Київський національний університет імені Тараса Шевченка
Факультет інформаційних технологій
Кафедра програмних систем і технологій

УДК 004.421

ВИПУСКНА КВАЛІФІКАЦІЙНА МАГІСТЕРСЬКА РОБОТА

Тема: Синтез архітектури контролю доступу для багатоклієнтських хмарних систем

Спеціальність 121 “Інженерія програмного забезпечення”

ПОЯСНЮВАЛЬНА ЗАПИСКА

(позначення)

Студент
Згура Назар Анатолійович

_____

Науковий керівник
док. тех. наук, доцент
Порєв Геннадій Володимирович

_____

Допускається до захисту з питань нормоконтролю
Завідувач кафедри
док. тех. наук, професор
Бичков Олексій Сергійович

_____

КИЇВ
2022

АНОТАЦІЯ

Випускна кваліфікаційна робота містить 63 сторіноки, 5 рисунків, 13 джерел та 2 додатки.

Тема: Синтез архітектури контролю доступу для багатоклієнтських хмарних систем

Об'єкт дослідження: Багатоклієнтські хмарні системи**.**

Мета роботи: Синтез архітектури гнучкої системи контролю доступу для багатоклієнтських SaaS систем.

Предмет дослідження: Архітектура контролю доступу**.**

Результати дослідження: Синтезована архітектура контролю доступу.

Висновок: Реалізовано прототип синтезованої архітектури.

Ключові слова: ХМАРНА СИСТЕМА, БАГАТОКЛІЄНТСЬКА СИСТЕМА, КОНТРОЛЬ ДОСТУПУ, ХМАРНЕ СЕРЕДОВИЩЕ AZURE, ПОЛІТИКИ ДОСТУПУ

ANNOTATION

The final qualifying work contains 63 pages, 5 drawings, 13 sources and 2 appendices.

Topic: Synthesis of access control architecture for multi-tenant cloud systems

Object of research: Multi-tenant cloud systems.

Objective: Synthesis of a flexible access control system architecture for multi-tenant SaaS systems.

Subject of research: Access control architecture.

Research results: Synthesized access control architecture.

Conclusion: The prototype of the synthesized architecture is developed.

Keywords: CLOUD SYSTEM, MULTI-TENANT SYSTEM, ACCESS CONTROL, AZURE CLOUD ENVIRONMENT, ACCESS POLICIES

ЗМІСТ

ЗМІСТ 2

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ 4

ВСТУП 6

РОЗДІЛ 1 ЗАДАЧА СИНТЕЗУ АРХІТЕКТУРИ КОНТРОЛЮ ДОСТУПУ 7

1.1. Опис особливостей багатоклієнтських хмарних систем 7

1.2. Проблематика забезпечення контролю доступу в хмарних системах 10

1.2.1. Обов’язковий контроль доступу 12

1.2.2. Дискреційний контроль доступу 13

1.2.3. Контроль доступу на основі ролей 13

1.2.4. Контроль доступу на основі атрибутів 15

1.2.5. Огляд обмежень існуючих моделей 17

1.3. Огляд методик та засобів синтезу архітектури інформаційних систем 18

1.3.1. Метод синтезу архітектури TOGAF 18

1.3.2. Документування та опис архітектури 20

РОЗДІЛ 2 СИНТЕЗ АРХІТЕКТУРИ КОНТРОЛЮ ДОСТУПУ ДЛЯ БАГАТОКЛІЄНТСЬКИХ ХМАРНИХ СИСТЕМ 21

2.1. Опис прикладу бізнес сценарію, який вирішує архітектура 21

2.2. Вимоги до архітектури 22

2.3. Опис розробленої моделі контролю доступу 23

2.3.1. Використані теоретичні моделі та концепції 23

2.3.2. Опис механізму роботи трирівневої моделі контролю доступу 24

ВИСНОВОК 30

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ 31

ДОДАТКИ 32

Додаток А. Лістинг коду модулю каталогу клієнтів 32

Додаток Б. Software Architecture Document 34

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

SaaS	**Software-as-a-Service**	Програмне забезпечення як послуга
MAC	**Mandatory Access Control**	Обов'язковий контроль доступу
DAC	**Discretionary Access Control**	Дискреційний контроль доступу
RBAC	**Role-Based Access Control**	Контроль доступу на основі ролей
ABAC	**Attribute-Based Access Control**	Контроль доступу на основі атрибутів
MLS	**Multi-Level Security**	Багаторівнева безпека
PAP	**Policy Administration Point**	Точка адміністрування політики
PEP	**Policy Enforcement Point**	Точка виконання політики
PDP	**Policy Decision Point**	Точка прийняття рішень
PIP	**Policy Information Point**	Точка інформації про політику
LDAP	**Lightweight Directory Access Protocol**	Полегшений протокол доступу до директорій / каталогів

TOGAF	**The Open Group Architecture Framework**	Архітектура Open Group Framework
ADM	**Architecture Development Method**	Метод розробки архітектури

ВСТУП

Програмне забезпечення як послуга (SaaS) стало однією з провідних тенденцій в галузі хмарних обчислень за останні кілька років. Постачальники SaaS орієнтуються на багатоклієнтське програмне забезпечення (multi-tenancy), в рамках якого всі клієнти спільно використовують екземпляри програмного забезпечення, платформу та сховища даних з метою найбільш оптимального використання хмарних ресурсів. Для багатоклієнтських SaaS систем, контроль доступу в більшості випадків організований лише на рівні програмного коду системи (application-level). Вказаний механізм контролю доступу є достатньо комплексним по суті, оскільки як постачальники, так і всі клієнти повинні мати можливість визначати власні політики доступу до ресурсів системи. Причому, вищевказані політики доступу повинні бути безпечно об’єднані та правильно застосовані в середовищі спільного доступу до ресурсів системи для кількох клієнтів. Результатом даної роботи є архітектура системи контролю доступу, що дозволяє як постачальнику, так і всім його клієнтам ефективно визначати свої політики доступу до хмарної SaaS системи. Архітектура надійно поєднує політики доступу всіх сторін, зв’язує та забезпечує їх виконання під час роботи з хмарною системою.

РОЗДІЛ 1
ЗАДАЧА СИНТЕЗУ АРХІТЕКТУРИ КОНТРОЛЮ ДОСТУПУ

1.1. Опис особливостей багатоклієнтських хмарних систем

Хмарні обчислення (Cloud computing) - модель для забезпечення повсюдних, зручних, доступних на вимогу, обчислювальних ресурсів (наприклад, мережі, сервери, сховища даних, програми та сервіси), які можна швидко надати для користування.

Дана модель має п'ять основних характеристик:

1. Легкодоступна на вимогу користувача (on-demand self-service).
2. Надає широкий доступ по мережі (broad network access).
3. Надає об'єднання ресурсів (Offers resource pooling).
4. Швидка масштабованість (rapid elasticity).
5. Надає послуги, які можна виміряти/тарифікувати (offers measured service).

Хмарні обчислення представлені трьома моделями послуг:

1. Програмне забезпечення як послуга (Software-as-a-Service, SaaS).
2. Платформа як послуга (Platform-as-a-Service, PaaS).
3. Інфраструктура як послуга (Infrastructure-as-a-Service, IaaS).

Та чотирма моделями розгортання:

1. Приватна хмара (Private Cloud).
2. Хмара спільноти (Community Cloud).
3. Публічна хмара (Public Cloud).
4. Гібридна хмара (Hybrid Cloud).

Програмне забезпечення як послуга (SaaS) - це модель розповсюдження програмного забезпечення, в якій програми розміщуються постачальником послуг у хмарному середовищі і доступні клієнтам через мережу, як правило, Інтернет. SaaS стає все більш поширеною моделлю доставки як базові технології. SaaS також часто асоціюється з моделлю ліцензування підписки з оплатою по мірі використання.

Застосунки SaaS також повинні мати можливість однаково взаємодіяти з іншими даними та іншими застосунками в широкому спектрі середовищ і платформ. SaaS тісно пов'язаний з іншими моделями хмарних обчислень. SaaS найчастіше впроваджується для забезпечення бізнес-застосунків та функціональності для корпоративних клієнтів за низькою ціною, дозволяючи цим клієнтам отримати ті ж самі переваги як і від комерційно ліцензованого, внутрішньо керованого програмного забезпечення, але без супутньої складності встановлення, управління, підтримки, ліцензування та високої початкової вартості [4].

Основні характеристики моделі SaaS:

1. *Доступ через Інтернет*: кінцеві користувачі отримують доступ до програми через Інтернет за допомогою стандарту http/https. Замість традиційного клієнтського доступу на ПК використовується веб-підхід доступу до ресурсів через корпоративну глобальну мережу.

2. *Підтримка постачальників*: застосунок не керується IT-відділом клієнта, а розміщується та керується командою розробників постачальника.

3. *Модель підписки*: замість того, щоб сплачувати авансові безстрокові ліцензії, клієнт сплачує регулярну (терміни визначаються контрактом між клієнтом та постачальником) плату за користування функціоналом системи.

4. *Мінімальна необхідність в попередніх налаштуваннях*: виконується обмежена кількість налаштувань на попередньому етапі перед безпосереднім початком використання. Застосунки є високо стандартизованими для клієнтів, часто розміщені в рамках багатоклієнтської архітектури.

5. *Керовані оновлення*: покращення функціональності повністю контролюються постачальником. Поширеною є практика частих циклів оновлення з невеликими обсягами покращень або змін функціональності.

6. *Необхідність у переході до ментальності на основі послуг*: модель SaaS вимагає зміни парадигми від підходу, орієнтованого на продукт, до моделі сервісного обслуговування. Постачальник несе відповідальність не тільки за розробку системи, а й за весь набір служб та підтримку програмного забезпечення. Окрім функціонуючого програмного коду, постачальник повинен надати високий

рівень досвіду користувача, що включає впровадження, тестування, навчання, усунення несправностей, технічне обслуговування, хостинг, оновлення та безпека.

7. *Дохід, що базується на успішності*: комерційний успіх постачальника критично пов'язаний з успішністю клієнтів. Клієнти не роблять авансових інвестицій у програмне забезпечення, апаратне забезпечення або ресурси для реалізації. Постачальнику платять лише в тому випадку, якщо клієнт задоволений хмарною системою і продовжує свою підписку. На відміну від традиційних програмних моделей, незадоволені користувачі SaaS можуть легко скасувати підписку та перейти до постачальника-конкурента.

Хмарна система, розроблена відповідно до SaaS, відповідає високоякісним вимогам до ресурсів і надає користувачам послуги з використанням багаторівневих абстракцій.

Клієнт (tenant) – це група користувачів, які мають однакове представлення у системі. Сюди входять дані, до яких вони мають доступ, конфігурацію, керування користувачами, конкретні функції та пов'язані нефункціональні властивості. Зазвичай дані групи є членами різних юридичних осіб. Це супроводжується відповідними обмеженнями (безпека даних, конфіденційність, тощо).

Багатоклієнтність (multi-tenancy) — це підхід до спільного використання екземпляра програми між кількома клієнтами шляхом надання кожному спеціальної «частки» екземпляра, який ізольований від інших в аспектах продуктивності та конфіденційності даних. Зазвичай аналогією для пояснення є житловий комплекс, де різні сторони використовують деякі свої ресурси, як-от централізоване опалення, щоб зменшити витрати, але також вимагають певного рівня ізоляції (наприклад, рівень шуму) [3].

Крім терміну багатоклієнтності, існує також поняття *клієнтського простору (tenant space)*. Клієнтський простір визначає підхід, при якому клієнти орендують заздалегідь визначений простір ресурсів, в якому вони можуть запускати різні екземпляри програми. Одним із прикладів цього є модель *Infrastructure-as-a-Service (IaaS)*, коли клієнт купує апаратні ресурси, на які він встановлює програмне забезпечення на свій розсуд.

