КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ БУДІВНИЦТВА І АРХІТЕКТУРИ

Автоматизації і інформаційних технологій
(факультет)

Кібербезпеки та комп'ютерної інженерії
(назва випускової кафедри)

КВАЛІФІКАЦІЙНА РОБОТА
ЗДОБУВАЧА СТУПЕНЯ ВИЩОЇ ОСВІТИ БАКАЛАВР

на тему:
Інтеграція та модернізація рольової моделі доступу в інформаційно-комунікаційній системі підприємтва

Дмитрієв Артем Євгенійович
(прізвище, ім'я та по батькові здобувача повністю)

Київ 2026 р.

КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ БУДІВНИЦТВА І АРХІТЕКТУРИ

Автоматизації і інформаційних технологій
(факультет)
Кібербезпеки та комп'ютерної інженерії
(назва кафедри)

ЗАТВЕРДЖУЮ
Завідувач кафедри
Делембовський М.М.
„___" _____2026 року

КВАЛІФІКАЦІЙНА РОБОТА ЗДОБУВАЧА СТУПЕНЯ ВИЩОЇ ОСВІТИ БАКАЛАВР

Інтеграція та модернізація рольової моделі, доступу в інформаційно-комунікаційній системі підприємства
(назва)

Я як здобувач вищої освіти КНУБА розумію і підтримую політику закладу з академічної доброчесності. Я не надавав(-ла) і не одержував(-ла) недозволену допомогу під час підготовки цієї роботи. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело.

Здобувач Дмитрієв Артем Євгенійович
(прізвище, ім'я та по батькові повністю)
123 «Комп'ютерна інженерія»
(спеціальність)
Комп'ютерні системи та мережі
(освітня програма)
Група КСМ-22
Керівник Ізмайлова О.В.
(прізвище та ініціали)
Кандидат технічних наук, доцент
(вчене звання, науковий ступінь)
Рецензент
(прізвище та ініціали)
Ідентичність підтверджую

Київ 2026 р.

КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ БУДІВНИЦТВА І АРХІТЕКТУРИ

Факультет: Автоматизації і інформаційних технологій

Випускова кафедра: Кібербезпеки та комп'ютерної інженерії

Ступінь вищої освіти: бакалавр

Спеціальність: 123 «Комп'ютерна інженерія»

Освітня програма: Комп'ютерні системи та мережі

ЗАТВЕРДЖУЮ
Завідувач кафедри
Делембовський М.М.
„___" _____2026 року

З А В Д А Н Н Я
ДО ВИКОНАННЯ КВАЛІФІКАЦІЙНОЇ РОБОТИ ЗДОБУВАЧА СТУПЕНЯ ВИЩОЇ ОСВІТИ БАКАЛАВР

Дмитрієв Артем Євгенійович
(прізвище, ім'я та по батькові здобувача)

1.Тема роботи «Інтеграція та модернізація рольвої моделі доступу в інформаційно-комунікаційній системі підприємства »
затверджена наказом ректора КНУБА № 576/52-15/13.2/26 від «14» травня 2026 року

2. Керівник роботи
Ізмайлова Ольга Василівна
кандидат технічних наук, доцент
(прізвище, ім'я та по батькові, науковий ступінь, вчене звання)

3. Термін подання здобувачем роботи до захисту_____
4. Зміст пояснювальної записки за розділами:
Р. 1. Компроративний аналіз моделей та нормативних вимог до керування доступом_____
Р. 2. Проєктування гібридної моделі rbac/mac для бізнес-середовища_____
Р. 3. Програмна реалізація гібридної моделі керування доступом rbac mac
Р. 4. _____
Р. 5. _____

5. Календарний план виконання роботи:

Види робіт та їх зміст	Дата виконаня
Розділ 1. Компроматичний аналіз моделей та нормативних вимог до керування доступом	03.05.2026
Розділ 2. Проєктування гібридної моделі rbac/mac для бізнес-середовища	14.05.2026
Розділ 3. Програмна реалізація гібридної моделі керування доступом rbac mac	27.05.2026
Розділ 4.	
Розділ 5	
Остаточне оформлення роботи	29.05.2026
Направлення роботи для перевірки на плагіат	__.__.2026
Попередній захист роботи на випусковій кафедрі	__.__.2026
Направлення роботи на рецензування	

6. Дата видачі завдання _____

Керівник _____ Дмитрієв А.Є.
(підпис) (прізвище та ініціали)

Здобувач _____ Ізмайлова О.В.
(підпис) (прізвище та ініціали)

РЕЗЮМЕ (SUMMARY) *до кваліфікаційної випускової роботи здобувача*	ПІБ *здобувача українською та англійською мовами* Дмитрієв Артем Євгенійович Dmytriiev Artem		
ЗВО	Київський національний університет будівництва і архітектури		
Тема *(українською та англійською)*	Інтеграція та модернізація рольвої моделі доступу в інформаційно-комунікаційній системі підприємства Integration and modernization of the role-based access model in the enterprise's information and communication system		
Освітній ступінь	бакалавр		
Факультет	Автоматизації і інформаційних технологій		
Випускова кафедра	Кібербезпеки та комп'ютерної інженерії		
Спеціальність	123 «Комп`ютерна інженерія»		
Освітня програма	Комп’ютерні системи імережі		
Керівник	Ізмайлова О.В.		
Обсяг роботи:	*Поснювальна записка, стор.*	*Розділів*	*Презентація, кількість слайдів*
	80	3	12
Розділ 1	Проаналізовано моделі MAC, RBAC та ABAC, їхні переваги й обмеження. Розглянуто нормативні вимоги до авторизації, мінімальних прав і аудиту.		
Розділ 2	Спроєктовано гібридну модель RBAC/MAC для бізнес-середовища. Описано архітектуру, алгоритм перевірки доступу, департаменти та міждепартаментне погодження.		
Розділ 3	Описано програмну реалізацію web-прототипу на FastAPI, SQLite та React. Реалізовано користувачів, ролі, ресурси, перевірку доступу, аудит і демонстраційні сценарії		
Висновки по роботі	У роботі розроблено гібридну систему керування доступом, яка поєднує RBAC і MAC для більш гнучкого та безпечного розмежування прав у бізнес-середовищі. Створений web-прототип підтвердив, що врахування ролей, департаментів, посад, рівнів допуску та аудиту дозволяє краще контролювати доступ до ресурсів підприємства.		
Ключові слова: Keywords:	RBAC, MAC, контроль доступу, інформаційна безпека, департамент, рольова модель, примусовий контроль, аудит, FastAPI, React, SQLite.		

Здобувач _Дмитрієв А.Є._ / _____

Керівник _Ізмайлова О.В_ / _____

АНОТАЦІЯ

Кваліфікаційна випускна робота присвячена дослідженню та програмній реалізації гібридної системи керування доступом, що поєднує рольову модель RBAC та примусову модель MAC. У роботі проаналізовано теоретичні засади моделей доступу, їх переваги й обмеження, а також обґрунтовано доцільність використання комбінованого підходу для бізнес-середовища. Запропонована система враховує ролі користувачів, департаментну належність, посадову ієрархію, рівні допуску до ресурсів, авторство документів та контрольоване міждепартаментне делегування прав. Практична частина реалізована у вигляді web-застосунку з backend на FastAPI, базою даних SQLite та frontend-інтерфейсом на React. Система підтримує автентифікацію, керування користувачами, ролями, ресурсами й департаментами, перевірку доступу, аудит подій і демонстраційні сценарії міждепартаментної взаємодії. Отриманий прототип підтверджує доцільність поєднання RBAC і MAC для підвищення безпеки, керованості та прозорості доступу в організації.

Ключові слова: RBAC, MAC, контроль доступу, інформаційна безпека, департамент, рольова модель, примусовий контроль, аудит, FastAPI, React, SQLite.

ABSTRACT

The qualification thesis is devoted to the research and software implementation of a hybrid access control system that combines the Role-Based Access Control model (RBAC) and the Mandatory Access Control model (MAC). The work analyses the theoretical foundations of access control models, their advantages and limitations, and substantiates the relevance of a combined approach for a business environment. The proposed system takes into account user roles, departmental affiliation, position hierarchy, clearance levels, document authorship and controlled cross-department permission delegation. The practical part is implemented as a web application with a FastAPI backend, an SQLite database and a React frontend interface. The system supports authentication, management of users, roles, resources and departments, access checks, event auditing and demonstration scenarios of cross-department cooperation. The developed prototype confirms the feasibility of combining RBAC and MAC to improve security, manageability and transparency of access within an organization.

Keywords: RBAC, MAC, access control, information security, department, role model, mandatory control, audit, FastAPI, React, SQLite.

ЗМІСТ

ВСТУП 11

РОЗДІЛ 1. КОМПАРАТИВНИЙ АНАЛІЗ МОДЕЛЕЙ ТА НОРМАТИВНИХ ВИМОГ ДО КЕРУВАННЯ ДОСТУПОМ 13

1.1. Роль керування доступом у сучасній інформаційній безпеці 13

1.2. Примусове керування доступом MAC та його роль у забезпеченні системної цілісності 14

1.3. Структурна ієрархія та обмеження рольової моделі RBAC 16

1.4. Архітектурна логіка ABAC та контекстно-залежна авторизація 17

1.5. Нормативно-правове регулювання та стандарти 18

1.6. Порівняльний аналіз моделей доступу 20

1.7. Ризики помилок авторизації та вимоги бізнес-середовища 21

1.8. Вимоги до зручності адміністрування та бізнес-термінології 23

1.9. Життєвий цикл прав доступу в організації 24

1.10. Критерії оцінювання якості політики доступу 26

1.11. Автентифікація як передумова авторизації 27

Висновки до розділу 1 29

РОЗДІЛ 2. ПРОЄКТУВАННЯ ГІБРИДНОЇ МОДЕЛІ RBAC/MAC ДЛЯ БІЗНЕС-СЕРЕДОВИЩА 30

2.1. Концептуальні засади інтеграції моделей RBAC та MAC 30

2.2. Архітектура гібридної моделі доступу 31

2.3. Алгоритм обробки запиту доступу 32

2.4. Департаментна ієрархія, системні користувачі та директорський рівень 36

2.5. Механізм міждепартаментного делегування доступу 38

2.6. Нефункціональні вимоги до системи 40
2.7. Формалізація політики доступу у вигляді предикатів 41
2.8. Модель загроз та відповідність механізмів контролю 42
2.9. Правила цілісності даних у проєктній моделі 43
2.10. Пояснюваність рішень і audit-ready підхід 45
2.11. Бізнес-сценарії ролей у матриці доступу 46
Висновки до розділу 2 47
РОЗДІЛ 3. ПРОГРАМНА РЕАЛІЗАЦІЯ ГІБРИДНОЇ МОДЕЛІКЕРУВАННЯ ДОСТУПОМ RBAC ТА MAC 49
3.1. Загальна характеристика програмної реалізації 49
3.2. Вибір інструментів та технологій розробки 50
3.3. Архітектура програмного прототипу 52
3.4. Структура бази даних 54
3.5. Реалізація автентифікації та системних користувачів 56
3.6. Реалізація RBAC-модуля 57
3.7. Реалізація MAC-модуля 58
3.8. Реалізація PDP, PEP та endpoint /access/check 59
3.9. Реалізація керування користувачами 61
3.10. Реалізація керування ресурсами 63
3.11. Реалізація департаментів і дерева департаментів 64
3.12. Реалізація міждепартаментного доступу 65
3.13. Реалізація frontend-частини 67
3.14. Підсистема аудиту та журналювання 68
3.15. Скрипти ініціалізації та міграції 68
3.16. Запуск і перевірка роботи системи 69

3.17. Тестування системи та демонстраційні сценарії 71
3.18. Результати практичної реалізації 72
Висновки до розділу 3 73
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ 77
Додаток А 81
Додаток Б 85

ВСТУП

Останніми роками бізнес-процеси дедалі частіше переносяться у цифрові сервіси, хмарні платформи й розподілені web-застосунки. У таких умовах традиційний захист периметра вже не може бути єдиною основою безпеки. Для організації важливим стає не тільки факт входу користувача в систему, а й те, у якому службовому контексті він виконує дію: до якого департаменту належить, яку посаду займає, який рівень допуску має та які операції вже виконував. Саме тому керування доступом розглядається як один із базових механізмів захисту корпоративних інформаційних ресурсів. [27]

Актуальність теми визначається потребою у такій моделі доступу, яка одночасно підтримує бізнес-гнучкість і не послаблює формальні вимоги безпеки. RBAC добре відображає організаційну структуру підприємства, однак сама по собі роль не завжди гарантує коректну роботу з ресурсами різної конфіденційності. MAC, навпаки, задає жорстку межу доступу за рівнем допуску, але в чистому вигляді може бути незручною для динамічних процесів. Тому в роботі обґрунтовується комбінований підхід, за якого рішення формується тільки після проходження рольової, мандатної та додаткових бізнес-перевірок. [1]

Об’єктом дослідження є процеси керування доступом до інформаційних ресурсів у корпоративних системах.

Предметом дослідження є гібридна модель доступу, яка поєднує механізми RBAC, MAC, департаментної ізоляції, посадової ієрархії та контрольованого міждепартаментного делегування прав. [1]

Метою роботи є теоретичне обґрунтування та програмна реалізація гібридної системи керування доступом, придатної для демонстрації в бізнес-середовищі з кількома департаментами, різними посадовими рівнями та ресурсами різного рівня конфіденційності. [25]

Для досягнення мети потрібно виконати такі завдання: проаналізувати моделі MAC, RBAC та ABAC; визначити нормативні й методичні джерела, на які спирається контроль доступу; спроєктувати гібридну архітектуру; розробити базу даних і API; реалізувати окремі панелі для адміністратора, директора, начальника департаменту і звичайного користувача; перевірити працездатність системи на демонстраційних сценаріях. [1]

Методи дослідження включають порівняльний аналіз моделей доступу, структурне моделювання, проєктування реляційної бази даних, програмну реалізацію web-застосунка, тестування API-запитів та аналіз результатів перевірки доступу. [19; 32]

Практичне значення роботи полягає в тому, що результатом є не лише опис моделі, а функціональний прототип із користувачами, департаментами, ресурсами, журналом аудиту та сценаріями міждепартаментної взаємодії. Такий прототип можна використовувати як демонстраційну основу для подальшого розвитку корпоративної системи авторизації. [7]

РОЗДІЛ 1. КОМПАРАТИВНИЙ АНАЛІЗ МОДЕЛЕЙ ТА НОРМАТИВНИХ ВИМОГ ДО КЕРУВАННЯ ДОСТУПОМ

1.1. Роль керування доступом у сучасній інформаційній безпеці

Керування доступом у межах цієї роботи розглядається як сукупність організаційних і програмно-технічних процедур, що визначають допустимі межі дій користувача. У простих системах така перевірка часто зводиться до введення логіна й пароля, але для корпоративного середовища цього недостатньо. Потрібно враховувати посаду, підрозділ, рівень допуску, тип ресурсу, запитувану дію та можливість подальшого аудиту. [7]

Актуальність контролю доступу підтверджується тим, що помилки авторизації належать до найбільш небезпечних класів уразливостей web-застосунків. MITRE описує improper access control як ситуацію, коли програмний продукт неправильно обмежує доступ до ресурсу для неавторизованого актора [17]. OWASP також розглядає авторизацію як окрему критичну площину перевірки застосунку [10].

У сучасній архітектурі безпеки доступ має бути не одноразовим фактом, а результатом повторюваної оцінки. Кожна дія користувача повинна проходити через механізм перевірки, оскільки навіть легітимний обліковий запис може бути скомпрометований. MITRE ATT&CK окремо виділяє техніку Valid Accounts, яка описує використання чинних облікових записів для початкового доступу, підвищення привілеїв або прихованої присутності в системі [18].

З огляду на це архітектура системи має спиратися на мінімально необхідні привілеї, заборону за замовчуванням, серверну перевірку кожної критичної дії та обов’язкове журналювання. У розробленому прототипі frontend лише ініціює запит, а остаточне рішення приймається на backend-рівні, де зосереджено повну політику доступу (табл. 1.1). [7]

Таблиця 1.1 - Ключові завдання керування доступом

Завдання	**Зміст у межах проєкту**	**Очікуваний ефект**
Ідентифікація суб’єкта	визначення користувача через login та X-User-Id	зв’язування дії з конкретним актором
Авторизація	перевірка RBAC, MAC, департаменту та посади	контроль допустимості дії
Обмеження привілеїв	поділ main.admin, director, department_head, employee	мінімізація надмірних прав
Аудит	запис результатів перевірок і дій	можливість аналізу інцидентів
Бізнес-гнучкість	міждепартаментні запити доступу	контрольована співпраця відділів

1.2. Примусове керування доступом MAC та його роль у забезпеченні системної цілісності

MAC у роботі трактується як примусова модель, у якій правила доступу задаються централізовано й не можуть бути змінені звичайним користувачем або власником ресурсу. Такий підхід особливо важливий для даних різного рівня конфіденційності, де порушення меж між класами інформації може призвести до витоку або неконтрольованого поширення відомостей. [1]

У створеному прототипі принцип MAC подано через числову шкалу допусків: користувач має поле clearance_level, а ресурс — required_clearance_level. Доступ надається лише тоді, коли рівень користувача не нижчий за рівень ресурсу. Така реалізація не претендує на повну модель багаторівневої безпеки, але для

бізнес-прототипу забезпечує зрозумілий і перевірюваний механізм обмеження (табл. 1.2). [1]

Практична цінність MAC полягає в тому, що роль не може самостійно “перекрити” рівень конфіденційності. Наприклад, працівник може мати право читання документів, але ресурс із вищою класифікацією все одно залишиться недоступним. Через це MAC у системі виконує функцію другого, примусового бар’єра після RBAC. [1]

Таблиця 1.2 - Рівні доступу MAC у прототипі

Рівень	**Назва**	**Приклад ресурсу**	**Коментар**
1	Public	загальні інструкції	мінімальні обмеження
2	Internal	внутрішні задачі департаменту	доступно більшості працівників
3	Confidential	управлінські звіти	потребує підвищеного допуску
4	Secret	стратегічні документи	доступ для керівництва
5	Top Secret	критичні системні політики	резерв для розширення системи

