КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
ІМЕНІ ТАРАСА ШЕВЧЕНКА

Факультет комп'ютерних наук та кібернетики
Кафедра теорії та технології програмування

Кваліфікаційна робота
на здобуття ступеня бакалавра
за спеціальністю 122 «Комп'ютерні науки»
на тему:

СЕРВІС УПРАВЛІННЯ ПРОЦЕСОМ ІНВЕНТАРИЗАЦІЇ МАЙНА ТА РЕСУРСІВ КОМПАНІЙ. ПІДСИСТЕМА КОРИСТУВАЧА

Виконала студентка 4-го курсу
Діна ФОРМАКІДОВ

(підпис)

Науковий керівник:
доцент, кандидат фіз.-мат. наук
Людмила ОМЕЛЬЧУК

(підпис)

Засвідчую, що в цій роботі немає запозичень з праць інших авторів без відповідних посилань
Студент

Роботу розглянуто і допущено до захисту на засіданні кафедри теорії та
технології програмування
« 05 » червня 2023 р.,
протокол № 18
Завідувач кафедри
Микола НІКІТЧЕНКО

Київ-2023

РЕФЕРАТ

Загальний обсяг роботи 50 сторінок, основний текст викладено на 46 сторінках, 16 ілюстрацій, 4 таблиці, 15 джерел посилань, 3 додатки.

ПРОГРАМА, МІКРОСЕРВІС, АРХІТЕКТУРА, АВТОРИЗАЦІЯ, ІНТЕГРОВАНЕ СЕРЕДОВИЩЕ РОЗРОБКИ, БАЗА ДАНИХ, МОВА ПРОГРАМУВАННЯ.

Об’єктом розроблення програмного засобу є процес інвентаризацію та управління майном компанії з точки зору користувача. Предметом роботи є вебсистема для автоматизації процесу інвентаризації та управління майном компанії, що включає управління наявним майном, закупівлю техніки та іншого обладнання, а також управління витратами компанії.

Метою кваліфікаційної роботи є розробка вебсистеми для управління ресурсами компанії з точки зору користувача.

Методи розроблення: комп’ютерне моделювання, розробка програмного продукту.

Інструменти розроблення: система розробки – операційна система Windows 11, інтегроване середовище розробки – Visual Studio, мова реалізації C#.

Результати роботи: було розроблено та впроваджено програмний продукт, за допомогою якого можна відстежувати витрати на обладнання, назначати за кожним співробітником техніку та назначати процес від покупки до отримання техніки співробітником.

ЗМІСТ

СКОРОЧЕННЯ ТА УМОВНІ ПОЗНАЧЕННЯ

API – Application Programming Interface, прикладний програмний інтерфейс;

CRUD – Create Read Update Delete, створення, читання, оновлення і вилучення;

CSV – Comma-separated values, значення, розділені комою;

CSRF – Cross-site Request Forgery, міжсайтова підробка запиту;

EF – Entity Framework, фреймворк;

HTTP – HyperText Transfer Protocol, протокол передачі даних;

JSON – JavaScript Object Notation, запис об'єктів JavaScript;

JWT – JSON Web Token, стандарт токена доступу на основі JSON;

OAuth – Open Authorization, відкритий стандарт авторизації;

REST – Representational State Transfer, передача репрезентативного стану;

SAML – Security Assertion Markup Language, мова розмітки декларації безпеки;

UI – User Interface, інтерфейс користувача;

URL – Uniform Resource Locator, єдиний вказівник на ресурс;

XML – Extensible Markup Language, розширювана мова розмітки;

XSS – Cross-site scripting, міжсайтовий скриптинг;

ООП – об'єктно-орієнтоване програмування.

ВСТУП

Оцінка сучасного стану об’єкта розробки. Сучасні компанії, в тому числі ІТ-компанії мають тенденцію до зростання та накопичення майна і ресурсів. Зазвичай, невід'ємною складовою кожної компанії є великий оббіг техніки, обладнання та різних витрат, які необхідно інвентаризувати. Для цього зазвичай використовують звичайні журнали та excel-таблиці, що не дозволяє зручної перевірки та обліку майна компанії.

Проблеми автоматизації процесу інвентаризації майна компаній були досліджені в працях наступних авторів С. Бардаш [1], В. Жук [2], Н. Олійник [3] та інших.

Актуальність роботи та підстави для її виконання. У зв'язку зі зростаючою кількістю інвентарю в кожній компанії, з'являється потреба у зручному та функціональному програмному забезпеченні (ПЗ) для відстеження та керування наявним майном та ресурсами.

Згідно з пунктом 1 статті 10 розділу III Закону України "Про бухгалтерський облік та фінансову звітність в Україні" [4]: “Інвентаризація активів застосовується юридичними особами, які створені відповідно до законодавства України, незалежно від їх організаційно-правових форм та форм власності. Ця практика також поширена серед представництв іноземних суб'єктів господарської діяльності.”

Затвердженими Міністерством фінансів України “Положеннями про інвентаризацію активів та зобов’язань” встановлюють вимоги щодо методології, термінів, облікової документації, процедур і звітності, які пов'язані з інвентаризацією [5].

Таким чином, тема кваліфікаційної роботи є актуальною.

Мета й завдання роботи. Метою роботи є розробка сервісу призначеного для управління процесом інвентаризації майна та ресурсів компаній.

Для досягнення цієї мети в роботі було поставлено наступні завдання:

- дослідити ринок та попит на інвентаризацію ресурсів;
- розробити концепцію програми;

- спланувати та розподілити обов'язки;
- розробити програмне забезпечення;
- впровадити систему у користування клієнтом.

Колективна розробка. Система інвентаризації ресурсів компаній є колективним проєктом, реалізованим спільно зі студентом 4 курсу вибіркового блоку «Теорія та технологія програмування» освітньо-професійної програми «Інформатика» Любомиром Маєвським.

Роботу було поділено на дві частини, щоб кожен з розробників отримав рівноцінний пласт роботи та досвід від використання нових для нього технологій. При цьому, обидва розробники розробили архітектуру сервісу, Діні Формакідов необхідно було реалізувати CRUD, Synchronization, Retrieving, роботу з Google Workspace, а Любомиру Маєвському розробити NuGet пакети для роботи API, створити Api Gateway, налаштувати Docker, розгорнути сервіс на платформі Azure DevOps та реалізувати користувацький інтерфейс.

Об'єкт, предмет, методи й засоби розроблення. Об’єктом розроблення програмного засобу є процес відстеження та керування інвентарем компанії. Предметом роботи є вебзастосунок для управління процесом інвентаризації майна та ресурсів компаній. Основу для проєкту склав аналіз потреб сучасних компаній. Проєкт було розроблено на Windows 11. Для реалізації проєкту використано середовище розробки Visual Studio, мову програмування C# , фреймворк ASP.NET та API Google Workspace.

Можливі сфери застосування. Розроблена система може бути інтегрована в різні компанії, в першу чергу у сфері IT, адже сучасні компанії надають своїм працівникам багато техніки, за якою потрібно контролювати.

РОЗДІЛ 1.

ОГЛЯД НАЯВНИХ НА РИНКУ СИСТЕМ

1.1 Огляд конкурентних систем

Для порівняння систем було обрано ринок продуктів, які потенційно можуть зацікавити власника компанії або ресурс-менеджера.

З метою створення оцінки конкурентності генеруємо пошуковий запит, який буде стосуватися систем трекінгу інвентаризації. В таблиці 1 наведено чотири з найпопулярніших систем інвентаризації.

Таблиця 1 – Наявні системи інвентаризації

Система інвентаризації	**Переваги**	**Недоліки**
Asset Panda	● підтримка налаштування полів; ● інтегрується з багатьма популярними платформами; ● має мобільні застосунки.	● велика вартість; ● через свою гнучкість, Asset Panda може бути складним для використання без належного навчання [6].
Snipe-IT	● відкрите програмне забезпечення; ● має активну спільноту користувачів та розробників для підтримки.	● через свою відкритість, Snipe-IT вимагає від користувачів наявності технічних знань для встановлення та налаштування; ● обмежена функціональність.

ManageEngine AssetExplorer	● дозволяє легко інтегрувати інвентаризацію з іншими процесами IT; ● пропонує розширений набір функцій.	● вартість; ● комплексність.
Spiceworks	● безкоштовність; ● підтримка спільноти.	● обмежена функціональність; ● багато реклами.

Asset Panda – це хмарне програмне рішення, яке допомагає компаніям керувати своїми активами та відстежувати їх (див. рис. 1).

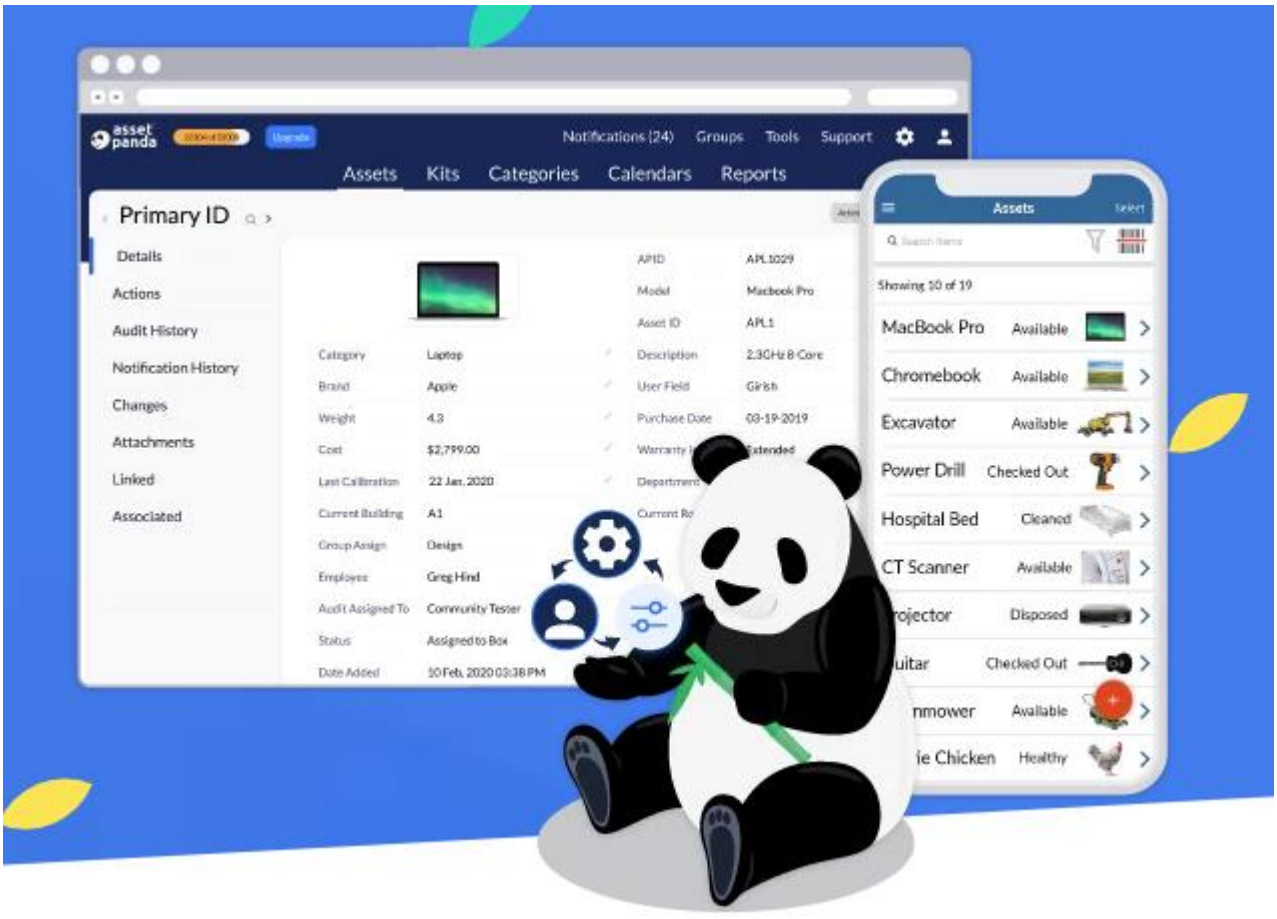


Рисунок 1 – Інтерфейс застосунку Asset Panda

Snipe-IT створено для управління IT-активами, щоб дозволити IT-відділам відстежувати, у кого який ноутбук, коли його було придбано, які ліцензії на програмне забезпечення та аксесуари доступні тощо. Це проєкт із безкоштовним відкритим кодом, побудований на Laravel. Проте за відгуками ймовірність рекомендації використання цієї системи є 61% [7].