Сценарії, коли кілька програм працюють в одному екземплярі одного і того ж середовища виконання (модель *Platform-as-a-Service, PaaS*), іноді також називаються багатоклієнтськими, зокрема з точки зору постачальників, оскільки їхні клієнти розгортають кілька програм. Цю концепцію коректно називати *багатопрограмним розгортанням(multi application deployment)*, а не багатоклієнтським середовищем, оскільки сутності, що відокремлюються, мають інші характеристики, а проблеми в цьому сценарії мають інший характер. Таким чином, багатопрограмне розгортання не слід ототожнювати до багатоклієнтської системи.

Спільне використання центрів обробки даних, віртуалізації та проміжного програмного забезпечення іноді розглядаються як один підхід до досягнення багатоклієнтського середовища. Всі це підходи базуються на одному екземплярі програми для кожного. Проте їм не вистачає розподілу ресурсів та ефективності.

1.2. Проблематика забезпечення контролю доступу в хмарних системах

Парадигма хмарних обчислень пов’язана з проблемами безпеки як з боку постачальників, так і з боку кінцевих споживачів. Постачальники хочуть переконатися, що їхні ресурси та послуги використовуються лише авторизованими користувачами, в той час як споживачі хотіли б переконатися, що їхні дані безпечно зберігаються в хмарі та убезпечені від несанкціонованого читання/запису [1].

Контроль доступу є фундаментальним аспектом інформаційної безпеки, який безпосередньо пов'язаний з такими основними характеристиками, як конфіденційність, цілісність і доступність. З точки зору доступу провайдери повинні забезпечити наступні основні функції:

- контроль доступу до сервісних функцій хмари на основі визначених політик і рівня послуги, придбаної клієнтом;
- контроль доступу до даних споживача та захист від доступу з боку інших клієнтів в багатоклієнтському середовищі;
- контроль доступу до спільних для всіх клієнтів функцій;

- підтримка актуального стану політик доступу та інформації користувачів.

Відносини між користувачами та ресурсами динамічні в хмарі, а постачальники послуг і користувачі, як правило, не знаходяться в одному домені безпеки. Модель безпеки на основі ідентифікації (наприклад, дискреційна або моделі обов’язкового контролю доступу) не можна використовувати у відкритому хмарному середовищі, де кожен ресурсний вузол може не бути знайомим або навіть не знати про інші вузли. Наприклад, коли користувачі хмари можуть отримувати доступ до послуг через Інтернет за допомогою різних засобів, таких як мобільний телефон, ноутбук або персональний комп’ютер, і неможливо ідентифікувати користувачів за фіксованими IP-адресами. У таких ситуаціях не можна використовувати традиційні брандмауери для фільтрації пакетів на основі фіксованих IP-адрес користувачів. У хмарі користувачі зазвичай ідентифіковані за своїми атрибутами чи характеристиками, а не за попередньо визначеними ідентичностями. Таким чином, потрібна динамічна система контролю доступу для досягнення міждоменної аутентифікації.

Звідси випливає, що контроль доступу є важливою частиною хмарної системи, що обмежує дії (наприклад, читання, запис), які суб’єкт може виконати над об’єктом в системі. Правила контролю доступу зазвичай відділяються від самої системи, яку вони обмежують і виражаються в модульних, декларативних політиках контролю доступу з міркувань відокремлення проблем і можливості модифікації. Контроль доступу на основі політик добре підходить для хмарних SaaS систем, тому що це дозволяє відділити логіку безпеки, специфічну для клієнта, та надає можливість її використання в режимі виконання (run-time).

Для декларативного вираження було запропоновано кілька моделей представлення контролю доступу:

1. Обов'язковий контроль доступу (MAC).
2. Дискреційний контроль доступу (DAC).
3. Контроль доступу на основі ролей (RBAC).
4. Контроль доступу на основі атрибутів (ABAC).

1.2.1. Обов’язковий контроль доступу

Обов’язковий контроль доступу (MAC) – це засіб обмеження доступу до об’єктів на основі чутливості інформації, що міститься в об’єктах, та офіційної авторизації суб'єктів на доступ до інформації. Відомою реалізацією MAC є багаторівневий захист (MLS), який, традиційно, доступний в основному на комп’ютерних і програмних системах, розгорнутих у дуже чутливих державних організаціях. Базова модель MLS була вперше введена Беллом і ЛаПадулою [6]. Модель викладається в термінах об'єктів і суб'єктів. Об'єктом є пасивна сутність, така як файл даних, запис або поле в межах запису. Суб’єкт – це активний процес, який може запитувати доступ до об'єктів. Кожному об’єкту присвоєна класифікація, а коженому предмету - дозвіл. Класифікації та дозволи разом називаються мітками (label).

Мітка – це частина інформації, яка складається з двох частин: ієрархічний компонент і набір невпорядкованих відсіків. Ієрархічний компонент визначає чутливість даних. Наприклад, військова організація може визначати рівні *абсолютно таємно, секретно, конфіденційно та несекретно*. Мітки частково впорядковані в матрицю наступним чином: Дано дві мітки L1 і L2, L1 >= L2 тоді і тільки тоді, коли ієрархічний компонент L1 більше або дорівнює L2, кажуть, що L1 домінує над L2 [7].

MLS накладає наступні два обмеження на доступ до даних:

- Властивість Simple Security або «Без читання»: суб'єкту дозволено доступ для читання до об'єкта тоді і тільки тоді, коли мітка суб’єкта домінує над міткою об'єкта.
- Властивість * (вимовляється як властивість зірки) або «Без запису»: суб’єкту дозволено доступ на запис до об’єкта тоді й лише тоді, коли мітка об’єкта домінує над міткою суб’єкта.

1.2.2. Дискреційний контроль доступу

Дискреційний контроль доступу (DAC) — модель контролю доступу, визначена як засіб обмеження доступу до об’єктів на основі ідентичності суб’єктів та/або груп, до яких вони належать. Дискреційний у тому сенсі, що суб’єкт з певним дозволом доступу може передати цей дозвіл (можливо, опосередковано) будь-якому іншому суб’єкту (якщо це не обмежено обов’язковим контролем доступу). [8]

Дискреційний контроль доступу зазвичай протиставляється обов'язковому контролю доступу (MAC). Іноді кажуть, що система в цілому має «дискреційний» або «чисто дискреційний» контроль доступу, коли в цій системі відсутній обов’язковий контроль доступу. З іншого боку, системи можуть одночасно реалізовувати як MAC, так і DAC, де DAC відноситься до однієї категорії засобів контролю доступу, які суб'єкти можуть передавати один одному, а MAC відноситься до другої категорії контролю доступу, яка накладає обмеження на першу.

1.2.3. Контроль доступу на основі ролей

Контроль доступу на основі ролей (RBAC) - модель обмеження доступу до системи авторизованим користувачам, нейтральний для політики механізм контролю доступу, визначений навколо ролей і привілеїв. Компоненти RBAC, такі як ролі-дозволи, відносини користувач-роль та роль-роль, полегшують виконання призначень користувачам [9].

Всередині організації ролі створюються для виконання різних посадових функцій. Дозволи на виконання певних операцій призначаються певним ролям. Членам або персоналу (або іншим користувачам системи) призначаються певні ролі, і через ці призначенні ролі вони отримують дозволи, необхідні для виконання певних системних функцій. Оскільки користувачам не призначаються дозволи безпосередньо, і вони лише отримують їх через свою роль (або ролі), керування окремими правами користувача стає питанням простого призначення відповідних ролей обліковому запису користувача; це спрощує звичайні операції, такі як додавання користувача або зміна відділу користувача.

Для RBAC визначено три основні правила:

1. *Призначення ролі*: суб’єкт може скористатися дозволом, лише якщо суб’єкт вибрав або йому було призначено роль.

2. *Авторизація ролі*: активна роль суб’єкта має бути авторизована для суб’єкта. З вищенаведеним правилом 1 це правило гарантує, що користувачі можуть виконувати лише ті ролі, на які вони уповноважені.

3. *Авторизація дозволу*: суб’єкт може використовувати дозвіл, лише якщо дозвіл надано для активної ролі суб’єкта. З правилами 1 і 2 це правило гарантує, що користувачі можуть використовувати лише ті дозволи, на які вони мають дозвіл.

При визначенні моделі RBAC зручними є наступні позначення:

- S = Суб'єкт - Особа або автоматизований агент
- R = Роль - Посадова функція або назва, яка визначає рівень повноважень
- P = Дозволи - Затвердження режиму доступу до ресурсу
- SE = сеанс - відображення, що включає S, R та/або P
- SA = Предметне завдання
- PA = Призначення дозволу
- RH = частково впорядкована ієрархія ролей. RH також можна записати: ≥ (Позначення: x ≥ y означає, що x успадковує права y.)

Виконуються наступні умови:

- Суб'єкт може виконувати кілька ролей.
- Роль може мати кілька предметів.
- Роль може мати багато дозволів.
- Дозвіл можна призначити багатьом ролям.
- Операції можна призначити багатьом дозволам.
- Дозвіл можна призначити багатьом операціям.

Обмеження накладає обмежувальне правило на потенційне успадкування дозволів від протилежних ролей, тому його можна використовувати для досягнення належного розподілу обов’язків. Наприклад, одній і тій же особі не можна

дозволяти створювати обліковий запис для входу та авторизувати створення облікового запису.

Таким чином, використовуючи позначення теорії множин:

$PA \subseteq P \times R$ і є дозволом багато-до-багатьох для відношення призначення ролі.

$SA \subseteq S \times R$ і залежить від відношення призначення ролі багато до багатьох.

$RH \subseteq R \times R$

Суб’єкт може мати кілька одночасних сесій з/в різних ролях.

1.2.4. Контроль доступу на основі атрибутів

Контроль доступу на основі атрибутів (ABAC) - модель контролю доступу, згідно з якою права доступу надаються користувачам за допомогою політик, які поєднують атрибути разом. Політики можуть використовувати будь-які типи атрибутів (атрибути користувача, атрибути ресурсу, об’єкти, атрибути середовища тощо) [10].

Дана модель узагальнює попередні моделі та виражає контроль доступу в термінах властивостей ключ-значення, які називаються атрибутами суб'єкта (наприклад, ідентифікатор суб'єкта, ім'я користувача або ролі), об'єкта (наприклад, ідентифікатор об'єкта, розташування або вміст) і середовища (наприклад, час, фізичне розташування або контекст). Атрибути забезпечують підвищену декларативність і пропонують одиницю передачі даних між різними компонентами або сторонами в рамках контролю доступу [2].

Атрибути - характеристики, які визначають конкретні аспекти суб’єкта, об’єкта, середовища, умови та/або запитувані дії, які попередньо визначені та призначені. Атрибути складаються з необов'язкової категорії, яка вказує на клас інформації, що надається атрибутом, ім'я та значення (наприклад, Class=HospitalRecordsAccess, Name=PatientInformationAccess, Value=MFBusinessHoursOnly) [11].

Суб'єкт - активна сутність (зазвичай фізична особа, процес або пристрій), яка створює інформацію для переходу між об'єктами або зміни стану системи. Це може

бути користувач, запитувач або механізм, що діє від імені користувача або запитувача. Суб'єктом може бути нефізична сутність, наприклад система або процес. Суб’єкти часто діють від імені певної людини чи організації. Суб'єктам можуть бути присвоєні атрибути, які описують їх ім'я, приналежність до організації, громадянство, тощо.

Об'єкт - пасивні дані, пов'язаним з інформаційною системою (наприклад, пристрої, файли, записи, таблиці, процеси, програми, мережі, домени), що містять або отримують інформацію. Доступ до об'єкта передбачає доступ до інформації, яку він містить. Це може бути ресурс або запитувана сутність, а також будь-що, над чим суб'єкт може виконувати операцію, включаючи дані, програми, послуги, пристрої і мережі. Об’єкти зазвичай вимагають певної форми захисту від неприпустимих операцій несанкціонованими суб'єктами.

