token)

Для підвищення безпеки передбачено обмежений строк дії JWT-токена (30 хвилин). Після завершення строку користувач повинен повторно автентифікуватися або отримати новий токен через механізм refresh token.

Такий підхід:

- зменшує ризик використання викраденого токена;
- підвищує рівень захисту сесії;
- дозволяє контролювати активність користувачів.

Висновок до розділу 3

У третьому розділі виконано безпосередню програмну реалізацію спроектованої системи контролю доступу до корпоративних ресурсів. Створено працездатний модульний серверний додаток на базі платформи Node.js та фреймворку Express.js із використанням пулу з'єднань СУБД PostgreSQL. Реалізовано проміжні програмні шари (middleware) для надійної верифікації цифрових підписів JWT-токенів та динамічної перевірки прав доступу на рівні REST API маршрутів (endpoints).

Клієнтську частину розроблено у вигляді односторінкового веб-застосунку (SPA) на базі бібліотеки React із впровадженням захищених маршрутів (ProtectedRoute) та механізму автоматичного екранування вмісту для захисту від XSS-атак. Інтегровано комплексні безпекові рішення на рівні коду: криптографічне хешування паролів за допомогою ітеративного алгоритму bcrypt із використанням солі (salt), token-based захист від CSRF-атак, обов'язкове шифрування трафіку через HTTPS, а також обмеження частоти запитів (Rate Limiting) для ефективної протидії автоматизованим brute-force атакам.

РОЗДІЛ 4. ТЕСТУВАННЯ СИСТЕМИ

4.1 Мета та задачі тестування

Тестування є обов'язковим етапом розробки програмного забезпечення, який дозволяє підтвердити коректність роботи системи контролю доступу, її відповідність функціональним вимогам та стійкість до типових помилок і збоїв.

Оскільки розроблена система забезпечує контроль доступу до корпоративних ресурсів, особливу увагу під час тестування приділено перевірці механізмів автентифікації, авторизації (RBAC) та журналювання дій користувачів.

Мета тестування

Метою тестування є перевірка працездатності розробленої системи контролю доступу, а також виявлення можливих дефектів у роботі серверної та клієнтської частин.

Основні задачі тестування

Для досягнення поставленої мети виконуються такі задачі:

1. Перевірити коректність автентифікації користувачів (логін/пароль, видача JWT).
2. Перевірити правильність авторизації відповідно до ролей користувачів (RBAC).
3. Перевірити захист REST API від несанкціонованих запитів.
4. Перевірити роботу адміністративної панелі управління користувачами та ролями.
5. Перевірити коректність журналювання подій у таблиці `access_logs`.
6. Оцінити стабільність роботи системи при типовому навантаженні.
7. Проаналізувати результати тестування та сформулювати висновки щодо якості розробленої системи.

4.2 Види тестування системи

Для забезпечення якості розробленої системи контролю доступу було застосовано комплексний підхід до тестування, який включає модульне, інтеграційне та системне тестування.

Використання різних рівнів тестування дозволяє перевірити як окремі компоненти системи, так і їх взаємодію в реальному середовищі експлуатації.

4.2.1 Модульне тестування

Модульне тестування (Unit Testing) передбачає перевірку окремих функціональних модулів системи незалежно один від одного.

У рамках розробленої системи було протестовано такі модулі:

- функцію автентифікації користувача;
- перевірку JWT-токена;
- middleware авторизації;
- функції створення користувача;
- механізм журналювання подій.

Приклад тестування перевірки токена

Лістинг 3.12

```
const jwt = require(«jsonwebtoken»);
const authMiddleware = require(«../middleware/authMiddleware»);

test(«Token validation», () => {
  const token = jwt.sign(
    { user_id: 1 },
    «SECRET_KEY»
  );

  const decoded = jwt.verify(
    token,
    «SECRET_KEY»
  );
```

```
expect(decoded.user_id).toBe(1);  
});
```

Модульне тестування дозволило виявити та усунути помилки логіки до інтеграції компонентів у загальну систему.

