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 АДМІНІСТРАТОРА

Виконав студент 4-го курсу
Любомир МАЄВСЬКИЙ

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

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 і технології програмування
« 05 » червня 2023 р., протокол №18
Завідувач кафедри
Микола НІКІТЧЕНКО

Київ-2023

РЕФЕРАТ

Загальний обсяг роботи 61 сторінка, основний тест викладено на 56 сторінках, 36 рисунків, 6 таблиць, 15 джерел посилань, 5 додатків.

ВЕБСЕРВІС, СИСТЕМА, ПАКЕТ, КОРИСТУВАЦЬКИЙ ІНТЕРФЕЙС, ІНТЕГРОВАНЕ СЕРЕДОВИЩЕ РОЗРОБКИ, БАЗА ДАНИХ, МОВА ПРОГРАМУВАННЯ.

Об’єктом розроблення програмного засобу є процес адміністрування інформації про інвентаризацію та управління майном компанії. Предметом роботи є адміністративна частина сервісу, призначеного для управління процесом інвентаризації майна компанії, управління закупівлями техніки та іншого обладнання й управління витратами компанії.

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

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 адміністративну частину програмного продукту, за допомогою якого можна відстежувати витрати на обладнання, назначати за кожним співробітником техніку та назначати процес від покупки до отримання техніки співробітником.

Інструменти розроблення: для реалізації проєкту використано середовище розробки Visual Studio, мову програмування C#, фреймворк ASP.NET, Docker, Azure, MsSQL.

ЗМІСТ

СКОРОЧЕННЯ ТА УМОВНІ ПОЗНАЧЕННЯ

API – Application Programming Interface, прикладний програмний інтерфейс;

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

HTML – HyperText Markup Language, мова розмітки гіпертексту;

JS – JavaScript, мова програмування;

JSX – JavaScript Syntax Extension, розширення синтаксису JavaScript;

LINQ – Language Integrated Query, запити, інтегровані в мову;

MsSQL – Microsoft SQL Server;

ORM – Object–relational mapping, об'єктно-реляційна проєкція;

SaaS – Software as a service, програма як послуга;

PaaS – Platform as a Service, платформа як послуга;

SSAS – SQL Server Analysis Services, служби аналізу від Microsoft;

SSIS – SQL Server Integration Services, платформа для інтеграції даних;

SSRS – SQL Server Reporting Services, служби звітування SQL Server;

URL – Uniform Resource Locator, єдиний вказівник на ресурс;

XML – Extensible Markup Language, розширювана мова розмітки;

ПЗ – програмне забезпечення;

СУБД – система управління базами даних.

ВСТУП

Оцінка сучасного стану об’єкта розробки. Сучасні компанії, а особливо IT-організації, мають тенденцію до накопичення активів і ресурсів, що призводить до збільшення їх кількості. Часто існує велика кількість машин, обладнання та інші витрати, які необхідно враховувати, що є необхідною частиною кожної компанії. Звичайно, для цього використовуються традиційні журнали та таблиці Excel, що часто ускладнює перевірку та облік активів компанії.

Проблеми автоматизації процесу інвентаризації майна компаній розглядаються в працях наступних авторів О. Артюх та Л. Фрундіна [1], Іваненков С. [2], Шквір, В., Загородній, А., Височан О. [3] та інших.

Актуальність роботи та підстави для її виконання. У зв'язку з постійно зростаючим обсягом запасів у кожній компанії виникає потреба в зручному та потужному програмному забезпеченні (ПЗ) для відстеження та управління наявним майном і ресурсами.

Відповідно до п.1 статті 10 розділу III Закону України “Про бухгалтерський облік та фінансову звітність в Україні” [4]: “Для забезпечення достовірності даних бухгалтерського обліку та фінансової звітності підприємства зобов’язані проводити інвентаризацію активів і зобов’язань, під час якої перевіряються і документально підтверджуються їх наявність, стан і оцінка”.

Порядок та строки проведення інвентаризації активів і зобов’язань юридичних осіб та організацій різних форм власності регулюється затвердженим Міністерством фінансів України “Положеннями про інвентаризацію активів та зобов’язань” [5].

Таким чином автоматизація процесу управління майном компанії є актуальною задачею.

Мета й завдання роботи. Метою даної роботи є розробка сервісу, спрямованого на управління процесом інвентаризації майна та ресурсів підприємства.

Для досягнення цієї мети в роботі було поставлено наступні завдання:

- проаналізувати потреби ринку з точки зору потреби в інвентаризації ресурсів;
- затвердити план реалізації програми;
- спроєктувати вебзастосунок;
- реалізувати вебзастосунок;
- протестувати вебзастосунок;
- розгорнути вебзастосунок для інвентаризації;
- впровадити в компанії.

Колективна розробка. Дана система є колективним проєктом, реалізованим спільно зі студенткою 4 курсу вибіркового блоку «Теорія та технологія програмування» освітньо-професійної програми «Інформатика» Діною Формакідов. 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 системи 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, Docker, Azure, MsSQL.

В кваліфікаційній роботі використано методи дослідження порівняння, аналіз та синтез.

Можливі сфери застосування. Систему можна інтегрувати в різних компаніях, в основному в IT-сфері, оскільки сучасні компанії надають співробітникам багато пристроїв, за якими потрібно контролювати.

РОЗДІЛ 1.

ОГЛЯД НАЯВНИХ НА РИНКУ СИСТЕМ

1.1 Огляд конкурентних систем

Для порівняння систем були обрані продукти на ринку, які можуть зацікавити власників компаній або менеджерів ресурсів. Для пошуку таких продуктів створено пошуковий запит, пов’язаний із системами відстеження запасів. Одними з найпопулярніших виявилися такі системи: Odoo [6], Comarch [7] (див. табл. 1.4).

Таблиця 1 – Наявні системи інвентаризації

Система інвентаризації	**Переваги**	**Недоліки**
Odoo	Наявність частин з відкритим кодом.	Складність впровадження через те, що програма є частиною більшої системи, яка включає багато інших процедур.
Comarch	Має мобільний застосунок; Можливість сканування інвентарних штрих-кодів.	Інтеграція є складною в сучасні корпоративні системи.

Жодна з цих систем не забезпечує гнучкого та легкого переходу до використання продукту.

Odoo [6] – програмне забезпечення для управління складом онлайн.

Odoo Inventory — це модуль системи планування ресурсів підприємства від Odoo, який був створений з метою надати компаніям засоби для ефективного керування їх запасами (див. рис. 1). Ця система є частиною ширшого набору застосунків Odoo, який включає бухгалтерію, електронну комерцію, виробництво, інвентаризацію тощо [6]. В контексті управління запасами, основні переваги Odoo включають:

- Система є гнучкою завдяки своїй модульній структурі, яка дозволяє компаніям вибирати лише ті модулі, які вони потребують.
- Оскільки частина коду Odoo є відкритою, розробники можуть легко модифікувати систему для специфічних потреб бізнесу.
- Odoo може легко інтегруватись з іншими застосунками і системами, що дозволяє створювати єдину систему управління бізнесом.

Однак Odoo також має свої недоліки, включаючи:

- Через те, що Odoo є частиною більшої системи, впровадження може бути складним, особливо для компаній без великого IT-відділу.
- Часто потрібна технічна експертиза для налаштування та модифікації системи, що може бути викликом для деяких компаній.

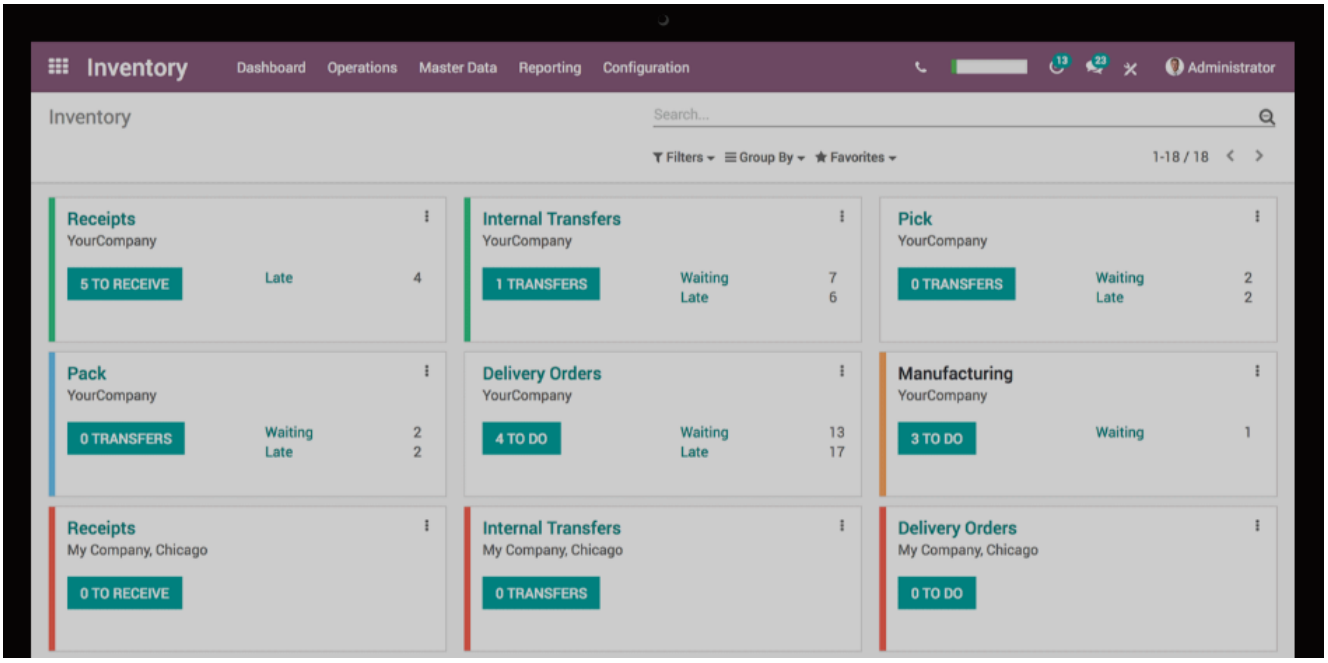


Рисунок 1 – Інтерфейс застосунку Odoo

Comarch [7] є глобальним постачальником інтегрованих IT-рішень, який має велику кількість продуктів в своєму портфелі. Спеціалізуючись на різних секторах, включаючи рішення для управління запасами (див. рис. 2), основні переваги Comarch включають:

- Comarch має мобільний застосунок, який дозволяє користувачам управляти своїми запасами в реальному часі, незалежно від їх місцезнаходження.

- Система Comarch дає змогу сканувати штрих-коди для швидкого доступу до інформації про запаси, що високо цінується в індустрії.
- Comarch є відомим і надійним рішенням для управління запасами, здатним масштабуватися для великих організацій.

Однак система Comarch також має свої недоліки:

- Відгуки вказують, що інтеграція Comarch в сучасні корпоративні системи може бути складною, що може створити проблеми для організацій, які вже використовують інші системи.
- Деякі функції Comarch можуть вимагати специфічного обладнання, що може викликати додаткові витрати для організацій.

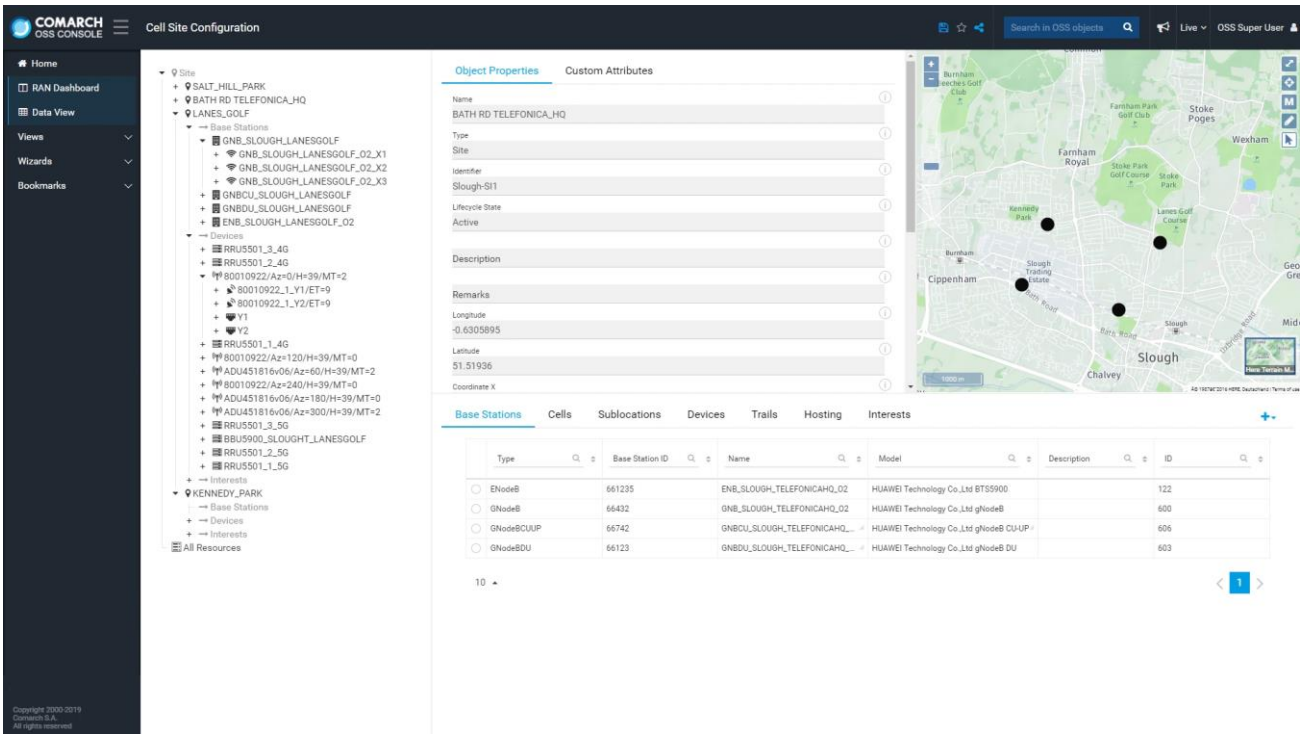


Рисунок 2 – Інтерфейс застосунку Comarch

Обидві системи – Odoo і Comarch – представляють значні переваги в області управління запасами. Однак важливо врахувати специфіку бізнесу, його розмір, а також поточну IT-інфраструктуру підприємства перед вибором конкретної системи. Як показують недоліки обох систем, вони вимагають певного рівня технічної експертизи для ефективного використання та інтеграції.

1.2. Переваги проєкту Equipy

Система Equipy відрізняється рядом переваг, що забезпечують її високу ефективність і привабливість для користувачів:

- Мікросервісна архітектура: оскільки система побудована на мікросервісній архітектурі, вона володіє високою ступенем гнучкості та масштабованості. Кожен компонент може бути розроблений, розгорнутий та масштабований незалежно від інших.
- Інтеграція з Google Workspace: інтеграція з Google Workspace дозволяє співробітникам легко управляти їх обладнанням та забезпечує простий доступ до інформації про обладнання.
- Багатофункціональність: система забезпечує широкий спектр функцій, включаючи відстеження обладнання, замовлення нового обладнання та зміну наявного.
- Надійність: завдяки використанню Docker, Kubernetes і Azure, система є високонадійною і забезпечує високу доступність.
- Безпека: використання Identity Server для аутентифікації та авторизації забезпечує високий рівень безпеки системи.
- Підтримка багатокористувацького доступу (Multi-tenancy): система може служити різним організаціям одночасно, забезпечуючи ізольованість даних і конфігурації для кожного користувача.
- Сучасні технології: використання сучасних технологій, таких як C#, .NET, RabbitMQ, MsSql, Firebase та ін., забезпечує високу продуктивність і гнучкість системи.

Зазначені переваги роблять систему Equipy високоефективним рішенням для управління інвентаризацією обладнання в організаціях будь-якого розміру. Порівняння системи Equipy з розглянутими сервісами наведено в табл. 2.

Таблиця 2 – Порівняння Equipy із наявними системами

Функціонал	Equipy	Odoo	Comarch
Управління співробітниками	+	+	+
Управління інвентарем	+	+	+
Google Workspace інтеграція	+	+	-
Microsoft 365 інтеграція	+	+	-
Автоматизована історія місцезнаходження та аудиту зображень обладнання	+	-	-
Platform-as-a-Service рішення	+	+	-
Системи запитів обладнання для співробітників	+	-	-

Проєкт Equipy, який було створено, є інформаційною системою для інвентаризації корпоративного обладнання, яку замовила певна організація. Виконання цього проєкту ініційовано замовником, з потреби в автоматизованому рішенні, яке б спрощувало та оптимізувало процеси управління обладнанням.

Замовник має відповідні вимоги до системи, які передбачають здатність відстежувати використання обладнання співробітниками в режимі реального часу. Це дозволить замовнику планувати бюджет на придбання обладнання більш ефективно. Крім того, система повинна надавати можливість управління даними про співробітників і контролювати виконання їх обов'язків.