Операція - виконання функції на вимогу суб'єкта над об'єктом. Операції включають читання, запис, редагування, видалення, автор, копіювання, виконання та модифікація.

Політика - представлення правил або відносин, які визначають набір допустимих операцій, які суб'єкт може виконувати над об'єктом в дозволених умовах середовища.

Модель ABAC підтримує булеву логіку, в якій політики містять оператори «IF, THEN» про те, хто робить запит, ресурс і дію. Наприклад, ЯКЩО запитувач є менеджером, ТО надайте доступ на читання/запис до конфіденційних даних.

На відміну від контролю доступу на основі ролей (RBAC), який використовує заздалегідь визначені ролі, які несуть певний набір пов'язаних з ними привілеїв і яким призначаються суб'єкти, ключовою відмінністю ABAC є концепція політик, що виражають складний набір булевих правил, які можуть оцінити багато різних атрибутів. Значення атрибутів можуть бути наборами або атомарними. Атрибути із наборами значень містять більше одного атомарного значення.

Основні концепції ABAC:

- Точка адміністрування політики – Policy Administration Point (PAP).
- Точка виконання політики – Policy Enforcement Point (PEP).

- Точка прийняття рішень – Policy Decision Point (PDP).
- Точка інформації про політику – Policy Information Point (PIP).

PEP відповідає за захист програм і даних, до яких ви хочете застосувати ABAC. PEP перевіряє запит і створює на його основі запит на авторизацію, який він надсилає до PDP.

PDP – ключова частина архітектури ABAC. Це частина, яка оцінює вхідні запити за політиками, з якими він був налаштований. PDP повертає рішення про дозвіл/відмову (permit/deny). PDP також може використовувати PIP для отримання відсутніх метаданих.

PIP з'єднує PDP із зовнішніми джерелами атрибутів, наприклад, LDAP або бази даних.

1.2.5. Огляд обмежень існуючих моделей

Використання хмарних систем SaaS в бізнесі вимагає від кінцевих користувачів швидко робити безліч дискретних рішень стосовно контролю доступу (наприклад, з ким вони бажають ділитися документами, кого запросити на онлайн конференцію, тощо). Широкий спектр функціональності, який зазвичай надає хмарна система, спонукає користувачів до помилок, що провокує витоки інформації через прийняття кінцевими користувачами неправильних рішень щодо обміну контентом. Традиційні моделі контролю доступу, такі як RBAC, малопридатні для використання людьми без досвіду інформаційної безпеки. Крім того, структурована модель контролю доступу, така як RBAC, не може бути пов'язана з низькими накладними витратами в ході ділового співробітництва. В результаті, в такому контексті кінцеві користувачі самі несуть відповідальність за визначення політики безпеки для об'єктів, які вони створили та якими мають намір поділитися з (потенційно) обмеженою групою колег та співробітників. Зазвичай це спонукає користувачів до вибору за замовчуванням, що в цілому призводить до зниження безпеки в рамках всієї системи [5].

1.3. Огляд методик та засобів синтезу архітектури інформаційних систем

1.3.1. Метод синтезу архітектури TOGAF

TOGAF – галузевий стандарт, метод розробки архітектури та ресурсної бази, який може вільно використовуватися будь-якою організацією, яка бажає розробити архітектуру програмного забезпечення для використання всередині даної організацію [12].

TOGAF розробляється і постійно розвивається представниками з середини 90-х років деяких із провідних світових організацій замовників і постачальників IT, які працюють у The Open Group's Architecture Forum.

Доступні дві версії TOGAF:

- TOGAF Версія 8 ("Enterprise Edition"), вперше опублікована в грудні 2002 року і переопублікована в оновленому вигляді як TOGAF Версія 8.1 у грудні 2003 року
- TOGAF Версія 7 («Технічне видання»), опублікована в грудні 2001 року

TOGAF Версія 8 використовує той самий базовий метод розробки архітектури, особливим акцентом на технічні архітектури, починаючи від TOGAF Версії 7. TOGAF Версія 8 застосовує цей метод розробки архітектури до всіх доменів організації, включаючи архітектуру бізнесу, даних та програмного забезпечення, а також технічну архітектуру.

Рис. 1. Складові методу TOGAF

Метод розробки архітектури TOGAF (ADM), зображений на рисунку пояснює,

як розробити специфічну для організації архітектуру, що відповідає вимогам бізнесу.

ADM забезпечує:

1. Надійний, перевірений спосіб розвитку архітектури.
2. Архітектурні види, які дозволяють архітектору задовольнити складний набір вимог, кожна з яких враховується належним чином
3. Посилання на практичні приклади
4. Рекомендації щодо інструментів для розробки архітектури

TOGAF Enterprise Continuum, зображений на рисунку, є «віртуальним сховищем» усіх активів архітектури - моделей, шаблонів, описів архітектури тощо – які існують і в організаціях та в IT-індустрії в цілому. У відповідних місцях по всьому TOGAF ADM, є нагадування про те, які наявні архітектурні активи від TOGAF Enterprise Continuum, які архітектор повинен використовувати.

Ресурсна база TOGAF, що являє собою набір ресурсів – рекомендацій, шаблонів, довідкової інформації, тощо - для полегшення використання ADM архітектором.

Ресурсна база TOGAF містить інформацію про наступне:

1. Архітектурні дошки.
2. Відповідність архітектурі.
3. Архітектурні договори.
4. Управління архітектурою.
5. Моделі зрілості архітектури.
6. Архітектурні шаблони.
7. Принципи архітектури.
8. Набір кваліфікацій архітектури.
9. Представлення архітектури.
10. Приклади конструкційних блоків.
11. Бізнес-сценарії.
12. Тематичні дослідження.
13. Словник термінології.

14. Інші архітектури / фреймворки та їх зв'язок з TOGAF.
15. Інструменти для розробки архітектури.

1.3.2. Документування та опис архітектури

Архітектурна документація повинна служити різним цілям. Вона має бути достатньо абстрактною, щоб бути зрозумілою для нових співробітників. Вона повинна бути достатньо структурованою, щоб виступати в якості специфікації для розробки. Для цього вона має містити достатньо інформації, що буде служити основою для аналізу. Архітектурна документація є як нормативною, так і наказовою описовою. Найкраща архітектурна документація, для аналізу продуктивності може відрізнятися від найкращої архітектурної документації для розробників системи [13].

Мета документування архітектури – записати її в представленнях, які нададуть змогу іншим успішно використовувати, підтримувати та розширювати систему, спираючись на даний документ.

Стандартом є наступні рекомендації документування архітектури:

1. Документувати потрібно з точки зору читача.
2. Уникати непотрібних повторень.
3. Уникати багатозначностей. Завжди пояснювати умовні позначення.
4. Використовувати стандартизовану структуру документу.
5. Обґрунтувавати записи.
6. Підтримувати документацію актуальною.
7. Перевіряти документацію на відповідність призначенню.

Для документування синтезованої в роботі архітектури використовувалася структура документації, розроблена та рекомендована Інститутом Програмного Забезпечення, що задовольняє вищевказаний критеріям.

РОЗДІЛ 2 СИНТЕЗ АРХІТЕКТУРИ КОНТРОЛЮ ДОСТУПУ ДЛЯ БАГАТОКЛІЄНТСЬКИХ ХМАРНИХ СИСТЕМ

2.1. Опис прикладу бізнес сценарію, який вирішує архітектура

Дана кваліфікаційна робота ставить за мету синтез архітектури, що може використовуватись для розробки рішень проблеми контролю доступу для багатоклієнтських систем, розгорнутих у хмарному середовищі.

Розглянемо наступний приклад такої системи, як цільової для даного рішення:

- Організація А є власником та розробником багатоклієнтської хмарної системи, основними функціями якої є забезпечення великої кількості клієнтів (навчальних закладів) автоматизованими засобами організації навчального процесу з використанням галузевих стандартів. Типовими клієнтами даної організації є коледжі та професійні навчальні заклади. Представники клієнта (зазвичай, викладачі або куратори) можуть створювати, завантажувати, редагувати, групувати навчальні матеріали. В той час користувачі системи (студенти) можуть переглядати та використовувати вищевказані матеріали.
- Контроль доступу є ключовим програмним механізмом даної системи. Він повинен задовольняти наступні вимоги:
    - Постачальник системи (Організація А) повинен мати механізм обмеження та контролю над клієнтами (наприклад, обмеження доступного функціоналу в залежності від типу обраної клієнтом тарифікації послуг)
    - Клієнти повинні бути повністю ізольованими, знаходячись при цьому в одному й тому ж клієнтському просторі і використовуючи одні й ті ж ресурси.
    - Кожен клієнт може мати власні, внутрішні, протоколи та правила безпеки та контролю доступу (наприклад, коледж Х дозволяє кураторам оцінювання успішності користувачів, в той час як коледж У має складніший процес перевірки успішності, що включає перевірку як куратора, так і

викладача курсу). Отже, система повинна надавати можливість кожному клієнту налаштовувати політики доступу індивідуально та застосовувати їх лише для користувачів даного клієнта.

- В той час, як ізоляція клієнтів є механізмом, що підтримується за замовчуванням, клієнти повинні мати можливість додавати виключення до даних політик (наприклад, коледж X міг укласти угоду про співпрацю з коледжем У і потребувати механізму надання часткового доступу до своїх даних користувачам коледжу У, і навпаки).

- В той час, як вищеописані вимоги самі по собі не є тривіальними, виникає також додаткова складність у підтримці різних варіантів та комбінацій описаних сценаріїв.

2.2. Вимоги до архітектури

Було сформовано наступні функціональні вимоги до архітектури:

1. Архітектура повинна дозволяти постачальнику обмежувати клієнтів.
2. Архітектура повинна дозволяти кожному окремому клієнту обмежувати своїх користувачів.
3. Архітектура повинна надійно поєднувати визначені політики контролю доступу всіх залучених організацій.
4. Архітектура повинна забезпечувати виконання політик контролю доступу для кожного окремого запиту до системи.

Також було сформовано наступні нефункціональні вимоги:

1. Архітектура повинна підтримувати розміщення в хмарному середовищі.
2. Архітектура повинна бути масштабованою та передбачати можливість, коли багатоклієнтська хмарна система може бути розміщена на різних обчислювальних кластерах.
3. Архітектура повинна забезпечувати надійне виконання всіх налаштованих політик.
4. Архітектура не повинна допускати несанкціонованого доступу до інформації.

2.3. Опис розробленої моделі контролю доступу

В даному розділі кваліфікаційної роботи поданий опис підходів та принципів, що забезпечують виконання вищевказаних вимог розробленою архітектурою. Одним з основних результатів роботи є представлена модель контролю доступу, що базується на описаних в теоретичній частині моделях, комбінуючи сильні сторони кожної.

2.3.1. Використані теоретичні моделі та концепції

Для побудови моделі були використані наступні моделі та підходи:

- Контроль доступу на основі ролей (RBAC)
    - Архітектура використовує дану модель для забезпечення можливості постачальнику системи та клієнтам визначати власні механізми контролю доступу декларативно і без необхідності в зміні програмного коду системи.
- Контроль доступу на основі атрибутів (ABAC)
    - Архітектура використовує дану модель для забезпечення простої та зрозумілої абстракції, що дозволяє керувати доступами користувачів на основі призначених їм атрибутів/властивостей. Також, дана модель надає механізм визначення правил доступу, що дозволяє виражати більшість правил та концепцій, наявних в вищевказаному бізнес-сценарії, таких як дозволи, авторство над об’єктом, ролі, розділення обов’язків, мітки дати-часу, зони відповідальності. З іншого боку, дана модель сама по собі не надає достатньо механізмів для задоволення всіх згаданих вимог, в першу чергу по причині того, що кожен атрибут для кожного об’єкта чи суб’єкта повинен визначатись вручну.
- Структуровані дерева політик (Structured policies trees)
    - Архітектура використовує дану концепцію для поєднання політик доступу клієнтів та постачальника, одночасно забезпечуючи дотримання ключових принципів безпеки.

