Міністерство освіти і науки України
Тернопільський національний технічний університет імені Івана Пулюя

Факультет комп'ютерно-інформаційних систем і програмної інженерії
(повна назва факультету)
Кафедра комп'ютерних систем та мереж
(повна назва кафедри)

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття освітнього ступеня

бакалавр
(назва освітнього ступеня)

на тему: Комп'ютеризована система електронного консультування пацієнтів з використанням хмарних сервісів

Виконав(ла): студент(ка) *IV* курсу, групи *СІ-41*
спеціальності *123 «Комп'ютерна інженерія»*
(шифр і назва спеціальності)

		Люлька А.В.
	(підпис)	(прізвище та ініціали)
Керівник		Луцків А.М.
	(підпис)	(прізвище та ініціали)
Нормоконтроль		Тиш Є.В.
	(підпис)	(прізвище та ініціали)
Завідувач кафедри		Осухівська Г. М.
	(підпис)	(прізвище та ініціали)
Рецензент		Гладьо Ю.Б.
	(підпис)	(прізвище та ініціали)

Тернопіль 2023

Міністерство освіти і науки України
Тернопільський національний технічний університет імені Івана Пулюя

Факультет *комп'ютерно-інформаційних систем і програмної інженерії*
(повна назва факультету)

Кафедра *комп'ютерних систем та мереж*
(повна назва кафедри)

ЗАТВЕРДЖУЮ
Завідувач кафедри
Осухівська Г.М.
(підпис) (прізвище та ініціали)
«__» _____ 2023 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

на здобуття освітнього ступеня *бакалавр*
(назва освітнього ступеня)

за спеціальністю *123 «Комп'ютерна інженерія»*
(шифр і назва спеціальності)

студенту Люльці Андрію Вікторовичу
(прізвище, ім'я, по батькові)

1. Тема роботи Комп'ютеризована система електронного консультування пацієнтів з використанням хмарних сервісів.

Керівник роботи Луцків Андрій Мирославович канд. тех. наук, доцент
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

Затверджені наказом ректора від «___» _____ *2023* року № _____

2. Термін подання студентом завершеної роботи 22 червня 2023р.

3. Вихідні дані до роботи Комп'ютеризована система електронного консультування пацієнтів з використанням хмарних сервісів.

4. Зміст роботи (перелік питань, які потрібно розробити)
1. Аналіз технічного завдання
2. Проектна частина
3. Практична частина
4. Безпека життєдіяльності, основи охорони праці

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень, слайдів)

1. Діаграма варіантів використання
2. Діаграма активності
3. Схема баз данних
4. Діаграма класів
5. Діаграма послідовностей
6. Схема загальної архітектури системи

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Безпека життєдіяльності, основи хорони праці	Пилипець М.І.		

7. Дата видачі завдання 02.03.2023

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів роботи	Термін виконання етапів роботи	Примітка
1.	Ознайомлення з завданням до кваліфікаційної роботи		*Виконано*
2.	Провести огляд літературних джерел по темі випускної бакалаврської роботи		*Виконано*
3.	Провести порівняльну характеристику програмних продуктів		*Виконано*
4.	Розробити функціональну схему роботи об’єкта проектування. Розробка моделі бази даних		*Виконано*
5.	Практична реалізація об’єкта проектування		*Виконано*
6.	Описати та розробити організаційні заходи спрямовані на поліпшення стану охорони праці		*Виконано*
7.	Виконання завдання до підрозділу «Безпека життєдіяльності»		*Виконано*
8.	Виконання завдання до підрозділу «Основи хорони праці»		*Виконано*
9.	Оформлення кваліфікаційної роботи		*Виконано*
10.	Нормоконтроль		*Виконано*
11.	Перевірка на плагіат		*Виконано*
12.	Попередній захист кваліфікаційної роботи		*Виконано*
13.	Захист кваліфікаційної роботи		*Виконано*

Студент _____ Люлька Андрій Вікторович
(підпис) (прізвище та ініціали)

Керівник роботи _____ Луцків Андрій Мирославович
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

Комп’ютеризована система електронного консультування пацієнтів з використанням хмарних сервісів. // Кваліфікаційна робота освітнього рівня «Бакалавр» // Луцків Андрій Мирославович // Тернопільський національний технічний університет імені Івана Пулюя, факультет комп’ютерно-інформаційних систем і програмної інженерії, кафедра комп’ютерних систем та мереж, група СІ-41 // Тернопіль, 2023 // С.-70, рис.-43, додатки-3, бібліогр.-18.

Ключові слова: IT, C#, Typescript, React, Asp .NET Core, бібліотека, фреймворк, DDD.

Робота складається з вступу, чотирьох розділів, висновку, списку використаних джерел і додатків.

У вступі обгрунтовається актуальність теми, йде опис загальних задач які будуть досліджені протягом розробки. У першому розділі описано загальні пояснення основних принципів роботиз хмарними сервісами, представниками різних хмарних сервісів, а також опис загальних принципів використаних при розробці поставленого завдання. У другому розділі представлено опис програмних продуктів які були використані при розробці, також наведнено детальне роз’нення архітектури системи, принципів та основних правил які були дотримані при проектуванні і розробці об’єкту проектування. У третьому розділі було детально розписано логіку роботи системи з точки зору користувачів системи та показано лістинги коду використовувані в комп’ютеризованій системі.

Результатом дипломного проектування і повнофункціональна комп’ютеризована система, інтерфейс користувача якої реалізований у вигляді Веб сайту, призначена для пацієнтів та лікарів для організації конференцій для наданя пацієнам отримання необхідних знань від лікарів віддалено, без фізичної взаємодії.

ANOTATION

A computerized system of electronic counseling of patients using cloud services. // Qualification work of the educational level "Bachelor" // Andriy Myroslavovych Lutskiv // Ternopil National Technical University named after Ivan Pulyu, Faculty of Computer Information Systems and Software Engineering, Department of Computer Systems and Networks, Group SI-41 // Ternopil , 2023// S.-70, fig.-43, appendices-3, bibliogr.-18.

Keywords: IT, C#, Typescript, React, Asp .NET Core, library, framework, DDD.

The work consists of an introduction, four chapters, a conclusion, a list of used sources and appendices.

In the introduction, the relevance of the topic is justified, there is a description of the general problems that will be investigated during the development. The first chapter describes general explanations of the basic principles of working with cloud services, representatives of various cloud services, as well as a description of the general principles used in the development of the task. The second section presents a description of the software products that were used in the development, as well as a detailed breakdown of the system architecture, principles and basic rules that were followed during the design and development of the design object. In the third chapter, the logic of the system's operation from the point of view of the system's users was described in detail and the code listings used in the computerized system were shown.

The result of the diploma design is a fully functional computerized system, the user interface of which is implemented in the form of a Web site, intended for patients and doctors to organize conferences to provide patients with the necessary knowledge from doctors remotely, without physical interaction.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ СКОРОЧЕНЬ, ОДИНИЦЬ, СИМВОЛІВ, ТЕРМІНІВ І ПОЗНАЧЕНЬ.

IT – інформаційні технології

JSON – JavaScript Object Notation

СУБД – Система управлінням бази данних

ПК – персональний комп’ютер

HTML - HyperText Markup Language

CSS – Cascading Style Sheets

Url - Uniform Resource Locator

UI – User Interface

CI/CD – Continuous Integration/ Continuous Delivery

ВСТУП

За останні 20 років у світі спостерігається дуже великий приріст нових технологій для автоматизацій підприємств, установ, приватних бізнесів тощо. Оскільки, з часом бізнес стає дедалі складнішими і також сильно зростає складність проектування бізнес логіки таких систем, тому з'явилася велика кількість різних програм, технологій та засобів для спрощення проектування таких систем. Автоматизація таких великих підприємств стала невід'ємною частиною ринку, оскільки дає змогу в рази спростити роботу власникам, а також зменшити кількість осіб для організації роботи підприємства. Також сильно виріз попит на хмарні технології, оскільки це дає гарантії безвідмовної роботи спроектованої за розгорнутої системи 99.9% часу в місяць і попри велику вартість своїх послуг, хмарні технології є надзвичайно гнучкими оскільки дозволяють налаштовувати тарифний план під конкретні вимоги систему і зменшити зайві затрати. Також ключовою перевагою популярності хмарних сервісів є те, що фізичне розміщення серверу вашої системи в Data центрі можна обрати самому доступну зону, що може додатково пришвидшити час оброблення певного запиту сервером.

На сьогоднішній день практично всі рішення проблем з вибором платформи є Web технології. Оскільки, на кожному комп'ютері є браузер у якому достатньо ввести URL посилання проекту і почати його використання без необхідності скачувати та встановлювати різні додатки, як у випадку з мобільною розробкою чи розробкою програм на ПК. Тому для відображення UI частини системи був використані Web технології.

Розвиток медицини сильно пришвидшився з появою систем автоматичного обліку пацієнтів у великих міських та обласних лікарнях та поліклініках, тому розвиток інформаційних систем з використанням хмарних технологій завжди буде ставати дедалі потрібнішим. Особливо актуальним завжди буде можливість огляду пацієнта з дому без фізичного контакту, що могло б також зменшити ризик зараження лікаря від контакту з пацієнтом, тому систему електронного консультування пацієнтів завжди залишатиметься потрібним на ринку.

Для створення система були досліджені та використані наступні середовища розробки:

– visual Studio 2022 Enterprise Edition та Rider - для розробки backend частини системи;

– visual studio Code та WebStorm - для розробки frontend частини системи;

– microsoft Sql Management Studio – СУБД для розробки та адмініструванням бази даних SQL Server.

У дипломній роботі пропонується створити систему електронного консультування пацієнта з використанням хмарних технологій Azure. Для входу в систему потрібно пройти просту реєстрацію, заповнивши відповідні поля і створивши надійний пароль, для цього можна використати будь який зручний браузер (Google Chrome, Microsoft Edge, Mozilla), те саме можна зробити навіть з мобільного телефону, що дозволяє системі бути загальнодоступною. Інтерфейс дуже простий та інтуїтивно зрозумілий і не містить ніякої зайвої інформації.

РОЗДІЛ 1 АНАЛІЗ ТЕХНІЧНОГО ЗАВДАННЯ

1.1 Характеристика об’єкта проектування

Об’єктом проектування виступає система організації відео консультацій на основі хмарних сервісів, яка також включає можливість отримання швидкої відповіді від підключеного онлайн моделі штучного інтелекту, яка зможе безплатно в формі переписки надати швидку пораду при конкретному захворюванні.

Актуальність створення такої системи зумовлена надзвичайно великою потребою ринку автоматизувати більшість процесів, які досі вирішувалися людськими ресурсами. За допомогою такої системи пацієнт матиме змогу в любий час отримати необхідну консультацію від лікаря в любій точці світу, тобто в випадку незадоволеності медициною в інших країнах, пацієнт матиме змогу без проблем отримати консультацію від лікаря з його власної країни.

Ідея віддаленого лікаря вперше була запроваджена на початку 1960 років з ветеринарами на сільськогосподарських тваринах. І вже в 1967 році Кеннет Берд запровадив цей метод лікування у Массачусетській лікарні. З тих пір цей спектр послуг віддаленого лікаря значно розширився, і тепер, у 2018 році, в епоху розумних технологій, громадяни в усьому світі відчувають гостру потребу.

