

**МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ**  
Харківський національний університет імені В.Н.Каразіна  
Факультет математики і інформатики  
Кафедра теоретичної та прикладної інформатики

## **Кваліфікаційна робота**

### **бакалавр**

на тему: Адаптація підходу аналізу формальних понять для побудови  
системи обмеження доступу до ресурсів

Виконав: студент 4 курсу, групи  
МФ-42  
спеціальність 122 «Комп'ютерні  
науки»  
освітньо-професійна програма  
«Інформатика»

Козак Г. С.

---

— (прізвище та ініціали)  
Керівник Волков І. В.  
(прізвище та ініціали)

Рецензент

---

(прізвище та ініціали)

Харків – 2023 року

# ЗМІСТ

<b>ПЕРЕЛІК СКОРОЧЕНЬ, УМОВНИХ ПОЗНАЧЕНЬ, ТЕРМІНІВ.....</b>	<b>3</b>
<b>1. ВСТУП.....</b>	<b>4</b>
1.1 Мета та задачі дослідження.....	4
1.2 Актуальність теми.....	4
1.3 Методи дослідження.....	6
<b>2. ОСНОВНА ЧАСТИНА.....</b>	<b>7</b>
2.1 Постановка задачі.....	7
2.2 Ролі та дозволи.....	7
2.3 OpenAPI специфікація.....	9
2.3.1 Розширення OpenAPI специфікації.....	12
2.4 Решітки та порядок.....	13
2.5 Введення в FCA.....	15
2.5.1 Імплікації.....	19
2.5.2 Виведення імплікацій та імплікаційний базис.....	20
2.6 Огляд найбільш популярних інструментів для роботи з FCA.....	21
2.7 Побудова системи обмеження доступу з використанням розширеної OpenAPI. 24	
2.8 Опис методу виявлення неконсистентності системи обмеження доступу на рівні ролей.....	26
2.9 Застосування методу.....	27
2.10 Аналіз результатів.....	33
<b>3. ВИСНОВКИ.....</b>	<b>34</b>
<b>4. СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ.....</b>	<b>35</b>
<b>5. ДОДАТКИ.....</b>	<b>36</b>

## ПЕРЕЛІК СКОРОЧЕНЬ, УМОВНИХ ПОЗНАЧЕНЬ, ТЕРМІНІВ

FCA (Formal Concept Analysis) – аналіз формальних концептів.

API (Application Programming Interface) – це набір правил, протоколів та інструментів, які дозволяють різним програмним компонентам взаємодіяти між собою.

REST (Representational State Transfer) – це архітектурний стиль для проєктування розподілених систем, який базується на принципах простоти, стандартизації та масштабованості.

RESTful API – API розроблений за принципами REST архітектури, який використовує протокол HTTP для передачі даних.

Endpoint (ендпоінт) – відносний шлях (URL), який надає доступ до функціональності веб-сервісу.

JSON (JavaScript Object Notation) – формат збереження даних у вигляді пар “ключ: значення”.

JDK (Java development kit) – інструментарій розробника додатків на Java.

RBAC (Role-Based Access Control) – контроль доступу, заснований на ролях.

# 1. ВСТУП

## 1.1 Мета та задачі дослідження

Метою цієї роботи є адаптація підходу аналізу формальних понять щодо побудови системи обмеження доступу до ресурсів з використанням розширеної OpenAPI специфікації.

Задля досягнення мети дипломної роботи були поставлені наступні завдання:

- ознайомитися з FCA;
- проаналізувати існуючі підходи використання аналізу формальних понять щодо побудови системи обмеження доступу до ресурсів;
- розглянути існуючі інструменти побудови решіток формальних концептів;
- ознайомитися з OpenAPI специфікацією та її структурою;
- розширити OpenAPI, щоб мати можливість застосовувати FCA;
- описати метод виявлення неконсистентності системи обмеження доступу на рівні ролей використовуючи FCA.

## 1.2 Актуальність теми

OpenAPI специфікація де-факто є стандартом для розробки публічного API, зокрема RESTful чи інших API поверх протоколу HTTP. Різноманітні підходи побудови систем обмеження доступу до ресурсів спираються або на низькорівневі підходи, або розглядають доступ до таких різновидів ресурсів як ендпоінти, як до ізольованого ресурсу не враховуючи обмеження на результат, що повертає ендпоінт. Слід зазначити, що в OpenAPI специфікації є поняття Security Schemes, але цей

опис не дає чіткого зв'язку між ендпоінтами та відповідями (responses). Відсутність узагальненого підходу щодо цих обмежень може призвести до неконсистентності в системі обмеження доступу.

Розглянемо наступний приклад: у процесі проектування було встановлено, що

- ендпоінт “POST /store/order” матиме наступні обмеження доступу: Development Engineer, QA Engineer, Customer Support;
- ендпоінт повертатиме такий об'єкт:

```
{
  "id": 112,
  "gameId": 32523,
  "quantity": 2,
  "shipDate": "2023-01-12T09:19:25.683Z",
  "status": "approved",
  "complete": false
}
```

В процесі життєвого циклу продукту структура, що повертається ендпоінтом була змінена таким чином:

```
{
  "id": 112,
  "gameId": 32523,
  "quantity": 2,
  "shipDate": "2023-01-12T09:19:25.683Z",
  "creditCard": "4266342681942102"
  "status": "approved",
  "complete": false
}
```

Зміни в структурі розширили перелік повертаємих даних так, що ролі Development Engineer, QA Engineer більше не відповідають рівню обмеження (було додано чутливе поле, яке зберігає номер банківської карти). Таким чином консистентність обмеження доступу було порушено. У цій роботі пропонується, зокрема, розширити існуючу OpenAPI та додати можливість формалізувати обмеження доступу на рівні окремих

ролей як для ендпоінтів, так і для вмісту відповідей, що повертають ендпоінти при виконанні методів, що дозволить в наступному вчасно виявляти порушення консистентності обмеження доступу.

**Об'єктом дослідження** є побудова системи обмеження доступу до ресурсів.

**Предметом дослідження** є розширення OpenAPI специфікації для побудови системи обмеження доступу до ресурсів та використання FSA для забезпечення її консистентності.

### 1.3 Методи дослідження

Для вирішення поставлених завдань та досягнення мети роботи було використано наступні методи дослідження:

- Вивчення та аналіз літератури. Було розглянуто існуючі книги, статті та публікації з аналізу формальних концептів та підходів щодо обмеження доступу до ресурсів.
- Програмні інструменти. Був проведений огляд та порівняння програмних інструментів, якими можна користуватися для використання методів FSA та перевірки гіпотез дослідження.
- Аналіз документації. Було проведено вивчення OpenAPI специфікації, її компонентів, структури та можливостей для вирішення завдань роботи.
- Аналіз результатів та висновки. Було проведено аналіз результатів, отриманих під час виконання практичної частини роботи та були зроблені висновки щодо досягнення поставленої мети та подальшого розвитку ідей та підходів наведених в ході дослідження.

## 2. ОСНОВНА ЧАСТИНА

### 2.1 Постановка задачі

Основною задачею роботи є побудова системи обмеження доступу до ресурсів за допомогою розширення існуючої OpenAPI специфікації, щоб мати можливість призначати рівні доступу безпосередньо програмним компонентам. Також наводиться опис та застосування методу виявлення неконсистентності системи обмеження доступу на рівні ролей з використанням FCA.

### 2.2 Ролі та дозволи

Розглянемо підхід щодо управління контролем доступу на прикладі моделі Role-Based Access Control (RBAC), яка спрощує адміністрування безпеки великих інформаційних систем.