4.2.2 Інтеграційне тестування

Інтеграційне тестування передбачає перевірку взаємодії між різними модулями системи.

У межах даної роботи було перевірено:

- взаємодію клієнтської частини з REST API;
- передачу JWT-токена у заголовках запитів;
- коректність перевірки ролей;
- запис подій у таблицю access_logs.

Приклад інтеграційного сценарію

1. Користувач виконує авторизацію.
2. Сервер генерує JWT-токен.
3. Користувач надсилає запит до захищеного ресурсу.
4. Middleware перевіряє токен.
5. RBAC перевіряє права доступу.
6. Записується подія у журнал.

4.2.3 Системне тестування

Системне тестування проводиться для перевірки повністю інтегрованої системи в умовах, максимально наближених до реального використання.

Було перевірено:

- роботу адміністративної панелі;
- створення та зміну ролей;
- доступ користувачів до ресурсів;
- обробку помилок 401 та 403;

- стабільність роботи при багаторазових запитах.

Таблиця 4.1

Порівняння рівнів тестування

Вид тестування	Об'єкт перевірки	Мета
Модульне	Окремі функції	Виявлення локальних помилок
Інтеграційне	Взаємодія модулів	Перевірка коректності інтеграції
Системне	Повна система	Перевірка відповідності вимогам



Рис. 4.1. Рівні тестування системи

4.3 Розробка тестових сценаріїв

Для перевірки працездатності системи контролю доступу було розроблено набір тестових сценаріїв (test cases), які охоплюють як позитивні, так і негативні варіанти використання.

Тестові сценарії дозволяють формалізувати процес перевірки та забезпечити відтворюваність результатів тестування.

4.3.1 Тестування автентифікації користувача

Таблиця 4.2

Тестування механізму логіну

№	Опис сценарію	Вхідні дані	Очікуваний результат
1	Коректний логін	username=admin, password=123456	Видається JWT-токен
2	Невірний пароль	username=admin, password=wrong	401 Unauthorized
3	Неіснуючий користувач	username=test	401 Unauthorized
4	Порожні поля	""	Помилка валідації

Результати тестування підтвердили коректність роботи механізму автентифікації.

4.3.2 Тестування авторизації (RBAC)

Таблиця 4.3

Перевірка прав доступу

№	Роль	Дія	Очікуваний результат	Результат
1	ADMIN	Доступ до /admin	200 OK	Успішно
2	MANAGER	Доступ до /admin	403 Forbidden	Успішно
3	USER	Перегляд власного профілю	200 OK	Успішно
4	USER	Зміна ролі іншого користувача	403 Forbidden	Успішно

Дані результати підтверджують правильність реалізації рольової моделі доступу.

4.3.3 Тестування журналювання подій

Таблиця 4.4

Перевірка запису у access_logs

№	Дія	Статус	Запис у БД
1	Успішний логін	SUCCESS	Є запис
2	Відмова у доступі	DENIED	Є запис
3	Спроба доступу без токена	DENIED	Є запис

Журнал подій коректно фіксує всі запити до системи, що відповідає вимогам інформаційної безпеки.

4.4 Тестування механізмів аутентифікації та авторизації

У даному підрозділі проведено детальний аналіз роботи механізмів автентифікації (JWT) та авторизації (RBAC) з точки зору безпеки та коректності обробки запитів.

4.4.1 Перевірка коректності JWT-токена

Під час тестування було перевірено:

- формування токена після успішного логіну;
- правильність структури payload;
- наявність строку дії (exp);
- перевірку цифрового підпису;
- реакцію системи на прострочений токен.