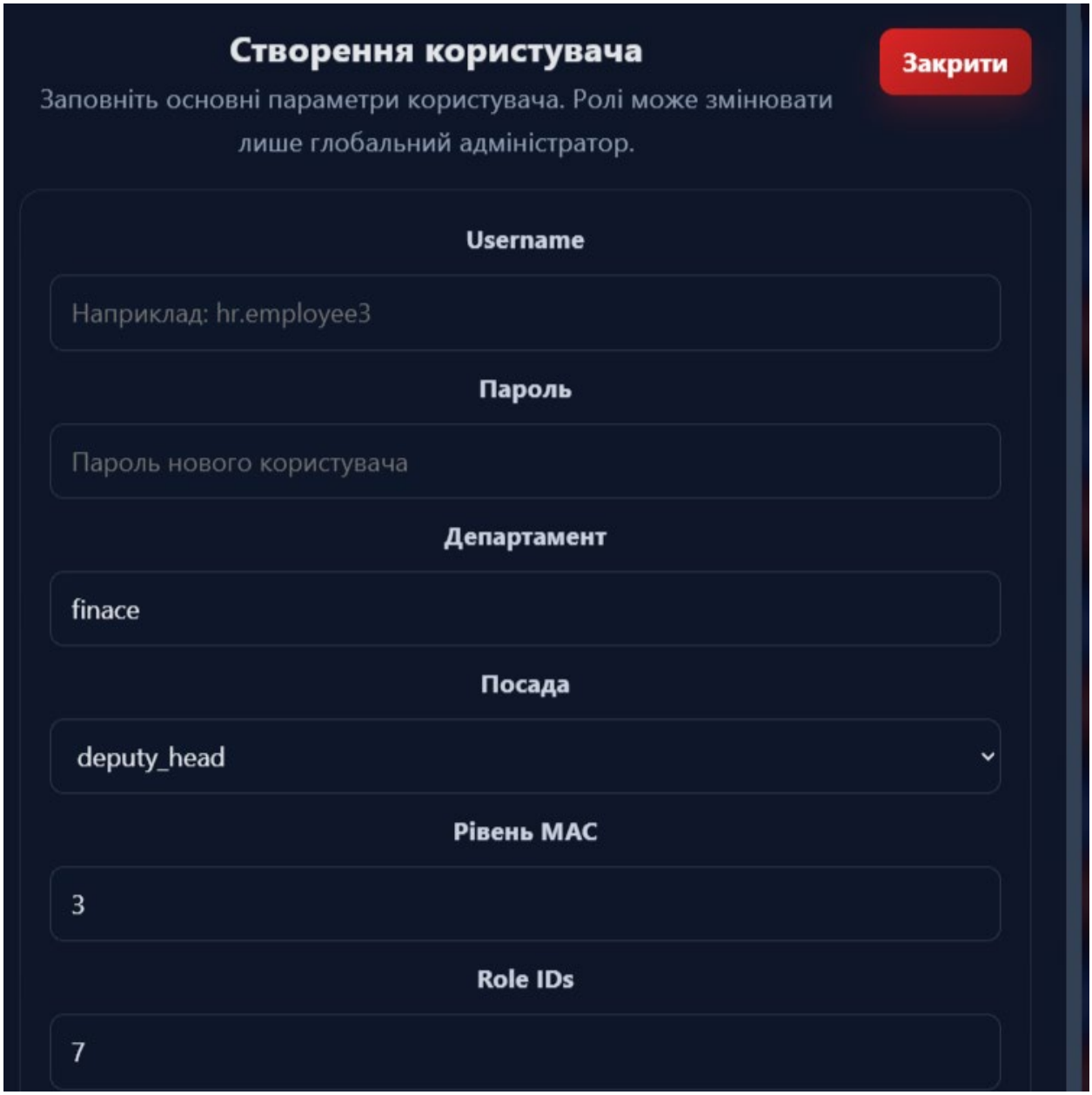


Рисунок 1.1 - Приклад ресурсу з MAC-рівнем

1.3. Структурна ієрархія та обмеження рольової моделі RBAC

RBAC виходить із того, що користувачі виконують у системі певні службові функції, а права зручніше призначати ролям, а не кожному обліковому запису окремо. Такий підхід добре узгоджується з бізнес-структурою, оскільки роль можна пов’язати з посадою, зоною відповідальності або адміністративним рівнем. [2]

У практичній частині RBAC використано для відображення організаційних ролей: адміністратора, директора, начальника департаменту, заступника та підлеглого. Водночас однакова роль у різних департаментах не повинна відкривати однакові ресурси всього підприємства, тому рольова логіка доповнюється департаментною областю дії. [2; 3; 5]

Типовою проблемою RBAC є накопичення великої кількості спеціалізованих ролей. Якщо кожну умову оформлювати окремою роллю, адміністрування швидко стає громіздким. Тому у проєкті роль використовується як базовий каркас, а департамент, посада, MAC-рівень і авторство ресурсу винесені в окремі атрибути перевірки(табл. 1.3). [1]

Таблиця 1.3 - Типові елементи RBAC

Елемент	**Опис**	**Відповідність у системі**
User	людина або службовий обліковий запис	таблиця users
Role	набір повноважень	таблиця roles та системні ролі
Permission	дозвіл на дію	логіка action: read/write/manage/delete
Session	активний контекст роботи	поточний користувач у frontend
Constraint	обмеження ролі	департамент, посада, MAC-рівень

1.4. Архітектурна логіка ABAC та контекстно-залежна авторизація

ABAC у роботі використовується передусім як концептуальне пояснення того, чому для сучасної авторизації недостатньо лише ролі. Рішення про доступ

часто залежить від атрибутів суб’єкта, об’єкта, дії та робочого процесу, тому частина ідей ABAC природно доповнює гібридну RBAC/MAC-логіку. [6]

Практичний прототип не містить повноцінної мови ABAC-політик, однак використовує окремі атрибутивні ознаки: департамент користувача, посаду, рівень допуску, автора ресурсу та статус міждепартаментного запиту. Завдяки цьому доступ можна уточнювати без створення надмірної кількості ролей (табл. 1.4). [1]

Підхід Zero Trust також підсилює аргументацію на користь контекстної перевірки доступу. NIST SP 800-207 наголошує на переході від статичного мережевого периметра до фокусування н а користувачах, активах і ресурсах [8]. Для дипломного прототипу це означає, що кожен запит має перевірятися окремо, навіть якщо користувач уже увійшов у систему.

Таблиця 1.4 - ABAC-атрибути, використані як додаткові обмеження

Категорія атрибута	**Приклад**	**Роль у прототипі**
Атрибут суб’єкта	department_position, clearance_level	уточнює права користувача
Атрибут об’єкта	required_clearance_level, department	визначає рівень захисту ресурсу
Атрибут дії	read, write, manage, delete	визначає тип операції
Атрибут процесу	status=approved для міждепартаментного запиту	дозволяє тимчасову співпрацю

1.5. Нормативно-правове регулювання та стандарти

Нормативна база систем керування доступом не обмежується однією моделлю. Для дипломного проєкту важливими є як класичні джерела з RBAC і MAC, так і документи, що описують загальні принципи контролів безпеки, аудиту та Zero Trust. NIST SP 800-53 містить каталог контролів безпеки й приватності,

який може використовуватися як методичне підґрунтя для обґрунтування потрібності доступу, аудиту та керування привілеями [7].

ISO/IEC 27001 і ISO/IEC 27002 розглядаються як джерела загального управління інформаційною безпекою. ISO/IEC 27001 визначає вимоги до системи управління інформаційною безпекою, а ISO/IEC 27002 надає практичні настанови щодо інформаційних контролів [15] [16]. У контексті роботи ці джерела підтверджують необхідність формалізованої політики доступу, регулярного перегляду прав і контролю дій привілейованих користувачів.

OWASP Authorization Cheat Sheet підкреслює важливість принципу deny by default і перевірки авторизації на серверному боці [10]. У програмному прототипі це реалізовано так, що видимість кнопки у frontend не є достатньою умовою доступу: будь-яке редагування або перегляд ресурсу додатково перевіряється backend-логікою.

XACML та пов’язана термінологія PEP/PDP мають значення для архітектурного пояснення системи. Хоча прототип не використовує XACML як мову політик, він запозичує принцип розділення точки виконання політики та точки прийняття рішення [9]. Це дає змогу описувати систему в термінах, сумісних із загальноприйнятою архітектурою контролю доступу.

Український контекст представлений вимогами технічного захисту інформації та практикою побудови комплексних систем захисту. У роботі ці вимоги враховуються через примусовий контроль, аудит, розмежування прав і неможливість звичайного користувача впливати на системну політику доступу (табл. 1.5). [7]

Таблиця 1.5 - Групи джерел, використані в роботі

Група	Приклади джерел	Призначення
Класичні моделі	Bell-LaPadula, Ferraiolo-Kuhn, Sandhu	теоретичне обґрунтування MAC/RBAC

Продовження таблиці 1.5

Стандарти й настанови	NIST, ISO, OWASP, CISA, ENISA	практичні вимоги до безпеки
Технологічна документація	FastAPI, SQLAlchemy, SQLite, React	обґрунтування інструментів
Проєктні джерела	репозиторій Diploma	підтвердження практичної реалізації

1.6. Порівняльний аналіз моделей доступу

Порівняння MAC, RBAC та ABAC показує, що кожна модель вирішує окремий клас задач. MAC добре працює в системах із формалізованими рівнями безпеки, RBAC зручна для відображення організаційної структури, а ABAC забезпечує гнучкість через атрибути та контекст. Однак у чистому вигляді жодна модель не є достатньою для опису всієї логіки сучасного бізнес-середовища (табл. 1.6). [1]

У дипломному проєкті обрано практичне поєднання RBAC і MAC з окремими атрибутивними обмеженнями. Це рішення дає змогу зберегти зрозумілу рольову структуру і водночас запобігти неконтрольованому доступу до ресурсів підвищеної конфіденційності. Отже, гібридна система не є механічним додаванням двох моделей, а формує багаторівневий алгоритм прийняття рішення. [1]

Важливою особливістю є принцип пріоритету заборони. Якщо хоча б одна з обов’язкових перевірок повертає негативний результат, доступ блокується. Це відповідає логіці безпечного проєктування: система не повинна дозволяти дію лише тому, що одна з моделей дала дозвіл, якщо інша модель виявила порушення політики. [21]

Таблиця 1.6 - Порівняння MAC, RBAC та ABAC

Критерій	MAC	RBAC	ABAC
Основний принцип	рівні та мітки безпеки	ролі та дозволи	атрибути та політики
Перевага	жорсткий контроль	просте адміністрування	гнучкість контексту
Недолік	жорсткість	вибух ролей	складність правил
Роль у проєкті	остаточний контроль	первинний фільтр	додаткові атрибути
Бізнес-цінність	захист конфіденційних ресурсів	зв’язок з посадою	гранулярне уточнення

1.7. Ризики помилок авторизації та вимоги бізнес-середовища

Під час проєктування гібридної системи важливо враховувати не тільки формальні властивості моделей, а й типові ризики корпоративного середовища. Найпоширенішою проблемою є наявність надмірних привілеїв, коли користувач зберігає доступ до ресурсів після зміни посади або переходу до іншого департаменту. У межах прототипу ця проблема врахована через можливість редагування департаменту, посадового рівня й ролей користувача, а також через серверну фільтрацію видимості. [21]

Другим ризиком є використання легітимного облікового запису не за призначенням. Якщо зловмисник отримує доступ до пароля працівника, проста перевірка login/password не гарантує безпеки. З огляду на це кожна дія в системі додатково перевіряється за департаментом, MAC-рівнем, посадою та авторством

ресурсу. Це не усуває потребу в сильній автентифікації, але зменшує наслідки компрометації одного облікового запису. [1]

Третій ризик пов’язаний із неузгодженим міжвідділовим обміном інформацією. У реальних організаціях департаменти не можуть бути повністю ізольованими, однак прямий доступ до чужих ресурсів створює загрозу витоку даних. Для цього в системі використано погоджений workflow: начальник департаменту-ініціатора формує запит, начальник цільового департаменту приймає рішення, а MAC-рівень залишається обов’язковим бар’єром. [1]

Четвертий ризик полягає в залежності безпеки від frontend-інтерфейсу. Якщо доступ контролюється лише приховуванням кнопок, користувач може спробувати виконати дію напряму через API. Тому у прототипі frontend виконує лише допоміжну роль: він показує або приховує елементи керування, але остаточне рішення завжди приймається backend-політикою. [19; 32]

П’ятий ризик стосується відсутності пояснюваності рішень доступу. Для адміністратора недостатньо знати, що доступ заборонено; потрібно розуміти причину. З огляду на це endpoint перевірки доступу повертає reason, а audit log зберігає деталі події. Це дає змогу відрізнити помилку конфігурації від справжньої спроби порушення політики. [7]

Отже, гібридна модель у межах проєкту розглядається як відповідь не лише на теоретичні недоліки RBAC і MAC, а й на практичні ризики бізнес-середовища: надмірні права, компрометацію облікових записів, міждепартаментний обмін, помилки інтерфейсу та відсутність аудиту (табл. 1.7). [1]

Таблиця 1.7 - Ризики авторизації та способи їх зменшення

Ризик	**Прояв у системі**	**Контрольний механізм**
Надмірні привілеї	користувач зберігає доступ після зміни посади	редагування департаменту, ролей і position

Продовження таблиці 1.7

Компрометація акаунта	легітимний користувач використовується зловмисником	перевірка кожної дії та аудит
Чужий департамент	потреба тимчасової співпраці	workflow погодження доступу
Помилка UI	кнопка показана помилково	обов’язкова backend-перевірка
Непояснена відмова	адміністратор не бачить причину	reason + audit log

1.8. Вимоги до зручності адміністрування та бізнес-термінології

Гібридна система має бути не тільки безпечною, а й придатною для щоденного використання адміністраторами та керівниками підрозділів. Тому під час проєктування було враховано принцип адміністративної зрозумілості: користувач системи повинен працювати з термінами, близькими до бізнесу, а не з внутрішніми технічними кодами. [27]

У frontend технічні значення department_head, deputy_head і employee відображаються як начальник департаменту, заступник начальника та підлеглий. Аналогічно, технічні назви департаментів hr, finance, lawyer і engineer подаються як HR-відділ, фінансовий, юридичний та інженерний відділи. Це не змінює модель даних, але підвищує зрозумілість інтерфейсу. [21]

Окремою вимогою є підтримка життєвого циклу співробітника. Працівник може бути створений, переведений до іншого департаменту, змінити посаду або отримати новий рівень допуску. З огляду на це модуль користувачів підтримує не лише створення, а й редагування department, department_position і clearance_level. [27]

Для керівників підрозділів важливо мати швидкий огляд власного департаменту. Тому дерево департаментів реалізовано у вигляді accordion: департамент розгортається, після чого видно користувачів, ресурси та дозволені дії. Глобальний адміністратор бачить усю структуру, а начальник відділу - тільки власний підрозділ. [27]

У звичайного працівника інтерфейс навмисно обмежений. Йому не потрібні ролі, системні налаштування або журнал аудиту. UserPanel показує профіль, доступні ресурси, недоступні ресурси та пояснення причин. Це знижує складність інтерфейсу та зменшує ризик випадкової адміністративної дії (табл. 1.8). [7]

Таблиця 1.8 - Бізнес-вимоги до інтерфейсу

Вимога	**Реалізація**	**Очікуваний результат**
Зрозумілі назви	відображення business labels у frontend	менше помилок під час демонстрації
Життєвий цикл співробітника	редагування департаменту й посади	підтримка переведень
Огляд структури	дерево департаментів	швидкий контроль підрозділів
Обмеження user panel	приховані адміністративні модулі	мінімізація зайвих дій

1.9. Життєвий цикл прав доступу в організації

Життєвий цикл прав доступу в організації не є статичним. Співробітник проходить етапи прийняття на роботу, призначення до департаменту, отримання ролей, зміни посади, тимчасового залучення до міжвідділової задачі та звільнення. Якщо система доступу не враховує ці зміни, права накопичуються й утворюють ризик надмірних привілеїв. [7; 10; 12; 27]

У межах запропонованого прототипу життєвий цикл моделюється через редагування користувача. Адміністратор або уповноважений начальник може

змінити department, department_position і clearance_level. Це дає змогу не створювати нового користувача при переведенні людини в інший відділ, а коректно оновити її атрибути доступу. [6]

Для систем безпеки важливо розділяти постійні та тимчасові права. Постійні права описуються ролями й департаментною належністю. Тимчасові права описуються міждепартаментним запитом зі статусом approved. Такий поділ дає змогу не засмічувати RBAC-структуру додатковими ролями для короткострокових задач. [2; 3; 5]

Особливої уваги потребує етап підвищення або зниження посади. Якщо employee стає deputy_head, система повинна дозволити йому ширший контроль тільки в межах його департаменту. Якщо ж користувач переходить на нижчу посаду, надлишкові права повинні бути відкликані шляхом зміни department_position або role_ids. [27]

Окремий випадок - зміна MAC-рівня. Підвищення clearance_level відкриває доступ до ресурсів вищого рівня лише за умови, що інші політики також дозволяють дію. Зниження clearance_level має негайно заблокувати доступ до ресурсів, які перевищують новий рівень користувача. [1]

Отже, гібридна модель розглядає права доступу як стан, що постійно змінюється. З огляду на це всі рішення мають обчислюватися під час кожного запиту, а не зберігатися як разовий дозвіл без повторної перевірки. [21]

У реальному впровадженні життєвий цикл прав може бути пов’язаний із кадровою системою або корпоративним каталогом користувачів. У дипломному прототипі ця інтеграція не реалізується, але структура даних уже дає змогу розвивати систему в цьому напрямі (табл. 1.9). [27]

Таблиця 1.9 - Життєвий цикл прав доступу

Етап	**Дані, що змінюються**	**Наслідок для доступу**
Прийняття працівника	username, department, role_ids	створення базових прав

Продовження таблиці 1.9

Перехід у департамент	department	зміна області видимості
Підвищення посади	department_position	зміна прав редагування
Зміна допуску	clearance_level	зміна MAC-доступу
Тимчасова задача	cross access request	точковий доступ до чужого ресурсу
Звільнення	видалення або деактивація	припинення доступу

1.10. Критерії оцінювання якості політики доступу

Оцінювання якості системи керування доступом має проводитися не лише за фактом наявності ролей. Важливо перевіряти, чи система зберігає мінімальність прав, чи може пояснити причину рішення, чи підтримує аудит і чи не дає змогу обходити політики через альтернативні шляхи. [7]

Першим критерієм є повнота політики. Для кожної критичної дії має існувати зрозуміле правило: хто може її виконувати, за яких умов і які дані перевіряються. У прототипі це забезпечено через розділення логіки на ролі, MAC-рівні, департаменти, посади й авторство. [1]

Другим критерієм є несуперечливість. Політики не повинні створювати ситуацію, де одна частина системи дає змогу дію, а інша мовчки її виконує без перевірки. З огляду на це frontend-обмеження дублюються серверною перевіркою. [23]

Третім критерієм є пояснюваність. Система повинна повертати reason і записувати подію в audit log. Без цього адміністратор не може якісно підтримувати політику та швидко виявляти неправильні налаштування. [7]

Четвертим критерієм є відновлюваність. Наявність main.admin як системного користувача дає змогу повернути контроль у разі помилки зі звичайними ролями.