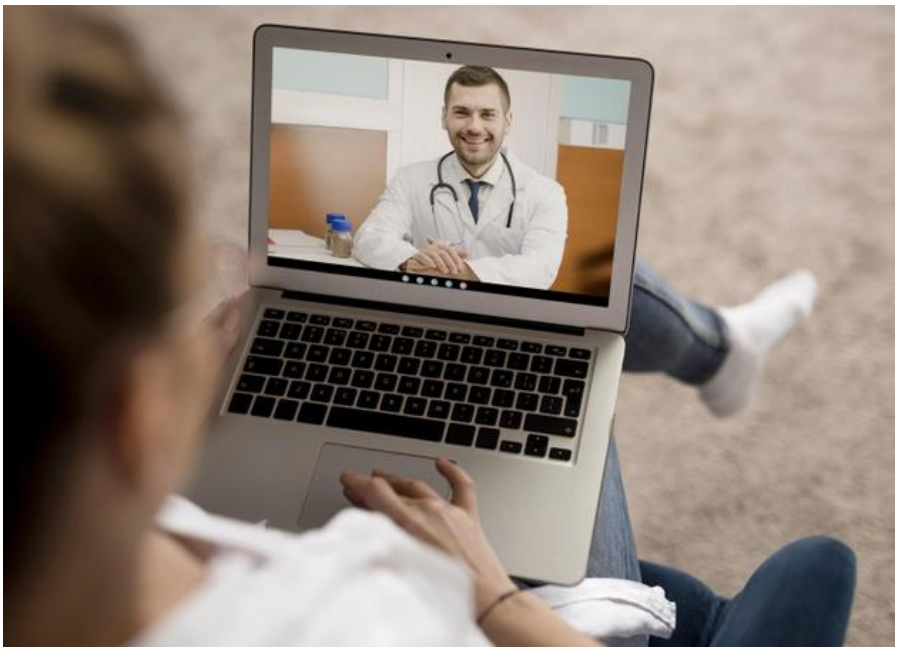

Рисунок 1.1 - Онлайн консультація з лікарем

На рис. 1.1 зображено приклад проведення онлайн консультації з лікарем.

За даними Statista, у 2018 році понад 36% населення світу користуються смартфонами і комп’ютерами, що безперечно призвело до відмови від консультацій лікарів в офісі. Не всі, але більшість випадків можна вирішити здалеку за допомогою простих відеодзвінків пацієнтів із дому. Фізичні огляди можна виконати відповідно до вказівок лікаря, а рекомендовані дослідження можна надіслати лікарю пізніше. Навіть лікарю не обов'язково бути в своєму кабінеті, щоб провести консультацію з пацієнтом.

Дистанційний лікар може легко запропонувати як планові, так і позапланові візити без зниження якості послуг як для цілей лікування, так і для подальшого спостереження. Хоча зазвичай перший візит до лікаря необхідний для загальної оцінки стану здоров’я, подальші спостереження через телекомунікаційні засоби зв’язку може ефективно звести до мінімуму непотрібні відвідування.

Особливо це може бути корисно в таких випадках:

– будь-яка надзвичайна ситуація, яка потребує швидкого вирішення, щоб зменшити ймовірність захворюваності та смертності, пов’язаної із затримкою в доїзді до медичного закладу;

– географічні бар'єри, наприклад, клініка розташована далеко, поганий транспорт тощо;

– несприятлива погода (наприклад, сильний снігопад, дощ);

– особи з обмеженими фізичними можливостями, яких транспортування до лікарні є обтяжливим завданням;

– економічна недостатність, коли вартість проїзду є додатковим тягарем.

Недалекий час, коли послуги телемедицини замінять госпіталізацію в багатьох випадках, оскільки медичні інструментальні технології просуваються до винаходу переносних пристроїв і пристроїв, які можна носити на собі. У цьому випадку це може звести до мінімуму інфекцію, отриману в лікарні, а також передачу інфекції від пацієнта до лікаря. Оскільки пацієнту не потрібно виходити з дому, він або вона може активно контролювати домашні справи, тоді як госпіталізація практично ізолює пацієнта. Перебування в комфортних домашніх умовах сприяє швидкому одужанню.

1.2 Аналіз обраної архітектури

Для розробки системи було прийнято рішення використовувати мікросервісну архітектуру розробки систем, через її надійність, здатність швидше та якісніше обробляти великі запити на відміну від монолітної архітектури.

Існує загалом два основні принципи створення великих систем, а саме:

– Монолітна – тип архітектури системи у якій компоненти дуже тісно зв’язані і використовують єдину вхідну точку в систему(entry point) для обробки даних, для авторизації і реєстрації, а також для комунікації через Web sockets. Такий тип архітектури є відносно швидкий і відповідно дешевий в створенні та підтримці, але має потенційні проблеми, а саме: погана продуктивність роботи великому навантажені, необхідність повної зупинки системи при будь якій проблемі яка вимагає її перезапуск.

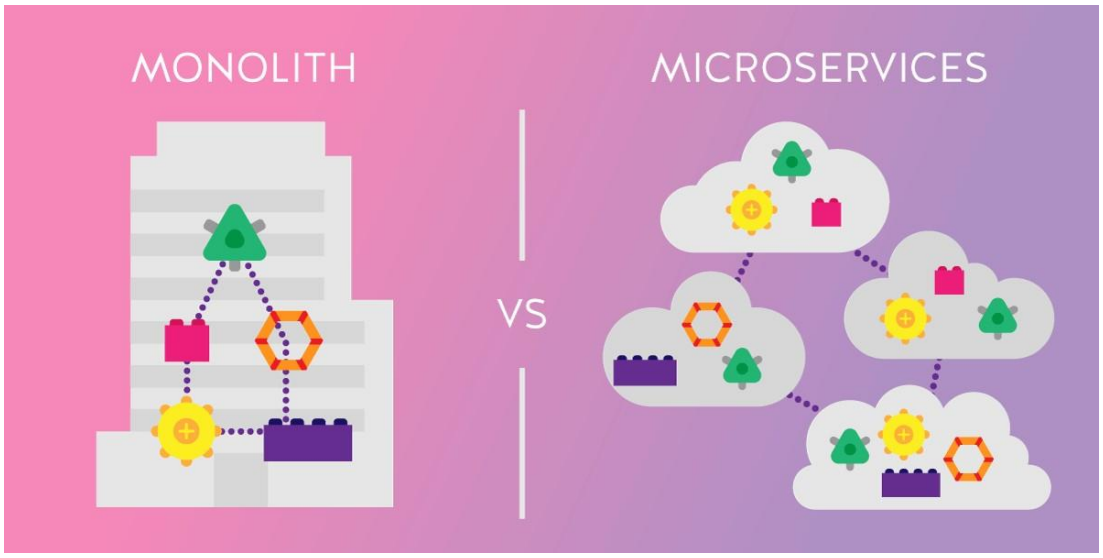

Рисунок 1.2 – Візуальне порівняння мікросервісної та монолітної архітектур

– Мікросервісна (сервіс орієнтована) - це тип архітектури системи при якому всі компоненти розбиті на повністю незалежні фрагменти і здатні працювати повністю самостійно. Тобто на відміну від монолітної, у системі усе навантаження є чітко розділеним і це дозволяє при перезапуску одного сервісу, все ще використовувати систему, і також якщо є велике навантаження, система може за необхідності створити ще один екземпляр сервісу який обробляє ці складні запити і система буде розширена і швидкість обробки запиту не буде збільшуватись зі збільшенням кількості клієнтів, на рис. 1.2 можна побачити візуально різницю мікросервісної архітектури від монолітної. Також існують певні недоліки у цієї системи, такі як суттєво довготриваліша робробка, оскільки ініціалізовувати кожен сервіс окремо займає набагато більше часу, тому відповідно це дорожче ніж монолітна архітектура. Також потрібен великий досвід для створення CI/CD для того щоб можна було автоматично доставляти написаний код замовнику без зупинки роботи системи, тому зазвичай при побудові мікросервісних архітектур залучають додаткових інженерів DevOps.

Мікросервісна архітектура повністю відповідає всім очікуванням які були поставлені на етапі проектування системи.

1.3 Загальний опис інфраструктури розгортання системи

Для того щоб опублікувати систему в глобальний інтернет була використана платформа надання хмарних послуг Azure, яка нараховує близько 200 продуктів і хмарних сервісів. Також він пропонує велику колекцію служб, яка включає можливості платформи як послуги (PaaS), інфраструктури як послуги (IaaS) і можливості служби керованої бази даних.

Azure, як і інші хмарні платформи, покладається на технологію, відому як віртуалізація. Більшість комп'ютерного обладнання можна емулювати програмно. Комп'ютерне обладнання — це просто набір інструкцій, які реалізовані на рівні процесора. Рівні емуляції використовуються для відображення інструкцій програмного забезпечення в інструкції апаратного забезпечення. Рівні емуляції дозволяють віртуалізованому апаратному забезпеченню працювати в програмному забезпеченні, як і саме апаратне забезпечення.

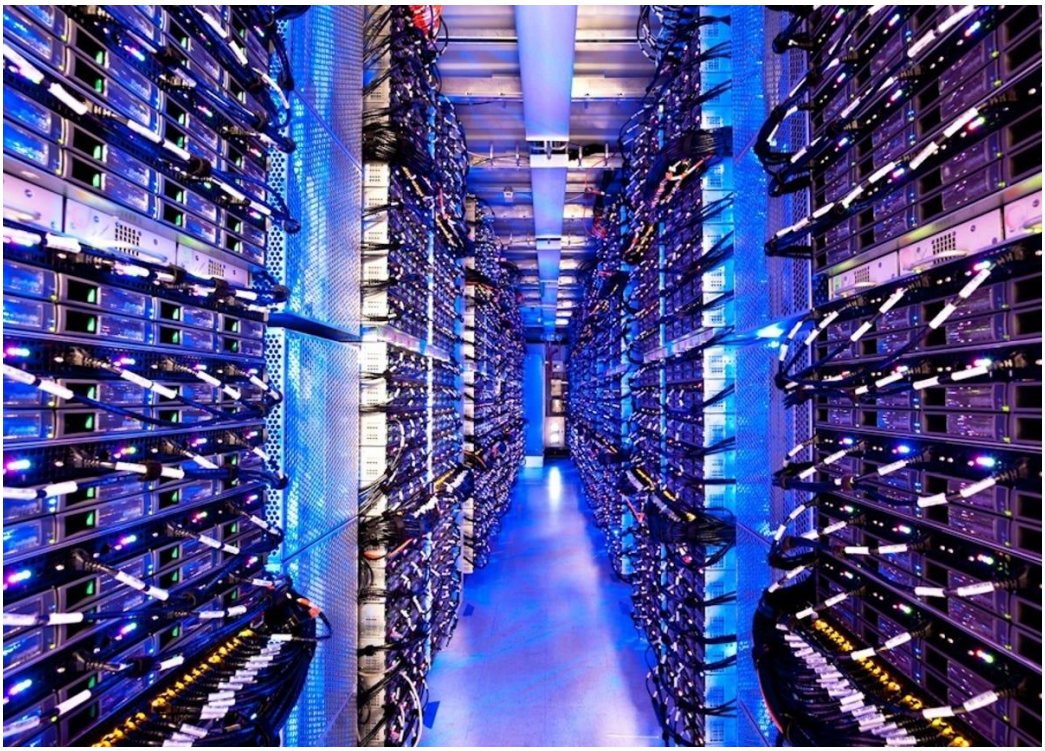

Рисунок 1.3 - Дата центр Azure з середини

По суті, хмара — це набір фізичних серверів в одному або кількох центрах обробки даних. Центри обробки даних використовують віртуалізоване обладнання для клієнтів. На рис. 1.3 зображено один з дата центрів Microsoft Azure розміщених в південній Америці. Такі дата центри створенні по всьому світу для забезпечення безперебійної і високошвидкісної роботи багатьох комп'ютерних систем.

Рисунок 1.4 - Дата центр Azure ззовні

На рис. 1.4 можна побачити один з Microsoft дата центрів ззовні. Усередині кожного центру обробки даних є набір серверів, розміщених у серверних стійках. Кожна серверна стійка містить багато блейд-серверів і один мережевий комутатор. Вони забезпечують підключення до мережі та блок розподілу живлення (PDU), який створює живлення. Стелажі іноді групуються разом у більші блоки, також відомі як кластери.

Серверні стійки або кластери вибираються для запуску віртуалізованих екземплярів обладнання для користувача. Однак на деяких серверах працює програмне забезпечення для керування хмарою, відоме як контролер тканини. Контролер тканини — це розподілена програма з багатьма обов'язками. Він

розподіляє служби, контролює працездатність сервера та запущених на ньому служб і відновлює роботу серверів у разі їх збою.

