МІНІСТЕРСТВО ОСВІТИ ТА НАУКИ УКРАЇНИ

ЛЬВІВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
ІМЕНІ ІВАНА ФРАНКА

Факультет прикладної математики та інформатики
(повне найменування назва факультету)

кафедра інформаційних систем
(повна назва кафедри)

ДИПЛОМНА РОБОТА

на тему:

Інформаційна технологія віддаленої комунікації
медичного закладу з пацієнтами
на основі мікросервісної архітектури

Виконали: студенти групи ПМІ-44
спеціальності 122 – комп'ютерні науки
(шифр і назва спеціальності)

Довганич М.А.
(підпис) (прізвище та ініціали)

Копина І.І.
(підпис) (прізвище та ініціали)

Керівник Соколовський Я.І.
(підпис) (прізвище та ініціали)

Рецензент _____
(підпис) (прізвище та ініціали)

Львів – 2023

ЗМІСТ

ЗМІСТ 2
ВСТУП 3
АНОТАЦІЯ 4
АКТУАЛЬНІСТЬ 5
ТЕХНІЧНЕ ЗАВДАННЯ 6
Об’єкт досліджень 7
Предмет досліджень 7
Мета роботи 7
Виконані завдання для поставленої мети 7
Практична значимість роботи 11
РОЗДІЛ 1 12
1.1 Аналіз існуючих аналогів 12
1.2 Аналіз існуючих технологій 14
1.3 Висновок 22
РОЗДІЛ 2 23
2.1 Архітектура проекту 23
2.1 UML діаграми проекту 26
2.2 Висновок 29
РОЗДІЛ 3 30
3.1 Математичне забезпечення 30
3.2 Висновок 33
РОЗДІЛ 4 34
4.1 Розробка програмного забезпечення 34
4.2 Структура бази даних 55
ВИСНОВКИ 56
СПИСОК ЛІТЕРАТУРИ 57
ДОДАТКИ 58

ВСТУП

З кожним роком світ все більше змінюється під впливом нових сучасних технологій. Багато речей, які зараз є для нас буденністю, були чимось незвичайним та здавались зовсім ненадійними у минулому. Це говорить про те, що багато відомих нам методів, які зараз використовуються можуть бути кардинально змінені. Важко не згадати немаючий аналогів додаток «Дія», який зміг відкрити нові полегшені можливості для зберігання важливих документів. Таким самим чином можна полегшити роботу людей в зовсім різних галузях праці.

Однією з таких галузей може бути медицина. В більшості випадків інформація про пацієнтів, їхні аналізи та історія хвороб зберігається в архіві або примітивній базі даних. Буде важко не погодитись, що такі способи збереження даних є не надійними і більше того – не зручними. Окрім цього, процес запису та обміну даними є досить не практичний, не говорячи уже про безпеку таких дій. Було б набагато простіше, якщо б існував спосіб швидко записувати дані у комп’ютер та правильно їх структурувати. А ще краще – створити спосіб, яким медичні працівники могли б обмінюватись даними між собою. Таке рішення проблеми представляло б собою мініатюрну соціальну мережу, головна задача якої – це забезпечення легкого способу передачі та збереження даних.

Суть дипломної – це зробити застосунок, який може об’єднати в собі усі вище сказані критерії та стати наступним кроком у оцифруванні медичних закладів. З його допомогою медичні працівники та вище керівництво лікарні зможуть значно полегшити свою роботу. В той же час процес запису до лікарні клієнтом стане набагато простішим, що без сумніву подарує позитивні емоції і гарне враження про медичний заклад.

АНОТАЦІЯ

Дипломна робота студентів Довганича Маркіяна та Копини Іллі на тему: «Інформаційна технологія віддаленої комунікації медичного закладу з пацієнтами на основі мікросрвісної архітектури» містить 35 сторінок, 3 додатки та 13 джерел за списком використаної літератури.

У дипломній роботі розглянуто сучасні алгоритми та підходи до розв'язування проблеми комунікації між медичним закладом та його клієнтами. Описані проблеми реалізації та можливі вирішення задач.

Уся інформація, яка була застосована для демонстрації новин медичного закладу була узяти з різних відкритих джерел і відношення до якогось певного медичного закладу немає. Також інформація про робітників та пацієнтів є вигаданою та немає ніякого відношення до справжніх осіб. Окрім того, фотографії, які застосовувались в дипломній роботі були взяті з мережі Інтернет і не мали ніякої правової цінності.

ABSTRACT

This is a diploma of Markian Dovganich and Kopyna Illia on the topic: "Information technology of remote communication of a medical institution with patients based on microservices architecture" that contains 35 pages, 3 addition items and 13 sources from the list of used literature.

The diploma considers modern algorithms and approaches to solving the problem of communication between the medical institution and its clients. Problems of realization and possible solutions to problems are described.

All the information that was used to demonstrate the news of the medical institution has been taken from various open sources and is not related to any particular medical institution. Also, information about workers and patients is fictitious and has nothing to do with real people. In addition, the photos used in the diploma have been taken from the Internet and had no legal value.

АКТУАЛЬНІСТЬ

Згідно з словами відомого американського лікаря Рубіна-Сеймора Купера: «Людина повинна відвідувати лікаря хоча б один раз в два тижні, щоб точно вважати себе здоровою».[1] З цих слів стає зрозуміло, що процес запису до лікаря, процес знаходження підходящого часу для зустрічі та ще багато інших деталей потрібно узгоджувати досить часто. Саме тому моя робота є як ніколи актуальною і більше того, вона дасть можливість ще більше розвиватись у області діджиталізації.

Діджиталізація має надзвичайну актуальність у сучасному світі. Завдяки ній, бізнеси можуть ефективно взаємодіяти зі споживачами, привертати їх увагу та задовольняти їх потреби через цифрові канали комунікацій. Таким чином, це дає можливість бізнесам досягати широкої аудиторії та збільшувати популярність торгівлі, оскільки вона забезпечує зручний доступ до послуг. Компанії, які не використовують можливість діджиталізації, ризикують відстати від конкурентів та втратити зв’язок з сучасними споживачами. А тим, хто використовує ці можливості, залишається боротись за продуктивність. А невід’ємною частиною цього процесу є вибір правильної архітектури програмного забезпечення. Адже правильно вибрана архітектура дозволить бізнесам грамотно оптимізувати свої процеси, підвищити ефективність та зменшити витрати за допомогою використання оптимальних цифрових інструментів. Тому, це є необхідним кроком для розвитку бізнесу, а також для задоволення потреб користувачів.

ТЕХНІЧНЕ ЗАВДАННЯ

Завдання – це створити веб застосунок на базі мікросервісної архітектури для лікарні, в якій можна об’єднати різний функціонал для збереження, створення та обробки інформації про медичних працівників та пацієнтів. Окрім цього веб застосунок має дати можливість швидко і, найголовніше – зручно записуватись на прийом до лікаря. Варто зазначити, що додаток буде мати можливість приймати та обробляти заявки на пошук сімейних лікарів. Також він має забезпечувати безпеку та конфіденційність усієї інформації згідно з правилами HIPAA.[2] І саме тому застосунок повинен включати в себе засоби запиту та підтвердження надання певної інформації про пацієнта. Іншими словами лікарі будуть робити запит на отримання додаткової інформації відносно історії хвороб пацієнта, а сімейний лікар, в свою чергу, буде надавати або відхиляти цей запит. До того ж застосунок має мати зручний спосіб пошуку медичних працівників та пацієнтів. В більшості випадків у лікарні є фармацевтична компанія з якою вони співпрацюють і тому потрібно організувати зручний та надійний спосіб передачі певних документів фармацевтам, наприклад рецептів з назвою відповідних ліків. Саме тому потрібно також створити систему пошуку відповідних ліків. І звісно ліки, як і лікарі, будуть мати свій рейтинг, який буде складатися з відгуків інших користувачів сайту. Для зручності веб застосунок повинен давати можливість створення і друк певних документів по визначеному шаблону у PDF форматі. Варто також зазначити, що він має бути готовий до використання багатьма людьми одночасно. Тому, застосунок повинен бути надійним, безпечним та найголовніше – продуктивним.

Об’єкт досліджень

Об’єкт досліджень – структура та процеси роботи медичного закладу, мікросервісна архітектури для розробки інформаційних систем.

Предмет досліджень

Предмет досліджень – способи вирішення проблем віддаленої комунікації між медичними працівниками та пацієнтами з використанням застосунку на базі мікросервісної архітектури

Мета роботи

Мета роботи – створити платформу на базі мікросервісної архітектури, яка допоможе налагодити зручну комунікацію між різними процесами медичного закладу.

Виконані завдання для поставленої мети

Завдання виконані разом:

- Аналіз існуючих інформаційних систем віддаленої комунікації медичних закладів з клієнтами.

- Аналіз існуючих технологій для розробки веб застосунків.

- Аналіз існуючих архітектурних рішень для програмного забезпечення.

- Аналіз існуючих хмарних платформ та рішень.

- Організувати розширювальну модель та процес розробки додатку.

- Розробити архітектуру програмного забезпечення.

- Забезпечити можливість збереження загальної інформації про пацієнта (електронна медична карта) та збереження усіх відвідувань пацієнта з описом лікування та діагностик, які проводились (медичні записи). З медичних записів будується медична історія пацієнта.

- Забезпечити планування зустрічей. Тобто - можливість для пацієнта записатись до потрібного лікаря та можливість для лікаря прийняти/відхилити запис від пацієнта (також лікар і пацієнт мають мати можливість бачити календар з подіями у своєму кабінеті). До цього пункту також можна віднести і створення відповідних направлень, які пізніше будуть додані до медичного запису.

- Забезпечити можливість передачі даних пацієнтів між різними медичними працівниками (лікарі, спеціалісти тощо). Це повинно робитися за допомогою передачі відповідного медичного запису іншому працівнику.

- Забезпечити можливість медичному працівникові робити запит діагностичних тестів, методів лікування та іншої інформацію з Медичної історії пацієнта (лікар робить запит, який має бути прийнятий сімейним лікарем).

- Забезпечити доступ до варіантів виписування, друк рецептів пацієнтам, а іноді й електронна передача рецептів від лікарів фармацевтам (цієї самої лікарні).

- Забезпечити надання реабілітаційних послуг на відстані за допомогою комунікації у чаті (також чат можна використовувати для того щоб владнати деякі деталі з лікарем, наприклад запитати де саме знаходиться кабінет або чи потрібно приходити натощак).

Завдання виконані Довганичом М.А.:

- Створити реляційну базу даних для зберігання інформації про медичний заклад та пацієнтів.

- Розробити незалежний сервіс, який буде керувати загальною інформацією пов’язаною з застосунком.

- Розробити незалежний сервіс, який буде керувати інформацією пов’язаною з лікарями.

- Розробити незалежний сервіс, який буде керувати інформацією пов’язаною з пацієнтами.

- Розробити незалежний сервіс, який буде керувати інформацією пов’язаною з постачальниками медикаментів.

- Розробити функціональність автентифікації користувачів на основі асиметричного шифрування.

- Розробити функціональність авторизації користувачів на основі ролей.

- Забезпечити надійність проекту за допомогою інтеграційних та модульних тестів.

- Забезпечити можливість налагодження проекту за допомогою API клієнту для можливості подальшої розробки та підтримки проекту.

- Підключити сервіс для надсилання email листів.

Завдання виконані Копиною І.І.:

- Налаштувати хмарний сервіс, на якому будуть розгортатись усі компоненти проекту.

- Налаштувати сховище для зберігання бінарних файлів.

- Налаштувати комунікацію між сервісами в межах хмарної інфраструктури.

- Налаштувати процес неперервної інтеграції та розгортання усіх сервісів.

- Налаштувати горизонтальна та вертикальне масштабування усіх компонентів проекту в межах хмарної інфраструктури.

- Налаштувати систему моніторингу працездатності усіх компонентів проекту.

- Налаштувати кешування запитів до веб сервісу.

- Створити клієнтську частину для доступу до головного функціоналу додатка.

- Додати можливість вибору між українською та англійською мовами.

- Забезпечити легку систему пошуку та фільтрації пацієнтів, лікарів та лікарств.

Практична значимість роботи

Безсумнівно дана робота буде дуже хорошим доповненням для будь якого медичного закладу і буде мати багато значимої користі для всіх її учасників.

Для пацієнта – це буде простий і водночас зручний спосіб комунікації з медичним закладом, а також його лікарями. Окрім цього, процес отримання ліків в фармацевта буде значно спрощений за рахунок того, що пацієнту не потрібно буде запам’ятовувати або пояснювати ліки, які йому потрібні.

Для лікаря – це перш за все простота отримання і заповнення медичних документів. Також лікарі зможуть значно простіше організовувати всі події на своїй роботі. А також це хороший спосіб знайти клієнтів, які потребують сімейного лікаря.

Для фармацевтів – зручний та легкий спосіб поставки товару медичному закладу та його клієнтам. Це допоможе збільшити кількість продажів тих чи інших фармацевтичних товарів та покращити контроль над ними.

Для власника медичного закладу – це чудовий спосіб збільшити кількість клієнтів, а це означає, що і збільшити прибуток. За рахунок зручного способу комунікації з фармацевтами, буде значно легше знайти бізнес партнерів, які хотіли б приєднати свою фармацевтичну компанію до медичного закладу. Також в подальшому проект можна покращити і додати багато функцій, які будуть спрощувати всю систему роботи, яка зараз робиться вручну і вимагає багато затраченого часу.

РОЗДІЛ 1

1.1 Аналіз існуючих аналогів

Точних аналогів роботи – немає. Серед найбільш подібних аналогів можна назвати комп'ютерні додатки від AdvancedMD та Dentist+. [3][4] Обидва додатки спрямовані на те, щоб полегшити роботу збору та структурування інформації про пацієнта. Проте AdvancedMD також застосовує свій додаток для того, щоб різні лікарі могли аналізувати результати обстежень пацієнта і давати один спільний вердикт. Іншими словами цей додаток спрямований більше на віддалене лікування пацієнта. Звісно, цей додаток також має можливість друку PDF файлів і також дає можливість лікарю заповняти відповідні довідки, які стосуються пацієнта, але по суті сам пацієнт не має доступу до цієї програми, що означає, що додаток був зроблений лише для лікарів.

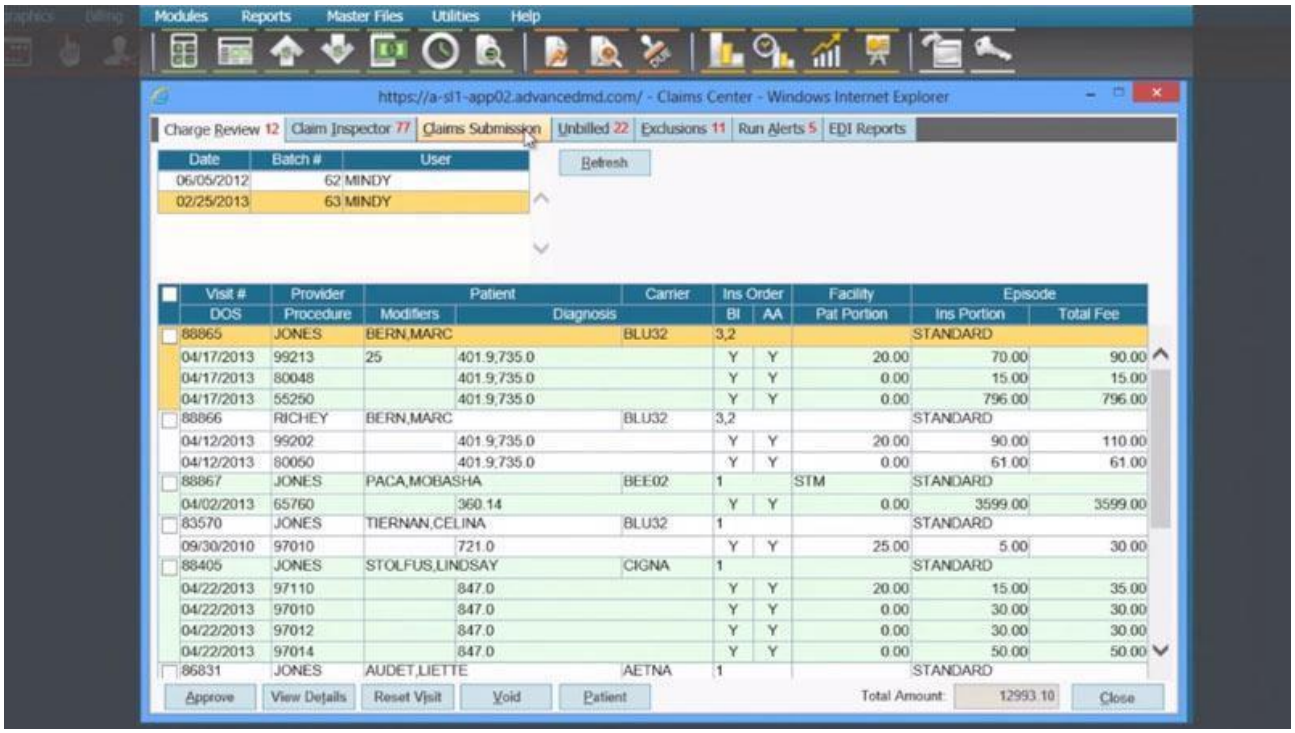

Рисунок 1 Програма AdvancedMD

Подібна ситуація і з Dentist+. Ця програма дає можливість зручно та швидко заповняти карту пацієнта, яку пізніше можна передавати між різними медичними працівниками. Вона дозволяє лікарям створювати і організовувати календар подій, де можна планувати зустрічі і не тільки. Великою перевагою цієї програми є те, що вона також веде фінансовий рахунок для медичних працівників, що справді вражає. Але сама програма ніяк не «контактує» з самим клієнтом, що знову ж таки робить цю програму корисною тільки для медичного персоналу. Саме тому участь пацієнта у цьому процесі потрібно робити за аналогією усім знайомої процедури – покупці товару в інтернет магазині, але замість товарів – лікарі. Також варто зазначити, що багато функціоналу, який використовується у цих рішеннях можна тим чи іншим способом перетворити на функціонал для пацієнтів. Для цього знадобиться багато зусиль, але використання уже продуманих рішень точно спростить загальну задачу та зробить її більш надійною.

