Міністерство освіти і науки України
«Київський національний університет імені Тараса Шевченка»

Факультет інформаційних технологій
Кафедра кібербезпеки та захисту інформації

ДОПУСТИТИ ДО ЗАХИСТУ:
В.о. завідувача кафедри
кібербезпеки та захисту інформації
_____Іван ПАРХОМЕНКО

«13» червня 2025 р.

ПОЯСНЮВАЛЬНА ЗАПИСКА

кваліфікаційної роботи

галузь знань 12 Інформаційні технології
(шифр і назва галузі знань)

спеціальність 125 Кібербезпека
(код і назва спеціальності)

освітній ступень бакалавр

освітня програма Кібербезпека
(назва освітньо-професійної програми)

на тему: «Програмний модуль адміністрування розподілу ресурсів і керування правами доступу»

Виконавець: студент IV курсу, групи КБ-44мс

_____ **Данило КРАМАРЕНКО**
(підпис) (ім'я, прізвище)

	Підпис	**Ім'я ПРІЗВИЩЕ**
Керівник		Юрій БАБЕНКО
Нормоконтроль		Інна МИХАЛЬЧУК

Київ 2025

Міністерство освіти і науки України
«Київський національний університет імені Тараса Шевченка»

Факультет інформаційних технологій
Кафедра кібербезпеки та захисту інформації

ЗАТВЕРДЖЕНО:
В.о. завідувача кафедри
кібербезпеки та захисту інформації
_____Іван ПАРХОМЕНКО

«29» листопада 2024 р.

ЗАВДАННЯ
на виконання кваліфікаційної роботи

спеціальності 125 Кібербезпека
(код і назва спеціальності)

освітньої програми Кібербезпека
(назва освітньо-професійної програми)

Студенту **КБ-44мс** **Крамаренку Данилу Васильовичу**
(група) (прізвище ім'я по-батькові)

Тема кваліфікаційної роботи «Програмний модуль адміністрування розподілу ресурсів і керування правами доступу»

1. ПІДСТАВИ ДЛЯ ПРОВЕДЕННЯ РОБОТИ

Тема кваліфікаційної роботи затверджена на засіданні кафедри кібербезпеки та захисту інформації протокол №6 від 28.11.2024 р.

2. ВИХІДНІ ДАНІ ДЛЯ ПРОВЕДЕННЯ РОБІТ

Розподіл ресурсів і керування правами доступу, СКБД MySQL, фреймворк для розробки додатків Qt 6.0+, бібліотека PySide6, мови Python 3.0+, SQL

3. ЗМІСТ РОЗРАХУНКОВО-ПОЯСНЮВАЛЬНОЇ ЗАПИСКИ

Аналіз та порівняння моделей контролю доступу, проектування інструменту адміністрування для розподілу інформаційних ресурсів і керування правами доступу, програмна реалізація інструменту адміністрування

4. ВИМОГИ ДО РЕЗУЛЬТАТІВ ВИКОНАННЯ РОБОТИ

Практична цінність Інструмент адміністрування для розподілу

інформаційних ресурсів і керування правами доступу

5. ДАТА ВИДАЧІ ЗАВДАННЯ

Дата видачі завдання: 29 листопада 2024 року

Завдання видав	_____	Юрій БАБЕНКО
	(підпис)	(ініціали, прізвище)
Завдання прийняв до виконання	_____	Данило КРАМАРЕНКО
	(підпис)	(ініціали, прізвище)

КАЛЕНДАРНИЙ ПЛАН

№ п/п	Найменування етапів робіт	Строки виконання робіт (початок-кінець)	Відмітка про виконання
1	Уточнення постановки задачі	29.11.2024 – 10.12.2024	*виконано*
2	Аналіз літератури	11.12.2024 – 29.12.2025	*виконано*
3	Огляд останніх досліджень і публікацій	10.01.2025 – 23.01.2025	*виконано*
4	Аналіз та порівняння моделей контролю доступу	24.01.2025 – 03.02.2025	*виконано*
5	Моделювання інструменту адміністрування та проектування бази даних	04.02.2025 – 19.02.2025	*виконано*
6	Опис архітектури інструменту адміністрування	19.02.2025 – 24.02.2025	*виконано*
7	Опис гібридної моделі управління доступом	25.02.2025 – 02.03.2025	*виконано*
8	Програмна реалізація і демонстрація роботи інструменту адміністрування	03.03.2025 – 02.04.2025	*виконано*
9	Аналіз та порівняння наявних рішень	03.04.2025 – 10.04.2025	*виконано*
10	Оформлення пояснювальної записки	11.04.2025 – 13.05.2025	*виконано*
11	Підготовка до захисту	14.05.2025 – 13.06.2025	*виконано*

Завдання видав	_____	Юрій БАБЕНКО
	(підпис)	(ініціали, прізвище)
Завдання прийняв до виконання	_____	Данило КРАМАРЕНКО
	(підпис)	(ініціали, прізвище)

Термін подання кваліфікаційної роботи до ЕК 13 червня 2025 року

РЕФЕРАТ

Кваліфікаційна робота складається зі вступу, трьох розділів, загальних висновків, списку використаних джерел, додатків, має 67 сторінок основного тексту, 13 таблиць та 19 рисунків. Список використаних джерел містить 33 найменування і займає 4 сторінки.

Метою кваліфікаційної роботи є підвищення ефективності керування правами доступу шляхом розробки інструменту адміністрування.

Об'єктом дослідження є процес надання доступу користувачам до розподілених інформаційних ресурсів.

Предметом дослідження є методи, засоби та моделі керування доступом.

Методами дослідження є метод аналізу та порівняльний метод.

Новизною роботи є запропонована гібридна модель управління доступом, за основу якої обрано модель контролю доступу на основі груп, додаючи принцип розподілу прав доступу до ресурсів на основі шаблонів прав доступу.

Для досягнення зазначеної мети були поставлені наступні завдання:

- розглянути особливості моделей керування доступом;
- проаналізувати та порівняти моделі контролю доступу;
- здійснити моделювання інструменту адміністрування для розподілу інформаційних ресурсів і керування правами доступу;
- здійснити проектування бази даних інструменту адміністрування;
- виконати програмну реалізацію інструменту адміністрування;
- проаналізувати та порівняти наявні рішення для розподілу інформаційних ресурсів і керування правами доступу.

Практичною цінністю є інструмент адміністрування для розподілу інформаційних ресурсів і керування правами доступу.

Ключові слова: контроль доступу, інструмент адміністрування, розподіл інформаційних ресурсів, налаштування прав доступу, моделі контролю доступу, гібридна модель управління доступом, GBAC, Python, SQL.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ ТА СКОРОЧЕНЬ

СКБД	–	Система керування базами даних
GBAC	–	Group Based Access Control
IoT	–	Internet of Things
MFA	–	Multi-factor authentication
2FA	–	Two-factor authentication
SFA	–	Single Factor Authentication
OTP	–	One-time password
DAC	–	Discretionary Access Control
MAC	–	Mandatory Access Control
RBAC	–	Role Based Access Control
ABAC	–	Attribute Based Access Control
OrBAC	–	Organization Based Access Control
RuleBAC	–	Rule Based Access Control
ReBAC	–	Relationship Based Access Control
UML	–	Unified Modeling Language
EERM	–	Extended Entity-Relationship Model
IAM	–	Identity and Access Management
REST	–	Representational State Transfer
API	–	Application Programming Interface
ООП	–	Об'єктно-орієнтоване програмування
SQL	–	Structured Query Language

ВСТУП

У сучасну цифрову епоху, коли організації все більше покладаються на IT-інфраструктуру, ефективне управління інформаційними ресурсами та правами доступу набуває вирішального значення. Стрімке зростання обсягів даних, програмних рішень і хмарних обчислень призводить до значних викликів забезпечення інформаційної безпеки [1].

Контроль доступу до інформаційних ресурсів і сервісів є фундаментальною основою безпеки. Протягом багатьох років розроблено безліч моделей контролю доступу, кожна з яких призначена для вирішення різних аспектів цієї проблеми [2].

Перспективи розвитку виражаються у створенні гібридних моделей управління доступом та нових рішень з контролю доступу для наступних сфер: хмарні обчислення, інтернет речей, блокчейн, мобільне хмарне середовище, інтелектуальні екосистеми спільної роботи, штучний інтелект, обмін даними на смарт-пристроях та розподілених базах даних [3].

Контроль доступу на основі груп – модель безпеки, яка регулює доступ до ресурсів шляхом призначення дозволів на основі членства користувача в групі. Контроль доступу на основі груп спрощує управління дозволами, налаштовуючи доступ користувачам на основі їхньої приналежності до заздалегідь визначених груп, підвищуючи загальну безпеку системи та ефективність адміністрування.

Отже, для підвищення ефективності та гнучкості налаштування прав доступу, необхідно розробити інструмент адміністрування для розподілу інформаційних ресурсів і керування правами доступу, заснований на гібридній моделі управління доступом, за основу якої обрано модель контролю доступу на основі груп, додаючи принцип розподілу прав доступу до ресурсів на основі шаблонів прав доступу.

Для досягнення зазначеної мети необхідно виконати наступні завдання:

- розглянути особливості моделей керування доступом;

- проаналізувати та порівняти моделі контролю доступу;
- здійснити моделювання інструменту адміністрування для розподілу інформаційних ресурсів і керування правами доступу;
- здійснити проектування бази даних інструменту адміністрування;
- виконати програмну реалізацію інструменту адміністрування;
- проаналізувати та порівняти наявні рішення для розподілу інформаційних ресурсів і керування правами доступу.

Новизною роботи є запропонована гібридна модель управління доступом, за основу якої обрано модель контролю доступу на основі груп, додаючи принцип розподілу прав доступу до ресурсів на основі шаблонів прав доступу.

Практичною цінністю є інструмент адміністрування для розподілу інформаційних ресурсів і керування правами доступу.

Інструмент адміністрування «Admin Console» покликаний:

- автоматизувати рутинні процеси, налаштовуючи права доступу одночасно для декількох сутностей, що значно підвищує ефективність процесу адміністрування;
- автоматизувати процес відкликання учасників груп, налаштувавши для них час відкликання з відповідних груп;
- надати гнучкий процес налаштування прав доступу, зменшуючи складність навчання користувачів для роботи з розробленим рішенням;
- забезпечити масштабованість та додатковий рівень захисту під час входу в систему, застосовуючи MFA;
- забезпечити економічну ефективність та оптимізацію бюджету у порівнянні з наявними рішеннями для розподілу ресурсів і керування правами доступу, такими як Azure AD та Google Cloud IAM.

Апробація результатів кваліфікаційної роботи відбулася на XVIII Міжнародній науково-практичній конференції «Modern Information and communication technologies in transport, industry and education» (Додаток А).

РОЗДІЛ 1
АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Огляд останніх досліджень і публікацій

Відповідно до Кодексу США, інформаційна безпека – це забезпечення захисту інформації та інформаційних систем з метою забезпечення конфіденційності, цілісності та доступності [4]. Інформаційна безпека виступає гарантією того, що інформація та інформаційні системи не потраплять до чужих рук, щоб ними не маніпулювали і не використовували в неправильних чи шкідливих цілях [5].

Контроль доступу обмежує те, що користувач може робити безпосередньо, а також те, що дозволено робити програмам, які виконуються від імені користувача. Таким чином, контроль доступу спрямований на запобігання активності, яка може призвести до порушення безпеки.

Важливо також розуміти, що контроль доступу не є повним рішенням для захисту системи, і з метою досягнення максимальної ефективності, він повинен поєднуватися з аудитом. Аудит полягає в апостеріорному аналізі всіх запитів і дій користувачів в системі. Аудит вимагає реєстрації (логування) всіх запитів і дій користувачів для їх подальшого аналізу [6].

Аутентифікація, авторизація та облік (Authentication, Authorization, and Accounting, AAA) – це система безпеки, яка контролює доступ до комп’ютерних ресурсів, забезпечує дотримання політик та аудит використання.

Аутентифікація передбачає надання користувачем інформації про те, ким він є. Користувачі надають облікові дані для входу, які підтверджують, що вони є тими, за кого себе видають.

Авторизація слідує за аутентифікацією. Під час авторизації користувачеві можуть бути надані привілеї на доступ до певних областей мережі або системи.

Області та набори дозволів, надані користувачеві, зберігаються в базі даних разом з ідентифікаційними даними користувача.

Облік відстежує активність користувачів у мережі, фіксуючи таку інформацію, як тривалість їхнього перебування в мережі, дані, які вони надсилали або отримували, IP-адреси, уніфіковані ідентифікатори ресурсів (URI), а також різні сервіси, до яких вони зверталися [7].

Процес встановлення особи користувача називається ідентифікацією та аутентифікацією (I&A). Мета полягає в тому, щоб тільки авторизовані користувачі мали доступ до комп'ютерної системи, мережі або певного сервісу. Кожна ідентичність асоціюється з обліковими даними аутентифікації, які відомі лише користувачеві і можуть бути перевірені системою. Передумова, що суб'єкт зберігає в таємниці свій обліковий запис, за винятком надання його певній обчислювальній системі, забезпечує автентичність ідентичності, пов'язаної з цим суб'єктом.

Існує три способи створення та надання секрету обчислювальній системі:

- Представлення чогось, що відомо користувачеві. Пароль, персональний ідентифікаційний номер (PIN-код) та парольна фраза – найпоширеніші схеми в цій категорії. Політики можуть накладати різні обмеження на паролі, наприклад, обмежувати алфавіт пароля, його синтаксис, довжину або термін дії. Поширення систем і додатків, які використовують паролі, зазвичай призводить до того, що один користувач має працювати з декількома паролями. На жаль, для пом'якшення таких проблем користувачі вдаються до використання слабких паролів, які легко запам'ятовуються. В результаті підвищується ризик витоку інформації [8]. Фактори, засновані на знаннях, здебільшого використовуються в однофакторній аутентифікації, коли для доступу до системи користувачеві потрібно надати дійсне ім'я користувача і відповідний пароль [5].

- Пред'явлення чогось, що є у користувача. Ця схема аутентифікації полягає у зберіганні облікової інформації на пристрої, який зазвичай є портативним і розміром з кредитну картку. Цей пристрій, відомий як токен, подається на вхід зчитувального пристрою, підключеного до основної

обчислювальної системи. Облікові дані, що зберігаються в токені, використовуються для аутентифікації користувача в системі на основі заздалегідь визначеного протоколу. Смарт-картки є поширеним прикладом такої техніки аутентифікації.

- Представлення того, ким є користувач. Ця схема спирається на біометричні ознаки, які надійно розрізняють користувачів. Приклади включають відбитки пальців, геометрію рук, форму очей, розпізнавання голосу та обличчя, а також власноручний підпис. Цей метод залишається обмеженим у використанні частково через додаткові витрати, які він несе, і, можливо, через неточності пов’язаних з ним технологій [8].

Нижче описані деякі поширені типи аутентифікації, до яких відносяться: аутентифікація за допомогою пароля, 2FA, біометрична аутентифікація.

Аутентифікація за допомогою пароля є найпоширенішим типом аутентифікації користувачів. Її також називають SFA. Вона вимагає від користувача введення відомого йому секрету. Цей секрет може бути паролем або PIN-кодом. Дослідження показують, що цей тип аутентифікації є менш безпечним і схильний до кількох кібератак, таких як підбір пароля, атаки грубої сили, фішинг та інші.

2FA є типом аутентифікації, який доповнює однофакторну аутентифікацію для забезпечення додаткового рівня безпеки. Це досягається шляхом запиту користувача на введення коду з програмного або апаратного токена після введення дійсного імені користувача та пароля. 2FA була розроблена для усунення деяких відомих вразливостей SFA. Хоча цей тип аутентифікації не забезпечує стовідсоткової безпеки, він є кращим і безпечнішим, ніж однофакторна аутентифікація. Деякі варіації 2FA включають смарт-картки, апаратні токени безпеки, токени на програмній основі та інші [5].

Міністерство внутрішньої безпеки США визначає біометрію як унікальні фізичні властивості, які можуть бути використані для автоматичного розпізнавання [9]. Біометрична аутентифікація покладається на фізіологічні характеристики користувача для цілей аутентифікації. Деякі фізичні

характеристики включають колір райдужної оболонки ока, візерунок відбитків пальців, розпізнавання обличчя тощо. Цей метод аутентифікації спрямований на вдосконалення 2FA та подолання деяких проблем безпеки, притаманних 2FA. Трифакторна аутентифікація покращує SFA, забезпечуючи фізичну присутність користувача.

За допомогою двофакторної аутентифікації можна запобігти атакам з використанням викрадених паролів, оскільки зловмисник повинен підтвердити другий фактор, перш ніж отримати доступ до облікового запису користувача. 2FA покладається на наявну парольну аутентифікацію (те, що знає користувач), на додаток до того, що користувач має, наприклад, апаратний токен або те, що він є, наприклад, відбиток пальця.

Хоча 2FA не обіцяє повної безпеки, за умови правильної реалізації вона гарантує, що зловмисникові буде складно здійснити атаку. 2FA може бути реалізована в багатьох формах, таких як одноразовий пароль, кнопки "Прийняти/Відхилити", сканування QR-коду, апаратні токени тощо [5].

Зберігання та використання лише односторонніх криптографічних перетворювань паролів недостатньо для того, щоб запобігти словниковим атакам на пароль. Атака за словником, яку також називають атакою з попередніми обчисленнями, полягає в тому, що зловмисник, знаючи деталі одностороннього перетворення, попередньо обчислює одностороннє кодування словника ймовірних паролів, отримує пароль в закодованому вигляді і шукає його в словнику на предмет можливого збігу з ним. Атаки грубої сили, які не залежать від попередньо створеного словника, також можуть бути використані для злому закодованих паролів.

Одна зміна у вхідних даних алгоритму одностороннього перебору призводить до отримання іншого перебору. Сіль є значенням, яке включається в розрахунок перетворення пароля, щоб запобігти атакам за допомогою словника. Псевдовипадкова генерація значень солі ускладнює їх підбір. Таким чином, використання солі допомагає уникнути зіткнення паролів і потенційно обмежує

кількість облікових записів користувачів, які можуть бути скомпрометовані одночасно.

В основі системи контролю доступу лежить безпечна оцінка того, чи має встановлена ідентичність доступ до певного обчислювального ресурсу, який також називається об’єктом.

Моделі контролю доступу допомагають покращити управління контролем доступу до рівня, який є лаконічним, а в деяких випадках навіть формальним [8].

Системи, засновані на властивостях (Capabilities), забезпечують набагато сильнішу підтримку точного, мінімального і змістовного делегування повноважень, що має фундаментальне значення для безпечної роботи [10].

Багаторівневий метод контролю доступу відомий наявністю декількох рівнів безпеки даних. Доступ регулюється залежно від рівня допуску користувача та рівня класифікації (безпеки) об’єкта. Модель Бела-ЛаПадули (Bell-LaPadula Model) заснована на багаторівневому методі контролю доступу. У цій моделі користувачеві з певним рівнем доступу не дозволяється читати інформацію вище цього рівня та записувати інформацію, яка засекречена нижче його/її рівня доступу. Модель Бела-ЛаПадули забезпечує конфіденційність в системі [3].

Запровадити багаторівневу безпеку в обов’язковому порядку, щоб ані користувачі, ані їхні програми не могли змінювати допуски користувачів або класифікацію файлів, також легко зробити. Таке пряме застосування багаторівневого захисту зазвичай називають простою безпекою в моделі Белла і ЛаПадули (Bell-LaPadula Model) [11].

Модель Біба (Biba Model) зосереджена на забезпеченні цілісності в системі. У простій властивості цілісності цієї моделі, користувачеві дозволяється читати інформацію, рівень безпеки якої вищий за його/її рівень допуску та записувати до об’єкту, якщо рівень безпеки об’єкту нижчий за рівень доступу користувача.

Дискреційний контроль доступу дозволяє власникам ресурсів надавати доступ до них іншим особам. Гнучкість цієї парадигми, однак, усуває будь-яку

можливість визначення меж, яких може досягти той чи інший стан захисту. DAC є найпоширенішою парадигмою контролю доступу, яка відповідає багатьом реальним процесам, які керують доступом до ресурсів на основі права власності [8]. Політики DAC мають найбільше застосування завдяки своїй гнучкості. DAC застосовується в операційних системах у поєднанні з іншими моделями контролю доступу [3].

Обов'язковий контроль доступу було введено в Критерії оцінки довірених комп'ютерних систем (TCSEC) [12], опубліковані Міністерством оборони США. В основі MAC лежить математична модель Белла-ЛаПадули. Характерною рисою MAC є проходження потоку даних в одному напрямку через матрицю міток безпеки [13]. Мітка, яка призначається користувачеві, називається допуском до системи безпеки. Мітка, призначена об'єкту, називається класифікацією безпеки. MAC-політика є обов'язковою, і користувач не може її змінити [3]. Адміністратори організації надають або відмовляють у доступі, присвоюючи класифікацію безпеки ресурсам та активним об'єктам, які називаються мітками та допусками відповідно. Рішення про доступ приймаються відповідно до частково впорядкованих відносин між мітками і допусками [8].

В моделі MAC, у найпростішому випадку рівень безпеки (допуск) є елементом ієрархічної впорядкованої множини. У військовій та цивільній сферах ієрархічна множина зазвичай складається з Цілком таємно (TS), Таємно (S), Конфіденційно (C) та Несекретно (U). Доступ суб'єкта до об'єкта надається лише за умови виконання певних співвідношень, залежно від типу доступу, між рівнями безпеки, пов'язаними з цими двома рівнями. Зокрема, обов'язковим є дотримання наступних двох принципів. Читання/запис – рівень доступу суб'єкта повинен переважати над рівнем безпеки об'єкта, що зчитується/записується [6].

Контроль доступу на основі ролей з'явився як узагальнена модель доступу. Характерною особливістю RBAC є те, що дозволи призначаються ролям, а користувачам призначаються відповідні ролі [8]. Результатом застосування RBAC є спрощене управління дозволами. Базова модель RBAC складається з набору користувачів, набору ролей і набору дозволів. RBAC підтримує такі

функції, як гнучкість, масштабованість, контроль робочого процесу та розподіл обов'язків. RBAC використовується в корпоративному програмному забезпеченні.

Контроль доступу на основі атрибутів заснований на «атрибутах», які є характеристиками суб'єктів та об'єктів. Суб'єктом може бути людина або пристрій. Об'єкт – це запитуваний ресурс програмної системи. Умови навколишнього середовища включають дату і час, а також місцезнаходження користувача. Механізм управління доступом ABAC оцінює атрибути, умови середовища і політики і приймає рішення про доступ. ABAC – масштабована і гнучка модель контролю доступу, яка застосовується в корпоративному програмному забезпеченні та веб-сервісах [3].