RBAC є поширеною моделлю управління доступом, яка забезпечує структурований підхід до керування дозволами та привілеями доступу в інформаційних системах організації. В RBAC рішення щодо управління доступом базуються на ролях, а не на окремих користувачах. Ролі являють собою конкретну робочу функцію, відповідальність або посаду в організації. Користувачам призначаються одна або декілька ролей відповідно до їх посадових вимог, а дозволи доступу асоціюються з цими ролями. Це спрощує управління контролем доступу, дозволяючи системним адміністраторам визначати дозволи на рівні ролей замість окремого призначення дозволів кожному користувачу.

Модель RBAC складається з наступних ключових компонентів:

- ролі являють собою різні функції або обов'язки, які залежать від предметної області. З кожною роллю пов'язана колекція дозволів, що

визначають дії, які можуть виконувати користувачі, призначені на цю роль. Користувачі можуть легко перепризначатися на різні ролі залежно від зміни вимог;

- дозволи визначають конкретні дії або операції, які може виконувати користувач з певною роллю. Вони можуть містити читання, запис, видалення, виконання або будь-які інші дії, що стосуються системи або додатку;
- користувачі – це люди, які взаємодіють з системою. Їм призначається одна або декілька ролей в залежності від їх вимог та обов'язків;
- ієрархія ролей: RBAC дозволяє створити ієрархію ролей. Ця ієрархія встановлює відношення між ролями, дозволяючи успадковувати дозволи. Ролі вищого рівня успадковують дозволи ролей нижчого рівня, що спрощує управління контролем доступу. Такі відношення між ролями можуть забезпечувати дотримання політик безпеки, включаючи розділення обов'язків і делегування повноважень;
- політики управління доступом: RBAC використовує політики контролю доступу для керування взаємодією між ролями та дозволами. Ці політики визначають правила й умови, за яких доступ надається або забороняється. Вони гарантують, що користувачі можуть виконувати лише дії, пов'язані з призначеними їм ролями.

Переваги RBAC включають підвищену безпеку, спрощене адміністрування та масштабованість. RBAC допомагає організаціям застосовувати принцип найменших привілеїв, надаючи користувачам лише необхідні права доступу відповідно до їх ролей. Вона також спрощує процес адміністрування шляхом централізації керування дозволами на рівні ролей, знижуючи складність керування дозволами окремих користувачів.



Загалом, RBAC забезпечує гнучкий і ефективний підхід до управління доступом, дозволяючи організаціям зберігати контроль над своїми інформаційними системами та надавати користувачам відповідні рівні доступу залежно від їх ролей і обов'язків. Докладніше про моделі RBAC можна прочитати в [1].

У нашому дослідженні ми використовуємо більш загальний підхід і не будемо використовувати дозволи доступу, а будемо працювати з відповідними ролями, безпосередньо визначаючи ролі для ендпоінтів та відповідей, які можуть повернути ендпоінти при виконанні методів. Дозволи, що відповідають ролі, задаються специфікацією.

Оскільки ми працюємо в межах специфікації OpenAPI, ми можемо неявно задавати модель дозволів доступу при описі ролей для ендпоінтів та об'єктів бізнес-логіки, що повертаються разом з відповіддю при виконанні операцій ендпоінту.

### **2.3 OpenAPI специфікація**

OpenAPI специфікація (раніше відома як специфікація Swagger) – це мова опису та форматування документації для RESTful веб-сервісів або API. Це відкритий стандарт, розроблений для опису, документування, тестування та взаємодії з API. OpenAPI специфікація використовує формат JSON або YAML (YAML Ain't Markup Language) для опису різних аспектів API, включаючи ендпоінти, параметри, запити, відповіді, моделі даних, аутентифікацію та інші аспекти API.

Основні компоненти OpenAPI специфікації включають:

- метадані описують основну інформацію про версію API, назву, опис, автора, контактні дані та іншу супутню інформацію;

- ендпоінти описують різні шляхи ендпоінтів, доступних у API, та відповідні HTTP-методи (GET, POST, PUT, DELETE і т.д.) для виконання операцій;
- параметри представляють різні параметри, які можна передавати в запитах до API, такі як параметри шляху (path parameters), параметри запиту (query parameters), заголовки (headers), параметри тіла запиту (request body parameters) та інші;
- відповіді описують можливі відповіді, які API може повертати при виконанні операцій, включаючи коди стану HTTP, заголовки та приклади даних відповідей;
- моделі даних визначають структуру даних, що використовуються в запитах та відповідях API, за допомогою схем даних (наприклад, JSON Schema або XML Schema);
- аутентифікація та авторизація описують механізми аутентифікації та авторизації, які потрібні для доступу до API, включаючи токени, ключі доступу та інші методи аутентифікації;
- приклади запитів та відповідей для наочного представлення використання API.

OpenAPI специфікація дозволяє розробникам легко розуміти, використовувати та взаємодіяти з API, а також автоматично генерувати клієнтський код, документацію та інструменти для тестування API. Це сприяє створенню зрозумілих та самодокументуючих API, спрощує їх розробку та інтеграцію, а також підвищує рівень взаємодії між розробниками та користувачами API.

Широко використовуваною конструкцією в OpenAPI специфікації є посилання “\$ref”, яке дозволяє посилатися на визначення (definition) або

компонент (component) всередині документа. Воно використовується для повторного використання та об'єднання частин специфікації, що робить її більш модульною, зрозумілою та легко підтримуваною.

В OpenAPI специфікації, ендпоінт – це конкретна кінцева точка входу (URL) веб-сервісу або API, до якої клієнти можуть звернутися для виконання певної операції або отримання доступу до певних ресурсів. Структура ендпоінту в OpenAPI специфікації зазвичай виглядає наступним чином:

- шлях (path) визначає конкретний URI, за допомогою якого можна звернутися до ендпоінту. Наприклад, “/customers” або “/products/{id}”;
- метод (method) вказує на HTTP-метод, який клієнт може використовувати для взаємодії з ендпоінтом;
- опис (description) – опціональне поле, яке містить текстовий опис ендпоінту та його функціональності;
- параметри (parameters): ендпоінт може містити параметри шляху, параметри запиту, заголовки та інші дані, які передаються в запиті до ендпоінту;
- тіло запиту (request body) – опціональне поле, яке визначає структуру та формат даних, що передаються в тілі запиту до ендпоінту. Наприклад, JSON або XML;
- тіло відповіді (response body) описує структуру та формат даних, що повертаються сервером у відповідь на запит до ендпоінту. Це може бути JSON, XML, файли та інше;

- коди відповідей (response codes) представляють можливі HTTP-коди стану, які можуть бути повернуті сервером у відповідь на запит до ендпоінту. Наприклад, 200 (OK), 404 (Not Found), 500 (Internal Server Error) та інші;
- аутентифікація та авторизація (authentication & authorization) визначають методи аутентифікації та авторизації, які повинні бути застосовані для доступу до ендпоінту, якщо такі механізми необхідні.

Це основні елементи структури ендпоінту в специфікації OpenAPI. Вони дозволяють описати інтерфейс API, визначити доступні ендпоінти, їх параметри та очікувані відповіді. Для глибшого розуміння OpenAPI та її структури звернутися до джерела [5].