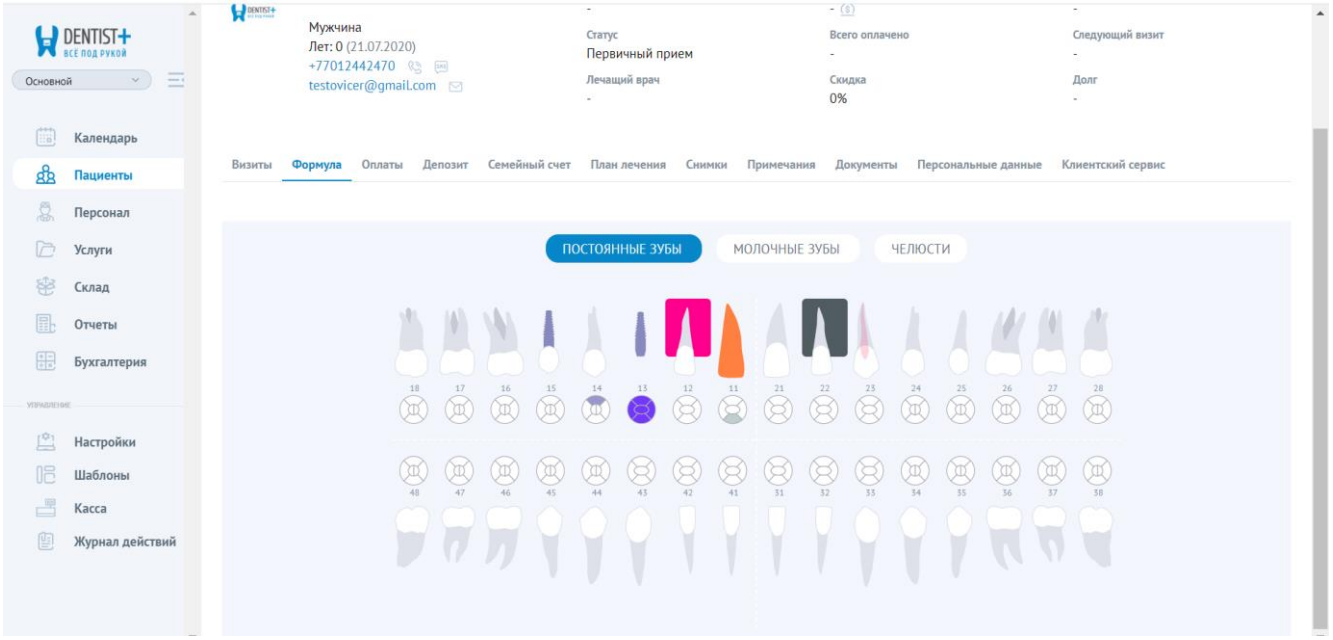

Рисунок 1.1 Програма Dentist+

Звісно, також існує безліч інших програм, суть яких зберігати в базі даних введену лікарем інформацію. Саме такі додатки здебільшого і застосовуються у різних приватних лікарнях. Такий спосіб збереження інформації є значно кращим за зберігання даних в паперовому вигляді, але все таки не є на достатньому рівні, щоб конкурувати з тими ж AdvancedMD чи Dentist+.

1.2 Аналіз існуючих технологій

Для розробки веб застосунку був обраний один з найпопулярніших шаблонів програмного забезпечення, а саме поділ веб застосунку на backend та frontend компоненти. Іншими словами, перший компонент відповідає за структурування даних та надсилання їх другій частині, а друга частина відповідає за використання цих даних в готовій сторінці, яка і відображається користувачу. Один з плюсів використання цього шаблону в тому, що ці два компоненти працюють практично повністю незалежно один від одного і тому для кожного з компонентів можна використовувати будь яку технологію не боячись, що вони можуть між собою конфліктувати. Також варто згадати, що для збереження інформації потрібно використовувати базу даних, яких є дуже багато. Окрім цього, потрібно обрати сервіс, який вміє зберігати бінарні файли та давати швидкий доступ до них. І не варто забувати про хмарне середовищем та про вибір певного постачальника цих хмарних послуг. Почнемо оглядати можливі варіанти кожного з компонентів.

Backend частина була побудована на базі мікросервісної архітектури. Проте давайте розглянемо не менш популярну альтернативу, а саме монолітну архітектуру. Мікросервісна архітектура має декілька ключових переваг над іншими типами архітектур розробки програмного забезпечення, включно з монолітом, нижче наведені деякі її переваги.

- Модульність та розшарування: Додаток, створений на основі мікросервісної архітектури, може бути розбитий на окремі сервіси, які є функціонально та логічно незалежні один від одного. Це дозволяє зосередитись на розробці кожного сервісу окремо, а також зменшує взаємопов'язаність та взаємозалежність між компонентами (модулями), що в свою чергу полегшує тестування, підтримуваність та, в перспективі, розширення аплікації.

- Гнучкість у виділенні ресурсів та масштабованість: Мікросервісна архітектура дає можливість горизонтально та вертикально масштабувати окремі модулі, в залежності від того, як вони навантажені. Вертикальне масштабування надає змогу збільшувати потужність тих сервісів, які вимагають більшого обсягу ресурсів (потужність процесора чи обсягу оперативної пам’яті), а горизонтальне масштабування надає змогу збільшити кількість одиниць кожного сервісу. Така гнучкість забезпечує ефективне використання ресурсів та зменшує витрати коштів для оплати послуг хмарного середовища, де розгорнута аплікація.

- Розгортання: Кожен сервіс додатку може бути розгорнутий незалежною. Така поведінка надає змогу швидко внести зміни в окремі модулі медичного додатку, та розгорнути їх без впливу на інші сервіси додатку. В результаті, розробка нової функціональності додатку або виправлень помилок можуть бути проведені без зупинки або ризику поломки інших сервісів.

- Тестування додатку: В силу того, що модулі аплікації є незалежні між собою, вони можуть бути окремо відтестовані за допомогою інтеграційних тестів та/або ручного тестування за допомогою API клієнтів.

- Документація: Через розподіленість архітектури, кожен модуль є відповідальний лише за одну групу завдань, тому це спрощує створення та оновлення документації системи.

Та все ж, попри вище наведені переваги мікросервісної архітектури, вона має недоліки, які слід враховувати при створенні архітектури додатку, а саме:

- Складність управління та оркестрування системи: При використанні багатьох мікросервісів, управління та координація між ними значно ускладняється у порівнянні із монолітною архітектурою, де є лише один-два сервіси. Окрім цього, такі процеси, як синхронізація між компонентами, логування, моніторинг, налаштування та комунікація між модулями аплікації значно ускладняються через взаємозалежності між сервісами та розподіленість системи.
- Мережеві затримки та надійність: Система, яка є побудована на основі мікросервісної архітектури, має більшу кількість компонентів, що взаємодіють та спілкуються між собою. Це в свою чергу, може призвести затримок мережі та збільшення ризику недоступності різних сервісів.

Кожен окремий компонент backend частини представляє собою Web API, яка дозволяє безпечно та без втрат передавати дані від одного програмного застосунку до іншого. Його можна створити з допомогою багатьох технологій, але найкращими серед них є – Web API створені мовами програмування Python, а саме за допомогою бібліотек Flask, Djano, FastApi, та C#, а саме ASP.NET Framework, ASP.NET Core.[5][6] В обох випадках є досить хороша документація та різні сучасні методи та готові алгоритми для роботи з Web API, тому обирати краще ту мову, яка буде зручнішою для написання. На мою думку, процес передачі різних даних та їх обробка є набагато простішою, якщо використовувати

строго типізовану мову програмування, адже в такому випадку можна чітко зрозуміти структуру об’єктів і програми в цілому. Тому була обрана саме C#. Проте все ще стоїть вибір створення Web API: за допомогою ASP.NET Framework чи за допомогою ASP.NET Core. На справді вибір тут очевидний, оскільки ASP.NET Core є більш новий і має багато різних доповнень, які не тільки полегшують процес розробки API, а ще й полегшують процес перенесення веб застосунку в хмарне середовище за рахунок того, що ASP.NET Core є крос-платформний і підходить не тільки на Windows.

Frontend частина може буде створена за допомогою однією з двох основних засобів: Angular та React. Обидва засоби дозволяють дуже зручно використовувати різні розвинуті способи оформлення веб сторінки і дозволяють дуже легко взаємодіяти з даними, які потрібно взяти з Web API. Але було вирішено обрати саме Angular, оскільки він використовує TypeScript (покращену версію JavaScript з багатьма доповненнями) для написання бізнес логіки тоді як React використовує JavaScript з невеликими доповненнями. Окрім цього Angular має дуже багато уже готових та красивих шаблонів різних компонентів сторінки, що значно спрощує процес написання інтерфейсу для користувача.

Для того, щоб отримувати та змінювати дані в базі даних потрібно підключити бібліотеку, яка буде допомагати нам впливати на структуру та доповнювати базу даних. Таких бібліотек є декілька, але серед них найбільш зручною, надійною та найпростішою в освоєнні є Entity Framework. Тому будемо використовувати саме його

База даних у нашому контексті повинна бути реляційною, оскільки вона має в собі багато різних таблиць, які так чи інакше взаємодіють між собою. Сам вибір бази даних не є критично важливим, оскільки всі вони для нас будуть представляти лише джерело даних, яке потрібно підключити до Entity Framework. Тому головними критеріями вибору

бази даних буде його простота підключення до Entity Framework, простота перегляду даних за допомогою спеціально розроблених для цього програм та простота перенесення бази даних в хмарне середовище. По першому пункту безсумнівними лідерами є MS SQL Server, MySQL Server та PostgreSQL Server. Проте по останньому пункті можна з впевненістю сказати, що найзручнішою програмою для перенесення в хмарне середовище є саме MS SQL Server, який добре працює з Azure. Його використання поєднує в собі красивий і зрозумілий інтерфейс, а також можливість легко змінювати базу даних по нашій потребі. Окрім цього MS SQL також є однією з найпопулярніших баз даних, які переносяться в хмарне середовище, тому документації по цьому більш ніж достатньо.

Для вибору хмарної платформи на якій будуть зберігатися та функціонувати всі компоненти програми потрібно враховувати три основні пункти. Перший пункт – це складність налаштування та об’єднання всіх частин програми в одному хмарному середовищі. Другий пункт – це продуктивність та швидкодія нашої програми. Третій пункт – це ціна. Опираючись на перший пункт, основними хмарними платформами можна вважати Azure, Heroku та AWS. По другому пункту Heroku не проходить, оскільки саме в ньому коефіцієнт продуктивності та швидкодії є найменшим. Залишається Azure та AWS. Давайте розглянемо ключові відмінності між ними.

Перш за все потрібно розглянути можливості масштабованості кожного з варіанті. Для її вирішення AWS використовує гнучкий хмарний сервіс обчислення (EC2), в якому доступний обсяг ресурсів може збільшуватись або зменшуватись за потребою завдяки можливостям гнучкого хмарного обчислення. Локальні кластери надають лише деяку частину загального пулу ресурсів, які доступні всім процесам одночасно. З іншого боку, в Azure для забезпечення

масштабованості та балансування навантаження використовуються віртуальні масштабні набори. Основна відмінність полягає в тому, що AWS може бути налаштований для різних цілей, тоді як Azure працюють у взаємодії з іншими інструментами для розгортання хмарних рішень. Також варто сказати, що успіх розгортання хмарних рішень залежить від наявності достатнього обсягу зберігання. Але з цього питання Azure і AWS майже рівнозначно сильні.

Тепер поговоримо про документації та простоту використання сервісів. AWS пропонує більшу простоту використання і є гарним вибором для користувачів хмарних платформ, які вперше з ними знайомляться. Першим засобом є панель керування, яка має багатий функціонал та є зручною для користувача. AWS також надає широку документацію для своїх хмарних сервісів. Для розміщення простого екземпляра EC2 користувачі можуть вводити свої запити у пошукове поле AWS і переходити до Документації для перегляду відео чи написаного уроку. Однак, додавання користувачів та прав доступу є складнішими в AWS. Azure зберігає всі облікові записи користувачів та інформацію в одному місці, хоча його документація та система рекомендацій менш інтуїтивні та зручні для пошуку.

Однією з переваг хмарних провайдерів є простота розгортання додатків. Користувачі в ролях розробників можуть бажати розмістити свої додатки на кількох віртуальних серверах за допомогою можливостей платформи як сервісу. Для підтримки цього, Azure пропонує різноманітні варіанти розгортання додатків, включаючи хмарні служби, контейнерні служби, функції, пакети та служби додатків. AWS також має подібні можливості з Elastic Beanstalk, Batch, Lambda, контейнери і багато інших. Однак, він має кілька недоліків у сфері хостингу додатків, наприклад, Azure забезпечує захист інтелектуальної власності та надає кращу обробку потоків даних з заднього плану.

З точки зору ціни, AWS та Azure обидві пропонують розумні тарифи та модель ціноутворення плати за використання. Крім того, обидва надають безкоштовні вступні пакети, щоб користувачі отримали уявлення про те, як їх системи можуть бути інтегровані з локальним програмним забезпеченням. AWS розраховується на годинній основі. У свою чергу, Azure розраховується на хвилинній основі, що означає, що користувачі можуть отримати більш точну цінову компоненту порівняно з AWS. Крім того, Azure дозволяє укладати короткострокові зобов'язання з передоплатою або щомісячними платежами. Короткострокові плани підписки на Azure надають більшу гнучкість. Проте також треба враховувати, що при порівнянні обох платформ, Azure, як правило, виявляється більш дорогою опцією при збільшенні ресурсів. Це можна зрозуміти на прикладі вартості сервісу, яка дуже сильно зростає у ціні зі збільшенням обсягу. Проте дешевше буде обрати саме Azure при виборі середньої кількості ресурсів.

Підсумувавши можна сказати, що Azure та AWS пропонують схожі можливості своїм користувачам, і обидва хмарні продукти надзвичайно подібні. Але на наш вибір в сторону Azure вплинула наявність більшої екосистеми Microsoft, включаючи різноманітні продуктивні інструменти, бізнес-додатки і вбудовані функції, які допомагають простіше налагодити зв’язок між різними хмарними середовищами. Окрім цього, в нашому випадку, дешевше буде обрати саме Azure адже кількість ресурсів, які ми плануємо використовувати в контексті студентської підписки, менша середнього.

Active container instances

$0.064 / vCPU-hour*
$0.007 / GB-hour*

When your application is processing requests, you switch from provisioned container instances to active container instances that consume both memory and compute resources. You pay for the compute and any additional memory counsumed in excess of the memory allocated by your provisioned container instances. App Runner automatically scales the number of active container instances up and down to meet the processing requirements of your application. You can set a maximum limit on the number of active container instances your application uses so that costs do not exceed your budget. When your active container instances are idle, App Runner scales back to your provisioned container instances (the default is 1 provisioned container instance).

All container instance processing is billed per second, rounded up to the next nearest second. There is a one minute minimum charge for vCPU resources every time a provisioned container instance starts processing requests.