Розробка системи відбувається з урахуванням цілей і вимог замовника, а також з дотриманням високих стандартів якості. Це забезпечує вчасне завершення проєкту і високий рівень задоволеності замовника.

РОЗДІЛ 2.

ОГЛЯД ВИКОРИСТАНИХ ТЕХНОЛОГІЙ

2.1. Обґрунтування використання технологій

Система Equipy — це вебзастосунок, який має бути продуктивним, легко масштабованим і простим у використанні з будь-яким інтерфейсом користувача. Тому при виборі методу розробки необхідно провести якісну оцінку вимог. Важливими були такі критерії:

- захищеність користувача;
- архітектурна гнучкість;
- поширення технології та її довгострокова підтримка;
- технологія має можливість прискорити процес розробки, зберігаючи при цьому високі стандарти якості.

2.2. Інкапсулювання логіки в NuGet пакети

Пакет, який пропонує велику кількість коду, бібліотек, налаштувань та інших компонентів для збагачення проєктів платформи .NET, називається пакетом NuGet [8].

NuGet служить менеджером пакунків для .NET Framework і спрощує встановлення, оновлення та організацію залежностей проєкту. Пакунки, доступні через NuGet, містять різноманітні компоненти програми, такі як бібліотеки класів, файли конфігурації та інші ресурси, які можна використовувати в проєктах .NET [8].

Пакети NuGet доступні через онлайн-репозиторії, такі як NuGet.org, де користувачі можуть переглядати та завантажувати їх. Після завантаження пакети можна встановити в проєкт через Visual Studio або через командний рядок.

Пакети NuGet служать розробникам зручним інструментом для інтеграції додаткових функцій у свої проєкти. Це не тільки скорочує час і зусилля, необхідні для написання нового коду, але й заохочує повторне використання коду, зрештою зводячи до мінімуму надмірність і максимізуючи продуктивність розробника.

Роботу з пакетами можна побачити на схемі, представленій на рис. 3.

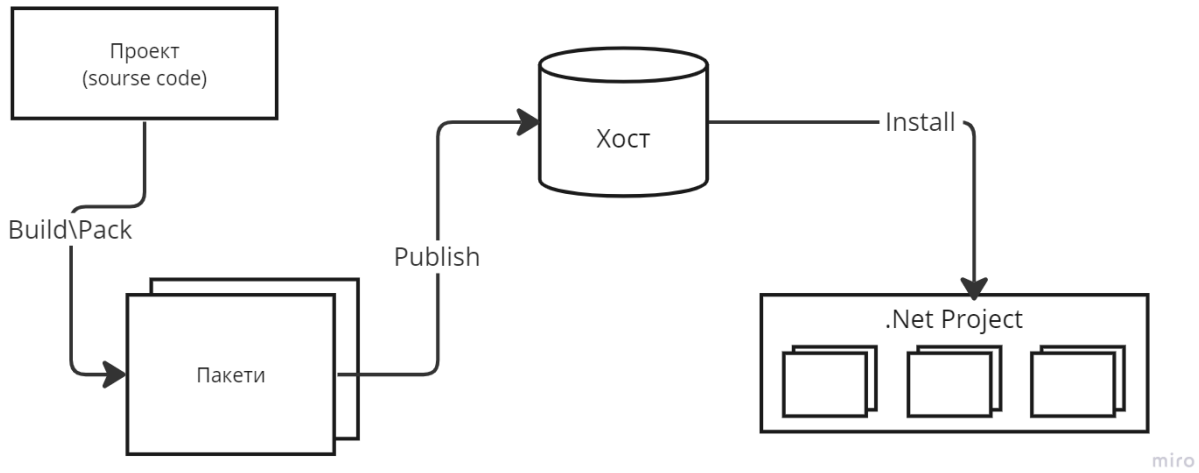


Рисунок 3 – Схема роботи з Nuget пакетами

Перевагами використання пакетів можна зазначити такі пункти:

- повторне використання: логіку, реалізовану в пакеті NuGet, можна використовувати в різних проєктах. Це зменшує дублювання коду та сприяє швидкому розгортанню нових проєктів;
- менше залежностей: пакети NuGet можуть залежати від інших пакетів, що дозволяє використовувати лише необхідні компоненти у вашому проєкті, зменшуючи загальний розмір вашого проєкту та покращуючи його продуктивність;
- простота обслуговування: надання логіки пакетам NuGet дозволяє легко оновлювати та підтримувати окремі компоненти. Це спрощує налагодження, тестування та захист вашого проєкту;

- швидше розгортання: пакети NuGet дозволяють швидко та легко додавати функціональність до проєктів, скорочуючи час розробки та випускаючи нові версії;
- спрощене розповсюдження: пакети NuGet можна легко розподіляти між різними проєктами та групами розробників, що дає змогу швидко й ефективно розподіляти функціональні можливості та скорочувати час, необхідний для налаштування робочого середовища.

2.3. Система Api Gateway

API Gateway — це серверний компонент, який полегшує керування та відправлення запитів до кількох мікросервісів у програмі на основі мікросервісів. API Gateway функціонує як вхідний портал до системи, він приймає всі запити від клієнтів і направляє їх до відповідних мікросервісів, обробляє помилки, забезпечує безпеку та автентифікацію [9].

API Gateway має додаткові можливості, включаючи збір показників, моніторинг, журналювання та обробку помилок, що робить його ефективним методом керування мікросервісами. Це також може полегшити кешування запитів, що підвищує продуктивність і зменшує навантаження на мікросервіси.

В системі Equipy є декілька мікросервісів: Equipment, Employee, Auth. Переваги використання Api Gateway можна зобразити при порівнянні (див. рис. 4 та 5).

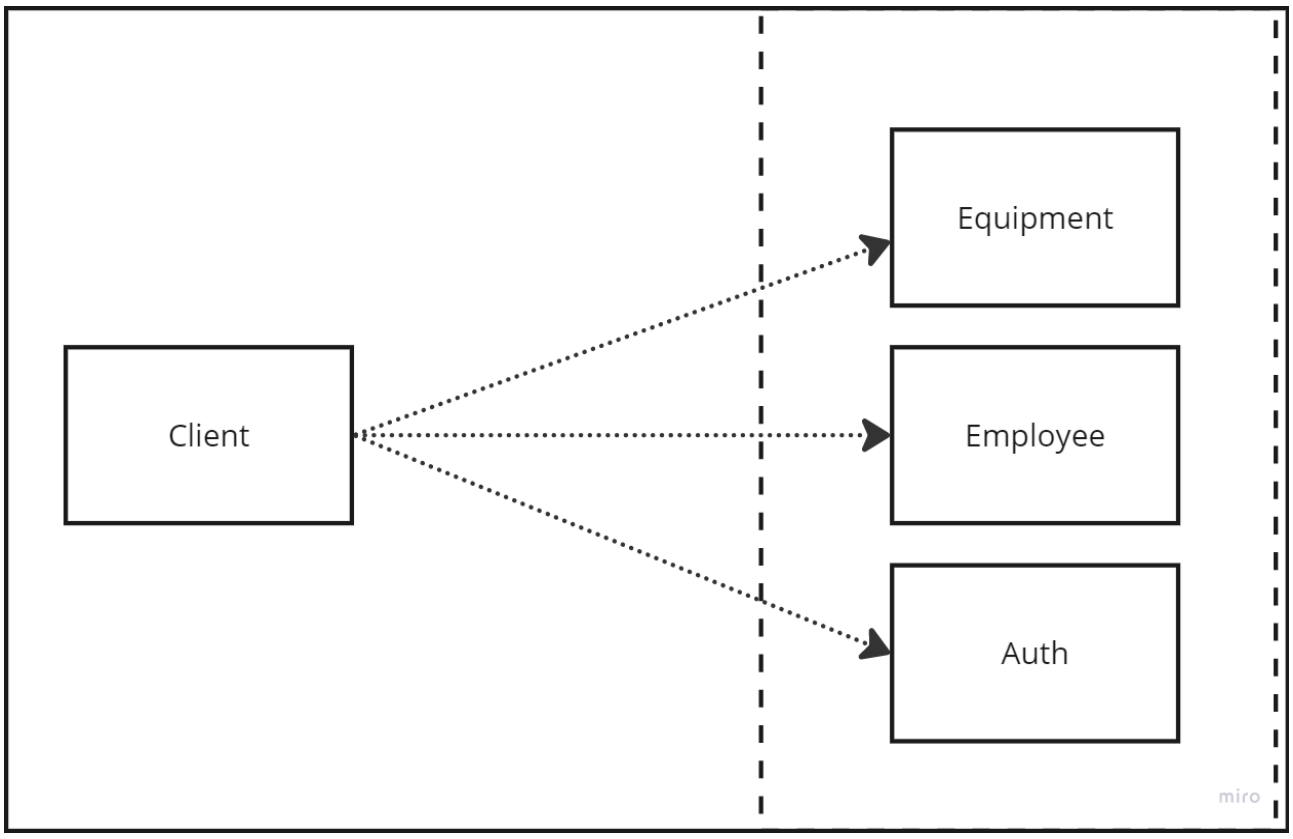


Рисунок 4 – Схема роботи сервісу без використання Api Gateway

У цьому дизайні клієнтська програма відіграє роль композитора API. Він викликає кілька служб і об’єднує результати. Клієнт повинен здійснити кілька запитів для отримання даних, а також виконувати їх послідовно. Це вимагає від розробника клієнта написання потенційно складного коду для компонування API.

Крім того, відсутність інкапсуляції ускладнює зміну API сервісу та може призвести до збоїв в роботі наявних клієнтів. Розробник сервісу іноді додає нові сервіси та може навіть змінювати API. Але якщо знання про сервіс складаються на стороні клієнта, це стає важкою задачею.

Деякі сервіси використовують протоколи, такі як gRPC або AMQP для обміну повідомленнями. Хоча ці протоколи працюють добре внутрішньо, вони можуть бути складними для споживання мобільними клієнтами. Також можуть виникати проблеми з адаптацією механізмів деяких протоколів до певних платформ клієнтів. Розв'язання цих проблем стає API Gateway (див. рис. 5).

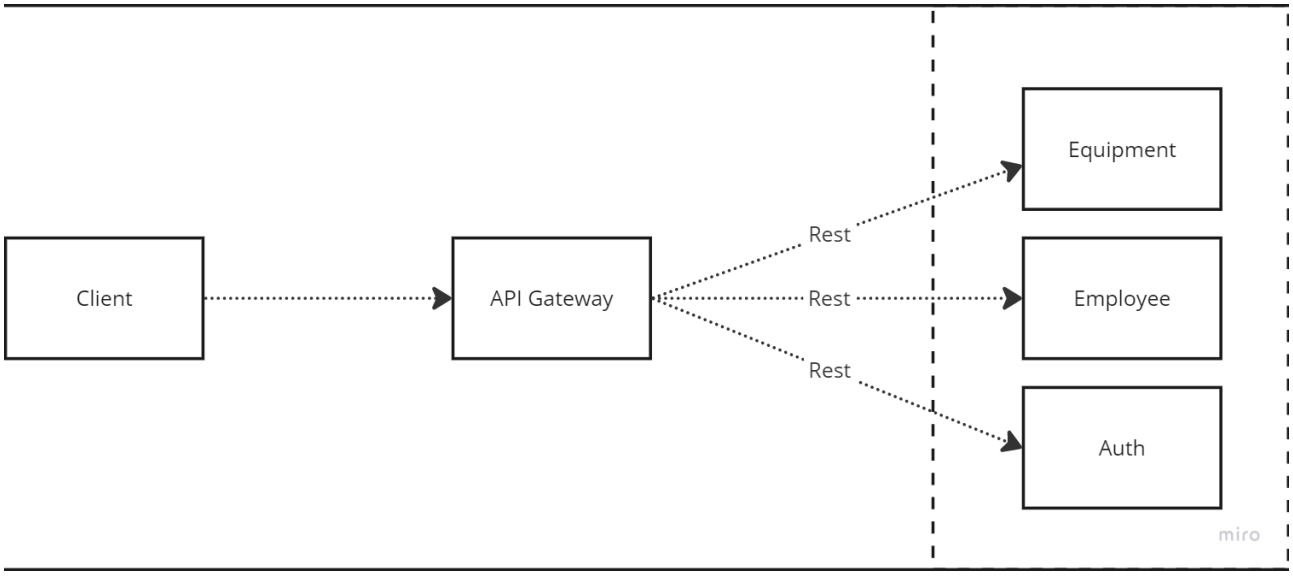


Рисунок 5 – Робота системи з використанням Api Gateway

2.4. Платформа .NET

.Net 6 – це платформа від Microsoft для створення програмного забезпечення. Середовище розробки .NET об’єктно-орієнтоване. Одним з принципів об’єктно-орієнтованого програмування є виділення коду, що повторюється в окремий модуль, який може використовуватися в різних об’єктах.

Це скорочує час розробки, полегшує подальшу підтримку програми та спрощує її модифікацію. ASP.NET забезпечує підвищену безпеку для вебпрограм, вебсайтів, мобільних програм тощо. Завдяки розширеній автентифікації Windows і керуванню конфігурацією код стає безпечним, а CLR надає дві важливі функції, такі як безпека доступу до коду та безпека на основі ролей, щоб забезпечити високий рівень безпеки для бізнес-застосунків [10].

2.5 Система контейнеризації Docker

Сучасні програмні системи все більше стикаються з вимогами до швидкості розгортання, масштабованості та переносимості. Відповідно до цього, контейнеризація стала популярним підходом, який дозволяє упакувати програмне забезпечення та всі його залежності в ізольовані контейнери, які

можуть працювати на будь-якому хості, що підтримує контейнери. Docker є однією з найпопулярніших систем контейнеризації, яка забезпечує широкі можливості для розгортання та управління контейнерами.

Docker — це програмне забезпечення з відкритим кодом, яке дозволяє пакувати, відправляти та запускати програми в контейнерах. Docker можна використовувати з C# для розгортання програм і служб .NET у контейнерах. Це дає змогу забезпечити послідовність і стабільність середовищ виконання застосунків і служб, зменшити витрати на інфраструктуру та збільшити швидкість розгортання [11].

Docker відіграє ключову роль у цьому проєкті, забезпечуючи ізоляцію та стабільність для кожного мікросервісу. Використовуючи Docker, можна гарантувати, що кожен мікросервіс працює у своєму власному середовищі, з власними залежностями, що не впливають на інші сервіси. Це створює стійкість та безпеку всієї системи.

Окрім того, Docker забезпечує консистентність між різними середовищами, розв'язувати проблему розрізненості між розробницьким, тестовим та продуктивним середовищами. Це означає, що програма, що працює в контейнері Docker, буде працювати однаково в будь-якому середовищі, що значно спрощує процес розгортання.

Також Docker спрощує масштабування та управління мікросервісами. За допомогою оркестраційних інструментів, таких як Kubernetes, можна автоматично масштабувати та управляти розгортанням застосунків.

Нарешті, Docker підтримує процеси неперервної інтеграції та доставки (CI/CD), автоматизуючи процеси збірки, тестування та розгортання застосунків. Це прискорює цикл розробки та покращує ефективність процесу розробки.

2.6 Система керування даними MsSQL

Microsoft SQL Server (MS SQL) [12] — це система керування базами даних Microsoft (СУБД), призначена для роботи з великими обсягами даних і обробки

транзакцій. Вона використовується для зберігання та управління структурованою інформацією, такою як дані про користувачів, майно, замовлення тощо.

MSSQL[12] розроблений корпорацією Майкрософт. Як правило, він включає механізм реляційної бази даних, який зберігає дані в таблицях, стовпцях і рядках, Служби інтеграції (SSIS), які є інструментом переміщення даних для імпорту, експорту та перетворення даних, Служби звітування (SSRS), які використовуються для створення звіти та надають звіти кінцевим користувачам, а також Analysis Services (SSAS), яка є багатовимірною базою даних, яка використовується для запиту даних від основної бази даних.

Деякі переваги MS SQL:

- висока продуктивність і швидкість: MS SQL має високу швидкість і продуктивність, що дозволяє ефективно обробляти великі обсяги даних і транзакцій;
- розширені функції безпеки: MS SQL має розширені функції захисту даних, такі як шифрування даних, аудит доступу, контроль доступу тощо;
- простота використання: MS SQL має простий та інтуїтивно зрозумілий інтерфейс користувача, що дозволяє легко створювати та керувати базами даних;
- широка масштабованість: MS SQL може легко масштабувати базу даних для задоволення потреб великих обсягів даних і користувачів.

2.7 Платформа для розгортання Azure

Microsoft Azure — це платформа, яка полегшує створення, розгортання та керування застосунками та службами. Azure має чисельні методи розгортання програм і служб.