Коли суб'єкт запитує доступ, механізм ABAC може прийняти рішення про управління доступом на основі призначених атрибутів запитувача, призначених атрибутів об'єкта, умов середовища і набору політик, які визначені в термінах цих атрибутів і умов [14].

Контроль доступу на основі організації зосереджений на правилах, які виражають контекстні дозволи, заборони, зобов'язання або рекомендації. Правила в OrBAC є специфічними для організації. Роль є зв'язком між суб'єктами та організаціями [3]. У динамічному середовищі, такому як хмарні обчислення, правила авторизації повинні бути виражені гнучким способом та можуть включати завдання, які можуть бути пропущені в деяких випадках, щоб тимчасово адаптуватися до змін контексту [15].

Контроль доступу на основі правил застосовується у децентралізованих системах та соціальних мережах. Мережа представляється графом, де користувачі є вузлами, а ребра виступають у якості зв'язків між користувачами. RuleBAC використовує концепцію ролей як політик. Існують обмеження доступу, пов'язані з типом, глибиною та рівнем довіри відносин з іншими користувачами. Глибина відносин – це найкоротший шлях, що відповідає відносинам між двома користувачами. Трансформація моделі підвищує гнучкість RuleBAC [3]. У структурі моделі RuleBAC, авторизовані суб'єкти

виражаються на основі форми відносин, глибини і ступеня довіри, які існують між користувачами мережі за допомогою RBAC на основі атрибутів [16].

Контроль доступу на основі відносин базується на відносинах між власником ресурсу і запитувачем ресурсу в соціальній мережі. ReBAC враховує контекст взаємовідносин. Характерними рисами моделі контролю доступу ReBAC є відстеження міжособистісних стосунків між користувачами та використання їхніх стосунків у політиках контролю доступу [3]. Однак, використання лише зв’язків часто є недостатнім для дотримання різних вимог щодо безпеки та конфіденційності, які відповідають очікуванням сучасних користувачів соціальних мереж [17].

Контроль доступу на базі блокчейн технології (Blockchain access control) заснована на технології блокчейн, що складається зі зв’язаних блоків, які не можуть бути змінені. Блоки блокчейну перевіряються учасниками, які називаються майнерами, в одноранговій мережі (P2P). Блокчейн – це децентралізована, розподілена, незворотна, відстежувана і захищена від підробки технологія. Кожен майнер ділиться набором пов’язаних блоків у блокчейні. Для вирішення проблеми безпеки та управління даними між граничними вузлами IoT та масовими гетерогенними пристроями в поєднанні з широким застосуванням технології блокчейн в управлінні безпекою даних розподілених систем, пропонується модель управління доступом до Інтернету речей на основі блокчейну (SC-ABAC) шляхом поєднання смарт-контрактів та управління доступом на основі атрибутів [18].

Контроль доступу на основі груп – модель безпеки, яка регулює доступ до ресурсів шляхом призначення дозволів на основі членства користувача в групі. Вона спрощує управління дозволами, надаючи або обмежуючи доступ користувачам на основі їхньої приналежності до заздалегідь визначених груп, підвищуючи загальну безпеку системи та ефективність адміністрування. Групи надають швидкий та ефективний механізм контролю доступу. Замість того, щоб керувати доступом для кожного користувача окремо, GBAC об’єднує користувачів у групи на основі їхніх ролей чи обов’язків [19].

1.2 Аналіз та порівняння моделей контролю доступу

Аналіз та порівняння моделей контролю доступу проведений за такими параметрами: гнучкість, масштабованість, розподіленість, динамічність, структура, збереження ідентичності користувача, контроль робочого процесу, деталізація політик, використання часу в політиках, делегування довіри, достовірність, сфера застосування.

Розподілені системи та системи управління робочими процесами вимагають динамічних моделей контролю доступу. ReBAC та OrBAC є динамічними моделями. ReBAC використовує контекст відносин. У TaskBAC хід виконання завдань підтримує динамічний контроль доступу. OrBAC можна комбінувати з TaskBAC, що робить OrBAC динамічним. GBAC можна комбінувати з іншими моделями, що робить GBAC динамічним.

Делегування довіри показує, чи передає модель привілеї від одного користувача до іншого на основі довірчих відносин між користувачами. Відносини у моделі контролю доступу на основі відносин використовують контекст, який підтримує делегування довіри.

Детально визначені політики означають здатність політик моделі забезпечувати більш детальну перевірку контролю доступу. Контроль доступу на основі ролей не є детально визначеною моделлю, тому що вона запобігає виконанню операції, але не захищає конкретні дані. Контроль доступу на основі атрибутів є детально визначеною моделлю завдяки політикам, які оцінюють атрибути.

Гнучкість є важливою характеристикою моделі контролю доступу, яка виражається у здатності політик моделі контролю доступу пристосовуватися до сфери застосування. Політики моделі дискреційного контролю доступу мають широке застосування в операційних системах завдяки своїй гнучкості. Модель контролю доступу на основі груп є гнучкою через можливість застосування різних типів політик. Політики моделей контролю доступу на основі ролей та

контролю доступу на основі правил є гнучкими, але ці моделі базуються на ролях. У моделі контролю доступу на основі атрибутів гнучкість досягається шляхом прийняття динамічного рішення про контроль доступу, яке базується на атрибутах.

Масштабованість є однією з найважливіших характеристик моделі контролю доступу, яка показує, чи може модель працювати зі збільшенням кількості користувачів та об'єктів програмної системи. Моделі контролю доступу RBAC, GBAC та ABAC масштабуються для корпоративних систем.

Такий параметр, як час, використовується в політиках моделей ABAC, ReBAC та RuleBAC. ABAC час включає в умови навколишнього середовища. ReBAC включає час в контекст відносин.

Достовірність означає, що дані передаються від довіреного користувача, або довіреного об'єкта, або довіреного контексту. ReBAC підтримує делегування довіри. У RuleBAC існують рівні довіри, які призначаються відносинам між користувачами.

Контроль робочого процесу показує, чи дозволяє модель відстежувати робочий процес за допомогою своїх політик. Моделі контролю доступу на основі ролей та контролю доступу на основі груп підтримують контроль робочого процесу. Модель контролю доступу на основі організації призначена для відстеження робочого процесу.

Розподіленість є важливою характеристикою моделі контролю доступу. Деякі моделі доступу призначені для розподілених систем. ReBAC застосовується в соціальних мережах і підтримує розподілений контроль доступу. TokenBAC і контроль доступу до блокчейну на основі смарт-контрактів (BACSC) застосовуються в блокчейні, який є розподіленою технологією. SEAC призначений для розподілених баз даних.

Модель DAC використовуються в операційних системах. Модель обов'язкового контролю доступу MAC використовується для військових застосувань. Модель ReBAC використовується для соціальних мереж в Інтернеті. Модель TokenBAC пов'язана з розподіленими додатками, блокчейн

технологією, IoT і хмарними обчисленнями. Модель CBAC використовується для захисту трафіку через брандмауери та IoT. Моделі RBAC, ABAC та GBAC застосовуються в корпоративному програмному забезпеченні [3].

1.3 Постановка задачі

Перспективи розвитку виражаються у створенні гібридних моделей управління доступом та нових рішень з контролю доступу для наступних сфер: хмарні обчислення, IoT, блокчейн, мобільне хмарне середовище, інтелектуальні екосистеми спільної роботи, штучний інтелект, обмін даними на смарт-пристроях та розподілених базах даних [3].

Отже, необхідно розробити інструмент адміністрування для розподілу інформаційних ресурсів і керування правами доступу, заснований на гібридній моделі управління доступом. За основу гібридної моделі управління доступом обрано модель контролю доступу на основі груп, додаючи принцип розподілу прав доступу до ресурсів на основі шаблонів прав доступу.

Для досягнення зазначеної мети необхідно виконати наступні завдання:

- здійснити моделювання інструменту адміністрування для розподілу інформаційних ресурсів і керування правами доступу;
- здійснити проектування бази даних інструменту адміністрування;
- виконати програмну реалізацію інструменту адміністрування;
- проаналізувати та порівняти наявні рішення для розподілу інформаційних ресурсів і керування правами доступу.

Висновки за розділом 1

У першому розділі було проведено огляд останніх досліджень і публікацій, зокрема розглянуто важливість інформаційної безпеки та контролю доступу, систему безпеки – аутентифікація, авторизація та облік й особливості моделей керування доступом.

Процес встановлення особи користувача називається ідентифікацією та аутентифікацією. Існує три способи створення та надання секрету обчислювальній системі: представлення чогось, що відомо користувачеві, пред'явлення чогось, що є у користувача і представлення того, ким є користувач. В основі системи контролю доступу лежить безпечна оцінка того, чи має встановлена ідентичність доступ до певного обчислювального ресурсу.

Моделі контролю доступу допомагають покращити управління контролем доступу до рівня, який є лаконічним, а в деяких випадках навіть формальним. Розглянуто особливості наступних моделей керування доступом: Bell-LaPadula, Biba, DAC, MAC, RBAC, ABAC, OrBAC, RuleBAC, ReBAC та GBAC.

Проаналізовано та порівняно моделі контролю доступу за такими параметрами: гнучкість, масштабованість, розподіленість, динамічність, структура, збереження ідентичності користувача, контроль робочого процесу, деталізація політик, використання часу в політиках, делегування довіри, достовірність, сфера застосування.

Контроль доступу на основі груп – модель безпеки, яка регулює доступ до ресурсів шляхом призначення дозволів на основі членства користувача в групі. Контроль доступу на основі груп спрощує управління дозволами, налаштовуючи доступ користувачам на основі їхньої приналежності до заздалегідь визначених груп, підвищуючи загальну безпеку системи та ефективність адміністрування.

Поставлено завдання розробити інструмент адміністрування для розподілу інформаційних ресурсів і керування правами доступу, заснований на гібридній моделі управління доступом. За основу гібридної моделі управління доступом обрано модель контролю доступу на основі груп, додаючи принцип розподілу прав доступу до ресурсів на основі шаблонів прав доступу.

РОЗДІЛ 2
ПРОЕКТУВАННЯ ІНСТРУМЕНТУ АДМІНІСТРУВАННЯ ДЛЯ РОЗПОДІЛУ РЕСУРСІВ І КЕРУВАННЯ ПРАВАМИ ДОСТУПУ

2.1 Моделювання інструменту адміністрування

UML – універсальна мова візуального моделювання, яка використовується для визначення, візуалізації, побудови та документування артефактів програмної системи. Вона охоплює рішення та уявлення про системи, які мають бути побудовані. UML використовується для розуміння, проектування, перегляду, конфігурації, підтримки та контролю інформації про такі системи. Вона призначена для використання з усіма методами розробки, стадіями життєвого циклу, доменами додатків і середовищами.

UML фіксує інформацію про статичну структуру та динамічну поведінку системи. Система моделюється як сукупність дискретних об’єктів, які взаємодіють для виконання роботи, що в кінцевому підсумку приносить користь зовнішньому користувачеві. Статична структура визначає типи об’єктів, важливих для системи та її реалізації, а також взаємозв’язки між об’єктами. Динамічна поведінка визначає історію об’єктів у часі та зв’язки між об’єктами для досягнення цілей. Моделювання системи з кількох окремих, але пов’язаних між собою точок зору дозволяє зрозуміти її для різних цілей [20].

Об’єктно-орієнтований підхід є важливим підходом, і він є одним з найпопулярніших підходів для розробки систем. Він спрямований на визначення вимог і використовує діаграми для моделювання та аналізу вимог до системи. Крім того, існує багато мов для об’єктно-орієнтованого програмування, таких як C, C++, Java та інші [21].

UML-діаграма варіантів використання (Use Case Diagram) моделює функціональність системи, як її сприймають зовнішні користувачі, які називаються акторами. Метою представлення варіантів використання є перелік

акторів і варіантів використання, а також демонстрація того, які актори беруть участь у кожному варіанті використання [20]. Діаграми варіантів використання в UML підтримують такі відносини, як включення (inclusion), розширення (extension) та узагальнення (generalization) [22].

Під час аналізу вимог, діаграми варіантів використання допомагають ідентифікувати акторів і визначити за допомогою варіантів використання поведінку системи. Також UML визначає невеликий набір зв'язків для структурування акторів та варіантів використання. Останні можуть бути пов'язані з іншими варіантами використання такими зв'язками, як розширення, включення та узагальнення [23]. UML-діаграма варіантів використання інструменту адміністрування «Admin Console» зображена на рис. 2.1.

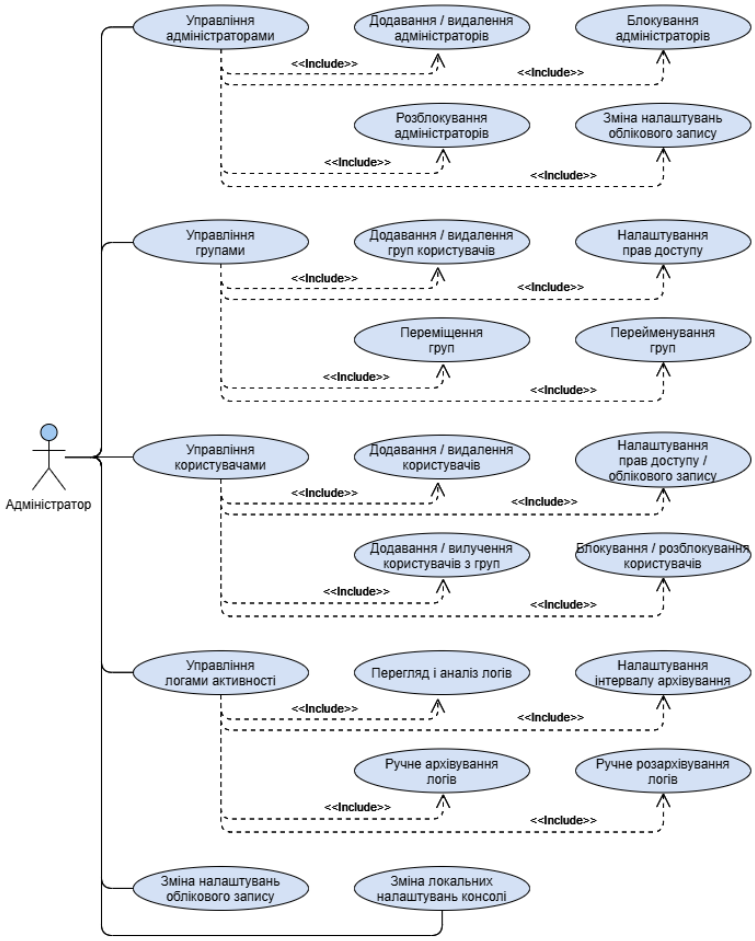


Рисунок 2.1 – UML-діаграма варіантів використання інструменту адміністрування «Admin Console»

UML-діаграма діяльності (Activity Diagram) є варіантом машини станів (State Machine), який показує обчислювальну діяльність, пов’язану з виконанням обчислень [20]. Діаграма діяльності є важливою діаграмою для моделювання динамічних аспектів системи. UML-діаграми діяльності можуть відображати дії (послідовні та паралельні), об’єкти даних, які вони споживають або виробляють, а також порядок виконання різних дій [24]. UML-діаграма діяльності інструменту адміністрування «Admin Console» зображена на рис. 2.2.

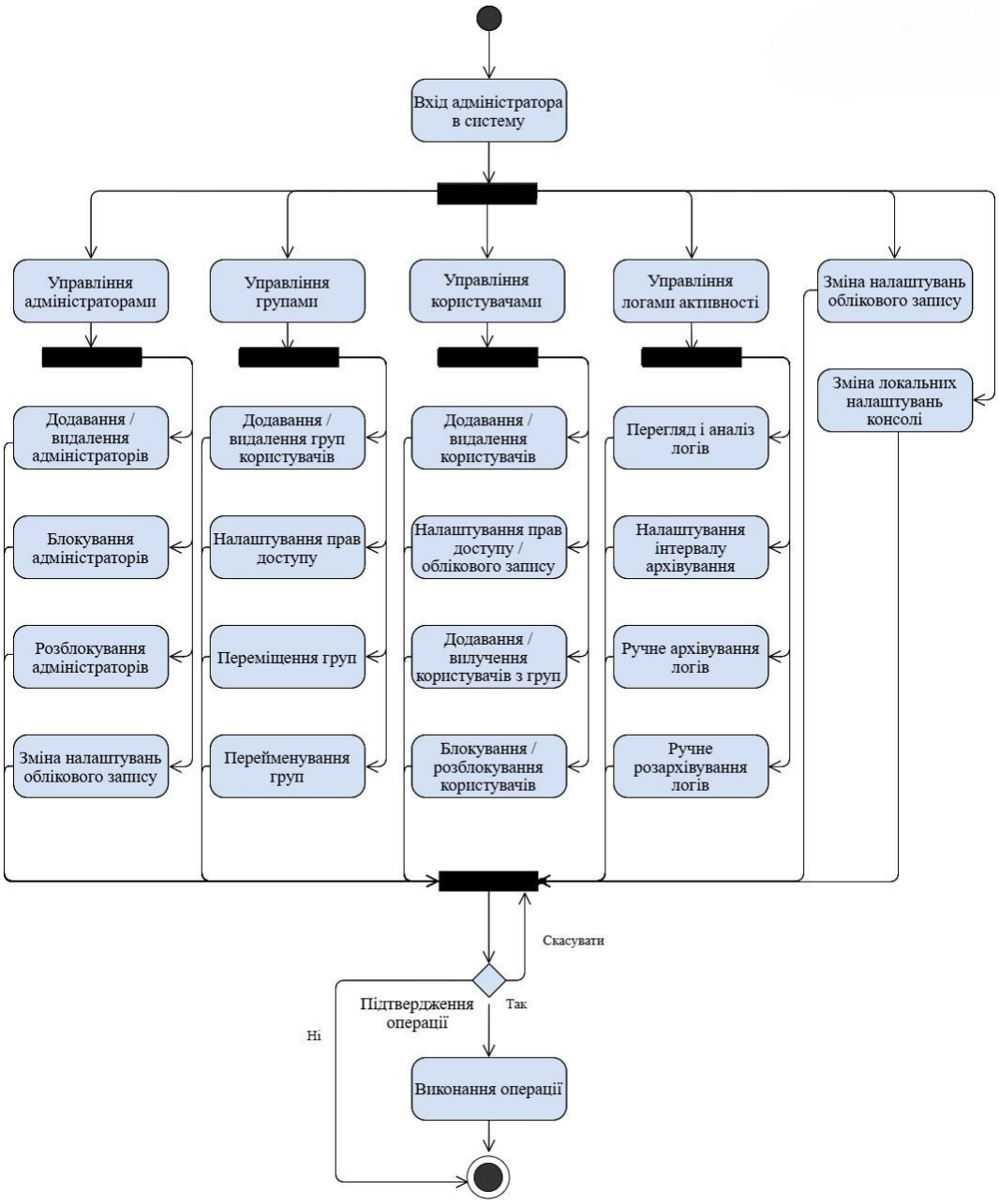


Рисунок 2.2 – UML-діаграма діяльності інструменту адміністрування «Admin Console»

UML-діаграма станів (State Machine Diagram) є графом станів і переходів. Машина станів містить стани, пов’язані між собою переходами. Кожен стан моделює період часу в житті об’єкта, протягом якого він задовольняє певним умовам. Коли відбувається подія, вона може викликати спрацьовування переходу, який переводить об’єкт у новий стан. Коли перехід спрацьовує, може бути виконана дія, пов’язана з цим переходом. Машини станів зображуються у вигляді діаграм станів (Statechart Diagram) [20]. UML-діаграма станів інструменту адміністрування «Admin Console» зображена на рис. 2.3.

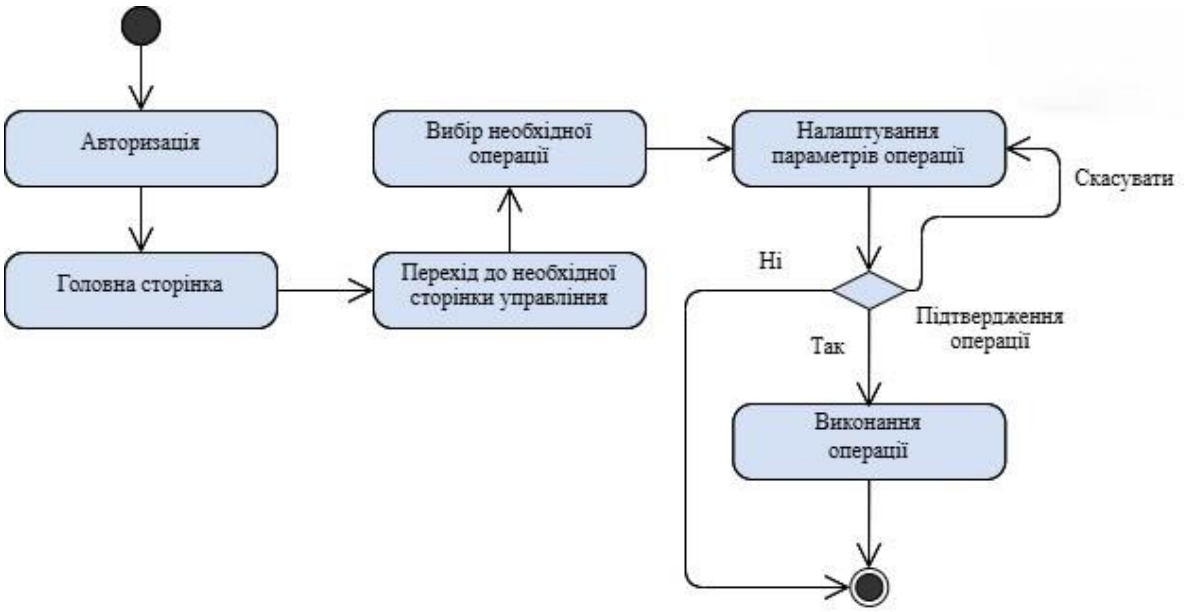


Рисунок 2.3 – UML-діаграма станів інструменту адміністрування «Admin Console»

UML-діаграма послідовності (Sequence Diagram) відображає взаємодію у вигляді двовимірної діаграми. Вертикальний вимір є віссю часу, який рухається вниз по сторінці. Горизонтальний вимір показує ролі класифікатора, які представляють окремі об’єкти у взаємодії. Кожна роль класифікатора представлена вертикальним стовпчиком – лінією життя. Повідомлення відображається у вигляді стрілки від лінії життя одного об’єкта до лінії життя іншого. Стрілки розташовані в часовій послідовності вниз по діаграмі [20].