Це практичне рішення для демонстраційної системи, де під час розробки ролі можуть змінюватися багато разів. [27]

П’ятим критерієм є масштабованість моделі. Система не повинна вимагати створення нової ролі для кожної контекстної умови. У проєкті це вирішено через додаткові атрибути: department, department_position, clearance_level і статус міждепартаментного запиту. [6]

Шостим критерієм є демонстраційна придатність. Дипломний прототип має бути зрозумілим під час захисту, тому рішення про доступ показується у вигляді дозволено/заборонено з поясненням, а не лише як внутрішній код (табл. 1.10). [7; 10; 11; 28]

Таблиця 1.10 - Критерії якості політики доступу

Критерій	**Питання перевірки**	**Реалізація у прототипі**
Повнота	чи всі критичні дії мають правило	access_policy + router checks
Несуперечливість	чи немає обходу через UI/API	backend validation
Пояснюваність	чи видно причину рішення	reason + audit log
Відновлюваність	чи можна повернути контроль	main.admin
Масштабованість	чи не виникає вибух ролей	атрибути замість надлишкових ролей
Демонстраційність	чи зрозуміло рішення комісії	frontend access check

1.11. Автентифікація як передумова авторизації

Хоча основна увага дипломного проєкту зосереджена на авторизації, автентифікація залишається її потрібною передумовою. Система не може коректно застосувати RBAC або MAC, якщо не визначено, хто саме виконує дію. Тому першим кроком будь-якого сценарію є встановлення особи користувача через login. [1]

У прототипі використано спрощену форму входу з username і password. Паролі не зберігаються у відкритому вигляді: backend використовує password_hash. Це принципово важливо, оскільки навіть навчальний прототип не повинен демонструвати небезпечну практику зберігання відкритих паролів. [10; 19; 24; 27]

Після входу система повертає не пароль і не повний внутрішній об’єкт бази даних, а контрольований набір атрибутів користувача. До нього входять id, username, department, department_position, clearance_level, roles і службові прапорці. Саме ці дані використовуються для маршрутизації користувача до відповідної панелі. [2; 3; 5]

У реальній системі автентифікацію потрібно було б посилити через JWT-токени, строк дії сесії, оновлення токена, захист cookie, двофакторну автентифікацію або інтеграцію з корпоративним SSO. Проте для дипломної демонстрації достатньо показати сам принцип: ідентичність користувача передує будь-якій перевірці доступу. [27]

Важливо, що успішний login не означає автоматичного доступу до ресурсів. Він лише створює контекст поточного актора. Подальші дії все одно проходять через RBAC, MAC, департаментну область, посадову ієрархію та аудит. [1]

Отже, у прототипі автентифікація і авторизація розмежовані. Автентифікація відповідає на питання «хто користувач», а авторизація - «що цей користувач може зробити в конкретному контексті». Таке розмежування робить архітектуру зрозумілішою і наближеною до практики побудови web-застосунків. [27]

Таблиця 1.11 - Зв’язок автентифікації та авторизації

Етап	Призначення	Результат
Login	перевірити username/password	визначений користувач
Hash check	не зберігати пароль відкрито	безпечніше зберігання
User context	передати id, roles, department	основа авторизації
Panel routing	визначити default_panel	правильний інтерфейс
Access check	перевірити конкретну дію	ALLOW або DENY

Висновки до розділу 1

У першому розділі встановлено, що керування доступом є фундаментальним елементом інформаційної безпеки, оскільки саме воно визначає допустимі межі взаємодії користувачів із ресурсами. MAC, RBAC та ABAC мають різну природу, але можуть бути взаємодоповнювальними в межах єдиної архітектури. [1]

MAC забезпечує централізований примусовий контроль, RBAC відображає організаційну структуру підприємства, а ABAC надає концептуальну основу для використання додаткових атрибутів і контексту. Аналіз джерел і стандартів показав, що сучасні системи мають спиратися на принципи мінімальних прав, deny by default, серверної перевірки авторизації та обов’язкового журналювання подій. [1]

Отже, для подальшого проєктування доцільно обрати гібридний підхід, у якому RBAC виконує роль первинного фільтра, MAC - роль обов’язкового обмежувального механізму, а бізнес-атрибути забезпечують деталізацію правил доступу. [1]

РОЗДІЛ 2. ПРОЄКТУВАННЯ ГІБРИДНОЇ МОДЕЛІ RBAC/MAC ДЛЯ БІЗНЕС-СЕРЕДОВИЩА

2.1. Концептуальні засади інтеграції моделей RBAC та MAC

Проєктування гібридної системи ґрунтується на розподілі відповідальності між RBAC і MAC. RBAC визначає, чи має користувач функціональне право на певну дію, а MAC перевіряє, чи дозволяє рівень допуску працювати з обраним ресурсом. У такій схемі жодна з моделей не замінює іншу; вони послідовно звужують простір дозволених операцій. [1]

Формально запит доступу можна описати як Request = (S, O, A), де S — суб’єкт, O — об’єкт, A — дія. Позитивне рішення можливе лише тоді, коли суб’єкт має достатні функціональні повноваження для дії A, його рівень допуску відповідає рівню ресурсу O, а область доступу не порушує меж департаменту або має погоджений виняток. [27]

У прототипі враховується не лише класична трійка “користувач — роль — дозвіл”. Модель доповнена департаментом, посадовою ієрархією та авторством ресурсу. Це дає змогу ближче відтворити реальну бізнес-логіку: начальник має ширші права у своєму відділі, але не отримує автоматичного контролю над чужими ресурсами. [21]

Для фінального рішення використано принцип пріоритету заборони. Якщо рольова перевірка дозволяє дію, але MAC її блокує, результатом буде відмова. Аналогічно, достатній MAC-рівень не відкриває доступ без відповідної ролі, департаментної області або погодженого винятку. [10]

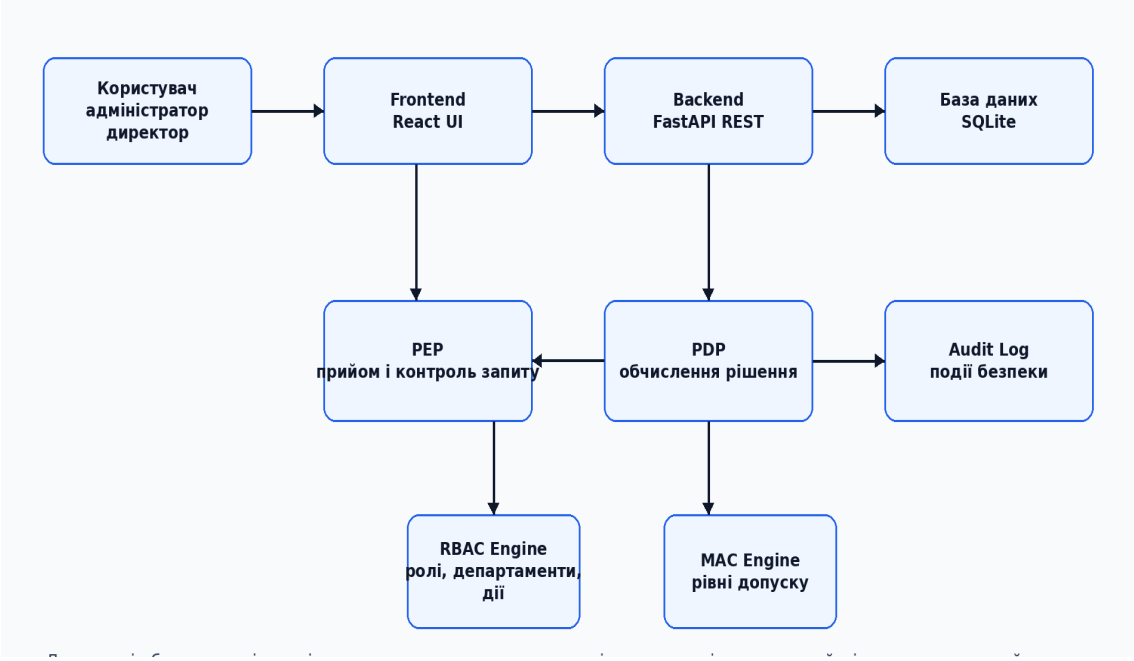


Рисунок 2.1 - Концептуальна архітектура гібридної системи

2.2. Архітектура гібридної моделі доступу

Архітектура системи побудована за принципом розділення відповідальності між інтерфейсом, точкою застосування політики та точкою прийняття рішення. Користувач ініціює дію через frontend, PEP контролює коректність запиту й передає його до PDP, а PDP виконує послідовні перевірки політики доступу. [9]

У прототипі роль PEP виконують API-маршрути та частина frontend-логіки. Вони не дозволяють змінювати ресурс напряму, минаючи backend. Водночас інтерфейс не вважається джерелом довіри: навіть якщо кнопка відображена помилково, сервер повторно перевіряє права. [9]

PDP реалізовано через модулі access_policy.py, auth_context.py та відповідні router-и. Він отримує користувача, ресурс і дію, після чого перевіряє системний статус, департамент, роль, посаду, MAC-рівень і міждепартаментний дозвіл. Результат повертається разом із поясненням причини. [1]

Роль Policy Storage у прототипі виконує база даних SQLite. У ній зберігаються користувачі, ролі, департаменти, ресурси, журнали аудиту та заявки

на міждепартаментний доступ. Завдяки цьому зміни в даних одразу впливають на правила доступу без переписування коду (табл. 2.1). [7]

Таблиця 2.1 - Логічні рівні архітектури

Рівень	**Реалізація у прототипі**	**Функція**
Access Layer	React UI, REST API	ініціювання запитів користувача
PEP	endpoint-и FastAPI, перевірка поточного актора	примусове застосування політики
PDP	access_policy.py, auth_context.py	обчислення рішення
Policy Storage	SQLite + SQLAlchemy models	зберігання політик і сутностей
Audit	audit_logs	реєстрація подій

2.3. Алгоритм обробки запиту доступу

Вхідними даними алгоритму є суб’єкт доступу, ресурс і дія. Формально запит можна подати як Request = (S, O, A), де S - користувач або службовий обліковий запис, O - ресурс системи, а A - дія, яку потрібно виконати. У практичній реалізації дія може мати значення read, write, manage або delete. Для кожного такого запиту система визначає контекст: департамент користувача, посадовий рівень, MAC-рівень, департамент ресурсу, рівень конфіденційності ресурсу, автора ресурсу та наявність погодженого міждепартаментного дозволу. [1]

Алгоритм обробки запиту доступу є центральним елементом проєктної моделі, оскільки саме він поєднує рольову логіку RBAC, примусові обмеження MAC, департаментну ізоляцію, посадову ієрархію та міждепартаментне

делегування. На відміну від спрощених схем, у яких достатньо перевірити лише наявність ролі, у запропонованій системі кожен запит проходить через послідовний набір контрольних етапів. Це дає змогу уникнути ситуацій, коли одна позитивна умова помилково відкриває доступ до ресурсу, що має додаткові обмеження. [1]

Першим етапом є автентифікація та ідентифікація поточного актора. У демонстраційному прототипі після успішного входу frontend зберігає ідентифікатор користувача та передає його до backend у заголовку X-User-Id. Такий механізм є спрощеним у порівнянні з повноцінною JWT-автентифікацією, однак він достатній для демонстрації логіки дипломного проєкту, оскільки кожна дія чітко прив’язується до конкретного користувача. [23]

Другим етапом є нормалізація запиту. Backend перевіряє, чи існує вказаний ресурс, чи підтримується запитувана дія, а також збирає атрибути, необхідні для прийняття рішення. На цьому етапі формується робочий контекст PDP: поточний користувач, його ролі, системний статус, департамент, department_position, clearance_level, ресурс, department ресурсу, required_clearance_level, required_position_level і created_by_user_id. [6]

Третім етапом є перевірка системного статусу. Користувач main.admin розглядається як службовий повний адміністратор, тому має найвищий рівень контролю. Разом із тим director не отримує автоматично повний доступ до всіх операцій: директорський рівень у моделі обмежений керуванням структурою департаментів. [21]

Четвертим етапом є RBAC-перевірка. Вона визначає, чи має користувач функціональне право виконувати дію A. На цьому рівні враховуються призначені ролі, посадовий рівень у департаменті та тип операції. Наприклад, employee може переглядати доступні ресурси й створювати власні записи, однак не повинен редагувати ресурс, створений начальником департаменту. [2; 3; 5]

П’ятим етапом є перевірка департаментної області доступу. Якщо користувач і ресурс належать до одного департаменту, система застосовує внутрішні правила ієрархії та авторства. Якщо ресурс належить іншому департаменту, прямий доступ

забороняється, доки не буде знайдено активного погодженого міждепартаментного дозволу для конкретного користувача, ресурсу та дії. [27]

Шостим етапом є MAC-перевірка. Вона виконується незалежно від того, наскільки високою є роль користувача в організаційній ієрархії. Основне правило полягає в тому, що clearance_level користувача має бути більшим або дорівнювати required_clearance_level ресурсу. Якщо рівень допуску нижчий, доступ блокується навіть у випадку, коли RBAC-логіка або міждепартаментне погодження дає позитивний результат. [1]

Сьомим етапом є застосування додаткових правил авторства та посадової ієрархії. Для операцій запису, редагування або видалення важливо враховувати не тільки факт належності до департаменту, а й те, хто створив ресурс. Employee може редагувати власні ресурси або ресурси свого рівня, deputy_head - ресурси свого рівня та нижчих рівнів, а department_head - ресурси всього свого департаменту. [27]

Восьмим етапом є формування фінального рішення за принципом deny overrides. Позитивне рішення можливе лише тоді, коли всі обов’язкові перевірки повертають дозвіл. Якщо хоча б один компонент - RBAC, MAC, департаментна політика, посадова ієрархія або міждепартаментний дозвіл - повертає заборону, результатом буде DENY (табл. 2.2). [1]

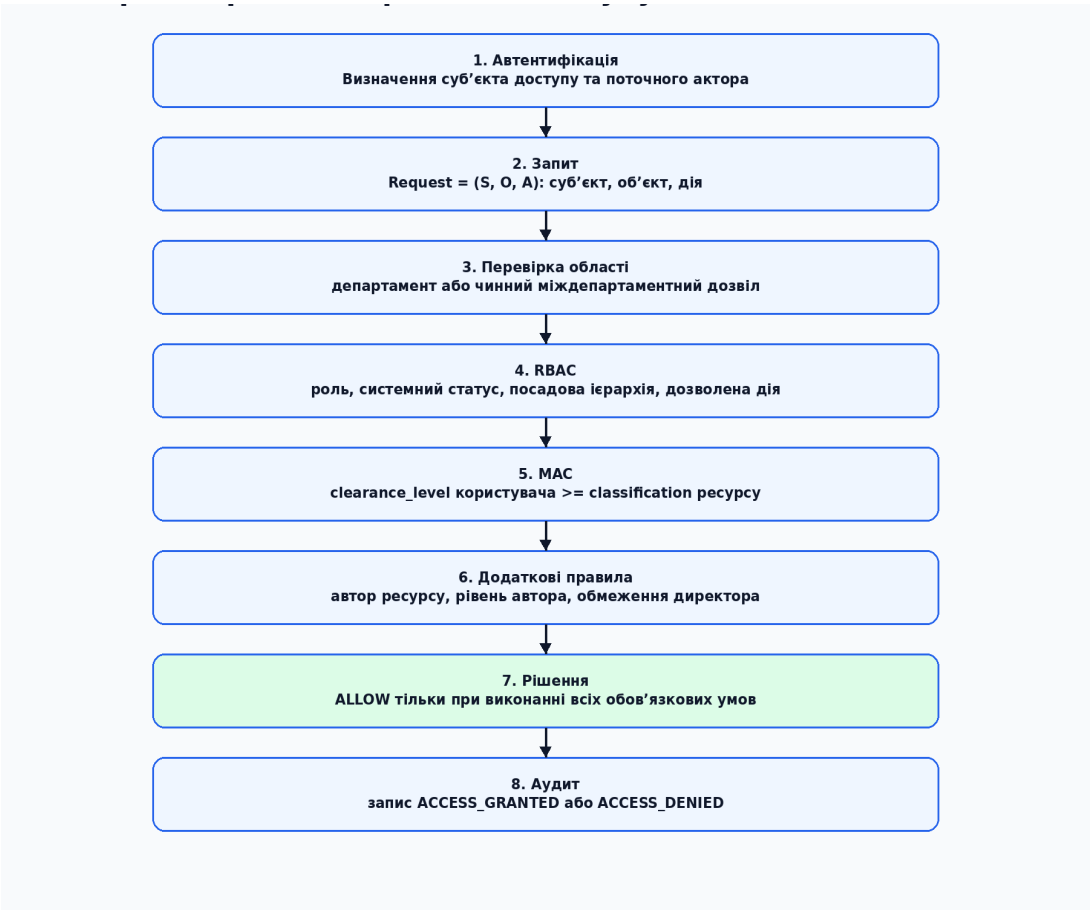


Рисунок 2.2 Алгоритм обробки запиту доступу

Завершальним етапом є аудит. Незалежно від того, чи було доступ дозволено або заборонено, система формує запис у журналі подій. У ньому фіксуються користувач, ресурс, дія, результат і причина. Це забезпечує трасованість роботи політики доступу та створює основу для виявлення потенційних інцидентів безпеки. [7]

Окрему роль відіграє пояснення причини рішення. Відповідь системи не обмежується лише булевим значенням true або false. Для адміністратора, користувача й комісії важливо бачити, чому доступ було дозволено або відхилено: через недостатній MAC-рівень, відсутність ролі, чужий департамент, недостатній посадовий рівень або відсутність погодженого міждепартаментного запиту. [1]

Таблиця 2.2 - Матриця deny overrides

RBAC	MAC	Додаткові правила	Фінальне рішення
ALLOW	ALLOW	ALLOW	ALLOW
ALLOW	DENY	ALLOW	DENY
DENY	ALLOW	ALLOW	DENY
ALLOW	ALLOW	DENY	DENY
DENY	DENY	DENY	DENY

2.4. Департаментна ієрархія, системні користувачі та директорський рівень

Для наближення прототипу до реального бізнес-середовища в системі введено департаменти: HR, finance, lawyer, engineer, а також прихований system-департамент. Кожен звичайний користувач належить до певного департаменту, а ресурси також мають департамент-власник. Це дає змогу реалізувати ізоляцію: начальник HR не може довільно редагувати ресурси finance, навіть якщо має високу посаду у своєму відділі. [25]

Усередині департаменту визначено три посадові рівні: employee, deputy_head і department_head. Вони відображають просту організаційну ієрархію. Підлеглий може створювати ресурси й редагувати ресурси свого рівня, заступник може редагувати ресурси підлеглих і свого рівня, а начальник - усі ресурси власного департаменту. При цьому жоден із них не отримує автоматичного права працювати в іншому департаменті (табл. 2.3). [27]

Окремо передбачено системних користувачів main.admin, director і deputy.director. Вони не є звичайними ролями в таблиці roles. Такий підхід захищає систему від ситуації, коли випадкове видалення ролі admin позбавляє можливості відновити повний контроль. main.admin має повний доступ, director керує департаментами, а deputy.director є службовим системним рівнем. [27]

Директорський рівень спроєктовано так, щоб директор міг створювати та видаляти департаменти, але не отримував автоматичного права редагувати людей усередині них. Це важливе бізнес-обмеження: керівник організаційної структури може змінювати склад відділів, але внутрішнє адміністрування співробітників залишається за начальниками відповідних департаментів або глобальним адміністратором(табл. 2.3). [2; 3; 10; 27]

Таблиця 2.3 - Посадова ієрархія в межах департаменту

Посада	**Рівень**	**Права у межах департаменту**	**Обмеження**
employee	1	створення власних ресурсів, перегляд доступних ресурсів	не редагує ресурси керівництва
deputy_head	2	редагування ресурсів employee/deputy	не редагує ресурси head
department_head	3	керування користувачами і ресурсами свого відділу	не керує чужими відділами
director	system	створення/видалення департаментів	не керує людьми всередині
main.admin	system	повний контроль системи	прихований службовий рівень