Azure Container Instances — це платформа, яка полегшує розгортання контейнерів Docker без необхідності керування кластером. Це полегшує швидке розгортання одного контейнера або невеликої групи контейнерів, які підтримують масштабування сховища та відсутність стану [13].

Azure DevOps — це всеосяжна платформа, яка полегшує розробку, тестування, розгортання та керування проєктами програмного забезпечення. Допомагає автоматизувати процес розгортання програм і надає інструменти для створення, тестування та розгортання програм.

Використання Microsoft Azure як платформи для розгортання даного проєкту було обрано через ряд переваг, які вона надає. Перш за все, вона ідеально підходить для застосунків, написаних на C# .NET, завдяки глибокій інтеграції, яку надає Microsoft.

Крім того, Azure має вбудовану підтримку Docker і Kubernetes, що спрощує процес розгортання та управління контейнеризованими застосунками.Платформа також дозволяє ефективно масштабувати рішення, адаптуючись до змінних обсягів навантаження, та забезпечує високий рівень надійності. Безпека є ще одним важливим аспектом, який Azure добре обслуговує, надаючи передові механізми захисту даних і широкий спектр інструментів для керування безпекою та дотриманням нормативних вимог [13].

2.8 Фреймворк React JS

React JS — це бібліотека JavaScript, яка використовується для створення інтерфейсів користувача у вебзастосунках. React був створений Facebook і продемонстрував свою ефективність у створенні складних інтерфейсів [14].

React JS є похідним від концепції компонентів, яка полегшує поділ великих інтерфейсів на менші, багаторазово використовувані частини. Кожен компонент може мати власний внутрішній стан (який може змінюватися залежно від дій користувача або впливу інших компонентів).

Особливості фреймворка ReactJS:

- JavaScript XML або JSX є розширенням синтаксису мови JavaScript. JSX — це простий JavaScript, який дозволяє цитувати HTML і використовує цей синтаксис тегів HTML для відтворення підкомпонентів. Можна також писати на чистому старому JavaScript.

- Односторонній потік даних: у React набір незмінних значень передається засобу відтворення компонентів як властивості в його тегах HTML. Компонент не може безпосередньо змінювати будь-які властивості, але може передавати функцію зворотного виклику, за допомогою якої можна вносити зміни (див. рис. 6).
- Virtual Document Object Model.

React створює в пам’яті кеш структури даних, який обчислює внесені зміни, а потім оновлює браузер. Це дозволяє використовувати спеціальну функцію, яка дозволяє програмісту кодувати так, ніби вся сторінка відображається при кожній зміні, тоді як бібліотека реагування відображає лише компоненти, які фактично змінюються.

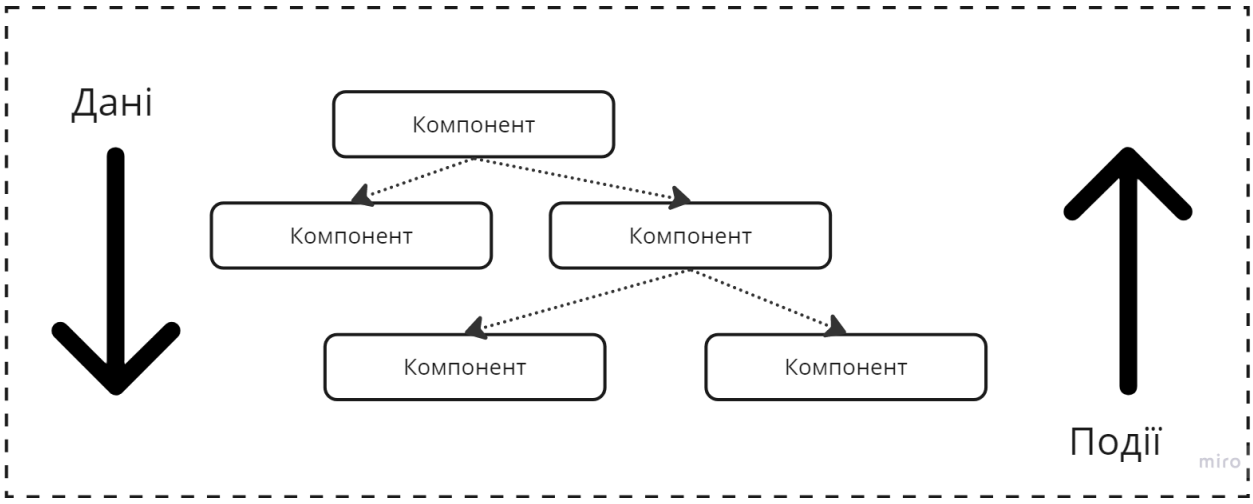


Рисунок 6 – Робота одностороннього потоку даних

2.9. Платформа для управління контейнеризованими робочими навантаженнями Kubernetes

Kubernetes – це відкрите програмне забезпечення, яке дозволяє автоматизувати розгортання, масштабування та управління контейнеризованими застосунками [15]. Воно було спеціально розроблено для управління розгортанням мікросервісів у контейнерах, як це зроблено в проєкті Equipy.

Kubernetes використовується в даному проєкті для автоматизації та управління розгортанням мікросервісів у Docker контейнерах. Це дозволяє легко масштабувати систему, додавати або видаляти мікросервіси за потреби, а також забезпечує високу доступність і надійність системи.

Використання Kubernetes також дає можливість автоматизувати процеси відновлення після збоїв, моніторингу стану сервісів та розподілу навантаження між контейнерами. Все це робить Kubernetes відмінним інструментом для управління розгортанням мікросервісів в проєкті.

РОЗДІЛ 3.

РЕАЛІЗАЦІЯ СИСТЕМИ

3.1. Інкапсуляція логіки NuGet Пакети

3.1.1. Впровадження підходу Multitenancy

В даному проєкті було обрано підхід MultiTenancy. Для того, щоб зрозуміти його переваги, спочатку подивимось на підхід single-tenant.

Ось деякі важливі характеристики, які визначають архітектуру з одним орендарем:

- унікальний запущений екземпляр зі збереженням стану для кожного клієнта;
- індивідуальна та індивідуальна база даних;
- повна ізоляція від інших програм і даних;
- виділена інфраструктура;
- можна розгорнути локально/в хмарі;
- резервні копії ізольовані та безпечні;
- необмежена кількість налаштувань;
- підтримує юридичні та відповідні вимоги.

Архітектура з кількома клієнтами прискорює розвиток архітектури з одним клієнтом. Оскільки говориться про SaaS-застосунок, очевидно, що для цього застосунка буде кілька клієнтів або орендарів. Іншими словами, є кілька груп користувачів, які належать до різних організацій і мають доступ до того самого програмного забезпечення.

Застосунки з кількома клієнтами розгортаються в одному місці. Подібним чином команда операцій та інфраструктури повинна підтримувати розгортання лише в одному місці. У зв’язку з цим розгортання, обслуговування та оновлення архітектури з кількома клієнтами спрощує процес розробки.

Основною розробкою даного підходу є підпрограмне забезпечення, яке допомагає працювати з орендарями. Також створюється TenantEntity (див. рис. 7):

```
public class TenantEntity : IAuditableEntity, ITenantEntity
{
    public Guid Id { get; set; }
    public Guid? OwnerId { get; set; }

    public string Name { get; set; }
    public string Subdomain { get; set; }
    public DateTime CreatedAt { get; set; }
    public DateTime ModifiedAt { get; set; }
}
```

Рисунок 7 – Клас TenantEntity

Після цього створюється TenantContext, що використовується для зберігання інформації, специфічної для кожного орендаря. Це є розширенням над стандартним контекстом від EF Core (див. рис. 8).

```
public class TenanDataContext : AuditDataContext, ITenantDbContext
{
    protected readonly ITenantIdentifyingService _tenantIdentifyingService;

    public TenanDataContext(DbContextOptions options, ITenantIdentifyingService tenantIdentifyingService) : base(options)
    {
        _tenantIdentifyingService = tenantIdentifyingService;
    }

    public DbSet<TenantEntity> Tenants { get; set; }

    protected override void OnModelCreating(ModelBuilder modelBuilder)
    {
        base.OnModelCreating(modelBuilder);

        modelBuilder.ApplyConfiguration(new TenantEntityConfiguration());
        modelBuilder.SeedDefaultTenant();

        TenantHelper.ConfigureTenancyForEntities(modelBuilder, _tenantIdentifyingService, this);
    }

    public override int SaveChanges()
    {
        TenantHelper.AddTenantInfo(_tenantIdentifyingService, this);
        return base.SaveChanges();
    }

    public override int SaveChanges(bool acceptAllChangesOnSuccess)
    {
        TenantHelper.AddTenantInfo(_tenantIdentifyingService, this);
        return base.SaveChanges(acceptAllChangesOnSuccess);
    }
```

Рисунок 8 – Клас DataContext для орендаря

3.1.2 Розробка Custom Exceptions

Response Result Core – наступний пакет, який активно використовується в проєкті.

Створення персоналізованого Response Result для контролерів відкриває нові можливості для оптимізації та гнучкості API. Таке рішення стандартизує формат відповіді, що спрощує використання API для клієнтів, оскільки вони можуть бути впевнені в очікуваному форматі відповіді.

Крім того, такий підхід покращує обробку помилок, адже дозволяє включати додаткову інформацію про помилку безпосередньо у відповідь [16]. Також важливою перевагою є можливість додавати в відповідь метадані, наприклад, для реалізації пагінації.

Нарешті, використання індивідуального Response Result дає більше свободи для налаштування відповідей на основі конкретних потреб розробників та клієнтів. Так, персоналізований Response Result може стати надзвичайно корисним для забезпечення більшого контролю, уніфікованості та адаптивності API відповідей.

Клас ExceptionHandling служить основною логікою повертання відповідей. Він допомагає оброблювати exceptions на прикладі BaseResponseResultException (див. рис. 9).

```
public abstract class BaseResponseResultException : Exception
{
    public BaseResponseResultException(IEnumerable<ResponseMessage> messages)
        => Messages = messages;

    public BaseResponseResultException(ResponseMessage message)
        => Messages = new List<ResponseMessage>() { message };

    public BaseResponseResultException(string message)
        => Messages = new List<ResponseMessage>() { new() { Type = ResponseMessageType.Error, Text = message } };

    public BaseResponseResultException(IEnumerable<string> messages)
        => Messages = messages.Select(text => new ResponseMessage() { Type = ResponseMessageType.Error, Text = text });

    public IEnumerable<ResponseMessage> Messages { get; set; }

    public abstract int HttpStatusCode { get; init; }
}
```

Рисунок 9 – Індивідуальний клас BaseResponseResultException

Для того, щоб скористуватись функціоналом, було створено контролер, який буде повертати відповідь (див. рис. 10):

```
[HttpGet("{id}")]
[ProducesResponseType(typeof(HttpResponseResult<EmployeeViewModel>), StatusCodes.Status200OK)]
[ProducesResponseType(typeof(HttpResponseResult<EmployeeViewModel>), StatusCodes.Status404NotFound)]
[ProducesResponseType(typeof(HttpResponseResult<EmployeeViewModel>), StatusCodes.Status404NotFound)]
[ProducesResponseType(typeof(HttpResponseResult<EmployeeViewModel>), StatusCodes.Status500InternalServerError)]
0 references
public async Task<IActionResult> Get([FromRoute] Guid id)
{
    var httpResponseResult = new HttpResponseResult<EmployeeViewModel>
    {
        Data = (await _entityManager.GetByIdAsync(id)).Adapt<EmployeeViewModel>()
    };

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

Рисунок 10 – Використання функціонала

Для того, щоб продемонструвати роботу, було зроблено запит з помилкою. На виході було отримано відповідь, що представлена на рис. 11.


```
{
  "isSuccessful": false,
  "messages": [
    {
      "type": 2,
      "text": "An error occurred while saving the entity changes. See the inner exception for details."
    },
    {
      "type": 2,
      "text": "The INSERT statement conflicted with the FOREIGN KEY constraint \"FK_Employees_EquipmentTypes_PositionTypeId\". The conflict occurred in database \"employee-database\", table \"dbo.EquipmentTypes\", column 'Id'.\r\nThe statement has been terminated."
    }
  ],
  "data": null
}
```


Download

Рисунок 11 – Приклад отримання помилки

3.1.3 Реалізація пагінації Paged List

PagedList – це важливий інструмент для оптимізації вебзастосунків, який допомагає керувати великими наборами даних. В проєкті створено наступний клас з параметрами, що представлені в табл. 3:

Таблиця 3 – Модель пагінації

Поле	**Опис**
int PageIndex	З якої сторінки починати
int PageSize	Кількість елементів
int TotalCount	Скільки всього елементів
int TotalPages	Скільки всього сторінок
int IndexFrom	З якого індексу починати
IList<TEntity> Items	Елементи
bool HasPreviousPage	Перевірка чи є сторінка перед

	поточною
bool HasNextPage	Перевірка чи є сторінка після поточної

Після реалізовано логіку роботи з пагінацією:

1. Зберігаються значення pageIndex (номер поточної сторінки), pageSize (розмір сторінки) і indexFrom (індекс, з якого починається нумерація) у відповідних властивостях класу PagedList.
2. Обчислюється загальна кількість елементів у вихідній колекції за допомогою методу Count() від запитувального типу.
3. Обчислюється загальна кількість сторінок, яка потрібна для розміщення всіх елементів.
4. Якщо pageIndex більше, ніж максимально можлива сторінка, то pageIndex змінюється на максимально можливу сторінку.
5. За допомогою методів Skip() та Take() відбираються елементи, які відповідають поточній сторінці, і зберігаються у властивості Items класу PagedList.

Відповідно, якщо є колекція зі 100 елементів і викликано конструктор PagedList з pageIndex = 2, pageSize = 10 і indexFrom = 0, то буде отримано об'єкт PagedList, який містить 10 елементів з 21 по 30, загальна кількість сторінок буде 10, а загальна кількість елементів – 100.

Для використання пагінації було створено контролер. Наприклад, отримання інформації про усіх співробітників.

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

    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<EmployeeViewModel>()
        .ToPagedListAsync(pageIndex, pageSize);

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

Рисунок 12 – Робота пагінації

Для цього було перевірено вхідні дані та викликано метод розширення ToPagedListAsync (див. рис. 12).

3.1.4 Entity Manager

Entity Manager – це шаблон проєктування, що використовується для керування об'єктами-сутностями в об'єктно-орієнтованій системі. В контексті C# і баз даних, Entity Manager зазвичай описує функціональність для зберігання, отримання, оновлення та видалення об'єктів з бази даних [17].

В системі Equipy це методи: GetAll, GetByIdAsync, CreateAsync, UpdateAsync, DeleteAsync.

Entity Manager є generic, тому він дозволяє використовувати будь-які моделі системи (див. рис. 13).

Для того, щоб користуватись менеджером потрібно додати налаштування:

```
public static class ProgramExtensions
{
    public static IServiceCollection AddApplicationCommonDependensies(this IServiceCollection services)
    {
        services.AddManagers();

        return services;
    }

    public static void AddManagers(this IServiceCollection services)
    {
        services.AddScoped(typeof(IEntityManager<>), typeof(EntityManager<>));
    }
}
```

Рисунок 13 – Підключення Entity Manager до проєкту

Це налаштування також входить в пакет, тому реалізація логіки в мікросервісах є інкапсульованою.

3.2 Робота з базою даних

У реалізованому проєкті використовується база даних Microsoft SQL Server (MsSQL) для зберігання та керування даними. Щоб підтримувати взаємодію з базою даних, було використано Entity Framework Core, потужний ORM (Object-

Relational Mapping) інструмент, який дозволяє працювати з даними на більш високому рівні абстракції.

Спочатку, було створено моделі даних у C#, які відображають структуру таблиць у базі даних. Далі, використовуючи механізм міграцій Entity Framework, можна автоматично створювати або змінювати структуру бази даних відповідно до моделей.

Для взаємодії з базою даних, було використано контекст даних, який представляє сесію з базою даних і дозволяє виконувати CRUD (створення, читання, оновлення та видалення) операції. Завдяки LINQ (Language Integrated Query) можна писати запити до бази даних прямо на C#, використовуючи звичайний синтаксис мови.

Крім того, використовуючи Entity Framework, можна легко впроваджувати розширені функції, такі як відстеження змін, ліниве завантаження, кешування, транзакції та багато іншого. Завдяки цьому, робота з MsSQL в цьому проєкті стає ефективною та гнучкою.

Оскільки було використано мікросервісну архітектуру, необхідно декілька баз даних: Equipment, Employee, Auth, Notification.

Нижче наведено рисунок діаграми класів для Equipment Service (див. рис. 14).

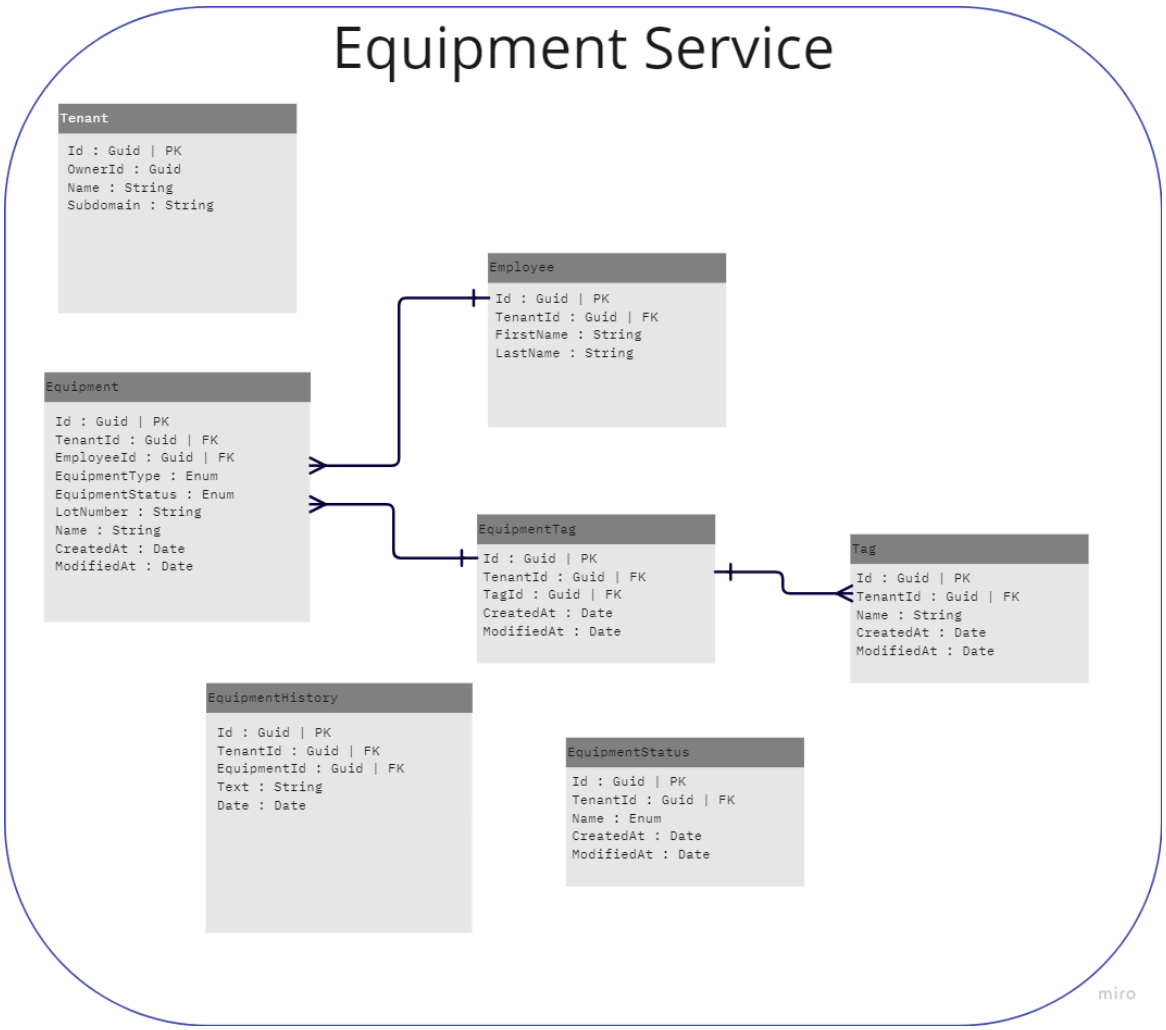


Рисунок 14 – Діаграма класів Equipment Service

В даному сервісі є такі сутності:

- Tenant;
- Equipment;
- Employee;
- Tag;
- EquipmentStatus;
- EquipmentTag;
- EquipmentHistory.

В табл. 4 наведено опис полів до кожної таблиці.

Таблиця 4 – Опис полів таблиць Equipment Service

Назва	**Опис**

Tenant	Id : Guid | PK OwnerId : Guid Name : String Subdomain : String
Equipment	Id : Guid | PK TenantId : Guid | FK EmployeeId : Guid | FK EquipmentType : Enum EquipmentStatus : Enum LotNumber : String Name : String CreatedAt : Date ModifiedAt : Date
Employee	Id : Guid | PK TenantId : Guid | FK FirstName : String LastName : String
Tag	Id : Guid | PK TenantId : Guid | FK Name : String CreatedAt : Date ModifiedAt : Date
EquipmentStatus	Id : Guid | PK TenantId : Guid | FK Name : Enum CreatedAt : Date ModifiedAt : Date
EquipmentTag	Id : Guid | PK TenantId : Guid | FK TagId : Guid | FK CreatedAt : Date ModifiedAt : Date
EquipmentHistory	Id : Guid | PK TenantId : Guid | FK EquipmentId : Guid | FK Text : String Date : Date

На рис. 15 наведено діаграму класів для Employee Service.

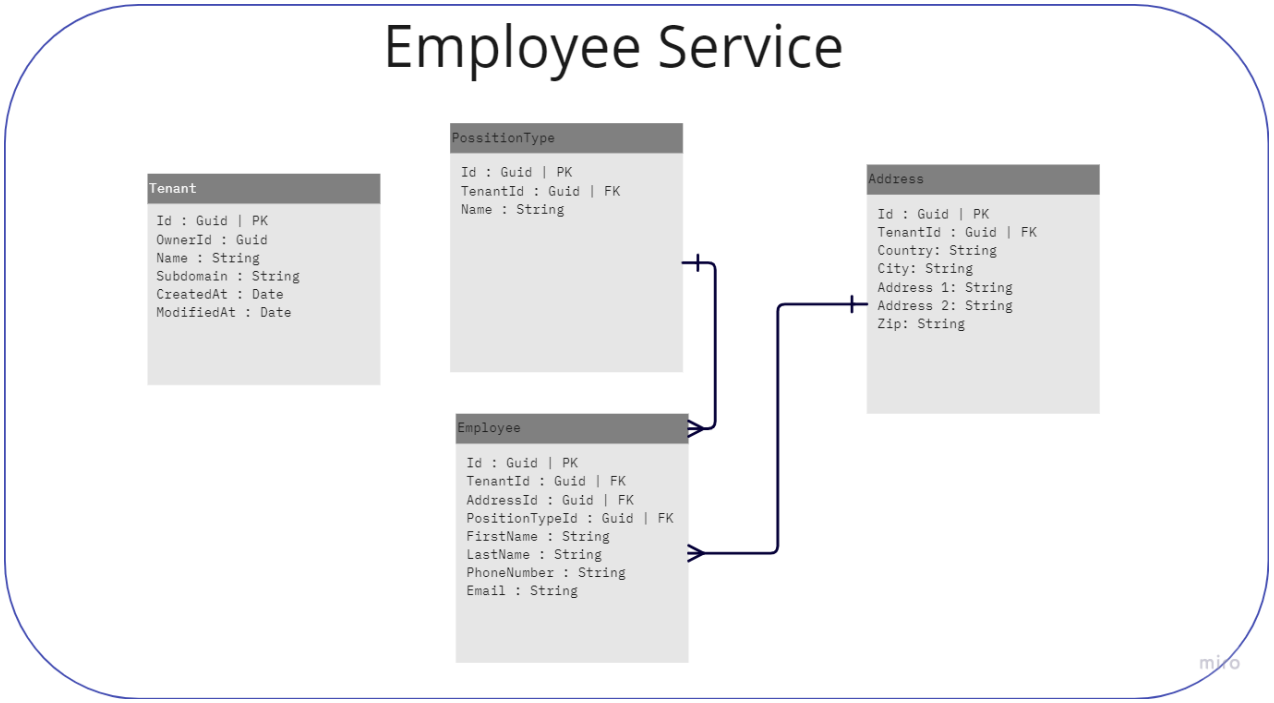


Рисунок 15 – Діаграма класів Employee Service

Employee Service має в собі Employee, Address, PositionType, Tenant.

Auth Service. Нижче наведено рисунок діаграми класів для Auth Service (див. рис. 16).

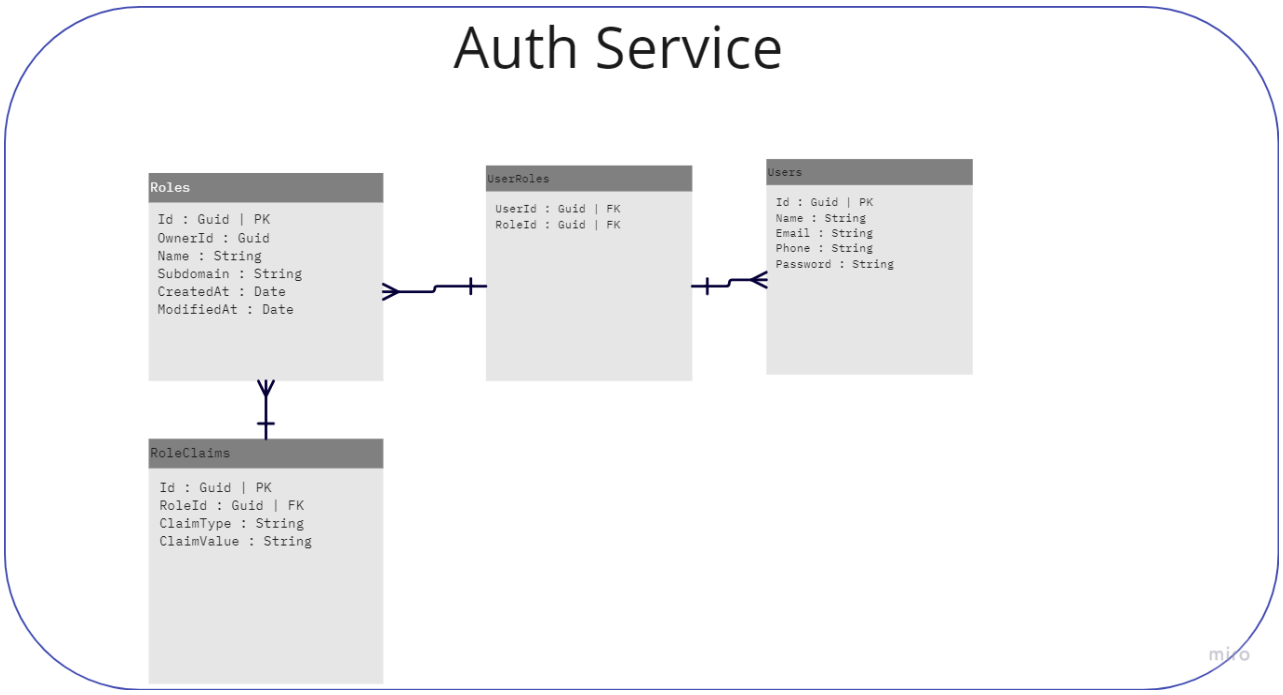


Рисунок 16 – Діаграма класів Auth Service

Notification Service.

Структуру бази для даного сервісу можна побачити в таблиці нижче (дис. табл. 16.).

Таблиця 5 – Опис полів таблиць Notification Service

Таблиця	Поля
Notifications	NotificationId NotificationTypeId UserId Message CreatedDate SentDate Status
Users	UserId FirstName LastName Email
NotificationTypes	TypeId Name

3.3. Розгортання системи на Azure

Для того, щоб розгорнути систему потрібно виконати наступні етапи.

Етап 1.

Здійснено комунікацію між сервісами за допомогою RabbitMQ.

RabbitMQ використовується в цьому проєкті як посередник повідомлень, що дозволяє мікросервісам спілкуватися між собою асинхронно, з декількома перевагами, такими як зменшення кількості зв'язків між сервісами, підтримка шаблонів розподілених систем, таких як відкладене виконання та патерн відправника-одержувача, і покращення надійності через відновлення повідомлень при відмові компонентів системи (див. додатки Д та Г)..

Щодо реалізації, було використано бібліотеку RabbitMQ .NET Client для взаємодії з сервером RabbitMQ. Нижче наведено основні кроки:

1. Встановлення з'єднання: для початку було встановлено з'єднання з сервером RabbitMQ, використовуючи об'єкт ConnectionFactory і вказуючи URL сервера.
2. Створення каналу: після встановлення з'єднання було створено канал, який служить основним шляхом для надсилання та отримання повідомлень.
3. Декларація черги: було задекларовано чергу, до якої будуть надходити повідомлення. Черга може бути вже наявної на сервері, або її можна створити автоматично.
4. Надсилання та отримання повідомлень: для надсилання повідомлень було використано метод BasicPublish, вказуючи ім'я черги та саме повідомлення. Для отримання повідомлень було використано метод BasicConsume, який створює споживача для певної черги і викликає зазначену функцію обробки кожного повідомлення.
5. Обробка помилок і відмов: було використано механізми RabbitMQ для обробки помилок та відмов, такі як повторне відправлення повідомлень, що не були успішно оброблені, а також використання "мертвих" черг для зберігання повідомлень, що не можуть бути доставлені.
6. Закриття з'єднання: після завершення роботи з сервером RabbitMQ, було закрито з'єднання і канал, що допомагає звільнити ресурси.