* The pricing of $0.007 / GB-hour and $0.064 / vCPU-hour is applicable for the following AWS Regions: US East (N. Virginia), US East (Ohio), US West (Oregon), and Europe (Ireland). For the Asia Pacific (Tokyo) Region, pricing is $0.009 / GB-hour and $0.081 / vCPU-hour.

Supported configurations

CPU	Memory values
0.25 vCPU	0.5 GB
0.25 vCPU	1 GB
0.5 vCPU	1 GB
1 vCPU	2GB
1 vCPU	3GB
1 vCPU	4GB
2 vCPU	4GB
2 vCPU	6 GB
4 vCPU	8 GB
4 vCPU	10 GB
4 vCPU	12 GB

Рисунок 1.2 Список конфігурацій у AWS та його ціна

Popular plans

Free F1 0.00 USD/Month (Estimated)
60 CPU Minutes/day included

Basic B1 12.41 USD/Month (Estimated)
ACU: 100, Memory: 1.75 GB, vCPU: 1

Premium V3 P1V3 113.15 USD/Month (Estimated)
ACU: 195, Memory: 8 GB, vCPU: 2

Premium V3 P2V3 226.30 USD/Month (Estimated)
ACU: 195, Memory: 16 GB, vCPU: 4

Premium V3 P3V3 452.60 USD/Month (Estimated)
ACU: 195, Memory: 32 GB, vCPU: 8

Isolated V2 I1V2 281.78 USD/Month (Estimated)
ACU: 195, Memory: 8 GB, vCPU: 2

Isolated V2 I2V2 563.56 USD/Month (Estimated)
ACU: 195, Memory: 16 GB, vCPU: 4

Isolated V2 I3V2 1,127.12 USD/Month (Estimated)
ACU: 195, Memory: 32 GB, vCPU: 8

Рисунок 1.3 Список конфігурацій у Azure та його ціна

1.3 Висновок

Як висновок – можемо побачити, що у кожного з вище перерахованих аналогів є багато чого, що можна запозичити і пізніше покращити в цьому веб застосунку. Проте також можна побачити, що додаток, де пацієнт бере безпосередню участь у деякій частині роботи медичного закладу є чимось новим і ще не освоєним, що робить цей проект, свого роду, унікальним.

Після розгляду існуючих технологій можна зробити висновок, що багато готових рішень мають схожий функціонал, але при тому мають різний рівень продуктивності. Це дає споживачам можливість обирати оптимальний варіант дивлячись на те, що працює краще на їх погляд. Саме тому, ретельний аналіз характеристик та порівняння аналогів допоможе знайти оптимальне рішення та покращити ефективність нашого проекту.

РОЗДІЛ 2

2.1 Архітектура проекту

Для реалізації додатку для віддаленої комунікації медичного закладу з пацієнтами було обрано мікросервісну архітектуру як основний підхід до розробки нашого застосунку. Її було обрано з урахуванням ряду факторів та вимог, які є критичними для медичного додатку, а саме:

- Надійність
- Підтримуваність
- Масштабованість
- Здатність до тестування

Розглянемо загальну архітектуру проекту

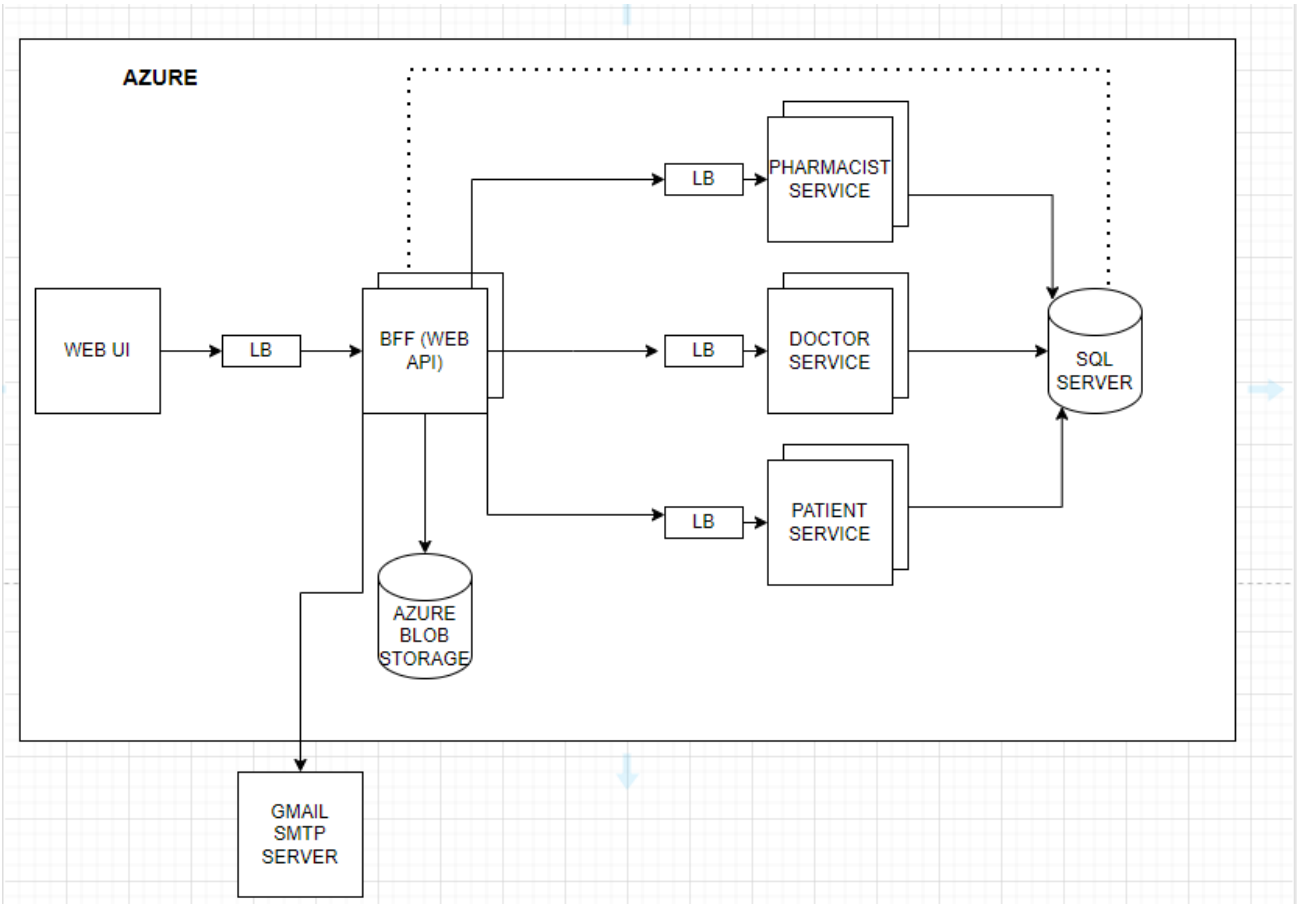

Рисунок 2 Архітектура додатка

На рисунку 2 зображено загальну архітектуру проекту: компоненти системи, їх зв'язки між собою та розташування у інфраструктурі.

Кожен модуль проекту має область роботи за яку він є відповідальний, розглянемо область відповідальності кожного з сервісів.

WEB UI - це клієнтський модуль, відповідальний за представлення даних та інтерфейсу користувача, візуальне відображення інформації, взаємодію з користувачем та забезпечення зручної та ефективної взаємодії з користувачем. Окрім того WEB UI модуль включає клієнтську валідацію інформації, механізм навігації по системі.

BFF (Backend for frontend) модуль - це компонент створений на основі BFF паттерна проектування, основна відповідальність якого полягає в таких процесах:

- Адаптація API: Backend for frontend відповідає за адаптацію API, наданих трьома мікросервісами (Pharmacist service, Doctor service, Patient service).
- Кешування та оптимізація: Даний модуль використовує механізм кешування запитів, що в свою чергу покращує швидкість відповідей на запити та зменшує час очікування користувача на відповідь системи.
- Авторизація та автентифікація: Даний модуль відповідальний за перевірку прав доступу та встановлення автентичності користувачів.
- Валідація та форматування даних: Одна з рівней відповідальності даного компонента є затвердження, перевірка та зміна даних наданих користувачем.

- Моніторинг та логування: Модуль забезпечує механізм постійного спостереження та логування взаємодії із клієнтами, а саме: збирання різних одиниць вимірювання про взаємодію з API, обробку запитів та помилки.
- Надсилання електронних листів: Модуль є відповідальний за делегування роботи пов'язаної зі електронною розсилкою до Gmail SMTP сервера.

Pharmacist service - це мікросервіс відповідальний за роботу з інформацією про фармацевтів та ліки, а саме:

- CRUD операції з класами медичних препаратів.
- CRUD операції з користувачами-фармацевтами медичного закладу.

Doctor service - мікросервіс відповідальний за роботу з лікарям, а саме:

- CRUD операції з користувачами-лікарями медичного закладу.

Patient service - мікросервіс відповідальний за роботу з пацієнтами, а саме:

- CRUD операції з користувачами-пацієнтами медичного закладу.

Azure BLOB (binary large objects) storage - сервіс наданий хмарним середовищем, для зберігання бінарної інформації, як-от: фотографії, PDF-файли, тощо.

Microsoft SQL Server - основна реляційна база даних для зберігання інформації системи та її об’єктів.

2.2 UML діаграми проекту

Для кращого представлення усіх можливостей веб застосунку та для кращого розуміння його структури були створені різні UML діаграми.[7]

Діаграма варіантів використання описує всіх суб’єктів програми та операції, які вони можуть виконати

Діаграма розгортання описує кінцеву основну структуру веб застосунку з використанням усіх її компонентів

Діаграма класів дозволяє краще зрозуміти з який моделей складається наша програма. У ній описані усі моделі та властивості цих моделів за допомогою яких відбувались передачі даних між frontend та backend частино, а також за допомогою яких були створені відповідні таблиці в базі даних за допомогою Entity Framework.

Діаграма послідовностей описує детальний шлях проходження запиту клієнта по усій програмі. Саме за допомогою цієї діаграми можна зрозумів, який компонент відповідає за ту чи іншу роботу.