Кожен екземпляр контролера Fabric підключений до іншого набору серверів, на яких працює програмне забезпечення оркестрування хмари, який називається інтерфейсом. На передній частині розміщено веб-сервіси, RESTful API і внутрішні бази даних Azure, які використовуються для всіх функцій у хмарі.

Наприклад, зовнішня частина містить служби, які обробляють запити клієнтів. Запити виділяють ресурси та служби Azure, наприклад віртуальні машини та Azure Cosmos DB. Інтерфейс перевіряє чи має користувач право розподіляти запитані ресурси. Якщо так, інтерфейс перевіряє базу даних, щоб знайти серверну стійку з достатньою місткістю, яка дає вказівку контролеру виділити ресурс.

Azure — це величезна колекція серверів і мережевого обладнання, на якому працює складний набір розподілених програм. Ці програми організовують конфігурацію та роботу віртуалізованого обладнання та програмного забезпечення на цих серверах. Оркестровка цих серверів робить Azure настільки потужним. З Azure користувачам не потрібно підтримувати та оновлювати своє обладнання, оскільки Azure робить це за лаштунками. Звичайно існують послуги для самостійного обслуговування фізичного обладнання сервера з використанням тільки самого інтерфейсу управління від Azure, такий вид послуг називається “On Premise”.

1.4 Аналіз архітектури програмної реалізації комп’ютеризованої системи

1.4.1 CQRS принцип

CQRS (Command and Query Responsibility Segregation) - принцип написання програмного коду коли всі запити розділені за їх значенням. У цьому принципі існують дві сутності “Commands” і “Queries”.

Queries – це всі HTTP запити, які супроводжуються вивантаженням даних з бази і поверненням на UI, тобто get-запити, при необхідності з використанням різних складних і простих фільтрів та з вказанням порядку сортування.

Commands – це всі HTTP запити, які супроводжуються зміною наявних даних у базі даних. До цього списку можуть відноситись: create, update та delete запити.

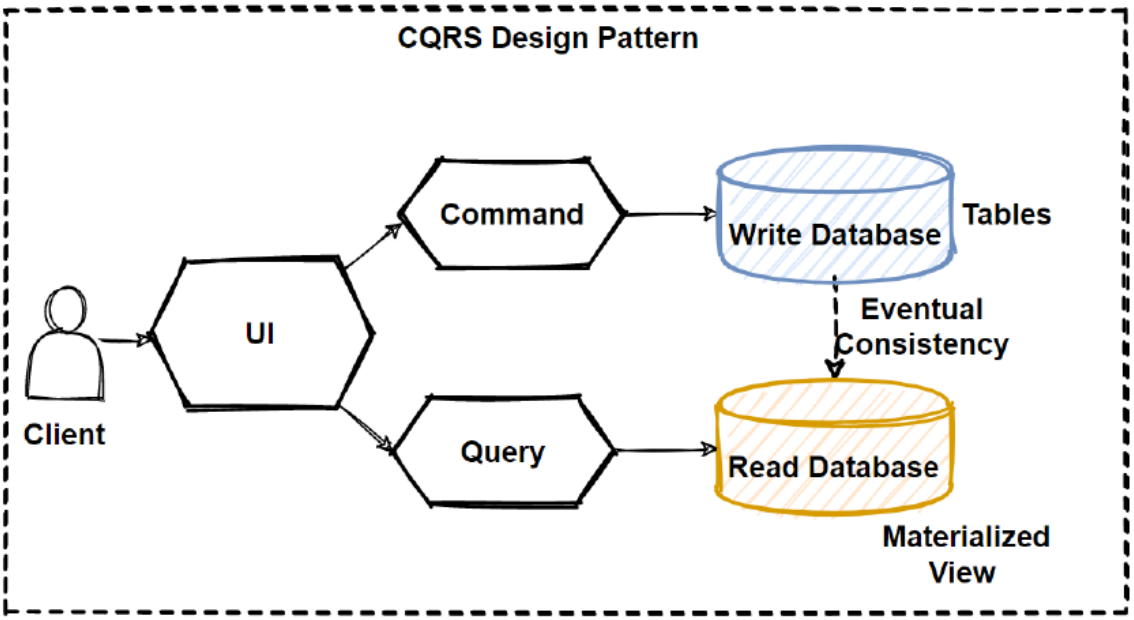

Рисунок 1.5 – Візуальний вигляд CQRS патерну

Цей патерн проектування архітектури написання програмного коду допомагає збільшити його читабельність і полегшує знаходження різних помилок у коді. На рис. 1.5 є візуальна картина принципу роботи CQRS патерну. Для того щоб розрізняти об’єкт який відноситься до кожного типу HTTP запиту, на проектах до них зазвичай додають суфікси Query та Command.

1.4.2 Багаторівнева архітектура

Для того щоб завжди мати чітке розуміння які частини програми яку роль виконують, у системі була використана багаторівнева архітектура.

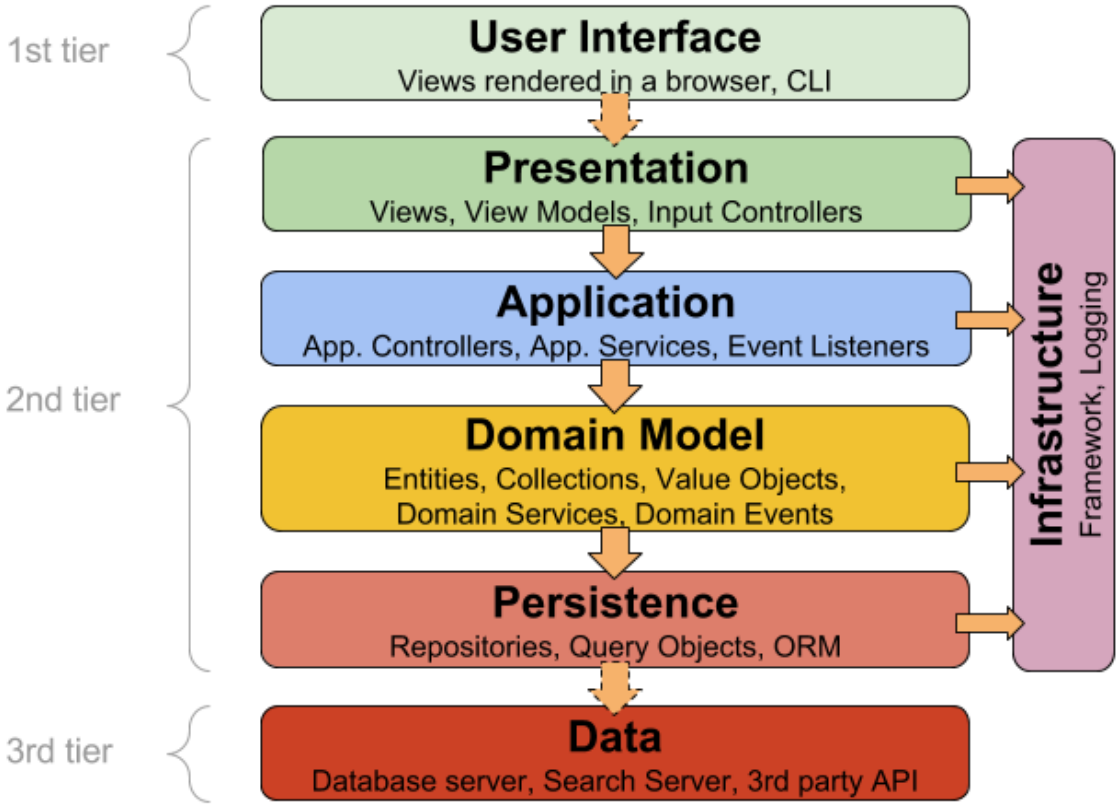

Рисунок 1.6 – Приклад багаторівневої архітектури

Зазвичай виділяють три основні рівні:

– Рівень представлення (Presentation layer) – рівень потрапляння даних у систему і виходу з цієї системи. У поточній систему, для розробки серверної частини кінцеві точки (endpoints) використовувались як рівень представлення.

– Рівень логіки системи (Business Logik Layer) – рівень у якому описана вся логіка системи, всі основні операції які є особливими конкретно для цієї системи.

– Рівень доступу до даних (Data Access) – рівень на якому повинен бути імплементований спосіб доступу до даних і взаємодія з базою даних.

На рис. 1.6 зображено приклад розбиття системи та рівні.

1.4.3 аналіз підходу до моделювання логіки системи.

Оскільки, логіка системи це є найскладніша та найпотрібніша частина системи, тому, відповідно, вона повинна бути написана правильно і працювати безпомилково. Для цього було створено багато різних способів для спрощення

складності написання складних бізнес завдань. Один з таких способів є предметно-орієнтоване проектування (Domain Driven Design).

Предметно-орієнтоване проектування (DDD або Domain Driven Design) – це підхід який використовується до проектування і моделювання складних об’єктно орієнтованих систем.

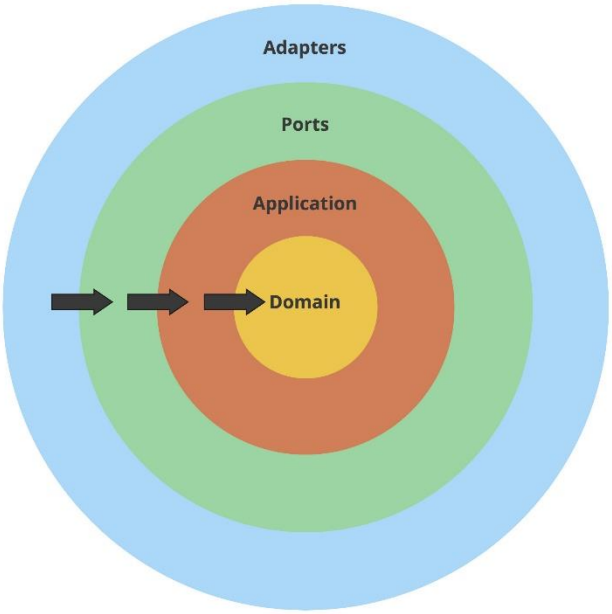

Рисунок 1.7 –Приклад графічного відображення предметно-орієнтованного проектування

Особливістю цього способу проектування, є те що вся бізнес логіка повинна бути написана безпосередньо в моделі, що наближає її максимально до того як вона працює в житті, і це суттєво спрощує розуміння системи. На рис. 1.7 відображено, що вся система розташовується навколо предметної області, де і зібрана вся найскладніша і ключова логіка системи.

РОЗДІЛ 2 ПРОЕКТНА ЧАСТИНА

2.1 Опис алгоритму роботи кожного мікросервісу

У системі організації онлайн консультацій використовується мікросервісна архітектура, тому існує певна кількість сервісів, кожен з яких має свій власний алгоритм роботи.

2.1.1 Опис алгоритму роботи MailApi сервісу

Основний сервіс у якому зібрана вся маніпуляція з даними. Тут реалізований алгоритм, створення, видалення та редагування різних об’єктів системи, таких як: консультація, хвороба та спеціалізований орган.

Алгоритм роботи даного серверу наступний, коли запит приходить на початку відбувається перевірка токену доступу, якщо він правильний, тоді відбувається перехід до самої бізнес логіки системи, якщо ні, то користувач отримає помилку з кодом 401, що ми не авторизовані, або 403 що у авторизованого користувача недостатньо прав для виконання даної процедури. У частині системи де описана бізнес логіка, запит проходить через конвеєр валідацій, для того щоб кожна операція була виконана тільки якщо на вхід були передані правильні дані і все за правильною логікою, як було задумано і як працює дана комп’ютеризована система. Якщо перевірка не проходить, то користувач отримає 400 код помилки і також повідомлення з поясненням, що було зроблено неправильно і чому користувач отримав цей код помилки. Коли перевірка вхідних даних пройшла успішно, результат повертається до користувача віддаючи йому відповідь з кодом 200 що означає що результат був оброблений успішно. Загалом згідно архітектурного планування системи, в системі повинні бути бути присутні фотографії хвороб, і вони будуть реалізовані

в подальшому.