Однією з особливостей RabbitMQ в проєкті є те, що він використовується для реалізації системи надсилання сповіщень. Коли користувач вносить зміни до обладнання, створюється повідомлення, яке потім відправляється через RabbitMQ до Notification Service (див. рис. 17). Це дозволяє відокремити процес створення повідомлень від їх відправлення, додаючи гнучкості та масштабованості до системи.

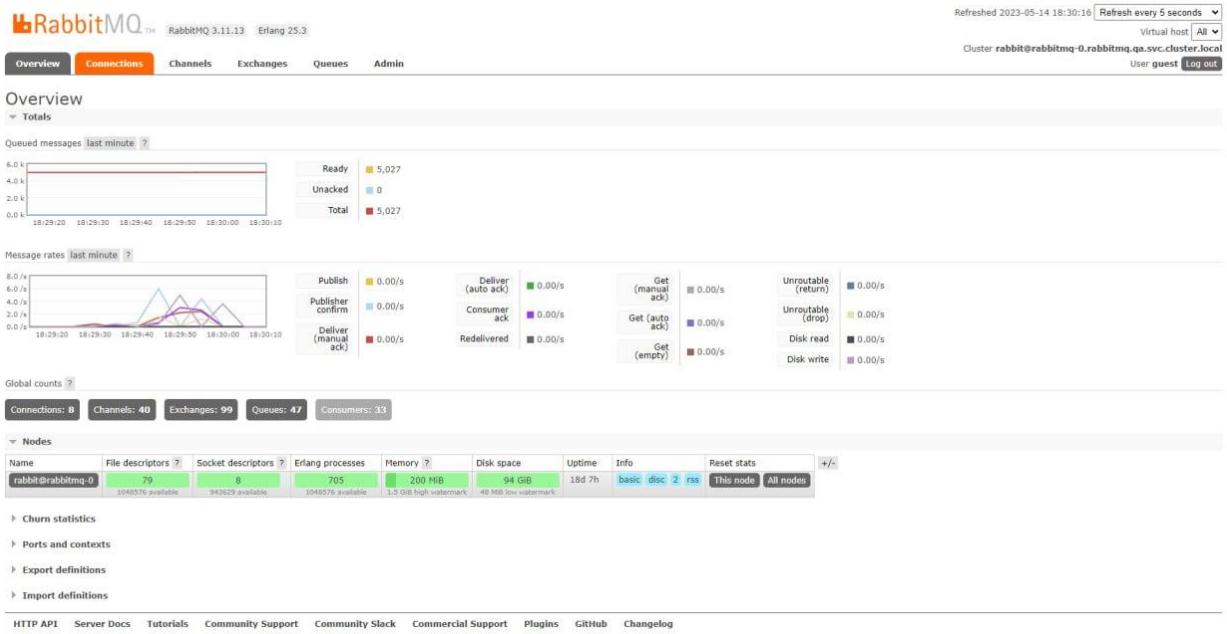


Рисунок 17 – Робота сервісу RabbitMQ

Етап 2. Налаштування Api Gateway.

Ocelot є легким API Gateway, створеним спеціально для .NET і ASP.NET Core. Він охоплює низку функцій, як-от маршрутизація, агрегація запитів, автентифікація та авторизація, кешування, обмеження та ін.

Для реалізації було створено проєкт для Api Gateway та додано Ocelot. Головним є додавання конфігурацій для визначення маршрутів. Кожен маршрут охоплює інформацію про вхідний і вихідний маршрутизатори.

Після успішного налаштування можна скористатися сервісами: https://localhost:44382/gateway/**employee** або https://localhost:44382/gateway/**equipment**, і т.д..

Етап 3. Налаштування Consul

Consul від HashiCorp - це розподілена система обслуговування для забезпечення високої доступності та розподіленої служби знаходження, конфігурації та оркестрації в масштабованих мікросервісних архітектурах. У проєкті Equipy, Consul використовується для виявлення служб та забезпечення їх доступності.

У даному вебзастосунку, Consul дозволяє мікросервісам автоматично знаходити інші служби, з якими їм потрібно взаємодіяти. Це означає, що не

потрібно жорстко кодувати IP-адреси або порти служб у коді, що робить систему гнучкою та легко масштабованою.

Consul інтегрується з Ocelot API Gateway, що дозволяє використовувати його для маршрутизації запитів до відповідних мікросервісів. Крім того, Consul забезпечує резервне копіювання і відновлення конфігурації, що є важливим для стійкості системи до відмов.

Щоб використовувати Consul в .NET Core, потрібно встановити відповідний пакет NuGet, а потім налаштувати Consul у файлі Startup.cs. Конфігурація Consul передбачає вказівку URL Consul-агента та реєстрацію служби в Consul. Після цього, мікросервіси можуть використовувати Consul для виявлення інших служб і надання своїх власних служб для виявлення.

Етап 4. Розгортання на Azure

Робота з Docker в контексті проєкту на основі мікросервісів включає кілька основних етапів.

1. Створення Dockerfile: Для кожного мікросервісу було створено Dockerfile, який визначає, як будується Docker-образ. Dockerfile охоплює вказівки, які визначають базовий образ (наприклад, образ .NET), додавання коду до образу, встановлення залежностей, визначення команди, яка запускає застосунок тощо.

2. Створення Docker-образів: Після того, як Dockerfile було додано, було створено Docker-образ для мікросервісу за допомогою команди docker build.

3. Запуск Docker-контейнерів: Після створення образу було запущено контейнер з цим образом за допомогою команди docker run. Було вказано різні параметри, такі як порти, які треба відкрити, або змінні середовища, які треба передати до контейнера.

4. Використання Docker Compose: Для спрощення управління множинними контейнерами було використано Docker Compose. Docker Compose дозволяє визначити всі служби в одному файлі docker-compose.yml і керувати ними як одним цілим. Можна використовувати

команди docker-compose up та docker-compose down для запуску та зупинки всіх служб разом.

5. Мережеві налаштування Docker: Оскільки мікросервіси потребують спілкуватися між собою, потрібно налаштувати мережу Docker так, щоб контейнери могли спілкуватися між собою. Для цього використано внутрішні мережі Docker, визначені в файлі Docker Compose.

Після того, як Docker було налаштовано, переходимо до розгортання. Для цього було створено ресурси на Azure. Після того, як створили всі необхідні ресурси, потрібно зібрати Docker-образи та завантажити їх до Azure Container Registry (див. рис. 18). Це можна зробити за допомогою команд docker build та docker push.

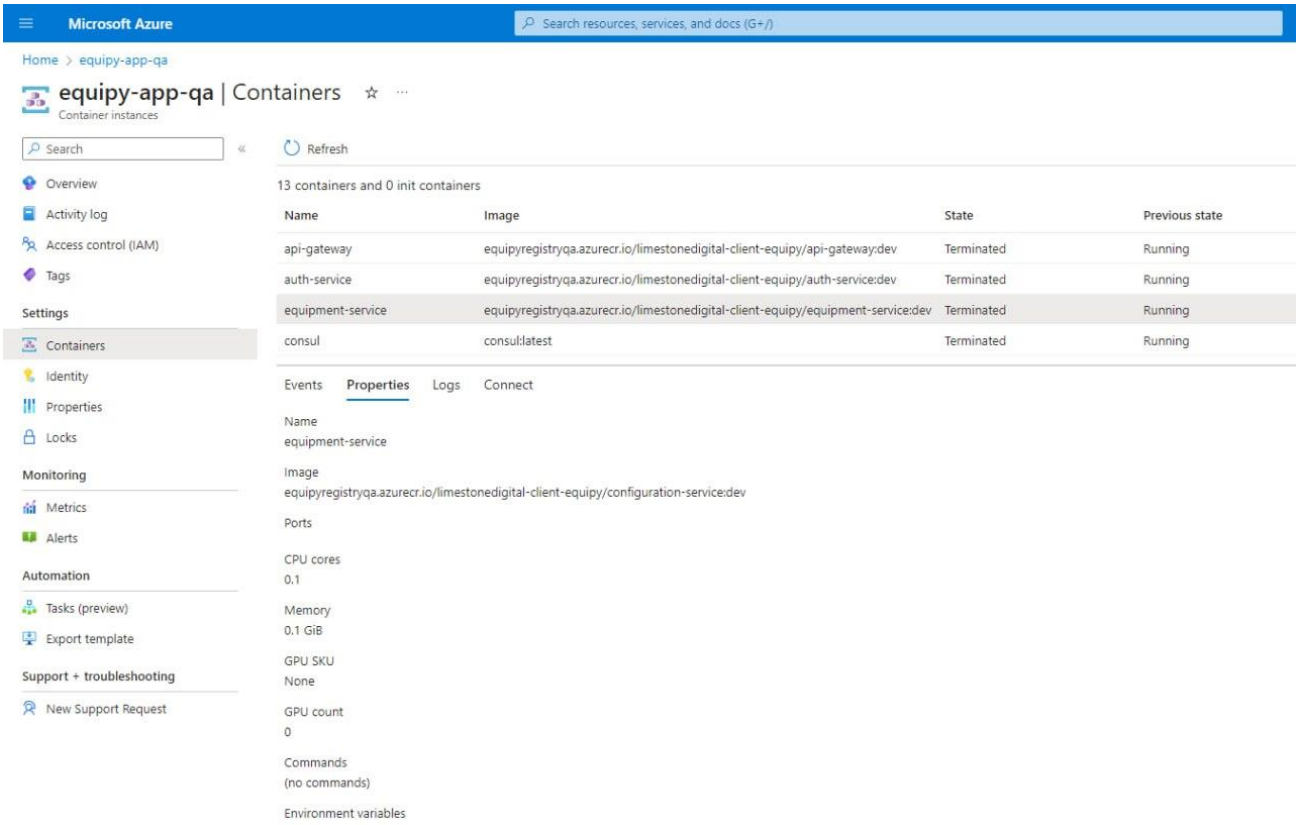


Рисунок 18 – Розгортання контейнерів Docker

В даному проєкті було також застосовано Kubernetes, отже для налаштувань переходимо до Azure Kubernetes Service. Це охоплює створення файлів конфігурації Kubernetes, таких як Deployment та Service, та застосування

цих конфігурацій за допомогою команди kubectl apply. Конфігурації показано на рисунку нижче (див.рис. 19).

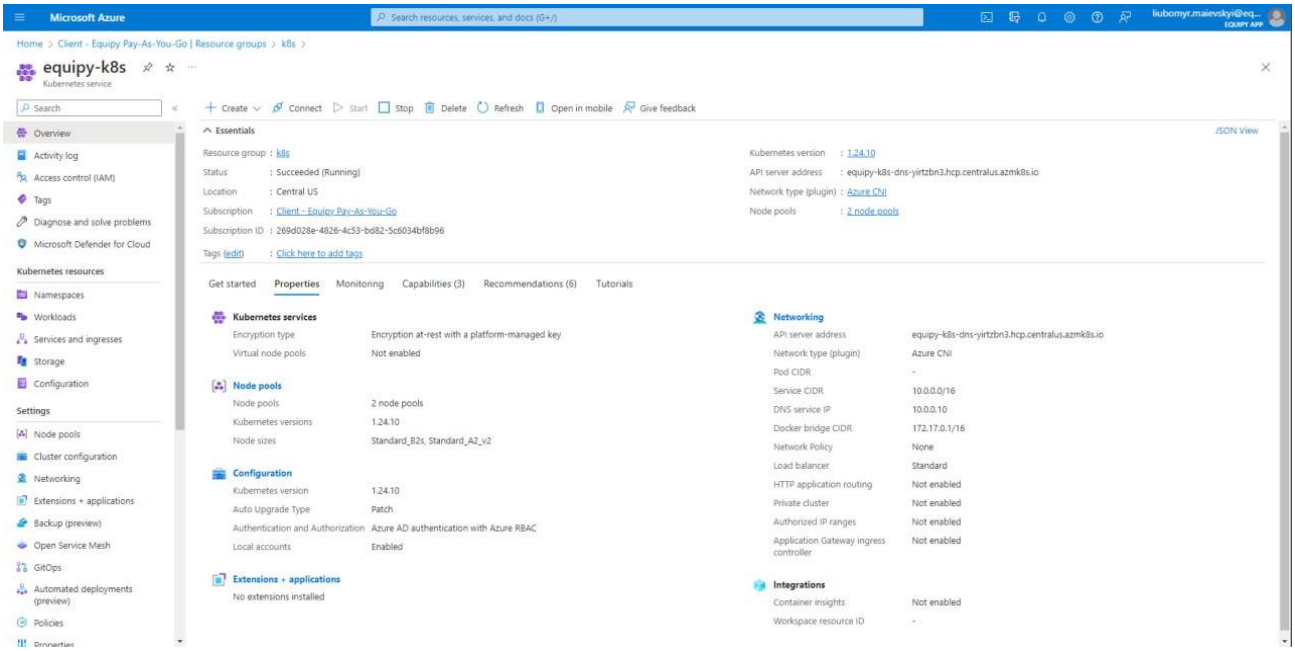


Рисунок 19 – Конфігурації Kubernetes

В результаті було такі ресурс-групи (див. рис. 20):

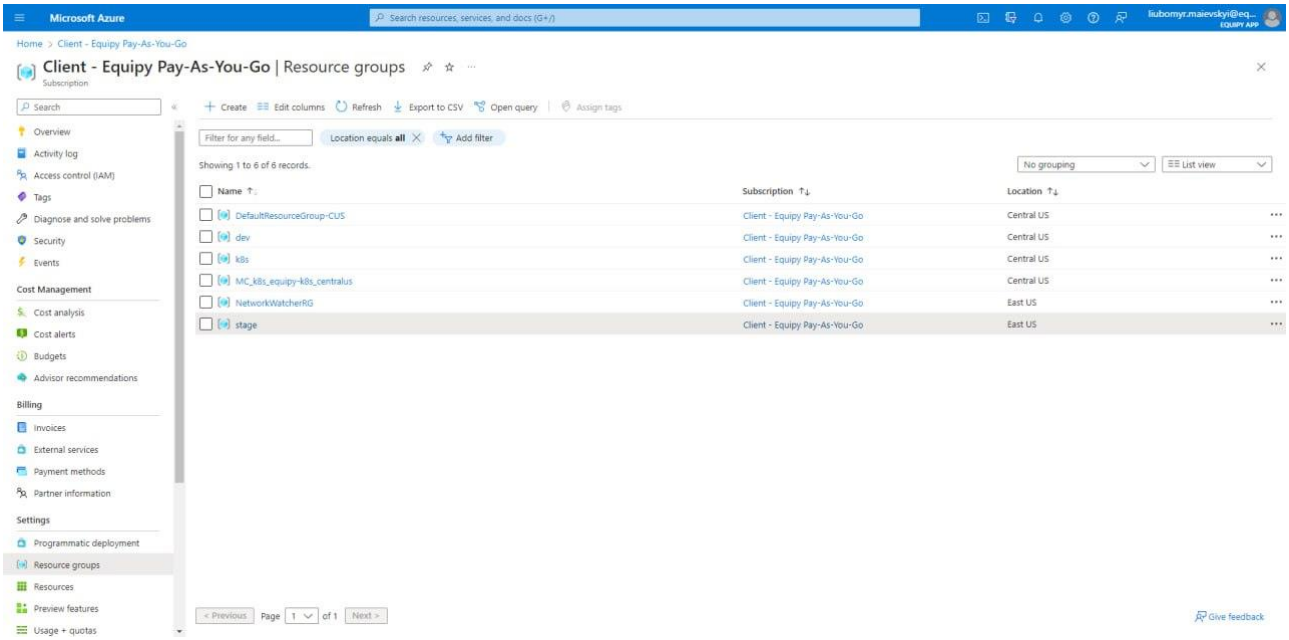


Рисунок 20 – Ресурс групи на Azure

Після цього розгорнуто усі сервіси (див. рис. 21).

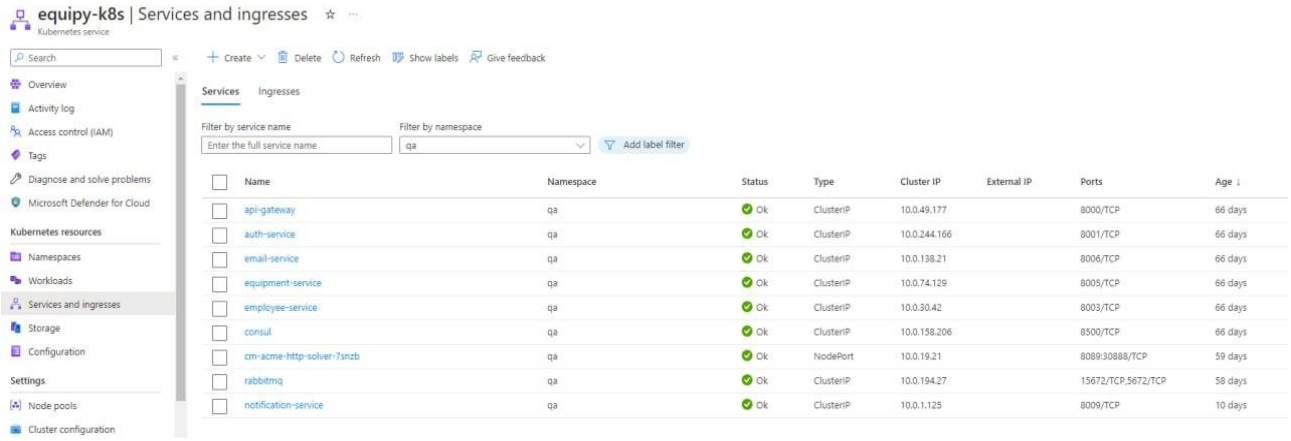


Рисунок 21 – Перелік усіх розгорнутих сервісів.

В Kubernetes Service існує два вузли: agentpool та spot. Поняття "node pools" належить до груп вузлів, які складають кластер Kubernetes. Кожен вузол являє собою віртуальну машину, яка виконує роботу вузла Kubernetes (див. рис. 22).

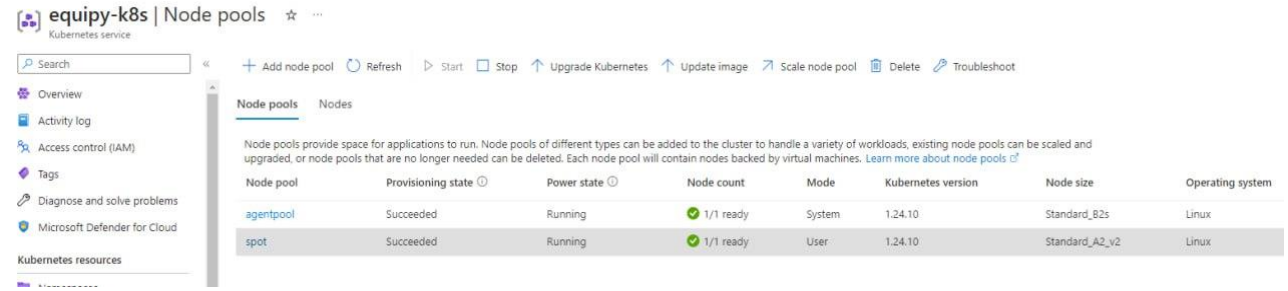


Рисунок 22 – Azure Node Pools

Створено Azure Key Vault. Це служба, яка керує ключами та секретами в хмарі Azure (див. рис. 23). Він забезпечує безпеку зберігання та управління конфіденційною інформацією, включаючи ключі шифрування, сертифікати, паролі та інше.

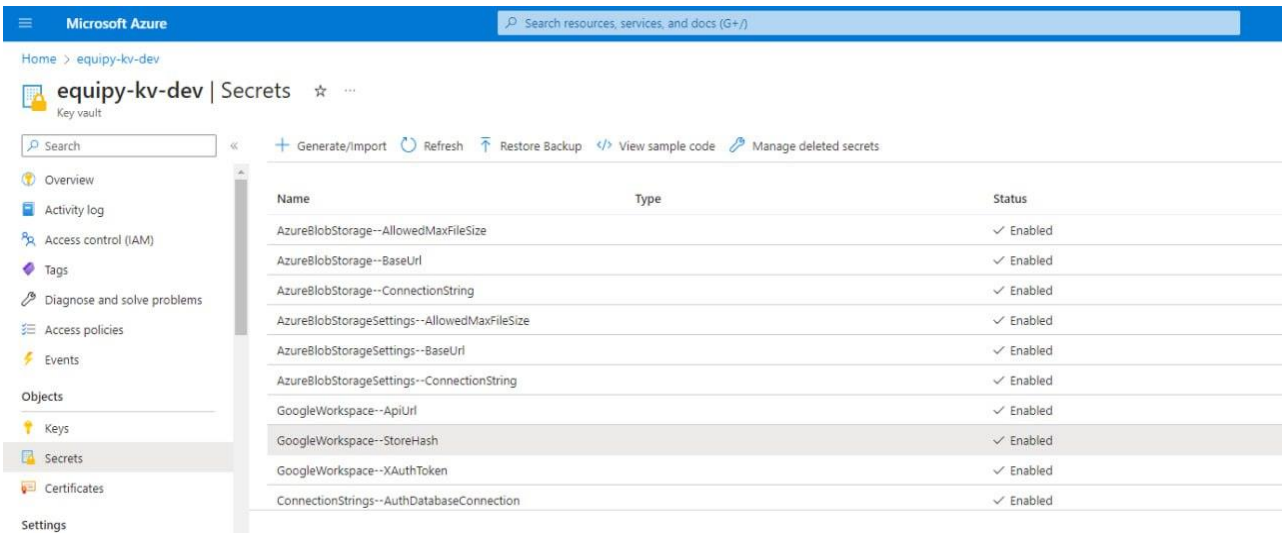


Рисунок 23 – Azure Key Vault

Окремою частиною є розгортання баз даних. Чотири бази були успішно розгорнені (див. рис. 24).