Таблиця 4.5

Результати тестування JWT

№	Сценарій	Очікуваний результат	Фактичний результат
1	Валідний токен	200 OK	Відповідає
2	Прострочений токен	401 Unauthorized	Відповідає
3	Підроблений токен	401 Unauthorized	Відповідає
4	Відсутній токен	401 Unauthorized	Відповідає

Система коректно обробляє всі випадки некоректної автентифікації.

4.4.2 Перевірка HTTP-кодів відповіді

Важливим аспектом тестування є перевірка коректності статус-кодів HTTP.

Таблиця 4.6

Перевірка HTTP-кодів

Сценарій	Очікуваний код
Успішний доступ	200 OK
Невірний пароль	401 Unauthorized

Недостатні права	403 Forbidden
Неіснуючий ресурс	404 Not Found

Під час тестування підтверджено відповідність стандартам REST API.

4.4.3 Edge-case тестування

Було проведено перевірку граничних випадків:

- багаторазові спроби логіну (імітація brute-force);
- одночасні запити від різних ролей;
- спроба доступу до endpoint без авторизації;
- некоректний формат JWT у заголовку.

Усі запити без валідного токена були заблоковані.

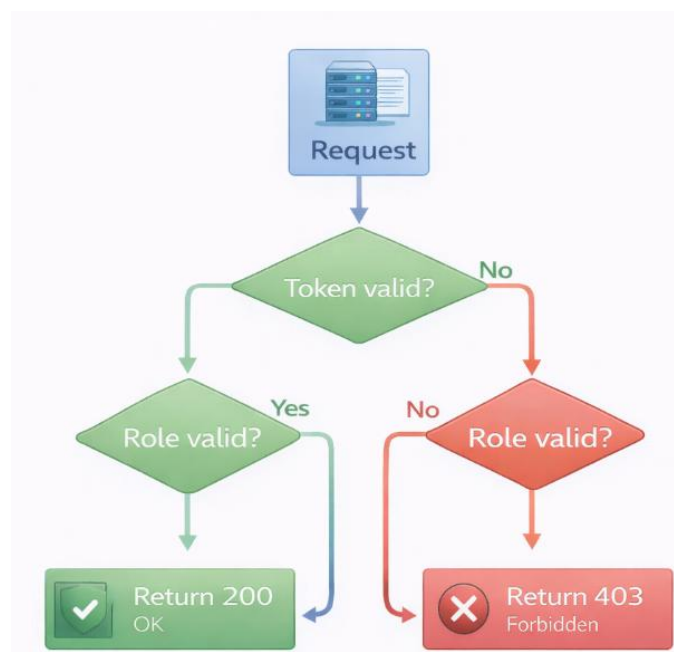


Рис. 4.2. Обробка помилок автентифікації та авторизації

Короткий опис:

На рисунку 4.4 зображено алгоритм обробки помилок під час перевірки запитів користувача. Після отримання HTTP-запиту система спочатку перевіряє валідність JWT-токена. У разі його відсутності або некоректності повертається помилка 401 Unauthorized. Якщо токен є дійсним, виконується перевірка ролі користувача відповідно до моделі RBAC. При недостатніх правах доступу

система повертає 403 Forbidden. У випадку успішної перевірки доступ надається з кодом 200 OK.

4.5 Оцінка продуктивності та надійності системи

Оцінка продуктивності та надійності системи контролю доступу є важливим етапом тестування, оскільки корпоративні інформаційні системи повинні забезпечувати стабільну роботу при одночасному обслуговуванні великої кількості користувачів.

Метою даного етапу є:

- визначення часу відповіді сервера;
- оцінка поведінки системи при збільшенні кількості запитів;
- перевірка стабільності роботи механізмів автентифікації та авторизації;
- аналіз навантаження на базу даних.

4.5.1 Методика проведення тестування

Тестування проводилось шляхом імітації одночасних HTTP-запитів до серверної частини системи.

Перевірялися такі сценарії:

1. Одночасний логін 10 користувачів.
2. Одночасний логін 50 користувачів.
3. Одночасний логін 100 користувачів.
4. Запити до захищеного ресурсу з валідним JWT.
5. Запити до захищеного ресурсу без токена.