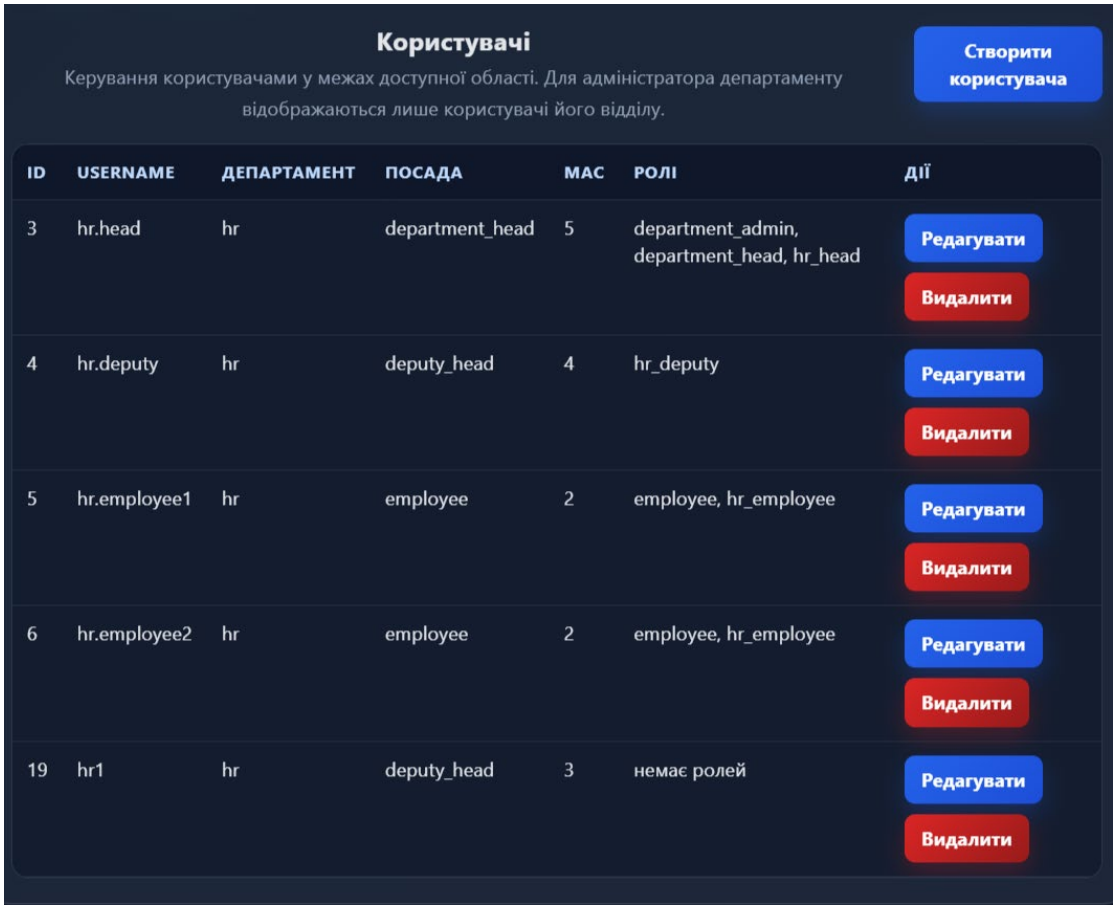


Рисунок 2.3 - Панель департаменту HR

2.5. Механізм міждепартаментного делегування доступу

У реальній організації повна ізоляція департаментів не завжди є достатньою. Іноді співробітник HR повинен тимчасово переглянути фінансовий документ, або інженерному департаменту потрібно отримати інформацію з юридичного відділу. Проте пряме відкриття всіх ресурсів іншого департаменту суперечить принципу мінімальних прав. Тому в системі спроєктовано механізм погодженого міждепартаментного доступу. [27]

Процес побудовано як workflow: начальник департаменту-ініціатора створює запит для конкретного співробітника на конкретний ресурс іншого департаменту. Запит має дію read або write, причину та статус pending. Начальник цільового департаменту переглядає вхідні запити й може схвалити або відхилити їх (рис.2.4). [27]

Після схвалення створюється активний дозвіл. Однак цей дозвіл не скасовує MAC-рівень. Якщо користувач не має достатнього clearance_level, доступ не буде надано навіть після погодження начальників. Отже, система поєднує бізнес-гнучкість з формальним контролем безпеки. [1]

Міждепартаментний доступ також не означає переведення користувача в інший департамент. Він діє тільки для конкретного user_id, конкретного resource_id і конкретної action. Це дає змогу уникнути надмірних привілеїв і забезпечує контрольованість тимчасової співпраці. [2; 3; 5]

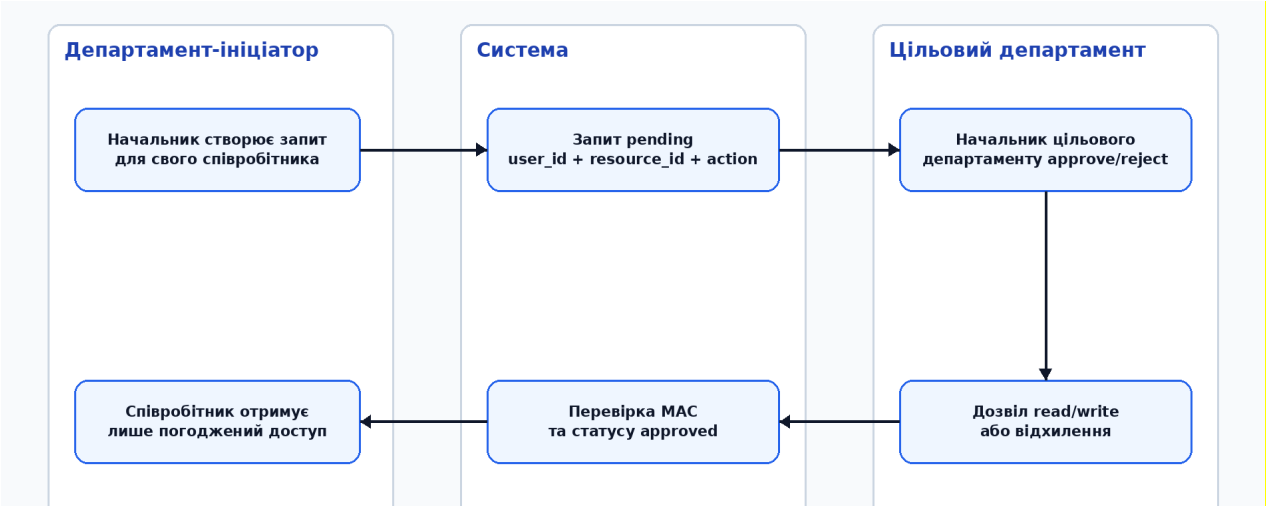


Рисунок 2.4 - Схема погодження міждепартаментного доступу

Таблиця 2.4 - Статуси міждепартаментного запиту

Статус	Значення	Дія системи
pending	очікує рішення	доступ ще не надано
approved	погоджено цільовим департаментом	доступ може бути врахований policy engine
rejected	відхилено	доступ не надається
revoked	відкликано	раніше наданий дозвіл припиняється

2.6. Нефункціональні вимоги до системи

Окрім функціональних можливостей, система має відповідати нефункціональним вимогам.

По-перше, вона повинна бути модульною, щоб зміна одного елемента не призводила до переписування всієї системи. З огляду на це backend поділено на router-и, моделі, схеми, політики доступу й контекст автентифікації. [21]

По-друге, система повинна бути прозорою у прийнятті рішень. Користувач або адміністратор має бачити не лише факт дозволу чи відмови, а й пояснення. Це має значення для тестування, для захисту дипломного проєкту та для реального аудиту безпеки. [7]

По-третє, система повинна бути розширюваною. Надалі можна замінити SQLite на PostgreSQL, додати JWT-токени, повноцінну ABAC-політику, ролі на рівні проєктів, терміни дії міждепартаментних дозволів або інтеграцію з корпоративним SSO. [6]

По-четверте, система має бути захищеною від помилок frontend-рівня. Кнопки редагування приховуються там, де користувач не має прав, але сервер усе одно виконує остаточну перевірку. Це відповідає принципу, що клієнтський інтерфейс не може бути джерелом довіри (табл. 2.5). [23]

Таблиця 2.5 - Нефункціональні вимоги

Вимога	Прояв у системі	Значення
Модульність	окремі router-и й компоненти	спрощення підтримки
Розширюваність	можливість додати PostgreSQL/JWT/ABAC	розвиток після дипломного проєкту
Прозорість	reason у відповіді access/check	пояснюваність рішень
Аудит	audit_logs	відстеження подій

Продоження таблиці 2.5

Безпека backend	серверна перевірка всіх дій	неможливість обходу через UI

2.7. Формалізація політики доступу у вигляді предикатів

Для уникнення неоднозначностей політика доступу в системі може бути подана як набір послідовних предикатів. Перший предикат визначає, чи є користувач системним актором. Другий перевіряє, чи належить ресурс до того самого департаменту, що й користувач. Третій оцінює роль і посадовий рівень, четвертий - MAC-рівень, а п’ятий - наявність погодженого міждепартаментного дозволу. [1]

Таке подання дає змогу формально відокремити різні причини відмови. Якщо користувач не належить до департаменту ресурсу, система не повинна маскувати це як помилку ролі. Якщо роль достатня, але MAC-рівень нижчий, причина має бути визначена як MAC_DENIED. Це має значення для аудиту й подальшої підтримки системи. [1]

Модель також враховує операції різної критичності. Для read достатньо перевірити можливість перегляду ресурсу, а для write або manage додатково перевіряється авторство та посадовий рівень автора. Видалення розглядається як найбільш ризикована дія, тому вона доступна тільки користувачам із підвищеними правами. [21]

Формальна логіка може бути описана як ACCESS = SYSTEM_OVERRIDE або (RBAC_ALLOW ∧ MAC_ALLOW ∧ SCOPE_ALLOW ∧ POSITION_ALLOW). Для ресурсу іншого департаменту SCOPE_ALLOW стає істинним лише за наявності approved-запиту міждепартаментного доступу. [1]

У такій моделі кожен елемент політики має окрему відповідальність. RBAC відповідає за функціональне право, MAC - за конфіденційність, департамент - за

область дії, посадова ієрархія - за вертикальні повноваження, а міждепартаментний запит - за контрольований виняток (табл. 2.6). [1]

Таблиця 2.6 - Предикати гібридної політики доступу

Предикат	**Зміст**	**Приклад причини відмови**
SYSTEM_OVERRIDE	main.admin або інший дозволений системний сценарій	system user is hidden або action not allowed
RBAC_ALLOW	роль або системний статус дозволяє дію	role does not allow action
MAC_ALLOW	clearance_level >= required_clearance_level	MAC level is not enough
SCOPE_ALLOW	той самий департамент або approved cross-access	another department
POSITION_ALLOW	посада достатня для зміни ресурсу	insufficient position level

2.8. Модель загроз та відповідність механізмів контролю

У межах проєктування доцільно визначити спрощену модель загроз, на яку реагує прототип. Першою загрозою є спроба звичайного працівника отримати доступ до ресурсу керівництва. Вона блокується через MAC-рівень, required_position_level і авторство ресурсу. [1]

Другою загрозою є спроба начальника одного департаменту працювати з ресурсами іншого департаменту. У системі це блокується департаментною ізоляцією. Навіть якщо посадовий рівень високий, він діє тільки в межах власного департаменту. [27]

Третьою загрозою є помилкове або зловмисне видалення системного користувача. Для цього main.admin, director і deputy.director виділено в system-

департамент і приховано від звичайних користувачів. Крім того, системні акаунти не повинні видалятися стандартними операціями користувацького CRUD. [27]

Четвертою загрозою є обхід frontend-обмежень через ручний API-запит. Відповіддю на неї є дублювання правил на backend-рівні: будь-яке редагування ресурсу або користувача повторно перевіряється сервером незалежно від того, чи була кнопка видима у браузері. [19; 32]

П’ятою загрозою є неконтрольоване міждепартаментне поширення доступу. Для її зменшення міждепартаментний дозвіл прив’язаний до конкретного ресурсу та дії, має статус і може бути відкликаний. Це не створює нову роль і не переносить користувача до чужого департаменту (табл 2.7). [2; 3; 5]

Таблиця 2.7 - Загрози та механізми контролю

Загроза	**Контроль такі правилау системі**	**Компонент реалізації**
Доступ підлеглого до ресурсу керівника	position + author check	access_policy.py
Доступ начальника до чужого відділу	department scope	auth_context.py/resources.py
Редагування system users	прихований system department	users.py/departments.py
Обхід UI через API	server-side validation	FastAPI routers
Надмірний міждепартаментний доступ	approved request per resource/action	cross_department_access.py

2.9. Правила цілісності даних у проєктній моделі

Проєктна модель даних має забезпечувати узгодженість між ролями, департаментами, ресурсами та системними користувачами. Для цього в системі

застосовано кілька правил цілісності, які зменшують ризик появи некоректних записів і суперечливих доступів. [21]

Перше правило полягає в тому, що кожен звичайний користувач повинен мати department. Без цього неможливо застосувати департаментну ізоляцію. Для системних користувачів використовується спеціальний system-департамент, який приховується від звичайної адміністративної логіки. [27]

Друге правило стосується ресурсів: кожен ресурс повинен належати певному департаменту. Це дає змогу визначити власника ресурсу та відокремити внутрішній доступ від міждепартаментного. [27]

Третє правило пов’язане з авторством. created_by_user_id дає змогу не тільки показувати автора, а й визначати право редагування. Без авторства система могла б перевіряти лише роль і департамент, але не могла б відрізнити документ начальника від документа підлеглого. [27]

Четверте правило стосується системних облікових записів. main.admin, director і deputy.director не повинні керуватися як звичайні RBAC-ролі. Їхній статус визначається через username і auth_context.py, що робить їх незалежними від таблиці roles. [2; 3; 5]

П’яте правило пов’язане з міждепартаментними заявками: унікальним має бути не тільки користувач або ресурс, а комбінація user_id, resource_id і action. Це запобігає неконтрольованому розширенню дозволу на інші дії. [2; 3; 5]

Такі правила не є повноцінною схемою бази даних enterprise-рівня, але вони достатні для демонстраційного прототипу й показують, як бізнес-логіка може бути перенесена у структуру даних (табл. 2.8). [27]

Таблиця 2.8 - Правила цілісності даних

Сутність	**Обов’язкове правило**	**Навіщо потрібно**
User	має department і clearance_level	контекст доступу

Продовження таблиці 2.8

Resource	має department і required_clearance_level	MAC та область доступу
Resource author	created_by_user_id	правила редагування
System user	username-based status	захист службових акаунтів
Cross request	user_id + resource_id + action	точковість дозволу

2.10. Пояснюваність рішень і audit-ready підхід

Пояснюваність політики доступу є окремою проєктною вимогою. У багатьох системах користувач бачить лише повідомлення про заборону, але не розуміє її причини. Для адміністратора це створює проблему підтримки, оскільки складно визначити, чи потрібно змінити роль, MAC-рівень, департамент або міждепартаментний дозвіл. [1]

У прототипі кожне рішення супроводжується текстовим reason. Це поле не повинно містити надмірно чутливих технічних деталей, але має бути достатнім для діагностики. Наприклад, повідомлення про недостатній MAC-рівень допомагає відрізнити проблему допуску від проблеми ролі. [1]

Пояснюваність важлива також для демонстрації. Під час захисту дипломного проєкту комісія може побачити не тільки результат true/false, а й логіку, за якою система прийшла до цього результату. Це перетворює прототип із чорної скриньки на навчальну модель контролю доступу. [21]

Audit-ready архітектура означає, що система одразу створюється з урахуванням потреб журналювання. У прототипі audit log не є додатковою функцією після перевірки; він є природним завершенням кожного рішення доступу. [7]

Важливо, що аудит не повинен залежати від успішності операції. Заборонені спроби часто є навіть важливішими для аналізу, ніж дозволені, оскільки саме вони можуть свідчити про помилку налаштувань або потенційну атаку. [7]

Надалі reason і audit log можуть бути використані для побудови звітів, автоматичного виявлення аномалій або формування рекомендацій адміністратору щодо перегляду політик (табл. 2.9). [7]

Таблиця 2.9 - Елементи пояснюваності та аудиту

Елемент	**Призначення**	**Приклад**
reason	пояснення рішення	MAC level is not enough
event_type	тип події	ACCESS_DENIED
actor_username	хто ініціював дію	hr.employee1
details	розширений контекст	resource, action, result
created_at	час події	timestamp

2.11. Бізнес-сценарії ролей у матриці доступу

Щоб гібридна модель була зрозумілою для бізнес-середовища, її доцільно описати не лише через формули, а й через типові ролі організації. Для цього виділено чотири основні сценарії: глобальне адміністрування, керування структурою департаментів, керування власним департаментом і робота звичайного працівника. [21]

Глобальний адміністратор виконує технічну функцію. Він має доступ до всіх модулів, оскільки відповідає за підтримку системи. Проте цей доступ відокремлений від бізнес-ролей, щоб не змішувати технічне адміністрування й управління конкретним відділом. [1; 2; 3; 10; 27]

Директор виконує організаційну функцію. Він може створити або видалити порожній департамент, але це не означає, що він автоматично редагує працівників усередині кожного відділу. Таке обмеження демонструє принцип розподілу обов’язків. [1; 2; 3; 10; 27]

Начальник департаменту виконує локальну адміністративну функцію. Він бачить користувачів і ресурси власного департаменту, однак не отримує доступу до системного департаменту або до ресурсів інших відділів без погодженого workflow. [27]

Заступник начальника виконує проміжну роль. Він може працювати з ресурсами нижчого або свого рівня, але не повинен змінювати документи начальника. Це відображає природну ієрархію відповідальності в організації. [27]

Підлеглий користувач має найменшу область дій. Він може бачити власну панель, доступні ресурси й створювати власні матеріали, але не керує ролями, департаментами або ресурсами керівництва. [27]

Матриця ролей дає можливість швидко перевірити, чи відповідає конкретна дія очікуваній бізнес-логіці. Якщо дія не вкладається в матрицю, її потрібно розглядати як виняток і оформлювати через міждепартаментний або адміністративний процес (табл. 2.10). [1; 2; 3; 10; 27]

Таблиця 2.10 - Матриця бізнес-ролей

Роль	**Основна область**	**Типові дії**	**Заборонені дії**
main.admin	уся система	повний контроль	видалення службового базису
director	структура організації	департаменти	Люди всередині відділів
department_head	власний департамент	Користувачі й ресурси відділу	чужі департаменти
deputy_head	власний департамент	ресурси нижчого рівня	ресурси head
employee	User panel	перегляд і власні ресурси	адміністративні модулі

Висновки до розділу 2

У другому розділі було спроєктовано гібридну модель керування доступом, яка поєднує RBAC, MAC, департаментну ієрархію та механізм погодженого міждепартаментного доступу. Основою моделі є послідовна перевірка умов, за якої доступ дає змогуться лише при позитивному результаті всіх обов’язкових етапів. [1]

Проєктна модель враховує реальні бізнес-вимоги: існування департаментів, керівників, заступників, підлеглих, системних користувачів і директора. Особливу увагу приділено тому, щоб права діяли в межах свого департаменту, а міждепартаментна співпраця відбувалася тільки через формальне погодження. [21]

Отримана архітектура є достатньо формалізованою для реалізації на backend-рівні і водночас зрозумілою для демонстрації через frontend-панелі. Це створює основу для практичного розділу, у якому описується програмна реалізація прототипу. [23]

РОЗДІЛ 3. ПРОГРАМНА РЕАЛІЗАЦІЯ ГІБРИДНОЇ МОДЕЛІКЕРУВАННЯ ДОСТУПОМ RBAC ТА MAC

3.1. Загальна характеристика програмної реалізації

Практична частина реалізована як web-прототип, що складається з FastAPI-backend, SQLite-бази даних, ORM-рівня SQLAlchemy і frontend-інтерфейсу на React. Система імітує роботу організації з кількома департаментами, різними посадовими рівнями та ресурсами різної конфіденційності. [19; 32]

Ключова особливість реалізації полягає в тому, що CRUD-операції не виконуються ізольовано від політики безпеки. Кожна дія над ресурсом проходить комбіновану перевірку: роль, департамент, посада, MAC-рівень, автор ресурсу, системний статус і погоджені міждепартаментні дозволи. [1]