### 2.3.1 Розширення OpenAPI специфікації

Специфікація OpenAPI надає можливість розширення її схеми шляхом додавання так званих розширень (extensions). Розширення дозволяють додавати додаткові метадані або функціональність до специфікації, не порушуючи її сумісність. Властивості розширень описуються за допомогою шаблонних полів, які називаються з префіксом "x-". Префікс вказує, що поле є розширенням і не є частиною стандартної специфікації. Слід пам'ятати, що є зарезервовані назви полів, які починаються з префікса "x-", наприклад, поля, що починаються з "x-oai" і "x-oas". Значенням поля розширення може бути null, масив, примітив або об'єкт. Таким чином, ми можемо додати поле розширення до специфікації на рівні ендпоінту, яке буде описувати його роль, наприклад, "x-endpoint-role", і додати поле на рівні об'єктів бізнес-логіки, які повертає ендпоінт разом з відповіддю – "x-payload-role". Розширення в специфікації

OpenAPI забезпечують гнучкість і можливість адаптувати специфікацію під конкретні вимоги проєкту. Однак, при використанні розширень слід бути уважним і переконатися, що вони сумісні з інструментами, які будуть використовувати специфікацію, або навіть написати власний парсер OpenAPI, який буде повертати значення полів розширень.

## 2.4 Решітки та порядок

**Визначення 1:** Бінарне відношення “ $\leq$ ” на множині  $C$  називається частковим порядком, якщо вона задовольняє наступні умови для всіх елементів  $x, y, z \in C$ :

- рефлексивність:  $x \leq x$ ;
- антисиметричність:  $x \leq y$  та  $y \leq x \Rightarrow x = y$ ;
- транзитивність:  $x \leq y$  та  $y \leq z \Rightarrow x \leq z$ .

**Визначення 2:**  $a$  називається нижнім сусідом (lower neighbor)  $b$ , якщо  $a < b$  і немає елементу  $c$ , що задовольняє нерівність  $a < c < b$ . В такому випадку  $b$  є верхнім сусідом (upper neighbor)  $a$  та записується  $a < b$ . Кожна скінченна впорядкована множина  $(C, \leq)$  може бути представлена у вигляді лінійної діаграми, яку також називають діаграмою Хассе. Якщо  $x, y \in C$ , при цьому  $x < y$ , то вершина, що відповідає  $y$  буде зображена над вершиною, що відповідає  $x$  та дві вершини з’єднуються відрізком прямої на діаграмі. З такої діаграми ми можемо зчитувати відношення порядку таким чином:  $x < y$  тоді й тільки тоді, коли вершина, що представляє  $y$ , може бути досягнута у висхідному шляху з вершини, що представляє  $x$ .

**Визначення 3:** Два елементи  $x, y$  впорядкованої множини  $(C, \leq)$

називаються порівняними якщо  $x \leq y$  або  $x \geq y$ , інакше вони непорівнянні.

**Визначення 4:** Нехай  $(C, \leq)$  є впорядкованою множиною і  $T$  є підмножиною  $C$ . Нижньою границею (lower bound)  $T$  називається елемент  $s$  з  $C$  такий, що  $s \leq t$  для всіх  $t \in T$ . Верхня границя  $T$  визначається дуально. Якщо існує найбільший елемент множини всіх нижніх границь  $T$ , він називається інфімум  $T$  та позначається  $\inf T$  або  $\bigwedge T$ . Аналогічно, найменша верхня границя називається супремум та позначається  $\sup T$  або  $\bigvee T$ . У випадку множини  $T = \{x, y\}$  з двох елементів пишемо  $x \wedge y$  для  $\inf T$  та  $x \vee y$  для  $\sup T$ . Інфімум і супремум часто також називають міт (meet) і джоін (join).

Решіткою  $(V, \leq)$  називають частково впорядковану множину, в якій супремум  $x \vee y$  та інфімум  $x \wedge y$  існує для кожної пари елементів  $x, y \in V$ . Як приклад можна навести нескінченну решітку  $(\mathbb{N}, \leq)$  натуральних чисел із заданим природним порядком “менше або дорівнює”.

**Визначення 5:** Решітка  $V$  називається повною, якщо супремум  $\bigvee X$  та інфімум  $\bigwedge X$  існує для будь-якої підмножини  $X \subseteq V$ . В кожній повній решітці  $V$  існує найбільший елемент  $\bigvee V$ , що називається одиницею (*unit*) решітки, позначається  $1_V$  і найменший елемент  $\bigwedge V$ , що називається нулем (*zero*) решітки, позначається  $0_V$ .

Наприклад, інтервал дійсних чисел з природнім відношенням порядку  $([0, 1], \leq)$  утворює повну решітку,  $x \vee y = \max(x, y)$ , а  $x \wedge y = \min(x, y)$ . З визначення повної решітки випливає, що супремум та інфімум існує для будь-якої підмножини  $X$ , зокрема  $X = \emptyset$ . Маємо  $\bigwedge \emptyset = 1_V$  та  $\bigvee \emptyset = 0_V$ , з цього випливає, що  $V \neq \emptyset$  для кожної повної решітки. Таким чином, кожна непорожня скінченна решітка є повною.

**Зауваження 1:** Можна відновити  $\leq$  відношення порядку решітки за допомогою операцій супремуму та інфімуму наступним чином:

$$x \leq y \Leftrightarrow x = x \wedge y \Leftrightarrow x \vee y = y$$

## 2.5 Введення в FCA

Аналіз формальних концептів (FCA) – це математичний фреймворк та метод для аналізу даних і представлення знань, що базується на теорії повних решіток. Основними компонентами FCA є формальний концепт та формальний контекст. Формальний контекст визначається трійкою  $K := (G, M, I)$ , де  $G$  – множина об'єктів,  $M$  – множина атрибутів контексту, а  $I$  задає бінарне відношення на  $G \times M$ . Запис  $(g, m) \in I$  або  $gIm$  читається як “у об'єкта  $g$  є атрибут  $m$ ”. Для того, щоб дати визначення формальному концепту необхідно ввести два оператори виведення (derivation operators). Для множини  $A \subseteq G$  об'єктів визначаємо:

$A' = \{m \in M \mid gIm, \forall g \in A\}$  – множина всіх атрибутів спільних для всіх об'єктів множини  $A$ .

Відповідно, для множини  $B \subseteq M$  атрибутів визначаємо:

$B' = \{g \in G \mid gIm, \forall m \in B\}$  – множина всіх об'єктів, що має всі атрибути з  $B$ .

Формальний концепт контексту  $K$  є парою  $(A, B)$ , для якої виконані наступні умови:  $A \subseteq G, B \subseteq M, A' = B, B' = A$ .  $A$  називають екстент та  $B$  називають інтент формального концепту  $(A, B)$ .

Формальний контекст  $K$  візуально подається у вигляді перехресної таблиці (cross table), прямокутної таблиці, у рядках якої перераховані назви об'єктів з  $G$ , а по стовпцях розташовані назви атрибутів з  $M$ , при цьому ми

позначаємо клітину  $(g, m)$  символом  $\times$  (в основному, можна використовувати інший символ), якщо об'єкт  $g$  має атрибут  $m$ , в іншому випадку клітина залишається порожньою. Перехресна таблиця є елементарною формою представлення даних.

Наведемо приклад таблиці, що представляє контекст різних напоїв та їх властивостей.

	напій	алкогольний	газований	неалкогольний	містить кофеїн	гарячий	Зроблений з винограду
Вино	$\times$	$\times$					$\times$
Шампанське	$\times$	$\times$	$\times$				$\times$
Пиво	$\times$	$\times$	$\times$				
Мінеральна вода	$\times$		$\times$	$\times$			
Трав'яний чай	$\times$			$\times$		$\times$	
Кола	$\times$		$\times$	$\times$	$\times$		
Кава	$\times$			$\times$	$\times$	$\times$	

Таблиця 1 – Приклад перехресної таблиці контексту



Для того, щоб множину концептів можна було впорядкувати, введемо наступне визначення.