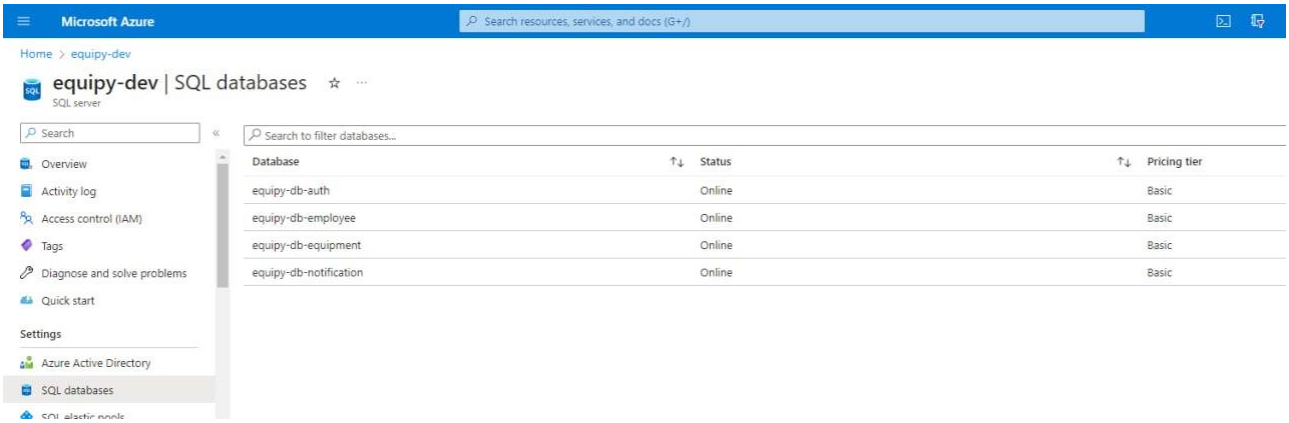


Рисунок 24 – Бази даних на Azure

Оскільки система має підтримку зберігання документів, фото тощо, створено Azure Storage Account. Це служба, яка забезпечує зберігання та керування різними типами даних, зокрема файлами, об’єктами, таблицями та чергами. Це передова система, яка зберігає, захищає та отримує доступ до даних в Azure.

Найважливіші компоненти Azure Storage Account включають:

- сховище BLOB-об’єктів: використовується для зберігання великих наборів даних, таких як зображення, відео, аудіо, документи та інші небагатовимірні об’єкти, і керування ними;

- зберігання файлів: для зберігання розподілених файлів, доступ до яких можна отримати одночасно з кількох вузлів у регіональній мережі;
- зберігання в черзі: для зберігання та керування повідомленнями в черзі, яка використовується для асинхронного зв'язку між компонентами системи;
- зберігання таблиць: для зберігання структурованих даних у формі рядків і стовпців, це полегшує швидкий доступ до великих обсягів даних;
- дискове сховище: для зберігання віртуальних дисків, які використовуються віртуальними машинами Azure.

Вигляд налаштування наведено на рис. 25.

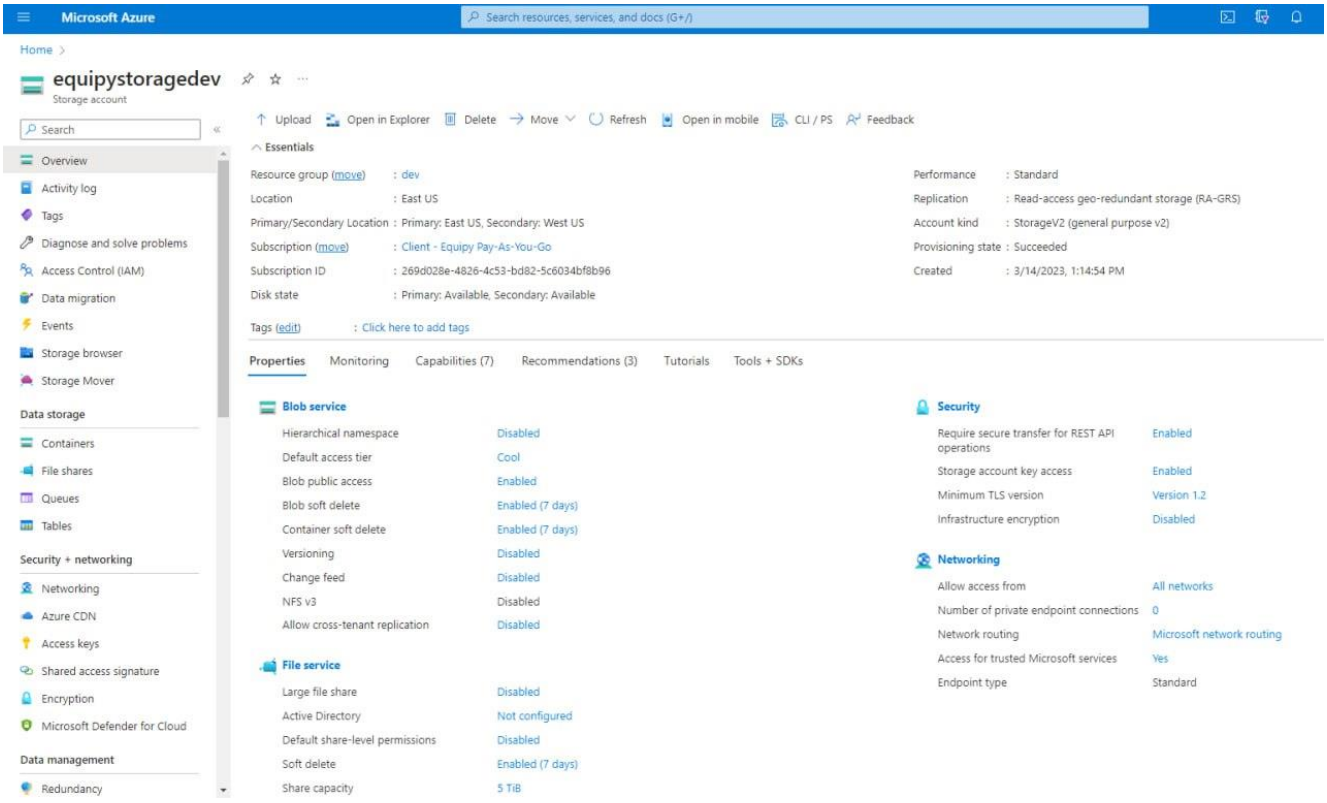


Рисунок 25 – Налаштування Azure Storage

Azure Storage має наступні контейнери:

- Employee Photos;
- Employee Documents;
- Equipment Photos.

В діаграмі архітектурного рішення, можна побачити, що веб-сайт взаємодіє із сервером за допомогою API Gateway. Для забезпечення безпеки, всі вхідні та вихідні запити проходять через мережевий екран (firewall). Сервер включає в себе модулі для різних функціональних потреб: обробки електронної пошти, управління працівниками та обладнанням, надсилання сповіщень за допомогою SignalR веб-сокетів, управління ідентифікацією користувачів та обробки запитів. Для зберігання файлів використовується спеціалізований модуль зберігання файлів. Частину загальних ресурсів представляють: система моніторингу стану (health check), Azure Functions, Key Vault та Storage Account. Всі ці компоненти, працюючи разом, забезпечують ефективну, безпечну і масштабовану роботу системи (див. рис. 26).

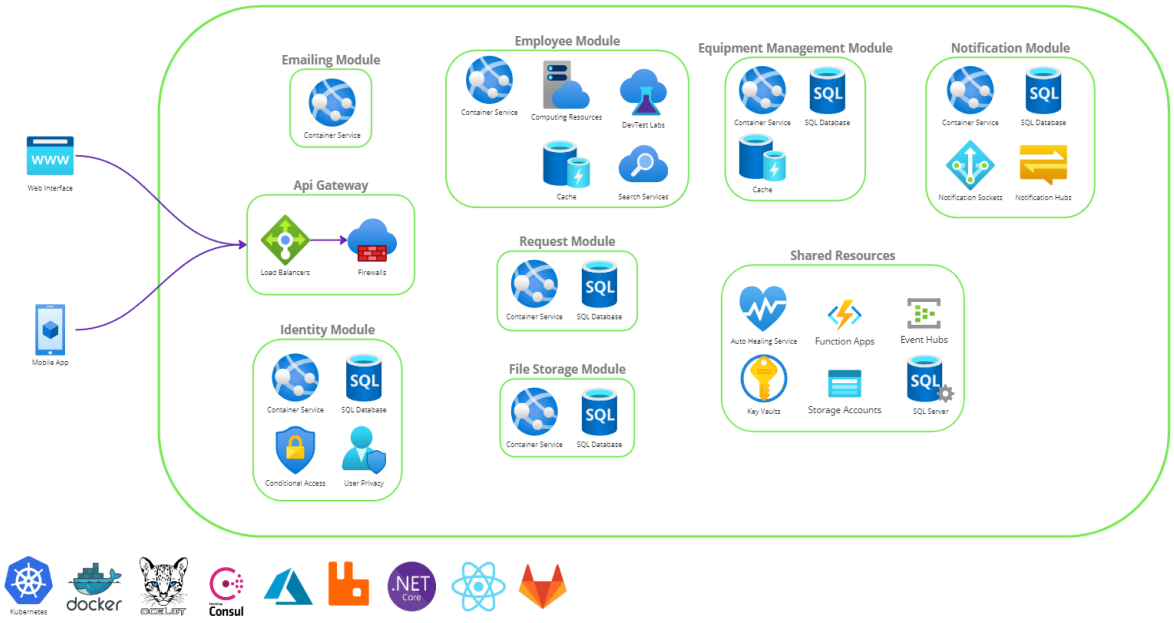


Рисунок 26 – Діаграма архітектурного рішення

3.4. Користувацький інтерфейс

Для розробки користувацького інтерфейсу було використано ReactJs та бібліотеку MUI.

Бібліотека MUI пропонує повний набір інструментів інтерфейсу користувача, які допоможуть швидше запроваджувати нові функції.

Структуру фронтенд частини можна побачити в додатку А.

Для побудови візуальної частини було створено компоненти, наведені в табл. 6.

Таблиця 6 – Опис компонентів

Компонент	**Опис**
Alert	Надає контекстні повідомлення зворотного зв’язку для типових дій користувача з кількома доступними та гнучкими попередженнями / сповіщеннями.
Avatar	Створює аватар із різними стилями. Він використовує MUI Avatar в основі.
Button	Надає різні стилі для створення кнопки. Він використовує кнопку MUI в базі.
DatePicker	Допоможе створити гарний засіб вибору дати.
Pagination	Робить гарну пагінацію списків

Крім вищезазначених компонентів, було створено Dropzone для додавання файлів компанії (див. рис. 27).

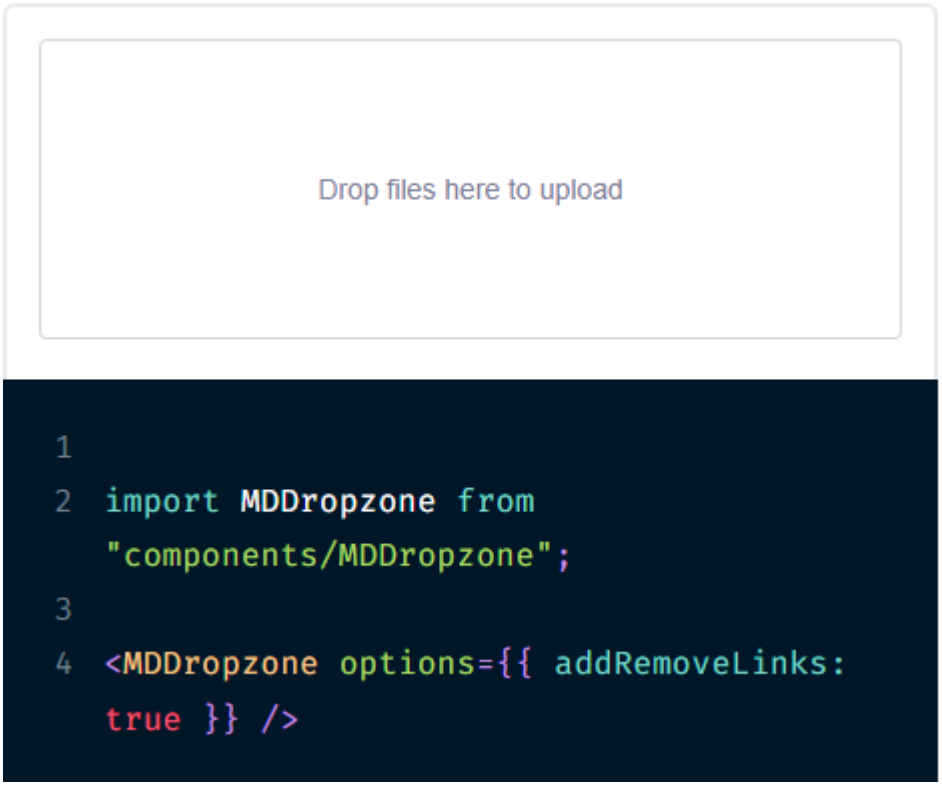


Рисунок 27 – Компонент Dropzone

Для того, щоб отримати інформацію по співробітниках було використано Employee Module, код якого наведено в додатку Б.

РОЗДІЛ 4.

ВИКОРИСТАННЯ СИСТЕМИ EQUIPY

4.1 Огляд користувацького інтерфейсу

Перейшовши по посиланню https://equipy-app.web.app відкривається сайт, де користувач може працювати з системою.

Першим кроком буде вхід в систему (див. рис. 28).

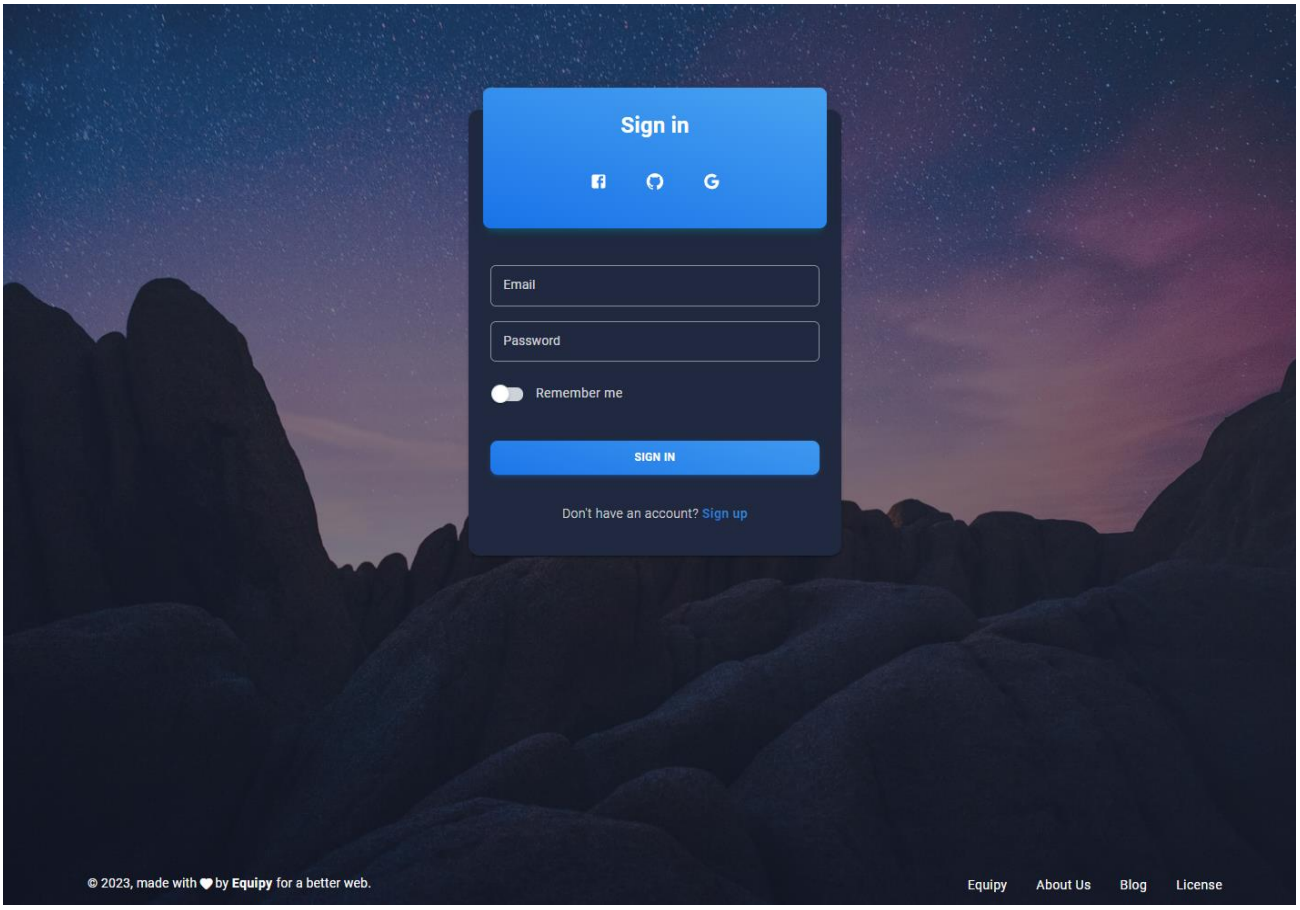


Рисунок 28 – Логін в систему

Користувач має змогу увійти в систему або зареєструватись. Є опція реєстрації через сторонні системи, такі як Google.

Після успішної авторизації, користувач, в залежності від ролі бачить систему. В якості прикладу, головною сторінкою для адміністратора буде моніторинг та статистика (див. рис. 29).

Рисунок 29 – Моніторинг витрат

При переході в меню на вкладку Employee, користувач може переглянути список усіх співробітників або створити нового (див. рис. 29 та 30).

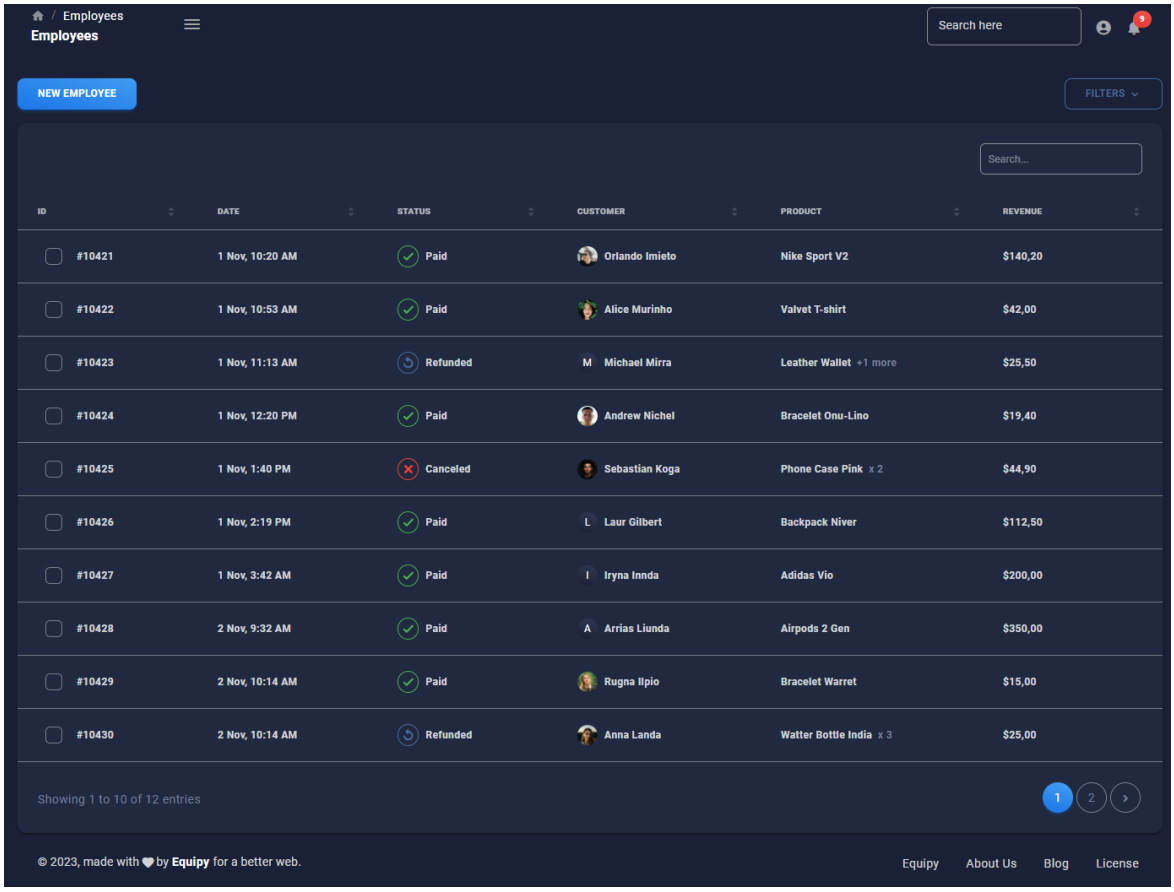


Рисунок 30 – Перегляд інформації про усіх співробітників

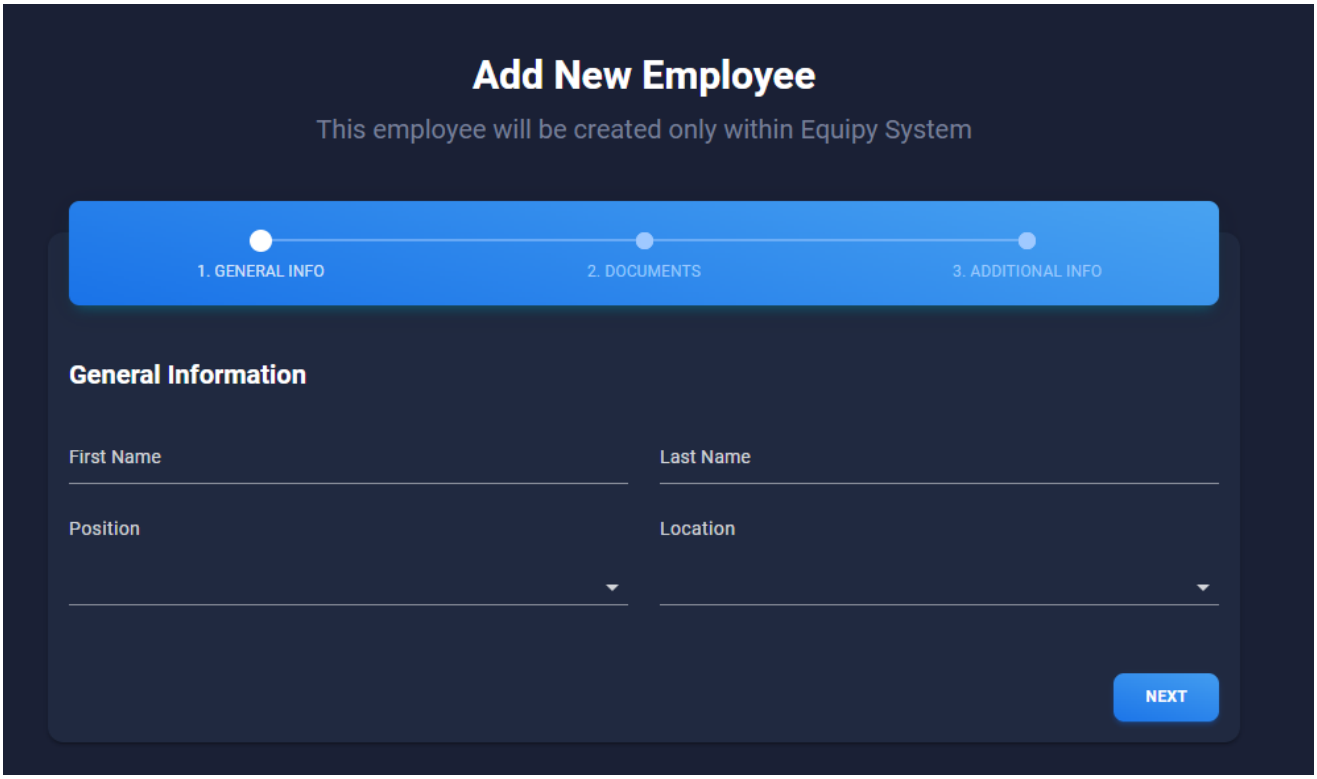


Рисунок 31 – Додавання інформації про нового співробітника

На рис. 32 наведено сторінку призначено для перегляду інформації про інвентар.

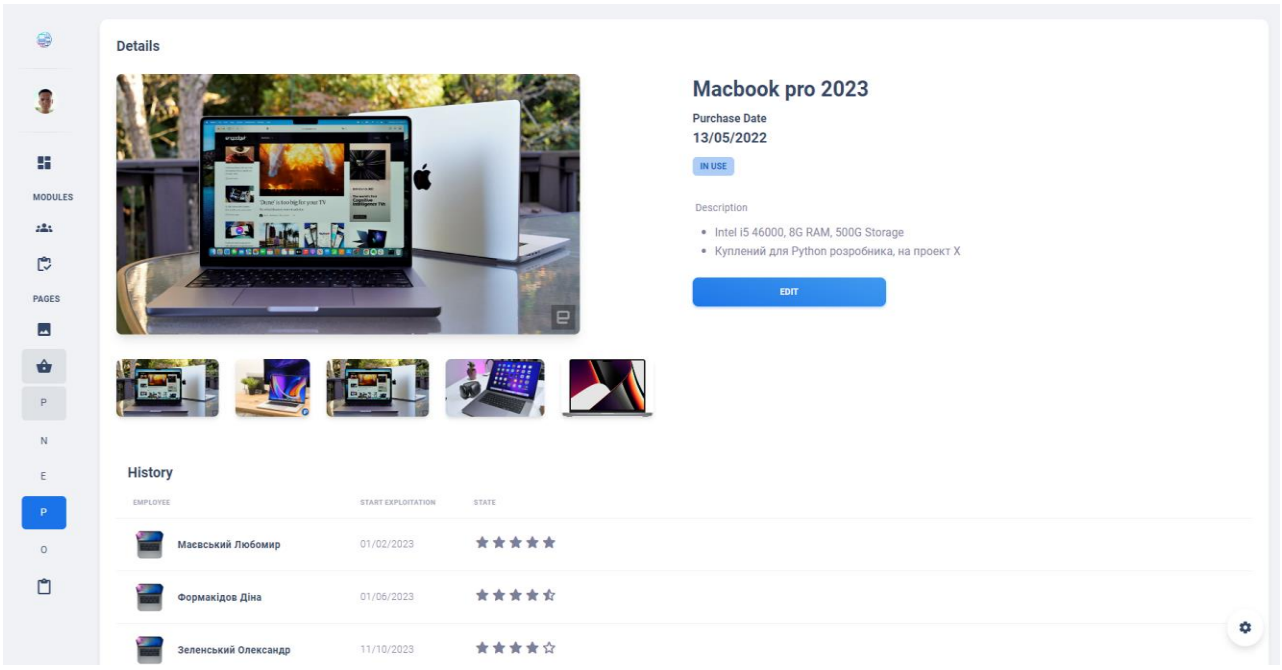


Рисунок 32 – Перегляд інформації про інвентар

На рис. 33 наведено сторінку створення запиту на придбання нового інвентарю

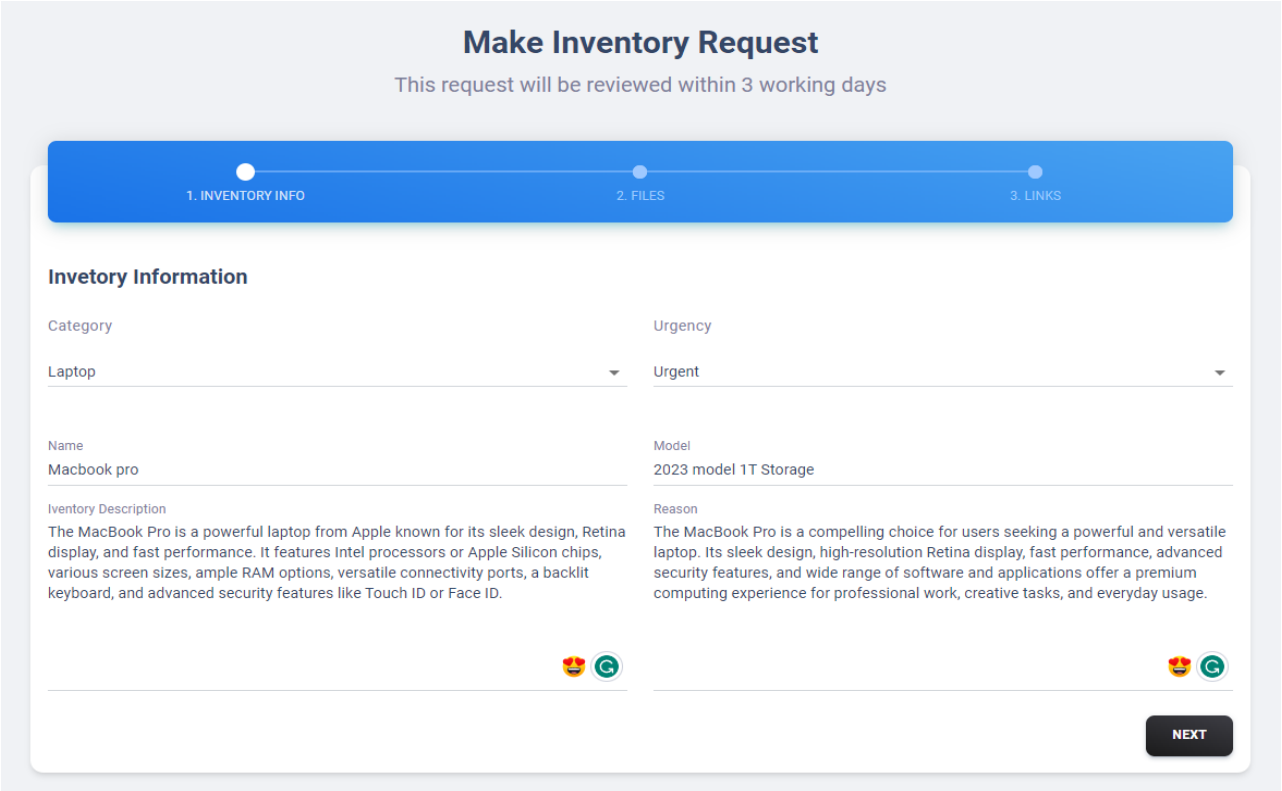


Рисунок 33 – Створення запиту на новий інвентар

4.2 Автогенерація штрих-кодів

Модуль інвентаризації "Equipy" включає в себе прогресивну функцію автоматичної генерації штрих-кодів (див. рис. 34).

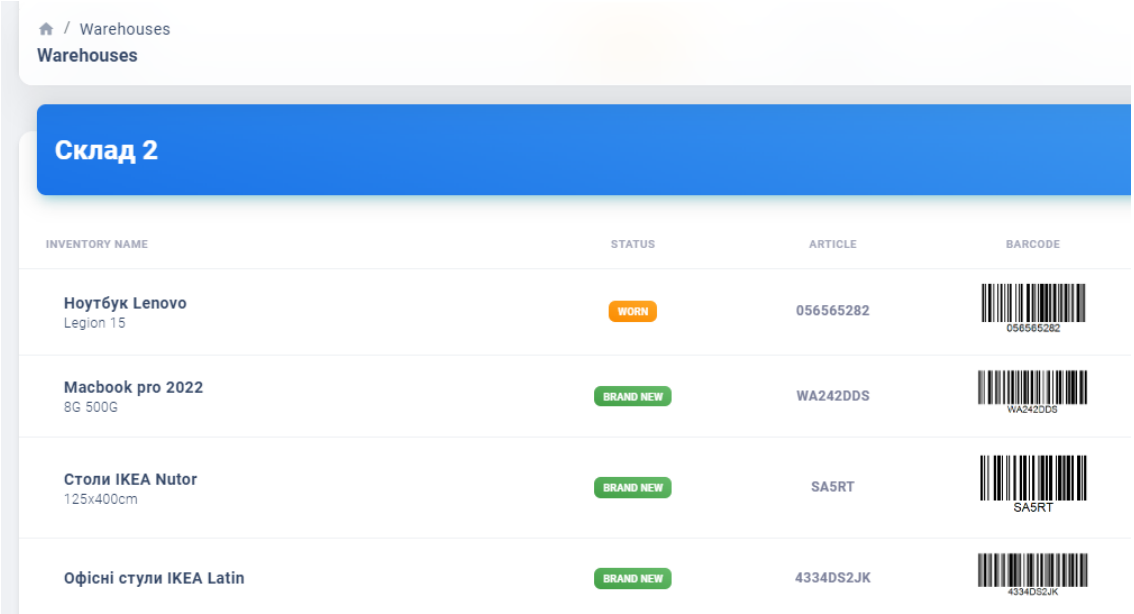


Рисунок 34 – Інвентарний список із автоматично створеними штрих-кодами

Використання цієї функції значно спрощує процес інвентаризації, оскільки генеруються унікальні штрих-коди для кожної одиниці інвентаря на основі їх артикула. Відповідно, це дозволяє забезпечити рівень контролю над обладнанням, що відповідає найвищим стандартам, а також сприяє ефективному пошуку та додаванню нового інвентаря.