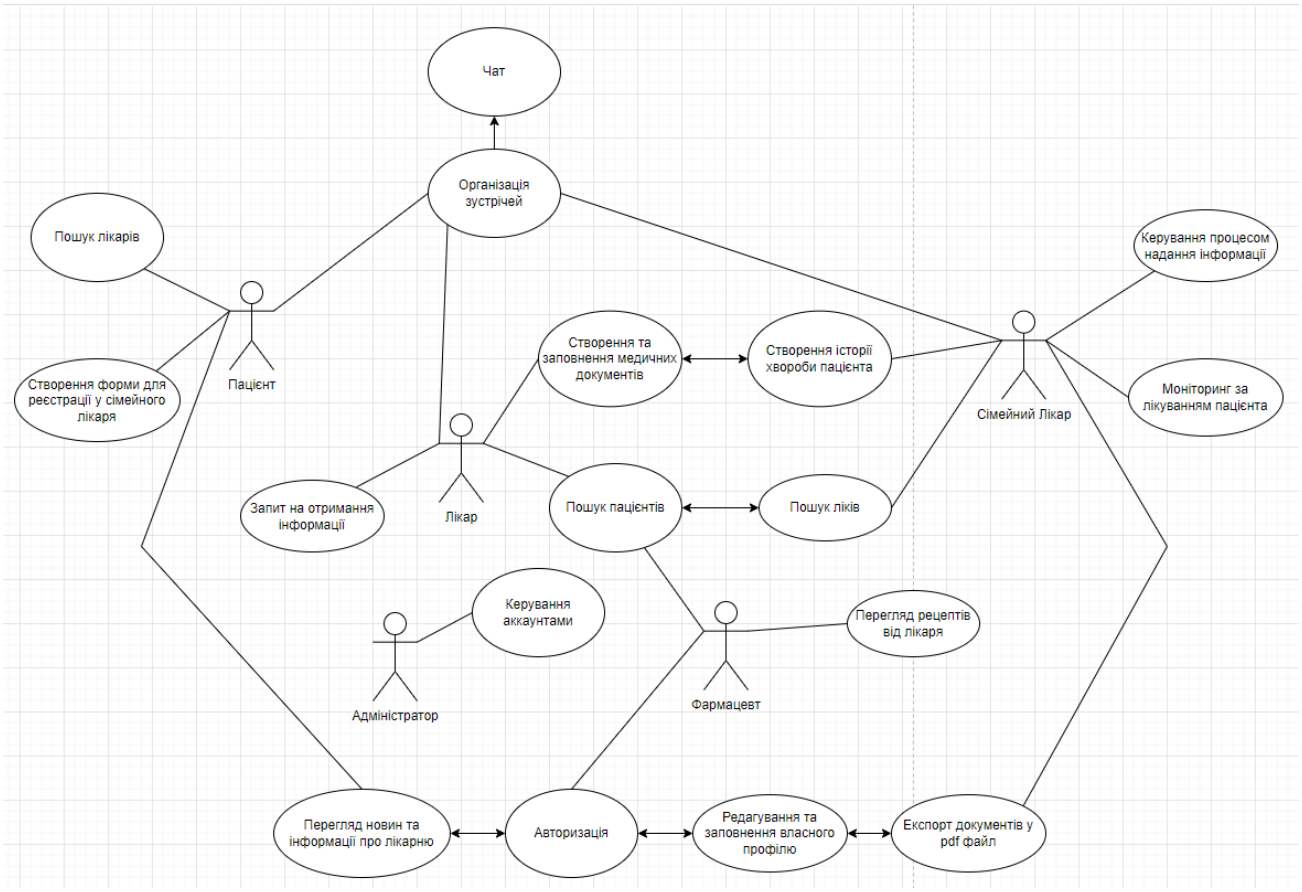

Рисунок 2.1 Діаграма варіантів використання

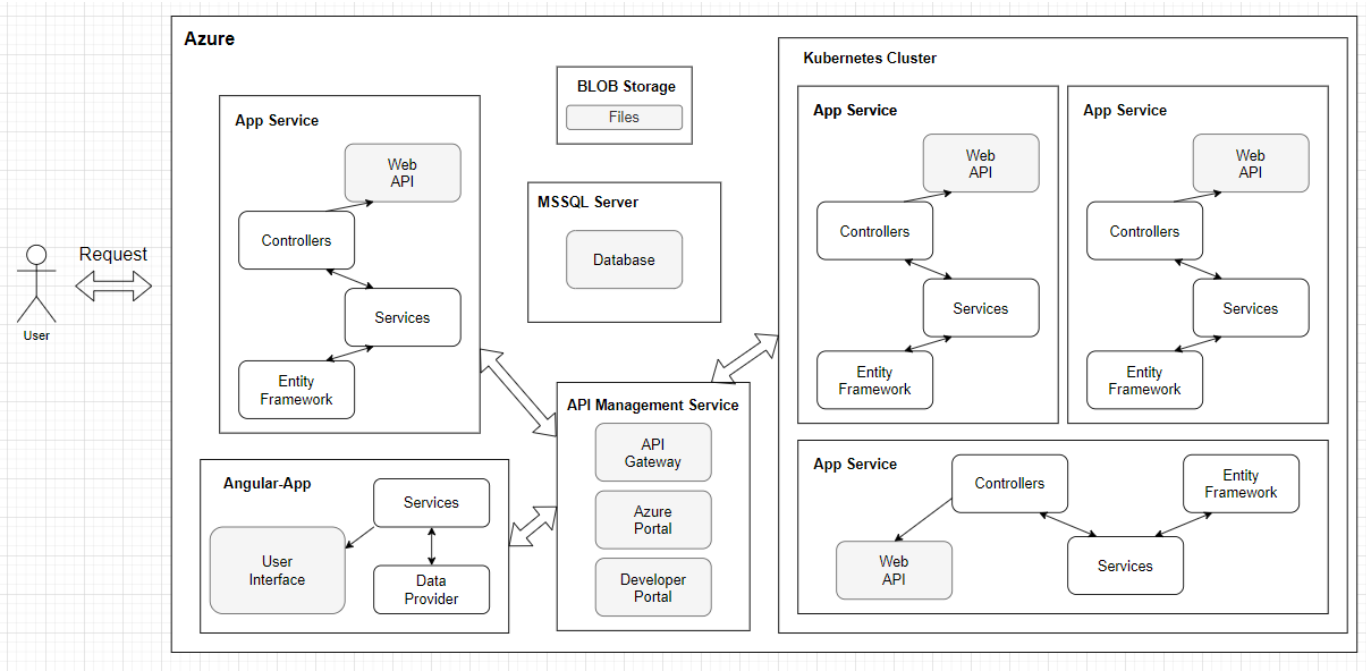

Рисунок 2.2 Діаграма розгортання

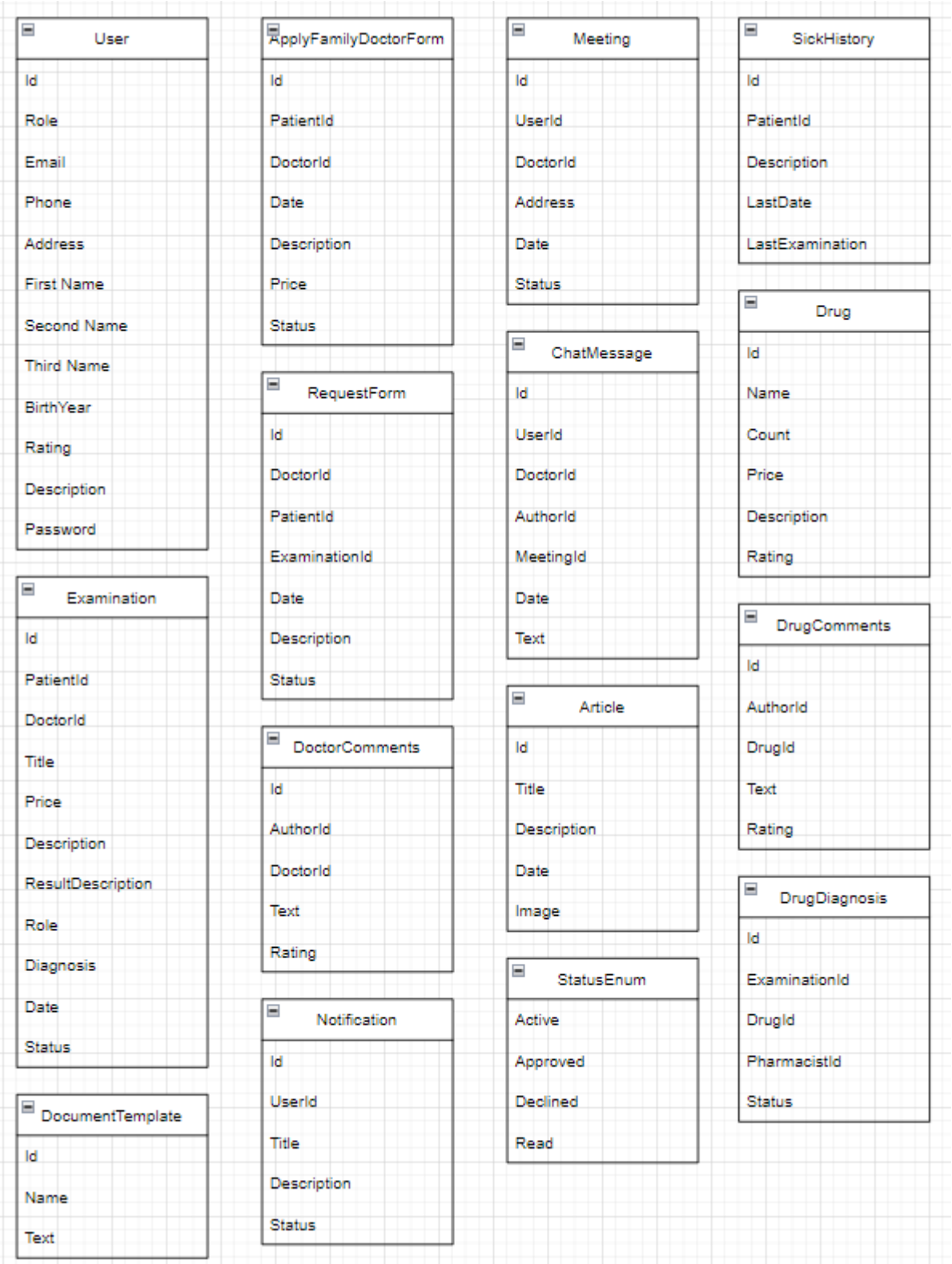

Рисунок 2.3 Діаграма класів

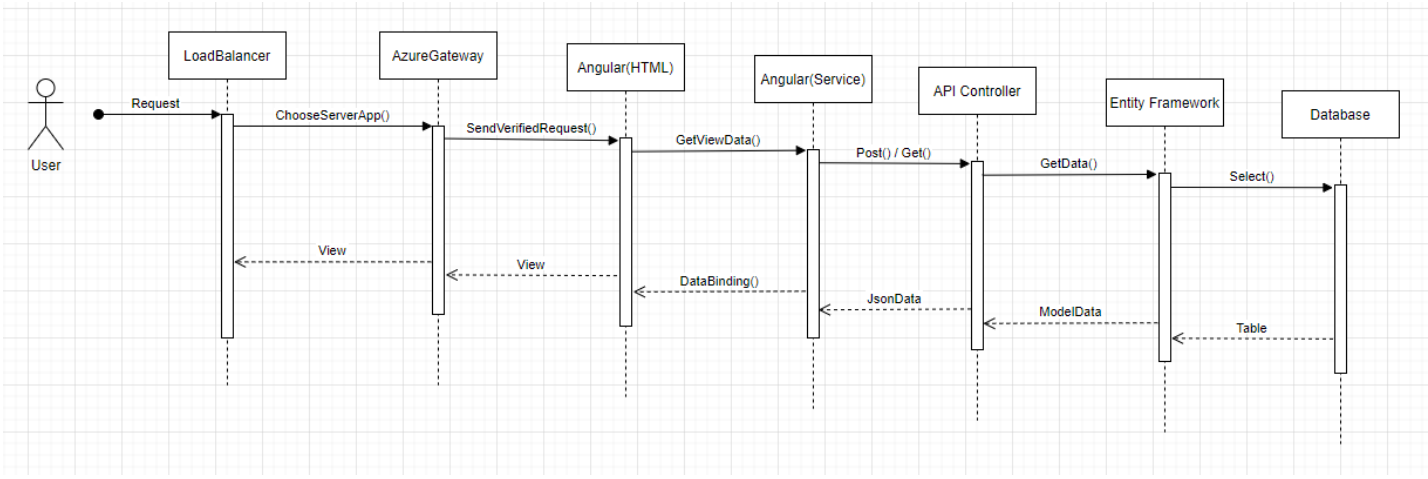

Рисунок 2.4 Діаграма послідовностей

2.3 Висновок

В ході створення архітектури системи, було проаналізовано переваги та недоліки мікросервісної архітектури. Окрім цього, були розглянуті потреби у вимогах до додатку та подальші рішення стосовно їх реалізації та покращення.

Також можна точно сказати, що UML-діаграми є потужним інструментом для моделювання та візуалізації різних аспектів проекту. Їх використання дозволяє створювати структуровану модель проекту, яка допомагає зрозуміти усі взаємозв’язки між компонентами системи, логіку взаємодії та потреби користувача. Це сприяє кращому аналізу переваг та недоліків різних частин проекту для подальшої його розробки та реалізації. Також варто сказати, що з їх допомогою легше виконувати аналіз вимог та документування системи на різних етапах розробки.

РОЗДІЛ 3

3.1 Математичне забезпечення

Для реалізації пошуку пацієнтів, лікарів та ліків використовувався бінарний пошук, а для їх сортування – алгоритм швидкого сортування. Розберемо кожен з цих алгоритмів:

Бінарний пошук – це один з алгоритмів знаходження значення у списку. Припустимо, що потрібно знайти Аспірин. Спочатку програма поділить весь список на дві рівні частини і почне перебирати першу частину. Якщо в першій половині елемент був знайдений то він виводиться, якщо ж ні то наступна частина ділиться ще на дві частини і пошук продовжується у першій частині нового розподілу. Цей алгоритм рекурсивно продовжується до того часу поки елемент не буде найдений або поки не будуть переглянуті усі елементи. Завдяки цьому методу трудомісткість алгоритму складає $1 + \log_2^n$, де n – кількість елементів у списку.

Швидке сортування – це алгоритм сортування, який виконується у середньому $n*\log^n$. Цей алгоритм був обраний, оскільки він використовує дуже прості цикли і операції і завдяки цьому він працює швидше за інші алгоритми, що мають таку ж асимптотичну складність. Ідея алгоритму полягає в переставлянні елементів масиву таким чином, щоб його можна було розділити на дві частини і кожний елемент з першої частини був не більший за будь-який елемент з другої. Впорядкування кожної з частин відбувається рекурсивно.[8]

Окрім вище наведених алгоритмів пошуку та сортування, для процесу авторизації та автентифікації користувачів було використано JSON Web Token, який є закодований за допомогою алгоритму шифрування HMAC-SHA512.

Алгоритм HMAC-SHA512 є одним з алгоритмів хешування з кодом автентифікації повідомлень. Він використовує функцію хешування SHA-512 у поєднанні з секретним ключем для створення коду автентифікації повідомлення.

Основна мета HMAC-SHA512 полягає в тому, щоб забезпечити цілісність, автентичність та некерованість повідомлення. В даному алгоритмі це досягається за допомогою комбінації секретного ключа та повідомлення, що потрібно закодувати. Код автентифікації, який отримують в результаті шифрування може бути перевірений на цілісність та автентичність.

Наведемо основні кроки роботи алгоритму HMAC-SHA512:

- Ініціалізація: Це процес, в ході якого, визначається секретний ключ, який використовується для створення коду автентифікації (це може бути довільна, випадкова стрічка символів, у випадку нашого додатку, вона випадково генерується та має довжину 16 символів).
- Підготовка ключа: Якщо секретний ключ довший, ніж розмір блоку хеш-функції SHA-512 (128 байтів), він хешується до розміру блоку за допомогою SHA-512. Якщо секретний ключ коротший, ніж розмір блоку, він доповнюється нульовими байтами до розміру блоку.
- Конструювання ініціалізаційного вектора: Ініціалізаційний вектор складається з повторення 0x36 байтів в розмірі блоку, а ініціалізаційний вектор для обчислення коду автентифікації складається з повторення 0x5C байтів в розмірі блоку.

- Обчислення коду автентифікації: На даному етапі роботи алгоритму, повідомлення обробляється двічі. По-перше, його з’єднується з ініціалізаційним вектором, і результат хешується за допомогою SHA-512. Потім, отриманий хеш з’єднується з ініціалізаційним вектором, і результат знову хешується за допомогою SHA-512. Остаточний результат є кодом автентифікації HMAC-SHA512.
- Перевірка коду автентифікації: При перевірці коду автентифікації HMAC-SHA512 для повідомлення, виконується той самий процес обчислення коду автентифікації з вхідним повідомленням та секретним ключем. Отриманий код порівнюється з переданим кодом автентифікації. Якщо вони збігаються, то в такому випадку повідомлення вважається цілим і автентичним.

Алгоритм HMAC-SHA512 забезпечує стійкість до атак на зміну повідомлення та використовується в багатьох протоколах і системах для забезпечення безпеки та автентифікації даних, саме тому ми обрали його для кодування JWT. Також варто зауважити, що для реалізації захисту токену на основі даного алгоритму ми використали інфраструктуру надану середовищем ASP.NET CORE, оскільки розробка алгоритму шифрування з нуля є дуже трудомістким та складним процесом, в ході якого можна допустити критичні помилки, що безпосередньо можуть вплинути на безпеку роботи з додатком, що є неприпустимим при роботі з медичною платформою, оскільки вона оперує приватною інформацією про користувачів.

3.2 Висновок

Математичне забезпечення є важливою складовою проекту, особливо у випадку, коли проект є досить великим і вимагає швидкої обробки інформації. Його ефективне використання може допомогти вдосконалити систему, знизити різні ризики та забезпечити оптимальне рішення для вирішення проблеми. А це напряму впливає на ефективність та успішність самого проекту.

РОЗДІЛ 4

4.1 Розробка програмного забезпечення

Для кращого розуміння структури медичного додатку розглянемо внутрішню будову кожного з компонентів проекту та їх програмні інтерфейси. Розпочнемо з підключення до кожного з наших Web API декілька важливих опцій. Перша опція це додавання Health Check перевірок для отримання стану того чи іншого сервісу в даний момент часу. Друга опція це підключення swagger, який допоможе пришвидшити тестування та розробку своєрідної API документації. Також вона допоможе краще бачити структуру того чи іншого сервісу не дивлячись в сам код. Тепер розглянемо кожний компонент окремо.

BFF модуль:

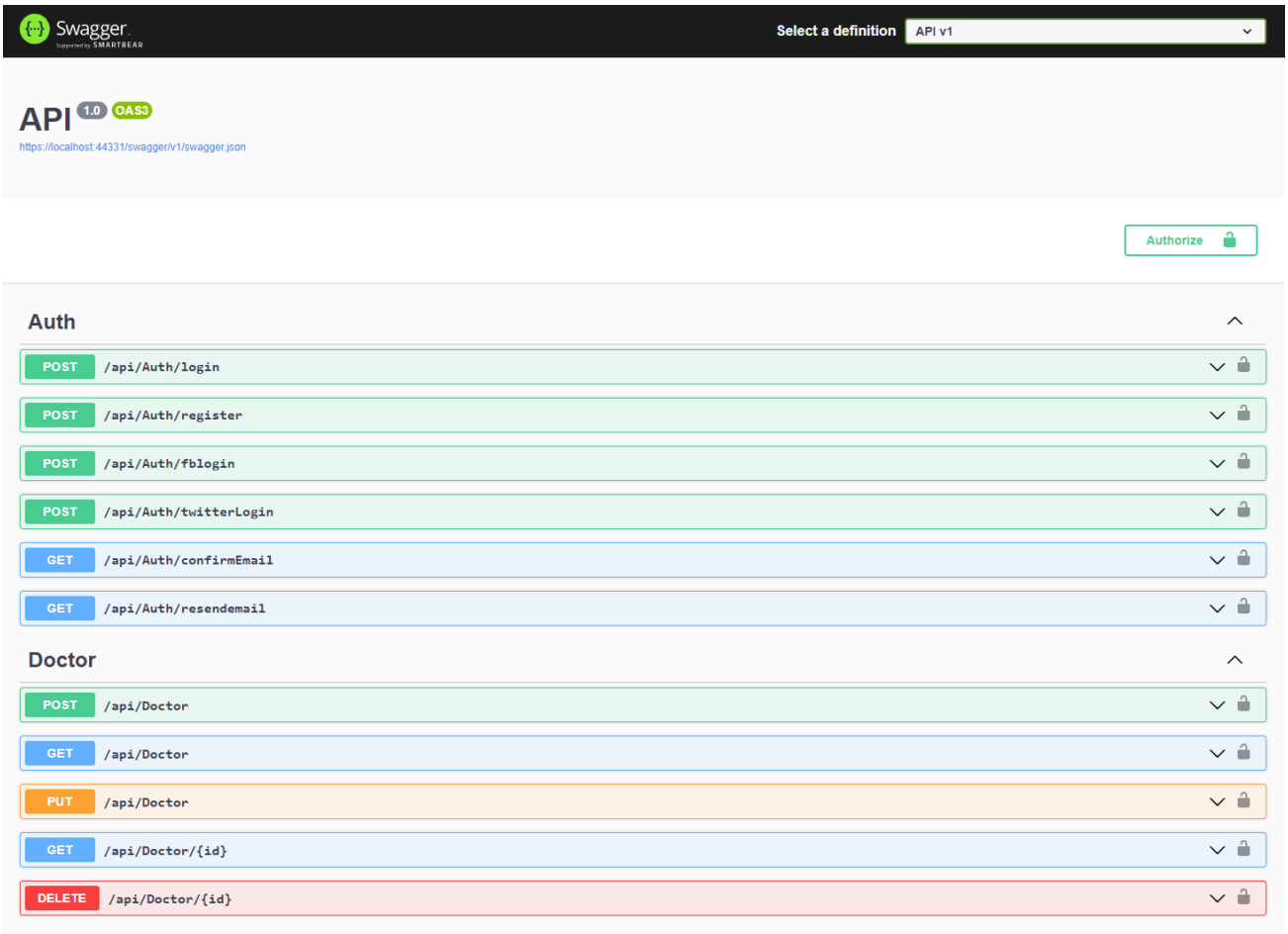

Рисунок 3 Swagger сторінка BFF модулю частина 1

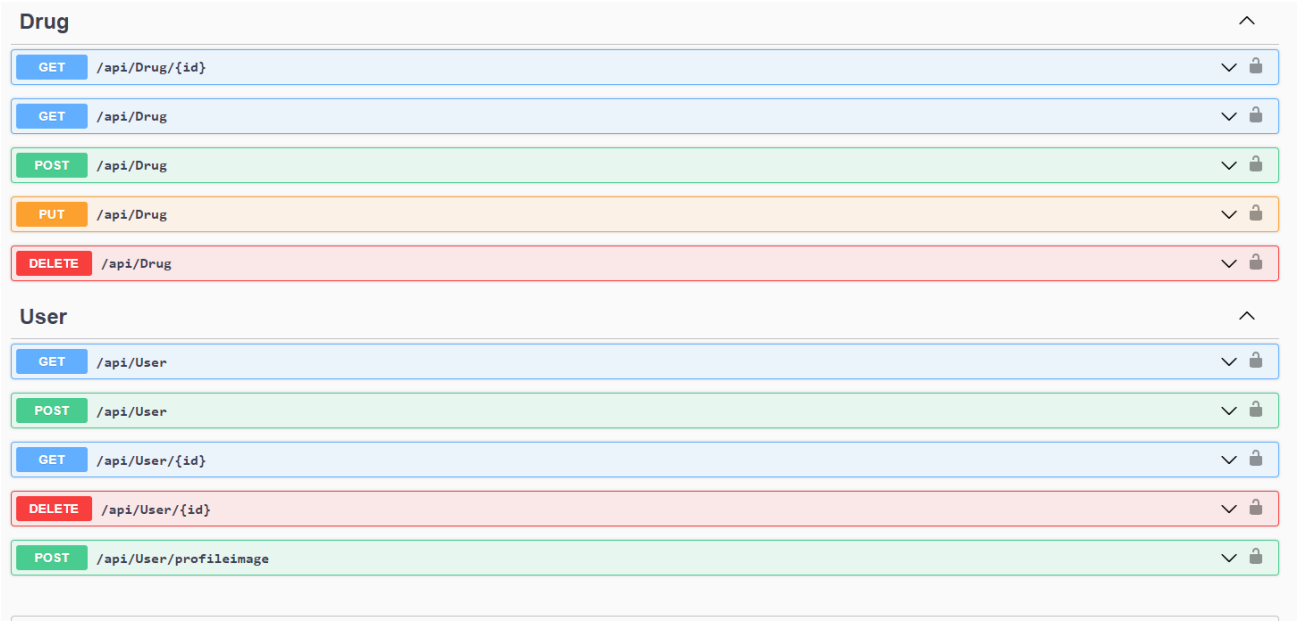

Рисунок 3.1 Swagger сторінка BFF модулю частина 2

На рисунках 3 та 3.1 зображено API компоненту BFF задокументовані за допомогою бібліотеки Swagger. Розглянемо кожен з ендпоінтів в межах своїх контролерів.