**Визначення 6:** Нехай  $(A_1, B_1)$  та  $(A_2, B_2)$  – два формальні концепти одного формального контексту  $(G, M, I)$ .  $(A_1, B_1)$  є підконцептом (subconcept)  $(A_2, B_2)$  якщо  $A_1 \subseteq A_2$  (еквівалентно  $B_1 \supseteq B_2$ ). Позначається  $(A_1, B_1) \leq (A_2, B_2)$ . Відповідно,  $(A_2, B_2)$  називається суперконцептом (superconcept)  $(A_1, B_1)$ . Відношення " $\leq$ " для формальних концептів можна розуміти як " $(A_1, B_1)$  більш специфічний ніж  $(A_2, B_2)$ ", еквівалентно " $(A_2, B_2)$  загальніший ніж  $(A_1, B_1)$ ". Очевидно, що концепт є підконцептом і суперконцептом самого себе. В такому випадку кажуть, що відношення " $\leq$ " встановлює ієрархічний порядок на множині концептів.

**Визначення 7:** Нехай  $(A_1, B_1)$  та  $(A_2, B_2)$  – два формальні концепти контексту  $(G, M, I)$ . Найбільшим спільним підконцептом ( $\inf$ )  $(A_1, B_1) \wedge (A_2, B_2)$  називається концепт вигляду  $(A_1 \cap A_2, (B_1 \cup B_2)'' )$ .  $(B_1 \cup B_2)'' = ((B_1 \cup B_2)')'$ . Через те, що відображення  $X \rightarrow X''$  є оператором замикання, потужність результуючої множини може бути більшою за потужність об'єднання  $B_1$  та  $B_2$ .

Нехай  $(A_1, B_1)$  та  $(A_2, B_2)$  – концепти, тоді:

- $(A_1, B_1) \wedge (A_2, B_2)$  є концептом;
- $(A_1, B_1) \wedge (A_2, B_2) \leq (A_1, B_1)$ ;
- $(A_1, B_1) \wedge (A_2, B_2) \leq (A_2, B_2)$ ;
- Для будь-якого концепту  $(A, B)$  такого, що  $(A, B) \leq (A_1, B_1)$  та  $(A, B) \leq (A_2, B_2)$  випливає, що  $(A, B) \leq (A_1, B_1) \wedge (A_2, B_2)$ .

**Визначення 8:** Найменшим спільним суперконцептом (*sup*)

$(A_1, B_1) \vee (A_2, B_2)$  називається концепт вигляду  $((A_1 \cup A_2)'', B_1 \cap B_2)$ .

$(A_1 \cup A_2)'' = ((A_1 \cup A_2)')'$ . Для найменшого спільного суперконцепту

(*sup*) також задані властивості:

- $(A_1, B_1) \vee (A_2, B_2) \in \text{концептом};$
- $(A_1, B_1) \leq (A_1, B_1) \vee (A_2, B_2);$
- $(A_2, B_2) \leq (A_1, B_1) \vee (A_2, B_2);$
- Для будь-якого концепту  $(A, B)$  такого, що  $(A_1, B_1) \leq (A, B)$  та  $(A_2, B_2) \leq (A, B)$  випливає, що  $(A_1, B_1) \wedge (A_2, B_2) \leq (A, B)$ .

Множина  $\mathcal{B}(G, M, I)$  формальних концептів контексту  $(G, M, I)$ , впорядкованих за допомогою відношення " $\leq$ " називається решіткою цього контексту. На лінійній діаграмі решітки вершини, що представляють загальніші концепти розташовані вище вершин репрезентуючих більш специфічні.

Однією з основних задач FCA є знаходження екстенту та інтенту концепту. Для знаходження усіх об'єктів концепту необхідно записати об'єкти присвоєні вершинам, що можна досягти низхідними лінійними шляхами на діаграмі репрезентуючій решітку, починаючи з вершини, що представляє відповідний концепт. Слідуючи усім лінійним шляхам, що йдуть вгору від вершини, перерахуємо усі атрибути концепту.

**Зауваження 2:** Множиною всіх об'єктів є екстент одиниці решітки,  $(\emptyset', \emptyset'') = (G, G')$ . Аналогічно  $M$  – це інтент нуля решітки,  $(\emptyset'', \emptyset') = (M', M)$ .

### 2.5.1 Імплікації

Лінійні діаграми решіток демонструють, як атрибути пов'язані один з одним, і тому можна прочитати деякі залежності між атрибутами безпосередньо з них. Ці залежності (умови типу) "якщо ..., то..." означають, що якщо об'єкт має деякі атрибути, то він також має деякі інші атрибути. Такі залежності називаються імплікаціями. Наприклад, якщо трикутник рівносторонній, то він рівнобедрений; якщо число ділиться на 20, то воно ділиться на 10. Імплікації можуть бути правдивими або неправдивими щодо даних (датасету), і пошук імплікацій, які зберігають дані (датасет), є важливим завданням аналізу даних.

**Визначення 9:** Імплікація на множині  $M$  – це запис вигляду:  $A \rightarrow B$  (з  $A$  впливає  $B$ ), де  $A, B \subseteq M$ .  $A$  називається передумовою імплікації, а  $B$  – висновком.

**Визначення 10:**  $T \subseteq M$  є моделлю  $A \rightarrow B$  (моделює імплікацію), якщо  $A \not\subseteq T$  або  $B \subseteq T$ . Позначається  $T \models A \rightarrow B$ . Замість того, щоб казати  $T$  – модель, ми іноді кажемо, що  $T$  задовольняє імплікацію  $A \rightarrow B$ .

**Визначення 11:**  $T$  є моделлю (моделює) множини імплікацій  $L$ , якщо  $\forall A \rightarrow B \in L: T \models A \rightarrow B$ . Позначається  $T \models L$ .

Ми визначили імплікацію для довільної множини  $M$ , але як це пов'язано з FCA? Дамо наступне визначення:

**Визначення 12:** Нехай  $A, B \subseteq M$ ,  $A \rightarrow B$  дотримується в  $K = (G, M, I)$  якщо  $\forall g \in G: g' \models A \rightarrow B$ . Аналогічно можна визначити імплікації на множині об'єктів  $G$ .

**Зауваження 3:** Нехай  $K \models A \rightarrow B \Leftrightarrow \forall g \in A': B \subseteq g' \Leftrightarrow B \subseteq A'' \Leftrightarrow A' \subseteq B'$ .

### 2.5.2 Виведення імплікацій та імплікаційний базис

**Визначення 13:** Нехай  $L$  – множина імплікацій,  $A \rightarrow B$  семантично (логічно) випливає з  $L$ ,  $L \models A \rightarrow B$ , якщо  $Z \models A \rightarrow B$  для всіх  $Z \subseteq M$  таких, що  $Z \models L$ . Для  $X \subseteq M$  визначимо оператор замикання:

$L(X) = \cap \{Y \mid X \subseteq Y \subseteq M, Y \models L\}$ . Наведемо приклад,  $M = N$ ,  $L = \{\{i\} \rightarrow \{i + 1\} \mid i \in N\}$ , тоді  $L(\{0\}) = N$ .

**Зауваження 4:** Множина імплікацій валідних для формального контексту може бути дуже великою. В крайньому разі в контексті можуть зовсім бути відсутні об'єкти,  $G = \emptyset$ . Тоді  $A'' = M$  для будь-якої множини атрибутів  $A \subseteq M$ , оскільки множина об'єктів порожня, то не існує контрприкладу для будь-якої імплікації:  $A \rightarrow B$  для всіх  $A \subseteq M, B \subseteq M$ .