ManageEngine AssetExplorer - це система управління активами IT, яка дозволяє організаціям управляти всіма своїми активами на одній платформі. Система допомагає в автоматизації інвентаризації, дозволяє відстежувати активи в реальному часі та має розширені можливості звітності.

Переваги ManageEngine AssetExplorer:

- ManageEngine AssetExplorer дозволяє легко інтегрувати систему інвентаризації з іншими IT-процесами, такими як служба підтримки, управління проектами, управління змінами тощо.
- ManageEngine AssetExplorer пропонує великий набір функцій, включаючи управління контрактами, ліцензіями, поставщиками, розподіл витрат тощо.

Недоліки ManageEngine AssetExplorer:

- Платна система, що може бути бар'єром для малого та середнього бізнесу.
- Система може виявитися складною для користувачів без належного навчання.

Spiceworks – це безкоштовне рішення для управління IT, яке допомагає організаціям відстежувати та управляти їх IT-активами. За допомогою Spiceworks користувачі можуть відстежувати інвентар, моніторити мережу, управляти службою підтримки, створювати звіти та ін.

Переваги Spiceworks:

- Spiceworks – безкоштовна система, що робить її доступною для всіх розмірів бізнесу.
- Spiceworks має активну спільноту користувачів, що може допомогти вирішувати проблеми та надавати поради.

Недоліки Spiceworks:

- Незважаючи на безкоштовність, Spiceworks не надає такий широкий набір функцій, як платні системи.
- Щоб підтримати безкоштовну модель, Spiceworks включає в себе рекламу, яка може бути нав'язливою для деяких користувачів.

1.2. Переваги проєкту Equipy

Іноді стандартні продукти не можуть задовольнити специфічні вимоги бізнесу, не пропонують достатньої інтеграції з іншими системами, або є занадто дорогими для малих та середніх підприємств.

На відміну від цього, система Equipy може бути налаштована для взаємодії з іншими бізнес-системами, може відповідати конкретним вимогам до інвентаризації, і, що досить важливо, надає повний контроль над безпекою даних.

Така система також надає можливість більшої масштабованості та гнучкості, дозволяючи легко внести зміни відповідно до зростання бізнесу або змін у потребах. Вона також надає можливість для індивідуального налаштування під конкретні потреби компанії та дозволяє швидко реагувати на можливі проблеми та розробляти нові функції.

У проєкті Equipy, замовником є організація, що виразила потребу в системі інвентаризації та контролю обладнання. Система спрямована на розв'язання конкретних задач замовника, які включають автоматизацію процесу інвентаризації обладнання, забезпечення перегляду й оновлення даних про співробітників, а також управління запитами на зміну чи замовлення нового обладнання.

Замовник бажає мати можливість відстежувати, як обладнання використовується співробітниками в реальному часі, що дозволить оптимізувати витрати та планувати придбання нового обладнання більш ефективно.

Ця система була розроблена з урахуванням специфічних вимог замовника та високих стандартів якості, що дозволить виконати проєкт вчасно і забезпечити високий рівень задоволеності замовника. Більш деталі вимоги описані у розділі 3.

РОЗДІЛ 2.

ОГЛЯД ВИКОРИСТАНИХ ТЕХНОЛОГІЙ

2.1. Обґрунтування використання технологій

Відповідно до вимог замовника, система Equipy є вебзастосунком, який має бути високопродуктивним, легко масштабованим та зручним для використання з будь-яким користувацьким інтерфейсом. Саме тому обираючи підхід до розробки треба якісно оцінити потреби. Основними стали такі критерії:

- інтегрованість в реальному світі;
- гнучкість архітектури;
- популярність технологій та їх довготривала підтримка;
- здатність технології пришвидшити процес розробки без шкоди для якості.

2.2. Мікросервісна архітектура

Розглянемо поняття мікросервісу.

Мікросервіси, які часто називають архітектурою мікросервісів – це архітектурний підхід, який передбачає поділ великих програм на менші функціональні одиниці, здатні функціонувати та спілкуватися незалежно [8].

Цей підхід виник у відповідь на обмеження монолітної архітектури. Оскільки моноліти – це великі контейнери, що містять усі програмні компоненти програми, вони сильно обмежені: негнучкі, ненадійні та часто розвиваються повільно.

Проте за допомогою мікросервісів кожен блок можна розгортати незалежно, але за потреби він може спілкуватися один з одним. Тепер можна досягти масштабованості, простоти та гнучкості, необхідних для створення складних програмних застосунків.

Кожен мікросервіс стосується певного аспекту та функції програми, наприклад, як в системі Equipy – робота з інформацією про співробітників, технікою, логування тощо. Разом ці мікросервіси утворюють єдину програму.

Залежно від конкретної проблеми кожна дія в системі адаптована до унікального набору рішень. За наявності додаткового коду, мікросервіси можуть розбиватися на більш дрібні сервіси. На рис. 2 представлено схему роботи мікросервісів:

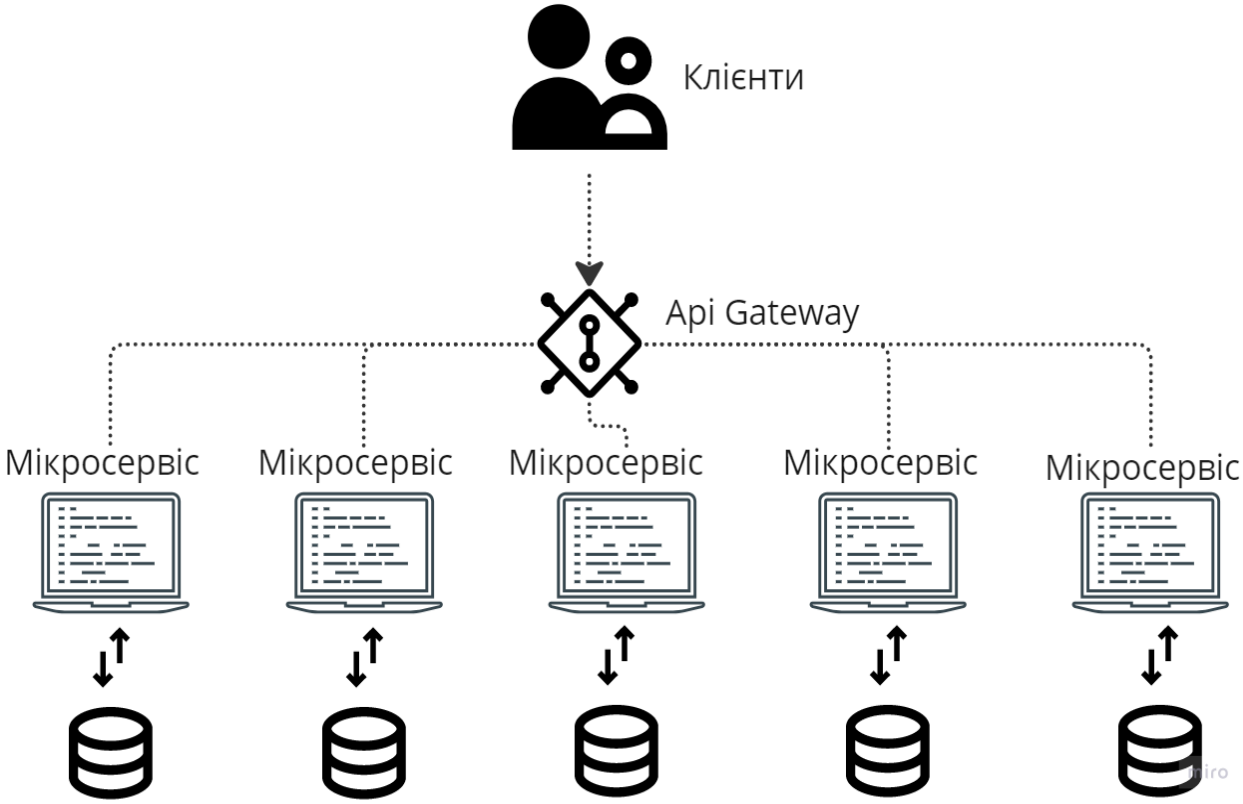


Рисунок 2 – Схема роботи мікросервісів

Однією з ключових переваг монолітної архітектури є те, що вона є автономною і незалежною від інших програм. Це забезпечує легке розгортання та розробку, спрощене тестування, легке налагодження та оптимізовану продуктивність [9].

Монолітна архітектура має свої недоліки, зокрема повільну швидкість розробки, проблеми з масштабованістю та той факт, що окремі помилки можуть вплинути на доступність усієї програми, порівняно з мікросервісною архітектурою.

З іншого боку, архітектура мікросервісів є швидшим, гнучким і більш масштабованим рішенням. Основною перевагою мікросервісів є те, що вони забезпечують безперервне розгортання та швидші цикли випуску. Це зручно в обслуговуванні та перевірці, а також забезпечує більшу гнучкість у технологічних параметрах.

Однак архітектура мікросервісів також може призвести до комплікованої розробки, збільшити витрати, а також до проблем з налагодженням через великі обсяги даних.

Ключові переваги такого підходу:

- менше зусиль щодо розробки: групи розробників можуть паралельно працювати над різними компонентами, щоб оновити наявні функції. Це значно полегшує масштабування незалежно від решти програми та вдосконалення програми;
- покращена масштабованість: мікросервіси самостійно запускають окремі сервіси, розроблені різними мовами чи технологіями; усі технологічні стеки сумісні, що дозволяє DevOps вибирати будь-який із найефективніших технологічних стеків, не побоюючись, чи добре вони працюватимуть разом.

  Ці невеликі служби працюють із відносно меншою інфраструктурою, ніж монолітні програми, вибираючи точну масштабованість вибраних компонентів відповідно до їхніх вимог;
- самостійне розгортання: кожен мікросервіс, що становить застосунок, повинен бути повним стеком. Це дає змогу розгортати мікросервіси незалежно в будь-якій точці.

  Архітектура мікросервісу є гнучкою, тому не потребує рішення усієї команди, щоб змінити програму шляхом додавання або зміни рядка коду або додавання чи виключення функцій;
- виділення помилок: у монолітних програмах відмова навіть невеликого компонента загальної програми може зробити її недоступною. У деяких випадках визначення помилки також може бути виснажливим.