AuthController:

- POST …/login - надає можливісь для логіну користувачів додатку.
- POST …/register - надає можливість для реєстрації нових користувачів додатку.
- POST …/fblogin - надає можливість для логіну та реєстрації користувачів за допомогою соціальної мережі facebook. Якщо користувач є не зареєстрований у додатку, то його буде зарєстровано, якщо ж він уже є зареєстрований, то його залогує.
- POST …/twitterlogin - надає можливість логіну та реєстрації користувачів, аналогічно до попереднього пункту, за допомогою соціальної мережі Twitter.
- GET …/confirmemail - інтерфейс для підтвердження емейл адреси користувачів додатку.

- GET …/resendemail - надає можливість повторно надіслати лист-підтвердження реєстрації на пошту користувача.

DoctorController:

- POST …/doctor - інтерфейс для створення лікарів.
- GET …/doctor - інтерфейс для отримання списку з інформацією про всіх лікарів.
- PUT …/doctor - інтерфейс для оновлення інформації про лікаря.
- GET …/doctor/{id} - інтерфейс для отримання інформації про конкретного лікаря.
- DELETE …/doctor/{id} - інтерфейс для видалення лікаря із системи.

Варто зазначити, що основна мета вищенаведених API, окрім валідації, агрегації та форматування інформації пов’язаної із лікарями полягає в тому, щоб делегувати запити до Doctor Service компоненту.

DrugController:

- GET …/drug{id} - інтерфейс для отримання конкртених ліків за допомогою їх id.
- GET …/drug - інтерфейс для отримання колекції із інформацією про усі ліки.
- POST …/drug - інтерфейс для створення (додавання) нових ліків у систему.
- PUT …/drug - інтерфейс для оновлення інформації про існуючі ліки.
- DELETE …/drug/{id} - інтерфейс для видалення ліків за їх id.

Програмні інтерфейси даного контроллера, також делегують роботу до відповідного мікросервісу.

UserController:

- GET …/user - інтерфейс для отримання списку усіх користувачів аплікації, незалежно від їх ролей.
- POST …/user - інтерфейс для створення користувачів.
- PUT …/user - інтерфейс для оновлення інформації про користувачів.
- GET …/user/{id} - інтерфейс для отримання користувачів аплікації, за допомогою їх id.
- POST …/user/profileimage - інтерфейс для додавання та оновлення зображення профілю користувачів аплікації.

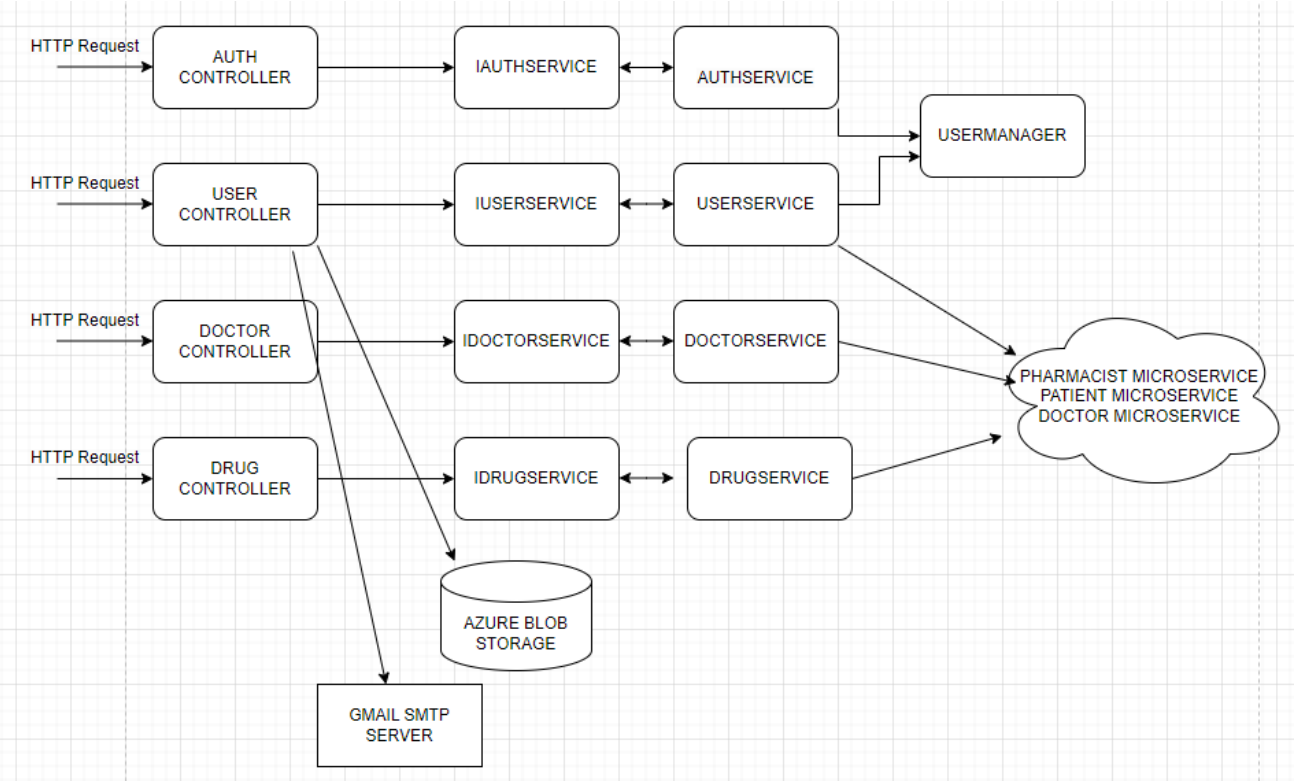

Рисунок 3.2 Внутрішня будова BFF модуля

На рисунку 3.2 зображено внутрішню будову BFF, з схеми видно, що для розробки даного компоненту було використано сервісний підхід розробки. В даному випадку відповідальність сервісів полягає в перевірці, компонування, форматування даних та у спілкуванні з відповідними мікросервісами.

Doctor Service мікросервіс:

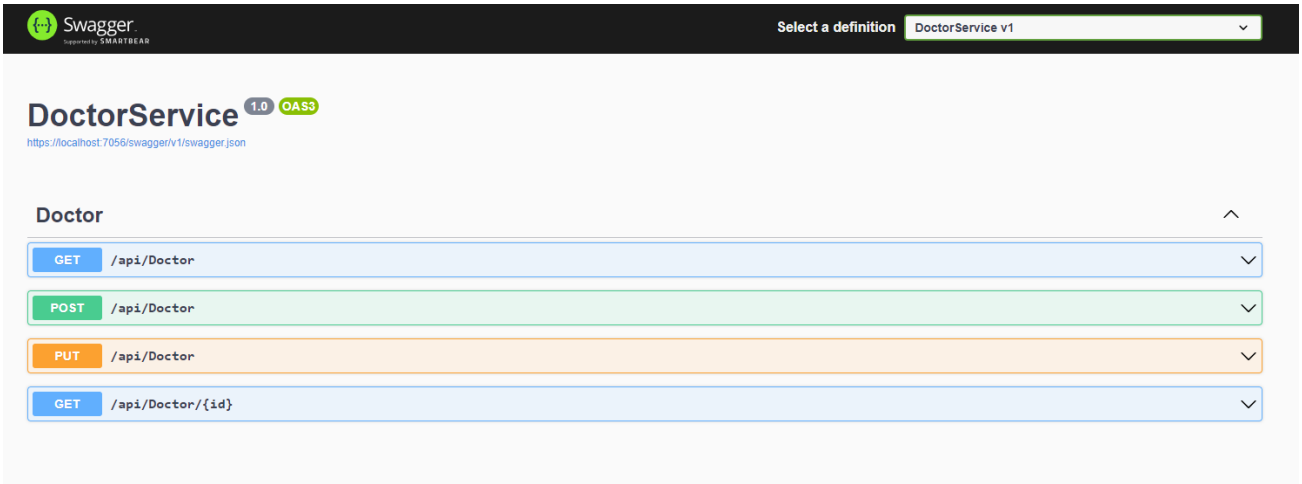

Рисунок 3.3 Swagger сторінка Doctor service

На рисунку 3.3 зображено API мікросервісу відповідального за роботу з лікарями, задокументовані за допомогою бібліотеки Swagger.

Розглянемо кожен з ендпоінтів.

- POST …/doctor - API для створення лікарів.
- GET …/doctor - API для отримання всіх лікарів.
- PUT …/doctor - API для зміни або оновлення інформації про лікарів закладу.
- GET …/doctor/{id} - API для отримання інформації про лікаря за допомогою його id.
- DELETE …/doctor/{id} - API для видалення лікарів за їх id.

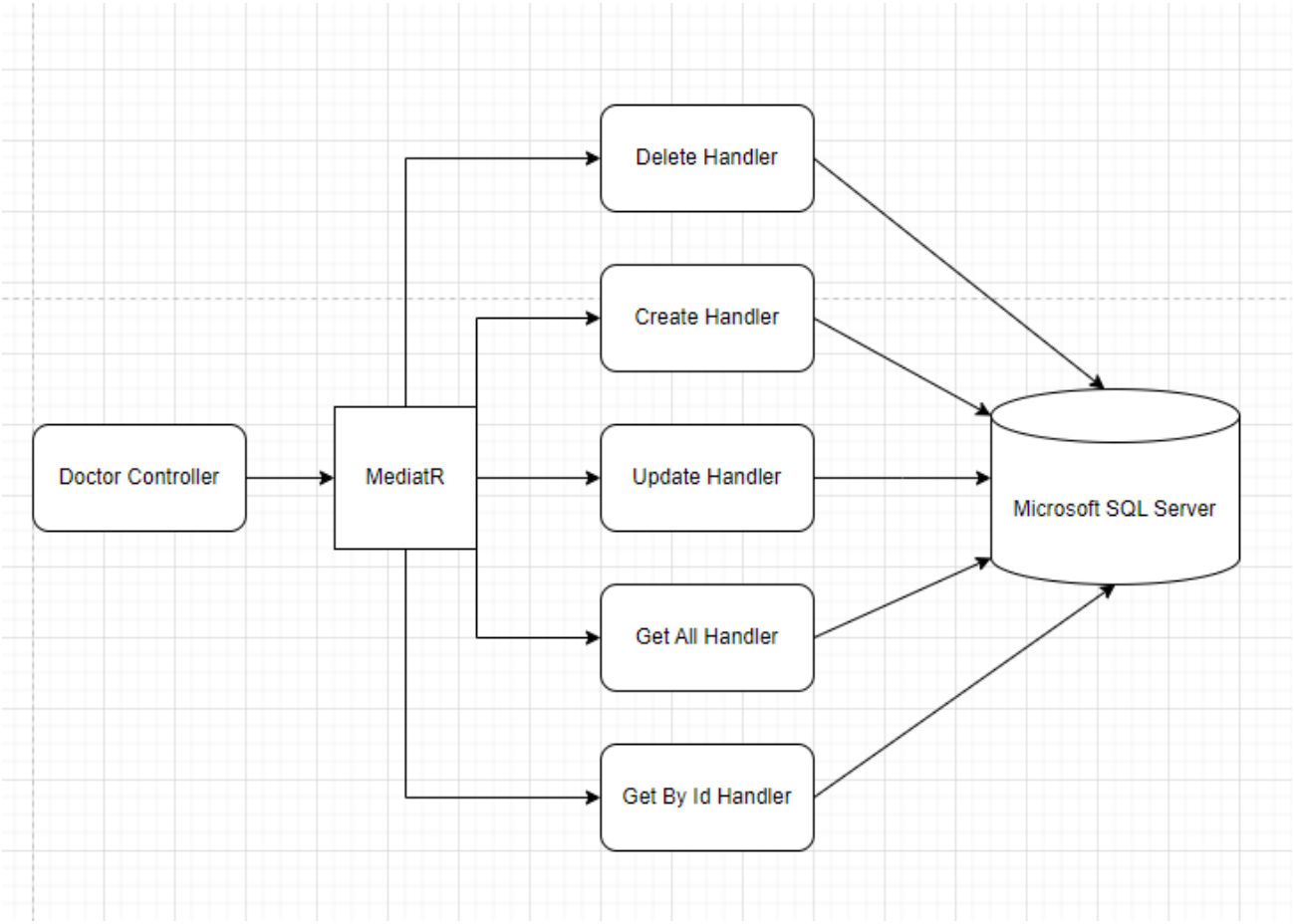

Рисунок 3.4 Структура Doctor service

На рисунку 3.4 зображено внутрішню структуру мікросервісу відповідального за роботою з лікарями, для реалізації даного модуля ми обрали архітектурний паттерн CQRS (Command-Query Responsibility Segregation), що в свою чергу, дозволило краще розділити бізнес логіку компоненту.

Patient Service мікросервіс:

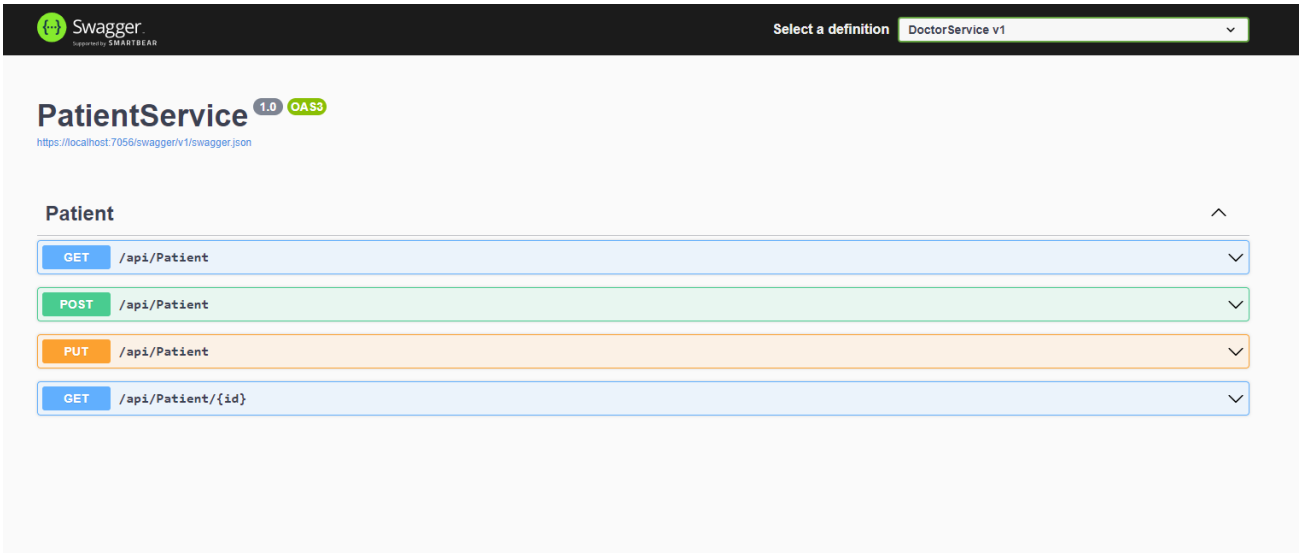

Рисунок 3.5 Swagger сторінка Patient service

На рисунку 3.5 зображено програмні інтерфейси мікросервісу відповідального за роботу з пацієнтами, задокументовані за допомогою бібліотеки Swagger. Розглянемо кожен з ендпоінтів.