Припустимо, що  $|M| = n$ , тоді булеан  $M = 2^n$ , оскільки імплікації виражаються двома множинами виходить  $2^{2n}$  можливих імплікацій для даного контексту. Цікавий факт: усі імплікації такого контексту випливають з єдиної імплікації  $\emptyset \rightarrow M$ . Звісно, ми не хочемо шукати множину всіх імплікацій, що задовольняють наш контекст, нам потрібний мінімальний набір валідних імплікацій з якого всі інші валідні імплікації семантично випливають, така множина імплікацій називається імплікаційним базисом контексту  $(G, M, I)$  і повинна задовольняти умовам:

- здоровий глузд (soundness):  $A \rightarrow B \in L \Rightarrow (G, M, I) \models A \rightarrow B \Leftrightarrow B \subseteq A''$  (базис має містити тільки валідні імплікації для  $(G, M, I)$ );
- повнота:  $(G, M, I) \models A \rightarrow B \Rightarrow L \models A \rightarrow B$ ;
- ненадмірність:  $A \rightarrow B \in L \Rightarrow L \setminus \{A \rightarrow B\} \not\models A \rightarrow B$ .

**Зауваження 5:** В умовах можна замінити твердження  $(G, M, I) \models A \rightarrow B$  на еквівалентне  $B \subseteq A''$ .

## 2.6 Огляд найбільш популярних інструментів для роботи з FCA<sup>1</sup>

Програмні інструменти аналізу формальних концептів призначені для аналізу та дослідження даних з використанням методів FCA. Вони реалізують функціональність для створення та представлення формальних контекстів у вигляді перехресних таблиць, побудови та візуалізації відповідних їм решіток концептів, а також дозволяють обчислювати залежності між атрибутами та визначати схожість або відмінності між об'єктами. Інструменти FCA широко використовуються в різних галузях, включаючи аналіз даних, інформаційний пошук, управління знаннями, біоінформатику та інші. Вони допомагають дослідникам та аналітикам виявляти приховані закономірності, структури та взаємозв'язки в даних, уможливлуючи прийняття обґрунтованих рішень та формулювання висновків на основі аналізу. Інструменти FCA відрізняються за користувацьким інтерфейсом, складністю та наявністю додаткових функціональних можливостей. Усі перераховані інструменти є open-source проєктами.

**ConExp.** Concept Explorer – це найбільш популярний інструмент, який реалізує базову функціональність, необхідну студентам та дослідникам в галузі FCA. Має простий користувацький інтерфейс, написаний на Java, але не працює з OpenJDK. Необхідно використовувати OracleJDK версії Java не вище восьмої. За допомогою ConExp можливо:

- створювати та редагувати контексти;
- побудувати решітку концептів за контекстом;
- знаходити базис імплікацій, що задовольняють заданому контексту;
- знаходити базис асоціативних правил, які виконуються у заданому контексті;

---

<sup>1</sup> узятих з офіційного сайту Formal Concept Analysis Homepage.

- виконувати дослідження атрибутів.

ConExp дозволяє працювати з наступними форматами даних:

- *sex* – нативний формат ConExp, заснований на XML. Зберігає інформацію про контекст, лінійну діаграму решітки, а також про імплікації й асоціативні правила, якщо вони були обчислені. Рекомендується використовувати даний формат;
- *sxt* – у такому форматі можуть зберігатися тільки дані про контекст;
- *csv* – значення, розділені комами. Фактичним роздільником є крапка з комою (;). Передбачається, що перший рядок містить назви атрибутів, при цьому перша клітина порожня. Наприклад, якщо контекст складається з атрибутів *a*, *b*, *c*, тоді перший рядок буде виглядати так: “;*a;b;c*”. Кожний наступний рядок повинен починатися з назви об’єкта та подальшої за ним послідовності з 0 та 1. У клітині таблиці в якій стоїть 1, буде відображено перетин в імпортованому контексті;
- *oal* – список об’єктів та атрибутів. Кожний рядок представляє інформацію про те, які атрибути є у заданого об’єкта. Якщо у об’єкта *g* є атрибути *a*, *b*, *c*, тоді рядок для об’єкту *g* буде виглядати так: “*g:a;b;c*”.

**ConExp-FX.** ConExp-FX також написаний мовою Java є частковою переімплементациєю класичного інструменту ConExp Франческо Крігелем. Надає покращений графічний інтерфейс та сфокусований на взаємодії з користувачем. При зміні формального контексту решітка концептів швидко перебудовується, завдяки вбудованому алгоритму iFox. Відмінність ConExp-FX полягає в тому, що він використовує паралельні потоки для зниження часу виконання більшості алгоритмів. На сьогодні його можна

запустити з офіційного сайту при наявності Java восьмої версії або вищої. Підтримує імпорт у форматах: sxt, csv, nt та нативний формат cfx (контекст та решітка), експорт контексту в різних форматах: HTML, PDF, PNG, SVG, LaTeX. Графічний інтерфейс підтримує вирази LaTeX в об'єктах та атрибутах формального контексту.

**ConExp-NG.** ConExp-NG – переімплементация Concept Explorer студентами Роберта Яшке (Robert Jäschke), написана з використанням бібліотеки FcaLib. ConExp-NG – простий інструмент, орієнтований на зручність (usability) та графічний інтерфейс користувача. Розповсюджується у вигляді jar файлу. Інструментарій практично не відрізняється від Concept Explorer.

**Lattice Miner.** Lattice Miner – програмний інструмент FCA із простим і зрозумілим інтерфейсом, який дозволяє будувати, візуалізувати та маніпулювати (Галуа) решітками формальних концептів та обчислювати асоціативні правила. Підтримує імпорт даних у форматі cex, xls, xlsx, slf (Galicia SLF Binary Context) і спеціальний формат Lattice Miner – lmb (Lattice Miner Binary Context).

**Conexp-clj.** Даніель Борхман (Daniel Borchmann) запропонував власну реалізацію утиліти аналізу формальних концептів, засновану на Concept Explorer. Conexp-clj – універсальний програмний інструмент представлений у вигляді оболонки командного рядка мови програмування Closure, який активно розвивається та підтримується. Його метою є спрощення виконання складних завдань над набором даних, крім того, він має низку інших застосувань. Ви можете користуватися Conexp-clj після завантаження і запуску попередньо скомпільованих двійкових файлів (jar файлів). Для використання останньої версії Conexp-clj необхідно завантажити JDK версії 20. У командному рядку можна подивитись

довідкову інформацію з різних функцій, на GitHub знаходиться докладна документація до Conexp-clj з прикладами. Для візуалізації решіток концептів можна запустити мінімальну графічну оболонку з jar файлу, виконавши команду “java -jar conexp-clj-2.3.0-SNAPSHOT-standalone.jar -g”.

## **2.7 Побудова системи обмеження доступу з використанням розширеної OpenAPI**

Запропонована нами система обмеження доступу буде побудована за допомогою ендпоінтів та ролей.

Спочатку необхідно описати ролі. Для їх опису будемо використовувати частково впорядковану множину з відношенням порядку  $\subseteq$  дозволів доступу, тому що не всі ролі є порівняними, наприклад, адміністратор HTTP сервісу і FTP сервісу, вони не мають доступу один до одного, тому є який-небудь Owner який буде для них супремумом (містити в собі дозволи доступу цих ролей). Для множини ролей побудуємо ієрархію ролей  $R$ . В ієрархії будуть знаходитися суперролі, наприклад, admin та спеціальні ролі, що інтерпретуються як відсутність ролі, наприклад guest. На діаграмі вершина, що відповідає важливішій ролі розташована вище ніж вершина, що відповідає менш важливій ролі. Непорівнянні ролі будуть розташовані на одному рівні.