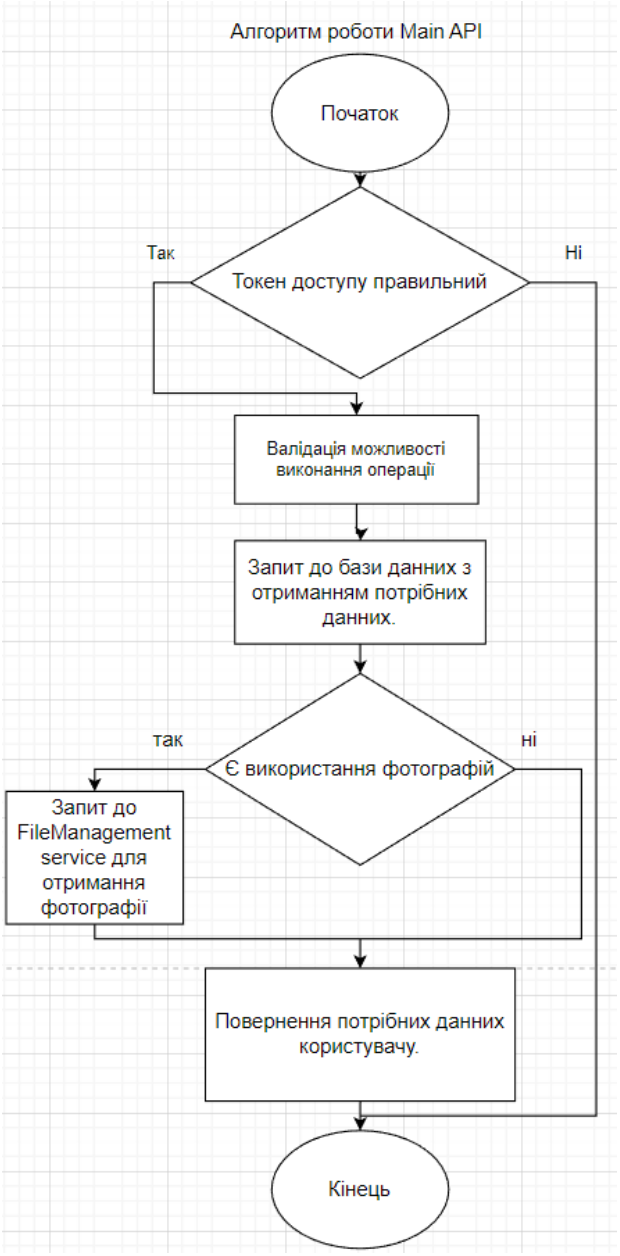

Рисунок 2.1 – Блок-схема алгоритму роботи MainApi сервісу

Перевірка на використання фотографій яку видно на рис. 2.1 повинна була кидати запит окремому сервісу FileManagmentAPI для отримання url посилання з фотографією але в поточній імплементації системи він не використовується.

2.1.2 Опис алгоритму роботи IdentityService сервісу

Оскільки система працює з персональними даними, і має персональний кабінет користувача, система повинна забезпечувати відповідний рівень безпеки для запобігання їх викраденню і сторонньому використанню. Алгоритм роботи даного сервісу наступний: він отримує введені користувачем дані, робить запит до бази щоб переконатись чи дані введені цим користувачем справді правильні, якщо це так то Identity сервіс віддає спеціальний токет (стрічка символів закодованих алгоритмом base64 і відображені у вигляді набору символів 16-ової системи числення), якщо ні то просто видає помилку, що авторизаційні дані були введені неправильно. Цей алгоритм зображений на рис. 2.2.

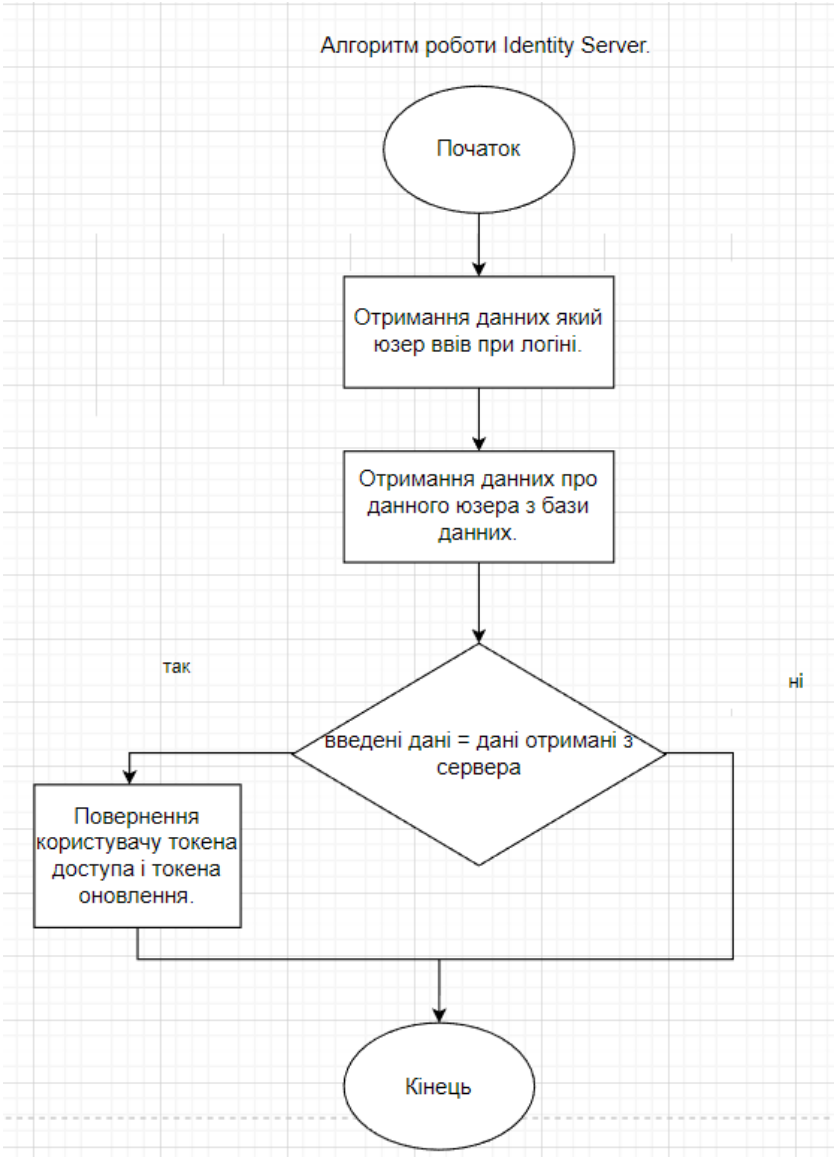

Рисунок 2.2 – Блок-схема алгоритму роботи IdentityService сервісу

Оскільки, запити по авторизації можуть бути дуже частими, оскільки користувачі можуть вводити неправильні дані багато раз і кожна спроба це окремий запит на сервер який витрачає його ресурси, було вирішено створити окремий сервіс для обробки запитів авторизації і автентифікації.

Приклад токену доступу можна побачити на рис. 2.3. Отриманий токен містить певну яка є зашифрованою з використанням технології цифрового підпису. Ця технологія називається JWT.

Рисунок 2.3 – Json відповідь отримана з IdentityService сервісу

Кожен токен містить певну інформацію всередині, за цією інформацією всі сервіси мають змогу перевіряти чи користувач який робить запит з цим токеном може виконати дану операцію чи ні. Час життя токену доступу обмежений, на рис. 2.3 можна побачити що цей токен буде актуальним 86400 секунд що дорівнює одній повній добі, це було зроблено для того щоб ніхто не зміг викрасти цей токен набагато пізніше ніж він був створений. Також при кожному отриманні токену доступу ми також отримуємо токен оновлення, потрібний нам для того щоб ми змогли оновити токен доступу коли він стане застарілий не вводячи пароль та логін знову.

Токен доступу складається з 3 частин:

– Алгоритм і тип токену;

– Дані;

– Інформація про цифровий підпис.

У полі з даними всередині токену можна знайти інформацію, яка передається на потрібний сервіс.

Рисунок 2.4 – Розшифрований токен доступу на “ HTTPs://jwt.io/”

Ми можемо легко побачити всю інформацію з токену на будь якому ресурсі який вміє розшифровувати base64. На рис. 2.4 ми можемо бачити дані які були в токені доступу, а саме ім’я користувача, роль, адреса електронної пошти та час через який токет перестане бути доступним.

2.1.3 Опис алгоритму роботи SessionHolding сервісу

Даний сервіс використовувався для організації web socket з’єднань та для обміну повідомленнями між користувачами конференції та для переривання всіх зв’язків коли конференція завершена.

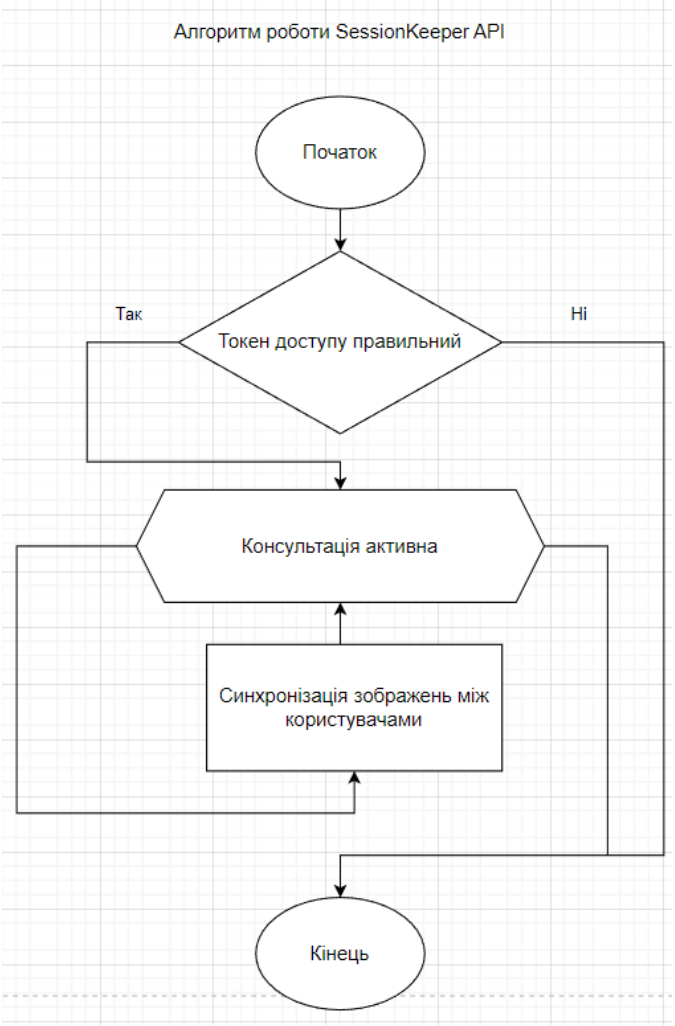

Рисунок 2.5 – Блок-схема алгоритму роботи SessionKeeper сервісу

На рис. 2.5 зображено алгоритм роботи даного сервісу. Комунікація через Web sockets відбувається тільки у цьому сервісі. Також через цей сервіс відбувається обмін повідомленнями у чаті між лікарем і користувачем.

2.1.4 Опис алгоритму роботи FileManagment сервісу

Данних сервіс був створений з метою розділення зобов’язань роботи з файлами системи з основним сервісом який працює з бізнес логікою.