Фіксувалися:

- середній час відповіді (мс);
- максимальний час відповіді (мс);
- кількість помилок;
- стабільність з'єднання з БД.

4.5.2 Результати тестування продуктивності

Таблиця 4.7

Результати тестування навантаження

Кількість одночасних запитів	Середній час відповіді (мс)	Максимальний час (мс)	Кількість помилок
10	45	70	0
50	82	130	0
100	145	220	0

Аналіз результатів

Отримані результати свідчать, що:

- система демонструє стабільну роботу при навантаженні до 100 одночасних запитів;
- час відповіді зростає лінійно зі збільшенням кількості запитів;
- помилки під час тестування відсутні;
- база даних PostgreSQL коректно обробляє паралельні звернення.

Архітектура Node.js з неблокуючою моделлю введення-виведення показала достатній рівень продуктивності для корпоративного середовища середнього масштабу.

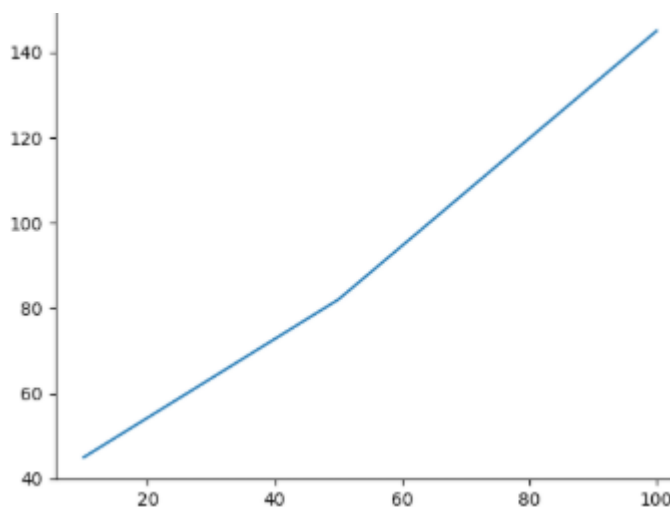


Рис. 4.3. Залежність часу відповіді від кількості одночасних запитів

Короткий опис:

На рисунку 4.5 представлено графік залежності середнього часу відповіді серверної частини системи від кількості одночасних запитів. Зі збільшенням навантаження спостерігається лінійне зростання часу обробки запитів, що є типовою характеристикою клієнт-серверних систем.

При навантаженні до 100 одночасних запитів система зберігає стабільність роботи та не демонструє критичних затримок або помилок. Отримані результати підтверджують достатній рівень продуктивності розробленої системи для використання в корпоративному середовищі середнього масштабу.

Висновки до розділу 4

У третьому розділі виконано безпосередню програмну реалізацію спроектованої системи контролю доступу до корпоративних ресурсів. Створено працездатний модульний серверний додаток на базі платформи Node.js та фреймворку Express.js із використанням пулу з'єднань СУБД PostgreSQL. Реалізовано проміжні програмні шари (middleware) для надійної верифікації цифрових підписів JWT-токенів та динамічної перевірки прав доступу на рівні REST API маршрутів (endpoints).

Клієнтську частину розроблено у вигляді односторінкового веб-застосунку (SPA) на базі бібліотеки React із впровадженням захищених маршрутів (ProtectedRoute) та механізму автоматичного екранування вмісту для захисту від XSS-атак. Інтегровано комплексні безпекові рішення на рівні коду: криптографічне хешування паролів за допомогою ітеративного алгоритму bcrypt із використанням солі (salt), token-based захист від CSRF-атак, обов'язкове шифрування трафіку через HTTPS, а також обмеження частоти запитів (Rate Limiting) для ефективної протидії автоматизованим brute-force атакам.