У застосунку передбачено кілька режимів роботи: глобальна адміністративна панель для main.admin/admin, директорський рівень для керування департаментами, панель начальника департаменту та обмежена панель звичайного користувача. [27]

Програмний код структурований так, щоб окремі підсистеми можна було тестувати й розвивати незалежно. Backend містить router-и для автентифікації, користувачів, ролей, ресурсів, департаментів, перевірки доступу, аудиту, dashboard і міждепартаментних запитів. [7]

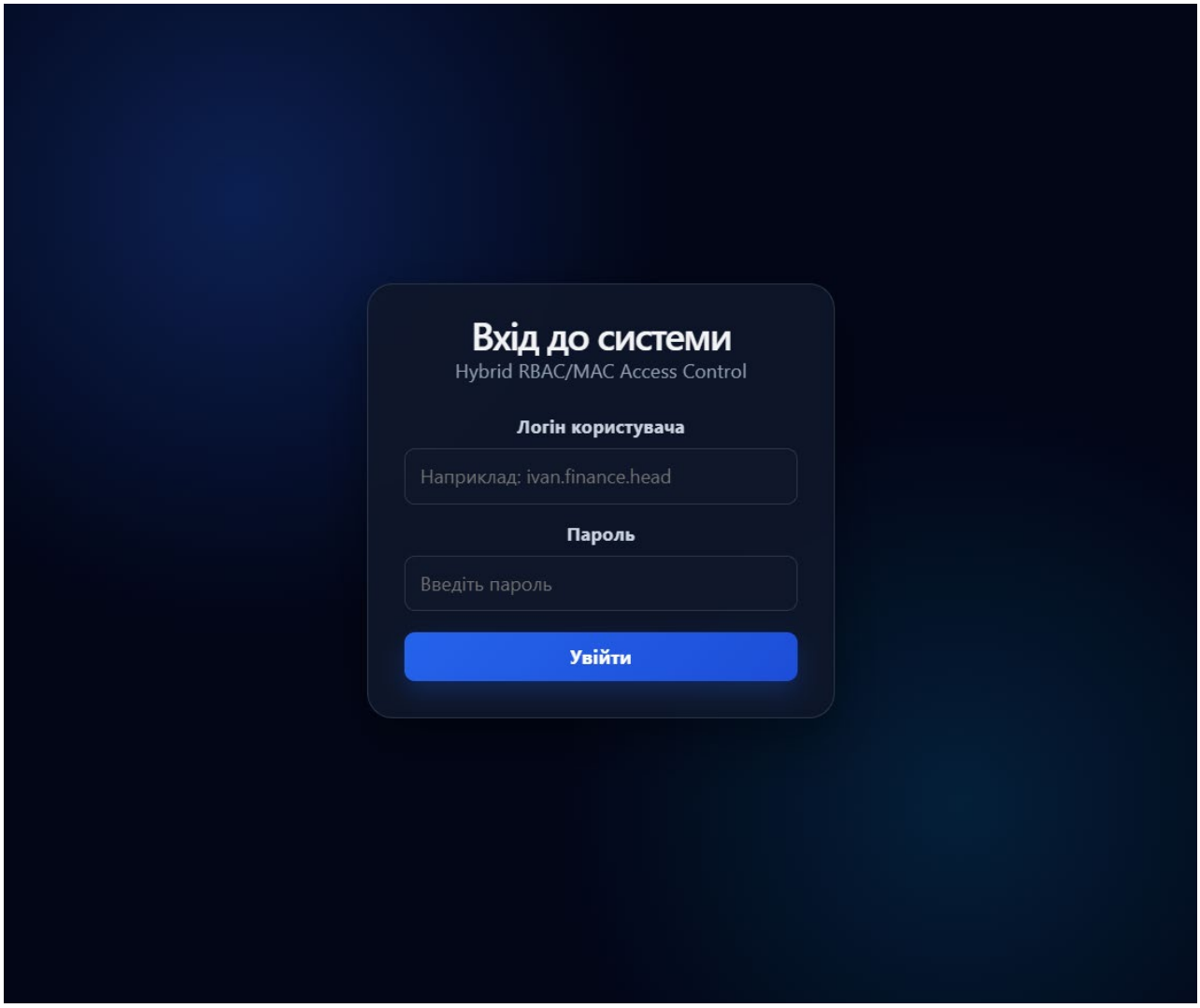


Рисунок 3.1 - Сторінка аутентифікації користувача

3.2. Вибір інструментів та технологій розробки

Для backend-частини використано Python і FastAPI. FastAPI обрано тому, що він підтримує типізовані запити, автоматичну OpenAPI-документацію та інтерактивний Swagger UI, що спрощує тестування endpoint-ів [19]. Для запуску застосунку використовується Uvicorn як ASGI-сервер [20].

Для роботи з базою даних використано SQLAlchemy. ORM-підхід дозволив описати таблиці у вигляді Python-класів і виконувати запити на рівні об’єктної моделі, що спрощує підтримку зв’язків між користувачами, ролями, ресурсами та журналом аудиту [21].

SQLite обрано як базу даних прототипу через її самодостатність, відсутність потреби в окремому сервері та простоту перенесення файлу бази даних [22]. Для дипломної демонстрації це доцільно, оскільки система може запускатися локально без складного адміністрування.

Frontend реалізовано на React. Компонентний підхід React дає змогу розділити інтерфейс на LoginScreen, AdminPanel, UserPanel, UsersSection, ResourcesSection, DepartmentsSection, AccessCheckSection та інші логічні частини [23]. Це зробило застосунок більш керованим та придатним до поступового розширення.

Pydantic використовується для валідації вхідних і вихідних структур API. Завдяки цьому backend може відхиляти некоректні запити до виконання бізнес-логіки, а також підтримувати зрозумілі схеми даних (табл 3.1).[24]

Таблиця 3.1 - Програмні засоби реалізації

Компонент	**Інструмент**	**Призначення**
Мова backend	Python	опис логіки доступу та API
Framework	FastAPI	REST API і Swagger UI
ASGI server	Uvicorn	локальний запуск backend
ORM	SQLAlchemy	моделі та робота з БД
Database	SQLite	локальне зберігання даних
Frontend	React	графічні панелі
Validation	Pydantic	перевірка схем запитів
Version control	Git/GitHub	збереження історії змін

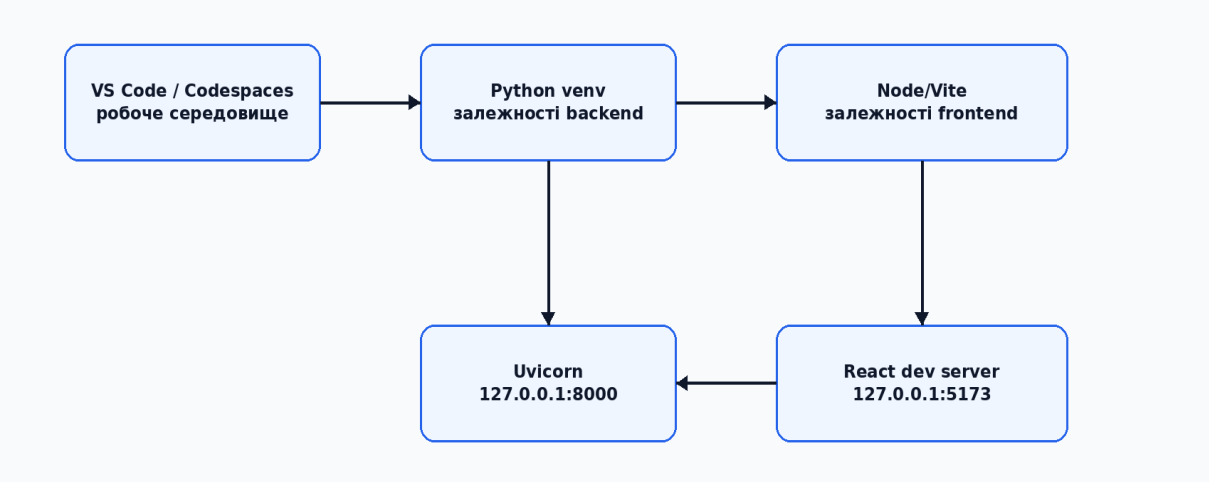


Рисунок 3.2 - Локальне розгортання backend/frontend

3.3. Архітектура програмного прототипу

База даних SQLite використовується як Policy Storage. У ній зберігаються користувачі, ролі, департаменти, ресурси, audit logs і заявки на міждепартаментний доступ. Завдяки цьому зміни в даних одразу впливають на правила доступу без переписування програмного коду. [7]

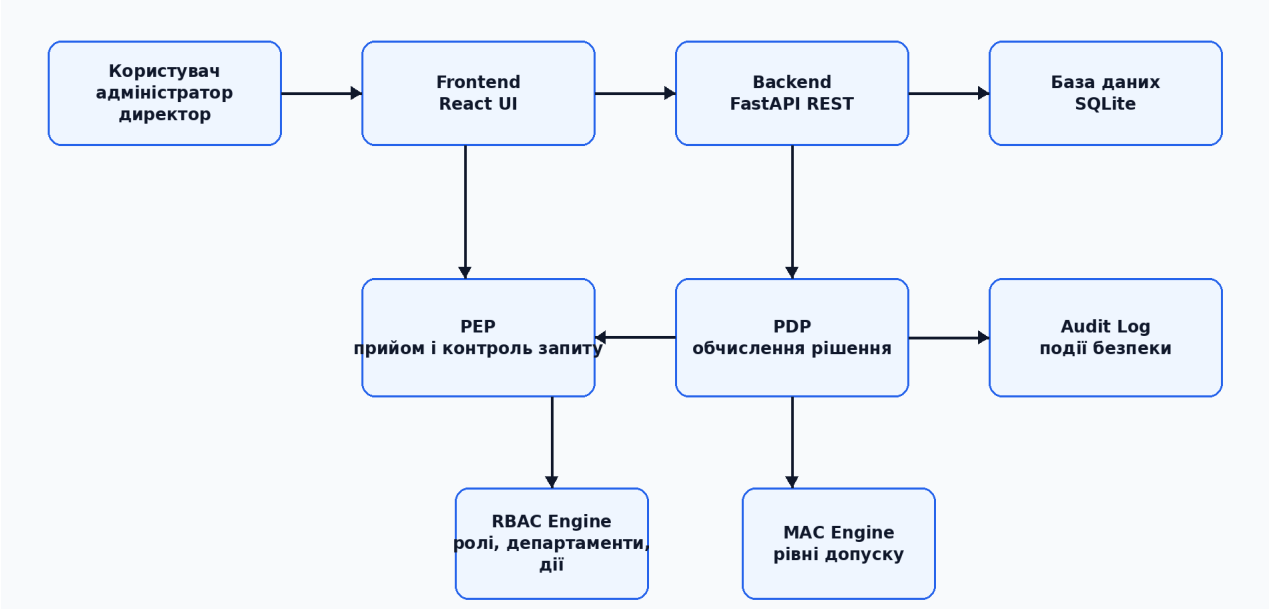


Рисунок 3.3 - Frontend-структура застосунку

Збереження стану користувача виконано через дані, отримані після login. Frontend передає X-User-Id у кожному запиті, що дає змогу backend визначити

актора. У виробничому середовищі такий механізм доцільно замінити на JWT або OAuth2, однак для дипломного прототипу він забезпечує прозорість перевірок і простоту демонстрації. [23]

На frontend-рівні застосунок поділено на окремі панелі: LoginScreen, AdminPanel, UserPanel, а також секції UsersSection, ResourcesSection, DepartmentsSection, AccessCheckSection і CrossDepartmentAccessSection. Видимість кнопок та форм залежить від прав користувача, але кожна критична дія додатково підтверджується backend-перевіркою. [23]

На backend-рівні основними компонентами є FastAPI router-и, SQLAlchemy-моделі, Pydantic-схеми, модуль auth_context.py і модуль access_policy.py. Router-и приймають запити, перевіряють поточного актора, звертаються до бази даних і передають контекст до політики доступу. Модуль access_policy.py акумулює правила RBAC/MAC, департаментної ізоляції, посадової ієрархії та міждепартаментного дозволу. [1]

Програмний прототип реалізовано як набір взаємопов’язаних backend- та frontend-модулів. Backend виконує роль централізованої точки прийняття й застосування рішень, а frontend забезпечує зручне подання цих рішень для різних категорій користувачів. Такий поділ важливий для безпеки, оскільки графічний інтерфейс не розглядається як джерело довіри(табл. 3.2). [23]

Таблиця 3.2 - Основні backend-модулі

Файл / модуль	**Призначення**
main.py	ініціалізація FastAPI та підключення router-ів
database.py	налаштування SQLAlchemy engine і SessionLocal
models.py	опис таблиць бази даних
schemas.py	Pydantic-схеми API

Продовження таблиці 3.2

security.py	хешування та перевірка паролів
auth_context.py	визначення поточного актора та ролей
access_policy.py	основна логіка RBAC/MAC
routers/*.py	REST API для підсистем

3.4. Структура бази даних

База даних має реляційну структуру та відображає основні сутності системи керування доступом. Центральною є таблиця users, оскільки саме користувач виступає суб’єктом запиту. Для нього зберігаються username, password_hash, clearance_level, department і department_position. [22]

Таблиця roles відповідає за рольову частину моделі. Оскільки один користувач може мати кілька ролей, а одна роль може бути призначена багатьом користувачам, зв’язок реалізовано через проміжну таблицю user_roles. Службові статуси main.admin і director винесені за межі звичайних ролей.

У таблиці resources зберігаються захищені об’єкти системи. Для кожного ресурсу визначено департамент-власник, required_clearance_level, required_position_level, description, content і created_by_user_id. Такий набір полів дає змогу перевіряти не тільки право читання, а й право редагування з урахуванням автора.

Таблиця cross_department_access_requests описує процес міждепартаментного погодження. Вона фіксує ініціатора, цільового керівника, користувача, ресурс, дію, статус і коментар до рішення.

Таблиця audit_logs використовується для журналювання результатів перевірок і адміністративних дій. У ній зберігаються event_type, actor_username, target_username, department, action, details і created_at.

Таблиця roles відповідає за рольову частину моделі. Оскільки один користувач може мати кілька ролей, а одна роль може бути призначена багатьом

користувачам, зв’язок реалізовано через проміжну таблицю user_roles. Службові статуси main.admin і director винесені за межі звичайних ролей.

Таблиця cross_department_access_requests описує процес міждепартаментного погодження. Вона фіксує ініціатора, цільового керівника, користувача, ресурс, дію, статус і коментар до рішення.

Таблиця audit_logs використовується для журналювання результатів перевірок і адміністративних дій. У ній зберігаються event_type, actor_username, target_username, department, action, details і created_at (табл. 3.3).

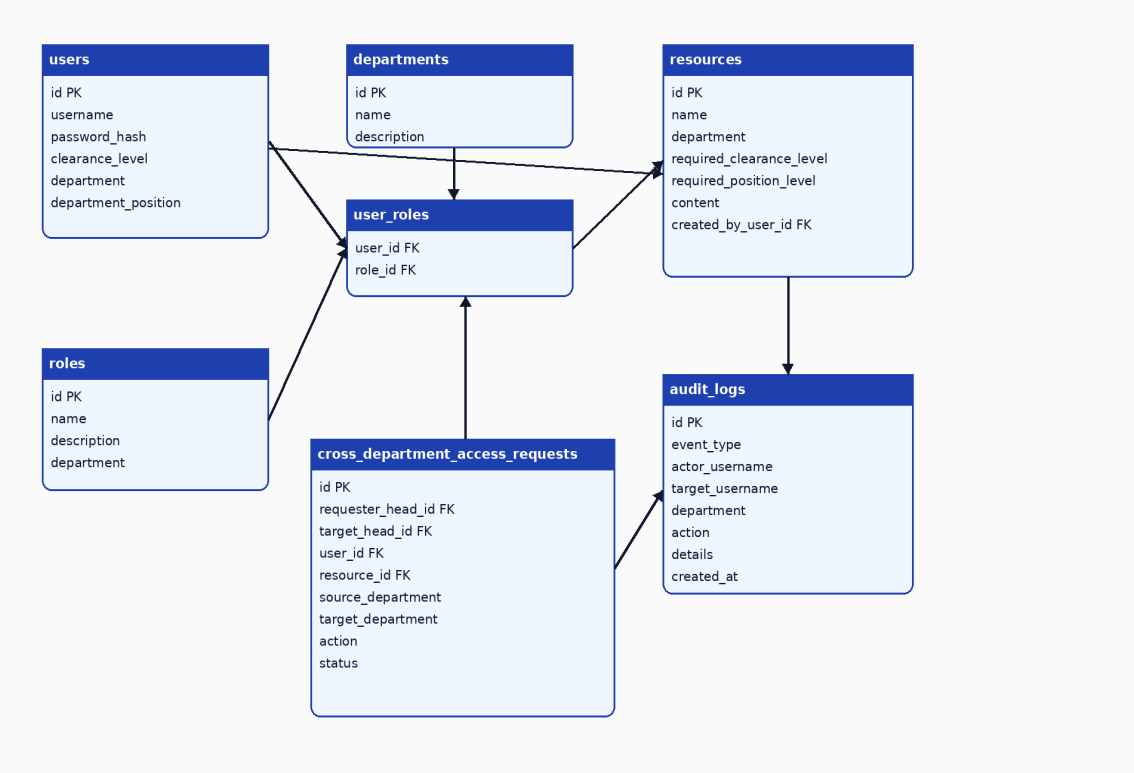


Рисунок 3.4 - ER-діаграма основних сутностей бази даних

Таблиця 3.3 - Основні таблиці бази даних

Таблиця	**Призначення**	**Ключові поля**
users	користувачі системи	id, username, clearance_level, department

Продовження таблиці 3.3

roles	RBAC-ролі	id, name, department
user_roles	зв’язок користувачів і ролей	user_id, role_id
departments	структурні підрозділи	id, name, description
resources	захищені ресурси	id, name, content, created_by_user_id
audit_logs	журнал подій	event_type, actor_username, action
cross_department_access_requests	міждепартаментні запити	user_id, resource_id, status

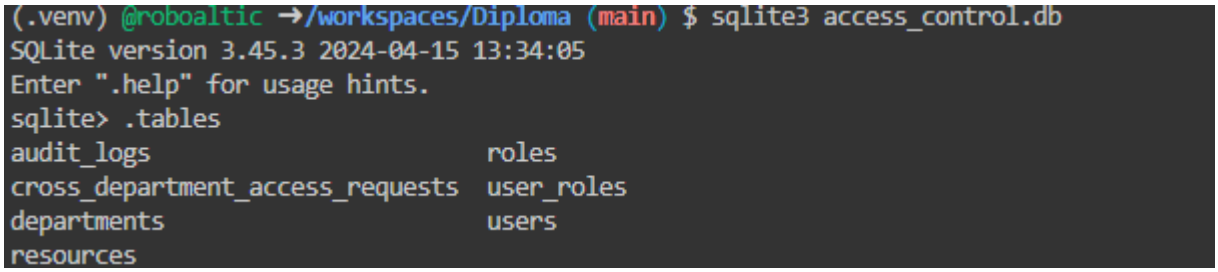


Рисунок 3.5 - Перегляд таблиць бази даних

3.5. Реалізація автентифікації та системних користувачів

Автентифікація реалізована через endpoint POST /auth/login. Користувач передає username і password, backend знаходить відповідний запис у таблиці users і перевіряє password_hash. Якщо перевірка успішна, frontend отримує дані користувача, його ролі, департамент, MAC-рівень і службові прапорці. [1]