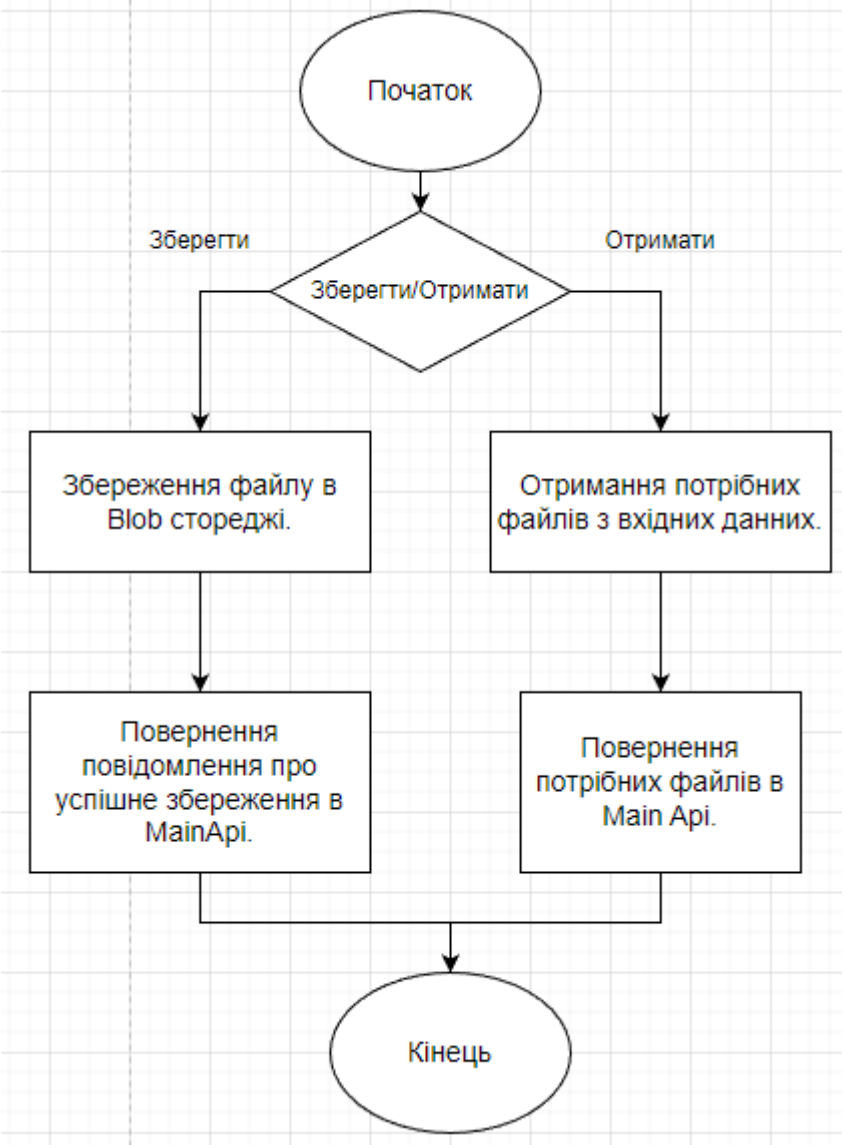

Рисунок 2.6 – Блок-схема алгоритму роботи FileManagement сервісу

Цей сервіс не був імплементований та запроваджений в систему в її поточній імплементації, проте він був архітектурно спланований і буде запроваджений налаштований в системі з часом. Тут імплементована вся логіка роботи з різними файлами, зокрема, фото. Оскільки лікар повинен мати змогу відправити користувачу рецепт з ліками, наприкінці консультації, а також користувачі повинні мати змогу завантажити фото видимих ділянок прояву хвороби, тому це доволі важлий сервіс. Як ми можемо бачити на рис. 2.6, алгоритм роботи цього сервісу має два варіанти в залежності від типу операції. Перший варіант це “збереження”, Main Api сервіс отримує файл в вигляді Base64 набору символів, потім використовуючи RabbitMQ сервіс для обміну повідомленнями між

сервісами. Він надсилає це повідомлення FileManagement сервісу, який в свою чергу зберігає отримане повідомлення з вигляді файлу “.jpeg” або “.png” в Blob storage сервісі в Azure.

2.2 Логічна архітектура системи

Для того щоб можна було зрозуміти як вся мікросервісна архітектура буде організована фізично і зрозуміти які сервіси хмарних послуг потрібно буде використати було створено дві схеми логічної архітектури системи. Одна логічна схема є планованою архітектурою повністю готової системи, інша схема архітектури, яка містить усі ключові властивості і є прототипом, але з меншою кількістю компонентів, й буде результатом виконання даної дипломної роботи.

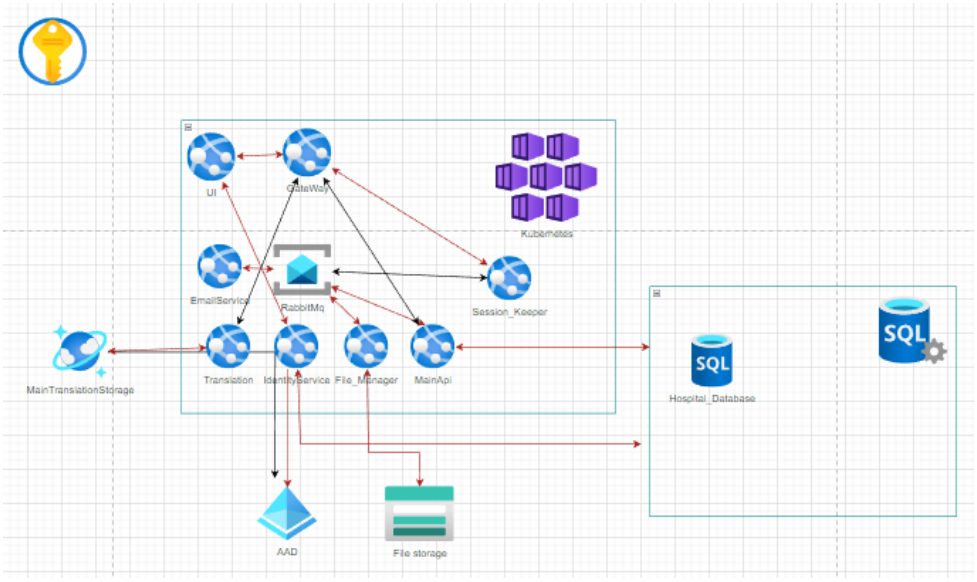

Рисунок 2.7 –Планована логічна архітектура системи

На рис. 2.7 ми можемо побачити заплановану архітектуру повністю готової системи.

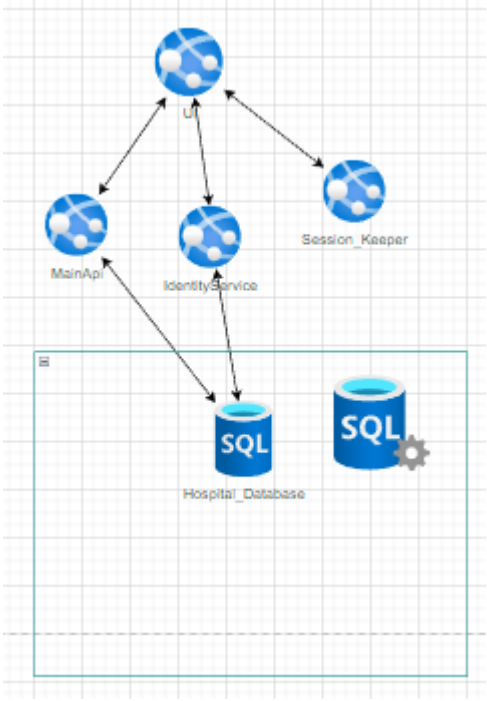

Рисунок 2.8 –Логічна архітектура системи (спрощений прототип планованої)

На рис. 2.8 зображено архітектуру системи, яка була створена як прототип планованої. Повна архітектура системи складається з 8 сервісів, які реалізовані у вигляді docker-контейнерів в Kubernetes. В UI контейнері розміщений Front End build написаний на Typescript з використання React framework, де знаходиться весь інтерфейс користувача. Решта сервісів написані на C# з використанням framework ASP .NET Core. Вся комунікація повинна відбуватись через Gateway сервіс який виконує функцію шлюза щоб перенапрявляти всі запити до відповідних сервісів. Такий чином для користувача буде видимість того що комунікація відбувається з єдиним доменним ім’ям, проте насправді шлюз перенапрявляє ці запити на різні сервіси. Решта компонентів створені як окремі логічні частини окремих сервісів.

Кожен сервіс має своє конкретне призначення: EmailService — розсилання електронних листів про початок конференції лікарю та пацієнту, Translation сервіс — для можливості давати користувачу вибрати мову на якій він хоче користуватись системою, для цього також було використане сховище нереляційних даних у базі даних CosmosDb, яка створена та підтримується Microsoft. У IdentityService загалом налаштована автентифікація з використанням токену доступу, а також сервісу Azure Active Directory для збереження історії та

списку користувачів з їхніми авторизаційними даними, для реалізації політик безпеки і генерування токена доступу була використана бібліотека IdentityService4. FileManagement сервіс використовується для маніпуляції з файлами. MailApi сервіс є основним в системі, тут була реалізована вся основна бізнес логіка, включаючи: створення, видалення, редагування та читання об'єктів системи, тобто конференцій та хвороб. SessionKeeper сервіс використовується для організації обміну повідомлення в реальному часі, це потрібно для обміну ключами щоб організувати веб відео конференцію, а також чат у якому буде відбуватись комунікація між лікарем та пацієнтом. Всі ключі системи зберігаються в спеціальному сховищі в хмарі, яке називається "KeyVault", який використовує асиметричний алгоритм шифрування, і є найбільш безпечним місцем де ключі можуть бути збережені.

На планованій архітектурі зображено логічну архітектуру системи в її звичайному поточному вигляді. Тут були реалізовані всі вимоги щодо створення та проведення консультації, мова налаштована лише англійська та немає email сервера, а також на поточному етапі не був реалізований FileManagement сервер для роботи з файлами.

2.3 Послідовність кроків доступних у системі

Загалом послідовність кроків у системі наступна. Після кожної дії яку робить користувач на UI, буде надіслано запит через gateway, на відповідний сервер, відповідь буде повернута також через цей самий gateway.

2.4 Фреймворки та бібліотеки, які використані при реалізації backend частини

2.4.1 ASP .NET Core

ASP .NET Core – це величезний фреймвок створений компанією Microsoft, який базується на платформі .NET Core. Він був створений як основний для розробки серверів з дуже складною логікою, оскільки, основна мова програмування на .NET це C#, яка є строго типізованою. Цей фреймворк є копією ASP .NET фреймворку з підтриманням крос-платформової розробки і є його сильно удосконаленим продовженням. ASP .NET Core має вбудовану систему модульного дистрибутива NuGet. Ця технологія має надзвичайно велику кількість різних бібліотек і стеків бібліотек. Також ASP .NET Core має сумісність за замовчуванням з Azure. Цей фреймворк такой має дуже велику перевагу в швидкості порівняно з інтерпретованими мовами (Python, PHP, JavaScript, тощо), також у ньому присутня зручна модель розширення функціоналу обробки запитів, який використовує технологію middlewares.

2.4.2 Entity Framework Core

Entity Framework Core – це фреймворк для об’єктно реляційного відображення, для зв’язування даних у рамках концепції об’єктно-орієнтованих мов програмування до даних у табличному вигляді в базі даних, з відкритим кодом. Цей фреймворк дозволяє розробникам працювати з даними використовуючи екземпляри класів не задумуючись про внутрішнє їх внутрішній вигляд в вигляді колонок і таблиць де ці дані зберігаються. Це створює можливості працювати на вищому рівні абстракції коли вони мають справу з даними і можуть створювати і маніпулювати з даними з меншою кількістю коду.