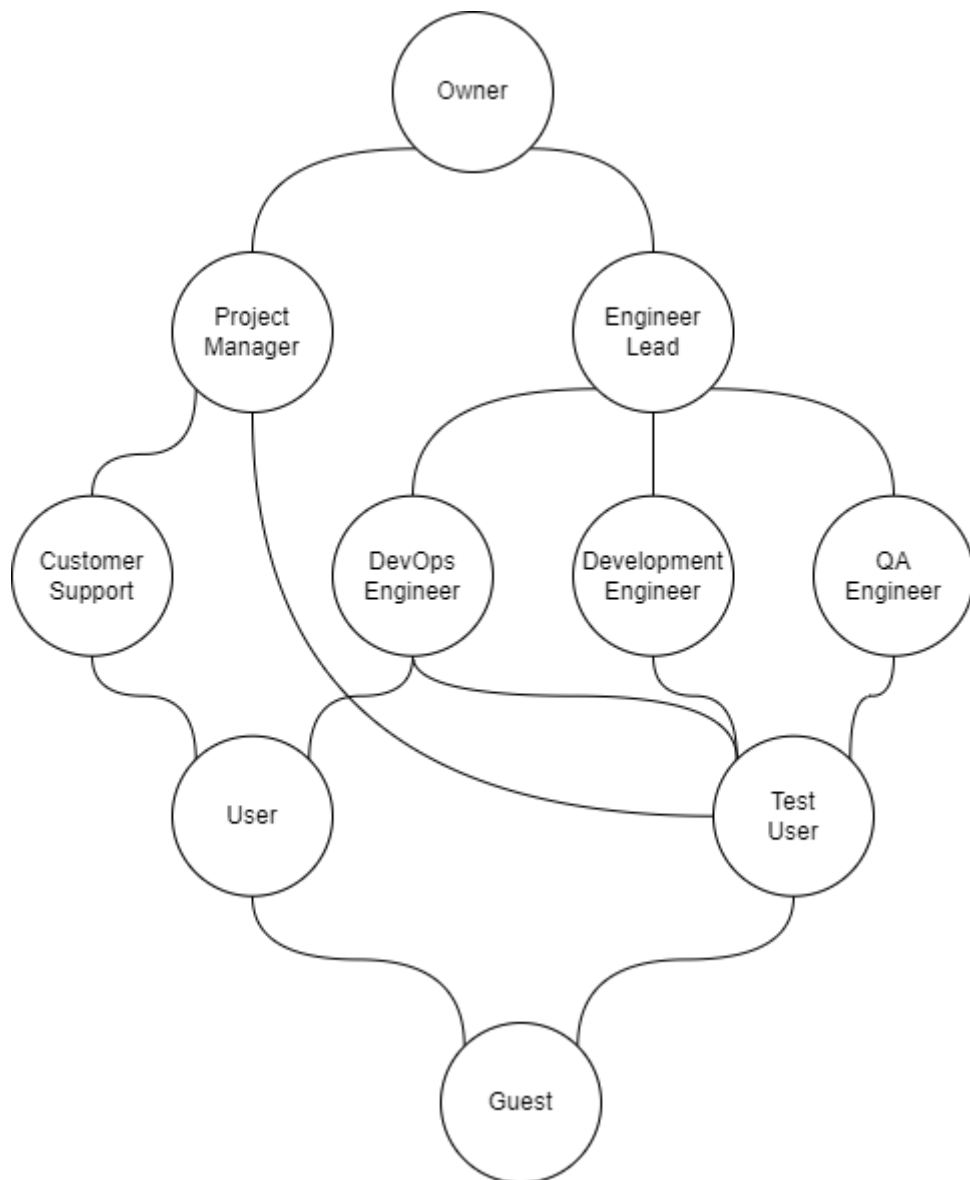


Рисунок 1 – Приклад ієрархії ролей

Ендпоінти визначаються специфікацією API. Маючи можливість розширити специфікацію OpenAPI, можемо призначати ролі для доступу до ендпоінтів і об'єктів бізнес-логіки, які повертаються ендпоінтами при виконанні HTTP-методів.

(\*) У нашій системі обмеження доступу задаємо правило: роль відповіді ендпоінту повинна бути не вищою за ієрархією ніж роль ендпоінту. Відповідь ендпоінту може і не містити об'єкт, а просто повертати HTTP-код стану та деякий текстовий опис відповіді. В такому

випадку немає сенсу накладати обмеження доступу на цю відповідь і ми не включатимемо її в контекст обмеження доступу. Роль відповіді визначається роллю поверненого об'єкту. Для композитного об'єкту відповіді ендпоінту, (умовною) роллю відповіді буде об'єднання ролей підоб'єктів.

## 2.8 Опис методу виявлення неконсистентності системи обмеження доступу на рівні ролей.

Знаючи структуру ендпоінтів та їх відповідей можна описати формальний контекст для системи обмеження доступу. Позначимо множину ендпоінтів як *Endpoint*, множину відповідей як *Response*, множину *Role<sub>1</sub>* ролей ендпоінтів та множину *Role<sub>2</sub>* ролей відповідей. Тоді

перший контекст для ролей ендпоінтів має вигляд

$K_1 = (Endpoint, Role_1, I_1)$ , де  $I_1 \subseteq Endpoint \times Role_1$ , відповідно

другий контекст для ролей відповідей –  $K_2 = (Response, Role_2, I_2)$ ,

$I_2 \subseteq Response \times Role_2$ .

Об'єднуємо контексти. Хочемо об'єднати контексти  $K_1$  та  $K_2$  і побудувати решітку об'єднаного контексту  $K$ . Об'єднання двох формальних контекстів може бути досягнуто шляхом об'єднання множин об'єктів і атрибутів кожного з них.

Множиною об'єктів результуючого об'єднаного контексту  $K$  буде множина  $Endpoint \cup Response$ .  $Role = Role_1 \cup Role_2$  — атрибути

контексту  $K$ . Визначаємо бінарне відношення

$I \subseteq (Endpoint \cup Response) \times Role$  нового контексту.

Таким чином отримали контекст обмеження доступу

$K = (Endpoint \cup Response, Role, I)$  об'єднання  $K_1$  та  $K_2$ .

Використаємо результати, отримані в статті [2], з відмінністю в тому, що потрібно робити валідацію не дозволів доступу, а ролей.

- будуємо  $B(\text{Endpoint} \cup \text{Response}, \text{Role}, I)$  решітку об'єднаного контексту  $K$ ;
- в термінах FCA правило (\*) можна сформулювати так: для решітки даного контексту  $K$  концепти, що містять у своєму екстенсті об'єкт  $\text{endpoint}$  множини  $\text{Endpoint}$  повинні бути підконцептами (специфічнішими, оскільки міститимуть ту ж саму або більшу кількість атрибутів (ролей)) концептів, що містять у своєму екстенсті  $\text{response}$  множини  $\text{Response}$ , що відповідає  $\text{endpoint}$ , еквівалентно  $\text{response}' \subseteq \text{endpoint}'$ ;
- рівні доступу будуть визначатися через максимальну роль в наборі атрибутів  $\text{endpoint}$  та  $\text{response}$  за ієрархією ролей  $R$ . Можливо, що  $\text{response}'$  не буде містити в собі супремум для всіх своїх ролей;
- знаходимо  $\text{endpoint}'$ ,  $\text{response}'$  обраних ендпоінту та відповіді ендпоінту. Перевіряємо, що  $\text{sup endpoint}'$  міститься в цій множині, а потім, що  $\text{sup response}'$  та  $\text{sup endpoint}'$  порівнянні та  $\text{sup response}' \leq \text{sup endpoint}'$  за ієрархією  $R$ . У такому разі ролі було призначено вірно у системі обмеження доступу.