Системний користувач main.admin виконує роль резервного повного адміністратора. Його права не залежать від таблиці roles. Це рішення важливе для відновлення системи у випадку пошкодження або видалення звичайних ролей. [27]

Користувач director відокремлений від global admin. Він може керувати департаментами, але не повинен автоматично редагувати користувачів усередині них. Таке розділення відповідає принципу мінімальних привілеїв: системна функція директора обмежена структурою організації. [27]

deputy.director також належить до system-департаменту, який приховується від звичайних користувачів і начальників департаментів. Це запобігає випадковому редагуванню або видаленню службових користувачів BAC-модуль у практичній реалізації не обмежується абстрактним описом ролей (табл. 3.4). [27]

Таблиця 3.4 - Системні користувачі

Обліковий запис	Призначення	Обмеження
main.admin	повний системний адміністратор	не видаляється і приховується
director	керування департаментами	не редагує людей усередині відділів
deputy.director	службовий рівень директора	обмежене системне представлення

3.6. Реалізація RBAC-модуля

Усі RBAC-рішення обмежуються областю департаменту. Роль начальника HR не дає автоматичного доступу до finance, lawyer або engineer. У разі потреби доступ до чужого ресурсу має проходити через workflow міждепартаментного делегування. [2; 3; 5]

Для department_head, deputy_head і employee RBAC-перевірка доповнюється посадовим рівнем. Це дає змогу відобразити не тільки факт наявності ролі, а й вертикальну ієрархію всередині департаменту. Начальник має повноваження над

усім своїм відділом, заступник - над ресурсами свого та нижчого рівня, а підлеглий - лише над власними або рівнозначними ресурсами. [2; 3; 5]

Окремо виділено системні повноваження, які не зберігаються як звичайні записи таблиці roles. До них належать main.admin, director і deputy.director. Такий підхід унеможливлює втрату повного контролю над системою внаслідок випадкового видалення RBAC-ролі. [2; 3; 5]

RBAC-модуль у практичній реалізації не обмежується абстрактним описом ролей, а працює з конкретними користувачами, департаментами й діями. Роль користувача зберігається через таблицю roles та проміжний зв'язок user_roles, що дає змогу одному користувачу мати кілька функціональних повноважень (табл .3.5). [2; 3; 5]

Таблиця 3.5 - Рольові правила в реалізації

Категорія	Дозволені дії	Коментар
admin/main.admin	усі дії	повний контроль системи
director	департаменти	створення/видалення порожніх департаментів
department_head	користувачі та ресурси свого департаменту	не бачить system users
deputy_head	ресурси свого й нижчого рівня	не редагує ресурси head
employee	перегляд і створення власних ресурсів	обмежена користувацька панель

3.7. Реалізація MAC-модуля

Особливо важливо, що міждепартаментне погодження не скасовує MAC-перевірку. Якщо начальник цільового департаменту схвалив запит, але користувач

не має достатнього рівня допуску, доступ усе одно залишається забороненим. Саме це забезпечує пріоритет примусової політики над бізнес-гнучкістю. [1]

У прототипі MAC-рівні мають значення від 1 до 5: Public, Internal, Confidential, Secret і Top Secret. Така шкала є достатньою для демонстрації бізнес-сценаріїв і може бути розширена у випадку переходу до складнішої політики класифікації. [1]

На відміну від RBAC, MAC не залежить від організаційної ролі користувача. Навіть начальник департаменту не може отримати доступ до ресурсу з вищим required_clearance_level, якщо його власний clearance_level недостатній. Це демонструє примусовий характер MAC-політики. [1]

MAC-модуль реалізовано через числові рівні допуску користувача й рівні захищеності ресурсу. У таблиці users зберігається clearance_level, а в таблиці resources - required_clearance_level. Під час кожної перевірки система порівнює ці значення й відхиляє запит, якщо рівень користувача нижчий за рівень ресурсу (табл. 3.6). [1]

Таблиця 3.6 - Приклади MAC-рішень

Користувач	**Рівень користувача**	**Ресурс**	**Рівень ресурсу**	**Результат**
hr.employee1	2	hr_internal_tasks	2	ALLOW
hr.employee1	2	hr_confidential_strategy	4	DENY
hr.head	5	hr_confidential_strategy	4	ALLOW
finance.deputy	3	finance_management_report	3	ALLOW

3.8. Реалізація PDP, PEP та endpoint /access/check

Алгоритм обробки запиту є ключовою частиною гібридної моделі, оскільки саме він визначає порядок перевірок. У системі не використовується підхід, за якого всі умови перевіряються хаотично. Навпаки, рішення формується

послідовно: спочатку визначається поточний користувач, далі перевіряється область доступу, потім рольова логіка, MAC-рівень і додаткові обмеження. [1]

Першим етапом є автентифікація та ідентифікація поточного актора. У демонстраційному прототипі після login frontend зберігає id користувача й передає його в заголовку X-User-Id. Такий спосіб є спрощеним для дипломного прототипу, однак він дає змогу чітко показати, від імені якого користувача виконується кожен API-запит. [19; 32]

Другим етапом є визначення ресурсу та дії. Дія може мати значення read, write, manage або delete. Для кожної дії система перевіряє, чи є вона допустимою для ролі та посадового рівня користувача. Наприклад, employee може читати і створювати власні ресурси, але не може редагувати ресурс, створений начальником департаменту. [27]

Третім етапом є MAC-перевірка. Вона не залежить від того, наскільки роль користувача висока. Якщо рівень допуску користувача нижчий за рівень ресурсу, доступ блокується. Це дає змогу запобігти ситуації, коли організаційна роль випадково відкриває критичні документи. [1]

Четвертим етапом є аудит. Незалежно від результату система формує запис, у якому фіксуються користувач, дія, ресурс, результат і причина. Це створює основу для подальшого аналізу інцидентів і підтвердження коректності політик доступу (табл 3.7). [7]

Таблиця 3.7 - Endpoint перевірки доступу

Параметр	Опис	Приклад
user_id	користувач, для якого перевіряється доступ	5
resource_id	ресурс, до якого виконується дія	2
action	тип операції	read/write/manage
access_granted	результат відповіді	true/false
reason	пояснення рішення	Access denied: MAC level

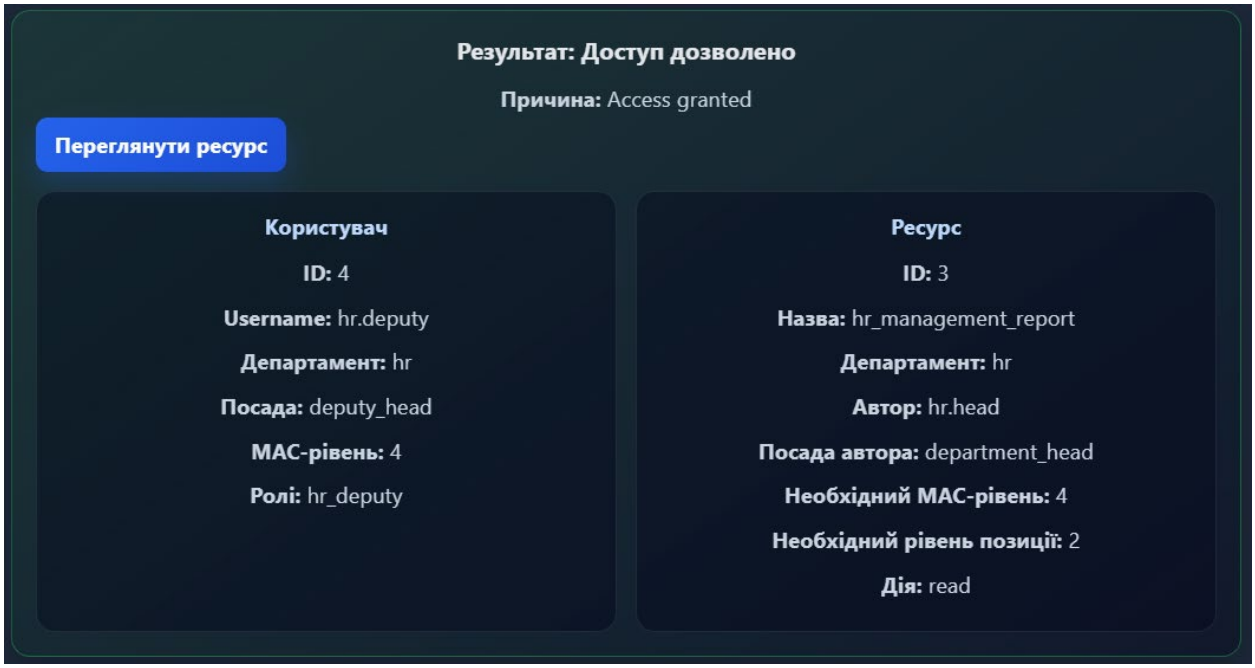


Рисунок 3.6 - Результат дозволеної перевірки доступу

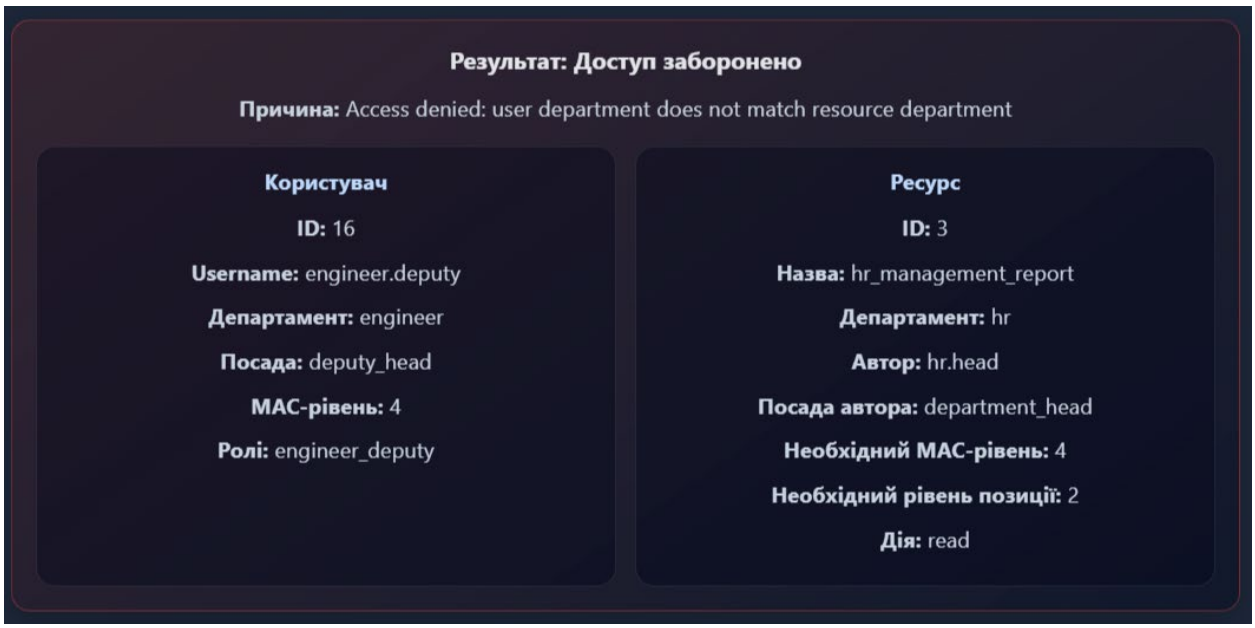


Рисунок 3.7 - Результат забороненої перевірки доступу

3.9. Реалізація керування користувачами

Модуль користувачів забезпечує створення, перегляд, редагування, видалення та скидання паролів. Під час створення користувача задаються username, password, department, department_position, clearance_level і role_ids. Для начальника департаменту створення обмежується власним департаментом. [27]

Frontend-частина реалізує створення користувача через модальне вікно. Це покращує зручність інтерфейсу, оскільки форма не займає постійне місце на сторінці. Після збереження список користувачів оновлюється через API. [19; 32]

Окремо варто зазначити приховування системних користувачів від звичайних ролей і department_head. main.admin, director і deputy.director повинні бути видимі тільки системним користувачам, оскільки їх випадкове редагування порушило б цілісність політики доступу. [27]

Редагування користувача також обмежується ієрархією. Начальник може змінювати підлеглих у своєму департаменті, але директор не отримує право змінювати співробітників конкретного відділу. Це дає змогу розділити управління структурою організації і управління персоналом департаменту (табл. 3.8). [27]

Таблиця 3.8 - Основні операції з користувачами

Операція	**Endpoint**	**Обмеження**
Список	GET /users/	фільтрація за роллю/департаментом
Створення	POST /users/	department_head створює тільки у своєму відділі
Редагування	PUT /users/{id}	за ієрархією та правами
Видалення	DELETE /users/{id}	system users не видаляються
Скидання пароля	POST /users/{id}/reset-password	для адміністраторів із правами

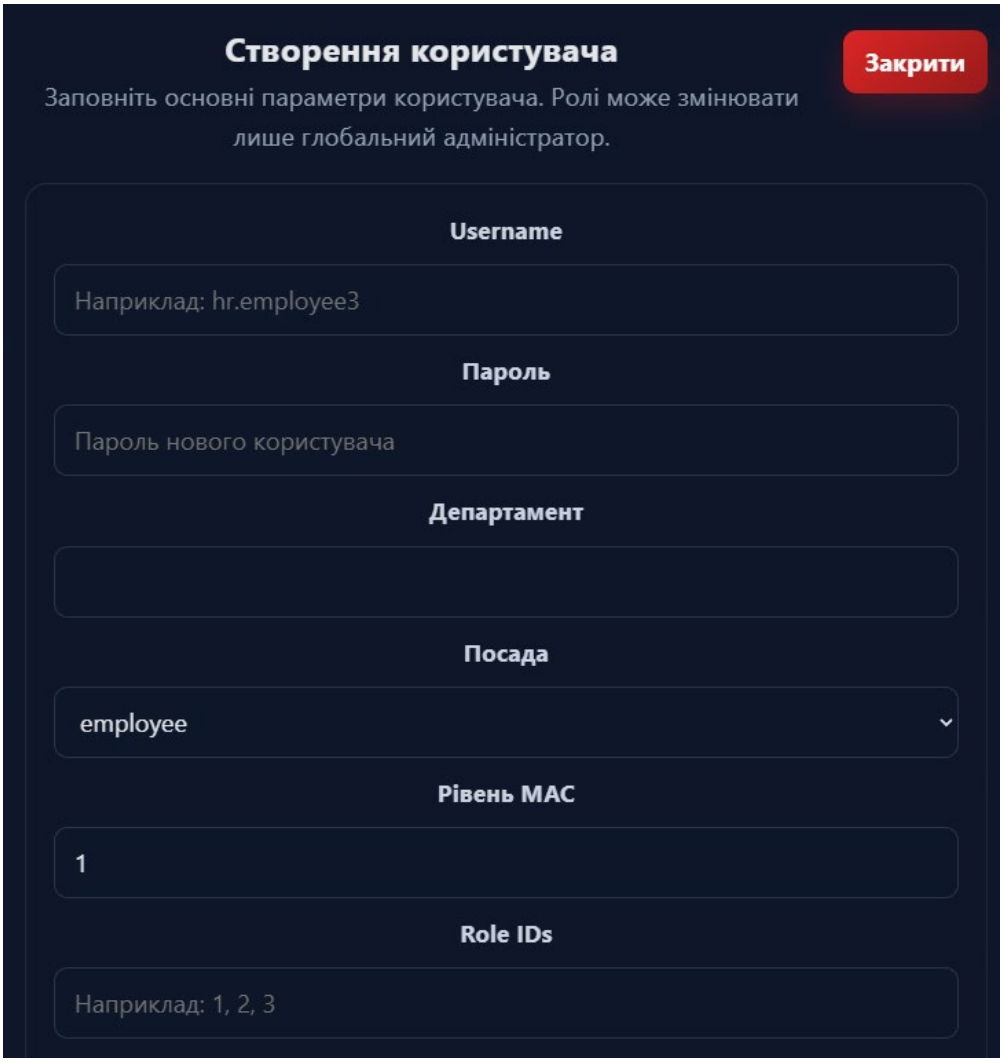


Рисунок 3.8 - Модальне вікно створення користувача

3.10. Реалізація керування ресурсами

Ресурс у системі розглядається не тільки як запис для перевірки доступу, а як об’єкт із власним вмістом. Для цього додано поле content: після успішної перевірки користувач може відкрити ресурс і переглянути його наповнення. [27]

Кожен ресурс пов’язаний з автором через created_by_user_id. У графічному інтерфейсі замість числового ідентифікатора показуються created_by_username і created_by_position, що робить інформацію зрозумілішою для адміністратора та користувача. [23]

Право редагування ресурсу визначається не тільки департаментом, а й рівнем автора. Employee може редагувати ресурси employee-рівня у своєму департаменті,

deputy_head - ресурси employee і deputy, а department_head - усі ресурси свого департаменту. main.admin може редагувати все. [27]

Кнопки редагування й видалення відображаються у frontend лише за наявності відповідних прав (табл. 3.9). Однак це лише зручність інтерфейсу: backend повторно перевіряє кожну критичну дію, щоб користувач не міг обійти політику через ручний API-запит. [19; 32]

Таблиця 3.9 - Поля ресурсу

Поле	Зміст	Використання у доступі
name	назва ресурсу	ідентифікація
department	департамент-власник	департаментна ізоляція
required_clearance_level	MAC-рівень ресурсу	MAC-перевірка
required_position_level	мінімальний посадовий рівень	ієрархія
content	внутрішній зміст	показ після дозволу
created_by_user_id	автор ресурсу	право редагування

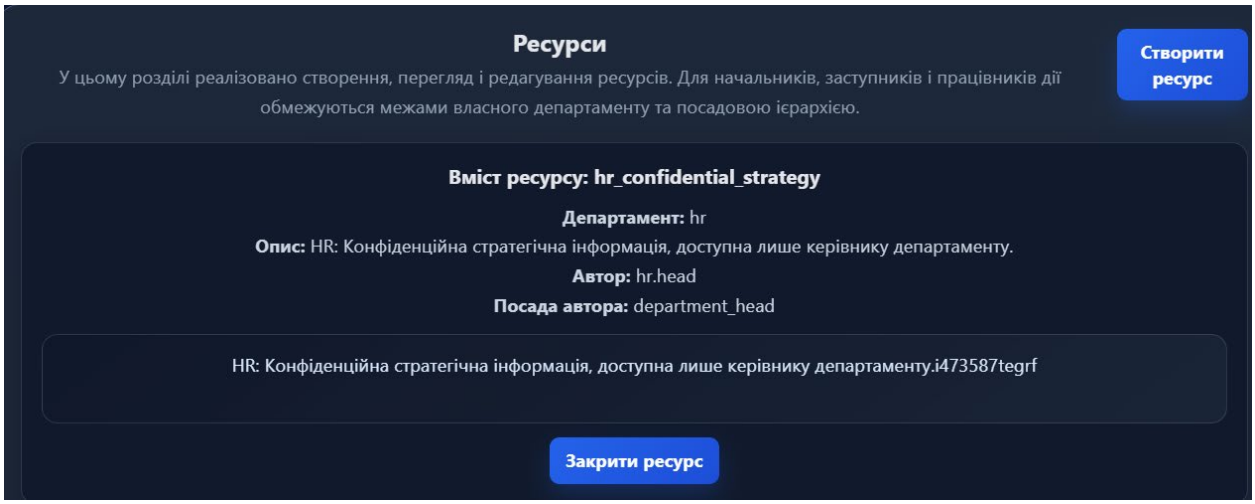


Рисунок 3.9 - Перегляд вмісту ресурсу після дозволу

3.11. Реалізація департаментів і дерева департаментів

Департаментна логіка реалізована через таблицю departments і поле department у користувачів та ресурсів. Для глобального адміністратора відображаються всі департаменти, для начальника - лише власний департамент, а system-департамент приховується від усіх, крім системних користувачів. [27]

Окремий endpoint /departments/tree повертає структуру департаменту з користувачами та ресурсами. Frontend відображає її у вигляді дерева/accordion: кожен департамент можна розгорнути, після чого показуються користувачі, ресурси, автори і доступні дії. [19; 32]

Для директора в дереві департаментів доступні дії створення, редагування й видалення департаменту. Однак кнопки редагування людей і ресурсів для директора не показуються, оскільки це не його зона відповідальності. [27]

Начальник департаменту бачить кнопки лише в межах свого відділу (табл. 3.10). Заступник і підлеглий не повинні бачити адміністративні дії, які не відповідають їхнім повноваженням. [10; 21; 27]

Таблиця 3.10 - Видимість департаментів

Користувач	**Що бачить**	**Що може змінювати**
main.admin	усі департаменти включно з system	усе
director	департаменти, включно із системним контекстом	структуру департаментів
hr.head	лише HR-відділ	користувачів і ресурси HR
employee	свою user panel	власні ресурси

3.12. Реалізація міждепартаментного доступу

Frontend-вкладка міждепартаментного доступу передбачена як окрема секція. Вона має містити три блоки: створення запиту, вхідні запити та активні дозволи.