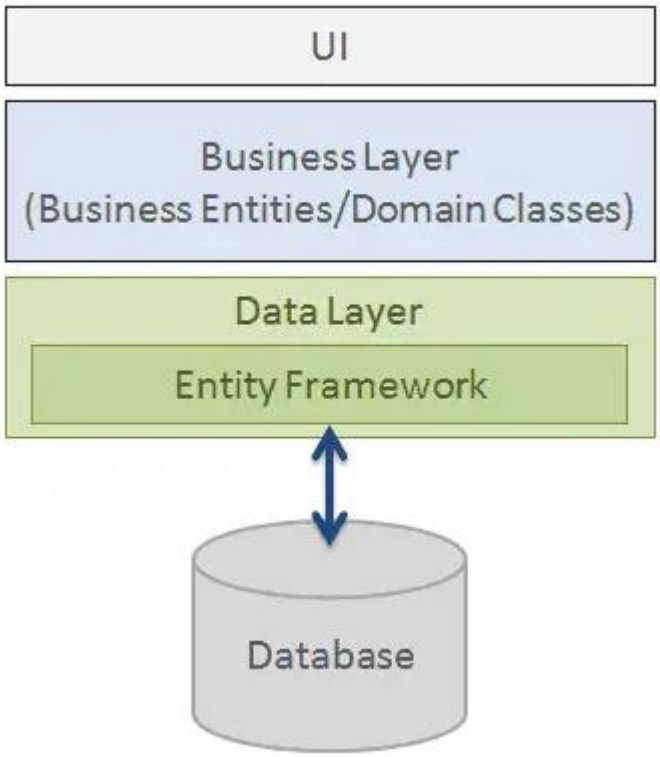

Рисунок 2.10 – Візуальне відображення роботи Entity Framework Core

На рис. 2.10 зображено принцип роботи Entity Framework. З даним фреймворком працюють на рівні роботи з даними, весь алгоритм роботи системи повинен взаємодіяти з базою даних використовуючи рівень роботи з даними за допомогою EntityFramework який буде вже безпосередньо сам робити потрібні запити до бази даних.

2.4.3 MediatR

MediatR – це бібліотека в C# яка використовується для зменшення рівня взаємодії між об’єктами. Це дозволяє створювати більш структуровану та непряму взаємодію між компонентами системи з використанням принципу CQRS.

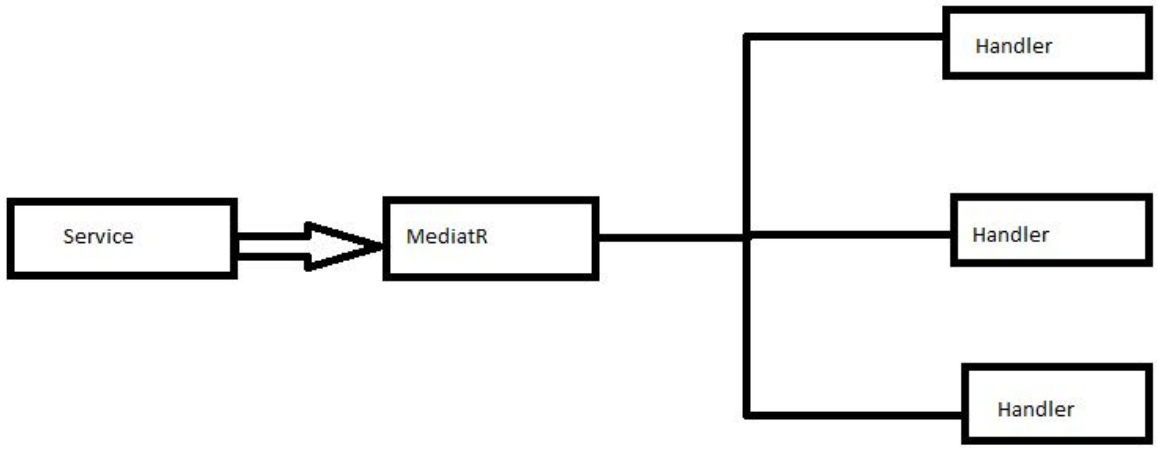

Рисунок 2.11 – Схема роботи бібліотеки MediatR

На рис. 2.11 можна спостерігати візуальне представлення роботи поточної бібліотеки. Вона дозволяє дотримуватись чітких правил при розробці, збільшити читабельність коду та уникнути майбутніх проблем з залежностями.

2.4.4 Бібліотека IdentityServer4

IdentityServer4 – це бібліотека з відкритим кодом, для написання серверів авторизації та автентифікації. Вона пропонує готові рішення для реалізації стандартів: OpenId, SSO та OAuth 2.0. Вона була створена так, щоб надати зручний функціонал для налаштування безпеки додатку на будь яких платформах, таких як web, mobile, Api або desktop. Автентифікація та авторизація потрібні для того щоб система розуміла який користувач зайшов в систему, чи користувач зареєстрований і чи пароль введений ним співпадає з паролем збереженим в базі даних, а також для фільтрації запитів користувача по
вибраним політикам, щоб лікарі не змогли отримати персональні дані пацієнтів, а пацієнти не змогли отримати доступ до персональних даних лікарів.

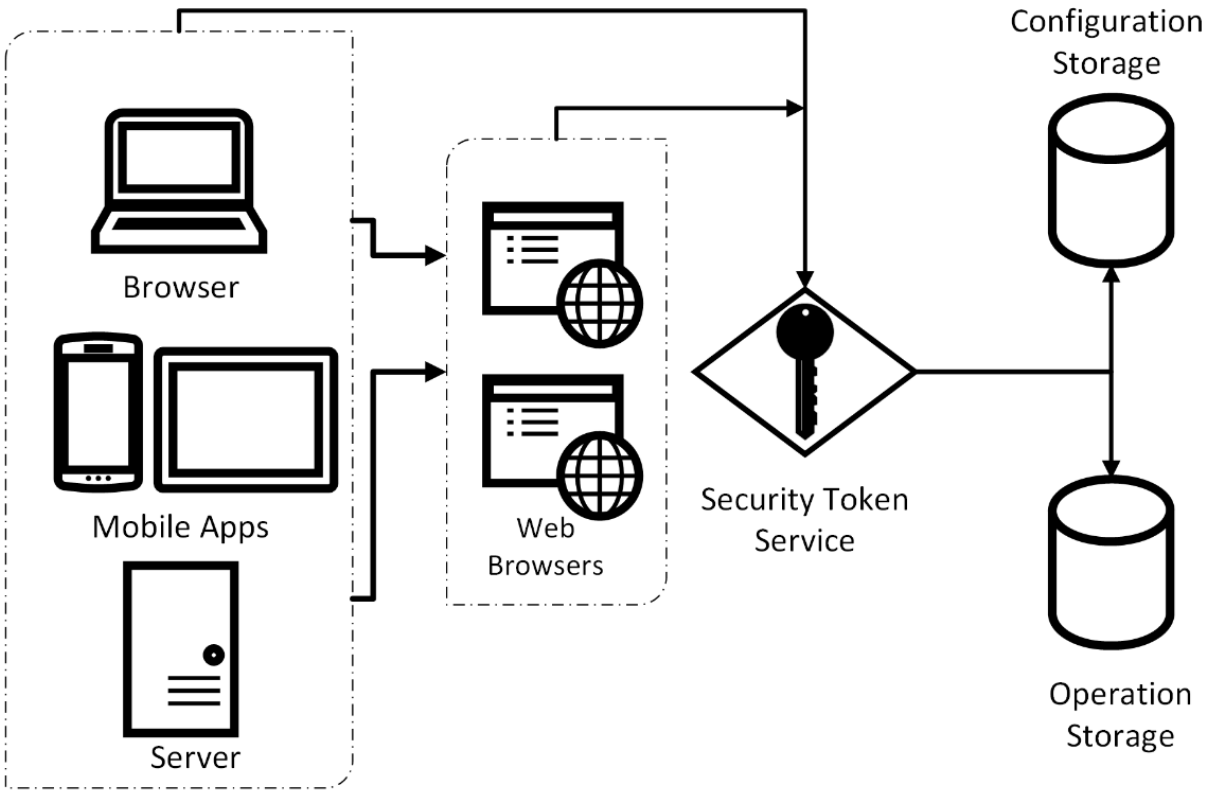

Рисунок 2.12 – Схема принципа роботи бібліотеки IdentityServer4

На рис. 2.12 видно що комунікація між користувачами і сервером відбувається тільки через спеціальний Security Token Service написаний з використанням бібліотеки IdentityServer4.

2.4.5 SignalR

SignalR – це бібліотека розроблена компанією Microsoft для можливості запровадження в ASP .NET Core фреймворку можливості роботи з web sockets. Web sockets в системі допомагають реалізувати комунікацію в реальному часі, коли не треба очікувати на відповідь після запиту, як це працює в REST.

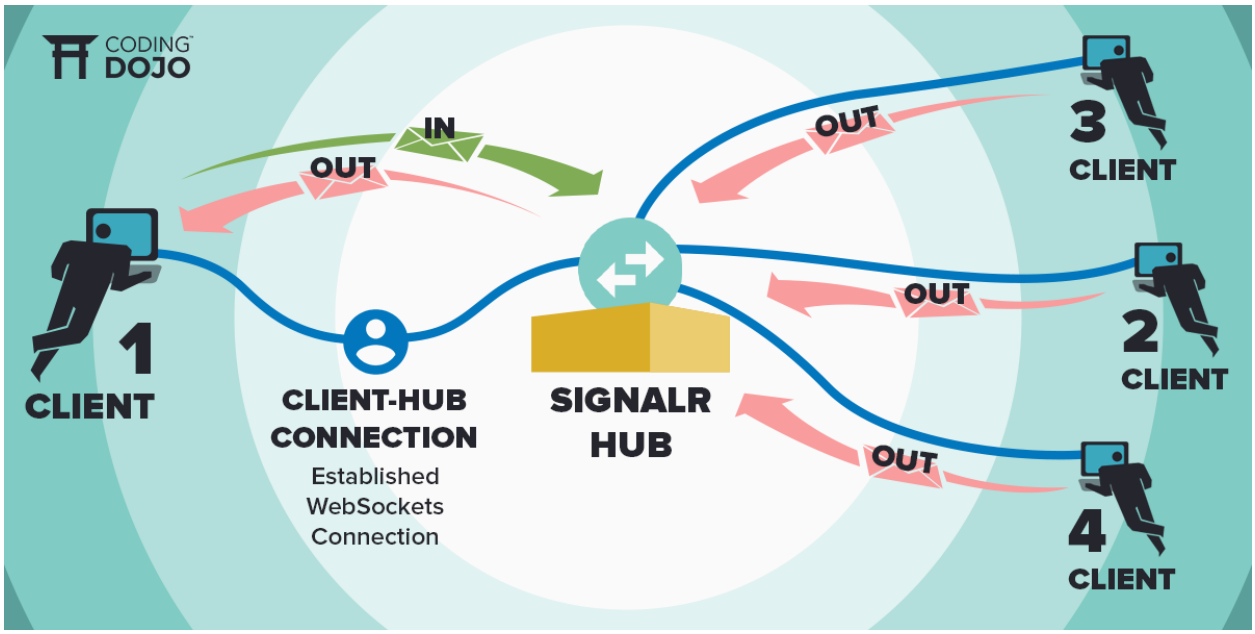

Рисунок 2.13 – Загальна хема роботи бібліотеки SignalR

Принцип роботи цієї бібліотеки базується на використанні хабів. Кожен клієнт повинен бути приєднаний до конкретної групи в хабі, або до всіх груп одночасно. На рис. 2.13 можна побачити що коли клієнт робить надсилання повідомлення, то воно розсилається до всіх користувачів приєднаних до тої самої групи в хабі, що й користувач, який надіслав повідомлення, таким чином і будується комунікація в реальному часі.

2.5 Фреймворки та бібліотеки використані на frontend частині

2.5.1 Typescript

Typescript – це безкоштовна мова програмування з відкритим кодом, створена компанією Microsoft, статично типізоване розширення мови JavaScript. В браузері ця мова перекомпільовується в кінцевому рахунку у JavaScript для того щоб виконатись. Але це створює певні обмеження для програмістів в написанні програмного коду, щоб зробити його чистішим, що дозволить створювати значно більші додатки і зменшити кількість потенційних помилок в системі. Спочатку ця мова програмування була створена для внутрішнього використанні в 2012

компанією Microsoft за два роки, але зрозумівши великі перспективи цієї мови, вони вирішили опублікувати свою розробку.