## 2.9 Застосування методу

Ми вже розказали як можна додати в OpenAPI специфікацію, використовуючи поля розширення, опис ролей ендпоінтів та відповідей. Припустимо, що в нашому API є два ендпоінти: “GET /api/Meal/{id}”, який повертає разом з відповіддю об'єкт `MealDto`, в описі якого міститься посилання на об'єкт `RecipeDto` та “GET /api/Ingredients”, який повертає

масив об'єктів `IngredientDto`. Ролі для прикладу беремо з ієрархії ролей наведеної на Рисунку 1. Нехай ми визначили ролі для ендпоінтів таким чином, що ендпоінт “GET /api/Meal/{id}” матиме рівень доступу `Project Manager`, а “GET /api/Ingredients” – `Customer Support`, одночасно визначаємо рівні доступу для об'єктів: `MealDto` призначена роль `DevOps Engineer`, `RecipeDto` – `Customer Support` та для `IngredientDto` визначена роль `User`.

Опишемо структуру ендпоінтів цієї специфікації у форматі JSON:

```
{
  .....
  "paths": {
    "/api/Meal/{id}": {
      "get": {
        .....
        "x-endpoint-role": "Project Manager",
        "responses": {
          "200": {
            "content": {
              "application/json": {
                "schema": {
                  "$ref": "#/components/schemas/MealDto"
                }
              }
            }
          }
        }
      }
    },
    "/api/Ingredients": {
      "get": {
        .....
        "x-endpoint-role": "Customer Support",
        "responses": {
          "200": {
            "content": {
              "application/json": {
                "schema": {
                  "$ref": "#/components/schemas/IngredientsDto"
                }
              }
            }
          }
        }
      }
    }
  }
}
```

```

    }
  },
  "components": {
    "schemas": {
      "MealDto": {
        "x-payload-role": "DevOps Engineer",
        "type": "object",
        "properties": {
          .....
          "recipe": {
            "$ref": "#components/schemas/RecipeDto"
          }
        }
      },
      "RecipeDto": {
        "x-payload-role": "Customer Support",
        .....
      },
      "IngredientsDto": {
        "type": "object",
        "properties": {
          "ingredients": {
            "type": "array",
            "items": {
              "$ref": "#/components/schemas/IngredientDto"
            }
          }
        }
      },
      "IngredientDto": {
        "x-payload-role": "User",
        .....
      }
    }
  }
}

```

Побудуємо перехресну таблицю контексту ендпоінтів та їх ролей.

	guest	user	customer support	product manager
“GET /api/Meal/{id}”	✗	✗	✗	✗
“GET /api/Ingredients”	✗	✗	✗	

Таблиця 2 – Перехресна таблиця контексту ендпоінтів та ролей

Створимо та сформуємо перехресну таблицю для контексту відповідей ендпоінтів та відповідних їм ролей. Об'єкти таблиці умовно позначатимемо за шаблоном “HTTP-код стану назва об'єкта, що повертається”. Отже, об'єктами формального контексту будуть:

- “200 MealDto”. Оскільки об'єкт MealDto є композитним, тобто містить в описі посилання на об'єкт RecipeDto, тоді “200 MealDto” матиме атрибутами ролі MealDto та RecipeDto, тобто customer support та devOps engineer;
- “200 Ingredients”. Ця відповідь містить масив об'єктів IngredientDto, що мають роль User, отже об'єкт формального контексту “200 Ingredients” матиме атрибут user.

	guest	user	customer support	devOps engineer
“200 MealDto”	✗	✗	✗	✗
“200 IngredientsDto”	✗	✗		

Таблиця 3 – Перехресна таблиця контексту відповідей та ролей

Об'єднаємо два формальних контексти та отримаємо формальний контекст обмеження доступу.

	guest	user	customer support	project manager	devOps engineer
“GET /api/Meal/{id}”	×	×	×	×	
“GET /api/Ingredient”	×	×	×		
“200 MealDto”	×	×	×		×
“200 IngredientsDto”	×	×			

Таблиця 4 - Перехресна таблиця контексту обмеження доступу

Побудуємо діаграму решітки контексту обмеження доступу за допомогою програмного інструменту ConExp-NG:

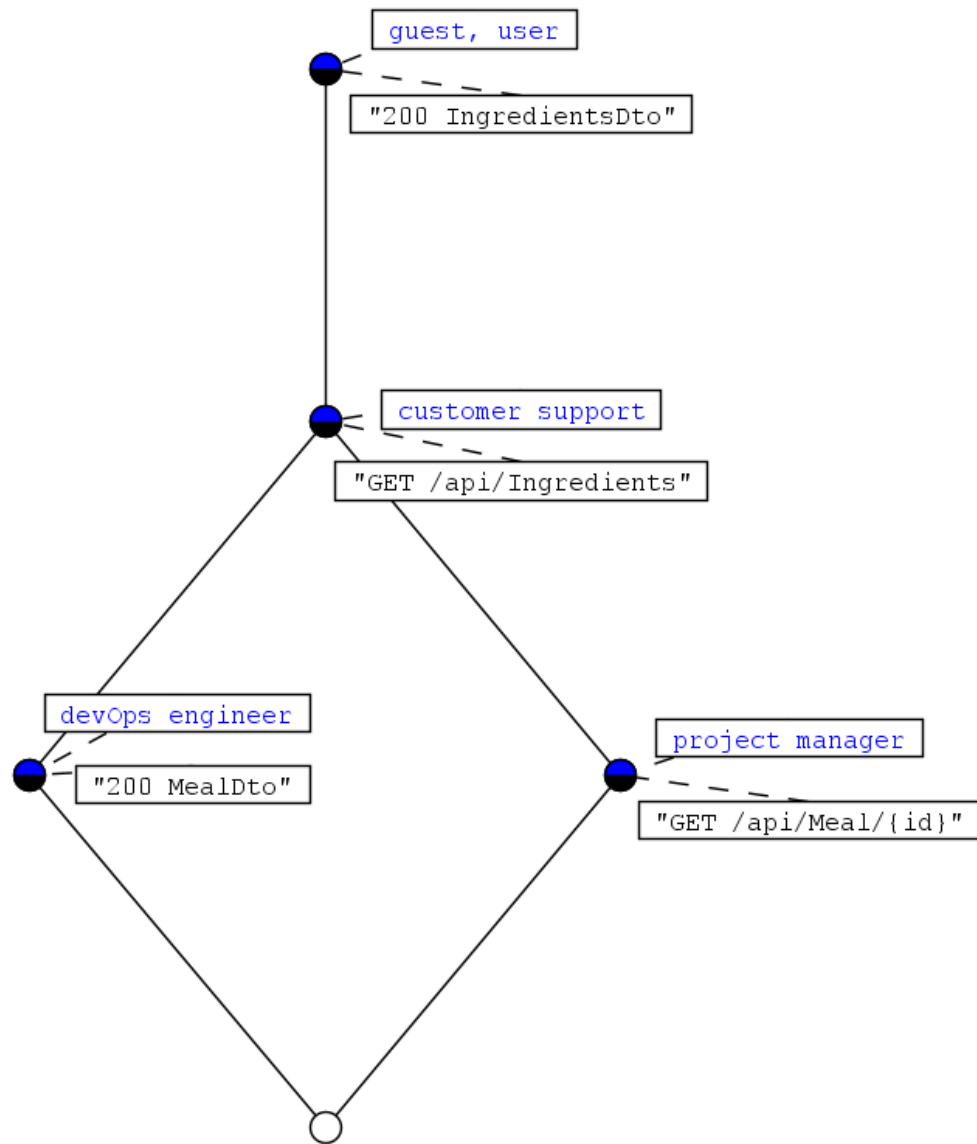


Рисунок 2 – Решітка контексту обмеження доступу

Перевіримо, чи не була порушена консистентність системи обмеження доступу. Розглянемо першу пару

(“GET /api/Ingredients”, “200 IngredientsDto”),

інтент {“GET /api/Ingredients”}' = {customer support, guest, user}



та інтент  $\{“200 IngredientsDto”\}' = \{guest, user\}$   
 $sup \{“GET /api/Ingredients”\}' \in \text{роль Customer Support},$   
 $sup \{“200 IngredientsDto”\}' = User.$

Оскільки  $User \leq Customer Support$ , тоді для цієї пари ролі призначені вірно.