Як один з двох видів діаграм взаємодії UML, діаграма послідовності (Sequence Diagram) показує взаємодію між об’єктами, розташованими в часовій

послідовності вертикального виміру, який рухається вниз по сторінці [25]. UML-діаграма послідовності інструменту адміністрування «Admin Console» зображена на рис. 2.4 (продовження діаграми послідовності на рис. 2.5).

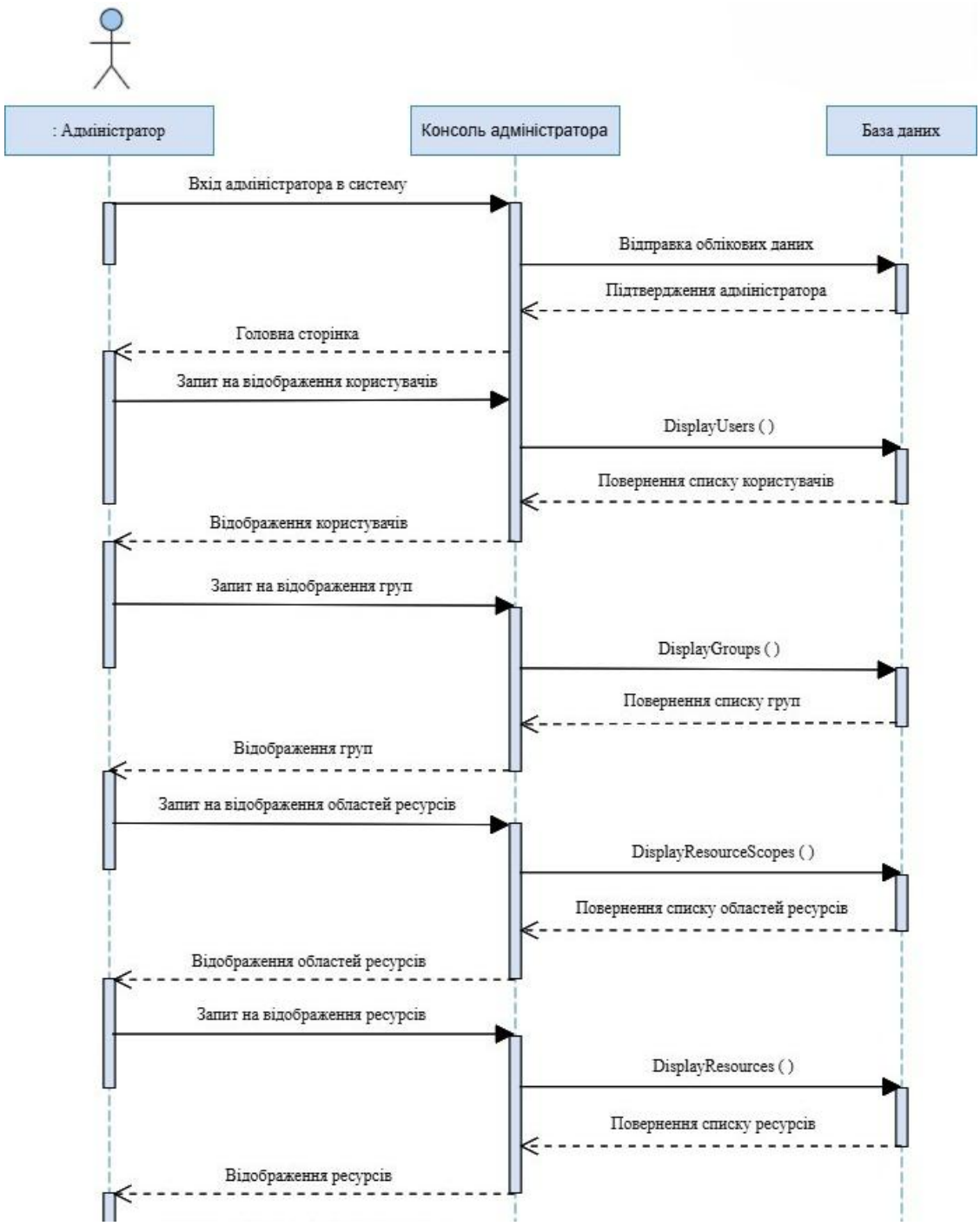


Рисунок 2.4 – UML-діаграма послідовності інструменту адміністрування «Admin Console»

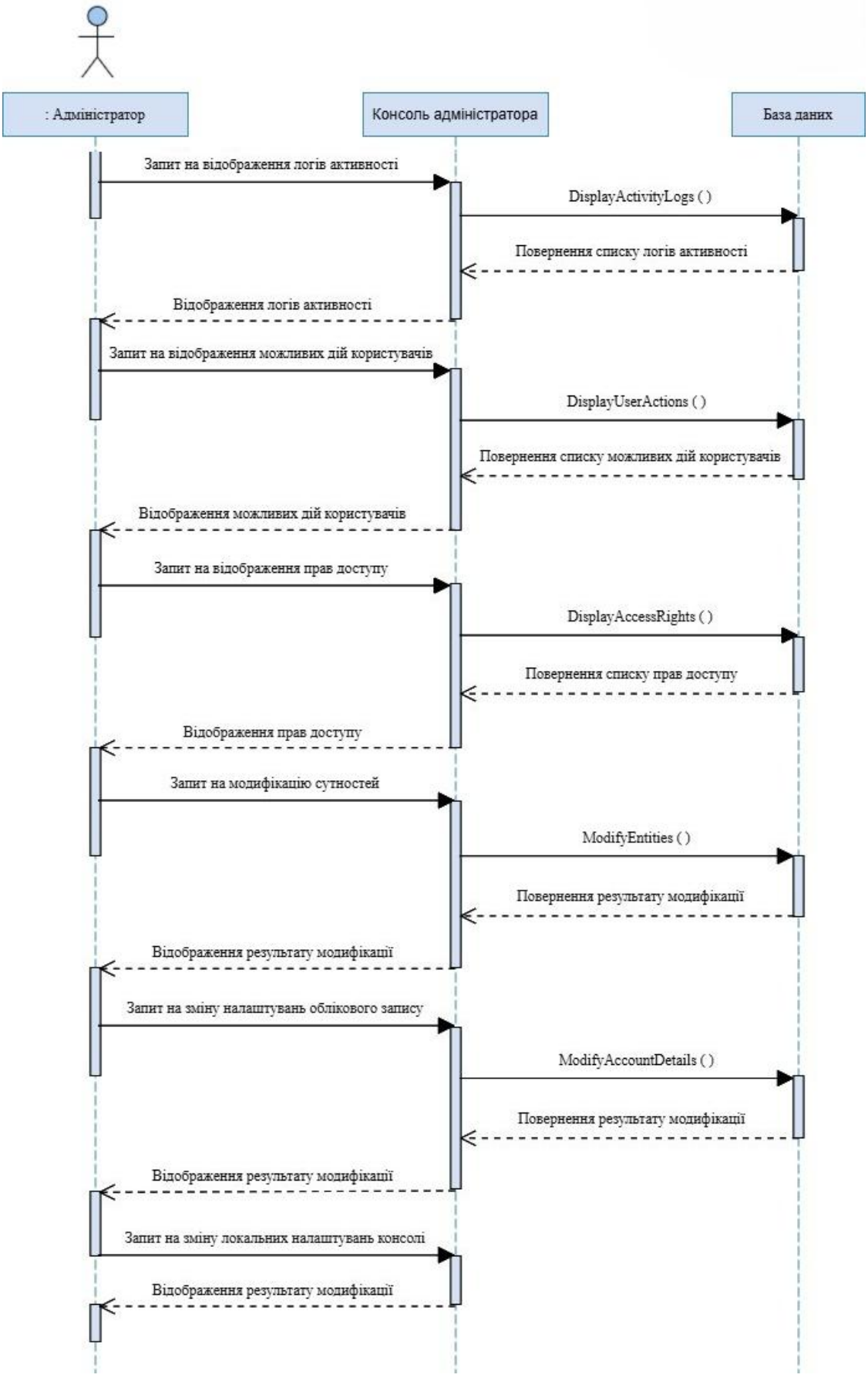


Рисунок 2.5 – Продовження UML-діаграми послідовності інструменту адміністрування «Admin Console»

Об’єктно-орієнтоване проектування забезпечує концептуальну і практичну основу для структурування програмних систем використовуючи фундаментальні принципи інкапсуляції, успадкування та поліморфізму. Комплексні системи використовують ці принципи для обмеження складності, забезпечення модульності та створення чітких абстракцій, які резонують з основними архітектурними патернами. Невід’ємна взаємодія між цими стовпами полегшує перетворення високорівневих концепцій проектування у надійні, підтримувані архітектури коду, а також слугує базовим механізмом для багатьох реалізацій патернів проектування [26].

Об’єктно-орієнтоване проектування полягає у процесі перетворення визначених вимог у специфікацію реалізації. Необхідно назвати об’єкти, визначити їхню поведінку та формально вказати, які об’єкти можуть активувати певну поведінку на інших об’єктах. Етап проектування полягає у перетворенні того, що має бути зроблено, у те, як це має бути зроблено. Результатом етапу проектування є специфікація реалізації. Виконання етапу проектування за один крок породжує процес перетворення вимог, визначених під час об’єктно-орієнтованого аналізу, на набір класів та інтерфейсів, які в подальшому в ідеалі можна реалізувати будь-якою об’єктно-орієнтованою мовою програмування.

Об’єктно-орієнтоване програмування є парадигмою програмування, яка розглядає програму як множину об’єктів, що взаємодіють між собою. Об’єктно-орієнтоване програмування супроводжується процесом перетворення проекту в робочу програму, відповідно до специфікації. Об’єкт виступає у ролі сукупності даних та пов’язаних з ними поведінкою. В об’єктно-орієнтованому моделюванні для позначення типу об’єкта використовується термін клас. Класи описують пов’язані між собою об’єкти. Вони схожі на шаблони для створення об’єктів. Дані представляють індивідуальні характеристики певного об’єкта, його поточний стан. Клас може визначати певні набори характеристик, які є частиною всіх об’єктів, що є членами цього класу. Будь-який конкретний об’єкт може мати різні значення даних для визначених характеристик. Поведінка – це дії, які можуть відбуватися над об’єктом. Поведінка, яка може бути виконана над

об’єктом певного класу, виражається у вигляді методів цього класу. На рівні програмування методи подібні до функцій у структурному програмуванні, але вони мають доступ до атрибутів, зокрема до змінних екземпляра з даними, пов’язаними з цим об’єктом. Як і функції, методи також можуть приймати параметри і повертати значення [27].

Об’єкти можуть взаємодіяти один з одним за чотирма принципами, такими як інкапсуляція, наслідування, поліморфізм та абстракція.

Принцип інкапсуляції є ключовим у підтримці цілісності стану об’єкта шляхом відокремлення даних об’єкта від зовнішніх маніпуляцій. Інкапсуляція полегшує визначення чітких інтерфейсів, тим самим мінімізуючи ризик небажаних побічних ефектів, які виникають внаслідок необмеженого впливу на стан. Вдосконалені реалізації забезпечують інкапсуляцію не лише за допомогою звичайних специфікаторів доступу, але й за допомогою динамічних перевірок під час виконання та валідації на основі декораторів.

Принцип наслідування (спадкування) – ще один наріжний камінь об’єктно-орієнтованого дизайну, який дозволяє формувати ієрархічні зв’язки, сприяючи повторному використанню коду та розширенню поведінки. Цей принцип дозволяє створювати нові класи на основі існуючих, фіксуючи спільну поведінку. Зазвичай успадкування застосовується в поєднанні з композицією, щоб уникнути глибоких і нестійких ієрархій. Цей механізм не лише відокремлює код від конкретних реалізацій, але й сприяє масштабованості, дозволяючи впроваджувати нову поведінку з мінімальними порушеннями.

Принцип поліморфізму проявляється як здатність замінювати різні конкретні реалізації в одному і тому ж операційному контексті. Ця здатність є важливою при застосуванні патернів проектування, таких як стратегія або шаблонний метод, які покладаються на абстрактні визначення, що дозволяють співіснувати декільком варіантам алгоритму. Абстракція, що забезпечується успадкуванням, дозволяє розширювати методи за допомогою перевизначення методів та обережного застосування віртуальних методів для зміни поведінки, зберігаючи при цьому існуючі інтерфейси [26].

Принцип абстракції полягає у тому, щоб представити об’єкт з мінімальним та в той же час необхідним набором атрибутів (властивостей) та методів. Абстракція дозволяє виконувати дії над об’єктами не вдаючись в особливості їх реалізації.

Завдяки підтримці принципу інкапсуляції в об’єктно-орієнтованому дизайні, доступ до змінних складних типів даних також спрощується. У програмі визначаються об’єкти (змінні) для відповідного класу (складного типу даних) і викликаються необхідні функції-члени з об’єктів, щоб отримати доступ до їхніх даних [28].

2.2 Проектування бази даних

СКБД MySQL – система керування реляційними базами даних, у якій дані зберігаються в окремих таблицях, що сприяє швидкодії та гнучкості. Зв'язок між таблицями реалізується за рахунок відносин, що сприяє реалізації складних запитів за участі даних з декількох пов'язаних між собою таблиць.

Методологія генерації розширеної моделі сутність-зв’язок з реляційної бази даних отримана завдяки аналізу схеми даних. Основна складність роботи з базами даних полягає у тому, що часто сенс даних втрачається, і ніхто не знає, що саме являють собою дані та взаємозв’язки між ними. Таке нерозуміння перешкоджає ефективному використанню даних в системі, а також знижує ймовірність правильного опрацювання даних. Такого розуміння можна досягти, лише піднявши рівень абстракції вище рівня самої бази даних і представивши її у вигляді концептуальної моделі. Оскільки EERM є найбільш широко використовуваними концептуальними моделями, достатньо згенерувати EERM з існуючої реляційної бази даних. Для цього застосовуються концепції зворотної інженерії програмного забезпечення.

Фундаментальними моделюючими конструкціями є сутності, зв’язки та пов’язані з ними атрибути. Атрибути представляють властивості, що описують сутності та зв’язки. Деякі атрибути можуть бути ідентифікаторами (ключами).

Ідентифікатори формують набір потенційних ключів для типу сутності, один з яких призначається первинним ключем. Оскільки тип зв’язку пов’язаний з декількома типами сутностей, первинний ключ типу зв’язку може бути сформований з первинних ключів типів сутностей, що беруть участь у ньому. Інші атрибути є дескрипторами, які описують властивості сутності або виникнення зв’язку.

Сутності за схожими ознаками зібрані в типи сутностей. Кожен тип сутності описується назвою та набором атрибутів. Сутності одного типу мають спільні атрибути. Сутності можна розділити на два типи: сильні сутності та слабкі сутності, виходячи з сили їхніх ідентифікаційних атрибутів. Сильні сутності мають внутрішні ідентифікатори, які однозначно визначають існування входжень сутності. Слабкі сутності не можуть бути однозначно ідентифіковані за значеннями їхніх власних атрибутів. Вони отримують свою унікальну ідентифікацію з ідентифікаційних атрибутів інших сутностей, які називаються ідентифікуючими власниками. Зв’язок, який пов’язує слабку сутність з її ідентифікуючим власником, називається ідентифікуючим зв’язком.

Зв’язки представляють асоціації між сутностями. У кожному виникненні зв’язку мають бути задіяні щонайменше два екземпляри сутностей, які беруть участь у зв’язку. Сутності, які беруть участь у зв’язку, називаються сутностями-учасниками зв’язку. Подібно до типів сутностей, зв’язки за схожими ознаками групуються у тип зв’язку.

Відношення класифікується переважно на основі властивостей його первинного ключа, тобто порівняння його первинного ключа з ключами інших відношень. Атрибути спочатку класифікуються залежно від того, чи є вони частиною первинного ключа відношення. Потім, залежно від типу відношення, атрибути первинного ключа поділяються на три категорії. Атрибути не первинного ключа поділяються на дві категорії. Атрибути, що не є первинними ключами, можуть містити потенційні ключі.

Атрибутами первинного ключа (Primary Key Attributes, PKA) є атрибути первинних ключів сильних зв’язків між сутностями, атрибути первинних ключів

слабких зв’язків між сутностями та атрибути первинних ключів зв’язків між сутностями, які також є ключами інших зв’язків. Атрибутами відокремленого ключа (Dangling Key Attributes, DKA) є атрибути первинного ключа слабкого зв’язку між сутностями, які не з’являються як ключові атрибути інших зв’язків. Атрибутами загального ключа (General Key Attributes, GKA) є атрибути первинного ключа зв’язку, які не є ключовими атрибутами інших зв’язків. Атрибутами зовнішнього ключа (Foreign Key Attributes, FKA) є підмножина атрибутів не первинного ключа даного відношення, яка з’являється як ключ іншого відношення сутності (сильний або слабкий). Неключовими атрибутами (Non Key Attributes, NKA) є решта всіх інших атрибутів [29].

Виділяють три типи зв’язків між сутностями: один-до-одного (1:1), один-до-багатьох (1:M) та багато-до-багатьох (M:M). У випадку, коли на кожен екземпляр сутності припадає один екземпляр іншої сутності, прийнято вважати, що між цими сутностями існує зв’язок один-до-одного (1:1), який позначається на EERM лінією з невеликою межею з обох кінців. Якщо на кожен екземпляр сутності припадає нуль, один або декілька екземплярів іншої сутності, прийнято вважати, що між цими сутностями існує зв’язок один-до-багатьох (1:M), який позначається на EERM лінією з невеликою межею з одного кінця (1:) та стрілкою з іншого (:M). У випадку, коли кілька екземплярів сутності з’єднані з декількома екземплярами іншої сутності, прийнято вважати, що між цими сутностями існує зв’язок багато-до-багатьох (M:M), який реалізується на EERM створивши асоціативну сутність між двома таблицями та вказавши зв’язок типу один-до-багатьох (1:M) між сусідньою та асоціативною сутностями відповідно.

Спроектована реляційна база даних містить наступні таблиці: користувачі, адміністратори, групи користувачів, учасники груп, області ресурсів, ресурси, модулі програмного забезпечення, можливі дії користувачів, права доступу до ресурсів, розширені права доступу до ресурсів, журнал авторизації та відмічені адміністратором. На рис. 2.6 зображено розширену модель сутність-зв’язок інструменту адміністрування «Admin Console».

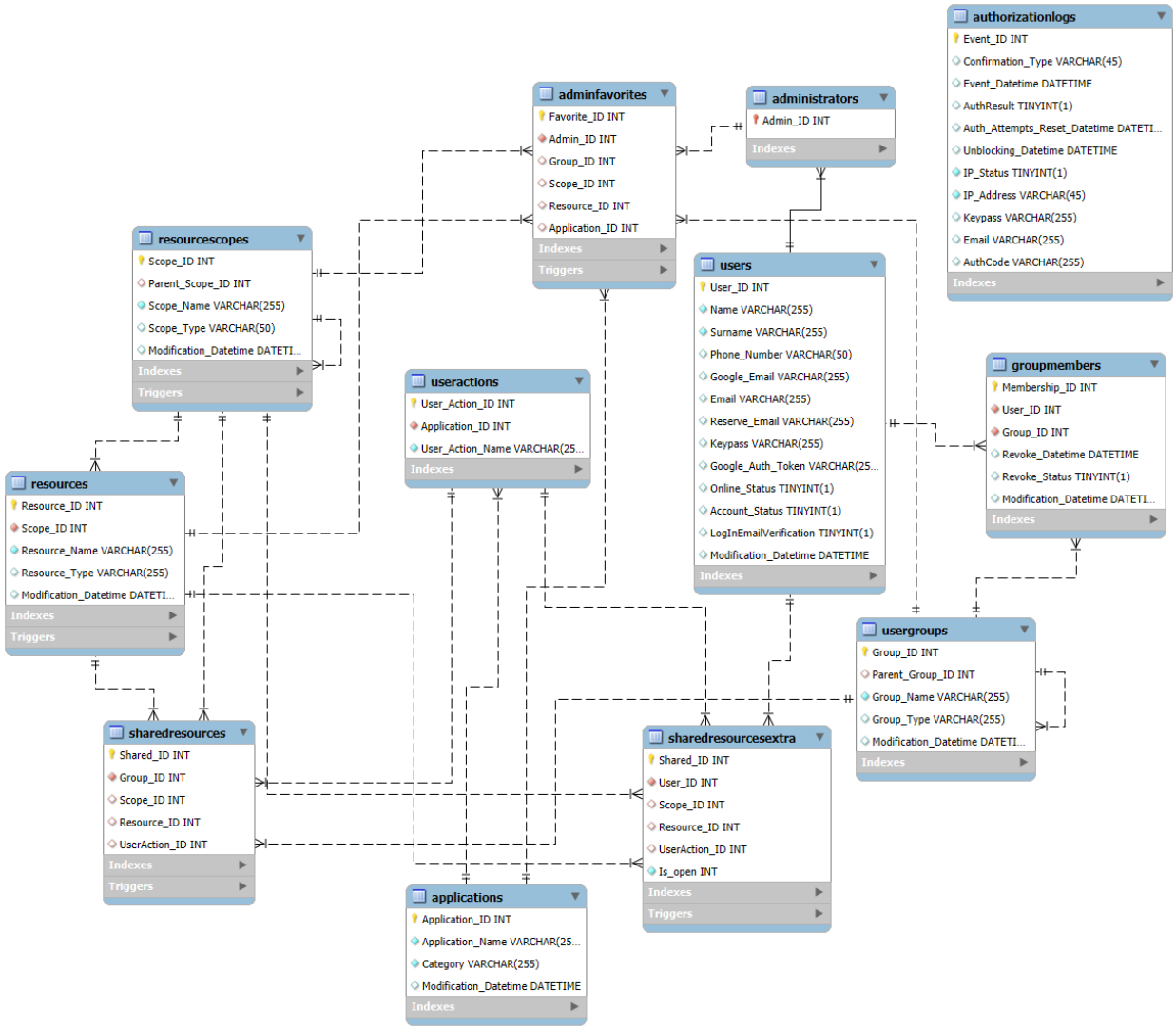


Рисунок 2.6 – Розширена модель сутність-зв’язок інструменту адміністрування «Admin Console»

В табл. 2.1 описано структуру таблиці користувачів (users), яка має зв’язок типу один-до-багатьох (1:M) з таблицею адміністраторів для виділення їх до таблиці з метою ефективного управління ними та відміченими сутностями.