Результуюча тришарова модель, базуючись на даних моделях та концепціях, поєднує їх в одну архітектуру та забезпечує підтримку сценаріїв та дотримання вимог, специфічних для багатоклієнтських систем хмарних систем.

2.3.2. Опис механізму роботи трирівневої моделі контролю доступу

Механізм забезпечення контролю доступу складається з управління атрибутами та політиками, які використовують ці атрибути. Архітектура розділяє процес управління між трьома залученими сторонами:

- постачальник системи;
- клієнти;
- система контролю доступу.

Перш за все, система контролю доступу заздалегідь визначає загальні атрибути та політики, які можна повторно використовувати між різними багатоклієнтськими системами. Постачальник багатоклієнтської системи, розуміючи специфіку доменної області, визначає атрибути та політики, специфічні для його системи. Постачальник також визначає атрибути та політики, яку використовуються для контролю доступу його клієнтів. Врешті решт, конкретний клієнт визначає атрибути та політики доступу для контролю над його користувачами.

Кожен наступний шар політик доступу базується на попередніх, таким чином спрощуючи загальний процес контролю доступу та полегшуючи можливість повторного використання визначених атрибутів та політик.

2.3.2.1. Управління атрибутами

Управління атрибутами передбачає два види дій:

1. Визначення можливих атрибутів для суб’єктів та ресурсів.
2. Присвоєння відповідних значень атрибутам.

Система контролю доступу попередньо визначає фіксований набір атрибутів, які постачальник може використовувати для своєї багатоклієнтської системи. Це

включає ідентифікатори суб'єкта чи ресурсу та пов'язаного з ним клієнта. Постачальник визначає атрибути ресурсів у його системі. Клієнти визначають власні специфічні атрибути.

Після визначення відповідних атрибутів кожна сторона відповідає за їх призначення щодо ресурсів і суб'єктів, які вона контролює. Система контролю доступу автоматично призначає атрибути у випадках, де це можливо, наприклад, ідентифікатор кожного нового об'єкта, який створюється у системі. Приклади трирівневого управління атрибутами представлені на рисунку.

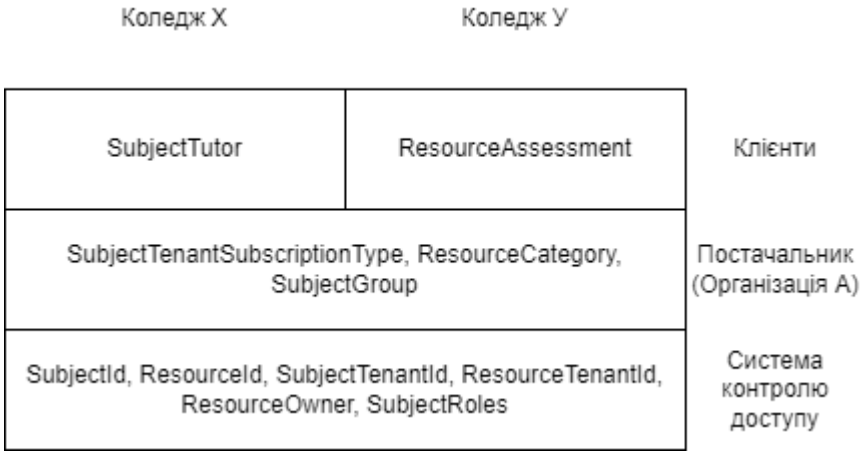

Рис. 2,1 Приклад визначених атрибутів на різних рівнях архітектури

2.3.2.2. Управління політиками доступу

Вищеописана трирівнева модель також застосовується і для управління політиками доступу. Політики на кожному рівні моделі визначаються з використанням атрибутів, доступних на тому рівні.

Система контролю доступу має визначені базові політики доступу за замовчуванням, основною з яких є політика ізоляція доступу між клієнтами. Політики даного рівня можуть використовувати лише атрибути, визначені системою контролю доступу.

Постачальник може визначати власні політики доступу, що будуть забезпечуватись для його клієнтів. Політики даного рівня можуть використовувати як атрибути попереднього рівня, так і атрибути, визначені постачальником. Наприклад, постачальник може визначити політику, яка забороняє певному клієнту

завантажувати більшу кількість навчальних матеріалів, ніж визначено в його тарифному плані.

Клієнт може визначати власні політики доступу, область дії яких буде обмежена лише колом його користувачів. Політики даного рівня можуть використовувати атрибути, визначені будь якою зі сторін. Наприклад, клієнт може визначити політику доступу, згідно з якою, куратор навчального курсу не може переглядати результати роботи користувача, який не знаходиться в певній групі.

На додачу до вказаного, архітектура надає можливість постачальнику та клієнтам визначати виключення до політики ізоляції клієнтів за замовчуванням. Політики об'єднуються у булеві вирази за допомогою оператора "І (AND)", виключення до політик - оператор "АБО (OR)".

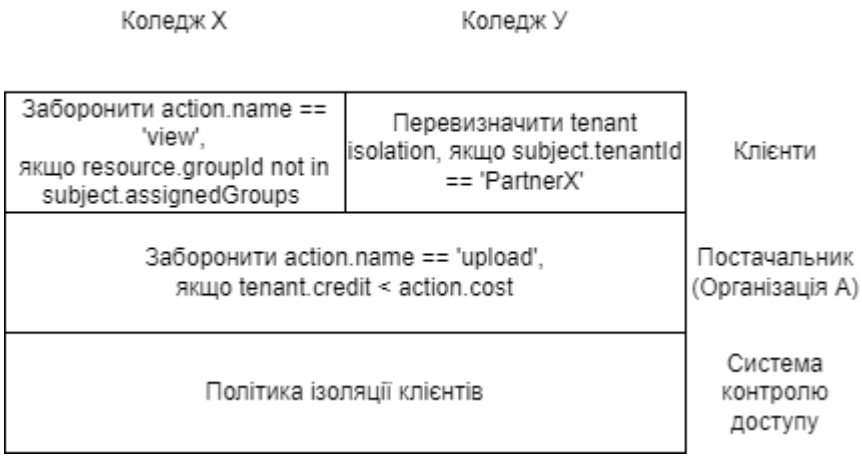

Рис. 2,2 Приклад визначення політик на різних рівнях моделі

2.3.2.3. Поєднання політик різних рівнів моделі

Система контролю доступу відповідає за поєднання політик контролю доступу всіх сторін так, щоб вони застосовувалися належним чином. Тобто, навіть з урахуванням політик усіх сторін, система все одно має гарантувати базові властивості безпеки. Головною такою властивістю є неможливість для клієнтів використовувати власні політики для перевизначення політик ізоляціїм іншого клієнта або постачальника.

Для досягнення цієї мети, система поєднує правила всіх сторін з використанням структури дерева. Листові вузли цього дерева являють собою

правила, які при обчисленні повертають результат - Дозвіл/Заборона (Permit/Deny) за певних умов. Проміжні вузли аналізують результати обчислення дочірніх вузлів та вказують, до яких запитів застосовувати результуючий вираз.

У разі додавання/модифікації політики однією зі сторін, дерево політики формується згідно алгоритму:

1. Побудова під-дерева ізоляції клієнтів:
    a. Побудова базової політики з вхідним параметром “будь-що” (any) та комбінаційним методом “Дозвіл на перевизначення” (Permit overrides).
    b. Додавання правила ізоляції клієнтів за замовчуванням
    c. Додавання виключень ізоляції, визначених постачальником
    d. Додавання виключень ізоляції, визначених кожним з клієнтів, у складі політик вигляду “якщо resource.tenantId == $tenantId”
2. Побудова фінального дерева політик:
    a. Побудова базової політики з вхідним параметром “будь-що” (any) та комбінаційним методом “Заборона на перевизначення” (Deny overrides)
    b. Додавання під-дерева, побудованого на попередніх кроках.
    c. Додавання політик стосовно клієнтів, визначених постачальником.
    d. Додавання політик кожного клієнта,у складі політики з вхідним параметром “subject.tenantId == $tenantId” та комбінаційним методом, визначеним клієнтом

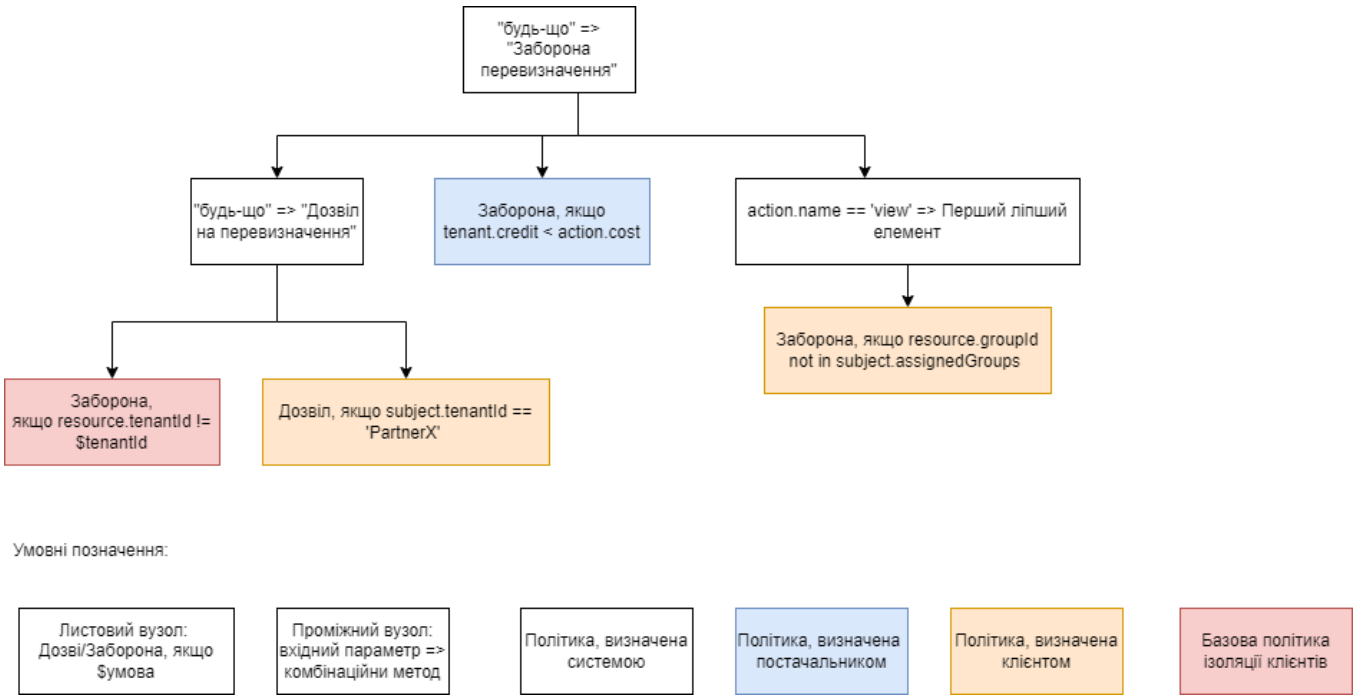

Рис. 2,3 Приклад результуючого дерева політик

Дерево політик гарантує коректність та забезпечення всіх політик. Перш за все, під-дерево політик ізоляції клієнта поєднується з політиками постачальника та політиками клієнтів за допомогою методу “Заборона на перевизначення”, тобто конкретний запит буде дозволено тільки у випадку, якщо політики постачальника, клієнта та політика ізоляції за замовчуванням дозволяють його.