Робимо перевірку для другої пари

$(“GET /api/Meal/{id}”, “200 MealDto”).$

Інтент ендпоінту

$\{“GET /api/Meal/{id}”\}' = \{guest, user, customer support, project manager\}$   
інтент відповіді

$\{“200 MealDto”\}' = \{guest, user, devOps engineer, customer support\};$

$sup \{“GET /api/Meal/{id}”\}' \in \text{роль Project Manager і}$

$sup \{“200 MealDto”\}'$  не міститься в цій множині, оскільки ролі DevOps Engineer та Customer Support непорівнянні за ієрархією ролей, тому  $sup \{“200 MealDto”\}' \in \text{роль Owner}$  яка включає дозволи DevOps Engineer та Customer Support. Отже, ролі для пари  $(“GET /api/Meal/{id}”, “200 MealDto”)$  були призначені невірно і консистентність системи обмеження доступу було порушено.

## 2.10 Аналіз результатів

В ході дослідження були отримані такі результати:

- вдалося розширити існуючу OpenAPI специфікацію, щоб додавати поля ролей доступу у структуру ендпоінтів та об'єктів, що

повертають ендпоінти при виконанні операцій та показано як це можна зробити на практиці;

- було описано метод перевірки правильності призначення ролей програмним сутностям та застосування методу на прикладі.

### 3. ВИСНОВКИ

У ході дослідницької роботи було виконано всі поставлені завдання. Основний текст роботи починається з теоретичної частини де надається загальна інформація про аналіз формальних концептів, підходи щодо управління контролем доступу та основні компоненти і структуру OpenAPI специфікації.

Практична частина цієї роботи дає детальний опис як застосувати теоретичний матеріал на практиці для розв'язання поставленої задачі. На прикладі було показано як розширити OpenAPI специфікацію та призначати ролі доступу ендпоінтам та об'єктам бізнес логіки, що повертають ендпоінти при виконанні HTTP-методів. Було надано огляд низки програмних інструментів, якими можна користуватися, щоб будувати та візуалізувати решітки формальних концептів. Таким чином, використовуючи зазначені компоненти ми показали як можна побудувати систему обмеження доступу до ресурсів з використанням розширеної OpenAPI специфікації та підтримувати її консистентність застосовуючи аналіз формальних концептів.

Отже, підхід щодо управління контролем доступу, запропонований в ході дослідження, є перспективним та потребує подальшої роботи та уточнення деталей реалізації, щоб його можна було повноцінно використовувати в реальних проєктах.

## 4. СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Ravi S. Sandhu' , Edward J. Coynek , Hal L. Feinsteink and Charles E. Youmank, Role-Based Access Control Models, IEEE Computer, Volume 29, Number 2, 38-47 (1996).
2. Ganter, B., Wille, R., Formal Concept Analysis. Mathematical foundations (1999).
3. Sobieski, S., Zielinski, B., Modeling Role Hierarchy Structure using The Formal Concept Analysis. Annales UMCS Informatica 2, 143–159 (2010).
4. Ch. Aswani Kumar, Modeling Access Permissions in Role Based Access Control Using Formal Concept Analysis, ICIP. Wireless Networks and Computational Intelligence pp 578–583 (2012).
5. OpenAPI Specification version 3.1.0.

## 5. ДОДАТКИ

Додаток 1. Приклад опису структури ендпоінтів у форматі JSON.

```
{
  "openapi": "3.0.1",
  "info": {},
  "paths": {
    "/api/FamilyMember": {
      "post": {
        "tags": [
          "FamilyMember"
        ],
        "summary": "Saves a user's family member",
        "description": "Set 'ExternalUserId' to null if adding new member and set all properties, otherwise set 'ExternalUserId' to real external user id and leave all properties except 'Info' and 'FamilyId' empty",
        "requestBody": {
          "content": {
            "application/json-patch+json": {
              "schema": {
                "$ref": "#/components/schemas/AddFamilyMemberDto"
              }
            },
            "application/json": {
              "schema": {
                "$ref": "#/components/schemas/AddFamilyMemberDto"
              }
            },
            "text/json": {
              "schema": {
                "$ref": "#/components/schemas/AddFamilyMemberDto"
              }
            },
            "application/*+json": {
              "schema": {
                "$ref": "#/components/schemas/AddFamilyMemberDto"
              }
            }
          }
        },
        "responses": {
          "200": {
            "description": "Success",
            "content": {
              "text/plain": {
                "schema": {
                  "type": "boolean"
                }
              },
              "application/json": {
                "schema": {
```

```

        "type": "boolean"
      }
    },
    "text/json": {
      "schema": {
        "type": "boolean"
      }
    }
  },
  "400": {
    "description": "Error during saving users's family member"
  },
  "401": {
    "description": "User was not authorized"
  }
},
"put": {
  "tags": [
    "FamilyMember"
  ],
  "summary": "Updates a user's family member",
  "requestBody": {
    "content": {
      "application/json-patch+json": {
        "schema": {
          "$ref": "#/components/schemas/UpdateFamilyMemberDto"
        }
      },
      "application/json": {
        "schema": {
          "$ref": "#/components/schemas/UpdateFamilyMemberDto"
        }
      },
      "text/json": {
        "schema": {
          "$ref": "#/components/schemas/UpdateFamilyMemberDto"
        }
      },
      "application/*+json": {
        "schema": {
          "$ref": "#/components/schemas/UpdateFamilyMemberDto"
        }
      }
    }
  },
  "responses": {
    "200": {
      "description": "Success",
      "content": {
        "text/plain": {

```

```

        "schema": {
          "type": "boolean"
        }
      },
      "application/json": {
        "schema": {
          "type": "boolean"
        }
      },
      "text/json": {
        "schema": {
          "type": "boolean"
        }
      }
    }
  },
  "400": {
    "description": "Error during updating users's family member"
  },
  "401": {
    "description": "User was not authorized"
  }
}
}
},
"components": {
  "schemas": {
    "AddFamilyMemberDto": {
      "type": "object",
      "properties": {
        "name": {
          "type": "string",
          "nullable": false
        },
        "userName": {
          "type": "string",
          "nullable": false
        },
        "dob": {
          "type": "string",
          "format": "date-time",
          "nullable": true
        },
        "info": {
          "type": "string",
          "nullable": true
        },
        "externalUserId": {
          "type": "integer",
          "format": "int32",
          "nullable": true
        }
      }
    }
  }
}

```

```

    },
    "familyId": {
      "type": "integer",
      "format": "int32"
    }
  },
  "additionalProperties": false
},
"UpdateFamilyMemberDto": {
  "type": "object",
  "properties": {
    "id": {
      "type": "integer",
      "format": "int32"
    },
    "info": {
      "type": "string",
      "nullable": true
    }
  },
  "additionalProperties": false
}
}
}
}

```