Процес використання модуля є вкрай інтуїтивним та ефективним. Від користувача потрібно лише відсканувати штрих-код обладнання, після чого вся необхідна інформація миттєво відображається на екрані (див. рис. 34 та 35). Такий підхід суттєво знижує час, що витрачається на пошук конкретного об'єкта, а також дозволяє миттєво інтегрувати нові елементи в систему. Ця автоматизація процесу інвентаризації не лише покращує ефективність управління обладнанням, але і відповідає сучасним вимогам до оптимізації бізнес-процесів.

Рисунок 35 – Сканування штрих-коду

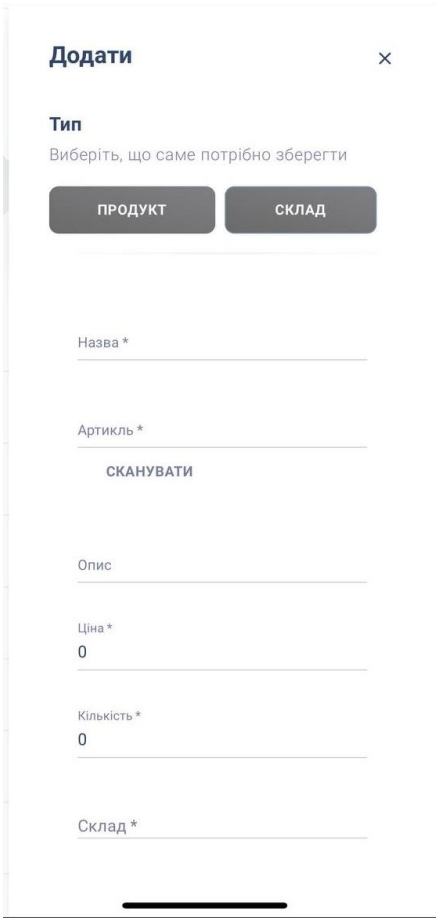


Рисунок 36 – Можливість вибрати сканування артиклю

4.3 Впровадження в компанію

Запуск програми Equipy в компанії став важливим кроком у вдосконаленні її бізнес-операцій. Ця програма була ефективно інтегрована в поточну інфраструктуру підприємства.

Нововведення цієї системи показало свою ефективність для підприємства, спрощуючи процеси інвентаризації обладнання. Система надає інтуїтивно зрозумілий інтерфейс для керування активами, а також дозволяє забезпечити прозорість та контроль всіх дій з активами.

З цього моменту, працівники підприємства можуть вести облік свого обладнання, подавати заявки на заміну або закупівлю нового обладнання безпосередньо через систему. Це полегшує процеси управління обладнанням для керівників і адміністраторів системи, дозволяючи їм оптимізувати

використання ресурсів підприємства. Довідку про впровадження наведено в додатку В.

ВИСНОВКИ

У ході виконання даної дипломної роботи було розроблено вебпрограму для інвентаризації техніки або обладнання компанії з використанням мови програмування C#. При цьому виконано наступні завдання: проаналізовано потреби ринку з точки зору потреби в інвентаризації ресурсів, затверджено план реалізації програм, спроєктовано, реалізовано, розгорнуто та впроваджено сервіс для інвентаризації устаткування.

У процесі розробки було реалізовано низку пакетів, що забезпечують роботу бекенду, які надають розширені можливості для розробки вебзастосунку. Це дозволило стандартизувати робочий процес та спростити подальше розширення функціональності програми.

Для розгортання програми було використано Docker і Kubernetes. Docker забезпечив ізольоване середовище для програми, що дозволило легко переносити та встановлювати її на різних середовищах. Kubernetes надавав можливість автоматичного масштабування програми та керування її контейнерами. Це сприяло забезпеченню стабільної та ефективної роботи програми навіть під високим навантаженням.

Фронтенд програми був розроблений з використанням ReactJS, що дозволяє створювати модульні та ефективні користувацькі інтерфейси. Завдяки використанню ReactJS, було досягнуто швидкість відклику та зручну навігацію для користувачів.

У результаті розробки даної кваліфікаційної роботи було успішно створено вебпрограму для інвентаризації техніки або обладнання компанії. Вона має зручний та інтуїтивно зрозумілий інтерфейс, що дозволяє користувачам ефективно взаємодіяти з програмою.

У майбутньому можна розглянути можливість додаткового розширення функціональності програми, наприклад, додати можливість генерації звітів або аналізу даних інвентаризації.

Результатом роботи є ефективна програма, яка спрощує процес інвентаризації техніки та обладнання в компанії, забезпечує зручний інтерфейс для користувачів та можливість розгортання на різних середовищах.

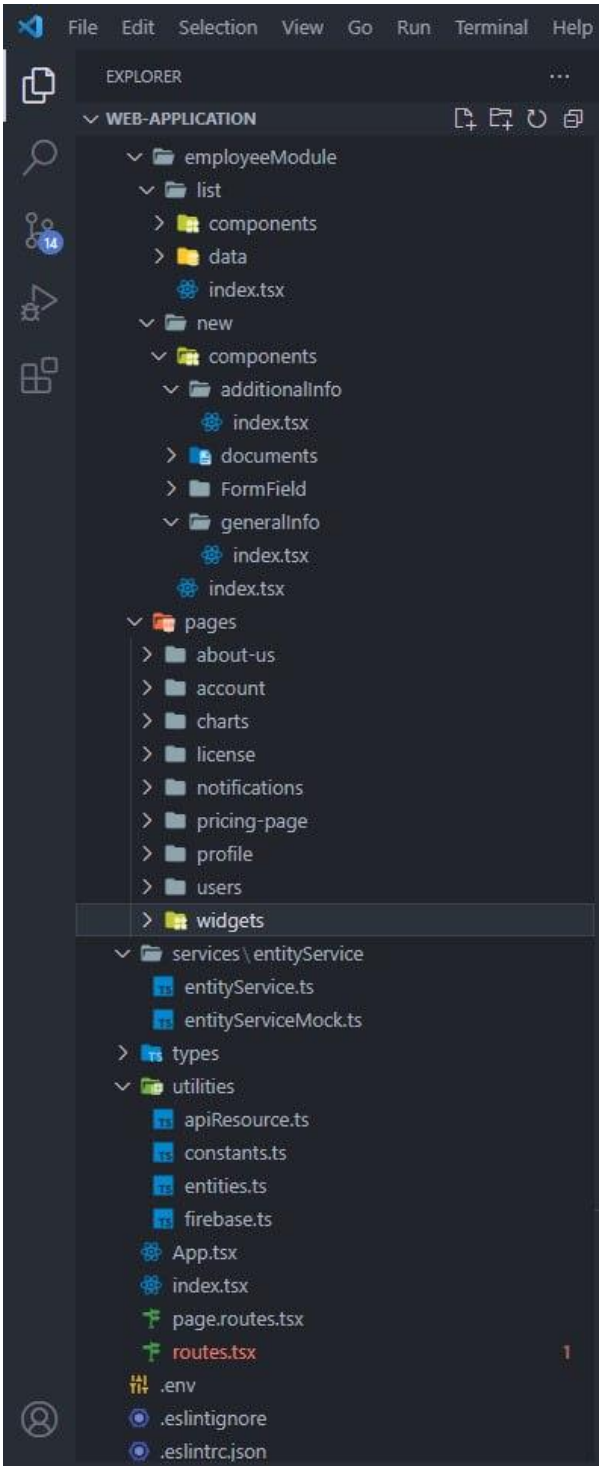
ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Артюх О. Вдосконалення процесу інвентаризації шляхом застосування комп'ютерних технологій [Електронний ресурс] / О. Артюх, Л. Фрундіна // ΛΌГОΣ. ОНЛАЙН. – 2020. – Режим доступу до ресурсу: https://ojs.ukrlogos.in.ua/index.php/2663-4139/article/view/7357.
2. Івахненков, С.В. (2010). Інформаційні технології аудиту та внутрішньогосподарського контролю в контексті світової інтеграції. Житомир: ПП «Рута».
3. Шквір, В.Д., Загородній, А.Г. & Височан, О.С. (2003). Інформаційні системи і технології в обліку. Львів: Видавництво Національного університету «Львівська політехніка».
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. Odoo Inventory [Електронний ресурс]. – 2023. – Режим доступу до ресурсу: https://www.odoo.com/app/inventory (дата звернення: 22.05.2023).
7. Comarch [Електронний ресурс]. – 2023. – Режим доступу до ресурсу: https://www.comarch.com/ (дата звернення: 22.05.2023).
8. Maarten B. Pro NuGet / Balliauw Maarten., 2012. – 338 с.
9. Theresa H. Mastering API Architecture / Howes Theresa., 2022. – 400 с. – (1).
10. Nagel C. Professional C# and .NET / Christian Nagel., 2022. – 1008 с.
11. McKendrick R. Mastering Docker / Russ McKendrick., 2020. – 568 с. – (4).
12. MsSQL [Електронний ресурс]. – 2022. – Режим доступу до ресурсу: https://metanit.com/sql/sqlserver/ (дата звернення: 22.05.2023).
13. Ritesh M. Deep-Dive Terraform on Azure: Automated Delivery and Deployment of Azure Solutions / Modi Ritesh., 2021. – 236 с.

14. Banks A. Learning React: Modern Patterns for Developing React Apps / Alex Banks., 2020. – 300 с.
15. Kubernetes [Електронний ресурс]. – 2022. – Режим доступу до ресурсу: https://kubernetes.io (дата звернення: 22.05.2023).
16. C# Exception Handling Best Practices [Електронний ресурс]. – 2022. – Режим доступу до ресурсу: https://stackify.com/csharp-exception-handling-best-practices/ (дата звернення: 22.05.2023).
17. Entity Manager [Електронний ресурс]. – 2020. – Режим доступу до ресурсу: http://breeze.github.io/doc-cs/entitymanager.html.

ДОДАТКИ

Додаток А. Структура фронтенду



Додаток Б. Фрагмент код фронтенд частини Employee

index.tsx ...\new | index.tsx ...\generalInfo | entities.ts | routes.tsx

src > layouts > employeeModule > new > index.tsx > NewEmployee