2.5.2 Фреймворк React

React – це безкоштовний фреймворк з відкритим кодом для створення користувацького інтерфейсу на мові JavaScript. Принцип його роботи базується на тому що він використовує принцип Single Page Application, який дозволяє не перевантажувати сторінку якщо ми переходимо на іншу сторінку, сам елемент <body> не змінюється, ми просто створюємо компоненти (частинки сторінки), яка встановлюються в потрібні місця в конкретні моменти часу. Сторінки всі створюються через компоненти, які ми ж самі і створюємо, що дозволяє використовувати одинакові користувацькі елементи на різних сторінках. Фреймвок був створений компанією Meta у 2013 році і наразі активно отримує нові версії і оновлення.

2.5.3 Бібліотека Axios

Axios – це бібліотека для створення HTTP-запитів на мові JavaScript за основі асинхронних обіцянок(promises). Всі запити через цю бібліотеку відбувається асинхронно, тобто інтерфейс користувача не зупиняється коли ми робимо цей запит, а продовжує функціонувати і ми самі можемо вказати яка дія повинна відбутися коли запит буде виконано. Ця бібліотека є доволі невелика та інтуїтивно зрозуміла у використанні, тому у фреймворку React вона є ключовою для створення інтеграції з сервером через Rest Api.

2.6 Архітектура бази даних системи

База даних була створена з використанням функції міграції в EntityFramework Core з використанням принципу Code First, згідно якого спочатку повинна бути створена модель в коді, після чого вже повинна бути створена таблиця яка буде відповідати створеній моделі. Архітектура в цілому налічує 12

таблиць і багато зв’язків між ними, включаючи: один до одного, один до багатьох і багато до багатьох.

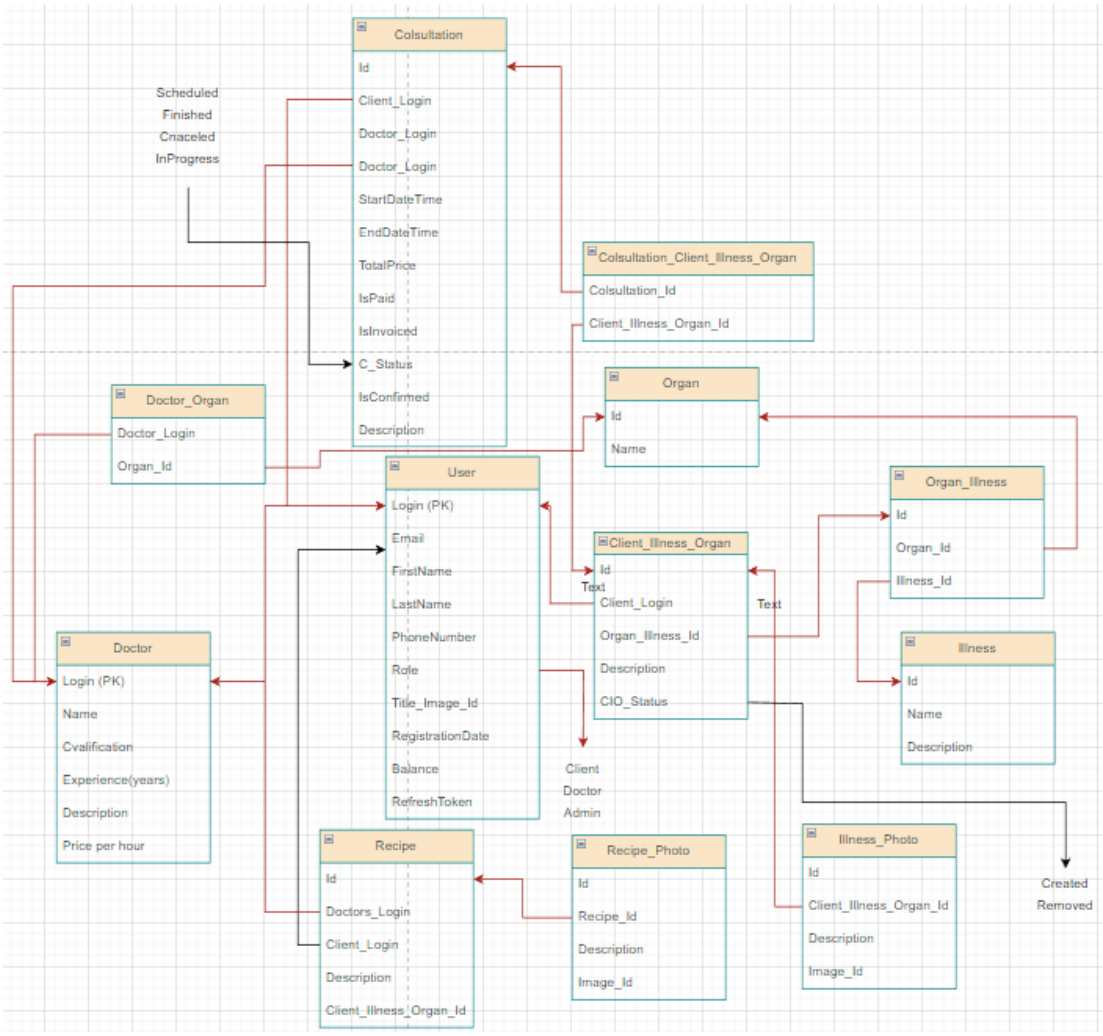

Рисунок 2.14 –Архітектура реляційної бази даних системи

Основним компонентом системи є таблиця ”Consultation”. Це сутність яка є ключовою в системі, оскільки реєструє час проведення консультації і користувачів які повинні бути там присутні. Вона може мати три різні статуси: запланована, закінчена, скасована та в прогресі. Ці статуси змінюються в залежності від того що наразі відбувається з консультацією, і до цього прикріплено багато бізнес логіки. Другим важливим компонентом системи є “Client_Illness_Organ”, який представляє собою сутність яку створює користувач вибравши потрібний орган і хворобу і зробивши додатковий опис хвороби. Схему таблиці можна спостерігати на рис. 2.14. Зв’язок між “Consultation” та “Client_Illness_Organ” був багато до

багатьох, для того щоб користувач мав змогу реєструвати декілька хвороб, і прикріплювати їх до консультації більше одного разу. Таблиці “Illnesses” “Organs” та їх таблиця для їх зв’язку багато до багатьох “Organ_Illness”, заповнені списком доступних для лікування хвороб і органів з самого початку, і можуть збільшуватись з допомогою адміністратора. Кожен користувач системи має таблицю “Users” у якій збережені всі його персональні дані, проте лікарі, окрім цієї таблиці мають ще таблицю “Doctors”, яка має зв’язок один до одного з таблицею “Users”. У таблиці “Doctors” також є зв’язок багато до багатьох з таблицею “Doctor_Organ” для того щоб кожен лікар міг самостійно обрати органи у яких він має досвід лікування і може проводити консультації з пацієнтами що мають скаргу на хворобу у цих органах або частинах тіла. У системі присутні ролі, від яких в кінцевому рахунку буде залежати меню та набір можливостей на інтерфейс користувача. Всього є три ролі, це адміністратор, лікар та клієнт.

2.7 Мова запитів SQL та Sql Server

SQL – це декларативна структурована мова запитів для роботи з базами даних і даними всередині цих баз. Реляційні бази даних зберігають інформацію у вигляді таблиць з стовпцями і рядками. SQL використовується для читання, запису, оновлення та отримання даних з таких баз даних. Зручність цієї мови полягає також в тому що вона зрозуміла і може бути виконана навіть на інших мовах програмування.

SQL Server – це система управління базами даних, створена компанією Microsoft. Як сервер, він виконує головну функцію по наданню даних по запиту інших застосунків та їх збереженню. Наразі ця СКБД є одною з найбільш відомих та потрібних на ринку, вона має дуже хорошу інтеграцію з іншими застосунками компанії Microsoft, такими як Azure C#. Тому SQL Server використовується практично завжди, якщо сервер був написаний на мові програмування створеній компанією Microsoft. Вперше ця СКБД була створена в 2005 році, проте вона

суттєво відрізнялась від сучасного SQL Server, оскільки, для збереження даних частково використовувала XML формат.

РОЗДІЛ 3 ПРАКТИЧНА ЧАСТИНА

3.1 Огляд інтерфейсу користувача системи

У даному розділі розглянемо реалізацію системи та взаємодію з нею. Зокрема, здійснимо огляд створеного інтерфейсу користувача, який є фронтенд-частиною програмної системи.

3.1.1 Огляд інтерфейсу авторизації

Коли ми переходимо за посиланням системи, використовуючи веб браузер ми бачимо сторінку авторизації, яку показано на рис 3.1.

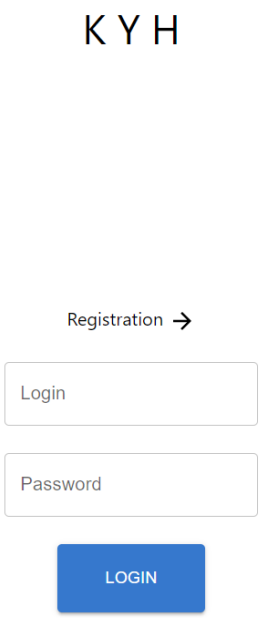

Рисунок 3.1 – Авторизаційна сторінка.

Якщо ми ввели неправильні дані, то ми отримаємо про це повідомлення.

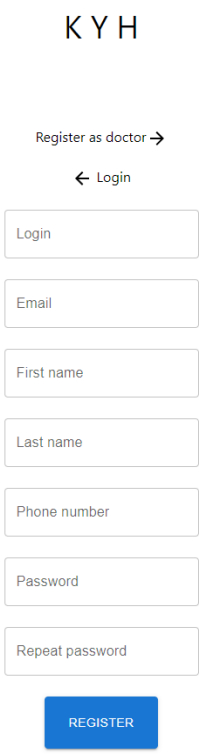

Рисунок 3.2 – Реєстраційна сторінка клієнта

На рис. 3.2 зображено сторінку яка буде показана клієнту в випадку коли він намагається створити обліковий запис. Користувач повинен заповнити всі необхідні поля і його обліковий запис буде зареєстрований. Для лікаря доступні додаткові поля для заповнення для вказання ціни консультацій і рівня кваліфікації.

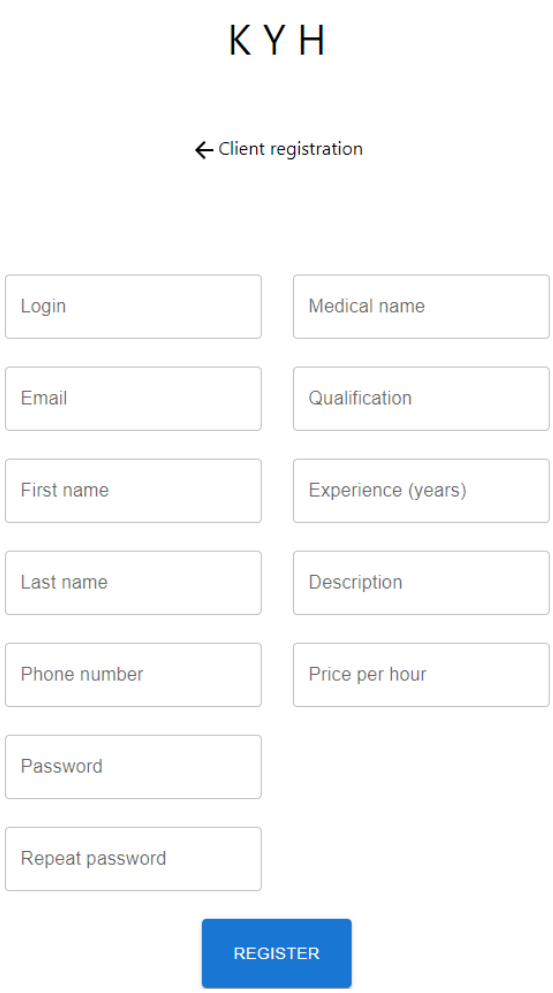

Рисунок 3.3 – Реєстраційна сторінка лікаря

Якщо ми натиснемо на “Register as doctor” ми будемо перенаправлені на сторінку, яка показана на рис. 3.3.