За допомогою мікросервісів легко виділити компонент, що викликає проблему, оскільки вся програма розділена на окремі, повністю функціональні програмні блоки. У разі виникнення помилок інші непов'язані підрозділи продовжуватимуть працювати;

- інтеграція з різними технологічними стеками: завдяки мікросервісам розробники мають свободу вибирати стек технологій, який найкраще підходить для конкретного мікросервісу та його функцій.

  Замість того, щоб вибирати один стандартизований стек технологій, який охоплює всі функції програми.

2.3. Підхід Clean Architecture (гексагональна архітектура)

Підходом Clean Architecture називається методологія проєктування систем, для якої пріоритетним є забезпечення високого рівня незалежності між різними компонентами. Основою є розмежування системи на рівні домену (бізнес-логіки) та його залежності від внутрішньої інфраструктури та зовнішніх систем. Підхід визначає, як розміщувати код на різних рівнях та як вони мають взаємодіяти між собою, забезпечуючи чистоту та легкість розуміння коду.

Концепція чистої архітектури була визначена Робертом К. Мартіном [10]. У цій архітектурі системи можна розділити на два основні елементи: політики та деталі. Політика – це бізнес-правила та процедури, а деталі – елементи, необхідні для виконання політики. Саме з цього поділу чиста архітектура починає відрізнятися від інших архітектурних моделей. Архітектор повинен створити спосіб, за допомогою якого система розпізнає політики як основні елементи системи, а деталі – як несуттєві для політик.

У гексагональній архітектурі немає необхідності вибирати базу даних або фреймворк на початку розробки, оскільки всі ці деталі не заважають політикам і, отже, можуть бути змінені з часом. Структуру архітектури можна описати схемою, представленою на рис. 3:

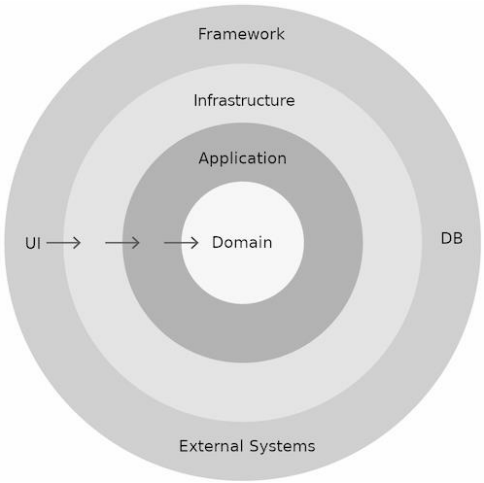


Рисунок 3 – Схема Clean Architecture

Ця архітектура поділяється на концентричні кола, кожне з яких має різний рівень абстракції. Від центру до периферії, існують такі складові:

1. Сутності: це об'єкти, які відображають основні бізнес-правила. Вони є незалежними від деталей реалізації.
2. Use Cases: це додатковий рівень, який виконує конкретні дії, які вимагаються від системи, такі як створення нового об'єкта або отримання даних.
3. Адаптери: цей рівень відповідає за адаптацію інтерфейсів між рівнями "Використання випадків" та "Фреймворки та драйвери". Адаптери перетворюють дані в формат, який може обробити рівень "Використання випадків" і навпаки.
4. Фреймворки та драйвери: це зовнішній рівень, який включає код, який взаємодіє з деталями реалізації, такими як база даних, UI, вебсервери тощо.

Важливим принципом "Чистої Архітектури" є "Принцип залежності", згідно з яким зовнішні рівні можуть залежати від внутрішніх рівнів, але ніколи не навпаки. Тобто, код, який знаходиться у внутрішніх рівнях, ніколи не залежить від коду, розташованого в зовнішніх рівнях. Це означає, що бізнес-правила вашої програми не залежать від деталей реалізації, таких як взаємодія

з базою даних або UI. Цей принцип дозволяє вам легко змінювати деталі реалізації без впливу на основну бізнес-логіку програми.

При використанні "Чистої Архітектури", кожен прошарок архітектури відповідає за конкретний аспект програми. Прошарок сутностей відповідає за бізнес-правила, прошарок використання випадків відповідає за дії, які можна виконати в системі, прошарок адаптерів відповідає за перетворення даних між прошарками, а прошарок фреймворків та драйверів відповідає за деталі реалізації.

Такий підхід до організації коду призводить до ряду переваг. Розробники можуть легко розробляти, тестувати та підтримувати окремі частини програми незалежно одна від одної. Бізнес-логіка ізольована від деталей реалізації, тому її можна легко змінювати або використовувати повторно. Крім того, програма стає більш стійкою до змін у вимогах чи технологіях.

Основними перевагами даного підходу є:

- зменшені залежності: розділення програмного забезпечення на різні рівні зменшує залежності між системними компонентами, полегшуючи тестування та розгортання;
- розширюваність: легке додавання до системи нового функціоналу та змінювати її;
- легкість тестування: тестування окремих компонентів системи без необхідності запускати весь застосунок;
- масштабованість: розмежування на різні рівні дозволяє масштабувати окремі компоненти системи незалежно один від одного;
- зменшення витрат на підтримку: кожен компонент системи може бути модифікований або замінений окремо, без впливу на інші компоненти.

2.4. Мова C#

C# є мовою програмування з безпечною системою типізації для платформи .NET, яка є об'єктно-орієнтованою. Вона була розроблена Андерсоном Хейлсбергом, Скотом Вілтамутом та Пітером Гольде від імені Microsoft Research, що належить компанії Microsoft [11].

C# використовується для розробки різноманітних застосунків, включаючи мобільні застосунки, вебзастосунки, десктопні застосунки, ігри та бізнес-застосунки.

Основними особливостями є:

- сильна типізація: усі змінні повинні бути оголошені заздалегідь, щоб компілятор міг перевірити їх типи;
- об'єктно-орієнтована парадигма: C# підтримує класи, об'єкти, наслідування, поліморфізм та інші основні концепції ООП;
- можливості збирання сміття: C# має вбудовану систему збирання сміття, яка автоматично видаляє непотрібні об'єкти з пам'яті;
- виняткова обробка: C# має вбудовану систему винятків, яка дозволяє обробляти помилки та непередбачувані ситуації;
- багатопотоковість: C# має вбудовану підтримку багатопотоковість, що дозволяє створювати багатопоточні застосунки;
- висока продуктивність: C# має оптимізований компілятор, що дозволяє досягати високої продуктивності;
- кросплатформність: з виходом .NET Core, C# став кросплатформним, що дозволяє розробляти застосунки на різних операційних системах.

Дана мова програмування ідеально підійшла для виконання проєкту, адже вона відповідає по усім необхідним критеріям: підтримка мікросервісної

архітектури, інтеграція з Identity Server, інтеграція з Google Workspace, забезпечення безпеки та продуктивності.

2.5. Фреймворк ASP.NET Web API

ASP.NET Web API – це фреймворк, що розроблений корпорацією Microsoft для створення і розгортання RESTful вебсервісів на платформі .NET. Він дозволяє розробникам створювати легкі та прості вебсервіси, які можуть бути використані для обміну даними між клієнтами та серверами [12].

Основні особливості ASP.NET Web API:

- підтримка HTTP-протоколу: ASP.NET Web API підтримує HTTP-протокол, що дозволяє легко створювати RESTful вебсервіси;
- маршрутизація: фреймворк має вбудований механізм маршрутизації, що дозволяє прив'язувати методи контролерів до конкретних URL-адрес;
- форматування даних: ASP.NET Web API дозволяє форматувати дані в різних форматах, включаючи XML, JSON, CSV та інші;
- підтримка кросплатформності: фреймворк підтримує розробку вебсервісів для різних операційних систем, включаючи Windows, macOS та Linux;
- підтримка тестування: ASP.NET Web API має вбудовану підтримку тестування, що дозволяє розробникам перевіряти правильність роботи вебсервісу;
- захист від атак: фреймворк має вбудовані механізми захисту від атак, такі як XSS, CSRF та інші.

2.6. Система авторизації Identity Server

Identity Server — це безкоштовний сервер автентифікації та авторизації з відкритим кодом, розроблений для платформи .NET. Він дозволяє створити централізовану систему авторизації та контролю доступу для різних програм і служб, що працюють на платформі .NET [13].

Identity Server підтримує різні протоколи автентифікації та авторизації, такі як OpenID Connect, OAuth 2.0, SAML 2.0 і WS-Federation. Це дозволяє використовувати його для різних типів програм, включаючи вебпрограми, мобільні програми та API. Ключові переваги сервера ідентифікації:

- простота використання та налаштування: Identity Server простий у використанні та може бути налаштований відповідно до ваших потреб;
- підтримка декількох протоколів: Identity Server підтримує кілька протоколів автентифікації та авторизації, що робить його корисним для багатьох різних типів програм;
- безпека: Identity Server має багато вбудованих механізмів безпеки для забезпечення безпеки авторизації та доступу до програм;
- розширюваність: Identity Server можна легко розширити для задоволення конкретних потреб замовника.

РОЗДІЛ 3.

ПРИЗНАЧЕННЯ, ВИМОГИ ТА РЕАЛІЗАЦІЯ

3.1 Загальний опис системи Equipy

Equipy – це система управління технікою та майном компанії, яка дозволяє відстежувати рух майна, його стан та використання. Система дозволяє також відстежувати співробітників, які використовують цю техніку та вести їх облік.

3.2 Основні функції системи Equipy

Основні функції системи Equipy, розробленої в рамках даного проєкту, включають:

- Інвентаризація техніки: система Equipy дозволяє створювати базу даних з переліком техніки та майна компанії. Кожен елемент має свій унікальний ідентифікатор та опис, включаючи фотографії, серійні номери та іншу інформацію.
- Відстеження руху техніки: система дозволяє відстежувати місцезнаходження техніки та майна компанії в режимі реального часу, що забезпечує контроль за рухом майна та його використанням.
- Облік співробітників: система дозволяє вести облік співробітників, які використовують техніку та майно компанії, з урахуванням їх ролі та повноважень.
- Контроль за обслуговуванням техніки: система дозволяє вести облік проведеного обслуговування техніки та майна компанії, що дозволяє забезпечувати їх надійність та продовжувати термін їх служби.
- Аналітика: система Equipy має вбудовані засоби аналітики, які дозволяють аналізувати дані про використання техніки та майна компанії, контролювати витрати на їх утримання та виявляти проблемні зони.

Сценарій взаємодії користувача з системою зображений на use-case діаграмі наведеній в додатку Б.

3.3 Цілі створення системи

Найголовнішою метою створення Equipy – це полегшення моніторингу техніки у великих компаніях. Наразі ситуація на ринку погана, адже наявні системи не завжди виконують необхідні функції та тому ресурс-менеджери вимушені користуватись таблицями (наприклад MS Excel), що призводить до неефективного аналізу, втрати ресурсів тощо.

Важливим є забезпечення ефективного управління використанням техніки та майна, що є ключовими ресурсами для бізнесу. Забезпечення контролю та відстеження техніки та майна зменшує ризики втрати або крадіжки майна та дозволяє швидко реагувати на непередбачувані ситуації. Збільшення продуктивності та зниження затрат є ключовими цілями більшості компаній. Equipy дозволяє забезпечити оптимальне використання техніки та майна, що зменшує витрати на їх обслуговування та підтримку.

Програма дозволяє вести облік проведеного обслуговування техніки та майна компанії, що забезпечує їх надійність та продовжує термін їх служби, що дозволяє зменшити кількість аварій та витрат на ремонт.