1. GET…/patient - API для отримання списку пацієнтів медичного закладу.
2. GET…/patient/{id} - API для отримання пацієнта медичного закладу за допомогою його id.
3. POST …/patient - API для створення/додавання нових пацієнтів у систему.
4. PUT …/patient - API для зміни або оновлення інформації про пацієнтів закладу.
5. DELETE …/patient/{id} - API для видалення пацієнтів із системи за їх id.

Ознайомившись із програмними інтерфейсами даного мікросервісу, перейдемо до огляду його внутрішньої структури.

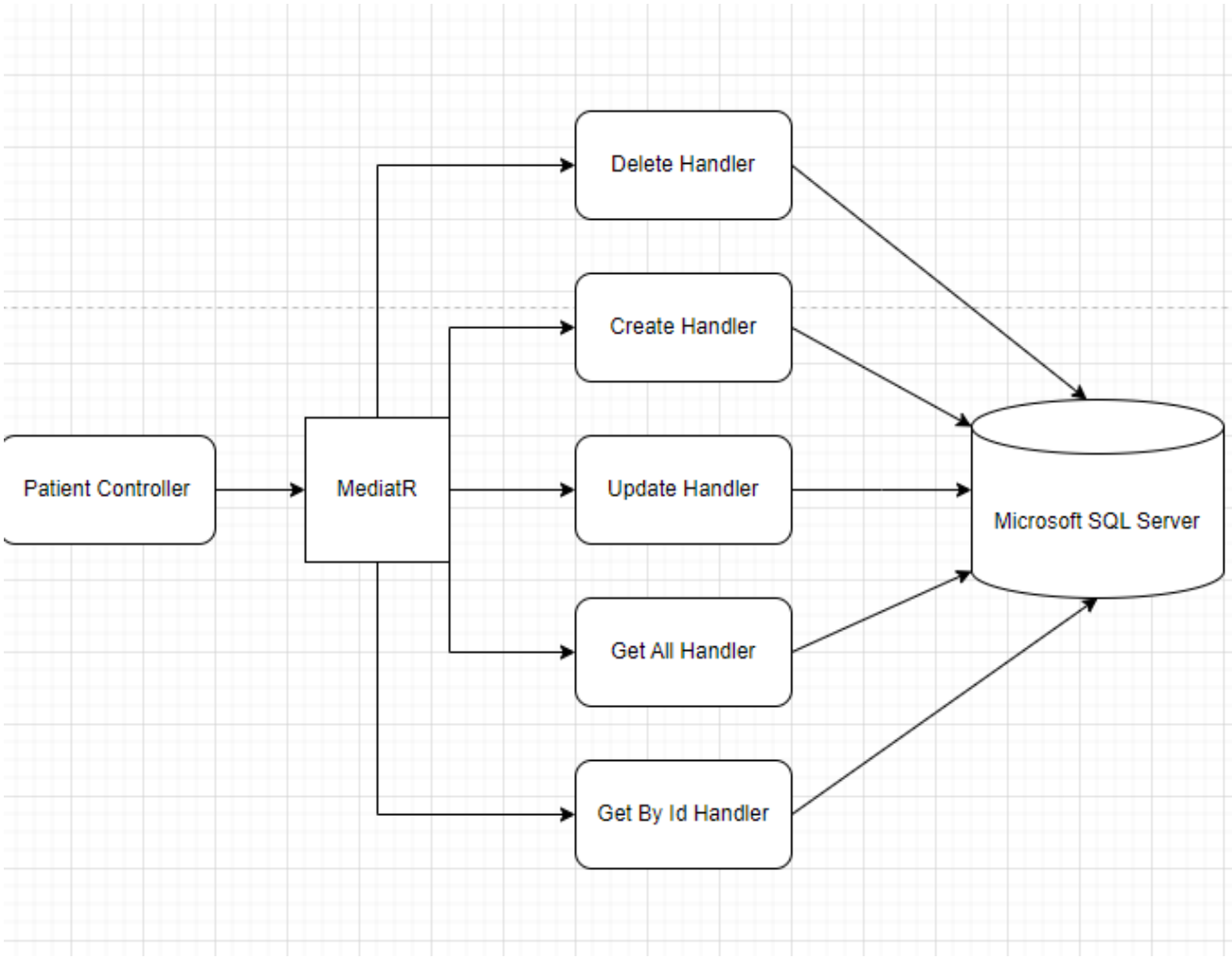

Рисунок 3.6 Структура Patient service

На рисунку 3.6 зображено внутрішню структуру мікросервісу відповідального за роботу з пацієнтами, для реалізації даного модуля також було обрано архітектурний паттерн Command-Query Responsibility Segregation.

Pharmacist Service мікросервіс:

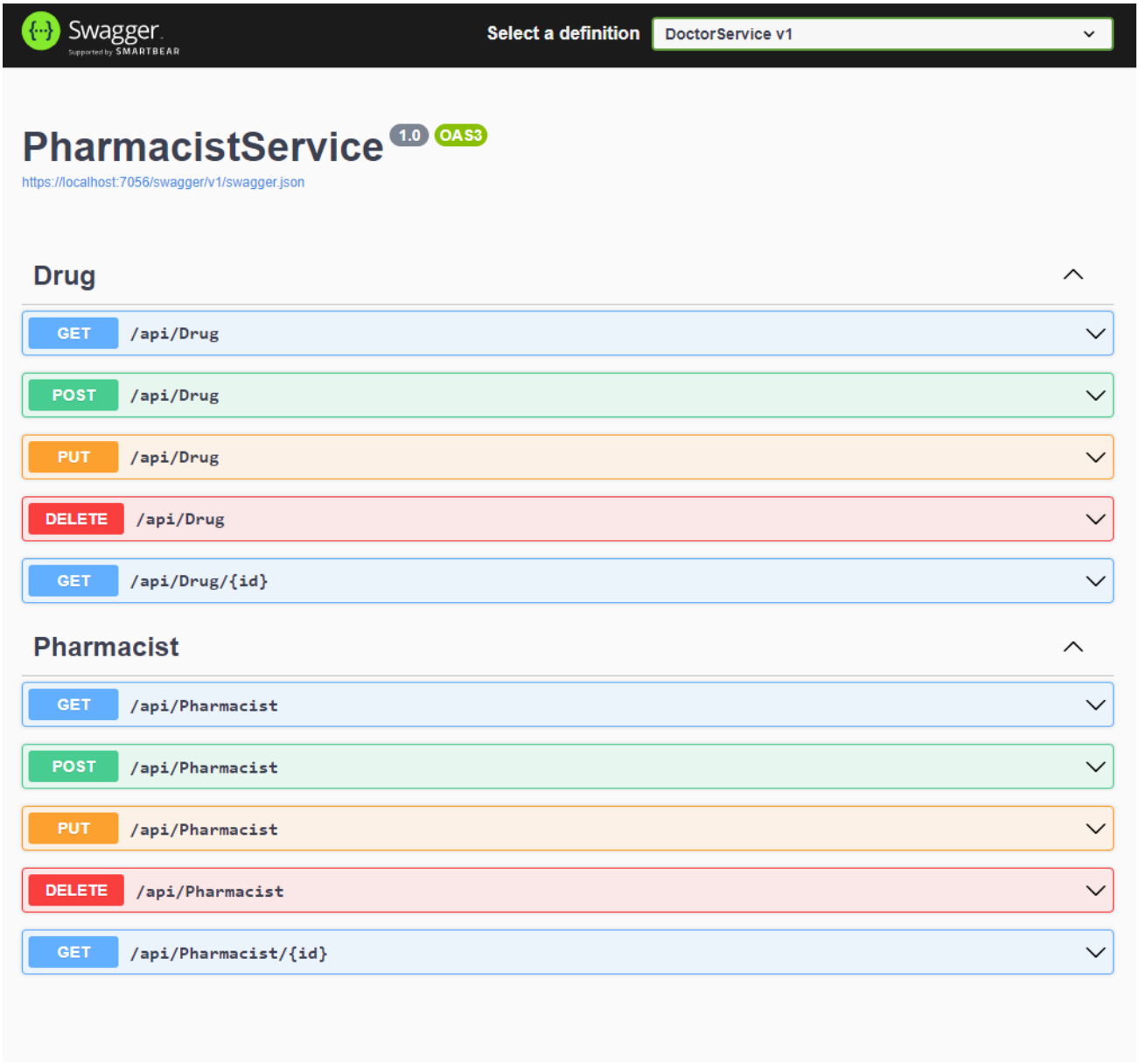

Рисунок 3.6 Swagger сторінка Pharmacist service

На рисунку 3.7 зображено програмні інтерфейси мікросервісу відовадального за роботу з фармацевтами та ліками.

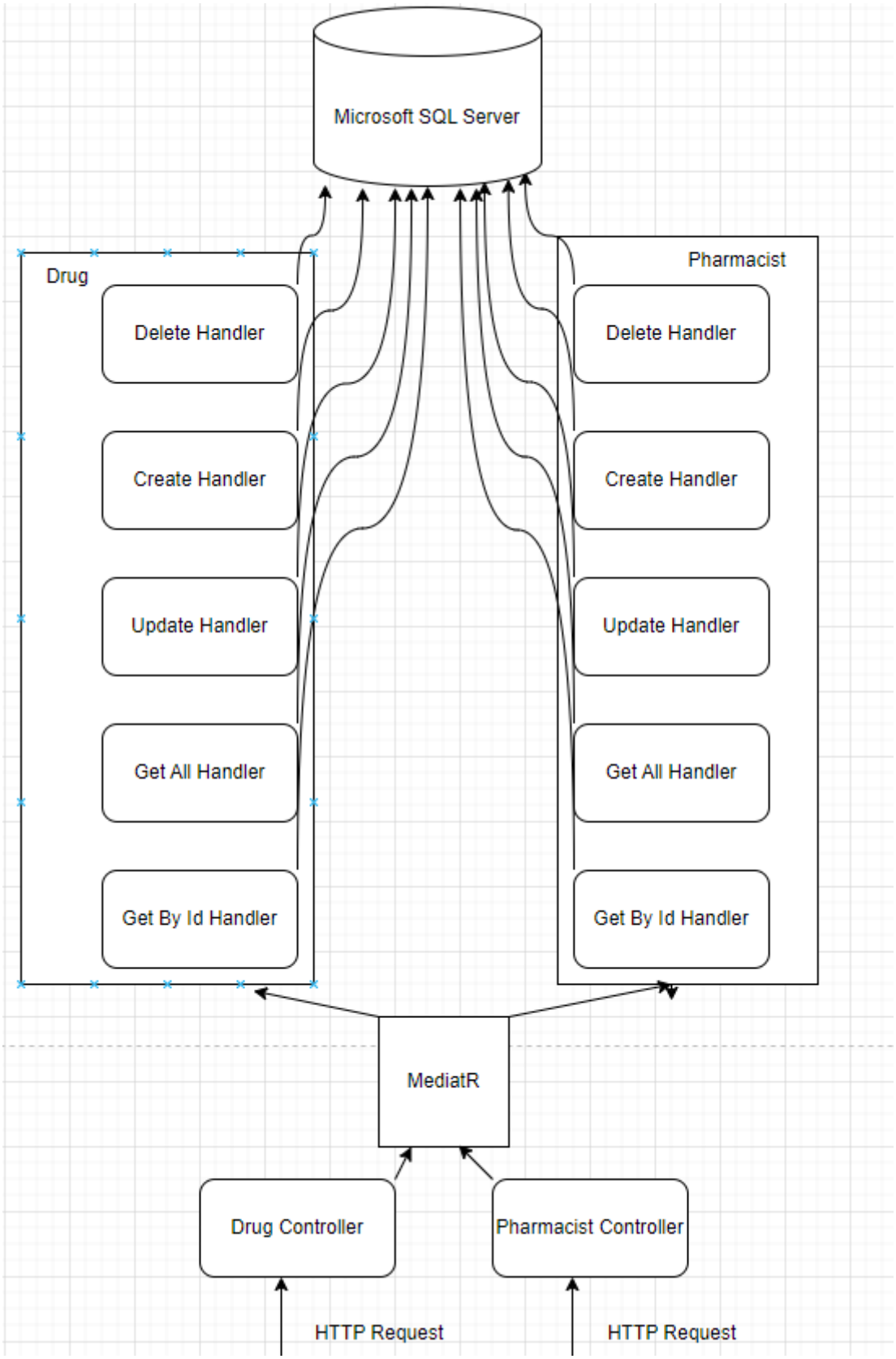

Рисунок 3.8 Неповна частина мікросервісної структури

На рисунку 3.8 зображено внутрішню структуру мікросервісу, аналогічно, як і у двох попередніх мікросервісах, для реалізації ми

обрали CQRS. Головна відмінність даного компоненту від інших мікросервісів полягає, в тому, що він оперує двома контролерами.

Тепер поговоримо трішки детальніше про написання відповідних компонентів у backend частині. Це є частина проекту, яка є однією з найвідповідальніших, оскільки саме від правильності структурування даних залежить кінцевий результат, який буде виведений на екран. Саме тому перед початком розробки самих контролерів, які і відповідають за точки видачі певних даних, потрібно зробити список того, що буде використовуватись на проекті. В цьому випадку дуже доречною стала діаграма варіантів використання, яка фактично й описувала всі методи, які будуть потрібні у проекті. Розглядаючи кожен з цих варіантів було створено методи, які б обробляли певні дані з бази даних і повертали готову колекцію, яку можна використовувати на frontend частині. Окрім того, усі методи були відсортовані по даних, які вони обробляли, і подібні методи, які робили подібні операції з даними були об'єднані в один контролер. За рахунок цього в результаті у одному компоненті вийшло близько 7 контролерів, які грамотно і логічно поділені. В майбутньому при додаванні нового функціоналу буде значно простіше розібратися куди він повинен бути доданий, що свідчить про правильність розробки і поділу backend частини. Також варто зауважити, що усі методи виконуються тільки тоді, коли користувач має певний рівень авторизації. Так наприклад використання методу отримання інформації про клієнта може тільки той лікар, який в даний момент має запис з цим пацієнтом або є його сімейним лікарем. Така система авторизації була створена за допомогою вбудованої Authorization бібліотеки у ASP.NET Core. Суть її полягає в тому, що при вході на сайті, створюється так званий JWT, який зберігає в собі інформацію про користувача, який увійшов в систему та зберігає в собі зашифрований ключ, яким можна підтвердити справжність цього токену. За рахунок того, що цей токен зберігає в собі інформацію про роль користувача

(User, Doctor, Pharmacist або Admin) можна легко перевіряти, який з методів є доступний для цієї ролі, а який ні. Цю перевірку можна написати вручну перевіряючи кожного разу чи роль користувача відповідає потрібній, але щоб не писати один і той самий код багато разів, можна використати атрибут Authorize та вказати всередині значення для відповідного параметру, наприклад Roles=Doctor, який потрібно поставити перед потрібним нам методом.

Тепер перейдемо до Frontend частини веб застосунку. Безсумнівно основна мета цієї частини – це креативність, зручність та привабливий вигляд кожної с сторінок проекту. Для того, щоб почати розробку цих сторінок, спочатку потрібен приблизний дизайнерський план того, як та чи інша сторінка буде виглядати. Для цього перед безпосереднім написанням коду для інтерфейсу сайті, був створений невеликий шаблон того, як будуть виглядати основні сторінки сайту. Шаблон був створений в сервісі Figma, оскільки цей сервіс є достатньо зручним і після створення шаблону можна отримати частковий код сторінки.