```tsx
});

function getStepContent(stepIndex: number): JSX.Element {
  switch (stepIndex) {
    case 0:
      return <GeneralInfo employee={employee} setEmployee={setEmployee} />;
    case 1:
      return <Documents />;
    case 2:
      return <AdditionalInfo employee={employee} setEmployee={setEmployee} />;
    default:
      return null;
  }
}

return (
  <DashboardLayout>
    <DashboardNavbar />
    <MDBox mt={5} mb={9}>
      <Grid container justifyContent="center">
        <Grid item xs={12} lg={8}>
          <MDBox mt={6} mb={8} textAlign="center">
            <MDBox mb={1}>
              <MDTypography variant="h3" fontWeight="bold">
                Add New Employee
              </MDTypography>
            </MDBox>
            <MDTypography variant="h5" fontWeight="regular" color="secondary">
              This employee will be created only within Equipy System
            </MDTypography>
          </MDBox>
          <Card>
            <MDBox mt={-3} mb={3} mx={2}>
              <Stepper activeStep={activeStep} alternativeLabel>
                {steps.map((label) => (
                  <Step key={label}>
                    <StepLabel>{label}</StepLabel>
                  </Step>
                ))}
              </Stepper>
            </MDBox>
            <MDBox p={2}>
              <MDBox>
                {getStepContent(activeStep)}
                <MDBox mt={3} width="100%" display="flex" justifyContent="space-between">
                  {activeStep === 0 ? (
```

Додаток В. Довідка про впровадження

Limestone Digital

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

ДОВІДКА

Видана студенту факультету комп'ютерних наук та кібернетики Київського національного університету імені Тараса Шевченка Маєвському Любомиру Андрійовичу в тому, що результати виконання його кваліфікаційної роботи бакалавра впроваджені на підприємстві «Limestone Digital s.r.o.».

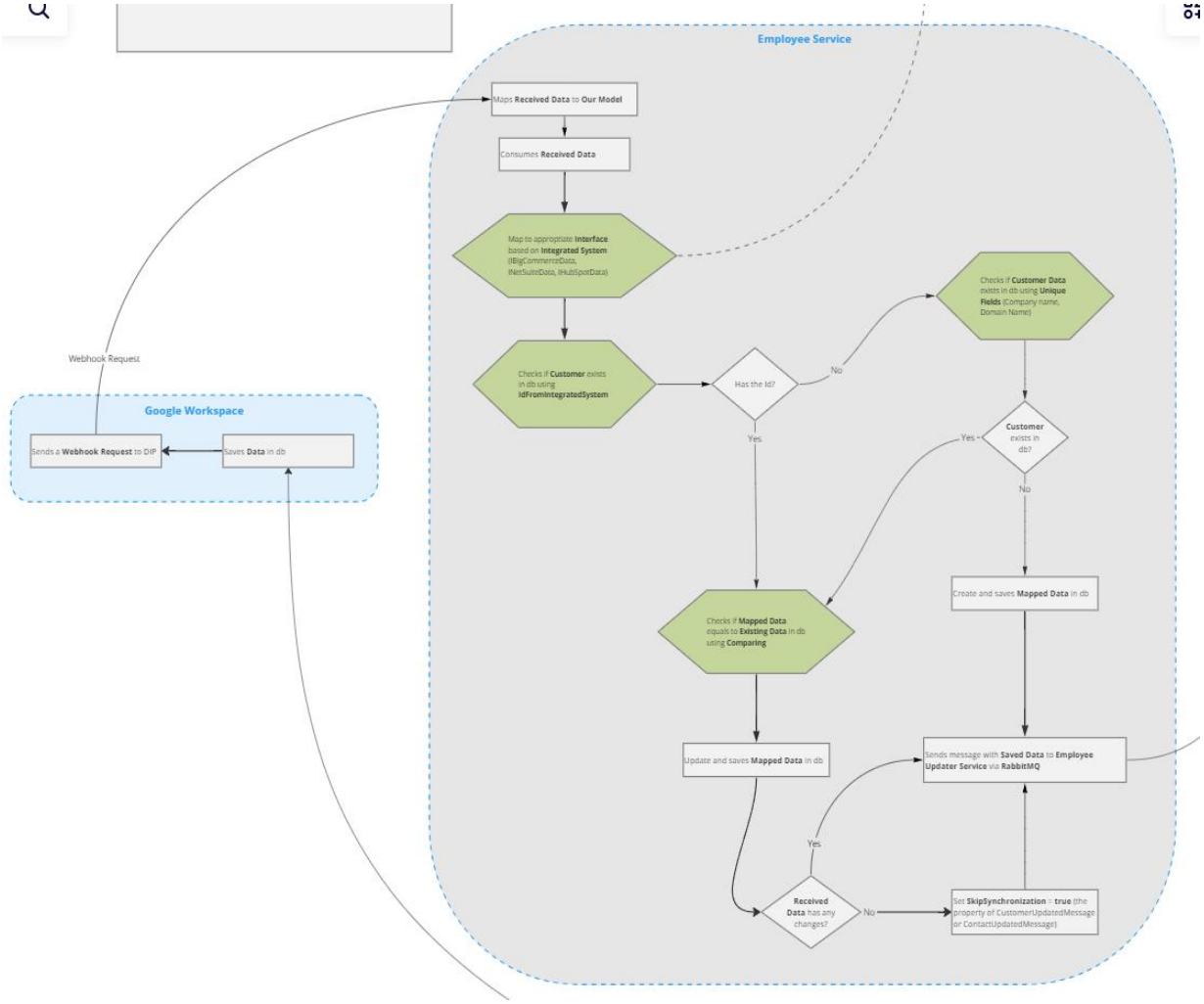
Зокрема, на підприємстві використовується запропонований веб-додаток для управління процесом інвентаризації майна та ресурсів компаній, а також можливістю нативної інтеграції із "Google Workspace" системою.

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

Посада HR manager Підпис ПІБ Пажасенко М.С.

Дата 22.05.23 Печатка

Додаток Г. Діаграма синхронізації працівників при отримані даних



Додаток Д. Діаграма синхронізації працівників при відправці даних