Ще одним важливим фактором є підвищення безпеки працівників: система Equipy дозволяє відстежувати, які співробітники використовують техніку компанії, що забезпечує безпеку їх праці та зменшує ризик виникнення нещасних випадків на робочому місці.

3.4 Користувачі

Продукт має підтримку різних груп користувачів, які мають різні права доступу для роботи з системою.

- Керівництво компанії: система дозволяє керівництву компанії вести облік техніки та майна, забезпечуючи контроль за їх використанням та зниженням ризиків втрати або крадіжки. Окрім моніторингу витрат вони мають право погодитись, заборонити або перенести наступну покупку або ремонт.
- Менеджери технічного відділу: програма Equipy допомагає менеджерам технічного відділу забезпечити ефективне використання техніки та майна. Вони мають можливість робити запити на покупку нової техніки або заміни старої за умови погодження керівництва.

3.5 Реалізація бекенд частини проєкту

3.5.1. Розробка мікросервісів

Проаналізувавши потреби функціонала, було почато розробку окремих мікросервісів, які виконують конкретну функціональність. Систему було розподілено на такі мікросервіси:

- Employee Service:

даний мікросервіс відповідає за керування співробітниками. Користувач може Додавати, Редагувати, Видаляти та Переглядати співробітників.
Головною функцією цього мікросервісу є інтеграція з Google Workspace.
Зазвичай компанії користуються великими системами для проведення щоденних комунікацій, таких як Google Workspace, MS Teams тощо.

Лідером ринку є Google Workspace, тому створено можливість імпортувати усіх співробітників звідти. Це дозволяє швидко почати користування системою, без зайвого внесення даних.

- Equipment Service:

даний мікросервіс відповідає за керування обладнанням. Користувач може Додавати, Редагувати, Видаляти та Переглядати обладнання.

Головною функцією є призначення техніки на співробітника, або помічати, що вона наразі не використовується ніким.

Асортимент обладнання в системі необмежений – від мишок та комп'ютерів до онлайн курсів або холодильників для офісу.

Equipment Service може обробляти запити від співробітників на заміну або замовлення нового обладнання. Він може відстежувати статус кожного запиту та надавати оновлення співробітникам.

- Auth Service:

даний мікросервіс відповідає за керування користувачами. Дозволяє створювати нових користувачів, надаючи їм відповідні права доступу.
Головною функцією цього мікросервісу є авторизація та автентифікація користувачів у системі, із визначенням прав доступу до ресурсів API.

- Api Gateway Service:

даний мікросервіс є шлюзом для всіх запитів, які надходять у систему. За необхідності створює додаткові копії мікросервісів, щоб регулювати балансування запитів на ресурси API. Також надає зовнішнім розробникам API документацію до усіх наявних ресурсів.

- RabbitMQ Service:

даний мікросервіс забезпечує стабільну та безпечну комунікацію між мікросервісами за допомогою черг та повідомлень. Кожен мікросервіс має змогу надіслати широкоформатне транслююче повідомлення, яке може бути отримане будь-якою відповідною чергою в іншому мікросервісі.

- Notification Service:

даний мікросервіс використовується для надсилання сповіщень, повідомлень або інформаційних повідомлень користувачам. Наприклад, при

створенні нового запиту на техніку інші пов'язані менеджери будуть повідомлені.

- Email Service:

  цей сервіс використовується для надсилання та отримання електронних листів між користувачами. Окрім нотифікацій, менеджери можуть отримувати лист про те, що на них був назначений запит. Це дозволяє позбавитись від постійного моніторингу системи на наявні зміни.

3.5.2. Розподіл на гексагональну архітектуру

Через те, що в роботі було обрано підхід Clean Architecture [10], кожен мікросервіс було розбито на такі рівні:

- Application

  Рівень, який містить всю бізнес-логіку застосунку. Тут знаходяться розширення, валідатори, які взаємодіють з іншими рівнями для виконання бізнес-логіки, а також сервіси, які використовуються в межах застосунка. Application Layer не залежить від інших рівнів і може бути тестований окремо.

- Domain

  Рівень, який містить універсальну бізнес-логіку, яка не залежить від фреймворків, бібліотек та інших залежностей. Тут знаходяться моделі домену. Рівень Domain повинен бути незалежним від будь-якої конкретної реалізації.

- Infrastructure

  Прошарок, який містить всі залежності, які потрібні для взаємодії з зовнішніми сервісами та ресурсами, такими як бази даних, логування та інше. Тут знаходяться конкретні реалізації сервісів. Цей прошарок повинен бути незалежним від реалізації бізнес-правил.

- WebApi

  Прошарок, який містить логіку для взаємодії з користувачем через API. Тут

знаходяться контролери, які взаємодіють з іншими компонентами застосунка для обробки запитів та відповідей. WebApi залежить від інших рівнів для виконання бізнес-логіки.

Розглянемо структуру на прикладі Equipment Service (див. рис. 4):

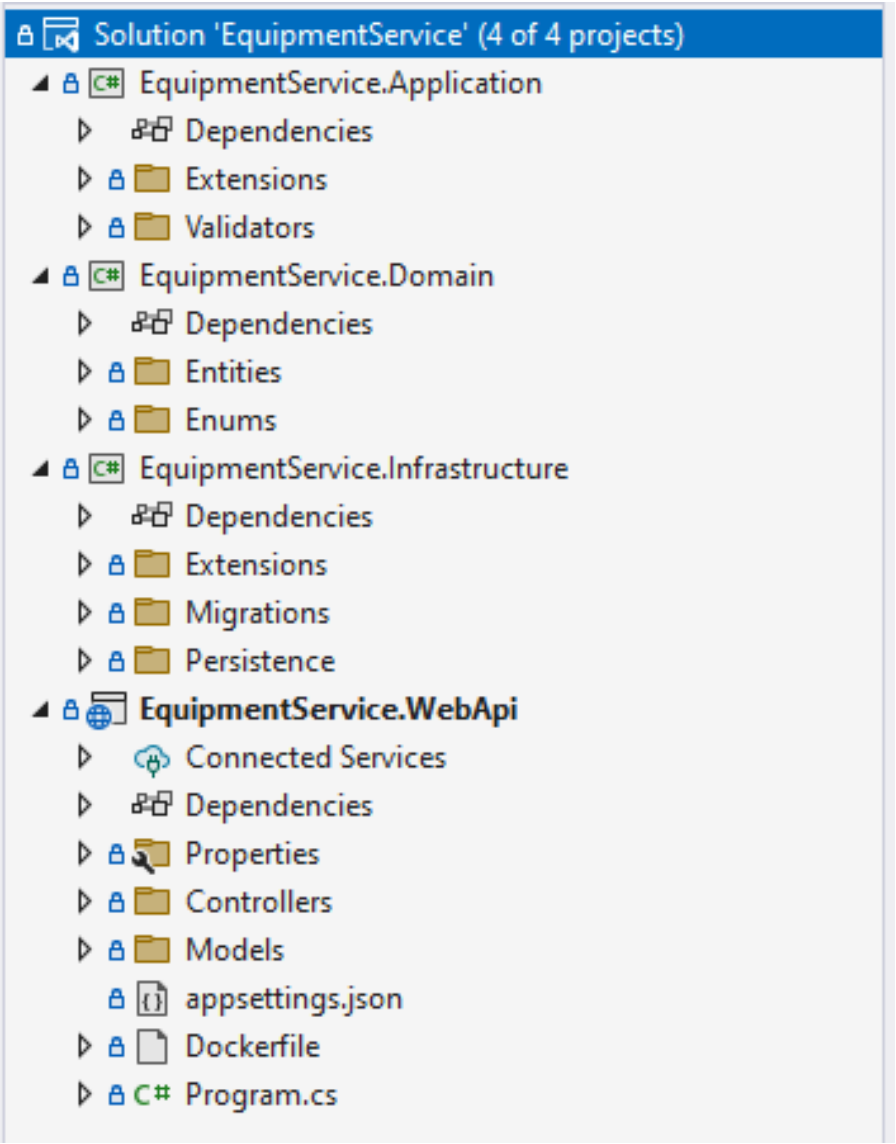


Рисунок 4 – Структура Equipment Service

3.5.3. Інтеграція з Google Workspace

До мікросервісу Employee було підключено логіку використання бази співробітників та інформацію про них з системи Google Workspace.

Google Workspace – це платформа від Google, яка містить набір інструментів для спільної роботи та комунікації в бізнесі та освіті. Google Workspace раніше називався G Suite, і включає такі інструменти, як Gmail, Google Drive, Google Docs, Google Sheets, Google Slides, Google Meet, Google Forms та інші.

Для реалізації етапу розробки інтеграції було придбано домен для відтворення роботи з Google Workspace (див. рис. 5).

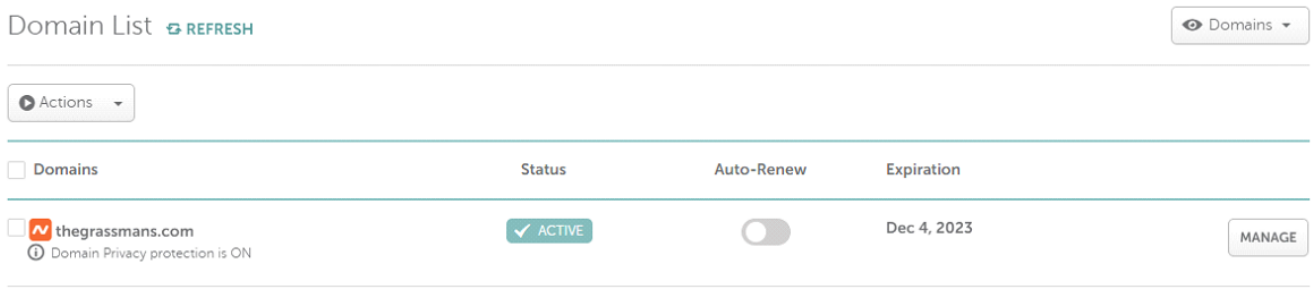


Рисунок 5 – Власний домен

На цей домен зареєстровано Google Workspace з тестовими користувачами (див. рис. 6).

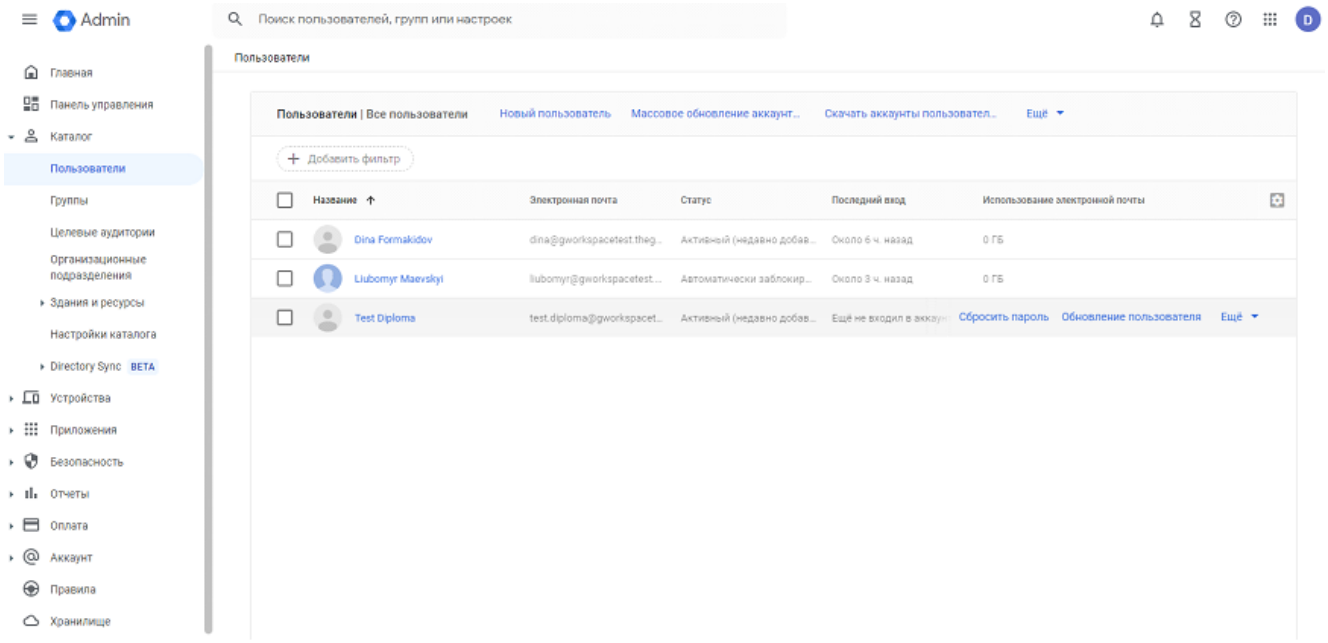


Рисунок 6 – Реєстрація Google Workspace з тестовими користувачами

Для використання API було створено проєкт на Google Cloud Console із забезпеченням необхідних ключів (див. рис. 7).