Разом з тим, політика ізоляції за замовчуванням та виключення з неї, визначені постачальником та клієнтом, поєднуються методом “Дозвіл на перевизначення”, тобто конкретний запит буде дозволено у випадку, коли він задовольняє хоча б одному виключенню.

Крім того, додатково додаються політики кожного з клієнтів, які будуть забезпечуватись лише для користувачів конкретного клієнта.

2.3.2.4. Реалізація архітектури

З точки зору архітектури, багатоклієнтська хмарна система складається з:

- каталогу наявних клієнтів;
- сховища каталогу клієнтів;
- вузлів з бізнес-логікою;

- набору баз даних кожного клієнта.

Система контролю доступу, що повинна бути реалізована у вигляді проміжного програмного забезпечення (middleware), складається з:

- Модуля точки прийняття рішень.
- Модуля автентифікації користувача.
- Панелі керування постачальника.
- Панелі керування клієнта.
- Сховища атрибутів.

Модуль точки прийняття рішень обчислює результат перевірок політик для запиту. Панелі постачальника/користувача надають змогу визначати на призначати атрибути, конфігурувати політики доступу, при зміні яких перебудовується дерево політик.

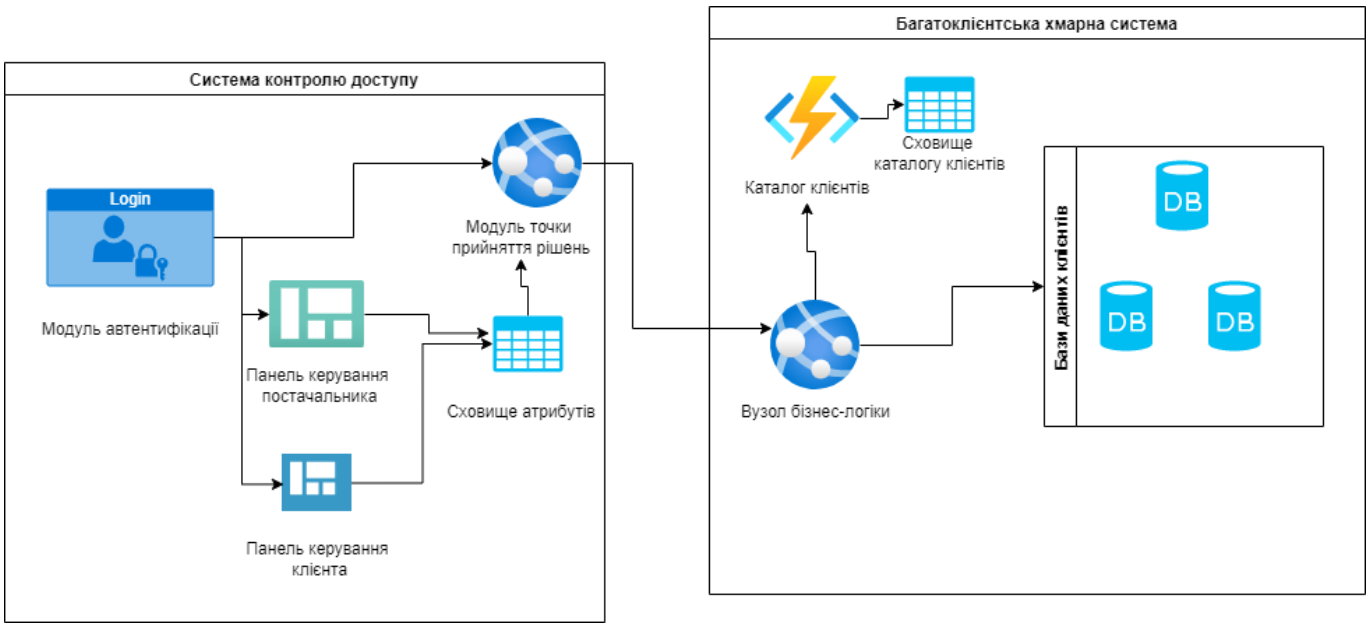

Рис. 2,4 Схема компонентів синтезованої архітектури

ВИСНОВОК

Аналіз існуючих моделей та підходів організації контролю доступу в багатоклієнтських системах показав, що на сьогоднішній день не існує досконалих рішень, здатних задовольнити всі вимоги до системи контролю доступу з урахуванням специфіки таких систем та необхідності в наданні можливості глибокого налаштування правил контролю доступу між різними сторонами, що користуються системою.

З метою синтезу архітектури контролю доступу для багатоклієнтської хмарної системи було розглянуто існуючі моделі, на основі яких розроблено результуючу модель. Синтезована архітектура враховує специфіку багатоклієнтських хмарних систем та покликана спростити процес побудови складних процесів контролю доступу та поєднання політик різних сторін. Прототип запропонованої в результатах кваліфікаційної роботи архітектури був реалізований в рамках проходження виробничої практики на підприємстві ТОВ “Вейвексес” та інтегрований з існуючою багатоклієнтською хмарною системою.

За результатами роботи сформовано Software Architecture Document.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Meghanathan N. Review of access control models for cloud computing. - Jackson State University, 2012. -78-79 с.
2. Decat M., Lagaisse B., Joosen W. Middleware for efficient and confidentiality-aware federation of access control policies. - Journal of Internet Services and Applications 2014.- 3-4 с.
3. Krebs R., Momm C., Kounev C. Architectural concerns in multi-tenant saas applications.- Karlsruhe Institute of Technology 2013.- 427-428 с.
4. Satyanarayana S. Cloud computing: SaaS. - GESJ: Computer Science and Telecommunications №.4(36) 2012.- 77-78 с.
5. Chari S. Using Recommenders for Discretionary Access Control.- IBM Research 2009.- 3-4 с.
6. Bell E., LaPadula L. Secure computer systems: Unified exposition and multics interpretation Technical Report MTR-2997 2003.- 5-6 с.
7. Bird P. A Multi-Purpose Implementation of Mandatory Access Control in Relational Database Management Systems.- IBM Toronto Software Laboratory 2004.- 1010 – 1011 с.
8. Blokdyk G. Trusted Computer System Evaluation Criteria.- United States Department of Defense 1985. – 4-6 с.
9. Kuhn D. Role-Based Access Control.- National Computer Security Conference, №15 1992. - 554–563 с.
10. Ferraiolo D. Attribute Based Access Control | CSRC | CSRC.- Computer Security Division, Information Technology Laboratory 2016.- 12-13.
11. Schnitzer D. Guide to Attribute Based Access Control (ABAC) Definition and Considerations.- NIST Special Publication №800-162 2013.- 7-8 с.
12. The Open Group and OMG TOGAF ADM and MDA, 2004. - 5-6 с.
13. Clements P. Documenting software architectures: views and beyond. International Conference on Software Engineering №25 2003. Proceedings. 740-741 с.

ДОДАТКИ

Додаток А. Лістинг коду модулю каталогу клієнтів

```
namespace TenantCatalog {
  using System;
  using System.IO;
  using System.Threading.Tasks;
  using Microsoft.AspNetCore.Mvc;
  using Microsoft.Azure.WebJobs;
  using Microsoft.Azure.WebJobs.Extensions.Http;
  using Microsoft.AspNetCore.Http;
  using Newtonsoft.Json;
  using Finbuckle.MultiTenant;
  using Microsoft.WindowsAzure.Storage;
  using Microsoft.WindowsAzure.Storage.Blob;
  using System.Collections.Generic;
  using System.Linq;
  using System.Text;

  public static class TenantResolver {
    [FunctionName("resolve")]
    public static async Task < IActionResult > Run(
      [HttpTrigger(AuthorizationLevel.Anonymous, "get", Route = "resolve/{tenant}")] HttpRequest
req, string tenant) {

      var ConnectionString =
Environment.GetEnvironmentVariable(Consts.AzureWebJobStorage);
      // Setup the connection to the storage account
      CloudStorageAccount storageAccount = CloudStorageAccount.Parse(ConnectionString);
      // Connect to the blob storage
      CloudBlobClient serviceClient = storageAccount.CreateCloudBlobClient();
      // Connect to the blob container
      CloudBlobContainer container = serviceClient.GetContainerReference(Consts.BlobCatalog);
```

```
      // Connect to the blob file
      CloudBlockBlob blob = container.GetBlockBlobReference(Consts.CatalogFileName);
      // Get the blob file as text

      string content;
      using(var memoryStream = new MemoryStream()) {
        await blob.DownloadToStreamAsync(memoryStream);

        memoryStream.Position = 0;

        using(var reader = new StreamReader(memoryStream, Encoding.Unicode)) {
          content = await reader.ReadToEndAsync();
        }
      }

      var availableTenants = JsonConvert.DeserializeObject < IEnumerable < TenantInfo >>
(content);
      var tenantData = availableTenants.FirstOrDefault(t => t.Identifier == tenant);

      if (tenantData == null) {
        return new NotFoundResult();
      }

      tenantData.ConnectionString =
Environment.GetEnvironmentVariable(Consts.ConnectionStringKeyPrefix + tenant,
EnvironmentVariableTarget.Process);

      return new OkObjectResult(tenantData);
    }
  }
}
```

Додаток Б. Software Architecture Document

“WAVEACCESS” LTD

Access control middleware for multi-tenant cloud-based applications
Software Architecture Document (SAD)

CONTENT OWNER: Nazar Zghura

DOCUMENT NUMBER:	RELEASE/REVISION:	RELEASE/REVISION DATE:
• 1	• 0.5v	• 15.03.2022
• 2	• 0.9v	• 20.04.2022
•	•	•
•	•	•
•	•	•
•	•	•

Table of Contents

Documentation Roadmap **3**
- Document Management and Configuration Control Information 3
- Purpose and Scope of the SAD 3
- How the SAD Is Organized 4
- Stakeholder Representation 5
- Viewpoint Definitions 5
  - Provider Viewpoint Definition 6
    - Abstract 7
    - Stakeholders and Their Concerns Addressed 7
    - Elements, Relations, Properties, and Constraints 7
    - Language(s) to Model/Represent Conforming Views 7
    - Applicable Evaluation/Analysis Techniques and Consistency/Completeness Criteria 7
    - Viewpoint Source 7
  - Infrastructure Viewpoint Definition 8
    - Abstract 8
    - Stakeholders and Their Concerns Addressed 8
    - Elements, Relations, Properties, and Constraints 8
    - Language(s) to Model/Represent Conforming Views 8
    - Applicable Evaluation/Analysis Techniques and Consistency/Completeness Criteria 8
    - Viewpoint Source 8
- How a View is Documented 8
- Relationship to Other SADs 9
- Process for Updating this SAD 10

Architecture Background **11**
- Problem Background 11
  - System Overview 11
  - Goals and Context 11
  - Significant Driving Requirements 11
- Solution Background 12
  - Architectural Approaches 12
  - Analysis Results 12
  - Requirements Coverage 12
  - Summary of Background Changes Reflected in Current Version 12

Product Line Reuse Considerations 13

Views **14**

Use-case View 15

View Description 15

View Packet Overview 15

Architecture Background 15

Variability Mechanisms 15

View Packets 15

View packet 1 16

Primary Presentation 16

Visual representation of use-cases with UML elements. 16

Element Catalog 16

Elements 16

Relations 16

Architecture Layers View 16

View Description 16

View Packet Overview 16

Architecture Background 16

Variability Mechanisms 16

View Packets 16

View packet 2 16

Primary Presentation 16

Element Catalog 17

Elements 17

Relations 17

Deployment View 17

View Description 17

View Packet Overview 17

Architecture Background 17

Variability Mechanisms 17

View Packets 17

View packet 3 17

Primary Presentation 17

Element Catalog 18

Elements 18

Relations 18

Relations Among Views **19**

General Relations Among Views 19
View-to-View Relations 19

Referenced Materials **20**

Directory **21**
Index 21
Glossary 21
Acronym List 22

Sample Figures & Tables **24**

List of Figures

Figure 1: Use-cases View presentation 24
Figure 2: Architecture Layers View presentation 25
Figure 3: Deployment View presentation 26

List of Tables

Table 1: Stakeholders and Relevant Viewpoints 8

1 Documentation Roadmap

1.1 Document Management and Configuration Control Information

CONTENTS OF THIS SECTION: This section identifies the version, release date, and other relevant management and configuration control information associated with the current version of the document. Optional items for this section include: change history and an overview of significant changes from version to version.