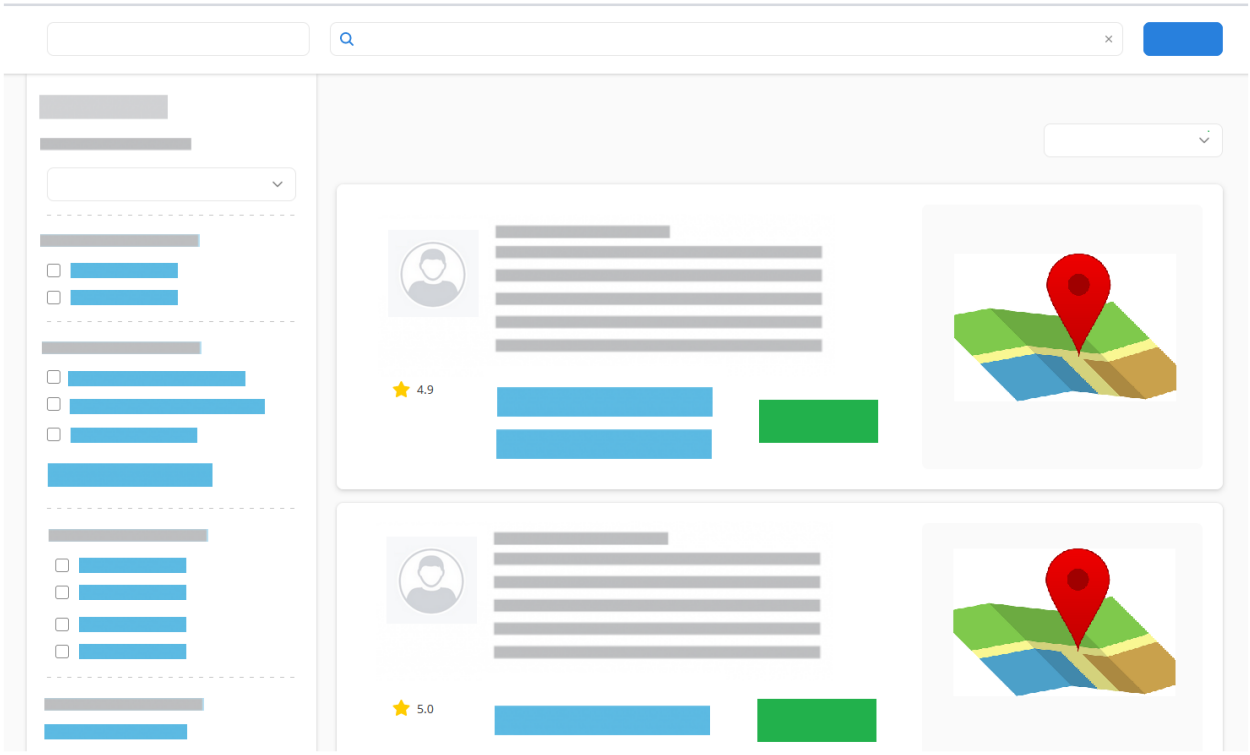

Рисунок 3.9 Макет сторінки пошуку

Одна з переваг використання Figma – це те, що він дозволяє використовувати уже готову палітру кольорів, що значно спрощує вибір основних кольорів для сторінки і сайту загалом. Також, що немало важливо – це те, що Figma пропонує свої зміни в той чи інший компонент сторінки. Так наприклад Figma запропонував використовувати внутрішні елементи у вигляді прямокутника з заокругленими кінцями, що зробило вигляд елементи набагато кращим і цікавішим. За рахунок таких невеликих змін візуалізація сторінки покращується, що безсумнівно сподобається користувачам.

Рисунок 3.10 Макет сторінки профілю

Немало важливим у створенні красивого інтерфейсу відіграла бібліотека стилів – Materials UI. Ця бібліотека має багато різних уже готових компонентів, такі як поля для введення, кнопки підтвердження операції та багато чого іншого.

Повертаючись до загальної структури Frontend частини, варто сказати, що вона складається з основних чотирьох частин – це сам візуальний компонент сторінки, сервісний компонент сторінки, сервіс та інтерсептори. Поговоримо детальніше про кожен з цих пунктів. Візуальний компонент сторінки – це HTML код, який використовує в собі певні стилі і певні готові дані для відображення. Сервісний компонент сторінки – це TypeScript код, який описує логіку поведінки сайту при натисканні на різні кнопки цієї сторінки, також цей компонент відповідає за передачу даних до візуального компонента сторінки. Сервіс – це відокремлений клас, суть якого в тому, щоб отримати дані з Web API та структурувати його в зручному для нього вигляді. Зручність цього сервісу є в тому, що він може застосовуватись одночасно в різних частинах програми, а самі методи цього сервісу зроблені так, щоб перевіряти потрібні ролі у користувача і блокувати запити у випадку не дотримання цих правил. Остання частина цієї структури – це інтерсептори. Інтерсептор – це є певний клас, який перехоплює всі запити, які програма хоча надіслати та додає або зчитує токен до цього запиту. Це потрібно для того, щоб запит на backend частині завжди мав прив’язаний токен, для того щоб завжди була можливість перевірки авторизації користувача. Також це запобігає можливості надсилання запиту, який має бути засекречений і дозволений для перегляду тільки певним користувачам.

Переходимо до останньої частини нашого веб застосунку – це його перенесення в хмарне середовище. Як уже було сказано, для цього був використаний Azure. Перш за все потрібно розуміти, що для перенесення і правильного функціонування нашого застосунку, потрібно створити дві окремі компоненти програми: для Frontend та Backend частини. Тому в Azure створюємо один App Service компонент з відповідним тарифним планом. У моєму випадку сервіс для Angular був створений з

стандартним тарифним планом, оскільки він вимагає більше пам’яті для анімації та кешування зображень.

Наступним кроком буде розгортання сховища для зберігання бінарних файлів. У нашому випадку це буде контролювати Azure BLOB storage. Для нього обираємо кращий тарифний план, оскільки всі файли, яку використовуються в проекті будуть зберігатися тут. Після його створення потрібно додати можливість зберігання та отримання файлів безпосередньо у код наших сервісів. Це не важка операція, оскільки документація Azure допомагає швидко з цим розібратись.

Тепер можемо переходити до розгортання backend частини і розпочнемо ми з однієї з найголовніших компонентів, а саме з розгортання BFF модуля. У хмарному середовищі BFF модуль буде представляти собою дві незалежні частини: App service та API Management Service. Перша частина потрібна для того, щоб мати змогу виконувати не довготривалі та не складні операції загального характеру, наприклад делегування роботи пов'язаної зі електронною розсилкою до Gmail SMTP сервера. Друга частина буде задовольняти усі інші вимоги до BFF модуля за рахунок вбудованого функціоналу, який дозволяє нам зробити все для нашої мети. Не менш важливою частиною є створення сервісів, основних backend компонентів, які відповідають за обробку більшості інформації додатку. Для того щоб побудувати хмарну мікросервісну архітектуру, спочатку потрібно створити кластер де вони будуть знаходитись. Для цього потрібно створити нову ресурс групу та додати до неї Kubernetes services опцію. Тепер залишається лише налаштувати новий кластер та обрати відповідний тарифний план. У нашому випадку, це стандартний тарифний план, який включає в себе можливість auto scale. Це дозволяє нам з коробки забезпечити ефективну роботу кластера з потрібною кількістю вузлів для поточних завдань. Це функція, яка динамічно адаптує кількість вузлів або екземплярів у

кластері на основі поточного навантаження. Воно допомагає забезпечити ефективну роботу кластера, автоматично збільшуючи або зменшуючи його розмір, щоб задовольнити потреби. За допомогою Azure ми також можемо визначити політику та пороги масштабування, які визначають, коли додавати або видаляти вузли з кластера. Ці політики можуть базуватися на метриках, таких як використання CPU, використання пам'яті або довжина черги запитів. У нашому випадку вони будуть базуватися на навантаженні CPU. Коли навантаження зростає, автомасштабування може додавати додаткові вузли для обробки додаткового навантаження, а коли навантаження зменшується, воно може видаляти непотрібні вузли для збереження ресурсів. Завдяки цьому ми оптимізували розподіл ресурсів та витрат. Тепер вона дозволить кластеру динамічно адаптуватись до змін у навантаженні та підтримувати бажаний рівень продуктивності. Та щоб зробити масштабування ще кращим, потрібно також додати опцію вертикального масштабування. Azure дозволяє встановити ліміти базуючись на тарифному плані, який ми обрали. Тому після встановлення цих лімітів наші сервіси зможуть короткочасно збільшити свої ресурси. Проте такі операції будуть коштувати відносно дорого, тому ми зупинимося лише на збільшенні пам’яті. Тепер нам залишається розгорнути у цьому кластері усі потрібні сервіси і перевірити, що вони можуть спілкуватися між собою. Для цього при створенні ми вказали про дозвіл отримання запитів в межах одного кластера.

Також потрібно розгорнути базу даних для того, щоб зберігати усі дані не на локальному сховищі. Для цього створюємо Azure Database for MS SQL Server. Результатом цього хмарного компоненту буде певний сервер на якому буде встановлена MS SQL база даних. Підключення до серверу відбувається за допомогою введення відповідної назви серверу та логіну з паролем до самої бази.