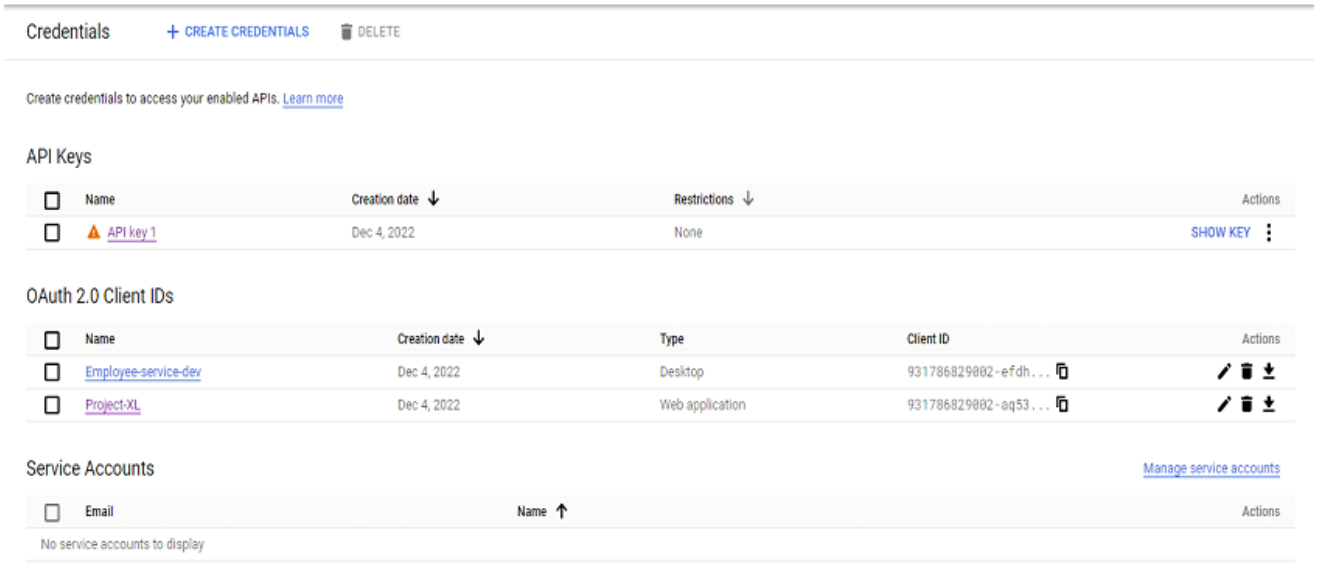


Рисунок 7 – Проєкт на Google Cloud Console

Після налаштування підключень йде процес отримання користувачів.

Створено інтерфейс IPeopleApiService та реалізацію до нього (див. рис. 8).

```
internal class PeopleApiService : IPeopleApiService
{
    // If modifying these scopes, delete your previously saved credentials
    // at ~/.credentials/people-dotnet-quickstart.json
    private static string[] Scopes = { DirectoryService.Scope.AdminDirectoryUser };

    private static ClientSecrets secrets = new ClientSecrets()
    {
        ClientId = "931786829002-efdh2ngc727pi0cq24ohtmha4cr7q66b.apps.googleusercontent.com",
        ClientSecret = "GOCSPX-8Yz_Oy8msdupZ_NNbósi-GV6mSsj"
    };

    public async Task<IQueryable<GoogleEmployee>> GetPeopleAsync()
    {
        UserCredential credential;

        string credPath = Environment.GetFolderPath(
            Environment.SpecialFolder.Personal);
        credPath = Path.Combine(credPath, ".credentials/people-dotnet-quickstart");

        credential = await GoogleWebAuthorizationBroker.AuthorizeAsync(
            secrets,
            Scopes,
            "ó",
            CancellationToken.None,
            new FileDataStore(credPath, true));

        // Create Drive API service.
        var service = new DirectoryService(new BaseClientService.Initializer()
        {
            HttpClientInitializer = credential
        });

        var request = service.Users.List();
        request.Domain = "thegrassmans.com";
        var results = request.Execute();
        return results.UsersValue.Adapt<IQueryable<GoogleEmployee>>();
    }
}
```

Рисунок 8 – Реалізація роботи з Google API

У цій реалізації створюється підключення до Google Workspace завдяки обліковим даним, які були отримані у Google Cloud Console. Після чого вказано домен та використано бібліотеку Google.Apis. Google.Apis – це набір бібліотек, які надають програмістам можливість легко та зручно взаємодіяти з багатьма сервісами Google API. Ці бібліотеки надають доступ до різних сервісів, таких як Gmail, Google Drive, Google Calendar, Google Maps та інших, та дозволяють отримувати та обробляти дані, що пов'язані з цими сервісами.

3.5.4. Розробка авторизації та автентифікації

Першим кроком до розробки Auth Service є аналіз вимог та ролей користувачів. Система пропонує наступні ролі (див. табл. 2):

Таблиця 2 – Ролі в системі

Роль	**Можливості**	**Рівень ієрархії**
Адміністратор	● Бачити сторінку аналітики та статистики ● Доступ до Employee ● Доступ до Equipment ● Обробляти нові запити та схвалювати їх	Вищий
Менеджер	● Доступ до Employee ● Доступ до Equipment ● Робити запит на покращення або заміну техніки будь-якому співробітнику ● Схвалювати запити, до яких менеджер має доступ	Середній
Користувач	● Бачити свою техніку ● Робити запит на покращення або заміну техніки	Нижчий

Загальну структуру роботи Identity Server можна побачити на рис. 9:

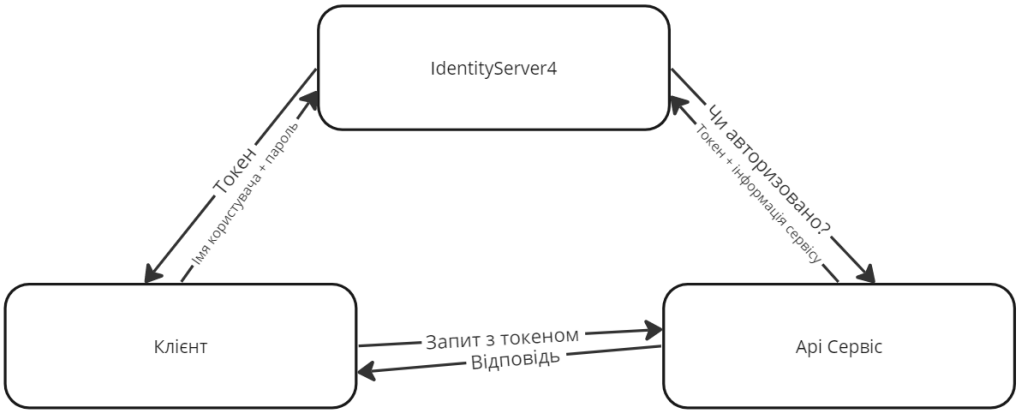


Рисунок 9 – Структура роботи Identity Server

Для початку потрібно встановити Identity Server як NuGet пакет. Для цього використовується команда Install-Package IdentityServer4.

Після цього було додано налаштування у клас Program (див. рис. 10).

```
builder.Services.AddIdentityServer()
        .AddInMemoryClients(Config.Clients)
        .AddInMemoryIdentityResources(Config.IdentityResources)
        .AddInMemoryApiScopes(Config.ApiScopes)
        .AddTestUsers(Config.TestUsers)
        .AddDeveloperSigningCredential();

builder.Services.AddControllersWithViews();


app.UseIdentityServer();
```

Рисунок 10 – Налаштування Identity Server

У цьому коді, AddIdentityServer() метод додає Identity Server до DI контейнеру. AddInMemoryClients(), AddInMemoryIdentityResources() та AddInMemoryApiScopes() методи налаштовують Identity Server для використання тестових даних.

З налаштованим Identity Server, тепер можна використовувати його для автентифікації. Для цього, потрібно додати контролер, який оброблятиме запити автентифікації.

Найголовнішими методами є Login та Register, які наведені нижче (див. рис. 11):

```
public virtual async Task<bool> Register(UserCreateModel model)
{
    User user = model.Adapt<User>();
    IdentityResult res = await _userManager.CreateAsync(user, model.Password);
    if (!res.Succeeded)
    {
        throw new Exception(res.Errors.FirstOrDefault()?.Description);
    }

    var encodedToken = HttpUtility.UrlEncode(await _userManager.GenerateEmailConfirmationTokenAsync(user));
    var encodedEmail = HttpUtility.UrlEncode(user.Email);

    string host = _httpContextAccessor.HttpContext.Request.Host.Value;

    var confirmationlink = "https://" + host + "/api/Auth/ConfirmEmail?email=" + encodedEmail + "&token=" + encodedToken;

    await _emailSender.SendEmailAsync(user.Email, "Confirm Email", "Click on the link to confirm email: " + confirmationlink);
    IdentityResult resAdd = await _userManager.AddToRoleAsync(user, "User");
    if (!res.Succeeded)
    {
        throw new Exception("Unable to assign user to role 'User'");
    }
    return true;
}

public virtual async Task<TokenPairModel> Login(LoginModel loginModel)
{
    TokenPairModel tokenPair = new TokenPairModel();
    User user = await _userManager.FindByEmailAsync(loginModel.Email);
    if (user == null)
    {
        throw new Exception("The User with such email doesn't exist");
    }
    if (!await _userManager.CheckPasswordAsync(user, loginModel.Password))
    {
        throw new Exception("Wrong password");
    }
    if (!await _userManager.IsEmailConfirmedAsync(user))
    {
        throw new Exception("Email is not confirmed");
    }
    string accessToken = GenerateJwtToken(user);

    var refreshToken = await _userManager.GenerateUserTokenAsync(user, "Equipy", "RefreshToken");
    await _userManager.SetAuthenticationTokenAsync(user, "Equipy", "RefreshToken", refreshToken);
    tokenPair.RefreshToken = refreshToken;
    tokenPair.AccessToken = accessToken;

    return tokenPair;
}
```

Рисунок 11 – Реалізація логіну та реєстрації

Автентифікація та авторизація користувачів дуже важливі під час розробки вебпроєктів. Було використано різні методи захисту програми від неавторизованих осіб і надано доступ до неї лише авторизованим користувачам. Одним із таких рішень є використання токенів.