- Revision Number: <<2>>
- Revision Release Date: << *20.04.2022*>>
- Purpose of Revision: << Outline viewpoints/ views of stakeholders>>
- Scope of Revision: <<*Viewpoints Definitions, Views*>>

1.2 Purpose and Scope of the SAD

CONTENTS OF THIS SECTION: This section explains the SAD's overall purpose and scope, the criteria for deciding which design decisions are architectural (and therefore documented in the SAD), and which design decisions are non-architectural (and therefore documented elsewhere).

This SAD specifies the software architecture for **access control middleware for multi-tenant cloud-bassed application.** All information regarding the software architecture may be found in this document, although much information is incorporated by reference to other documents.

What is software architecture? The software architecture for a system[1] is the structure or structures of that system, which comprise software elements, the externally-visible properties of those elements, and the relationships among them [Bass 2003]. "Externally visible" properties refers to those assumptions other elements can make of an element, such as its provided services, performance characteristics, fault handling, shared resource usage, and so on. This definition provides the basic litmus test for what information is included in this SAD, and what information is relegated to downstream documentation.

Elements and relationships. The software architecture first and foremost embodies information about how the elements relate to each other. This means that architecture specifically omits certain information about elements that does not pertain to their interaction. Thus, a software architecture is an *abstraction* of a system that suppresses details of elements that do not affect how they use, are used by, relate to, or interact with other elements. Elements interact with each other by means of interfaces that partition details about an element into public and private parts. Software architecture is concerned with the public side of this division, and that will be documented in this SAD accordingly. On the other hand, private details of elements—details having to do solely with internal implementation—are not architectural and will not be documented in a SAD.

Multiple structures. The definition of software architecture makes it clear that systems can and do comprise more than one structure and that no one structure holds the irrefutable claim to being the architecture. The neurologist, the orthopedist, the hematologist, and the dermatologist all take a different perspective on the structure of a human body. Ophthalmologists, cardiologists, and podiatrists concentrate on subsystems. And the kinesiologist and psychiatrist are concerned with different aspects of the entire arrangement's behavior. Although these perspectives are pictured differently and have very different properties, all are inherently related; together they describe the architecture of the human body. So it is with software. Modern systems are more than complex enough to make it difficult to grasp them all at

[1] Here, a system may refer to a system of systems.

once. Instead, we restrict our attention at any one moment to one (or a small number) of the software system's structures. To communicate meaningfully about an architecture, we must make clear which structure or structures we are discussing at the moment—which *view* we are taking of the architecture. Thus, this SAD follows the principle that documenting a software architecture is a matter of documenting the relevant views and then documenting information that applies to more than one view.

For example, all non-trivial software systems are partitioned into implementation units; these units are given specific responsibilities, and are the basis of work assignments for programming teams. This kind of element will comprise programs and data that software in other implementation units can call or access, and programs and data that are private. In large projects, the elements will almost certainly be subdivided for assignment to sub-teams. This is one kind of structure often used to describe a system. It is a very static structure, in that it focuses on the way the system's functionality is divided up and assigned to implementation teams.

Other structures are much more focused on the way the elements interact with each other at runtime to carry out the system's function. Suppose the system is to be built as a set of parallel processes. The set of processes that will exist at runtime, the programs in the various implementation units described previously that are strung together sequentially to form each process, and the synchronization relations among the processes form another kind of structure often used to describe a system.

None of these structures alone is *the* architecture, although they all convey architectural information. The architecture consists of these structures as well as many others. This example shows that since architecture can comprise more than one kind of structure, there is more than one kind of element (e.g., implementation unit and processes), more than one kind of interaction among elements (e.g., subdivision and synchronization), and even more than one context (e.g., development time versus runtime). By intention, the definition does not specify what the architectural elements and relationships are. Is a software element an object? A process? A library? A database? A commercial product? It can be any of these things and more.

These structures will be represented in the views of the software architecture that are provided in Section 3.

Behavior. Although software architecture tends to focus on structural information, *behavior of each element is part of the software architecture* insofar as that behavior can be observed or discerned from the point of view of another element. This behavior is what allows elements to interact with each other, which is clearly part of the software architecture and will be documented in the SAD as such. Behavior is documented in the element catalog of each view.

1.3 How the SAD Is Organized

CONTENTS OF THIS SECTION: This section provides a narrative description of the major sections of the SAD and the overall contents of each. Readers seeking specific information can use this section to help them locate it more quickly.

This SAD is organized into the following sections:

- **Section 1 ("Documentation Roadmap") provides information about this document and its intended audience**. It provides the roadmap and document overview. Every reader who wishes to find information relevant to the software architecture described in this document should begin by reading Section 1, which describes how the document is organized, which stakeholder viewpoints are represented, how stakeholders are expected to use it, and where information may be found. Section 1 also provides information about the views that are used by this SAD to communicate the software architecture.

- **Section 2 (“Architecture Background”) explains why the architecture is what it is.** It provides a system overview, establishing the context and goals for the development. It describes the background and rationale for the software architecture. It explains the constraints and influences that led to the current architecture, and it describes the major architectural approaches that have been utilized in the architecture. It includes information about evaluation or validation performed on the architecture to provide assurance it meets its goals.
- **Section 3 (“Views”) and Section 4 (“Relations Among Views”) specify the software architecture**. Views specify elements of software and the relationships between them. A view corresponds to a viewpoint (see Section 1.5), and is a representation of one or more structures present in the software (see Section 1.2).
- **Sections 5 (“Referenced Materials”) and 6 (“Directory”) provide reference information for the reader.** Section 5 provides look-up information for documents that are cited elsewhere in this SAD. Section 6 is a *directory*, which is an index of architectural elements and relations telling where each one is defined and used in this SAD. The section also includes a glossary and acronym list.

1.4 Stakeholder Representation

This section provides a list of the stakeholder roles considered in the development of the architecture described by this SAD. For each, the section lists the concerns that the stakeholder has that can be addressed by the information in this SAD.

Each stakeholder of a software system—customer, user, project manager, coder, analyst, tester, and so on—is concerned with different characteristics of the system that are affected by its software architecture. For example, the user is concerned that the system is reliable and available when needed; the customer is concerned that the architecture can be implemented on schedule and to budget; the manager is worried (in addition to cost and schedule) that the architecture will allow teams to work largely independently, interacting in disciplined and controlled ways. The developer is worried about strategies to achieve all of those goals. The security analyst is concerned that the system will meet its information assurance requirements, and the performance analyst is similarly concerned with it satisfying real-time deadlines.

This information is represented as a matrix, where the rows list stakeholder roles, the columns list concerns, and a cell in the matrix contains an indication of how serious the concern is to a stakeholder in that role. This information is used to motivate the choice of viewpoints chosen in Section 1.5.

1.5 Viewpoint Definitions

CONTENTS OF THIS SECTION: This section provides a short textual definition of a viewpoint and how the concept is used in this SAD. The section describes viewpoints that may be used in the SAD. The specific viewpoints will be tailored by the organization.

The SAD employs a stakeholder-focused, multiple view approach to architecture documentation, as required by ANSI/IEEE 1471-2000, the recommended best practice for documenting the architecture of software-intensive systems [IEEE 1471].

As described in Section 1.2, a software architecture comprises more than one software structure, each of which provides an engineering handle on different system qualities. A *view* is the specification of one or more of these structures, and documenting a software architecture, then, is a matter of documenting the relevant views and then documenting information that applies to more than one view [Clements 2002].

ANSI/IEEE 1471-2000 provides guidance for choosing the best set of views to document, by bringing stakeholder interests to bear. It prescribes defining a set of viewpoints to satisfy the stakeholder

community. A viewpoint identifies the set of concerns to be addressed, and identifies the modeling techniques, evaluation techniques, consistency checking techniques, etc., used by any conforming view. A view, then, is a viewpoint applied to a system. It is a representation of a set of software elements, their properties, and the relationships among them that conform to a defining viewpoint. Together, the chosen set of views show the entire architecture and all of its relevant properties. A SAD contains the viewpoints, relevant views, and information that applies to more than one view to give a holistic description of the system.

The remainder of Section 1.5 defines the viewpoints used in this SAD. The following table summarizes the stakeholders in this project and the viewpoints that have been included to address their concerns.

Table 1: Stakeholders and Relevant Viewpoints

Stakeholder	**Viewpoint(s) that apply to that class of stakeholder's concerns**
Developer	Provider Viewpoint
DevOps (TechOps/Infrastructure Engineer)	Infrastructure Viewpoint
Software architect	Provider Viewpoint
Quality Assurance Engineer	Provider Viewpoint

1.5.1 Provider Viewpoint Definition

1.5.1.1 Abstract

Defines views, applicable to the developer, supporting and extending the system

1.5.1.2 Stakeholders and Their Concerns Addressed

Developer - security, performance, user interface, database.
System Architect - security, performance, data consistency, extendability.
Quality Assurance Engineer - acceptance criteria, user interface, stability.

1.5.1.3 Elements, Relations, Properties, and Constraints

Use-cases, Layers, Models

1.5.1.4 Language(s) to Model/Represent Conforming Views

UML, Visual elements

1.5.1.5 Applicable Evaluation/Analysis Techniques and Consistency/Completeness Criteria

1.5.1.6 Viewpoint Source

1.5.2 Infrastructure Viewpoint Definition

1.5.2.1 Abstract

Defines views, applicable to the infrastructure engineer.

1.5.2.2 Stakeholders and Their Concerns Addressed

Infrastructure engineer - security, performance, database, deployment

1.5.2.3 Elements, Relations, Properties, and Constraints

Azure Web App, Azure Function, Blob Storage

1.5.2.4 Language(s) to Model/Represent Conforming Views

UML, Visual elements

1.5.2.5 Applicable Evaluation/Analysis Techniques and Consistency/Completeness Criteria

1.5.2.6 Viewpoint Source

1.6 How a View is Documented

Section 3 of this SAD contains one view for each viewpoint listed in Section 1.5. Each view is documented as a set of view packets. A view packet is the smallest bundle of architectural documentation that might be given to an individual stakeholder.

Each view is documented as follows, where the letter *i* stands for the number of the view: 1, 2, etc.:

- Section 3.i: Name of view.
- Section 3.i.1: View description. This section describes the purpose and contents of the view. It should refer to (and match) the viewpoint description in Section 1.5 to which this view conforms.
- Section 3.i.2: View packet overview. This section shows the set of view packets in this view, and provides rationale that explains why the chosen set is complete and non-duplicative. The set of view packets may be listed textually, or shown graphically in terms of how they partition the entire architecture being shown in the view.
- Section 3.i.3: Architecture background. Whereas the architecture background of Section 2 pertains to those constraints and decisions whose scope is the entire architecture, this section provides any architecture background (including significant driving requirements, design approaches, patterns, analysis results, and requirements coverage) that applies to this view.

- Section 3.i.4: Variability mechanisms. This section describes any architectural variability mechanisms (e.g., adaptation data, compile-time parameters, variable replication, and so forth) described by this view, including a description of how and when those mechanisms may be exercised and any constraints on their use.
- Section 3.i.5: View packets. This section presents all of the view packets given for this view. Each view packet is described using the following outline, where the letter *j* stands for the number of the view packet being described: 1, 2, etc.