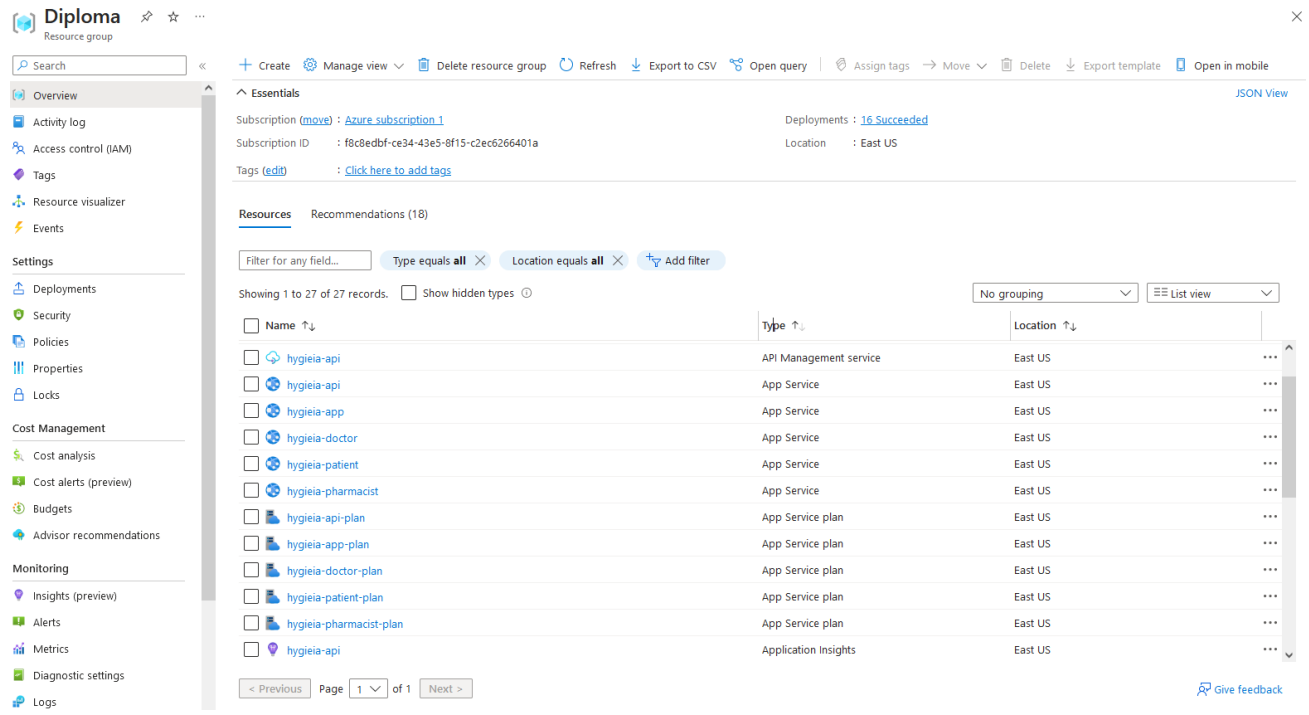

Рисунок 3.11 Ресурси в Diploma групі частина 1

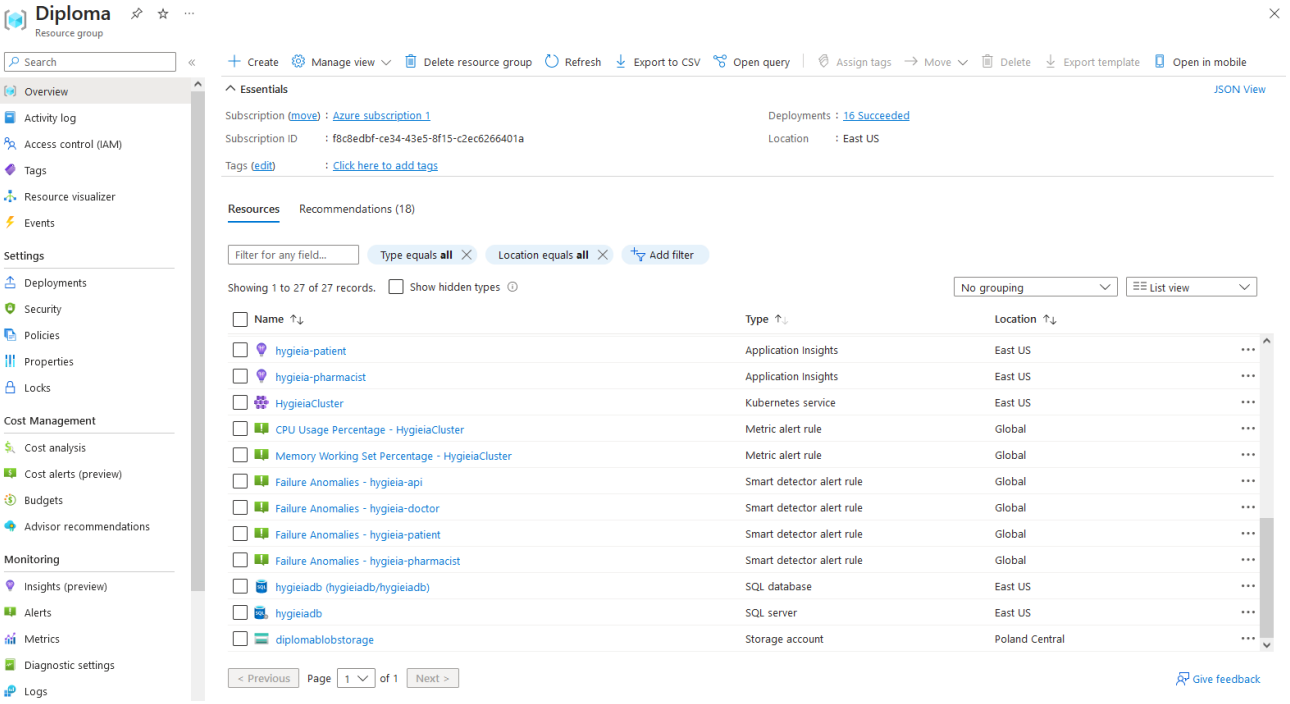

Рисунок 3.12 Ресурси в Diploma групі частина 2

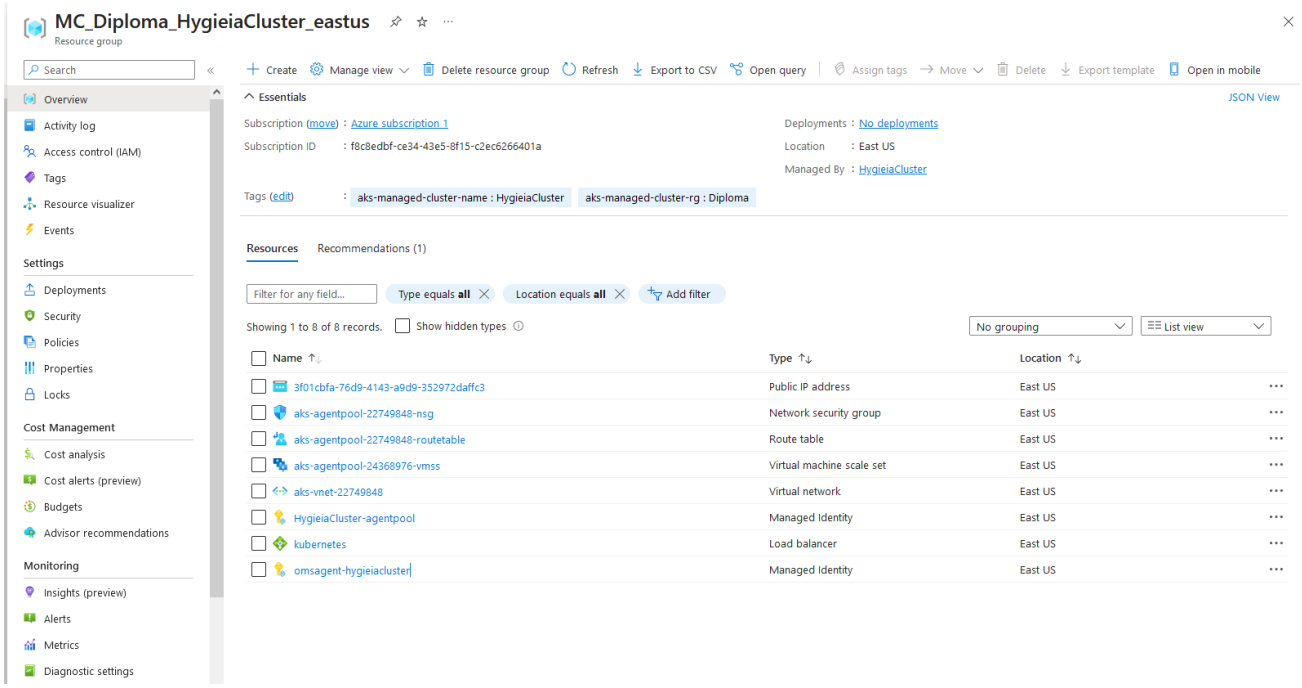

Рисунок 3.13 Ресурси в Diploma Cluster групі

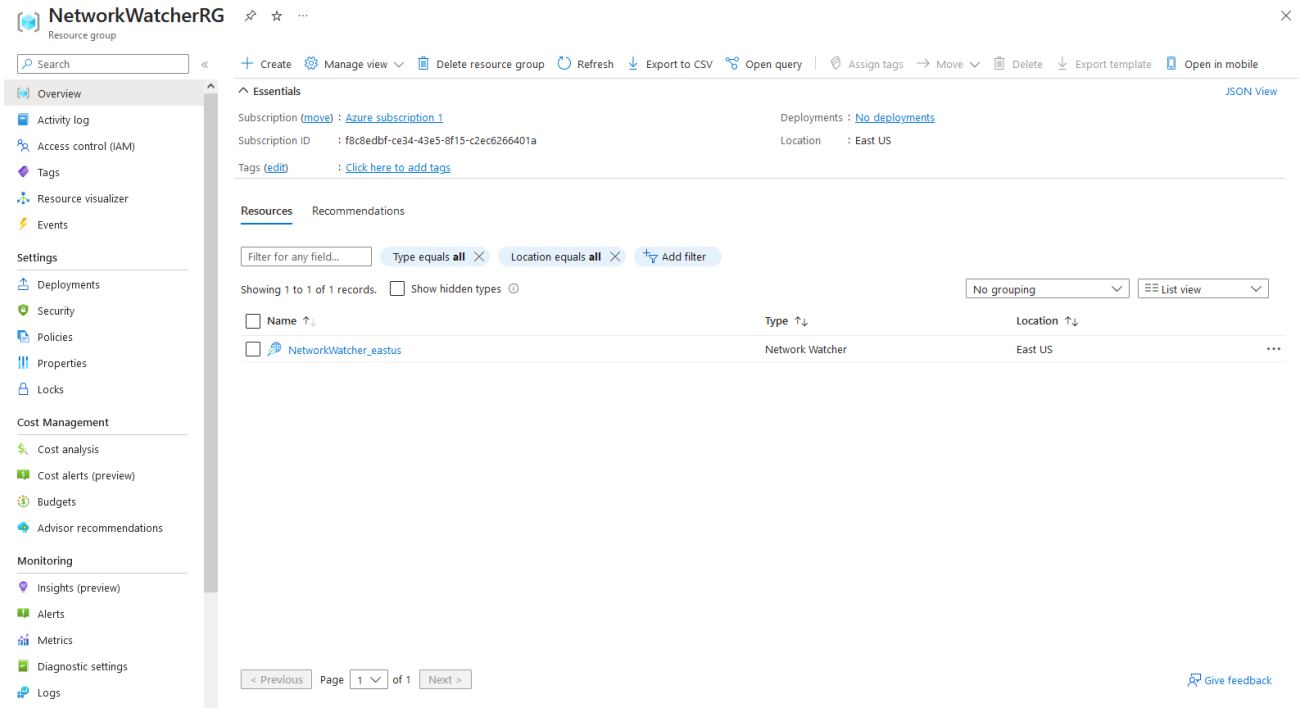

Рисунок 3.14 Ресурси в Network WatcherRG групі

Також потрібно додати додаткові сервіси для спостереження за статусом всіх компонентів в хмарній інфраструктурі. І після виконання цього етапу, було налаштовано декілька опцій для надсилання теперішнього стану того чи іншого компоненту на пошту адміністраторів ресурс груп. Іншими словами, якщо щось в поведінці якогось сервісу зміниться або буде потреба в його масштабуванні то адміністратори одразу будуть повідомлені про це. Це досить зручно, оскільки дається можливість перевірити правильність виконання компонентів та в разі чого полагодити компонент вручну.

Також варто згадати про налаштування Load Balancer для нашого кластера. Він автоматично додається в ресурс групу після створення кластера та не вимагає ніяких додаткових налаштувань для його базового функціонування. Тому достатньо просто ввімкнути його у відповідній вкладці на Azure.

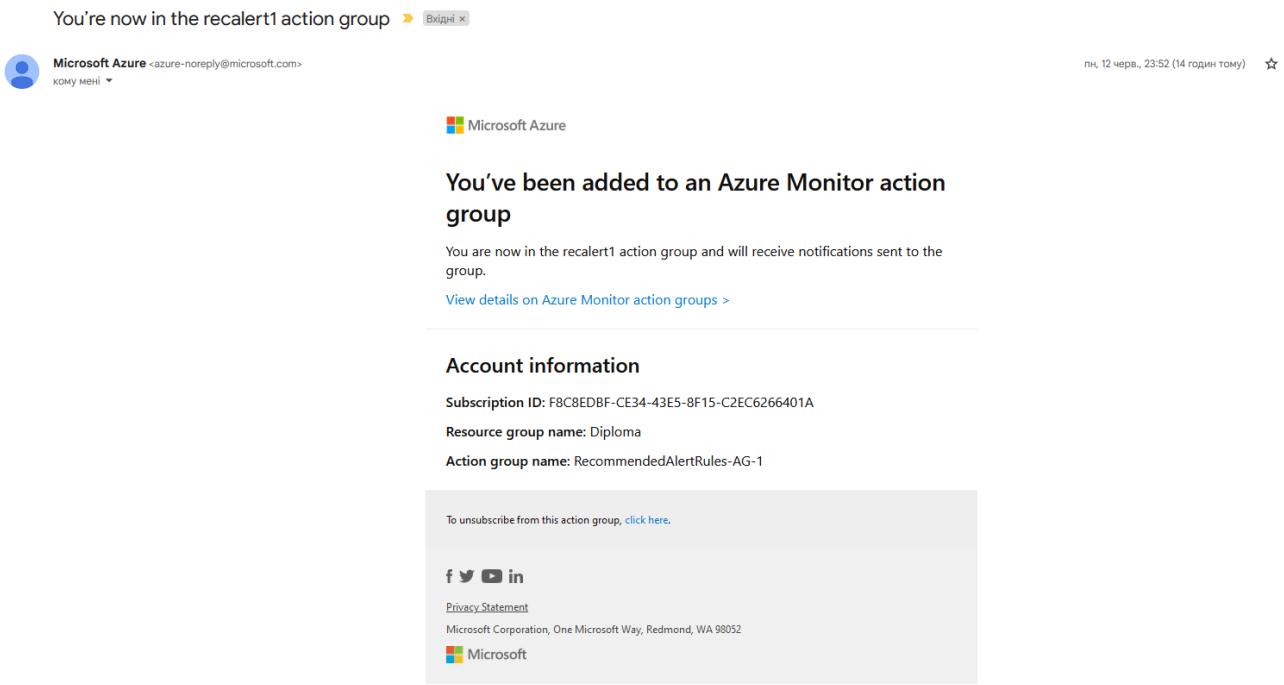

Рисунок 3.15 Лист про моніторинг хмарних компонентів

Важливим моментом налаштування усіх хмарних компонентів є правильне заповнення Firewall правил. За замовчуванням правила не допускають прийом та відправлення даних на різних рівнях інфраструктури, а усі компоненти у нас - на різному. Тому для цього потрібно було заповнити правила так, щоб можна було отримувати та надсилати дані на ті компоненти, які є в одній ресурс групі. Тому потрібно створити ресурс групу і додати всі компоненти до неї. Після цього потрібно додати правило в ресурс групу, що дані між компонентами всередині можуть передаватися без обмежень. Тепер хмарне середовище готове до використання і є доступним за відповідним посиланням.

Також варто згадати два допоміжні сервіси, які розширяють можливості веб застосунку. Перший сервіс – це SendGrid, а другий – це ITextSharp. Перший сервіс використовується для того, щоб надсилати email пошту. Наприклад для підтвердження реєстрації користувача. Окрім того в майбутньому можна буде використовувати розсилку для іншого функціоналу. Другий сервіс допомагає редагувати та створювати .docx файли. Цей сервіс потрібний нам для того, щоб заповняти документи з готовим шаблоном і друк PDF файлу цього документа. Зараз він використовується для лікарів, які заповнюють медичні документи, як результат відвідування пацієнтом лікаря. Також він використовується для роздрукування рецептів, які виписує лікар і отримує фармацевт.

DOCTOR'S NOTE

Name	Olivia Frere	**Gender**	Intege
Email	mpreuvost0@hexun.com	**Age**	5
Phone	(84) 844-9612		

Start Date	08/26/1970	**No. of Days**	5
End Date	08/26/1970		

Medical Diagnosis

Integer ac leo.

Description of the diagnosis

Vestibulum ante ipsum primis in faucibus orci luctus et ultrices posuere cubilia Curae; Nulla dapibus dolor vel est. Donec odio j

Medical Advice/Prescription

Duis bibendum, felis sed interdum venenatis, turpis enim blandit mi, in porttitor pede justo eu massa. Donec dapibus. Duis at velit e

ABC GEN HOSPITAL

Signature of the Doctor

Arther Van Weedenburg

Integer ac leo. P

Integer ac leo. Pell

(84) 844-9612

Рисунок 3.16 Приклад медичного документа

4.2 Структура бази даних

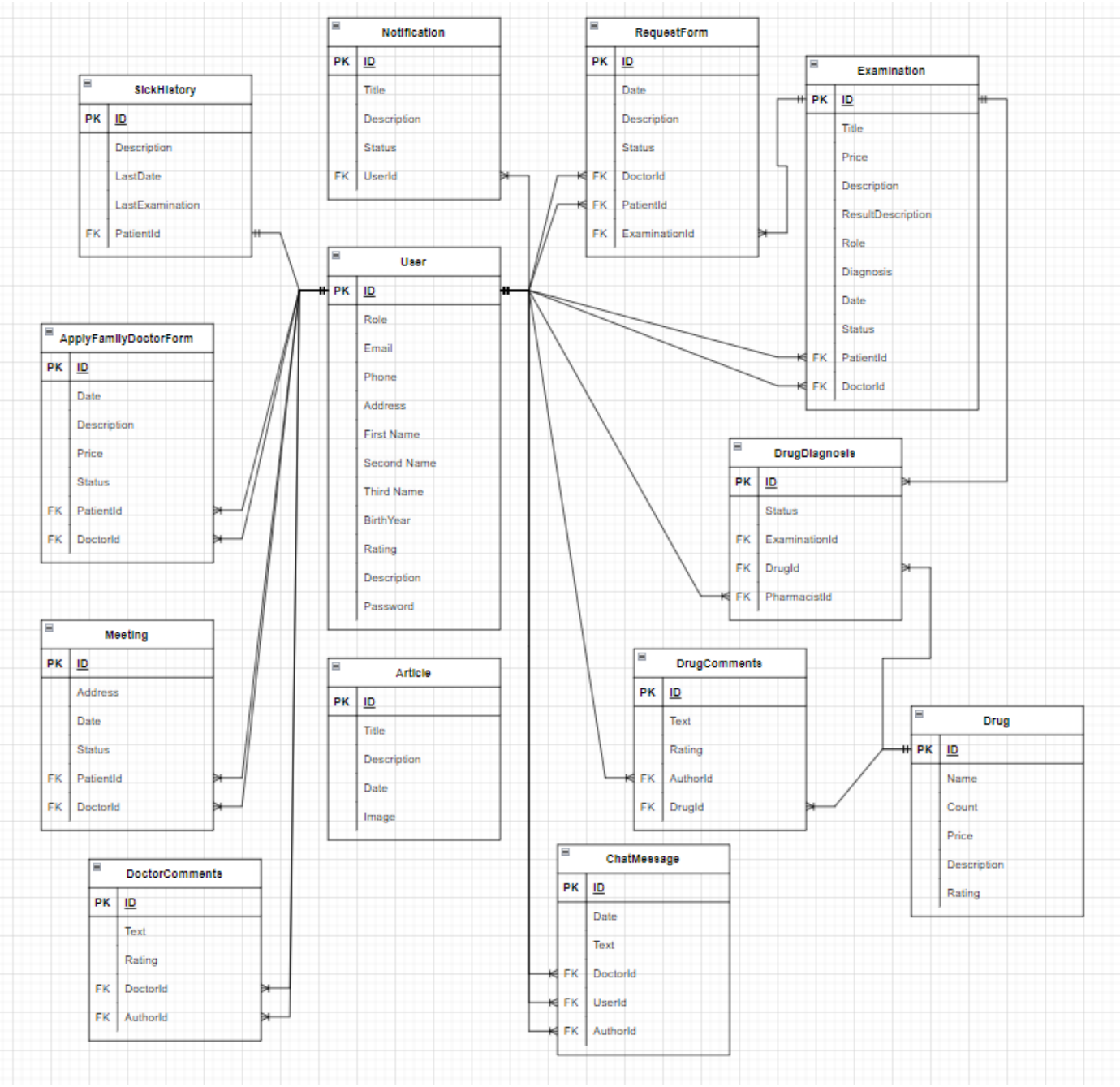

Рисунок 4 ER діаграма

ВИСНОВКИ

У ході виконання даної дипломної роботи було створено програму, яка представляє собою невелику соціальну мережу в межах одного медичного закладу. Створюючи цей веб застосунок, ми багато чого навчилися і багато чого зрозуміли. Адже, щоб створити цей додаток, потрібно було повністю дослідити питання структури лікарні та всіх процесів, які відбуваються всередині. Окрім цього, ми навчилися використовувати нові та сучасні технології розробки програмного забезпечення, що неабияк збільшило наші знання в програмування і медичній справі.

Можна точно сказати, що ця дипломна робота буде дуже добрим прикладом того, як має виглядати сучасна система у кожній лікарні. Безсумнівно подібна система буде дуже корисною і зручною для кожного, хто буде її використовувати. Окрім цього, вона може стати хорошим стартом для розвитку подібної технології в цілому.

Усі використані для розробки методи, матеріали та дослідження описані у наведеній роботі.

СПИСОК ЛІТЕРАТУРИ

1. https://www.usatoday.com/story/sponsor-story/highkey-agency/2021/07/02/top-docs-top-10-list-doctors-look-2021/7829828002/
2. https://www.hhs.gov/hipaa/for-individuals/guidance-materials-for-consumers/index.html
3. https://www.advancedmd.com/medical-billing/software/
4. https://dentist-plus.com/tur-po-sisteme/vhod-v-sistemu
5. https://realpython.com/api-integration-in-python/#rest-and-python-tools-of-the-trade
6. https://docs.microsoft.com/en-us/aspnet/core/tutorials/first-web-api?view=aspnetcore-6.0&tabs=visual-studio
7. https://evergreens.com.ua/ua/articles/uml-diagrams.html
8. https://en.wikipedia.org/wiki/Quicksort
9. https://www.wopop.com/website/template.aspx
10. https://www2.deloitte.com/us/en/insights/industry/health-care/hospital-business-models-of-the-future.html
11. https://www.betterhealth.vic.gov.au/health/servicesandsupport/hospital-staff-roles
12. https://howtostartanllc.com/how-to-build-a-website
13. https://devdocs.io/angular~10/
14. Microsoft Azure Infrastructure Services for Architects by John Savil
15. Building Microservices: Designing Fine-Grained Systems by Sam Newman

ДОДАТКИ

Код програми на Github: https://github.com/ignars3/M_I_Term_Paper

Посилання на готовий сайт: https://hygieia-app.azurewebsites.net