На цей час існують деякі галузеві стандарти. Розглянемо використання JSON Web Token (JWT).

JWT – галузевий стандарт RFC7519 [14]. Він використовується для автентифікації та авторизації користувачів у системах, які спілкуються одна з одною. JWT можна використовувати для багатьох проблем, таких як автентифікація користувачів, безпека вебслужб та інформаційна безпека. JWT є дуже популярним і бажаним методом захисту ресурсів. Якщо потрібно отримати доступ до захищеного ресурсу, перше, що потрібно зробити, це отримати токен.

Токен, створений за допомогою JWT, складається з 3 основних частин, закодованих за допомогою Base64. Це частини Header (Заголовок), Payload (Data), Signature. Якщо звернути увагу на приклад токена на діаграмі нижче, то у бланку токена є 3 поля, розділені крапками (див. рис. 12).

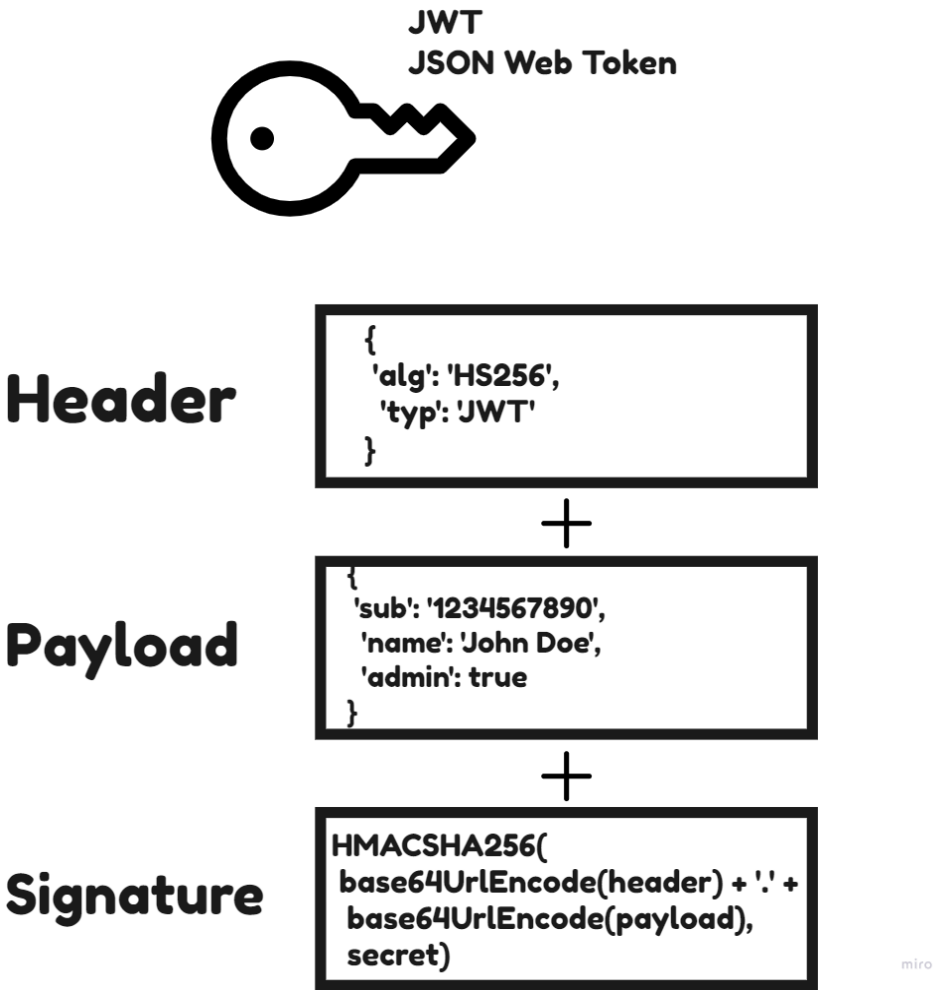


Рисунок 12 – Схема структури JWT

- Header: ця частина для використання у JWT написана у форматі JSON і складається з 2 полів. Це тип токена та назва алгоритму, який буде використовуватися для підпису.
- Payload:цей розділ включає claims. Дані, які зберігаються в цьому розділі, є унікальними для клієнта токена і сервера. Ця інформація про claims також забезпечує цю унікальність.
- Signature: ця частина є останньою частиною токена. Для створення цієї частини потрібні Header, Payload та секретний ключ. З частиною підпису гарантується цілісність даних.

3.5.5 Розробка мікросервісів Employee та Equipment Service

Employee Service.

Розглянемо наповнення сервісу за структурою.

- Application. Містить в собі абстракції для інтеграції з Google Workspace та валідатори на основі FluentValidation. FluentValidation — популярна бібліотека перевірки для програм .NET. Він забезпечує зручний інтерфейс для визначення правил перевірки та застосування їх до об’єктів або моделей.

За допомогою FluentValidation можна легко визначити логіку перевірки властивостей об’єктів за допомогою вільного синтаксису. Він підтримує широкий спектр правил перевірки, таких як обов’язкові поля, обмеження довжини рядка, регулярні вирази, правила порівняння та налаштована логіка перевірки.

- Domain. Складається з основних моделей сервісу. Має наступні класи, що представлені в табл. 3:

Таблиця 3 – Класи рівня Domain

Клас	Поля	Опис

BaseEntity	Guid Id	Є базовим класом, від якого усі наслідуються
Address	Guid TenantId string? Country string? City string? Address1 string? Address2 string? StateOrRegion string? Zipcode ICollection<Employee> Employees DateTime CreatedAt DateTime ModifiedAt	Адреса співробітника
Employee	Guid TenantId Guid? AddressId Guid? PositionTypeId string? FullName string? Email string? PhoneNumber DateTime CreatedAt DateTime ModifiedAt	Співробітники
PositionType	Guid TenantId string? Name ICollection<Employee> Employees DateTime CreatedAt DateTime ModifiedAt	Позиція співробітника в компанії

Кожна сутність наслідується від інтерфейсів ITenantDependentEntity, IAuditableEntity. Ця логіка знаходиться в індивідуально розроблених пакетах SharedPackages.

В розробленій системі використовується підхід Multitenancy. Це архітектура, у якій один екземпляр програмного забезпечення обслуговує кількох клієнтів. Кожен клієнт називається орендарем. Орендарям можна надати можливість налаштувати деякі частини програми, наприклад колір інтерфейсу користувача або бізнес-правила, але вони не можуть налаштувати код програми. В програмі Equipy було обрано вид Multitenancy під назвою Single application, single database.

У цій моделі програма розроблена для роботи з кількома клієнтами, тобто вона може диференціювати та ізолювати дані, що належать різним орендарям. Кожен орендар зазвичай має власний набір користувачів, конфігурацій і даних, але всі вони співіснують в одному екземплярі програми та базі даних. Отже, кожен екземпляр має свій, унікальний для клієнта, TenantId.

IAuditableEntity це концепція, яка використовується в розробці програмного забезпечення для відстеження та запису змін, внесених до сутності або об’єкта в системі. Зазвичай він використовується для ведення контрольного логу та збору інформації про те, хто вніс зміни та коли їх було внесено. Саме тому кожен клас містить два поля - CreatedAt, ModifiedAt.

- Infrastructure. Містить в собі конфігурації та розширення, реалізацію роботи з Google Workspace, а також роботу з базою даних та міграції.

Кожна модель бази даних має конфігурації та обмеження. Наприклад, Employee (див. рис. 13):

```
internal class EmployeeConfiguration : IEntityTypeConfiguration<Employee>
{
    0 references
    public void Configure(EntityTypeBuilder<Employee> builder)
    {
        builder.HasKey(x => x.Id);
        builder.HasIndex(x => new { x.TenantId, x.Email }).IsUnique();

        builder.Property(x => x.FullName).IsRequired();
        builder.Property(x => x.Email).IsRequired();

        builder.HasOne(x => x.Address)
            .WithMany(x => x.Employees)
            .HasForeignKey(x => x.AddressId)
            .OnDelete(DeleteBehavior.Restrict);
    }
}
```

Рисунок 13 – Конфігурація моделі Employee

Таким чином налаштовано зв'язки між сутностями, визначено обмеження та гарантована правильна робота орендарів.

- WebApi. Містить в собі контролери та моделі, які використовуються в API.

Кожен контролер містить CRUD. CRUD (Create, Read, Update, Delete) контролери використовуються для обробки HTTP-запитів, пов'язаних зі створенням, отриманням, оновленням та видаленням ресурсів (див. рис. 14).

```
public class AddressController : ControllerBase
{
    private readonly IEntityManager<Address> _entityManager;

    0 references
    public AddressController(IEntityManager<Address> entityManager)
        => _entityManager = entityManager;

    [HttpGet("{pageIndex=10}/{pageSize=0}")]
    [ProducesResponseType(typeof(HttpResponseResult<IPagedList<AddressViewModel>>), StatusCodes.Status200OK)]
    [ProducesResponseType(typeof(HttpResponseResult<IPagedList<AddressViewModel>>), StatusCodes.Status400BadRequest)]
    [ProducesResponseType(typeof(HttpResponseResult<IPagedList<AddressViewModel>>), StatusCodes.Status500InternalServerError)]
    0 references
    public async Task<IActionResult> Get([FromRoute] int pageIndex, [FromRoute] int pageSize)
    {
        var httpResponseResult = new HttpResponseResult<IPagedList<AddressViewModel>>();

        if (pageIndex < 0 || pageSize < 1)
        {
            httpResponseResult.AddBadRequestErrorMessage("Page index must be >= 0 and page size >= 1");
            return StatusCode(httpResponseResult.StatusCode, httpResponseResult);
        }

        httpResponseResult.Data = await _entityManager
            .GetAll()
            .ProjectToType<AddressViewModel>()
            .ToPagedListAsync(pageIndex, pageSize);

        return StatusCode(httpResponseResult.StatusCode, httpResponseResult);
    }
```

Рисунок 14 – Контролер рівня WebApi

У кожному контролері використовується індивідуальна обробка відповідей за допомогою класу HttpResponseResult, що дозволяє більш інформативно та якісно опрацьовувати запити (див. рис. 3.5.11).

Крім цього, для методу GetAll використовується пагінація IPagedList. В параметри передаємо pageIndex та pageSize. Це дозволяє розподілити результати запиту на сторінки та забезпечити зручний спосіб навігації через ці сторінки.

Логіка роботи з моделями проходить через IEntityManager. Він дозволяє робити Створення, зчитування, оновлення та видалення (CRUD) сутностей, керувати транзакціями, а також виконувати запити до бази даних.

Equipment Service.

Структура даного сервісу схожа на Employee Service, проте має свої відмінності. Розглянемо рівень Domain та його моделі (див. табл. 4).

Таблиця 4 – Класи Equipment Service

Клас	**Поля**
BaseEntity	Guid Id DateTime CreatedAt DateTime ModifiedAt
Employee	Guid TenantId string? FirstName string? LastName ICollection<Equipment> Equipments
EquipmentTag	Guid TenantId Guid? EquipmentId Guid? TagId
Tag	Guid TenantId string? Name ICollection<EquipmentTag> EquipmentTags
Equipment	Guid TenantId string? Name string? LotNumber EquipmentStatus Status Guid? EmployeeId

	ICollection<EquipmentTag> EquipmentTags
Laptop : Equipment	string? GPU string? RAMType string? RAMQuantity
Mouse : Equipment	bool IsWireless bool HasBluetooth

Наразі система має два класи, що наслідуються від Equipment – Laptop та Mouse. Це найголовніші класи для використання програми, проте завдяки гнучкій архітектурі, можна додавати варіації техніки на запит користувача.