- Section 3.i.5.j: View packet #j.

- Section 3.i.5.j.1: Primary presentation. This section presents the elements and the relations among them that populate this view packet, using an appropriate language, languages, notation, or tool-based representation.
- Section 3.i.5.j.2: Element catalog. Whereas the primary presentation shows the important elements and relations of the view packet, this section provides additional information needed to complete the architectural picture. It consists of the following subsections:
  - Section 3.i.5.j.2.1: Elements. This section describes each element shown in the primary presentation, details its responsibilities of each element, and specifies values of the elements' relevant *properties*, which are defined in the viewpoint to which this view conforms.
  - Section 3.i.5.j.2.2: Relations. This section describes any additional relations among elements shown in the primary presentation, or specializations or restrictions on the relations shown in the primary presentation.

1.7 Relationship to Other SADs

CONTENTS OF THIS SECTION: This section describes the relationship between this SAD and other architecture documents, both system and software. For example, a large project may choose to have one SAD that defines the system-of-systems architecture, and other SADs to define the architecture of systems or subsystems. An embedded system may well have a *system* architecture document, in which case this section would explain how the information in here traces to information there.

If none, say "Not applicable."

Not applicable.

1.8 Process for Updating this SAD

CONTENTS OF THIS SECTION: This section describes the process a reader should follow to report discrepancies, errors, inconsistencies, or omissions from this SAD. The section also includes necessary contact information for submitting the report. If a form is required, either a copy of the blank form that may be photocopied is included, or a reference to an online version is provided. This section also describes how error reports are handled, and how and when a submitter will be notified of the issue's disposition.

Not applicable.

2 Architecture Background

2.1 Problem Background

CONTENTS OF THIS SECTION: The sub-parts of Section 2.1 explain the constraints that provided the significant influence over the architecture.

Cloud-based Software-as-a-service (SaaS) applications have been growing in popularity for several years. SaaS providers are evolving to application-level multi-tenancy (all tenants share the application instances, platform and data store with the aim of maximizing resource sharing). For multi-tenant SaaS applications, access control often is organized on application level. But, such access control is complicated because both the provider and all tenants need to specify access rules for the whole application. Moreover, these rules must all be securely combined and correctly enforced in the shared multi-tenant application.

2.1.1 System Overview

CONTENTS OF THIS SECTION: This section describes the general function and purpose for the system or subsystem whose architecture is described in this SAD.

System enables the provider and tenants to express their access rules over cloud-based application, combines these rules and enforces them at runtime. In order to express their rules, the system uses the model of attribute-based access control. To simplify the overall access control management, system employs a three-layered approach in which the provider builds on the attributes and policies defined by system, and the tenants build on the attributes and policies defined by the provider.

2.1.2 Goals and Context

CONTENTS OF THIS SECTION: This section describes the goals and major contextual factors for the software architecture. The section includes a description of the role software architecture plays in the life cycle, the relationship to system engineering results and artifacts, and any other relevant factors.

The goal is to simplify the process of building and managing multi-tenant cloud-based applications by taking of the complexity of implementing and maintaining complicated access-control infrastructure.

2.1.3 Significant Driving Requirements

CONTENTS OF THIS SECTION: This section describes behavioral and quality attribute requirements (original or derived) that shaped the software architecture. Included are any scenarios that express driving behavioral and quality attribute goals, such as those crafted during a Quality Attribute Workshop (QAW) [Barbacci 2003] or software architecture evaluation using the Architecture Tradeoff Analysis Method[2]SM (ATAM[SM]) [Bass 2003].

1. The system should enable the provider to constrain the tenants.
2. The system should enable each tenant to constrain its users.
3. The system should combine the policies of all involved organizations securely.
4. The system should enforce the access rules appropriate for each request in the shared application.
5. The system should be applicable for cloud-based applications, hosted on Azure.

[2]SM Quality Attribute Workshop and QAW and Architecture Tradeoff Analysis Method and ATAM are service marks of Carnegie Mellon University.

2.2 Solution Background

CONTENTS OF THIS SECTION: The sub-parts of Section 2.2 provide a description of why the architecture is the way that it is, and a convincing argument that the architecture is the right one to satisfy the behavioral and quality attribute goals levied upon it.

2.2.1 Architectural Approaches

CONTENTS OF THIS SECTION: This section provides a rationale for the major design decisions embodied by the software architecture. It describes any design approaches applied to the software architecture, including the use of architectural styles or design patterns, when the scope of those approaches transcends any single architectural view. The section also provides a rationale for the selection of those approaches. It also describes any significant alternatives that were seriously considered and why they were ultimately rejected. The section describes any relevant COTS issues, including any associated trade studies.

TOGAF Architecture Development Method (ADM) was used as a prime approach for developing the architecture of the system under design.

2.2.2 Analysis Results

CONTENTS OF THIS SECTION: This section describes the results of any quantitative or qualitative analyses that have been performed that provide evidence that the software architecture is fit for purpose. If an Architecture Tradeoff Analysis Method evaluation has been performed, it is included in the analysis sections of its final report. This section refers to the results of any other relevant trade studies, quantitative modeling, or other analysis results.

As a result of analysis, the best suitable approach is to architect the access control system as a middleware, which can be integrated into http request pipeline for effective run-time policies binding and evaluation with the minimal overhead on top of the cloud-based application.

2.2.3 Requirements Coverage

CONTENTS OF THIS SECTION: This section describes the requirements (original or derived) addressed by the software architecture, with a short statement about where in the architecture each requirement is addressed.

1. The system enables the provider to build policies that are used to constrain the tenants

2. The system enables tenants to build their own policies.

3. The system uses decision trees to combine all involved stakeholder’s policies

4. The system handles each request separately, using the middleware approach to evaluate all access policies for the request.

5. The system is designed to be used in cloud-based environment and act as a middleware for Azure Web Application

2.2.4 Summary of Background Changes Reflected in Current Version

CONTENTS OF THIS SECTION: For versions of the SAD after the original release, this section summarizes the actions, decisions, decision drivers, analysis and trade study results that became decision drivers, requirements changes that became decision drivers, and how these decisions have caused the architecture to evolve or change.

2.3 Product Line Reuse Considerations

Not Applicable.

3 Views

CONTENTS OF THIS SECTION: The sub-parts of Section 3 specify the views corresponding to the viewpoints listed in Section 1.5.

This section contains the views of the software architecture. A view is a representation of a whole system from the perspective of a related set of concerns [IEEE 1471]. Concretely, a view shows a particular type of software architectural elements that occur in a system, their properties, and the relations among them. A view conforms to a defining viewpoint.

Architectural views can be divided into three groups, depending on the broad nature of the elements they show. These are:

- Module views. Here, the elements are modules, which are units of implementation. Modules represent a code-based way of considering the system. Modules are assigned areas of functional responsibility, and are assigned to teams for implementation. There is less emphasis on how the resulting software manifests itself at runtime. Module structures allow us to answer questions such as: What is the primary functional responsibility assigned to each module? What other software elements is a module allowed to use? What other software does it actually use? What modules are related to other modules by generalization or specialization (i.e., inheritance) relationships?
- Component-and-connector views. Here, the elements are runtime components (which are principal units of computation) and connectors (which are the communication vehicles among components). Component and connector structures help answer questions such as: What are the major executing components and how do they interact? What are the major shared data stores? Which parts of the system are replicated? How does data progress through the system? What parts of the system can run in parallel? How can the system's structure change as it executes?
- Allocation views. These views show the relationship between the software elements and elements in one or more external environments in which the software is created and executed. Allocation structures answer questions such as: What processor does each software element execute on? In what files is each element stored during development, testing, and system building? What is the assignment of the software element to development teams?

These three kinds of structures correspond to the three broad kinds of decisions that architectural design involves:

- How is the system to be structured as a set of code units (modules)
- How is the system to be structured as a set of elements that have run-time behavior (components) and interactions (connectors) ?
- How is the system to relate to non-software structures in its environment (such as CPUs, file systems, networks, development teams, etc.)?

Often, a view shows information from more than one of these categories. However, unless chosen carefully, the information in such a hybrid view can be confusing and not well understood.

The views presented in this SAD are the following:

Name of view	Viewtype that defines this view	Types of elements and relations shown		Is this a module view?	Is this a component-and-connector view?	Is this an allocation view?
Use-case	Provider viewpoint	Visual	Visual	Yes	No	No
Architecture Layers	Provider Viewpoint	Visual	Visual	Yes	No	No
Deployment	Infrastructure viewpoint	Visual	Visual	Yes	No	No

3.1 Use-case View

CONTENTS OF THIS SECTION: For each view documented in this SAD, the sub-parts of Section 3.1 specify it using the outline given in Section 1.6. This part of the template assumes you are using view packets to divide up a view into management chunks. If not, then see the note in Section 1.6 as to what outline to use for each view.

3.1.1 View Description

Describes scenarios of use of functionality of the system, conforming to Provider Viewpoint.

3.1.2 View Packet Overview

This view has been divided into the following view packets for convenience of presentation:

View Packet 1

3.1.3 Architecture Background

View is developed with the driving requirements in mind. Separate use-cases described for Provider Administrator, Tenant Administrator and Application end-user.

3.1.4 Variability Mechanisms

Main variability mechanisms are graphically represented and outlined as configuration flow.

3.1.5 View Packets

CONTENTS OF THIS SECTION: For each view packet in the view, this section describes it using the outline given in Section 1.6.

3.1.5.1 View packet 1

3.1.5.1.1 Primary Presentation

Visual representation of use-cases with UML elements.

3.1.5.1.2 Element Catalog
Figure 1

3.1.5.1.2.1 Elements
User, system logic element, system data storage, system user interface.

3.1.5.1.2.2 Relations
Authorization flow, configuration flow

3.2 Architecture Layers View

CONTENTS OF THIS SECTION: For each view documented in this SAD, the sub-parts of Section 3.1 specify it using the outline given in Section 1.6. This part of the template assumes you are using view packets to divide up a view into management chunks. If not, then see the note in Section 1.6 as to what outline to use for each view.

3.2.1 View Description
Describes the layers of composition, of which the architecture is composed.

3.2.2 View Packet Overview

This view has been divided into the following view packets for convenience of presentation:

View Packet 2

3.2.3 Architecture Background
View is developed with the driving requirements in mind. It complies with the concerns separation principle.

3.2.4 Variability Mechanisms
Not applicable.

3.2.5 View Packets

CONTENTS OF THIS SECTION: For each view packet in the view, this section describes it using the outline given in Section 1.6.

3.2.5.1 View packet 2

3.2.5.1.1 Primary Presentation

Visual presentation of architecture layers and relations between them.

3.2.5.1.2 Element Catalog

Figure 2

3.2.5.1.2.1 Elements

Architecture layer

3.2.5.1.2.2 Relations

Utilize flow

3.3 Deployment View

CONTENTS OF THIS SECTION: For each view documented in this SAD, the sub-parts of Section 3.1 specify it using the outline given in Section 1.6. This part of the template assumes you are using view packets to divide up a view into management chunks. If not, then see the note in Section 1.6 as to what outline to use for each view.

3.3.1 View Description

Describes cloud-based units that comprise the system.

3.3.2 View Packet Overview

This view has been divided into the following view packets for convenience of presentation:

View Packet 3

3.3.3 Architecture Background

View is developed with the driving requirements in mind. It describes units, based on the available cloud services, provided by Azure, and the relation between these units, required for sucessful deploy/update process,

3.3.4 Variability Mechanisms

Not applicable.

3.3.5 View Packets

CONTENTS OF THIS SECTION: For each view packet in the view, this section describes it using the outline given in Section 1.6.

3.3.5.1 View packet 3

3.3.5.1.1 Primary Presentation

Visual representation of infrastructure units and relation between them.