Таблиця 2.1

Структура таблиці користувачів

Назва	Тип даних	Опис
1	2	3
User_ID	INT	Первинний ключ, унікальний ІД користувача

Продовження табл. 2.1

1	2	3
Name	VARCHAR	Ім’я користувача
Surname	VARCHAR	Прізвище
Phone_Number	VARCHAR	Номер телефону
Google_Email	VARCHAR	Електронна пошта в домені Google
Email	VARCHAR	Основна електронна пошта
Reserve_Email	VARCHAR	Резервна електронна пошта
Keypass	VARCHAR	Хеш ключа-пароля з сіллю
Google_Auth_Token	VARCHAR	Токен доступу OAuth 2.0
Online_Status	TINYINT	Статус в мережі
Account_Status	TINYINT	Статус облікового запису
LogInEmailVerification	TINYINT	Статус застосування підтвердження електронної пошти через OTP
Modification_Datetime	DATETIME	Час модифікації

Таблиця користувачів має зв’язок типу один-до-багатьох (1:M) з таблицею розширених прав доступу до ресурсів (sharedresourcesextra) з метою зберігання прав доступу, які безпосередньо налаштовані для користувачів. В табл. 2.2 описано структуру таблиці адміністраторів (administrators).

Таблиця 2.2

Структура таблиці адміністраторів

Назва	**Тип даних**	**Опис**
Admin_ID	INT	Первинний ключ, унікальний ІД адміністратора

Таблиця адміністраторів (administrators) має зв’язок типу багато-до-одного (M:1) з таблицею користувачів (users) для виділення перших до окремої таблиці з метою ефективного управління адміністраторами та їхніми відміченими

сутностями за допомогою застосування зв’язку типу один-до-багатьох (1:M) з таблицею відмічені адміністратором (adminfavorites). В табл. 2.3 описано структуру таблиці груп користувачів (usergroups).

Таблиця 2.3

Структура таблиці груп користувачів

Назва	**Тип даних**	**Опис**
Group_ID	INT	Первинний ключ, унікальний ідентифікатор групи
Parent_Group_ID	INT	Ідентифікатор батьківської групи
Group_Name	VARCHAR	Ім’я групи
Group_Type	VARCHAR	Тип групи
Modification_Datetime	DATETIME	Час модифікації

Таблиця груп користувачів (usergroups) має зв’язок типу один-до-багатьох (1:M) з таблицею учасників груп з метою забезпечення ефективного управління користувачами попередньо додавши їх до відповідних груп. Таблиця груп користувачів має зв’язок типу один-до-багатьох (1:M) з таблицею прав доступу до ресурсів з метою зберігання прав доступу, які безпосередньо налаштовані для груп користувачів. Таблиця груп користувачів має зв’язок типу один-до-багатьох (1:M) з таблицею відмічені адміністратором з метою ефективного контролю відміченими групами користувачів. Також таблиця груп користувачів посилається на себе, реалізуючи рекурсивний зв’язок типу багато-до-одного (M:1), з метою ефективного управління групами користувачів різних рівнів вкладеності. В табл. 2.4 описано структуру таблиці учасників груп (groupmembers).

Таблиця 2.4

Структура таблиці учасників груп

Назва	**Тип даних**	**Опис**
1	2	3
Membership_ID	INT	Первинний ключ, унікальний ІД членства

Продовження табл. 2.4

1	2	3
User_ID	INT	Ідентифікатор користувача
Group_ID	INT	Ідентифікатор групи
Revoke_Datetime	DATETIME	Час відкликання користувача
Revoke_Status	TINYINT	Статус відкликання користувача
Modification_Datetime	DATETIME	Час модифікації

Таблиця учасників груп (groupmembers) має зв’язок типу багато-до-одного (M:1) з таблицями користувачів (users) та груп користувачів (usergroups) з метою забезпечення ефективного управління користувачами попередньо додавши їх до відповідних груп. В табл. 2.5 описано структуру таблиці областей ресурсів (resourcescopes).

Таблиця 2.5

Структура таблиці областей ресурсів

Назва	**Тип даних**	**Опис**
Scope_ID	INT	Первинний ключ, унікальний ідентифікатор області ресурсів
Parent_Scope_ID	INT	Ідентифікатор батьківської області
Scope_Name	VARCHAR	Ім’я області
Scope_Type	VARCHAR	Тип області
Modification_Datetime	DATETIME	Час модифікації

Таблиця областей ресурсів має зв’язок типу один-до-багатьох (1:M) з таблицею ресурсів з метою забезпечення ефективного управління ресурсами попередньо розподіливши їх по відповідним областям ресурсів. Таблиця областей ресурсів має зв’язок типу один-до-багатьох (1:M) з таблицею прав доступу до ресурсів з метою зберігання прав доступу, які безпосередньо налаштовані для груп користувачів. Таблиця областей ресурсів має зв’язок типу один-до-багатьох (1:M) з таблицею розширених прав доступу до ресурсів з

метою зберігання прав доступу, які безпосередньо налаштовані для користувачів. Таблиця областей ресурсів має зв’язок типу один-до-багатьох (1:M) з таблицею відмічені адміністратором з метою ефективного контролю відміченими областями ресурсів. Також таблиця областей ресурсів посилається на себе, реалізуючи рекурсивний зв’язок типу багато-до-одного (M:1), з метою забезпечення ефективного управління областями ресурсів різних рівнів вкладеності. В табл. 2.6 описано структуру таблиці ресурсів (resources).

Таблиця 2.6

Структура таблиці ресурсів

Назва	**Тип даних**	**Опис**
Resource_ID	INT	Первинний ключ, унікальний ідентифікатор ресурсу
Scope_ID	INT	Ідентифікатор області ресурсів
Resource_Name	VARCHAR	Ім’я області
Resource_Type	VARCHAR	Тип області
Modification_Datetime	DATETIME	Час модифікації

Таблиця ресурсів (resources) має зв’язок типу багато-до-одного (M:1) з таблицею областей ресурсів (resourcescopes) з метою забезпечення ефективного управління ресурсами попередньо розподіливши їх по відповідним областям ресурсів. Таблиця ресурсів (resources) має зв’язок типу один-до-багатьох (1:M) з таблицею прав доступу до ресурсів (sharedresources) з метою зберігання прав доступу, які безпосередньо налаштовані для груп користувачів. Таблиця ресурсів (resources) має зв’язок типу один-до-багатьох (1:M) з таблицею розширених прав доступу до ресурсів (sharedresourcesextra) з метою зберігання прав доступу, які безпосередньо налаштовані для користувачів. Таблиця ресурсів (resources) має зв’язок типу один-до-багатьох (1:M) з таблицею відмічені адміністратором (adminfavorites) з метою ефективного контролю відміченими ресурсами. В табл. 2.7 описано структуру таблиці модулів програмного забезпечення (applications).

Таблиця 2.7

Структура таблиці модулів програмного забезпечення

Назва	**Тип даних**	**Опис**
Application_ID	INT	Первинний ключ, унікальний ідентифікатор модуля
Application_Name	VARCHAR	Ім’я модуля програмного забезпечення
Category	VARCHAR	Категорія
Modification_Datetime	DATETIME	Час модифікації

Таблиця модулів програмного забезпечення (applications) має зв’язок типу один-до-багатьох (1:M) з таблицею можливих дій користувачів (useractions) з метою зв’язування можливих дій користувачів з модулями програмного забезпечення та зв’язок типу один-до-багатьох (1:M) з таблицею відмічені адміністратором (adminfavorites) для зберігання відмічених сутностей. В табл. 2.8 описано структуру таблиці можливих дій користувачів (useractions).

Таблиця 2.8

Структура таблиці можливих дій користувачів

Назва	**Тип даних**	**Опис**
User_Action_ID	INT	Первинний ключ, унікальний ІД дії
Application_ID	INT	Ідентифікатор модуля програмного забезпечення
User_Action_Name	VARCHAR	Ім’я можливої дії користувача

Таблиця можливих дій користувачів має зв’язок типу багато-до-одного (M:1) з таблицею модулів програмного забезпечення з метою зв’язування можливих дій користувачів з модулями програмного забезпечення. Таблиця можливих дій користувачів має зв’язок типу один-до-багатьох (1:M) з таблицею прав доступу до ресурсів з метою зберігання прав доступу, які безпосередньо налаштовані для груп користувачів. Таблиця можливих дій користувачів має

зв’язок типу один-до-багатьох (1:M) з таблицею розширених прав доступу до ресурсів з метою зберігання прав доступу, які безпосередньо налаштовані для користувачів. В табл. 2.9 описано структуру таблиці прав доступу до ресурсів (sharedresources).

Таблиця 2.9

Структура таблиці прав доступу до ресурсів

Назва	**Тип даних**	**Опис**
Shared_ID	INT	Первинний ключ, унікальний ідентифікатор права доступу
Group_ID	INT	Ідентифікатор групи користувачів
Scope_ID	INT	Ідентифікатор області ресурсів
Resource_ID	INT	Ідентифікатор ресурсу
UserAction_ID	INT	Ідентифікатор можливої дії користувача

Таблиця прав доступу до ресурсів (sharedresources) має зв’язок типу багато-до-одного (M:1) з таблицями груп користувачів (usergroups), областей ресурсів (resourcescopes), ресурсів (resources), можливих дій користувачів (useractions) з метою зберігання прав доступу, які безпосередньо налаштовані для груп користувачів. В табл. 2.10 описано структуру таблиці розширених прав доступу до ресурсів (sharedresourcesextra).

Таблиця 2.10

Структура таблиці розширених прав доступу до ресурсів

Назва	**Тип даних**	**Опис**
1	2	3
Shared_ID	INT	Первинний ключ, унікальний ідентифікатор права доступу
User_ID	INT	Ідентифікатор користувача

Продовження табл. 2.10

1	2	3
Scope_ID	INT	Ідентифікатор області ресурсів
Resource_ID	INT	Ідентифікатор ресурсу
UserAction_ID	INT	Ідентифікатор можливої дії користувача
Is_open	INT	Значення права доступу

Таблиця розширених прав доступу до ресурсів має зв’язок типу багато-до-одного (M:1) з таблицями користувачів, областей ресурсів, ресурсів та можливих дій користувачів для зберігання прав доступу, налаштованих для користувачів. В табл. 2.11 описано структуру таблиці журналу авторизації (authorizationlogs).

Таблиця 2.11

Структура таблиці журналу авторизації

Назва	**Тип даних**	**Опис**
Event_ID	INT	Первинний ключ, унікальний ІД події
Confirmation_Type	VARCHAR	Тип підтвердження
Event_Datetime	DATETIME	Час настання події
AuthResult	TINYINT	Результат
Auth_Attempts_Reset_ Datetime	DATETIME	Час скидання спроб
Unblocking_Datetime	DATETIME	Час розблокування користувача
Keypass	VARCHAR	Хеш ключа-пароля з сіллю
Email	VARCHAR	Електронна пошта
AuthCode	VARCHAR	Одноразовий пароль (OTP)

Таблиця журналу авторизації (authorizationlogs) є незалежною сутністю, яка використовується для зберігання відомостей журналу авторизації (authorizationlogs). В табл. 2.12 описано структуру таблиці відмічених адміністратором (adminfavorites).

Таблиця 2.12

Структура таблиці відмічених адміністратором

Назва	Тип даних	Опис
Favorite_ID	INT	Первинний ключ, унікальний ідентифікатор
Admin_ID	INT	Ідентифікатор адміністратора
Group_ID	INT	Ідентифікатор групи користувачів
Scope_ID	INT	Ідентифікатор області ресурсів
Resource_ID	INT	Ідентифікатор ресурсу
Application_ID	INT	Ідентифікатор модуля програмного забезпечення

Таблиця відмічених адміністратором (adminfavorites) має зв’язок типу багато-до-одного (M:1) з таблицями адміністраторів (administrators), групами користувачів (usergroups), областей ресурсів (resourcescopes), ресурсів (resources), модулів програмного забезпечення (applications) з метою управління відміченими сутностями.

2.3 Аналіз наявних рішень для розподілу ресурсів і керування правами доступу

Google Cloud Identity and Access Management – рішення для детального контроль доступу та видимістю для централізованого керування хмарними ресурсами. Керування ідентифікацією та доступом дозволяє адміністраторам налаштовувати, хто може виконувати дії з певними ресурсами, надаючи повний контроль і видимість для централізованого керування ресурсами Google Cloud. Google Cloud IAM надає інструменти для управління дозволами на ресурси з високим рівнем автоматизації. Користувачі отримують доступ лише до того, що їм потрібно для виконання роботи, а адміністратори можуть легко надавати дозволи за замовчуванням цілим групам користувачів.

Google Cloud IAM дає змогу надавати доступ до хмарних ресурсів на детальному рівні, що виходять далеко за межі доступу на рівні проекту. Google Cloud IAM дозволяє налаштовувати більш деталізовані політики контролю доступу до ресурсів на основі таких атрибутів, як статус безпеки пристрою, IP-адреса, тип ресурсу, дата/час. Ці політики допомагають гарантувати, що під час надання доступу до хмарних ресурсів застосовуються належні засоби контролю безпеки. Адміністратори можуть переглядати повну історію авторизації, видалення та делегування дозволів [30].

Надання прав доступу в Google Cloud IAM включає три компоненти:

- Принципал – ідентифікатор особи або системи;
- Роль – набір дозволів, які потрібно надати принципалу;
- Ресурс – ресурс Google Cloud, який необхідно надати принципалу.

Щоб надати принципалу дозвіл на доступ до ресурсу, потрібно призначити йому пов’язану роль, пов’язану з ресурсом, за допомогою політики дозволів [31].

Azure Active Directory – хмарне рішення для управління ідентифікацією та доступом від корпорації Майкрософт. Azure AD є основою системи Office 365, вона може синхронізуватися з локальною Active Directory і забезпечувати аутентифікацію в інших хмарних системах за допомогою OAuth. Azure AD використовує хмарні протоколи аутентифікації, такі як OAuth2, SAML і WS-Security, для аутентифікації користувачів.

Кожен екземпляр Azure AD називається "орендарем", який являє собою плоску структуру користувачів і груп. Адміністратори об’єднують користувачів у групи, а потім надають групам доступ до програм і ресурсів. Azure AD використовує служби доменів Azure AD для керування серверами, які працюють у хмарному середовищі віртуальних машин Azure. Azure AD є новою системою, яку Microsoft розробила з нуля для підтримки хмарної інфраструктури. Azure AD використовує REST API для передачі даних з однієї системи до інших хмарних додатків і систем, які підтримують REST.

Користувачі та групи є основними будівельними блоками Azure AD. Користувачі розподіляються по групам, за схожими ознаками. Можна об’єднати

команду управління в одну групу Azure AD і надати дозволи на рівні групи, тож коли користувачі звільняються з організації, потрібно буде деактивувати лише один обліковий запис, а решта групи залишиться незмінною [32].

Висновки за розділом 2

У другому розділі було розглянуто моделювання інструменту адміністрування для розподілу ресурсів і керування правами доступу «Admin Console» за допомогою уніфікованої мови моделювання – UML, зокрема наведено: UML-діаграму варіантів використання (Use Case Diagram) інструменту адміністрування «Admin Console», UML-діаграму діяльності (Activity Diagram), UML-діаграму станів (State Machine Diagram) та UML-діаграму послідовності (Sequence Diagram).

Проведено проектування бази даних інструменту адміністрування для розподілу ресурсів і керування правами доступу «Admin Console». Згенеровано EERM, яка точно представляє вимоги складних баз даних і дозволяє відобразити сутності, їх атрибути та зв'язки, які представляють асоціації між сутностями. Методологія генерації EERM з реляційної бази даних отримана завдяки аналізу схеми даних.

Використано СКБД MySQL – систему керування реляційними базами даних, у якій дані зберігаються в окремих таблицях, що сприяє швидкодії та гнучкості. Зв'язок між таблицями реалізується через відносини, що сприяє реалізації складних запитів за участі декількох пов'язаних між собою таблиць.

Описано структуру наступних таблиць реляційної бази даних: користувачі, адміністратори, групи користувачів, учасники груп, області ресурсів, ресурси, модулі програмного забезпечення, можливі дії користувачів, права доступу до ресурсів, розширені права доступу до ресурсів, журнал авторизації та відмічені адміністратором.

Проведено аналіз наявних рішень для розподілу ресурсів і керування правами доступу.

РОЗДІЛ 3
РОЗРОБКА ІНСТРУМЕНТУ АДМІНІСТРУВАННЯ ДЛЯ РОЗПОДІЛУ РЕСУРСІВ І КЕРУВАННЯ ПРАВАМИ ДОСТУПУ

3.1 Архітектура інструменту адміністрування «Admin Console»

Клієнт-сервер – це модель архітектури системи, що складається з двох частин, клієнтської та серверної систем, які взаємодіють через комп'ютерну мережу. Клієнт-серверний додаток – це категорія розподіленої системи, що складається з клієнтського та серверного програмного забезпечення. Клієнт-серверний додаток забезпечує розширений спосіб розподілу робочого навантаження.

Клієнтський процес постійно встановлює з'єднання з сервером, тоді як серверний процес постійно очікує на запити від будь-якого клієнта, програмного забезпечення, який отримує доступ до сервісу, що надається сервером. На стороні сервера встановлене спеціальне програмне забезпечення, призначене для надання послуг та ресурсів для задоволення потреб клієнтів. Залежно від виконуваного сервісу, це може бути файловий сервер, сервер баз даних, домашній медіа-сервер, принт-сервер, веб-сервер або хмарні сервери, на яких розміщуються віртуальні машини.

Клієнт-серверна архітектура поділяється на чотири типи: однорівнева, дворівнева, трирівнева та N-рівнева архітектура. Однорівневий (автономний) додаток має всі рівні, такі як рівень представлення, бізнес-рівень та рівень доступу до даних, в одному програмному комплексі. Дворівнева архітектура додатку поділяється на дві частини: клієнтську програму (клієнтський рівень) і базу даних (рівень даних). Клієнтська система працює на рівні представлення та рівні додатків, а серверна система обробляє рівень бази даних. Вона також відома як клієнт-серверний додаток. Взаємодія відбувається між клієнтом і сервером. Клієнт надсилає запит на сервер, а сервер обробляє запит і повертає дані запиту

клієнту. Трирівнева архітектура, відома як архітектура веб-додатків, поділяється на три частини: рівень представлення (клієнтський рівень), рівень додатку (бізнес-рівень) і рівень бази даних (рівень даних). Клієнтська система керує рівнем представлення, сервер додатків керує рівнем додатків, а система сервера баз даних керує рівнем баз даних. N-рівнева архітектура характеризується наявністю N-рівнів додатку. N-рівневий додаток розподіляє систему по принципу, схожим на трирівневу архітектуру, але кількість серверів додатків збільшена і представлена на окремих рівнях, для декомпозиції бізнес-логіки таким чином, щоб логіка була розподіленою [33].

Клієнт-серверна архітектура зосереджена навколо циклу запит-відповідь. Клієнт надсилає запит на сервер для отримання даних або послуги. Сервер отримує запит, обробляє його, формуючи відповідь. Завершивши формування відповіді, сервер надсилає оброблені дані або сервіс у відповідь клієнту. В дворівневій архітектурі клієнт взаємодіє з сервером. На сервері розміщується база даних і бізнес-логіка, де він безпосередньо відповідає на запити клієнта. Трирівнева архітектура додає проміжний рівень (бізнес-рівень) між клієнтом та сервером, що сприяє розподіленню навантаження та підвищення гнучкості і масштабованості. N-рівнена архітектура розширяє трирівневу архітектуру, виділяючи окремий рівень на відповідний набір функцій, що дозволяє забезпечити принцип модульності, масштабованості та розподілити навантаження між декількома рівнями. Трирівнева клієнт-серверна архітектура інструменту адміністрування «Admin Console» зображена на рис. 3.1.

Рисунок 3.1 – Трирівнева клієнт-серверна архітектура інструменту адміністрування «Admin Console»

3.2 Опис гібридної моделі управління доступом

Контроль доступу до ресурсів і сервісів є фундаментальною основою безпеки. Протягом багатьох років було розроблено безліч моделей контролю доступу, кожна з яких призначена для вирішення різних аспектів цієї проблеми [2].

Перспективи розвитку виражаються у створенні гібридних моделей управління доступом та нових рішень з контролю доступу для наступних сфер: хмарні обчислення, IoT, блокчейн, мобільне хмарне середовище, інтелектуальні екосистеми спільної роботи, штучний інтелект, обмін даними на смарт-пристроях та розподілених базах даних [3].

За основу гібридної моделі управління доступом, обрано модель контролю доступу на основі груп, додаючи принцип розподілу прав доступу до ресурсів на основі шаблонів прав доступу.

Контроль доступу на основі груп – модель безпеки, яка регулює доступ до ресурсів шляхом призначення дозволів на основі членства користувача в групі. Контроль доступу на основі груп спрощує управління дозволами, налаштовуючи доступ користувачам на основі їхньої приналежності до заздалегідь визначених груп, підвищуючи загальну безпеку системи та ефективність адміністрування.

Комбінуючи модель контролю доступу на основі груп з розподілом прав доступу до ресурсів на основі шаблонів прав доступу, досягається більша ефективність та гнучкість налаштування. Групам користувачів налаштовуються шаблони прав доступу в ручному режимі або в режимі копіювання наперед визначених прав доступу для інших груп. Додаючи користувачів до груп, можна швидко застосовувати для користувачів наперед визначений шаблон прав доступу групи, в режимі перезаписування або додавання до наявних прав доступу. Також є можливість модифікації налаштованих прав доступу для користувачів та груп, копіюючи права доступу з інших шаблонів, керуючи рекурсивним наслідуванням прав доступу в режимі перезаписування або додавання до наявних прав доступу та налаштовуючи права доступу в ручному

режимі. Вибираючи сутності (користувачів або груп), можна одночасно налаштовувати права доступу одразу для кількох обраних сутностей, підвищуючи ефективність та зменшуючи час адміністрування.

Запропонована гібридна модель управління доступом підвищує ефективність та надає більшу гнучкість налаштування прав доступ.

3.3 Програмна реалізація інструменту адміністрування для розподілу ресурсів і керування правами доступу «Admin Console»

Qt – кросплатформне середовище розробки додатків для створення графічних інтерфейсів користувача, а також кросплатформних додатків, які працюють на різних програмних і апаратних платформах, таких як Linux, Windows, macOS, Android, залишаючись при цьому нативним додатком з нативними можливостями і швидкістю.

PySide – прив’язка мови Python кросплатформного інструментарію графічного інтерфейсу Qt, розроблена компанією The Qt Company, як частина проекту Qt for Python.