Такий поділ робить процес зрозумілим для начальників департаментів і зменшує ризик помилкової видачі прав. [23]

Для припинення раніше наданого дозволу передбачено статус revoked. Це має значення для бізнес-процесів, де міждепартаментна співпраця має бути тимчасовою або обмеженою за задачею. [10; 12; 28; 27]

Після створення заявка отримує статус pending. Начальник цільового департаменту бачить її у вхідних запитах і може виконати approve або reject. У разі approve дозвіл стає активним, але не відкриває весь департамент: він прив’язаний до user_id, resource_id і action. [10; 12; 28; 27]

Запит створює начальник департаменту-ініціатора. Під час створення backend перевіряє, що користувач належить до департаменту ініціатора, ресурс належить іншому департаменту, дія має допустиме значення read або write, а MAC-рівень користувача в принципі може відповідати рівню ресурсу. [1]

Міждепартаментний доступ реалізовано окремим backend-модулем cross_department_access.py(табл. 3.11). Він не замінює основну політику доступу, а додає до неї контрольований виняток для конкретного користувача, конкретного ресурсу та конкретної дії. [2; 3; 5]

Таблиця 3.11 - Endpoint-и міждепартаментного доступу

Endpoint	**Призначення**
POST /cross-department-access/	створити запит
GET /cross-department-access/incoming	вхідні запити для цільового департаменту
GET /cross-department-access/outgoing	вихідні запити
GET /cross-department-access/active	активні погоджені дозволи
POST /{id}/approve	погодити запит
POST /{id}/reject	відхилити запит
POST /{id}/revoke	відкликати дозвіл

3.13. Реалізація frontend-частини

Frontend реалізовано як набір React-компонентів. Основний компонент App.jsx відповідає за стан поточного користувача, вибір активної панелі, завантаження dashboard-даних і передачу props дочірнім компонентам. [23]

Для адміністратора використовується AdminPanel. У ньому розміщено статистичні картки, меню розділів і секції керування користувачами, ролями, ресурсами, департаментами, перевіркою доступу та аудитом. Для department_head заголовок змінюється на «Панель департаменту», а статистика фільтрується за його відділом. [7]

UserPanel призначена для звичайного користувача. Вона показує профіль, департамент, доступні ресурси та недоступні ресурси з поясненням причин. Перегляд ресурсу відбувається тільки після повторного запиту /access/check. [27]

Дизайн оформлено у темних кольорах із картками(табл. 3.12),

плавними hover-ефектами, модальними вікнами та адаптивними таблицями. Це робить демонстрацію системи зручнішою і візуально зрозумілою. [23; 31; 27]

Таблиця 3.12 - Frontend-компоненти

Компонент	**Призначення**
LoginScreen	вхід у систему
AdminPanel	глобальна панель
UserPanel	панель звичайного користувача
UsersSection	користувачі
ResourcesSection	ресурси
DepartmentsSection	дерево департаментів
AccessCheckSection	перевірка доступу
CrossDepartmentAccessSection	міждепартаментні запити

3.14. Підсистема аудиту та журналювання

Підсистема аудиту фіксує спроби доступу та адміністративні дії. Вона необхідна не тільки для безпеки, а й для пояснюваності роботи системи. Якщо доступ заборонено, адміністратор може побачити, хто виконав запит, до якого ресурсу, яку дію та з якою причиною система відмовила. [7]

NIST SP 800-53 містить окремі групи контролів, пов'язані з аудитом, обліком подій і керуванням доступом [7]. У дипломному прототипі ці принципи реалізовано спрощено, але достатньо для демонстрації: кожна перевірка через /access/check записує ACCESS_GRANTED або ACCESS_DENIED.

Надалі підсистему аудиту можна розширити: додати фільтри за датою, експорт у CSV, збереження IP-адреси, user-agent, типу сесії, а також сповіщення про підозрілі дії. Проте навіть поточна реалізація забезпечує базову трасованість рішень доступу(табл. 3.13). [7]

Таблиця 3.13 - Структура audit log

Поле	Призначення
event_type	тип події
actor_username	хто виконав дію
target_username	над ким або для кого дія
department	контекст департаменту
action	read/write/manage/delete
details	причина й деталі
created_at	час події

3.15. Скрипти ініціалізації та міграції

Під час розробки використовувалися окремі скрипти для створення системних користувачів, заповнення департаментів, ролей, ресурсів та виконання

міграцій. Це дозволило швидко відновлювати демонстраційну базу після видалення access_control.db або зміни структури таблиць. [27]

Скрипт seed_system_users.py створює main.admin, director і deputy.director, а також переносить їх у прихований system-департамент. Це потрібно для стабільності системи та приховування службових облікових записів від начальників департаментів. [27]

Скрипти seed_department_demo.py і seed_department_resources.py створюють набір користувачів і ресурсів для HR, finance, lawyer та engineer. Завдяки цьому можна швидко демонструвати сценарії доступу без ручного введення великої кількості даних. [27]

Міграційні скрипти використовувалися для додавання content, created_by_user_id і таблиці cross_department_access_requests без повного видалення бази. Це демонструє поступовий розвиток схеми даних у процесі проєктування(табл. 3.14). [27]

Таблиця 3.14 - Скрипти проєкту

Скрипт	Призначення
seed_system_users.py	створення main.admin/director/deputy.director
seed_department_demo.py	демо-користувачі департаментів
seed_department_resources.py	ресурси з content
migrate_add_resource_author_content.py	додавання автора і вмісту ресурсу
migrate_cross_department_access.py	таблиця міждепартаментних запитів

3.16. Запуск і перевірка роботи системи

Робоче середовище створюється через python -m venv.venv. Офіційна документація Python описує venv як механізм створення легких ізольованих

середовищ із власними site-packages [25]. Це має значення, оскільки залежності дипломного проєкту не змішуються з глобальними пакетами системи.

Після активації середовища встановлюються залежності backend: fastapi, uvicorn, sqlalchemy, pydantic, python-dotenv, pytest, httpx та бібліотеки для хешування паролів. Далі backend запускається командою python -m uvicorn app.main:app --reload. [19; 32]

Frontend запускається окремо через npm run dev. У режимі розробки React-застосунок доступний на 127.0.0.1:5173, а backend - на 127.0.0.1:8000. Взаємодія між ними відбувається через REST API. [19; 32]

Для тестування використовувалися Swagger UI, curl із jq та ручні сценарії через frontend(табл. 3.15)(рис. 3.11). Така комбінація дає змогу перевірити як низькорівневу відповідь API, так і зручність роботи з графічним інтерфейсом. [19; 32]

Таблиця 3.15 - Команди запуску

Етап	**Команда**
Створення venv	python -m venv.venv
Активація	source.venv/bin/activate
Встановлення backend	pip install -r requirements.txt
Запуск backend	python -m uvicorn app.main:app --reload
Запуск frontend	npm run dev -- --host 0.0.0.0

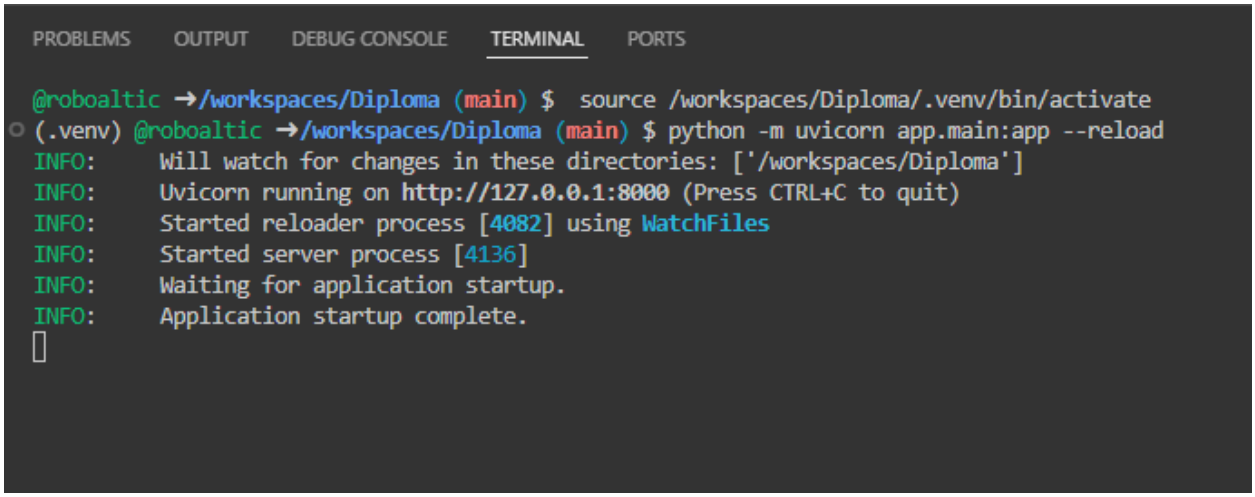


Рисунок 3.10 - Запуск backend у терміналі

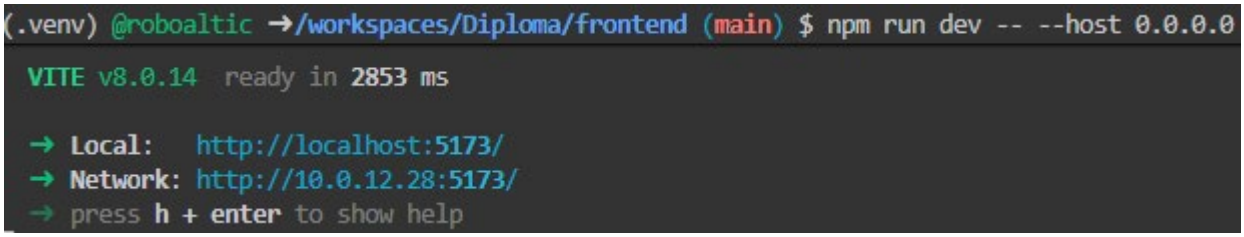


Рисунок 3.11 - Запуск frontend у терміналі

3.17. Тестування системи та демонстраційні сценарії

Тестування системи виконувалося через набір типових бізнес-сценаріїв. Перший сценарій - вхід під main.admin і перевірка повного контролю. Другий - вхід під director і керування департаментами. Третій - вхід під hr.head і перевірка, що видно тільки HR-відділ. Четвертий - робота hr.employee1 у user panel. [27]

Окремо тестувалися ситуації відмови. Наприклад, employee з рівнем MAC 2 не повинен отримувати доступ до ресурсу рівня 4. Також employee не повинен редагувати ресурс, створений department_head. Такі тести підтверджують, що система враховує не лише роль, а й MAC-рівень і авторство ресурсу. [1]

Міждепартаментний сценарій перевіряється так: hr.head створює запит для hr.employee1 на ресурс finance; finance.head погоджує його; після цього employee отримує доступ лише до конкретного ресурсу й лише для дозволеної дії. Якщо MAC-рівень недостатній, доступ усе одно блокується. [1]

Під час тестування важливо перевіряти не тільки наявність кнопок, а й backend-відповідь. Це потрібно тому, що UI може приховувати або показувати кнопки, але реальна безпека повинна забезпечуватися серверною логікою (табл. 3.16). [10; 11; 27]

Таблиця 3.16 - Демонстраційні тестові сценарії

Сценарій	**Кроки**	**Очікуваний результат**
main.admin	login -> users/resources/roles	видно всю систему
director	login -> departments -> create/delete	керує департаментами
hr.head	login -> panel department	видно тільки HR
hr.employee1	login -> user panel -> resources	видно доступні ресурси
MAC deny	employee -> high resource	відмова через MAC
Cross access	request -> approve -> check	обмежений доступ після погодження

3.18. Результати практичної реалізації

У результаті практичної частини створено робочий прототип гібридної системи керування доступом. Система підтримує користувачів, ролі, департаменти, ресурси, MAC-рівні, авторство ресурсів, журнал аудиту, системних користувачів і міждепартаментні запити доступу. [1]

Практична реалізація підтвердила, що поєднання RBAC і MAC є доцільним для бізнес-середовища. RBAC забезпечує зручне адміністрування відповідно до посад і департаментів, а MAC не дає змогу ролям відкривати ресурси з вищим рівнем конфіденційності. [1]

Розроблений frontend дає змогу демонструвати систему без потрібності працювати лише через curl або Swagger. Це має значення для захисту дипломної роботи, оскільки комісія може побачити не тільки код, а й завершений графічний інтерфейс. [19; 32]

Наявність міждепартаментного workflow підсилює практичну цінність проєкту. Він показує, що система враховує не лише статичні правила, а й реальні потреби співпраці між відділами, не руйнуючи при цьому модель безпеки(табл. 3.17). [21]

Таблиця 3.17 - Досягнуті результати

Результат	**Стан**
Backend API	реалізовано
SQLite база даних	реалізовано
RBAC/MAC policy	реалізовано
Адмін-панель	реалізовано
Користувацька панель	реалізовано
Департаментна ієрархія	реалізовано
Міждепартаментні запити	реалізовано
Аудит	реалізовано

Висновки до розділу 3

У третьому розділі було реалізовано програмний прототип гібридної системи керування доступом RBAC/MAC. Реалізація підтвердила можливість поєднання рольової логіки, MAC-рівнів, департаментної ізоляції та бізнес-сценаріїв міждепартаментної співпраці в одному застосунку. [1]

Backend-частина на FastAPI забезпечує централізовану перевірку доступу, роботу з базою даних, аудит і API для frontend. Frontend-частина на React надає

окремі панелі для main.admin, director, department_head і звичайного користувача. [7]

Отриманий прототип є достатнім для показу логіки дипломного проєкту. Він показує ключові властивості гібридної моделі: гнучкість RBAC, жорсткість MAC, принцип deny overrides, аудит і контрольовану взаємодію між департаментами. [1]

ЗАГАЛЬНИЙ ВИСНОВОК

У межах кваліфікаційної роботи було розглянуто проблему керування доступом у корпоративних інформаційних системах та запропоновано підхід, який поєднує рольову модель RBAC і примусову модель MAC. Під час роботи стало зрозуміло, що використання лише однієї моделі не завжди дає потрібний результат. RBAC добре підходить для опису посад, ролей і службових обов'язків, але без додаткових обмежень може надати користувачу надто широкі права. MAC, навпаки, краще контролює рівні конфіденційності, однак у чистому вигляді є занадто жорсткою для реальних бізнес-процесів.

У роботі запропоновано гібридну модель, у якій рішення про доступ приймається не за однією умовою, а після послідовної перевірки кількох факторів. До них належать роль користувача, його департамент, посадовий рівень, рівень допуску, належність ресурсу до певного відділу, авторство ресурсу та наявність погодженого міждепартаментного доступу. Такий підхід робить систему більш гнучкою, але водночас не дозволяє обходити вимоги безпеки.

Окрему увагу в роботі приділено департаментній структурі. Це дало змогу наблизити модель до реального підприємства, де начальник відділу має права в межах свого департаменту, але не отримує автоматичного доступу до ресурсів інших підрозділів. Для випадків, коли між відділами потрібна співпраця, було передбачено механізм погодженого міждепартаментного доступу. Він працює точково: для конкретного користувача, конкретного ресурсу і конкретної дії.

Практична частина підтвердила, що запропоновану модель можна реалізувати у вигляді робочого web-застосунку. Backend-частина на FastAPI відповідає за перевірку доступу, роботу з базою даних, користувачами, ресурсами, департаментами й журналом аудиту. Frontend-частина на React дає змогу працювати із системою через зрозумілий графічний інтерфейс, без потреби постійно використовувати ручні API-запити.

У результаті було створено прототип, який підтримує автентифікацію, керування користувачами, ролями, ресурсами й департаментами, перевірку

доступу, аудит подій та міждепартаментне делегування прав. Важливо, що система не покладається лише на видимість кнопок у frontend: остаточне рішення завжди приймається на backend-рівні. Це зменшує ризик обходу правил через прямі запити до API.

Розроблений прототип показує, що поєднання RBAC і MAC є доцільним для бізнес-середовища. RBAC забезпечує зручне адміністрування відповідно до посад і структури організації, а MAC додає обов’язковий контроль рівня конфіденційності. Завдяки цьому користувач не отримує доступ до критичних ресурсів лише тому, що має певну роль.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Bell D. E., LaPadula L. J. Secure Computer System: Unified Exposition and Multics Interpretation. URL: https://csrc.nist.gov/files/pubs/conference/1998/10/08/proceedings-of-the-21st-nissc-1998/final/docs/early-cs-papers/bell76.pdf (с. 1–159).
2. Ferraiolo D. F., Kuhn D. R. Role-Based Access Controls. URL: https://csrc.nist.gov/files/pubs/conference/1992/10/13/rolebased-access-controls/final/docs/ferraiolo-kuhn-92.pdf (с. 1–7).
3. Sandhu R. S. et al. Role-Based Access Control Models. URL: https://csrc.nist.gov/csrc/media/projects/role-based-access-control/documents/sandhu96.pdf (с. 38–47).
4. Sandhu R., Ferraiolo D., Kuhn R. The NIST Model for Role-Based Access Control. URL: https://profsandhu.com/confrnc/rbac/rbac-nist.pdf (с. 1–12).
5. ANSI. Role Based Access Control (RBAC), INCITS 359. URL: https://blog.ansi.org/ansi/role-based-access-control-rbac-incits-359/ (розділи «Role Based Access Control (RBAC), INCITS 359», «RBAC Standard», «ANSI/INCITS 359»).
6. NIST SP 800-162. Guide to Attribute Based Access Control (ABAC). URL: https://nvlpubs.nist.gov/nistpubs/specialpublications/nist.sp.800-162.pdf (с. 1–8, 13–23).
7. NIST SP 800-53 Rev. 5. Security and Privacy Controls for Information Systems and Organizations. URL: https://csrc.nist.gov/pubs/sp/800/53/r5/upd1/final (розділи «AC — Access Control», «AU — Audit and Accountability»).
8. NIST SP 800-207. Zero Trust Architecture. URL: https://nvlpubs.nist.gov/nistpubs/specialpublications/NIST.SP.800-207.pdf (с. 1–17, 25–38).
9. OASIS. eXtensible Access Control Markup Language (XACML) Version 3.0. URL: https://docs.oasis-open.org/xacml/3.0/xacml-3.0-core-spec-os-en.html

(розділи «Policy Enforcement Point», «Policy Decision Point», «Policy Administration Point», «Policy Information Point»).