3.3.5.1.2 Element Catalog

Figure 3

3.3.5.1.2.1 Elements

Azure Web Application visual presentation element
Azure Function visual presentation element
Azure Blob Storage visual presentation element

3.3.5.1.2.2 Relations

Read data flow, write data flow, utilize flow

4 Relations Among Views

Each of the views specified in Section 3 provides a different perspective and design handle on a system, and each is valid and useful in its own right. Although the views give different system perspectives, they are not independent. Elements of one view will be related to elements of other views, and we need to reason about these relations. For example, a module in a decomposition view may be manifested as one, part of one, or several components in one of the component-and-connector views, reflecting its runtime alter-ego. In general, mappings between views are many to many. Section 4 describes the relations that exist among the views given in Section 3. As required by ANSI/IEEE 1471-2000, it also describes any known inconsistencies among the views.

4.1 General Relations Among Views

CONTENTS OF THIS SECTION: This section describes the general relationship among the views chosen to represent the architecture. Also in this section, consistency among those views is discussed and any known inconsistencies are identified.

Each View described the system from the point of view of a group of stakeholders

4.2 View-to-View Relations

CONTENTS OF THIS SECTION: For each set of views related to each other, this section shows how the elements in one view are related to elements in another.

5 Referenced Materials

CONTENTS OF THIS SECTION: This section provides citations for each reference document. Provide enough information so that a reader of the SAD can be reasonably expected to locate the document.

IEEE 2010	J.M. Alcaraz Calero, N. Edwards, J. Kirschnick, L. Wilcock, and M. Wray. *Toward a multi-tenancy authorization system for cloud services.*
IEEE 2008	W. Sun, X. Zhang, C.J. Guo, P. Sun, and H. Su. *Software as a service: Configuration and customization perspectives.*
CEC/EEE 2007	C.J. Guo, W. Sun, Y. Huang, Z.H. Wang, and B. Gao. *A framework for native multi-tenancy application development and management.*
IFIP 2011	K. Fatema, D. Chadwick, and S. Lievens. *A multi-privacy policy enforcement system.*
IEEE 1471	ANSI/IEEE-1471-2000, *IEEE Recommended Practice for Architectural Description of Software-Intensive Systems*, 21 September 2000.

6 Directory

6.1 Index

CONTENTS OF THIS SECTION: This section provides an index of all element names, relation names, and property names. For each entry, the following are identified:

- the location in the SAD where it was defined
- each place it was used

Ideally, each entry will be a hyperlink so a reader can instantly navigate to the indicated location.

6.2 Glossary

CONTENTS OF THIS SECTION: This section provides a list of definitions of special terms and acronyms used in the SAD. If terms are used in the SAD that are also used in a parent SAD and the definition is different, this section explains why.

Term	Definition
software architecture	The structure or structures of that system, which comprise software elements, the externally visible properties of those elements, and the relationships among them [Bass 2003]. "Externally visible" properties refer to those assumptions other elements can make of an element, such as its provided services, performance characteristics, fault handling, shared resource usage, and so on.
view	A representation of a whole system from the perspective of a related set of concerns [IEEE 1471]. A representation of a particular type of software architectural elements that occur in a system, their properties, and the relations among them. A view conforms to a defining viewpoint.
view packet	The smallest package of architectural documentation that could usefully be given to a stakeholder. The documentation of a view is composed of one or more view packets.
viewpoint	A specification of the conventions for constructing and using a view; a pattern or template from which to develop individual views by establishing the purposes and audience for a view, and the techniques for its

	creation and analysis [IEEE 1471]. Identifies the set of concerns to be addressed, and identifies the modeling techniques, evaluation techniques, consistency checking techniques, etc., used by any conforming view.
provider	Organization which develops and supports cloud-based application
tenant	Represents a customer organization, using the cloud-based application

6.3 Acronym List

API	Application Programming Interface; Application Program Interface; Application Programmer Interface
ATAM	Architecture Tradeoff Analysis Method
CMM	Capability Maturity Model
CMMI	Capability Maturity Model Integration
CORBA	Common object request broker architecture
COTS	Commercial-Off-The-Shelf
EPIC	Evolutionary Process for Integrating COTS-Based Systems
IEEE	Institute of Electrical and Electronics Engineers
KPA	Key Process Area
OO	Object Oriented
ORB	Object Request Broker
OS	Operating System
QAW	Quality Attribute Workshop
RUP	Rational Unified Process
SAD	Software Architecture Document
SDE	Software Development Environment
SEE	Software Engineering Environment
SEI	Software Engineering Institute Systems Engineering & Integration Software End Item
SEPG	Software Engineering Process Group
SLOC	Source Lines of Code
SW-CMM	Capability Maturity Model for Software
CMMI-SW	Capability Maturity Model Integrated - includes Software Engineering

UML	Unified Modeling Language

7 Sample Figures & Tables

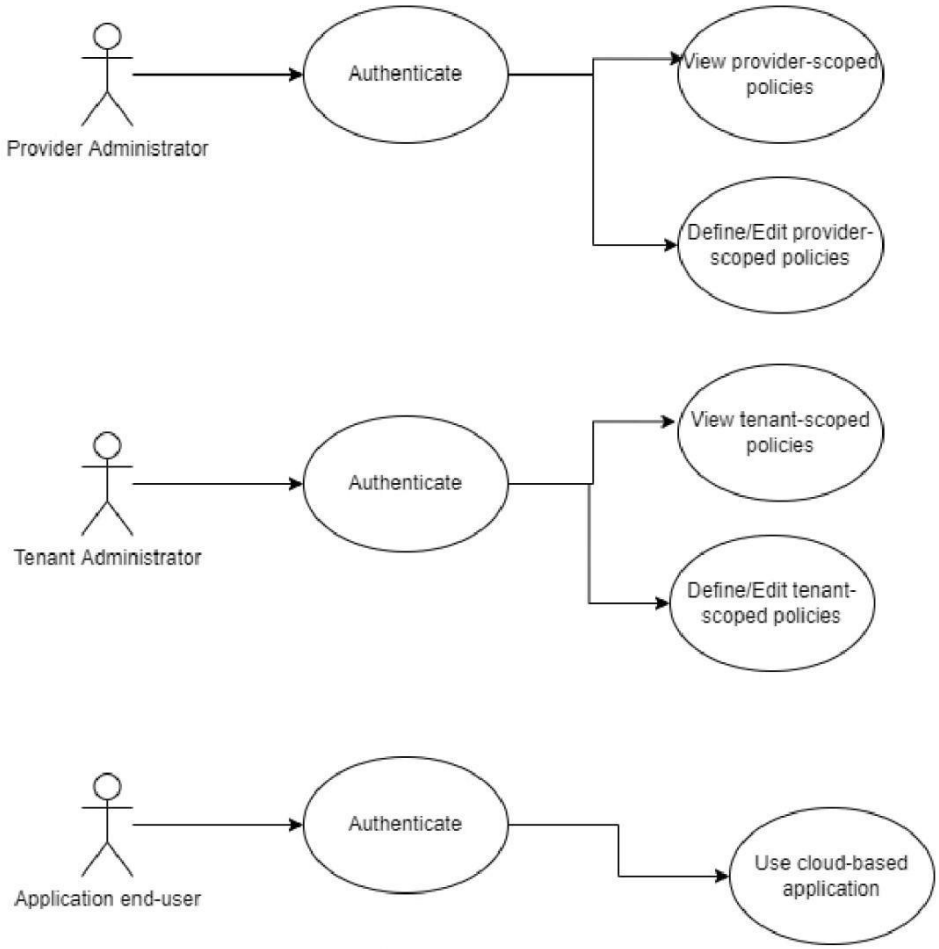

Figure 1: Use-cases View presentation

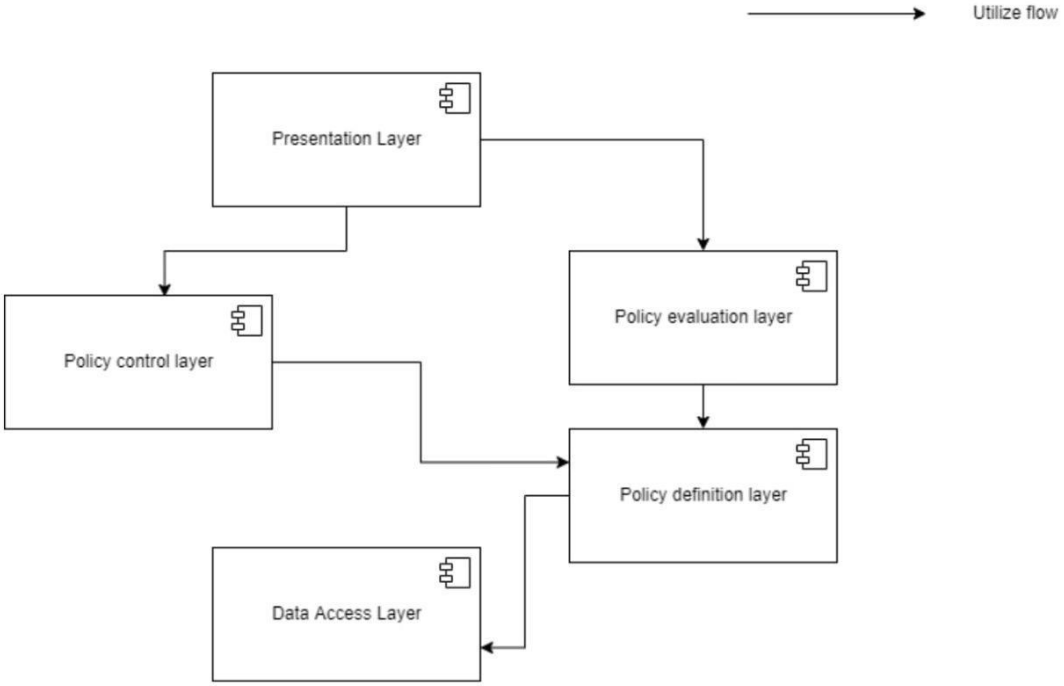

Figure 2: Architecture Layers View presentation

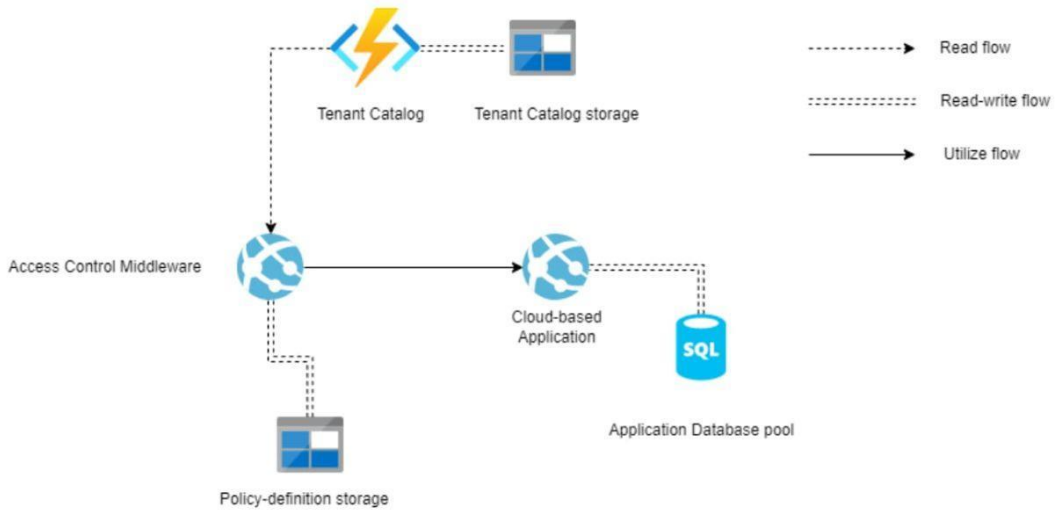

Figure 3: Deployment View presentation