Модуль Qt Widgets надає набір елементів (віджетів) інтерфейсу користувача для створення класичних графічних інтерфейсів користувача у кросплатформному стилі. Віджети є основними елементами для створення інтерфейсів користувача в Qt. Віджети можуть відображати дані та інформацію про стан, отримувати дані від користувача та надавати контейнер для інших віджетів, які потрібно згрупувати разом.

Інструмент адміністрування спроектовано в ООП стилі, розроблено на мові програмування Python версії 3.0+ та розбито на окремі модулі, згруповані в пакети.

Розроблено наступні віджети пакету widgets:

- ActivityLogWidget.py – віджет подій в журналі логування при відображенні подій на сторінці головного вікна програми;

- AdministratorWidget.py – віджет адміністраторів при відображенні адміністраторів на сторінці головного вікна програми;
- DefaultGroupSelectionAddingWidget.py – віджет за замовчуванням груп користувачів при відображенні груп у діалоговому вікні додавання користувачів до відповідних груп;
- DefaultGroupSelectionMoveWidget.py – віджет за замовчуванням груп користувачів при відображенні груп у діалоговому вікні переміщення груп (зміна батьківської групи);
- DefaultGroupSelectionWidget.py – віджет за замовчуванням груп користувачів при відображенні груп у діалоговому вікні налаштування прав доступу;
- GroupSelectionAddingWidget.py – віджет груп користувачів при відображенні груп у діалоговому вікні додавання користувачів до відповідних груп;
- GroupSelectionMoveWidget.py – віджет груп користувачів при відображенні груп у діалоговому вікні переміщення груп (зміна батьківської групи);
- GroupSelectionWidget.py – віджет груп користувачів при відображенні груп у діалоговому вікні налаштування прав доступу;
- GroupWidget.py – віджет груп користувачів при відображенні груп на сторінці головного вікна програми;
- ListRecordWidget.py – віджет груп користувачів при відображенні груп у діалоговому вікні перегляду та зміни груп, до яких доданий користувач);
- ListSelectedUserWidget.py – віджет користувача при відображенні користувачів у діалоговому вікні додавання користувачів до відповідних груп;
- NavigationWidget.py – віджет навігації при відображенні відповідних віджетів на кожній сторінці;
- ResourceSelectionWidget.py – віджет ресурсу при відображенні ресурсів у діалоговому вікні налаштування прав доступу;

- ResourceWidget.py – віджет ресурсу при відображенні ресурсів на сторінці головного вікна програми;
- ScopeSelectionWidget.py – віджет області ресурсів при відображенні областей ресурсів у діалоговому вікні налаштування прав доступу;
- ScopeWidget.py – віджет області ресурсів при відображенні областей ресурсів на сторінці головного вікна програми;
- UserActionSelectionWidget.py – віджет набору можливих дії користувача при відображенні можливих дії користувача у прив’язці до модулів програмного забезпечення у діалоговому вікні налаштування прав доступу;
- UserActionWidget.py – віджет набору можливих дії користувача при відображенні можливих дії користувача у прив’язці до модулів програмного забезпечення на сторінці головного вікна програми;
- UserSelectionWidget.py – віджет користувача при відображенні користувачів у діалоговому вікні налаштування прав доступу та діалоговому вікні додаванні користувачів до відповідних груп;
- UserWidget.py – віджет користувача при відображенні користувачів на сторінці головного вікна програми.

Лістинг фрагмента Python3 коду віджета «AdministratorWidget» наведений в Додатку Б.

Розроблено наступні модулі пакету scripts:

- BrowseFilesFolderScript.py – модуль обробки вибору шляху до файлів та каталогів;
- ConfigScript.py – модуль обробки взаємодії з конфігураційними файлами налаштувань;
- ShowContextMenu.py – модуль обробки відображення контекстного меню;
- ShowContextMenuWithMemory.py – модуль обробки відображення контекстного меню з пам’яттю;
- ShowMessageBoxScript.py – модуль обробки відображення спливаючих повідомлень інформаційного, попереджувального та критичного характеру.

Лістинг фрагмента Python3 коду модуля «ShowMessageBoxScript» наведений в Додатку В.

Розроблено наступні модулі пакету gui:

- MainWindow.py – модуль обробки логіки головного вікна програми;
- MainWindowGUI.py – модуль визначення компонентів графічного інтерфейсу головного вікна програми;
- AccessRightsSelectGroupsUsersDialogGUI.py – модуль обробки логіки вибору груп та користувачів у діалоговому вікні налаштування прав доступу;
- AccessRightsSelectResourcesDialogGUI.py – модуль обробки логіки вибору ресурсів у діалоговому вікні налаштування прав доступу;
- AccountRegisterConfirmationGUI.py – модуль обробки логіки підтвердження електронної пошти через OTP у діалоговому вікні реєстрації нового облікового запису;
- AddNewGroupDialogGUI.py – модуль обробки логіки створення нової групи у відповідному діалоговому вікні;
- AddNewScopeDialogGUI.py – модуль обробки логіки створення нової області ресурсів у відповідному діалоговому вікні;
- AddUsersToGroupsDialogGUI.py – модуль обробки логіки додавання користувачів до груп у відповідному діалоговому вікні;
- ChangeEntityNameDialogGUI.py – модуль обробки логіки перейменування сутностей у відповідному діалоговому вікні;
- CreateAccountDialogGUI.py – модуль обробки логіки реєстрації нового облікового запису у відповідному діалоговому вікні;
- DialogDefaultPathForDialogsGUI.py – модуль обробки логіки вибору шляху за замовчуванням для діалогових вікон у відповідному діалоговому вікні;
- GroupMoveDialogGUI.py – модуль обробки логіки переміщення групи користувачів (зміна батьківської групи) у відповідному діалоговому вікні;
- LogInConfirmationGUI.py – модуль обробки логіки підтвердження електронної пошти через OTP у діалоговому вікні входу в обліковий запис;

- LogInGUI.py – модуль обробки логіки входу в обліковий запис у відповідному діалоговому вікні;
- LogsArchivingIntervalDialogGUI.py – модуль обробки логіки налаштування інтервалу архівування логів у відповідному діалоговому вікні;
- PasswordResetGUI.py – модуль обробки логіки відновлення паролю у відповідному діалоговому вікні;
- UserAccountDetailsDialogGUI.py – модуль обробки логіки редагування відомостей облікового запису користувача у відповідному діалоговому вікні;
- UserGroupsInfoDialogGUI.py – модуль обробки логіки перегляду та зміни груп, до яких доданий користувач у відповідному діалоговому вікні;
- UserRevokeTimeDialogGUI.py – модуль обробки логіки налаштування часу відкликання користувача з групи у відповідному діалоговому вікні.

Лістинг фрагмента Python3 коду модуля налаштування прав доступу «AccessRightsSelectGroupsUsersDialogGUI» наведений в Додатку Г.

Для взаємодії з розробленою реляційною базою даних, написано SQL-запити на отримання, оновлення та видалення записів до наступних таблиць: користувачі, адміністратори, групи користувачів, учасники груп, області ресурсів, ресурси, модулі програмного забезпечення, можливі дії користувачів, права доступу до ресурсів, розширені права доступу до ресурсів, журнал авторизації та відмічені адміністратором.

Лістинг фрагмента Python3 коду SQL-запиту на оновлення прав доступу до ресурсів до реляційної системи керування базами даних MySQL наведений в Додатку Д.

3.4 Демонстрація роботи інструменту адміністрування «Admin Console»

Інструмент адміністрування дозволяє додавати нових користувачів та адміністраторів у діалоговому вікні реєстрації нового облікового запису (рис. 3.2). Заповнивши необхідні відомості, необхідно підтвердити електронну

пошту через OTP у наступному діалоговому вікні для успішної реєстрації нового облікового запису.

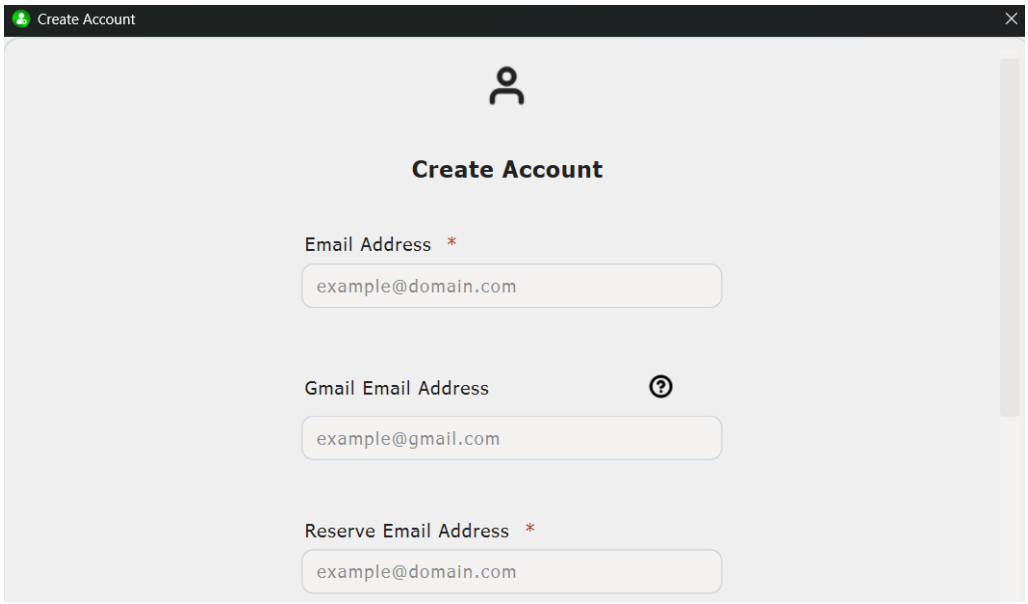


Рисунок 3.2 – Реєстрація нового облікового запису

Зареєстрований користувач може здійснити вхід в обліковий запис у діалоговому вікні входу в обліковий запис (рис. 3.3), заповнивши поля електронної пошти та відповідного паролю.

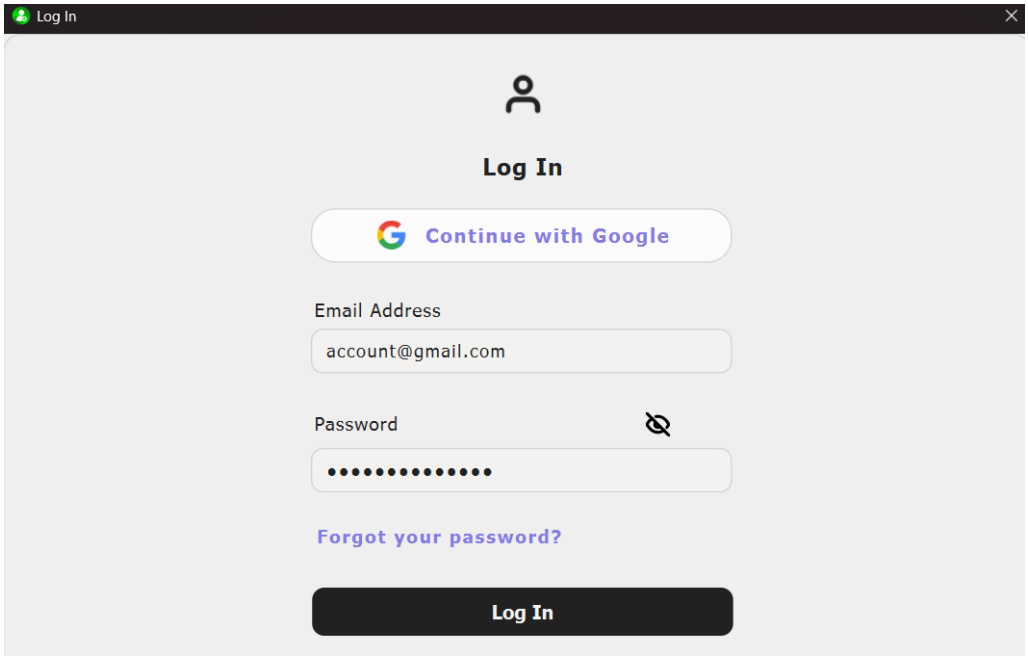


Рисунок 3.3 – Вхід в обліковий запис

Двофакторна аутентифікація потребує пройти підтвердження електронної пошти через OTP у наступному діалоговому вікні (рис. 3.4) для успішного входу в обліковий запис.

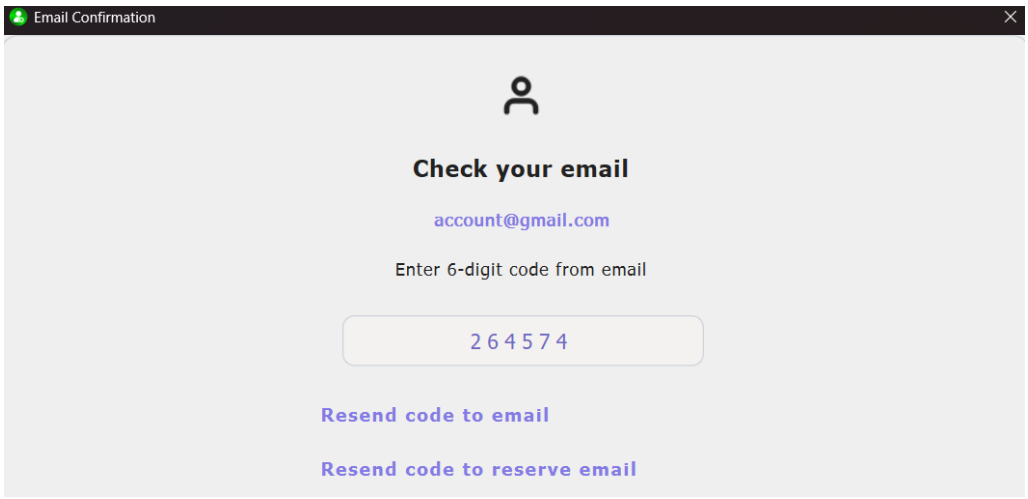


Рисунок 3.4 – Підтвердження електронної пошти через OTP

Головна сторінка інструменту адміністрування (рис. 3.5) відображає всіх зареєстрованих користувачів в системі.

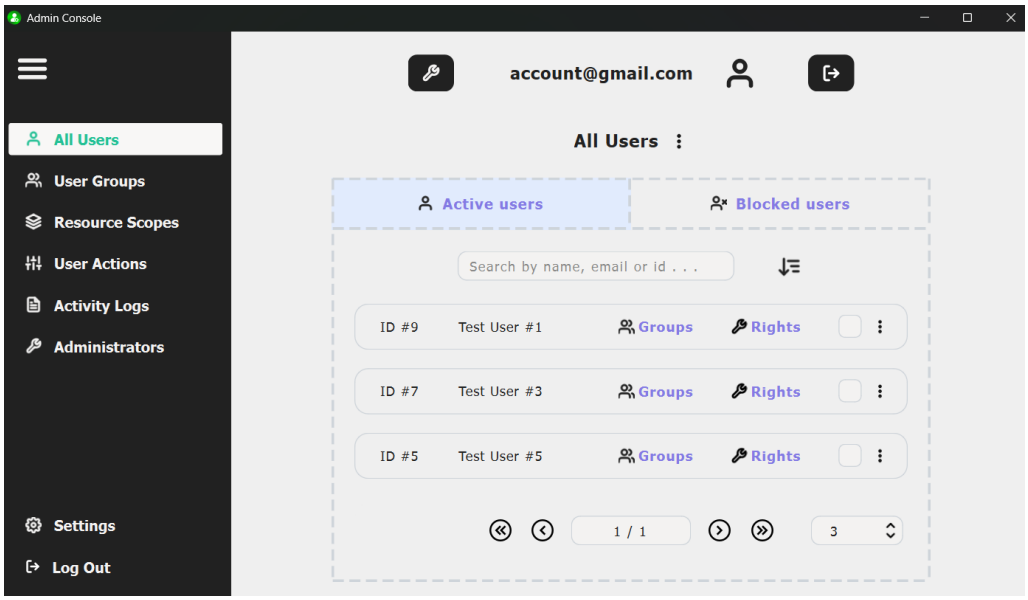


Рисунок 3.5 – Головна сторінка інструменту адміністрування «Admin Console»

Панель меню з лівого боку дозволяє вийти з облікового запису натиснувши кнопку «Log Out» та здійснювати перехід між сторінками: всіх користувачів (All

Users), груп користувачів (User Groups), областей ресурсів (Resource Scopes), можливими діями користувачів (User Actions), журналу подій (Activity Logs), адміністраторами (Administrators) та налаштувань інструменту адміністрування (Settings). Панель керування зверху дозволяє здійснити вихід з облікового запису, відкрити діалогове вікно одночасного налаштування вибраних сутностей та перейти до сторінки облікового запису адміністратора.

Управління групами здійснюється на сторінці груп користувачів (рис. 3.6). Для кожної групи можна налаштувати власний шаблон прав доступу. Є два типи груп користувачів: «Subgroups» та «Members». Група з типом «Subgroups» містить дочірні підгрупи, тоді як група з типом «Members» може містити лише користувачів, попередньо доданих до відповідної групи (рис. 3.8). Адміністратор може налаштовувати права доступу для користувачів вручну, надаючи доступ до областей ресурсів, самих ресурсів та дій користувача у прив’язці до розроблених модулів програмного забезпечення.

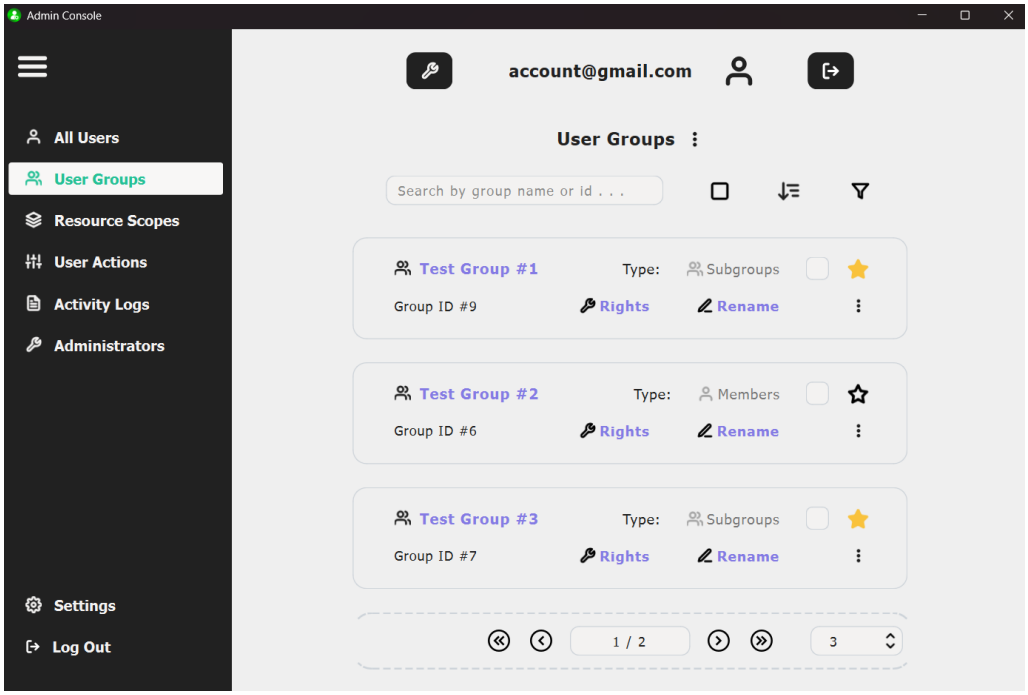


Рисунок 3.6 – Сторінка груп користувачів

Додаючи користувачів до груп, адміністратор може швидко застосовувати для користувачів наперед визначений шаблон прав доступу групи, в режимі

перезаписування або додавання до наявних прав доступу. Є можливість застосовувати для користувачів наперед визначений шаблон прав доступу групи під час одночасного налаштування прав доступу для вибраних користувачів або з випадаючого меню відповідної групи користувачів. Зберігається можливість налаштування часу відкликання членів груп (рис. 3.7).

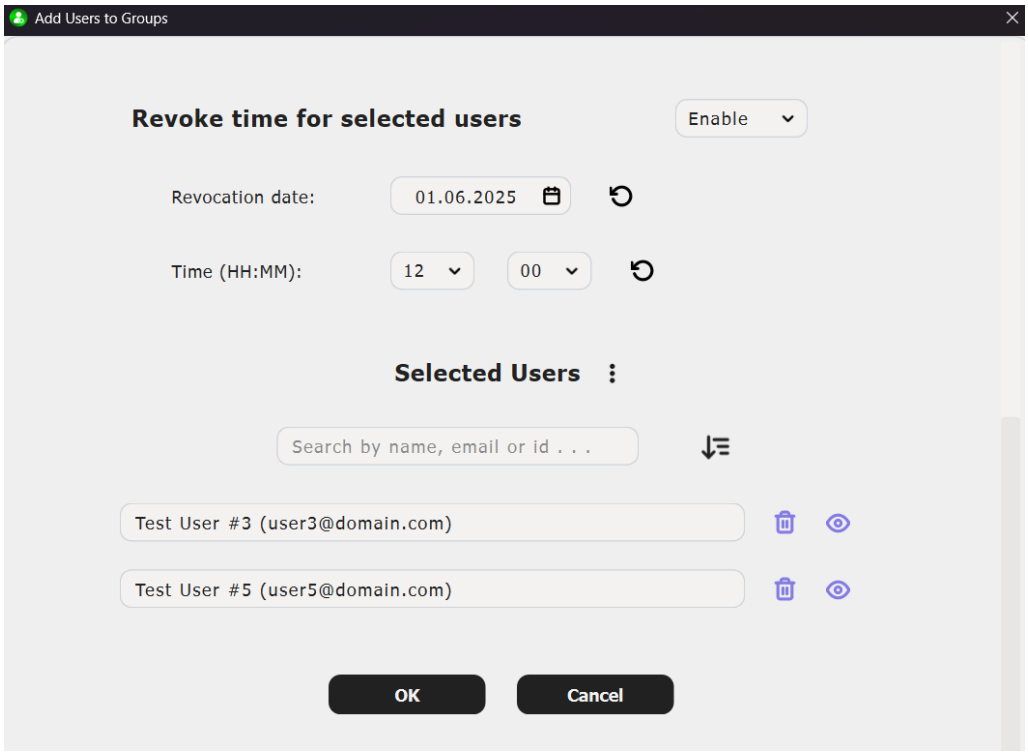


Рисунок 3.7 – Налаштування часу відкликання для учасників груп

Натискаючи на кнопку групи користувачів «Groups», можна переглянути всі групи, до яких доданий відповідний користувач. Натискаючи на кнопку права доступу «Rights», можна налаштувати права доступу для користувача або групи. Вибираючи сутності (користувачів або груп), можна одночасно налаштувати права доступу для вибраних сутностей.