Важливою частиною даного сервісу є процес запиту нової техніки. Потік процесу можна побачити на діаграмі в додатку А.

РОЗДІЛ 4.

ВИКОРИСТАННЯ СИСТЕМИ EQUIPY

4.1. Робота зі Swagger

Для роботи з бекенд частиною використовується Swagger. Swagger – це набір інструментів, створений компанією SmartBear, щоб допомогти у процесі розробки API та документації.

OpenAPI – це стандартна специфікація Restful API, а Swagger складається з інструментів, які використовуються для реалізації OpenAPI.

Специфікація OpenAPI (раніше відома як специфікація Swagger) – це формат опису API для REST API. Файл, сумісний зі специфікацією OpenAPI, дозволяє описати повний REST API. Зазвичай він пишеться на YAML або у форматі JSON[15].

У даному проєкті Swagger використовується для автоматизації процесу документування API, що спрощує процес розробки, тестування та інтеграції між різними мікросервісами. Кожен мікросервіс (EmployeeService, EquipmentService, та AuthService) має власний Swagger UI, що дає можливість переглядати та тестувати всі доступні кінцеві точки.

Swagger UI автоматично генерує інтерактивну документацію, яка відображає всі доступні кінцеві точки, їх параметри, формати відповідей, статуси HTTP відповідей, та іншу корисну інформацію. Це значно спрощує процес інтеграції мікросервісів, адже розробники можуть легко дізнатися про всі доступні API без необхідності читати код.

Крім того, Swagger дозволяє проводити тестування API прямо у браузері. Розробники можуть відправляти запити до API та переглядати відповіді в реальному часі, що спрощує процес тестування та виявлення помилок.

Використання Swagger у даному проєкті значно спрощує розробку та інтеграцію мікросервісів, покращує якість документації та забезпечує ефективне тестування API (див. рис. 15).

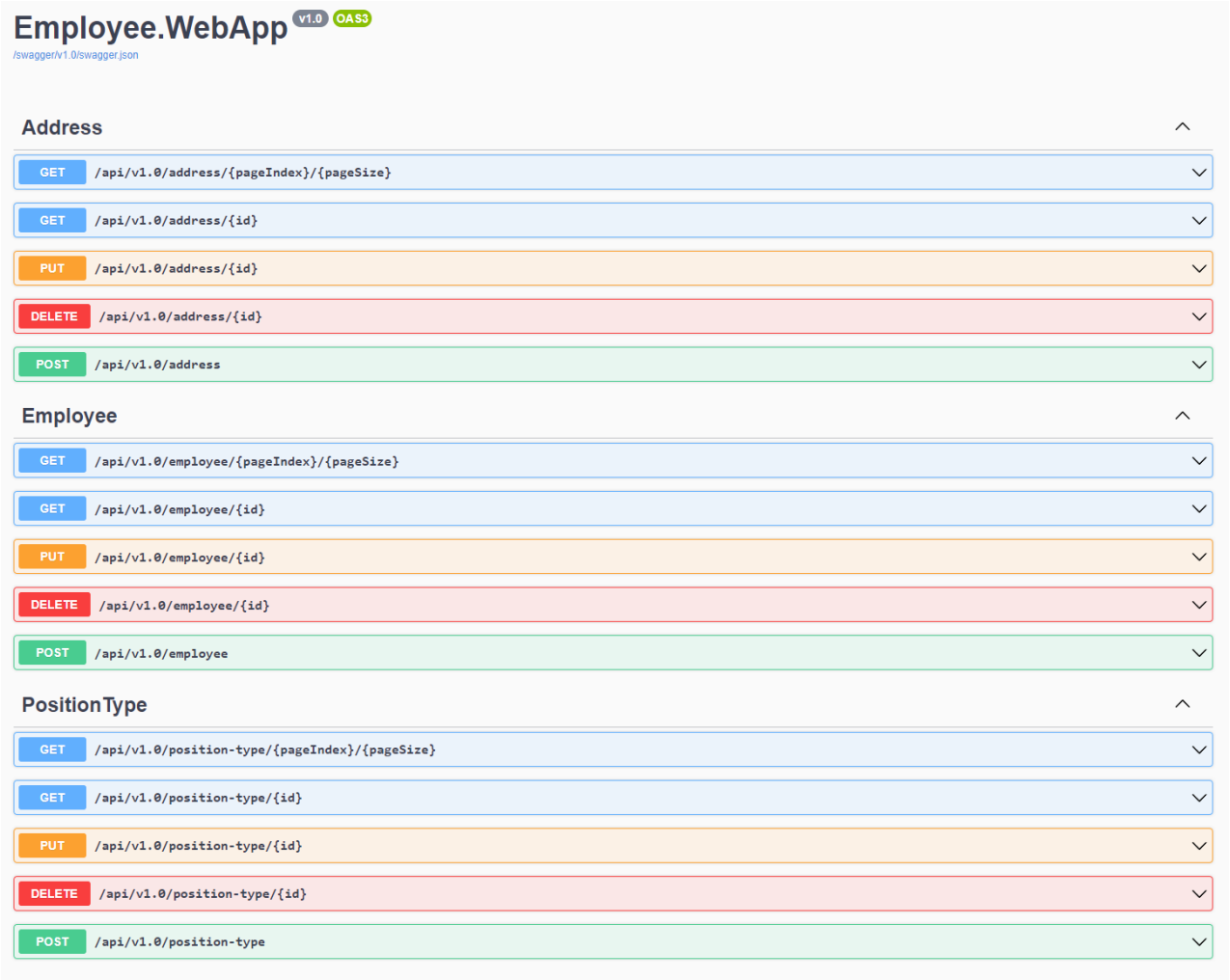


Рисунок 15 – Робота зі Swagger

4.2. Робота з Google Workspace

Для того, щоб отримати працівників з Google Worspace існує окрема кінцева точка. Це дозволяє мігрувати наявних людей у мережі Google до системи Equipy. Після чого можна присвоювати техніку, робити запити, тощо (див. рис. 16).

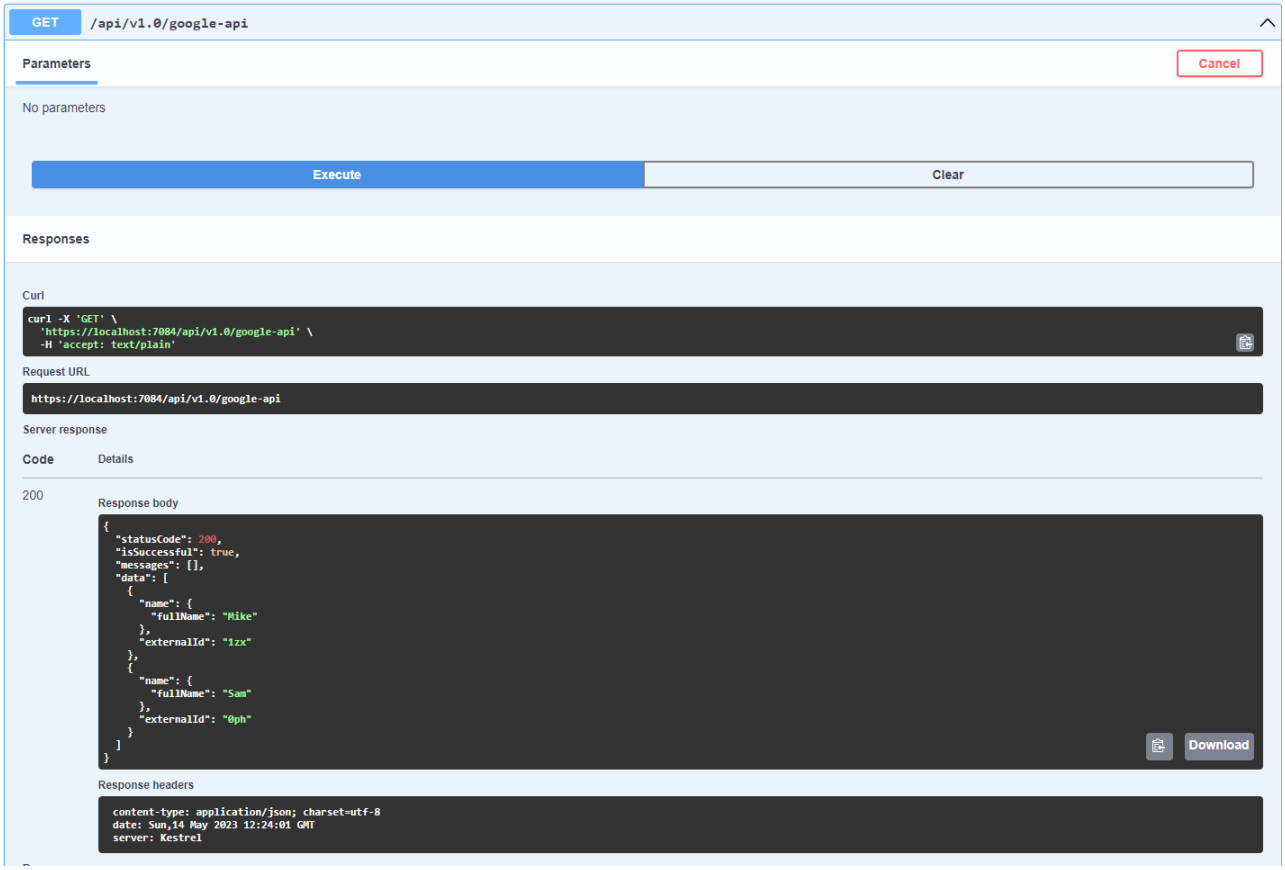


Рисунок 16 – Отримання співробітників з Google

4.3 Впровадження в компанію

Впровадження програми Equipy стало ключовим етапом у розвитку її бізнес-процесів. Програма, розроблена на базі C# .Net і використовуючи мікросервісну архітектуру, була успішно інтегрована в наявну інфраструктуру компанії.

Розроблена система виявилася ефективним рішенням для компанії, оскільки вона значно спрощує процес інвентаризації техніки та обладнання. Вона надає зручний інтерфейс для управління активами, а також дозволяє забезпечити прозорість і відслідковуваність всіх операцій з активами.

Відтепер, співробітники компанії мають змогу вести облік власного обладнання, робити запити на заміну або замовлення нового обладнання прямо через систему. Це своєю чергою полегшує процес управління обладнанням для менеджерів і адміністраторів системи, дозволяючи їм максимально ефективно використовувати ресурси компанії. Довідку про впровадження можна побачити в додатку В.

ВИСНОВКИ

В ході виконання кваліфікаційної роботи розроблено сервіс, призначений для управління процесом інвентаризації майна та ресурсів компаній. При цьому було виконано наступні завдання: досліджено ринок та попит на інвентаризацію ресурсів; розроблено концепцію програми; сплановано та розподілено обов'язки між учасниками розробки; спроєктовано, розроблено, протестовано та впроваджено систему у користування замовником.

В роботі подемонстровано, що створення нової системи інвентаризації може надати значні переваги порівняно з використанням готових рішень.

Система Equipy, розроблена спеціально для конкретних потреб бізнесу, забезпечує значну гнучкість, масштабованість та можливість налаштування.