ВИСНОВКИ

У ході виконання бакалаврської роботи було розроблено та програмно реалізовано систему контролю доступу до корпоративних ресурсів із використанням сучасних методів автентифікації та рольової моделі авторизації.

Актуальність теми зумовлена необхідністю забезпечення захисту корпоративної інформації в умовах зростання кіберзагроз, поширення хмарних сервісів та віддаленого доступу до інформаційних систем. Ефективний контроль доступу є одним із ключових компонентів інформаційної безпеки організації.

У процесі виконання роботи було досягнуто поставленої мети та вирішено основні задачі дослідження.

У першому розділі проведено аналіз предметної області, розглянуто особливості корпоративних інформаційних систем, основні загрози інформаційній безпеці, методи автентифікації та авторизації. Проаналізовано існуючі рішення управління доступом та обґрунтовано вибір рольової моделі RBAC як оптимальної для систем середнього масштабу.

У другому розділі виконано проектування системи контролю доступу. Розроблено архітектуру клієнт-серверного застосунку, визначено структуру компонентів серверної частини, змодельовано рольову модель доступу, спроектовано структуру реляційної бази даних та описано алгоритми автентифікації й авторизації на основі JWT.

У третьому розділі здійснено програмну реалізацію системи із використанням середовища Node.js та фреймворку Express для серверної частини, а також React для клієнтського інтерфейсу. Реалізовано механізми:

безпечної автентифікації з використанням bcrypt;

- token-based авторизації на основі JWT;
- рольової моделі доступу RBAC;
- журналювання дій користувачів;
- захисту REST API від несанкціонованого доступу.

Система забезпечує централізоване управління користувачами та ролями, а також відповідає принципу мінімальних привілеїв.

У четвертому розділі проведено тестування системи, яке включало модульне, інтеграційне та системне тестування. Було розроблено тестові сценарії для перевірки механізмів автентифікації, авторизації та журналювання. Результати тестування підтвердили коректність реалізації функціональних вимог.

Оцінка продуктивності показала, що система стабільно працює при одночасному обслуговуванні до 100 запитів без втрати даних та виникнення помилок. Час відповіді зростає лінійно із збільшенням навантаження, що свідчить про прогнозовану поведінку системи.

Практичне значення отриманих результатів полягає у можливості використання розробленої системи як основи для впровадження в корпоративному середовищі або інтеграції з іншими інформаційними системами підприємства.

Перспективами подальшого розвитку системи є:

- впровадження двофакторної автентифікації (2FA);
- інтеграція з OAuth 2.0 та SSO;
- реалізація атрибутивної моделі доступу (ABAC);
- додавання механізмів моніторингу та виявлення аномалій доступу.

Таким чином, поставлена мета роботи досягнута, а розроблена система контролю доступу відповідає сучасним вимогам інформаційної безпеки та програмної інженерії.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Ferraiolo D., Kuhn R., Chandramouli R. Role-Based Access Control. – 3rd ed. – Artech House, 2023. – 340 p.
2. Stallings W. Cryptography and Network Security: Principles and Practice. – 8th ed. – Pearson, 2022. – 816 p.
3. OWASP Foundation. OWASP Top 10 – 2021: The Ten Most Critical Web Application Security Risks [Електронний ресурс]. – Режим доступу: <https://owasp.org>
4. Tschofenig H., Hardt D. The OAuth 2.1 Authorization Framework // IETF Draft. – 2024.
5. Jones M., Bradley J., Sakimura N. JSON Web Token (JWT) // IETF RFC 7519. – 2015.
6. NIST Special Publication 800-207. Zero Trust Architecture // National Institute of Standards and Technology. – 2020. – 50 p.
7. Node.js Foundation. Node.js v22 Long Term Support Documentation [Електронний ресурс]. – Режим доступу: <https://nodejs.org/docs/>
8. Express.js Documentation. Minimal and flexible Node.js web application framework [Електронний ресурс]. – Режим доступу: <https://expressjs.com>
9. Martin R. Clean Architecture: A Craftsman's Guide to Software Structure and Design. – Prentice Hall, 2018. – 432 p.
10. Bishop M. Computer Security: Art and Science. – Addison-Wesley, 2018. – 1136 p
11. Bayer E. React Up & Running: Building Web Applications. – 2nd ed. – O'Reilly Media, 2022. – 310 p.
12. PostgreSQL Global Development Group. PostgreSQL 16 Documentation [Електронний ресурс]. – Режим доступу: <https://www.postgresql.org/docs/>
13. Aniche M. Effective Software Testing: A developer's guide. – Manning Publications, 2022. – 320 p.
14. Jest JavaScript Testing Framework. Delightful JavaScript Testing Documentation [Електронний ресурс]. – Режим доступу: <https://jestjs.io>