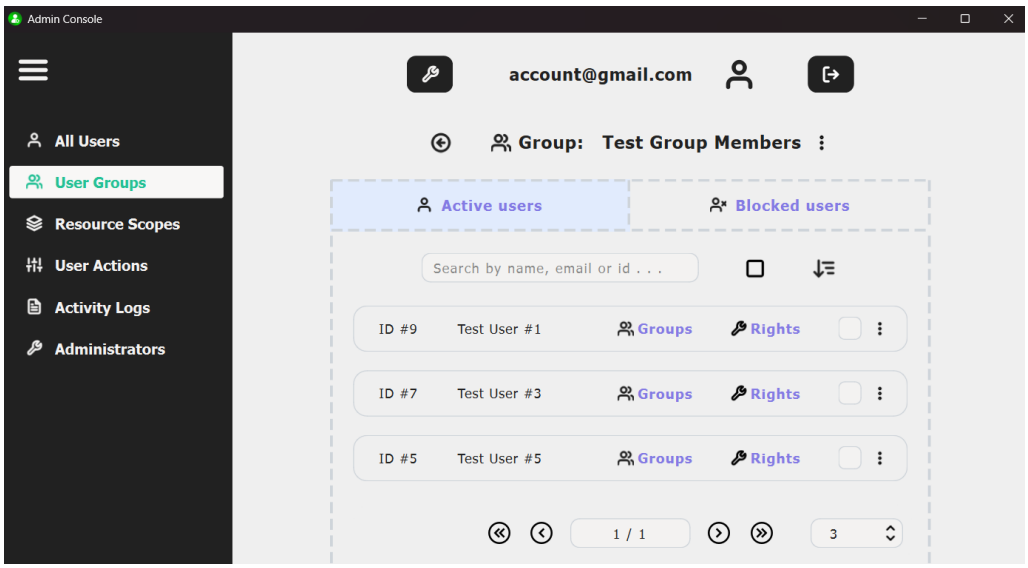


Рисунок 3.8 – Учасники групи з типом «Members»

Також є можливість налаштування прав доступу зі сторінки областей ресурсів (рис. 3.9) натискаючи на кнопку сутності «Entities» і вибираючи користувачів або груп у діалоговому вікні налаштування прав доступу. Можна вибирати декілька областей ресурсів та відкривати доступ до них обраним користувачам або групам у діалоговому вікні налаштування прав доступу (рис. 3.10).

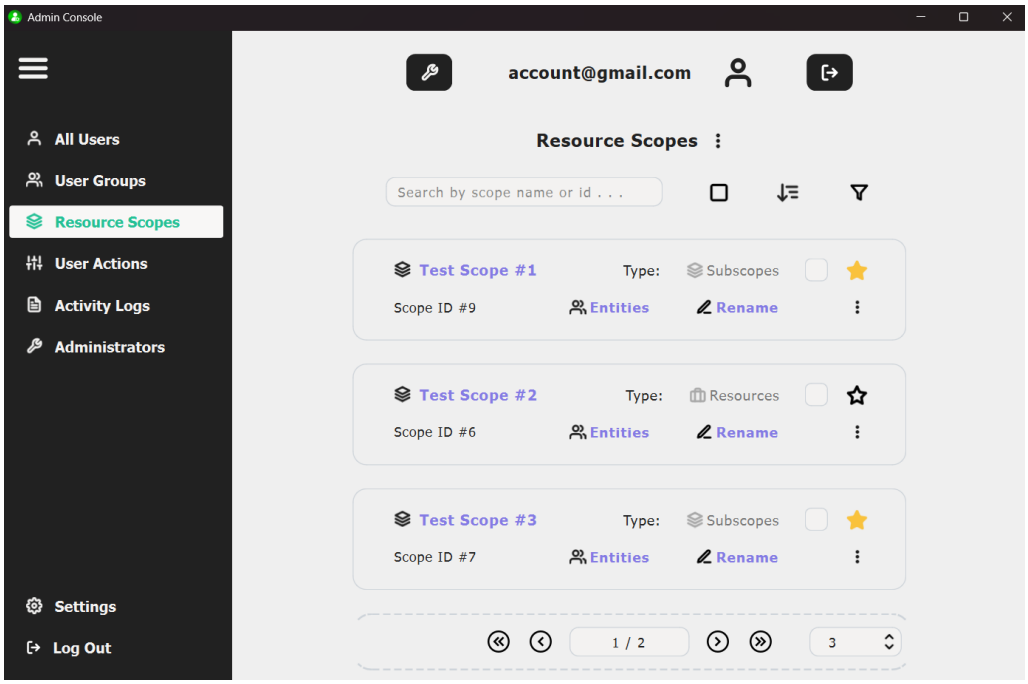


Рисунок 3.9 – Сторінка областей ресурсів

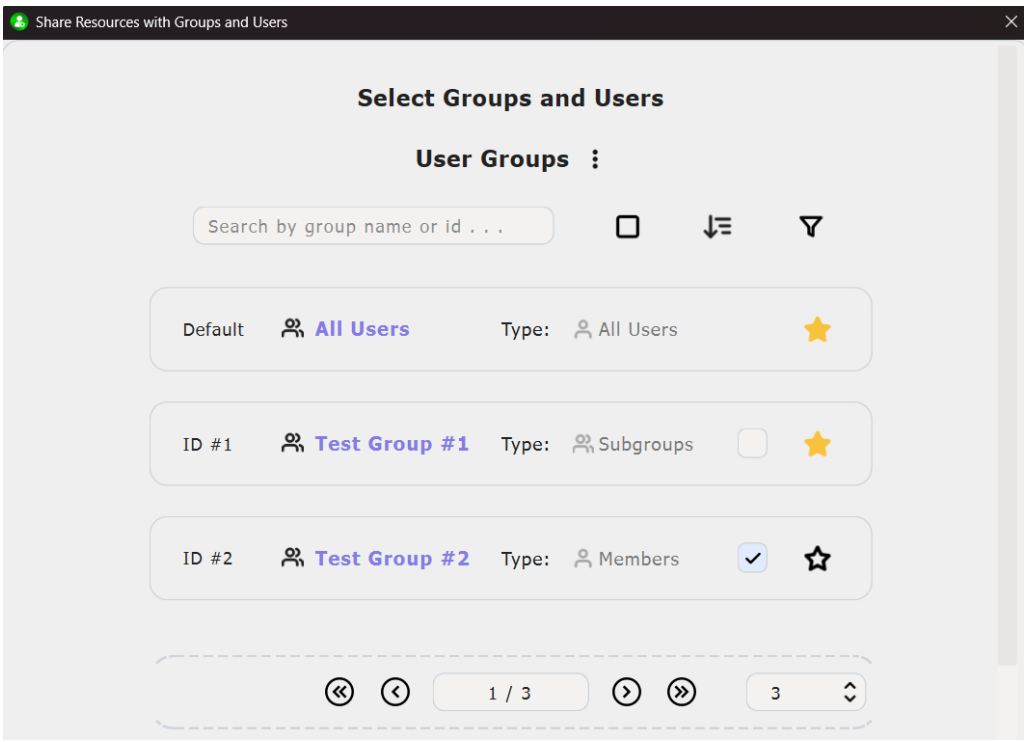


Рисунок 3.10 – Діалогове вікно налаштування прав доступу

На сторінці можливих дій користувачів (рис. 3.11) є можливість одночасного налаштування прав доступу для сутностей (користувачам або групам), вибираючи перелік дозволених дій під час роботи користувачів з відповідними модулями програмного забезпечення.

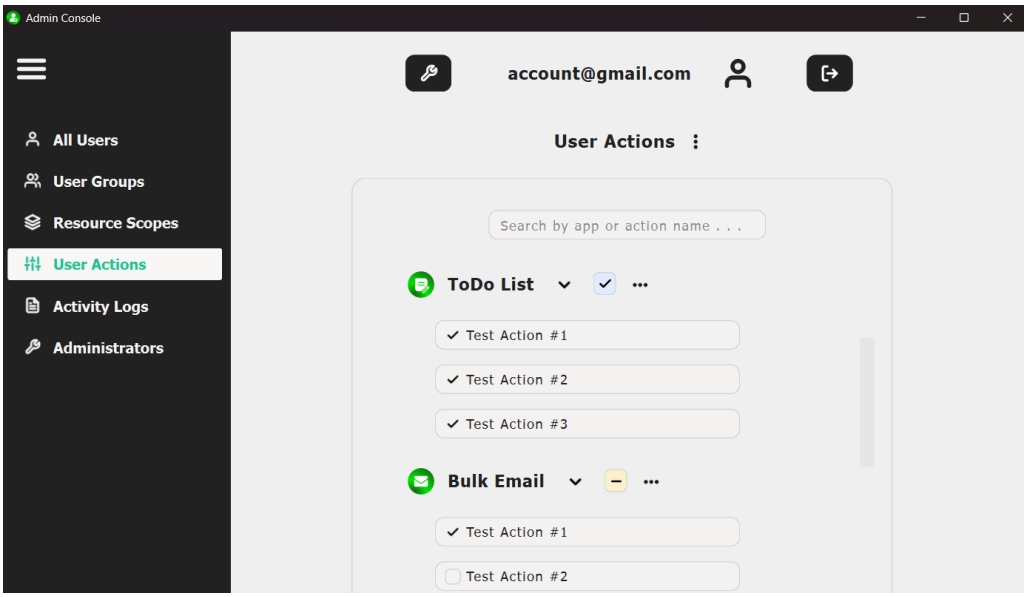


Рисунок 3.11 – Сторінка можливих дій користувачів

Перегляд та аналіз логів активності здійснюється на сторінці журналу подій (рис. 3.12).

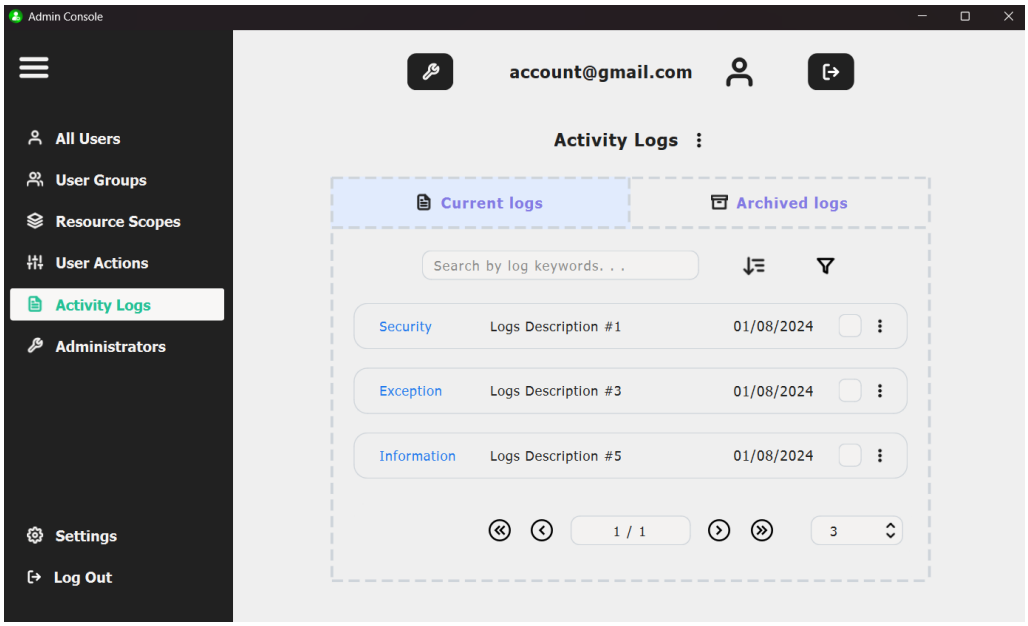


Рисунок 3.12 – Сторінка журналу подій

Управління адміністраторами здійснюється на сторінці адміністраторів (рис. 3.13).

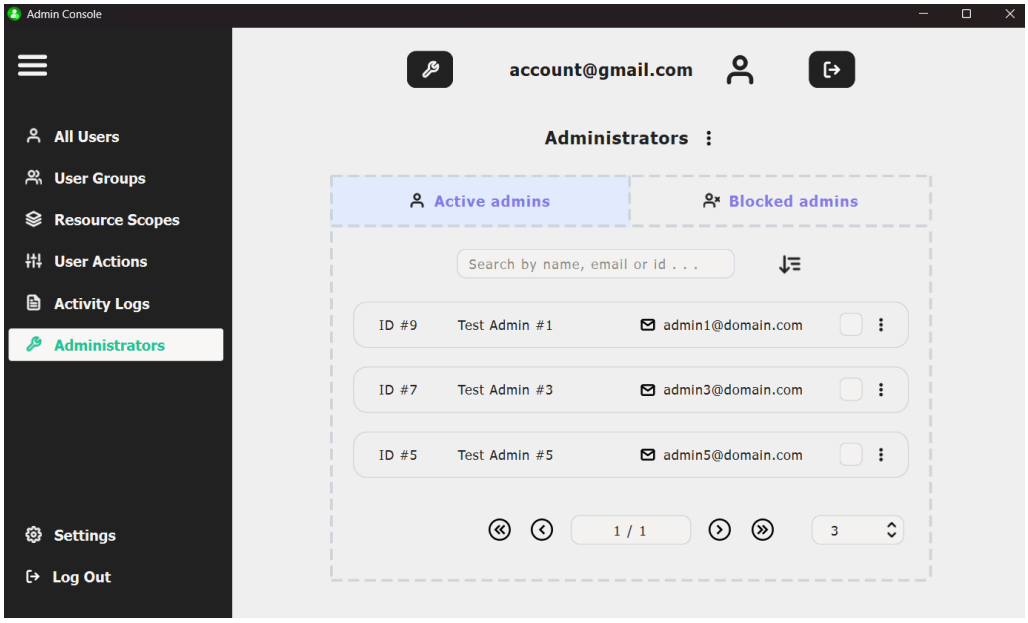


Рисунок 3.13 – Сторінка управління адміністраторами

3.5 Порівняння інструменту адміністрування «Admin Console» з наявними рішеннями для розподілу ресурсів і керування правами доступу

В табл. 3.1 наведено порівняльну характеристику наявних рішень для розподілу ресурсів і керування правами доступу.

Таблиця 3.1

Порівняння наявних рішень для розподілу ресурсів і керування правами доступу

Параметр	**Admin Console**	**Azure AD**	**Google Cloud IAM**
Багатофакторна аутентифікація	+	+	+
Масштабованість	+	+	+
Оптимізація бюджету	+	–	–
Автоматизація рутинних процесів	+	–	+
Складність навчання роботи з рішенням	–	+	+
Гнучкість керуванням діями користувачів при роботі з вбудованими модулями програмного забезпечення	+	–	–
Незалежність від екосистеми Microsoft	+	–	+
Розробка власних інтеграцій через API	–	+	+
Інтуїтивний та гнучкий процес налаштування прав доступу	+	–	–

Інструмент адміністрування для розподілу ресурсів і керування правами доступу «Admin Console» заснований на гібридній моделі управління доступом. За основу гібридної моделі управління доступом обрано модель контролю доступу на основі груп, додаючи принцип розподілу прав доступу до ресурсів на основі шаблонів прав доступу. Для ефективного управління користувачами, адміністратори створюють групи та налаштовують шаблони прав доступу для груп. Додаючи користувачів до груп, можна швидко призначити користувачам визначений набір прав доступу з можливістю їх подальшої модифікації.

Інструмент адміністрування автоматизує рутинні процеси, дозволяючи налаштовувати права доступу одночасно для декількох сутностей, що значно підвищує ефективність процесу адміністрування. Також інструмент адміністрування надає можливість автоматизувати процес відкликання учасників груп, налаштувавши час відкликання користувачів з відповідних груп.

Інструмент адміністрування «Admin Console» надає інтуїтивно зрозумілий та гнучкий процес налаштування прав доступу, зменшуючи складність навчання користувачів для роботи з розробленим рішенням та додатково забезпечуючи гнучкість керуванням діями користувачів при роботі з вбудованими модулями програмного забезпечення. Інструмент адміністрування надає додатковий рівень захисту під час входу в систему, за рахунок впровадження MFA. Інструмент адміністрування «Admin Console» підтримує масштабованість і забезпечує економічну ефективність та оптимізацію бюджету у порівнянні з розглянутими наявними рішеннями для розподілу ресурсів і керування правами доступу.

Висновки за розділом 3

У третьому розділі було розглянуто процес розробки інструменту адміністрування для розподілу ресурсів і керування правами доступу «Admin Console», заснованого на гібридній моделі управління доступом.

Описано трирівневу клієнт-серверну архітектуру інструменту адміністрування «Admin Console». Клієнт-серверна архітектура зосереджена

навколо циклу запит-відповідь. Клієнт надсилає запит на сервер для отримання даних або послуги. Сервер отримує запит, обробляє його, формуючи відповідь. Трирівнева архітектура додає проміжний рівень (бізнес-рівень) між клієнтом та сервером, що сприяє розподіленню навантаження та підвищення гнучкості і масштабованості.

Описано гібридну модель управління доступом, за основу якої обрано модель контролю доступу на основі груп, додаючи принцип розподілу прав доступу до ресурсів на основі шаблонів прав доступу. Запропонована гібридна модель управління доступом підвищує ефективність та надає більшу гнучкість налаштування прав доступ.

Розглянуто програмну реалізацію інструменту адміністрування для розподілу ресурсів і керування правами доступу «Admin Console». Інструмент адміністрування спроектовано в ООП стилі, розроблено на мові програмування Python версії 3.0+ з використанням фреймворку Qt 6.0+ та розбито на окремі модулі, згруповані в наступні пакети: widgets (віджети), scripts (модулі обробки логіки) та gui (модулі обробки графічного інтерфейсу користувача). Об’єктно-орієнтоване програмування – парадигма програмування, яка розглядає програму як множину об’єктів, що взаємодіють між собою.

Для взаємодії з розробленою реляційною базою даних, написано SQL-запити на отримання, оновлення та видалення записів до наступних таблиць: користувачі, адміністратори, групи користувачів, учасники груп, області ресурсів, ресурси, модулі програмного забезпечення, можливі дії користувачів, права доступу до ресурсів, розширені права доступу до ресурсів, журнал авторизації та відмічені адміністратором.

Продемонстровано роботу інструменту адміністрування «Admin Console» та описано графічний інтерфейс користувача. Проведено порівняння інструменту адміністрування «Admin Console» з наявними рішеннями для розподілу ресурсів і керування правами доступу.

ВИСНОВКИ

У даній кваліфікаційній роботі було розроблено інструмент адміністрування для розподілу ресурсів і керування правами доступу, заснований на гібридній моделі управління доступом. Новизною роботи є запропонована гібридна модель управління доступом, яка підвищує ефективність та надає більшу гнучкість налаштування прав доступу.

У першій частині кваліфікаційної роботи проведено огляд останніх досліджень і публікацій, зокрема розглянуто важливість інформаційної безпеки та моделей керування доступом. В основі системи контролю доступу лежить безпечна оцінка того, чи має встановлена ідентичність доступ до певного обчислювального ресурсу. Проаналізовано та порівняно наступні моделі керування доступом: Bell-LaPadula, Biba, DAC, MAC, RBAC, ABAC, OrBAC, RuleBAC, ReBAC та GBAC.

У другій частині кваліфікаційної роботи проведено моделювання інструменту адміністрування для розподілу ресурсів і керування правами доступу «Admin Console» за допомогою уніфікованої мови моделювання – UML, зокрема наведено: UML-діаграму варіантів використання (Use Case Diagram), UML-діаграму діяльності (Activity Diagram), UML-діаграму станів (State Machine Diagram) та UML-діаграму послідовності (Sequence Diagram). Виконано проектування бази даних інструменту адміністрування для розподілу ресурсів і керування правами доступу «Admin Console». Використано СКБД MySQL – систему керування реляційними базами даних. Згенеровано EERM, яка точно представляє вимоги складних баз даних і дозволяє відобразити сутності, їх атрибути та зв'язки, які представляють асоціації між сутностями. Описано структуру таблиць спроектованої реляційної бази даних.

У третій частині кваліфікаційної роботи розглянуто процес розробки інструменту адміністрування для розподілу ресурсів і керування правами доступу «Admin Console», заснованого на гібридній моделі управління доступом.

Описано трирівневу клієнт-серверну архітектуру інструменту адміністрування та гібридну модель управління доступом, за основу якої обрано модель контролю доступу на основі груп, додаючи принцип розподілу прав доступу до ресурсів на основі шаблонів прав доступу. Запропонована гібридна модель управління доступом підвищує ефективність та надає більшу гнучкість налаштування прав доступ. Розглянуто програмну реалізацію інструменту адміністрування «Admin Console». Інструмент адміністрування спроектовано в ООП стилі, розроблено на мові програмування Python версії 3.0+ з використанням фреймворку Qt 6.0+ та розбито на окремі модулі, згруповані в пакети. Для взаємодії з розробленою реляційною базою даних, написано SQL-запити на отримання, оновлення та видалення записів до таблиць бази даних. Продемонстровано роботу інструменту адміністрування «Admin Console» та описано графічний інтерфейс користувача. Проведено порівняння інструменту адміністрування «Admin Console» з наявними рішеннями для розподілу ресурсів і керування правами доступу.

Виходячи із поставленої мети кваліфікаційної роботи були виконані наступні завдання:

- розглянуто особливості моделей керування доступом;
- проведено аналіз та порівняння моделей контролю доступу;
- виконано моделювання інструменту адміністрування для розподілу ресурсів і керування правами доступу;
- здійснено проектування бази даних інструменту адміністрування;
- виконано програмну реалізацію інструменту адміністрування;
- проведено аналіз та порівняння наявних рішень для розподілу ресурсів і керування правами доступу.

Апробація результатів кваліфікаційної роботи відбулася на XVIII Міжнародній науково-практичній конференції «Modern Information and communication technologies on a transport, in industry and education» [1] (Додаток А).

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Babenko Y. M., Kramarenko D. V., Administration tool for resource allocation and access rights management // XVIII International Conference “Modern Information and communication technologies on a transport, in industry and education”, Dnipro, 12-13 December, 2024, p. 173.

2. Karp, A. H., Haury, H., Davis, M. H. From ABAC to ZBAC: The evolution of access control models. Journal of Information Warfare. – 2010. – Vol. 9, no. 2. – P. 38–46 [Електронний ресурс]. URL: https://www.jstor.org/stable/26486786 (дата звернення: 11.01.2025).

3. Penelova, M. Access control models. Cybernetics and Information Technologies Journal. – 2021. – Vol. 21, no. 4. – P. 77–104 [Електронний ресурс]. URL: http://doi.org/10.2478/cait-2021-0044 (дата звернення: 12.01.2025).