Мікросервісна архітектура, що була використана в проєкті, дозволила створити високопродуктивну та легко масштабовану систему. Кожен мікросервіс був розроблений таким чином, щоб він виконував свої задачі ефективно та надійно.

Інтеграція з Google Workspace дозволила забезпечити зручну та ефективну роботу з даними співробітників, а використання Swagger полегшило процес документування та тестування API.

Таким чином, результатом даного проєкту стала високоякісна, гнучка та ефективна система інвентаризації, яка здатна задовольнити потреби бізнесу. Подальший розвиток та оптимізація системи можуть принести ще більші переваги для компанії.

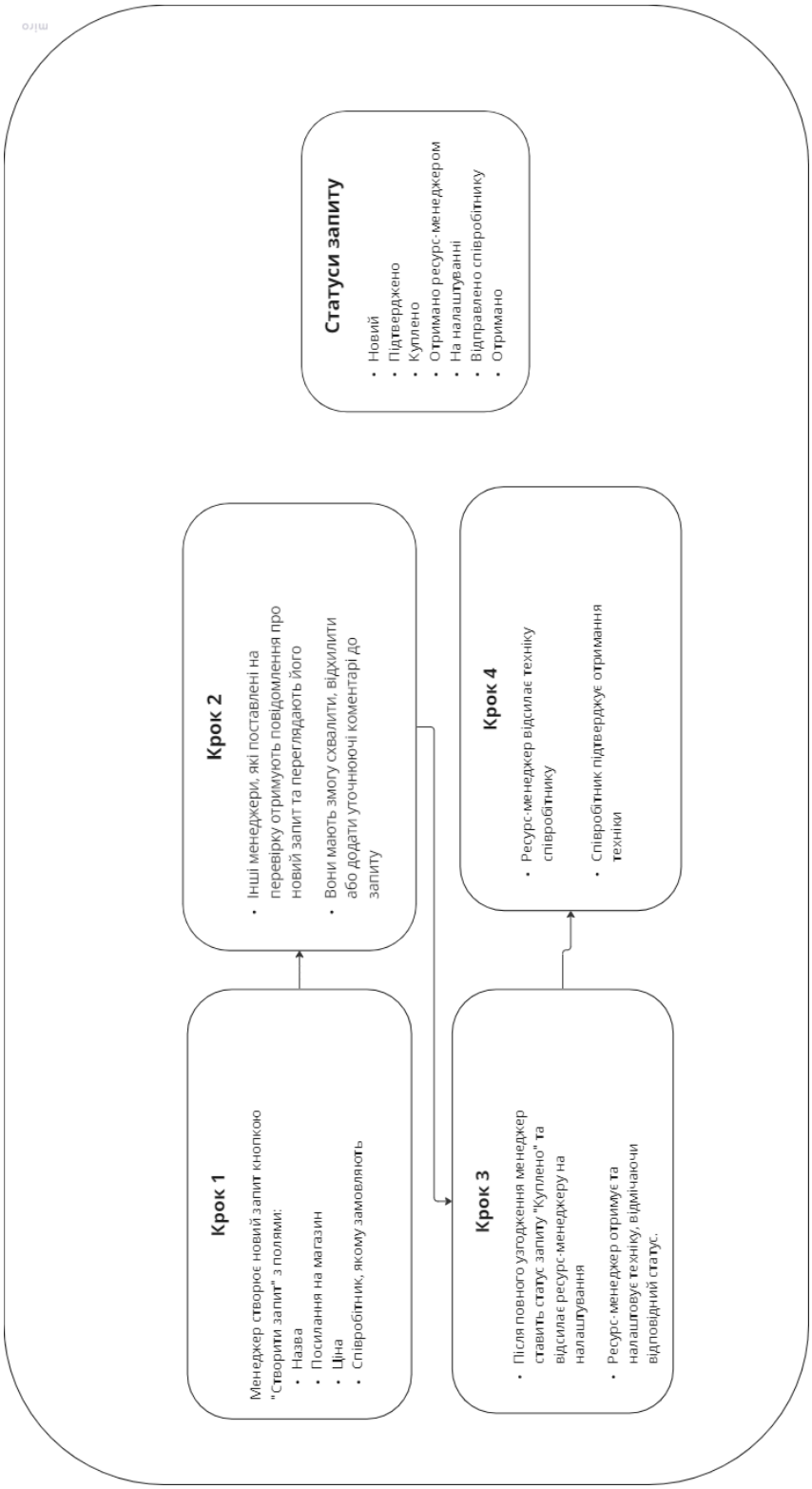
ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Бардаш С. В. Інвентаризація при аудиті виробничих запасів. / Сергій Володимирович Бардаш. – Житомир: ЖІТІ, 2000.
2. Жук В. М. Актуальні проблеми бухгалтерського обліку і їх вирішення / Валерій Миколайович Жук. – Житомир, 2009. – (7).
3. Олійник Н. Р. Проблеми та напрями вдосконалення організаційно–інституційного забезпечення фіскального регулювання трансфертного ціноутворення в Україні. / Н. Р. Олійник., 2015. – (1).
4. Про бухгалтерський облік та фінансову звітність в Україні: Закону України від 16.07.1999 р. Закон № 996-XIV (зі змінами від 10.08.2022).
5. Положення про інвентаризацію активів та зобов’язань: затверджений Міністерством фінансів України 02.09.2014 за № 879 (зі змінами від 26.05.2022).
6. Asset Panda Review [Електронний ресурс]. – 2023. – Режим доступу до ресурсу: https://www.forbes.com/advisor/business/asset-panda-review/#:~:text=Asset%20Panda%20is%20a%20cloud,manage%20and%20track%20their%20assets.
7. Snipe-IT Reviews [Електронний ресурс]. – 2023. – Режим доступу до ресурсу: https://www.capterra.com/p/150016/Snipe-IT/reviews/.
8. Microservice Architecture: Aligning Principles, Practices, and Culture – Sebastopol: OReiley Media, 2016.
9. Newman S. Monolith to Microservices / Sam Newman. – California: O'Reilly Media, Inc., 2019.
10. Martin R. Clean Architecture: A Craftsman’s Guide to Software Structure and Design / Robert C. Martin., 2017.
11. Anne J. Murach's C# / J. Anne, J. Murach., 2015. – 908 с. – (6).
12. Darren S. Mastering ASP.NET Web API / S. Darren, H. Malendra., 2017. – 330 с.

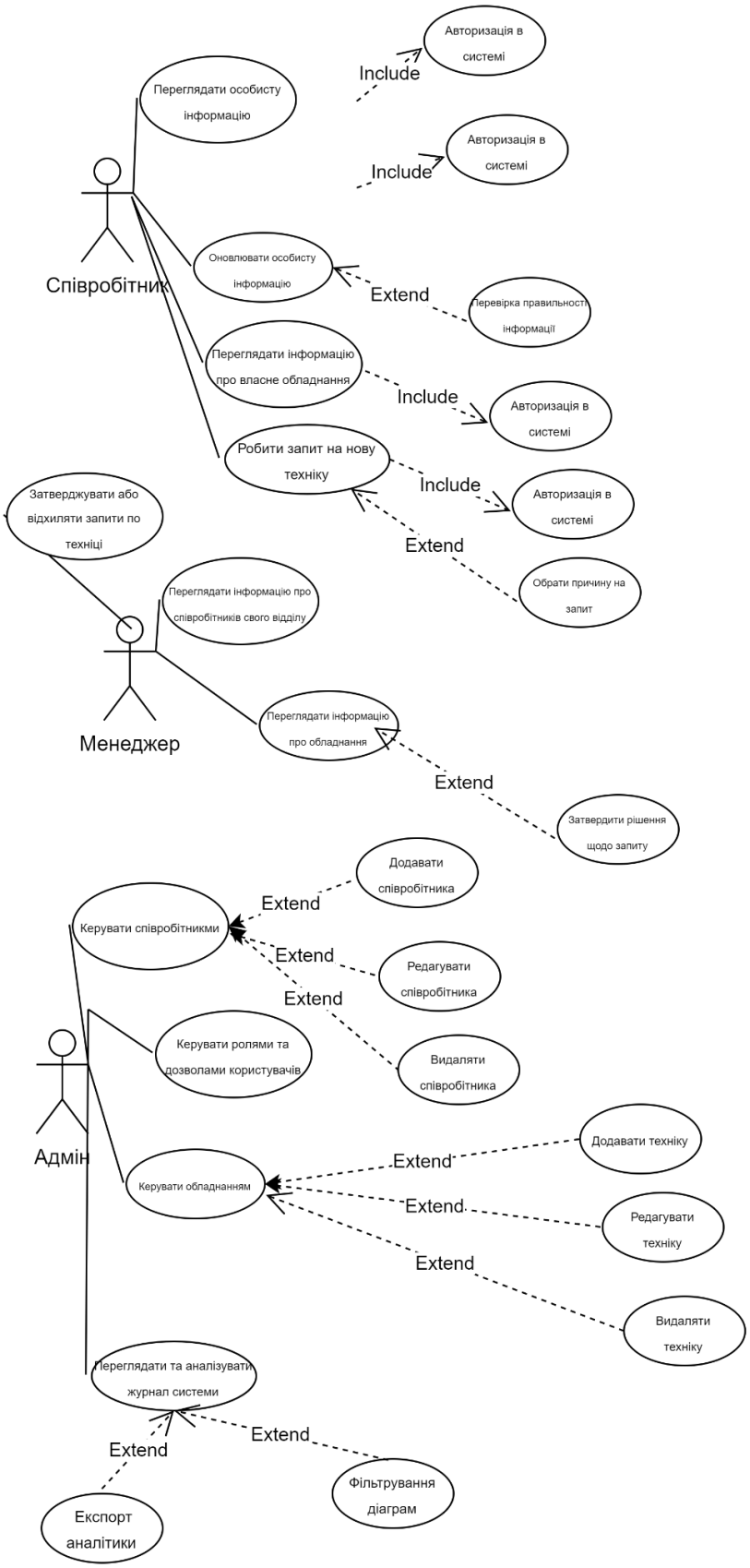
13. IdentityServer4 [Електронний ресурс]. – 2020. – Режим доступу до ресурсу: identityserver4.readthedocs.io.
14. JSON Web Token (JWT) RFC 7519 [Електронний ресурс]. – 2022. – Режим доступу до ресурсу: https://datatracker.ietf.org/doc/html/rfc7519.
15. Open API Specification [Електронний ресурс]. – 2022. – Режим доступу до ресурсу: https://www.openapis.org.

ДОДАТКИ

Додаток А. Процес запиту нової техніки



Додаток Б. Use Case діаграма



Додаток В. Довідка про впровадження

Limestone Digital

Czech Republic
Prague, Pernerova 697/35
+420 721 063 716
sales@limestonedigital.com

ДОВІДКА

Видана студентці факультету комп'ютерних наук та кібернетики Київського національного університету імені Тараса Шевченка Формакідов Діні в тому, що результати виконання її кваліфікаційної роботи бакалавра впроваджені на підприємстві «Limestone Digital s.r.o.».

Зокрема, на підприємстві використовується запропонований веб-додаток для управління процесом інвентаризації майна та ресурсів компаній, а також можливістю нативної інтеграції із "Google Workspace" системою.

Планується подальше виконання робіт щодо розширення функціональності системи та інтеграції її із "Microsoft 365"

Посада HR manager Підпис ПІБ Панасенко М.

Дата 22.05.23 Печатка