15. Grafana k6 Documentation. Performance and load testing for developers [Электронный ресурс]. – Режим доступа: <https://k6.io/docs/>
16. ISO/IEC 27001:2022. Information security, cybersecurity and privacy protection – Information security management systems – Requirements. – 3rd ed. – ISO, 2022. – 48 p.
17. Indrasiri K., Siriwardena P. API Security in Action. – Manning Publications, 2022. – 390 p.
18. React Working Group. React v18 & v19 Architectural Blueprints [Электронный ресурс]. – Режим доступа: <https://react.dev>
19. Kleppmann M. Designing Data-Intensive Applications: The Big Ideas Behind Reliable, Scalable, and Maintainability Systems. – O'Reilly Media, 2023. – 580 p.
20. Zandstra M. PHP, MySQL, and PostgreSQL: Modern Enterprise Web Development. – 6th ed. – Apress, 2024. – 610 p.
21. Microsoft Learn. Azure Active Directory (Entra ID) Architecture Fundamentals [Электронный ресурс]. – Режим доступа: <https://learn.microsoft.com/en-us/entra/>
22. Okta Developer Relations. Modern Identity and Access Management (IAM) Patterns [Электронный ресурс]. – Режим доступа: <https://developer.okta.com>

ДОДАТОК А

Лістинг вихідного коду ключових компонентів системи контролю доступу

А.1 Фрагмент коду проміжного шару (middleware) для верифікації JWT-токенів (middleware/authMiddleware.js)

```
const jwt = require("jsonwebtoken");

// Middleware для перевірки JWT-токена та ідентифікації користувача
module.exports = function (req, res, next) {
  // Отримання токена із HTTP-заголовка Authorization
  const authHeader = req.headers.authorization;

  if (!authHeader) {
    return res.status(401).json({ message: "Помилка: Відсутній токен  
доступу" });
  }

  // Вилучення токена з формату "Bearer <token>"
  const token = authHeader.split(" ")[1];

  try {
    // Верифікація цифрового підпису токена за допомогою секретного  
ключа
    const decoded = jwt.verify(token, process.env.JWT_SECRET ||  
"SECRET_KEY");

    // Збереження декодованих даних користувача (id, role) для  
наступних обробників
    req.user = decoded;
    next();
  } catch (err) {
```

```
        return res.status(401).json({ message: "Помилка: Недійсний або  
прострочений токен" });  
    }  
};
```

A.2 Фрагмент коду для перевірки прав доступу за роллю RBAC (middleware/roleMiddleware.js)

```
// Замикання для гнучкої передачі необхідної ролі в маршрутизатор  
module.exports = function (requiredRole) {  
    return (req, res, next) => {  
        // Перевірка наявності ідентифікованого користувача та  
        відповідності його ролі  
        if (!req.user || req.user.role !== requiredRole) {  
            return res.status(403).json({  
                message: "Доступ заборонено: недостатньо привілеїв для  
виконання операції"  
            });  
        }  
    }  
  
    // Якщо роль збігається, запит передається до контролера  
    next();  
};  
};
```