4. United States Code, 2011 Edition, Supplement 5, Title 44 Public Printing and Documents (дата звернення: 15.01.2025).

5. Yeboah-Boateng, E. O., Kwabena-Adade, G. D. Remote access communications security: Analysis of user authentication roles in organizations. Journal of Information Security. – 2020. – Vol. 11, no. 3. – P. 161-175 [Електронний ресурс]. URL: http://doi.org/10.4236/jis.2020.113011 (дата звернення: 15.01.2025).

6. Sandhu, R. S., Samarati, P. Access control: principle and practice. IEEE Communications Magazine. – 1994. – Vol. 32, no. 9. – P. 40-48 [Електронний ресурс]. URL: http://doi.org/10.1109/35.312842 (дата звернення: 15.01.2025).

7. What Is AAA Security? Fortinet [Електронний ресурс]. URL: https://www.fortinet.com/resources/cyberglossary/aaa-security (дата звернення: 15.01.2025).

8. Benantar, M. Access Control Systems: Security, Identity Management and Trust Models. Springer Science & Business Media, 2005. 284 p.

9. Biometrics | Homeland Security. U.S. Department of Homeland Security [Електронний ресурс]. URL: https://www.dhs.gov/biometrics (дата звернення: 15.01.2025).

10. Miller, M. S., Yee, K. P., Shapiro, J. Capability myths demolished. Johns Hopkins University Systems Research Laboratory, 2003 [Електронний ресурс]. URL: http://srl.cs.jhu.edu/pubs/SRL2003-02.pdf (дата звернення: 17.01.2025).

11. Gasser, M. Building a secure computer system. New York : Van Nostrand Reinhold Company, 1998. 288 p.

12. Trusted Computer System Evaluation Criteria (TCSEC). Department of Defence, USA, 5200.28-STD, 1983.

13. Osborn, S., Sandhu, R., Munawer, Q. Configuring Role-Based Access Control to Enforce Mandatory and Discretionary Access Control Policies. ACM Transactions on Information and System Security Journal. – Vol. 3, May 2000, no. 2. – P. 85-106 [Електронний ресурс]. URL: http://doi.org/10.1145/354876.354878 (дата звернення: 17.01.2025).

14. Hu, C.V. NIST Special Publication 800-162. Guide to Attribute Based Access Control (ABAC) definition and considerations. National Institute Standards and Technology, 2016. 47 p. [Електронний ресурс]. URL: http://doi.org/10.6028/NIST.SP.800-162

15. Guesmia, K., Boustia, N. OrBAC from access control model to access usage model. Applied Intelligence Journal. – 2018. – Vol. 48, P. 1996-2016 [Електронний ресурс]. URL: http://doi.org/10.1007/s10489-017-1064-3 (дата звернення: 17.01.2025).

16. Aftab, M. U., Hamza, A., Oluwasanmi, A., Nie, X., Sarfraz, M. S., Shehzad, D., Qin, Z. and Rafiq, A. Traditional and hybrid access control models: A detailed survey. Security and Communication Networks Journal. – 2022. – Vol. 2022, no. 1. – 12 p. [Електронний ресурс]. URL: http://doi.org/10.1155/2022/1560885 (дата звернення: 20.01.2025).

17. Cheng, Y., Park, J., Sandhu, R. Attribute-Aware Relationship-Based Access Control for Online Social Networks. Springer Berlin Heidelberg. – 2014, P. 292-306

[Електронний ресурс]. URL: http://doi.org/10.1007/978-3-662-43936-4_19 (дата звернення: 20.01.2025).

18. Zhonghua, C., Goyal, S. B., Rajawat, A. S. Smart contracts attribute-based access control model for security & privacy of IoT system using blockchain and edge computing. The Journal of Supercomputing. – 2024. – Vol. 80, no. 2. P. 1396-1425 [Електронний ресурс]. URL: http://doi.org/10.1007/s11227-023-05517-4 (дата звернення: 20.01.2025).

19. What Is Group-Based Access Control (GBAC)? StrongDM [Електронний ресурс]. URL: https://www.strongdm.com/what-is/group-based-access-control-gbac (дата звернення: 20.01.2025).

20. Jacobson, L., Booch, J. R. G. The unified modeling language reference manual. Addison-Wesley Longman Ltd., GBR, 2021. 550 p.

21. Abdulmonim, D. A., Muhamad, Z. H., Alathari, B. Using the Object Mapping Approach from Analysis to Implementation for Developing Student Registration System. Indonesian Journal of Electrical Engineering and Computer Science. – 2019. – Vol. 14, no. 2. P. 1030-1038 [Електронний ресурс]. URL: http://doi.org/10.11591/ijeecs.v14.i2.pp1030-1038 (дата звернення: 04.02.2025).

22. Bonilla-Morales, B., Crespo, S., Clunie, C. Reuse of Use Cases Diagrams: An Approach based on Ontologies and Semantic Web Technologies. IJCSI International Journal of Computer Science Issues. – 2012. – Vol. 9, no. 2. P. 24-29 [Електронний ресурс]. URL: https://www.academia.edu/80738564/Reuse_of_Use_Cases_Diagrams_An_Approach_based_on_Ontologies_and_Semantic_Web_Technologies (дата звернення: 07.02.2025).

23. von der Maßen, T., Lichter, H. Modeling variability by UML use case diagrams. In Proceedings of the International Workshop on Requirements Engineering for product lines. Technical Report: ALR-2002-033. Citeseer. – 2002. P.19-25 [Електронний ресурс]. URL: https://swc.rwth-aachen.de/docs/2002_PLE_Essen.pdf (дата звернення: 07.02.2025).

24. Ahmad, T., Iqbal, J., Ashraf, A., Truscan, D., Porres, I. Model-based testing using UML activity diagrams: A systematic mapping study. Computer Science Review

Journal. – 2019. – Vol. 33. P. 98-112 [Електронний ресурс]. URL: http://doi.org/10.1016/j.cosrev.2019.07.001 (дата звернення: 09.02.2025).

25. Li, X., Liu, Z., Jifeng, H. (2004, April). A formal semantics of UML sequence diagram. Australian Software Engineering Conference. Proceedings. – 2004. – P. 168-177 [Електронний ресурс]. URL: http://doi.org/10.1109/ASWEC.2004.1290469 (дата звернення: 11.02.2025).

26. Jones, L. Mastering Python Design Patterns for Scalable Applications: Unlock the Secrets of Expert-Level Skills. Walzone Press, 2025. 574 p.

27. Lott, S. F., Phillips, D. Python Object-Oriented Programming: Build Robust and Maintainable Object-oriented Python Applications and Libraries. Packt Publishing Ltd, 2021. 714 p.

28. Rangisetti, A. K. Hands-On Object-Oriented Programming: Mastering OOP Features for Real-World Software Systems Development. Apress, 2024. 608 p.

29. Chiang, R. H., Barron, T. M., Storey, V. C. Reverse engineering of relational databases: Extraction of an EER model from a relational database. Data & knowledge engineering Journal. – 1994. – Vol. 12, no. 2. – P. 107-142 [Електронний ресурс]. URL: http://doi.org/10.1016/0169-023X(94)90011-6 (дата звернення: 13.02.2025).

30. Identity and Access Management. Google Cloud [Електронний ресурс]. URL: https://cloud.google.com/security/products/iam (дата звернення: 04.04.2025).

31. IAM overview | IAM Documentation. Google Cloud [Електронний ресурс]. URL: https://cloud.google.com/iam/docs/overview (дата звернення: 04.04.2025).

32. What is Azure Active Directory? A Complete Overview. Varonis [Електронний ресурс]. URL: https://www.varonis.com/blog/azure-active-directory (дата звернення: 04.04.2025).

33. Kumar, S. A Review on Client-Server based applications and research opportunity. International Journal of Recent Scientific Research. – 2019. – Vol. 10, no. 7. P. 33857-3386. doi/10.24327/ijrsr.2019.1007.3768 [Електронний ресурс]. URL: https://www.researchgate.net/publication/335015436_a_review_on_client-server_based_applications_and_research_opportunity (дата звернення: 07.04.2025).

ДОДАТКИ

ДОДАТОК А

Апробація результатів дослідження

Babenko Y. M., Kramarenko D. V., Administration tool for resource allocation and access rights management // XVIII International Conference “Modern Information and communication technologies on a transport, in industry and education” (Dnipro, 12-13 December, 2024), p.173.

ДОДАТОК Б

Лістинг фрагмента коду віджета «AdministratorWidget»

```
from PySide6.QtCore import Qt, QSize, QCoreApplication
from PySide6.QtGui import QIcon
from PySide6.QtWidgets import (QDialog, QWidget, QLabel, QPushButton,
    QHBoxLayout, QVBoxLayout, QSizePolicy, QSpacerItem, QCheckBox)
from UserAccountDetailsDialogGUI import UserAccountDetailsDialogGUI
from scripts.ShowContextMenu import show_custom_context_menu


class AdministratorWidget(QWidget):
    def __init__(self, id, name, email, is_active):
        super().__init__()

        self.id = id # admin id
        self.name = name # admin name
        self.email = email # admin email
        self.is_active = is_active # active / blocked admin

        self.container = QWidget()
        self.verticalLayout = QVBoxLayout(self)
        self.verticalLayout.setSpacing(0)
        self.verticalLayout.setContentsMargins(0, 0, 0, 0)

        ### UI CODE BEGIN ###
        ### UI CODE END ###

        self.verticalLayout.addWidget(self.container)

        self.RecordContextMenu.clicked.connect(self.context_menu_button)


    def context_menu_button(self):
        menu_items = [
            ["Admin details", "Unblock admin"],
            ["Admin details", "Block admin"]][self.is_active]
        item_actions = [self.context_menu_action_1, self.context_menu_action_2]

        show_custom_context_menu(self, menu_items, item_actions)

    def context_menu_action_1(self):
        # admin details
        dialog_user_details = UserAccountDetailsDialogGUI(self.name)
        dialog_user_details.show()
        dialog_user_details.exec()
```

ДОДАТОК В

Лістинг фрагмента коду модуля «ShowMessageBoxScript»

```
from PySide6.QtCore import QSize
from PySide6.QtGui import QIcon
from PySide6.QtWidgets import QMessageBox


def show_message(msg_type, buttons, default_button, message, title):
    msg_box = QMessageBox()
    msg_box.setWindowTitle(title)
    msg_box.setText(message)
    msg_box.setIcon(msg_type)
    msg_box.setStandardButtons(buttons)
    msg_box.setDefaultButton(default_button)

    reply = msg_box.exec()
    if reply == QMessageBox.Cancel:
        return None
    else:
        return reply == QMessageBox.Yes

def show_info_message(message):
    return    show_message(QMessageBox.Information,    QMessageBox.Ok,    QMessageBox.Ok,
message, title="Information")

def show_warning_message(message):
    return   show_message(QMessageBox.Warning,   QMessageBox.Yes   |   QMessageBox.No   |
QMessageBox.Cancel,
                          QMessageBox.Cancel, message, title="Warning")

def show_critical_message(message):
    return show_message(QMessageBox.Critical, QMessageBox.Ok, QMessageBox.Ok, message,
title="Error")
```

ДОДАТОК Г

Лістинг фрагмента коду модуля налаштування прав доступу «AccessRightsSelectGroupsUsersDialogGUI»

```
from PySide6 import QtSvg  # noqa: F401
from PySide6.QtCore import QCoreApplication, QSize, Slot
from PySide6.QtGui import QIcon
from PySide6.QtWidgets import QDialog
from auto_gui import resources_rc # noqa: F401
from auto_gui.access_rights.AccessRightsSelectGroupsUsersDialog import Ui_Dialog
from widgets.GroupSelectionWidget import GroupSelectionWidget
from widgets.DefaultGroupSelectionWidget import DefaultGroupSelectionWidget
from widgets.UserSelectionWidget import UserSelectionWidget
from widgets.NavigationWidget import NavigationWidget
from scripts.ShowContextMenuWithMemory import show_context_menu_with_memory
from scripts.ShowContextMenu import show_custom_context_menu


class AccessRightsSelectGroupsUsersDialogGUI(QDialog):
    def __init__(self):
        super(AccessRightsSelectGroupsUsersDialogGUI, self).__init__()
        self.ui = Ui_Dialog()
        self.ui.setupUi(self)
        self.setModal(True)

        # Groups Page
        self.GroupsNavigation = NavigationWidget()

self.GroupsNavigation.updateNavigationSignal.connect(self.display_groups_records)

        self.replace_groups_navigation()

        ### UI CODE BEGIN ###
        ### UI CODE END ###

        self.switch_to_groups_page()

    def scroll_to_top(self, scroll_area):
        vertical_scrollbar = scroll_area.verticalScrollBar()
        vertical_scrollbar.setValue(0)

    def replace_groups_navigation(self):
        self.ui.verticalLayout_54.replaceWidget(self.ui.NavigationWidget_11,
self.GroupsNavigation)
        self.ui.NavigationWidget_11.close()
        self.ui.verticalLayout_54.update()

    @Slot()
    def      switch_to_page_by_id_and_type(self,      item_type=None,      item_id=None,
item_to_show=None, item_parent_id=None):
        widget_switcher = {
            "Groups": (self.switch_to_groups_page, None),
            "Subgroups": (self.switch_to_subgroups_page, None),
            "Members": (self.switch_to_group_memebers_page, None),
            "All Users": (self.switch_to_all_users_page, None),
        }
```

```
        widget_switcher_func, arg = widget_switcher[item_type]
        widget_switcher_func() # call the appropriate function with the argument

    def switch_to_groups_page(self):
        self.ui.StackedWidget.setCurrentIndex(0)
        self.groups_page_search_all = True
        self.ui.SearchAllButton.click() # deactivate search all button
        self.prepare_groups_records()
        self.GroupsNavigation.update_records_count(len(self.groups_records))

    def     prepare_groups_records(self,     ids=None,     names=None,     types=None,
groups_is_shared=None, is_favorites=None):
        # remove group widgets
        for widget in self.groups_records:
            self.ui.PageTableQVBoxLayout_2.removeWidget(widget)
            widget.close()
        self.groups_records.clear()

        # add default group widgets
        all_users_widget = DefaultGroupSelectionWidget(None, "All Users", "All Users")

all_users_widget.showPageByIDNameSignal.connect(self.switch_to_page_by_id_and_type)
        self.groups_records.append(all_users_widget)

        # add group widgets
        for  id,  name,  type,  is_shared,  is_favorite  in  zip(ids,  names,  types,
groups_is_shared, is_favorites):
            widget = GroupSelectionWidget(id, name, type, is_shared, is_favorite)
            widget.showPageByIDNameSignal.connect(self.switch_to_page_by_id_and_type)
            self.groups_records.append(widget)

    def display_groups_records(self, is_render, page_record_begin, page_record_end):
        if is_render:
            # display group widgets
            self.remove_groups_records()
            for widget in self.groups_records[page_record_begin:page_record_end]:
                self.ui.PageTableQVBoxLayout_2.addWidget(widget)
                widget.setVisible(True)
            # scroll a QScrollArea to the top
            self.scroll_to_top(self.ui.ScrollAreaFullPage_2)

    def remove_groups_records(self):
        for widget in self.groups_records:
            self.ui.PageTableQVBoxLayout_2.removeWidget(widget)
            #widget.close()
            widget.setVisible(False)
        self.ui.PageTableQVBoxLayout_2.update()


    def groups_page_context_menu(self):
        menu_items = ["Select filtered", "Unselect filtered"]
        item_actions                =                [self.groups_context_menu_action_1,
self.groups_context_menu_action_2]
            show_custom_context_menu(self, menu_items, item_actions)
```

ДОДАТОК Д

Лістинг фрагмента коду SQL-запиту на оновлення прав доступу до ресурсів до реляційної СКБД MySQL

```
import mysql.connector

def add_shared_resource(user_id=None, group_id=None, scope_id=None, resource_id=None,
user_action_id=None, is_open=1):
    """ Adds records to sharedresources or sharedresourcesextra (group_id –
sharedresources, user_id – sharedresourcesextra) """
    connection = mysql.connector.connect(
        host = os.environ["MYSQL_HOST"]
        user = os.environ["MYSQL_USER"]
        password = os.environ["MYSQL_PASSWORD"]
        database = "ac2"
    )
    cursor = connection.cursor()

    try:
        if not user_id and not group_id:
            print("missing args: user_id or group_id")
            return

        table = "sharedresourcesextra" if user_id else "sharedresources"
        entity_id = user_id if user_id else group_id
        entity_column = "User_ID" if user_id else "Group_ID"

        if user_id:
            upsert_query = f"""
            INSERT INTO `ac2`.`{table}` ({entity_column}, Scope_ID, Resource_ID,
UserAction_ID, Is_open)
            VALUES (%s, %s, %s, %s, %s)
            ON DUPLICATE KEY UPDATE Is_open = VALUES(Is_open);"""
            cursor.execute(upsert_query, (entity_id, scope_id or None, resource_id or
None, user_action_id or None, is_open))
        else:
            query = f"""
            INSERT IGNORE INTO `ac2`.`{table}` ({entity_column}, Scope_ID, Resource_ID,
UserAction_ID)
            VALUES (%s, %s, %s, %s);"""
            cursor.execute(query, (entity_id, scope_id or None, resource_id or None,
user_action_id or None))

        connection.commit()
        print(f"Resource added into {table}.")

        # In case Scope_ID added, add all nested Scope_ID and linked resources
        if scope_id:
            add_child_scopes_and_resources(cursor, table, entity_column, entity_id,
scope_id, user_id, is_open)
            connection.commit()
            print("Added all nested Scope_ID and linked resources")

    except mysql.connector.Error as err:
        print(f"Error: {err}")

    finally:
```

```
        cursor.close()
        connection.close()


def    add_child_scopes_and_resources(cursor,    table,    entity_column,    entity_id,
parent_scope_id, is_user, is_open=1):
    """ Recursively adds all nested Scope_ID into sharedresources/extra included nested
resources """
    query = """
        WITH RECURSIVE ScopeHierarchy AS (
            SELECT Scope_ID FROM resourcescopes WHERE Parent_Scope_ID = %s
            UNION ALL
            SELECT rs.Scope_ID
            FROM resourcescopes rs
            INNER JOIN ScopeHierarchy sh ON rs.Parent_Scope_ID = sh.Scope_ID
        )
        SELECT Scope_ID FROM ScopeHierarchy;"""
    cursor.execute(query, (parent_scope_id,))
    child_scopes = [row[0] for row in cursor.fetchall()]

    if child_scopes:
        if is_user:
            query = f"""
            INSERT INTO `ac2`.`{table}` ({entity_column}, Scope_ID, Is_open)
            VALUES (%s, %s, %s)
            ON DUPLICATE KEY UPDATE Is_open = VALUES(Is_open);"""
            cursor.executemany(query, [(entity_id, scope_id, is_open) for scope_id in
child_scopes])
        else:
            query = f"""
            INSERT IGNORE INTO `ac2`.`{table}` ({entity_column}, Scope_ID)
            VALUES (%s, %s);"""
            cursor.executemany(query,   [(entity_id,   scope_id)   for   scope_id   in
child_scopes])

        # Find resources
        query = """
        SELECT Resource_ID FROM resources WHERE Scope_ID IN (%s);""" % ",".join(str(sid)
for sid in child_scopes)
        cursor.execute(query)
        child_resources = [row[0] for row in cursor.fetchall()]
        if child_resources:
            if is_user:
                query = f"""
                INSERT INTO `ac2`.`{table}` ({entity_column}, Resource_ID, Is_open)
                VALUES (%s, %s, %s)
                ON DUPLICATE KEY UPDATE Is_open = VALUES(Is_open);"""
                cursor.executemany(query, [(entity_id, res_id, is_open) for res_id in
child_resources])
            else:
                query = f"""
                INSERT IGNORE INTO `ac2`.`{table}` ({entity_column}, Resource_ID)
                VALUES (%s, %s);"""
                cursor.executemany(query,   [(entity_id,   res_id)   for   res_id   in
child_resources])

# SQL-query call
# add_shared_resource(user_id=3, scope_id=5, is_open=1)
add_shared_resource(group_id=2, scope_id=5)
```

ДОДАТОК Е

Презентація доповіді та роздатковий матеріал

Київський національний університет імені Тараса Шевченка

Факультет інформаційних технологій

Кафедра кібербезпеки та захисту інформації

Кваліфікаційна робота

на здобуття наукового ступеня бакалавра

«Програмний модуль адміністрування розподілу ресурсів і керування правами доступу»

Виконав: студент групи КБ-44мс, Крамаренко Данило Васильович

Науковий керівник: доктор філософії Бабенко Юрій Михайлович

Київ-2025

Актуальність роботи

У сучасну цифрову епоху, коли організації все більше покладаються на IT-інфраструктуру, ефективне управління інформаційними ресурсами та правами доступу набуває вирішального значення.

Стрімке зростання обсягів даних, програмних рішень і хмарних обчислень призводить до значних викликів забезпечення інформаційної безпеки. Контроль доступу до інформаційних ресурсів і сервісів є фундаментальною основою безпеки.

Протягом багатьох років розроблено безліч моделей контролю доступу, кожна з яких призначена для вирішення різних аспектів цієї проблеми. Перспективи розвитку виражаються у створенні гібридних моделей управління доступом та нових рішень з контролю доступу.

Тому кваліфікаційна робота з розробки програмного модуля адміністрування розподілу ресурсів і керування правами доступу є актуальною та полягає у вирішенні проблеми реалізації ефективного управління інформаційними ресурсами і правами доступу та забезпечення інформаційної безпеки.

2

Мета роботи

Метою кваліфікаційної роботи є підвищення ефективності керування правами доступу шляхом розробки інструменту адміністрування.

Об'єктом дослідження є процес надання доступу користувачам до розподілених інформаційних ресурсів.

Предметом дослідження є методи, засоби та моделі керування доступом.

Методами дослідження є метод аналізу та порівняльний метод.

Практичною цінністю є інструмент адміністрування для розподілу інформаційних ресурсів і керування правами доступу.

Новизною роботи є запропонована гібридна модель управління доступом, за основу якої обрано модель контролю доступу на основі груп, додаючи принцип розподілу прав доступу до ресурсів на основі шаблонів прав доступу.

3

Задачі, які необхідно вирішити для досягнення поставленої мети

1. Розглянути особливості моделей керування доступом.
2. Проаналізувати та порівняти моделі контролю доступу.
3. Здійснити моделювання інструменту адміністрування для розподілу інформаційних ресурсів і керування правами доступу.
4. Здійснити проектування бази даних інструменту адміністрування.
5. Виконати програмну реалізацію інструменту адміністрування.
6. Проаналізувати та порівняти наявні рішення для розподілу інформаційних ресурсів і керування правами доступу.