10. OWASP. Authorization Cheat Sheet. URL: https://cheatsheetseries.owasp.org/cheatsheets/Authorization_Cheat_Sheet.html (розділи «Deny by Default», «Validate Permissions on Every Request», «Enforce Least Privileges»).
11. OWASP. Application Security Verification Standard. URL: https://owasp.org/www-project-application-security-verification-standard/ (розділи «V4: Access Control», «V2: Authentication», «V14: Configuration»).
12. CISA. Zero Trust Maturity Model Version 2.0. URL: https://www.cisa.gov/sites/default/files/2023-04/zero_trust_maturity_model_v2_508.pdf (с. 1–16, 23–40).
13. ENISA. Digital Identity and Data Protection. URL: https://www.enisa.europa.eu/topics/digital-identity-and-data-protection (розділи «Digital Identity and Data Protection», «Private Sector», «Data Protection»).
14. ENISA. 2024 Report on the State of the Cybersecurity in the Union. URL: https://www.enisa.europa.eu/publications/2024-report-on-the-state-of-the-cybersecurity-in-the-union (розділи «State of Cybersecurity in the Union», «Publication details», «Additional materials»).
15. ISO/IEC 27001:2022. Information security management systems. URL: https://www.iso.org/standard/27001 (розділи «What is ISO/IEC 27001?», «Why is ISO/IEC 27001 important?», «Benefits», «General information»).
16. ISO/IEC 27002:2022. Information security controls. URL: https://www.iso.org/standard/75652.html (розділи «Information security controls», «General information», «Preview»).
17. MITRE. CWE-284: Improper Access Control. URL: https://cwe.mitre.org/data/definitions/284.html (розділи «Description», «Extended Description», «Common Consequences»).

18. MITRE ATT&CK. T1078: Valid Accounts. URL: https://attack.mitre.org/techniques/T1078/ (розділи «Description», «Procedure Examples», «Mitigations»).
19. FastAPI documentation. Features and automatic docs. URL: https://fastapi.tiangolo.com/features/ (розділи «Automatic docs», «Editor support», «Standards-based benefits», «Validation»).
20. Uvicorn documentation. ASGI server project. URL: https://www.uvicorn.org/ (розділи «Introduction», «Quickstart», «Running Uvicorn»).
21. SQLAlchemy ORM Documentation. URL: https://docs.sqlalchemy.org/orm/ (розділи «ORM Quick Start», «Declarative Mapping», «Session Basics»).
22. SQLite official site. URL: https://sqlite.org/ (розділи «About SQLite», «Serverless», «Self-contained»).
23. React documentation. Describing the UI. URL: https://react.dev/learn/describing-the-ui (розділи «Your First Component», «Importing and Exporting Components», «Writing Markup with JSX»).
24. Pydantic documentation. URL: https://pydantic.dev/docs/ (розділи «Validation», «Models», «JSON Schema», «Type hints»).
25. Python documentation. venv — Creation of virtual environments. URL: https://docs.python.org/3/library/venv.html (розділ «Creation of virtual environments»).
26. Git official site. URL: https://git-scm.com/ (розділи «About», «Documentation», «Getting Started»).
27. GitHub repository of the practical project. URL: https://github.com/roboaltic/Diploma (розділи репозиторію «app/», «frontend/», «scripts/»; файли «models.py», «main.py», «access_policy.py», «auth_context.py», frontend-компоненти та скрипти ініціалізації).
28. OWASP. Security Principles. URL: https://devguide.owasp.org/en/02-foundations/03-security-principles/ (розділи «Least Privilege», «Defense in Depth», «Secure Defaults»).

29. ENISA. Guideline on Security Measures under the EECC. URL: https://www.enisa.europa.eu/sites/default/files/publications/ENISA%20-%20Guideline%20on%20Security%20Measures%20under%20the%20EECC-%204th%20edition.pdf (с. 1–12, 25–45).
30. CISA. Zero Trust Maturity Model overview page. URL: https://www.cisa.gov/zero-trust-maturity-model (розділи «Zero Trust Maturity Model», «Pillars», «Implementation guidance»).
31. MDN Web Docs. CORS. URL: https://developer.mozilla.org/en-US/docs/Web/HTTP/CORS (розділи «Functional overview», «Examples of access control scenarios», «Preflighted requests»).
32. OpenAPI Specification. URL: https://spec.openapis.org/oas/latest.html (розділи «OpenAPI Document», «Paths», «Operations», «Request Bodies», «Responses»).

Додаток А

Файл models.py

```python
from datetime import datetime

from sqlalchemy import (
    Column,
    DateTime,
    ForeignKey,
    Integer,
    String,
    Table,
    Text,
)
from sqlalchemy.orm import relationship

from app.database import Base


user_roles = Table(
    "user_roles",
    Base.metadata,
    Column("user_id", Integer, ForeignKey("users.id"), primary_key=True),
    Column("role_id", Integer, ForeignKey("roles.id"), primary_key=True),
)


class Department(Base):
    __tablename__ = "departments"

    id = Column(Integer, primary_key=True, index=True)

    name = Column(String, unique=True, index=True, nullable=False)

    description = Column(String, nullable=True)


class User(Base):
    __tablename__ = "users"

    id = Column(Integer, primary_key=True, index=True)

    username = Column(String, unique=True, index=True, nullable=False)

    password_hash = Column(String, nullable=False, default="")

    clearance_level = Column(Integer, nullable=False, default=1)

    department = Column(String, nullable=False)

    department_position = Column(String, nullable=False, default="employee")

    roles = relationship(
        "Role",
        secondary=user_roles,
        back_populates="users",
    )

    created_resources = relationship(
        "Resource",
        back_populates="created_by",
        foreign_keys="Resource.created_by_user_id",
    )
```

```python
class Role(Base):
    __tablename__ = "roles"

    id = Column(Integer, primary_key=True, index=True)

    name = Column(String, unique=False, nullable=False)

    description = Column(String, nullable=True)

    department = Column(String, nullable=True)

    users = relationship(
        "User",
        secondary=user_roles,
        back_populates="roles",
    )


class Resource(Base):
    __tablename__ = "resources"

    id = Column(Integer, primary_key=True, index=True)

    name = Column(String, unique=True, index=True, nullable=False)

    department = Column(String, nullable=False)

    required_clearance_level = Column(Integer, nullable=False, default=1)

    required_position_level = Column(Integer, nullable=False, default=1)

    description = Column(String, nullable=True)

    content = Column(Text, nullable=True)

    created_by_user_id = Column(
        Integer,
        ForeignKey("users.id"),
        nullable=True,
    )

    created_by = relationship(
        "User",
        back_populates="created_resources",
        foreign_keys=[created_by_user_id],
    )


class AuditLog(Base):
    __tablename__ = "audit_logs"

    id = Column(Integer, primary_key=True, index=True)

    event_type = Column(String, nullable=False)

    actor_username = Column(String, nullable=True)

    target_username = Column(String, nullable=True)

    department = Column(String, nullable=True)

    action = Column(String, nullable=False)
```

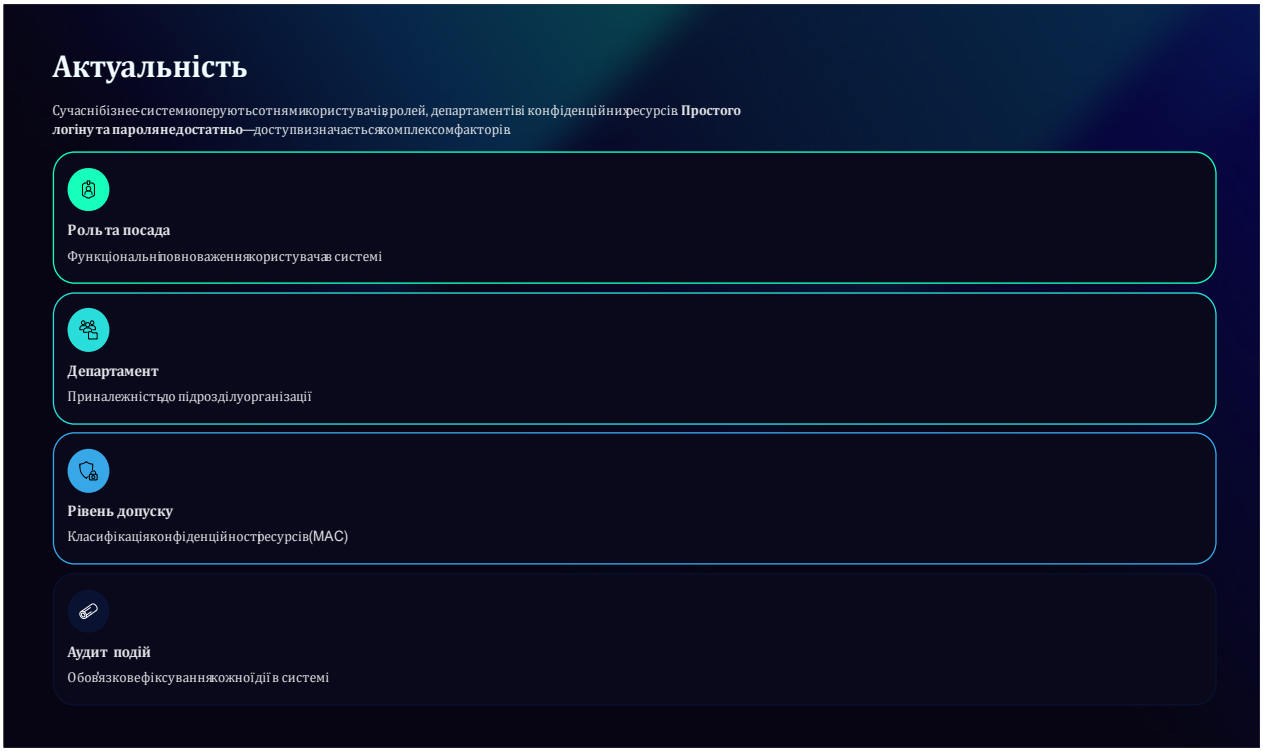
```
    details = Column(Text, nullable=True)

    created_at = Column(DateTime, default=datetime.utcnow)


class CrossDepartmentAccessRequest(Base):
    __tablename__ = "cross_department_access_requests"

    id = Column(Integer, primary_key=True, index=True)

    requester_head_id = Column(Integer, ForeignKey("users.id"), nullable=False)

    target_head_id = Column(Integer, ForeignKey("users.id"), nullable=True)

    user_id = Column(Integer, ForeignKey("users.id"), nullable=False)

    resource_id = Column(Integer, ForeignKey("resources.id"), nullable=False)

    source_department = Column(String, nullable=False)

    target_department = Column(String, nullable=False)

    action = Column(String, nullable=False, default="read")

    reason = Column(Text, nullable=True)

    status = Column(String, nullable=False, default="pending")

    response_comment = Column(Text, nullable=True)

    created_at = Column(DateTime, default=datetime.utcnow)

    resolved_at = Column(DateTime, nullable=True)

    requester_head = relationship(
        "User",
        foreign_keys=[requester_head_id],
    )

    target_head = relationship(
        "User",
        foreign_keys=[target_head_id],
    )

    user = relationship(
        "User",
        foreign_keys=[user_id],
    )

    resource = relationship(
        "Resource",
        foreign_keys=[resource_id],
    )
```

Файл – main.py

```python
from fastapi import FastAPI
from fastapi.middleware.cors import CORSMiddleware

from app.database import Base, engine
from app import models

from app.routers import auth
from app.routers import users
from app.routers import roles
from app.routers import resources
from app.routers import access
from app.routers import audit
from app.routers import departments
from app.routers import dashboard
from app.routers import health
from app.routers import system


Base.metadata.create_all(bind=engine)


app = FastAPI(
    title="Hybrid RMAC Access Control System",
    description="API for combined RMAC access control model with department hierarchy.",
    version="1.0.0"
)


origins = [
    "http://localhost:5173",
    "http://127.0.0.1:5173",
    "http://localhost:3000",
    "http://127.0.0.1:3000",
]


app.add_middleware(
    CORSMiddleware,
    allow_origins=origins,
    allow_credentials=True,
    allow_methods=["*"],
    allow_headers=["*"],
)


app.include_router(auth.router)
app.include_router(users.router)
app.include_router(roles.router)
app.include_router(resources.router)
app.include_router(access.router)
app.include_router(audit.router)
app.include_router(departments.router)
app.include_router(dashboard.router)
app.include_router(health.router)
app.include_router(system.router)


@app.get("/")
def root():
    return {
        "message": "Hybrid RMAC Access Control API is running"
    }
```

Додаток Б

Слайд - 1



Слайд – 2

Мета та завдання роботи

Мета

Розробити гібридну систему керування доступомчяка поєднує **RBAC**і **MAC**для корпоративного середовища з повним аудитом та контрольованим міждепартаментним доступомς

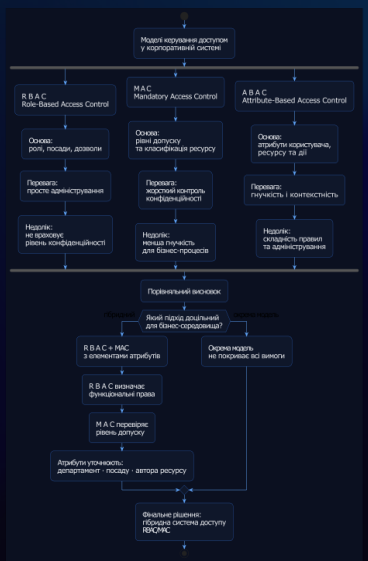
Завдання

- ¾§ Проаналізувати моделі і 9 ! : υq ! : та ! 9 ! :
- ⅝ς Спроєктувати гібридну модель доступу
- ⅞§ Реалізувати вебЌпрототип
- , ς Створити роліυдепартаменти та ресурси
- , ς Перевірити систему на демонстраційних сценаріях

Слайд – 3

Порівняння моделей доступу

Три основні моделі керування доступом вирішують різні задачі. У роботі обрано комбінацію **RBAC + MAC**доповнену атрибутами для департаменту , посади та автора ресурсу.



RBAC
Ролі та посади. Зручний для корпоративних структур.

MAC
Рівні конфіденційності. Обов'язкова перевірка допуску.

ABAC
Атрибути та контекст. Гнучкий, але складний.

Слайд – 4

Ідея гібридної моделі RBAC/MAC

Два незалежних фільтри забезпечують подвійний контроль доступ надається **лише за умови схвалення обома механізмами**.

RBAC-фільтр

Чи має користувач **роль і посаду**, що дозволяють виконати цю дію?

- Роль та функція
- Департамент
- Посада
- Права на ресурс

MAC-фільтр

Чи дозволяє **рівень допуску** користувача працювати з цим ресурсом?

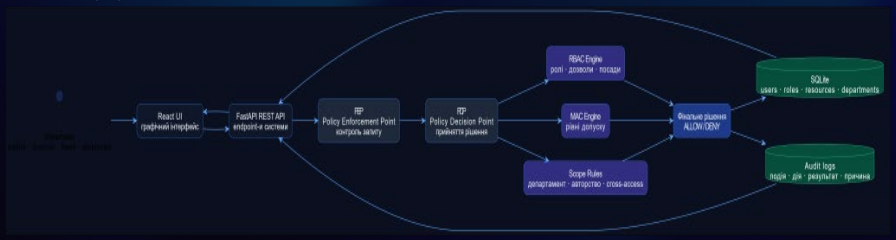
- Рівень конфіденційності
- Класифікація ресурсу
- Правила потоку інформації

Фінальне правило: RBAC: дозволено **і** MAC: дозволено → Доступ дозволено

Слайд – 5

Архітектура системи

Система побудована як сучасний веб-застосунок з чітким поділом на клієнтську та серверну частини. Ядром безпеки є модуль `access_policy.py`, що реалізує логіку PDP/PEP.



Кожен запит проходить через React UI до FastAPI, де PDP/PEP-модуль перевіряє права доступу, після чого дані зберігаються у SQLite та фіксуються в журналі аудиту.

Frontend
React — інтерфейс для всіх ролей

Backend
FastAPI + SQLAlchemy ORM

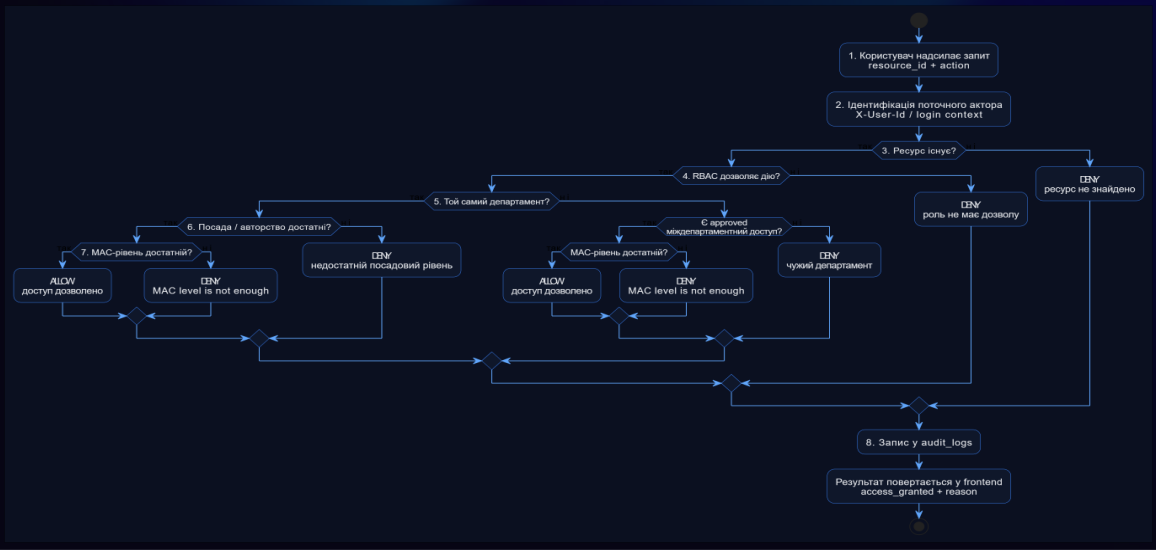
БД
SQLite з інтегрованим аудитом

Безпека
access_policy.py — ядро PDP/PEP

Слайд – 6

Алгоритм перевірки доступу

Кожен запит проходить **послідовних перевірок**ç Відмова на будь-якому кроці зупиняє процес і записує подію в журнал аудиту ç



Слайд – 7

Департаментна ієрархія

Структурадепартаменту

У системі реалізовано чотири департаменти[í vv°ę Ăę ÂÔ›l Ă₣ ¼Ğ› Dę¦ ²ę ÔÔÇ Кожен має тривівневу ієрархію ç

- Ë î Ôź Ă Ç Ę Ôę _ Ę Þ Ô Â Î А керує виключно власним відділом
- Ë î Ôź Þ ę ¼ Ę Þ Ô Â Î А заступник начальника
- Ë Ô Ę ź ą Ç ¼ Ô Ô А рядовий співробітник

Доступ до ресурсів чужого департаменту можливий **лише через офіційний запит погодження** А автоматичний перехід заблоковано ç

Слайд – 8

Слайд – 9

Слайд – 10

Слайд – 11

Слайд - 12