A.3 Фрагмент контролера автентифікації користувача (controllers/authController.js)

```
const bcrypt = require("bcrypt");  
const jwt = require("jsonwebtoken");  
const pool = require("../config/db");  
const auditService = require("../services/auditService");  
  
// Функція обробки запиту на вхід до системи
```

```
exports.login = async (req, res) => {  
  const { username, password } = req.body;  
  
  try {  
    // Пошук облікового запису користувача в базі даних  
    const user = await pool.query(  
      "SELECT * FROM users WHERE username=$1",  
      [username]  
    );  
  
    if (!user.rows.length) {  
      return res.status(401).json({ message: "Користувача не знайдено" });  
    }  
  
    // Перевірка криптографічного хешу пароля  
    const validPassword = await bcrypt.compare(password,  
user.rows[0].password_hash);  
    if (!validPassword) {  
      // Фіксація невдалої спроби входу в журналі аудиту  
      await auditService.logAccess(user.rows[0].id, null, 'LOGIN',  
'DENIED', req.ip);  
      return res.status(401).json({ message: "Невірний пароль" });  
    }  
  
    // Генерація JWT-токена із вищитими правами доступу (термін дії -  
30 хвилин)  
    const token = jwt.sign(  
      {  
        user_id: user.rows[0].id,  
        role: user.rows[0].role  
      },  
    );  
  }  
}
```

```

        process.env.JWT_SECRET || "SECRET_KEY",
        { expiresIn: "30m" }
    );

    // Фіксація успішного входу
    await auditService.logAccess(user.rows[0].id, null, 'LOGIN', 'SUCCESS',
req.ip);
    res.json({ token });

    } catch (err) {
        ... // Логування системних помилок
        res.status(500).json({ message: "Внутрішня помилка сервера" });
    }
};

```

A.4 Конфігурація захищених маршрутів API (routes/userRoutes.js)

```

const express = require("express");
const router = express.Router();
const userController = require("../controllers/userController");

// Імпорт проміжних шарів безпеки
const auth = require("../middleware/authMiddleware");
const role = require("../middleware/roleMiddleware");

// Відкритий маршрут для реєстрації
router.post("/register", userController.createUser);

// Захищений маршрут: отримання списку користувачів (тільки для
ADMIN)
router.get("/", auth, role("ADMIN"), userController.getUsers);

```

```
// Захищений маршрут: зміна ролі співробітника (тільки для ADMIN)
router.put("/:id/role", auth, role("ADMIN"), userController.updateRole);
```

```
... // Інші маршрути API
```

```
module.exports = router;
```

A.5 Структура основних таблиць бази даних (database/schema.sql)

```
-- Створення реляційної таблиці облікових записів користувачів
```

```
CREATE TABLE users (  
    id SERIAL PRIMARY KEY,  
    username VARCHAR(100) UNIQUE NOT NULL,  
    email VARCHAR(150) UNIQUE NOT NULL,  
    password_hash VARCHAR(255) NOT NULL,  
    role VARCHAR(50) DEFAULT 'EMPLOYEE',  
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP  
);
```

```
-- Створення таблиці журналу аудиту дій (Access Logs)
```

```
CREATE TABLE access_logs (  
    id SERIAL PRIMARY KEY,  
    user_id INT REFERENCES users(id) ON DELETE CASCADE,  
    resource_id INT,  
    action VARCHAR(50) NOT NULL,  
    status VARCHAR(20) NOT NULL, -- 'SUCCESS' або 'DENIED'  
    ip_address VARCHAR(45),  
    timestamp TIMESTAMP DEFAULT CURRENT_TIMESTAMP  
);
```

```
-- Створення індексів для оптимізації пошуку по журналу аудиту
```

```
CREATE INDEX idx_access_logs_user_id ON access_logs(user_id);  
CREATE INDEX idx_access_logs_timestamp ON access_logs(timestamp);
```