4

Аналіз моделей керування доступом

Проведено огляд останніх досліджень і публікацій, зокрема розглянуто важливість інформаційної безпеки та контролю доступу.

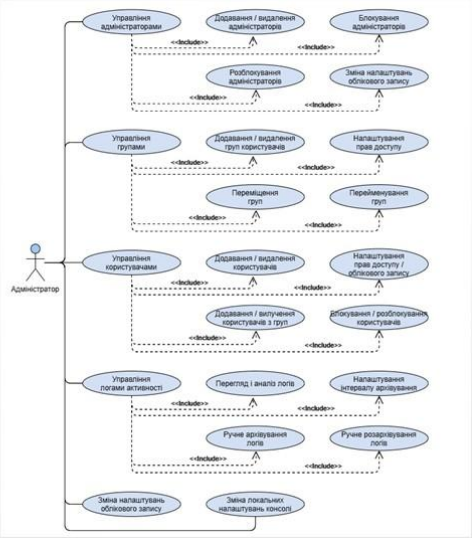
Розглянуто особливості моделей керування доступом: *Bell-LaPadula, Biba, DAC, MAC, RBAC, ABAC, OrBAC, RuleBAC, ReBAC та GBAC.*

Проаналізовано та порівняно моделі керування доступом за такими параметрами: гнучкість, масштабованість, розподіленість, динамічність, структура, контроль робочого процесу, деталізація політик, використання часу в політиках, делегування довіри, сфера застосування та інші.

5

Моделювання інструменту адміністрування

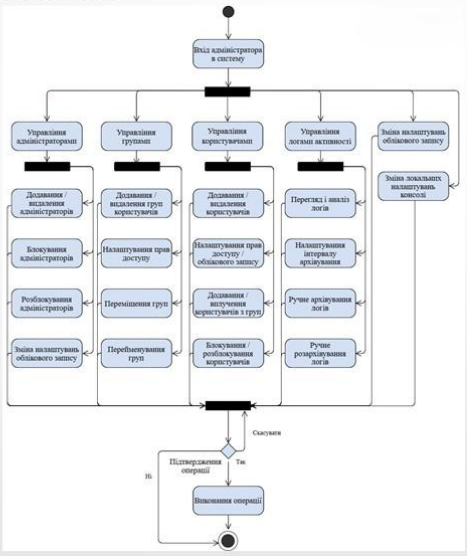
UML-діаграма варіантів використання відображає перелік акторів і варіантів використання, а також взаємодію між ними.



6

Моделювання інструменту адміністрування

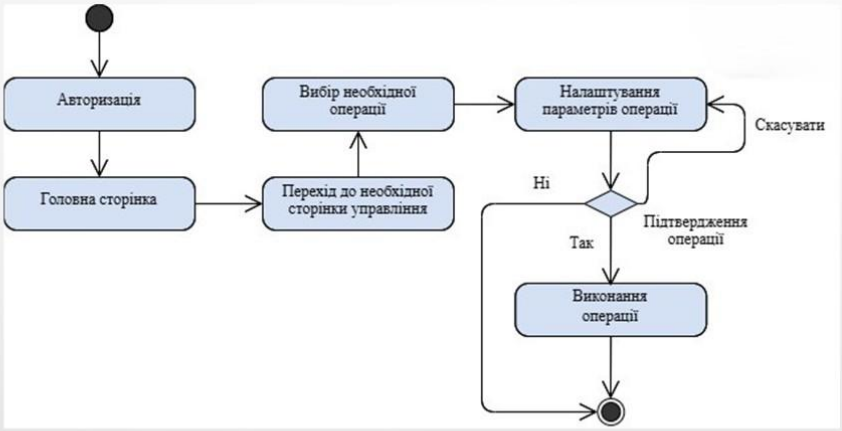
UML-діаграма діяльності є важливою діаграмою для моделювання динамічних аспектів системи.



7

Моделювання інструменту адміністрування

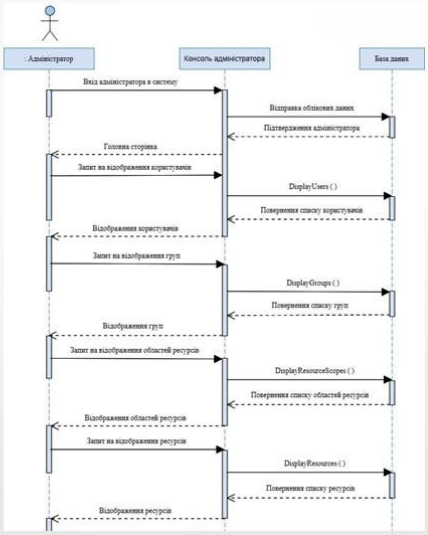
UML-діаграма станів є графом станів і переходів. Машина станів містить стани, пов'язані між собою переходами.



8

Моделювання інструменту адміністрування

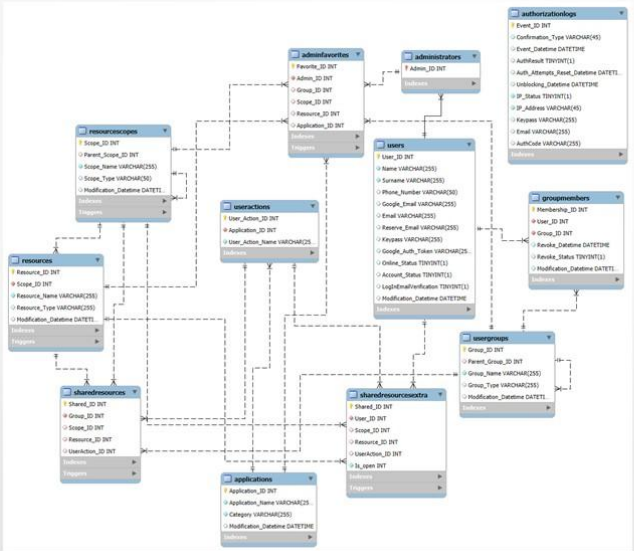
UML-діаграма послідовності показує взаємодію між об'єктами, розташованими в часовій послідовності за лінією життя.



9

Проектування бази даних

Розширена модель сутність-зв'язок (EER) інструменту адміністрування.



10

Проектування бази даних

Структура таблиці учасників груп

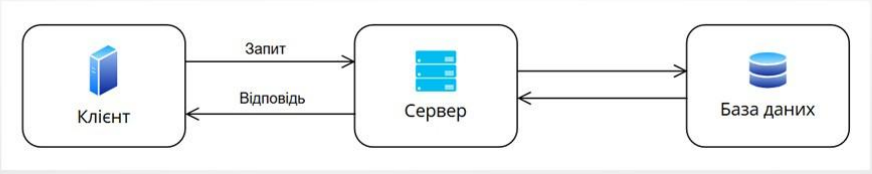
Назва	Тип даних	Опис
Membership_ID	INT	Первинний ключ, унікальний ІД членства в групі
User_ID	INT	Ідентифікатор користувача
Group_ID	INT	Ідентифікатор групи
Revoke_Datetime	DATETIME	Час відкликання користувача
Revoke_Status	TINYINT	Статус відкликання користувача
Modification_Datetime	DATETIME	Час модифікації

11

Архітектура інструменту адміністрування

В клієнт-серверній архітектурі клієнт надсилає запит на сервер для отримання даних або послуги. Сервер отримує запит, обробляє, формує відповідь та надсилає оброблені дані або сервіс у відповідь клієнту.

Трирівнева архітектура додає проміжний рівень (бізнес-рівень) між клієнтом та сервером, сприяє розподіленню навантаження та підвищення гнучкості і масштабованості.

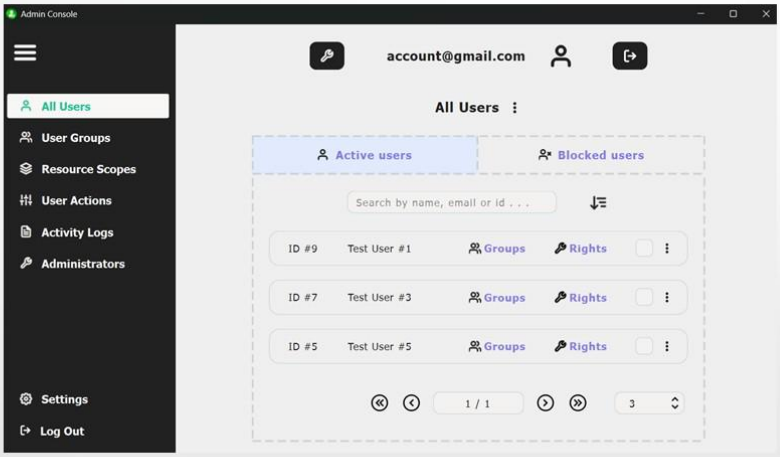


Трирівнева клієнт-серверна архітектура інструменту адміністрування «Admin Console»

12

Демонстрація роботи інструменту адміністрування

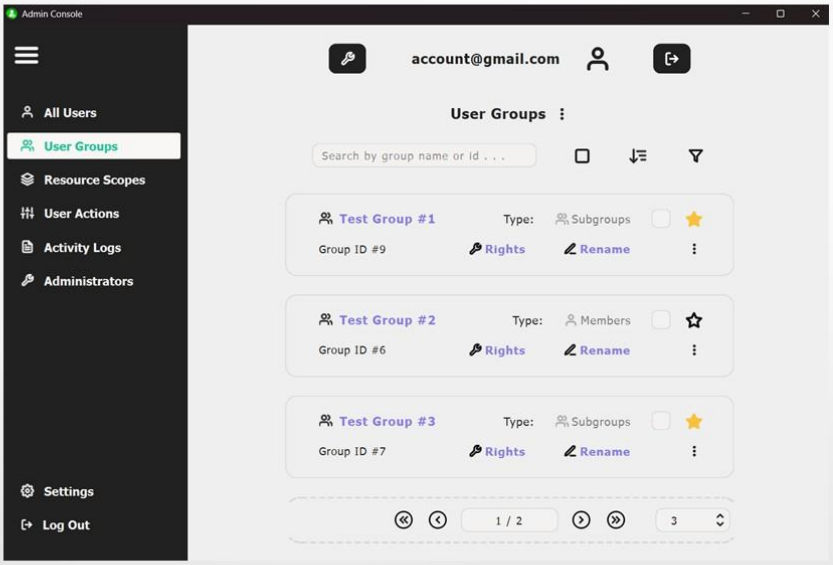
Інструмент адміністрування «Admin Console» спроектовано в ООП стилі, розроблено на мові програмування Python версії 3.0+, фреймворку Qt версії 6.0+ та розбито на окремі модулі, згруповані в пакети.



Головна сторінка управління користувачами

13

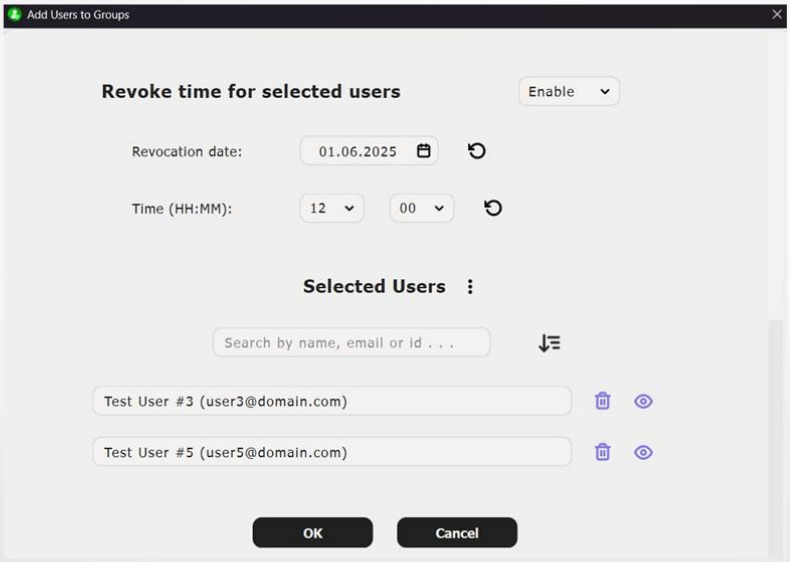
Демонстрація роботи інструменту адміністрування



Сторінка управління групами користувачів

14

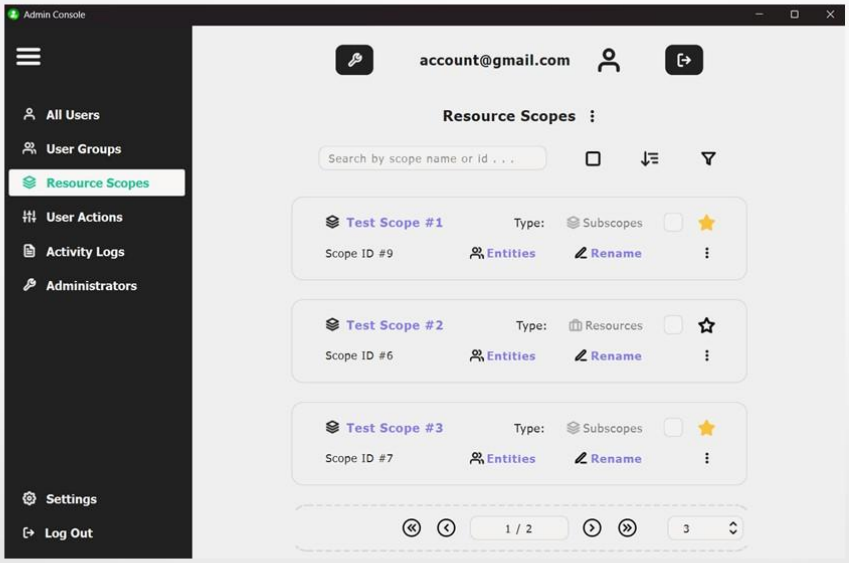
Демонстрація роботи інструменту адміністрування



Налаштування часу відкликання для учасників груп

15

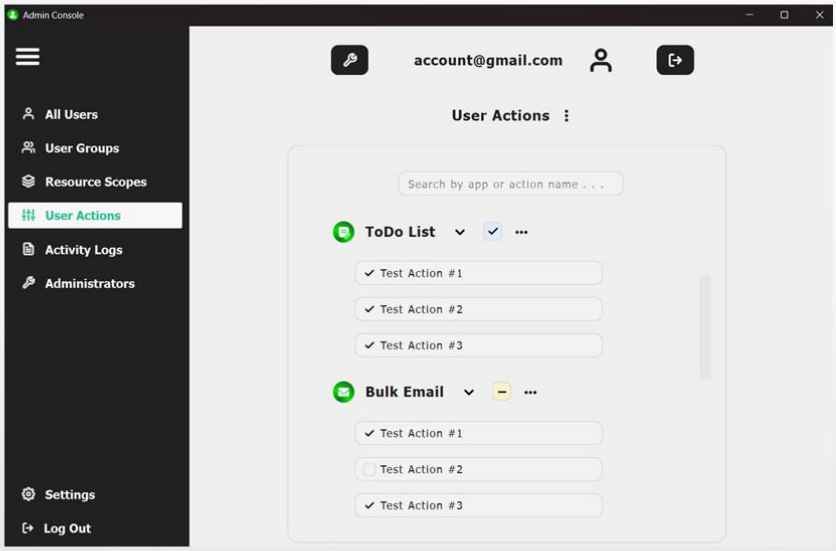
Демонстрація роботи інструменту адміністрування



Сторінка управління областями ресурсів

16

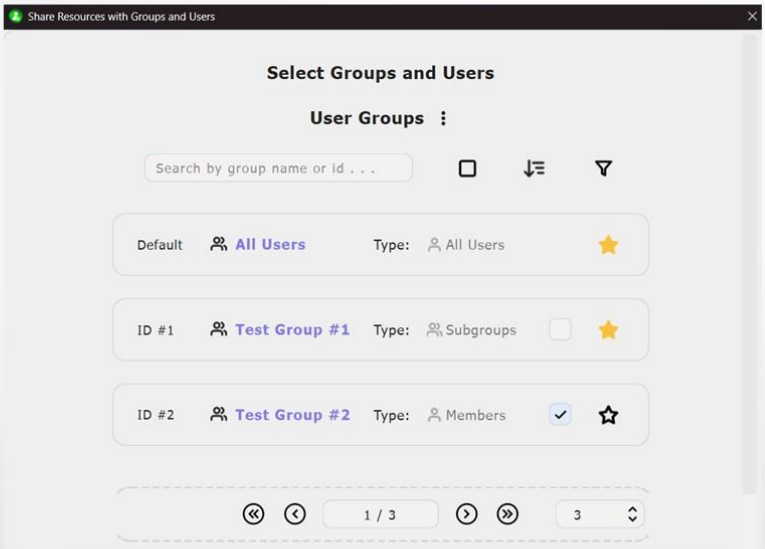
Демонстрація роботи інструменту адміністрування



Сторінка управління доступними діями користувачів

17

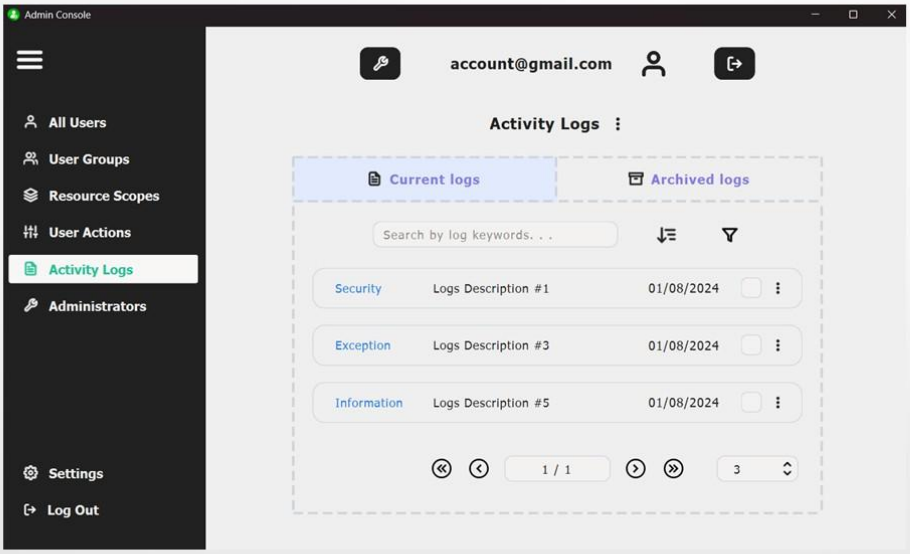
Демонстрація роботи інструменту адміністрування



Діалогове вікно налаштування прав доступу

18

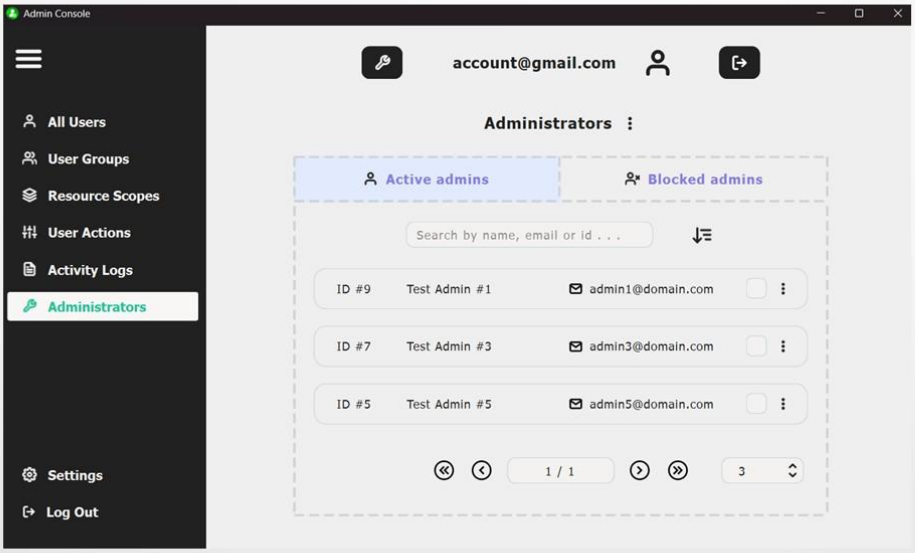
Демонстрація роботи інструменту адміністрування



Перегляд та аналіз логів активності

19

Демонстрація роботи інструменту адміністрування



Сторінка управління адміністраторами

20

Порівняння наявних рішень для розподілу ресурсів і керування правами доступу

Параметр	Admin Console	Azure AD	Google Cloud IAM
Багатофакторна аутентифікація (MFA)	+	+	+
Масштабованість	+	+	+
Оптимізація бюджету	+	–	–
Автоматизація рутинних процесів	+	–	+
Складність навчання роботи з рішенням	–	+	+
Гнучкість керуванням діями користувачів при роботі з вбудованими модулями програмного забезпечення	+	–	–
Незалежність від екосистеми Microsoft	+	–	+
Розробка власних інтеграцій через API	–	+	+
Інтуїтивний та гнучкий процес налаштування прав доступу	+	–	–

21

Порівняння наявних рішень для розподілу ресурсів і керування правами доступу

В результаті порівняння інструменту адміністрування «Admin Console» з наявними рішеннями, кількість переваг переважає у бік розробленого рішення, зокрема завдяки:

- автоматизації рутинних процесів (автоматизований процес відкликання учасників груп, налаштування прав доступу одночасно для декількох сутностей);
- гнучкості процесу налаштуванню прав доступу;
- інтуїтивно зрозумілому процесу налаштування прав доступу;
- зменшенню складності навчання роботи з розробленим рішенням.

22

Апробація результатів

Babenko Y. M., Kramarenko D. V., Administration tool for resource allocation and access rights management // XVIII International Conference “Modern Information and communication technologies on a transport, in industry and education” (Dnipro, 12-13 December, 2024), p.173.

23

Висновки

В ході роботи були виконані наступні задачі:

- Розглянуто особливості моделей керування доступом.
- Проаналізовано та порівняно моделі контролю доступу.
- Здійснено моделювання інструменту адміністрування для розподілу інформаційних ресурсів і керування правами доступу.
- Здійснено проектування бази даних інструменту адміністрування.
- Виконано програмну реалізацію інструменту адміністрування.
- Проаналізовано та порівняно наявні рішення для розподілу інформаційних ресурсів і керування правами доступу.

Таким чином, всі поставлені в роботі задачі виконано, мета роботи досягнена, роботу виконано.

24


Дякую за увагу!