3.1.2 Огляд інтерфейсу кабінету лікаря

Зразу після входу в систему ми потрапимо на стартову сторінку де користувачі буде представлене стартове повідомлення.

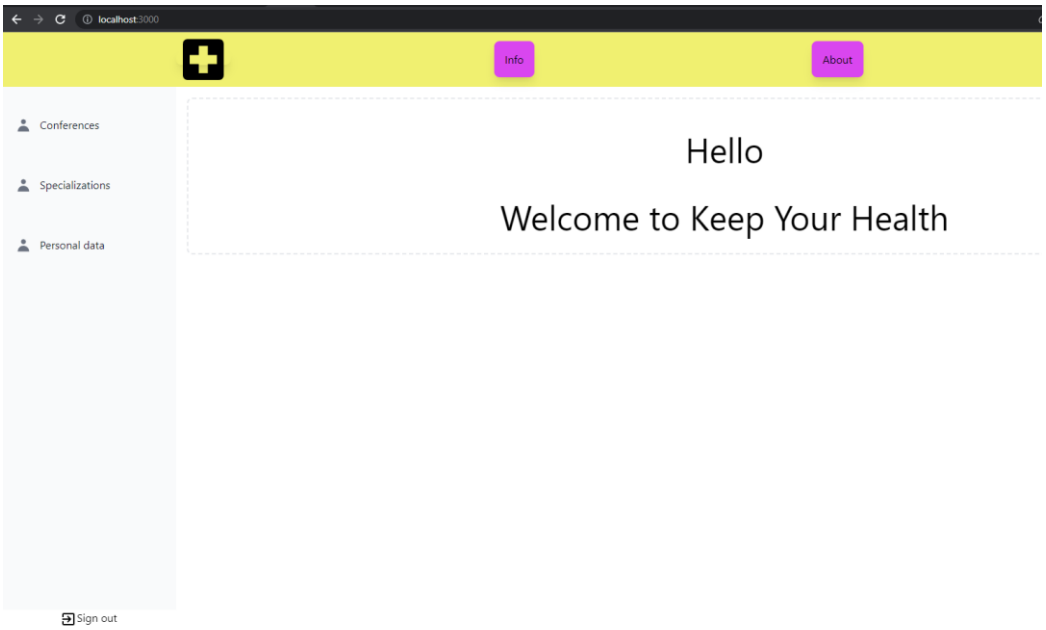

Рисунок 3.4 – Стартова сторінка особистого кабінету лікаря

Для того щоб почати користуватись системою, користувач повинен обрати один з трьох запропонованих пунктів меню як ми можемо спостерігати на рис. 3.4:

– Конференції.

– Спеціалізації.

– Персональні дані.

Перейшовши на пункт “Конференції” ми зможемо отримати список всіх запланованих конференції, конференцій які зараз проводяться, конференцій які були проведені, і всі вони обов’язково повинні бути оплачені, не оплачені конференції не будуть відображатись в списку.

Рисунок 3.5 – Сторінка списку всіх доступних конференцій.

Як ми бачимо на рис. 3.5, у лікаря є консультації що завершились та консультації що заплановані. Якщо користувач натисне на на будь яку консультації його буде перенаправлено на окрему сторінку з виведеними деталями про цю консультацію. Якщо натиснути на будь яку конференцію нас буде перенапрямлено на деталі, як поканазо на рис. 3.6.

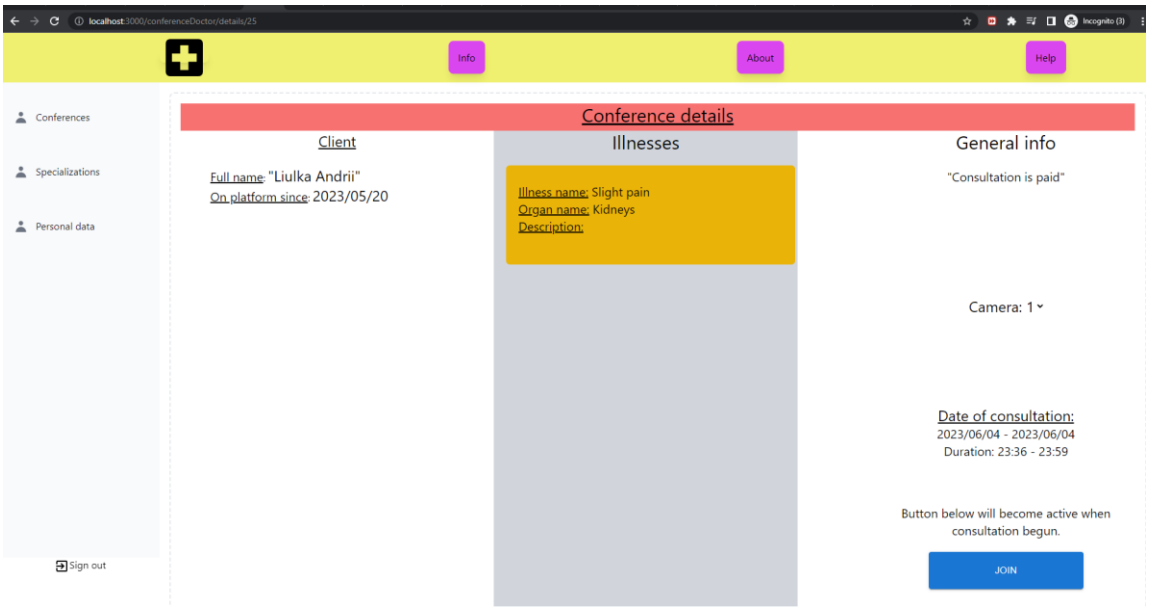

Рисунок 3.6 – Сторінка деталей про конференцію

На сторінці деталей про конференцію лікар може побачити коротку інформацію про пацієнта з яким буде проведена консультація, також інформацію про хвороби на частини тіла які є суб'єктом цієї консультації і дату та час проведення цієї консультації. Кнопка "Join" стає активноб тільки в тому випадку коли зараз час проведення конференції.

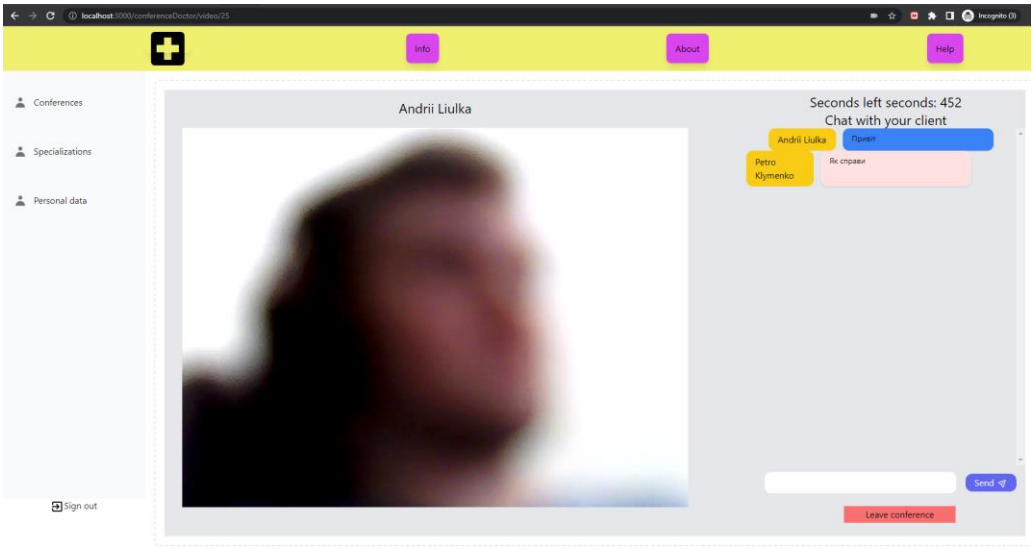

Рисунок 3.7 – Сторінка початої конференції з ввімкнутим відео пацієнта

Відео вмикається тільки в тому випадку коли клієнт приєднався, якщо пацієнт ще під'єднується, то замість відео буде вікно загрузки. Як видно з рис. 3.7, що вся комунікація відбувається через чат. Також як ми можемо спостерігати на сторінці присутній таймер який показує відлік часу в секундах скільки залишилось до кінця конференції, відразу після того як час виходить, конференція автоматично закривається і перенаправляє нас на стартову сторінку.

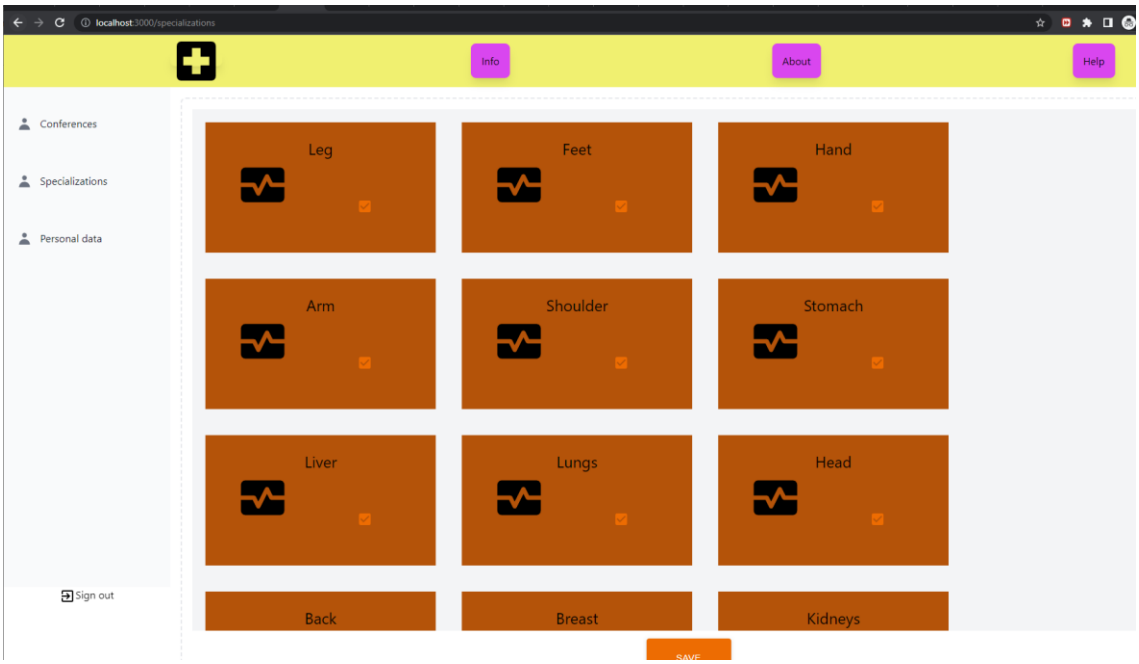

Рисунок 3.8 – Сторінка спеціалізацій

Перейшовши на сторінку “Спеціалізації” лікар отримає змогу обрати органи та частини тіла по яких від зможе надавати кваліфіковану консультацію. Як видно на рис. 3.8 всі органи та частини тіла відображаться в вигляді прапорців які потрібно обрати або видалити з обраних, і тоді всі пацієнти при створені консультацій на цю частину тіла будуть бачити ваш профіль серед інших. Для того щоб зберегти зміни які ми внесли до спеціалізацій, нам потрібно натиснути кнопку “Save” і все спеціалізації будуть автоматично збережені.

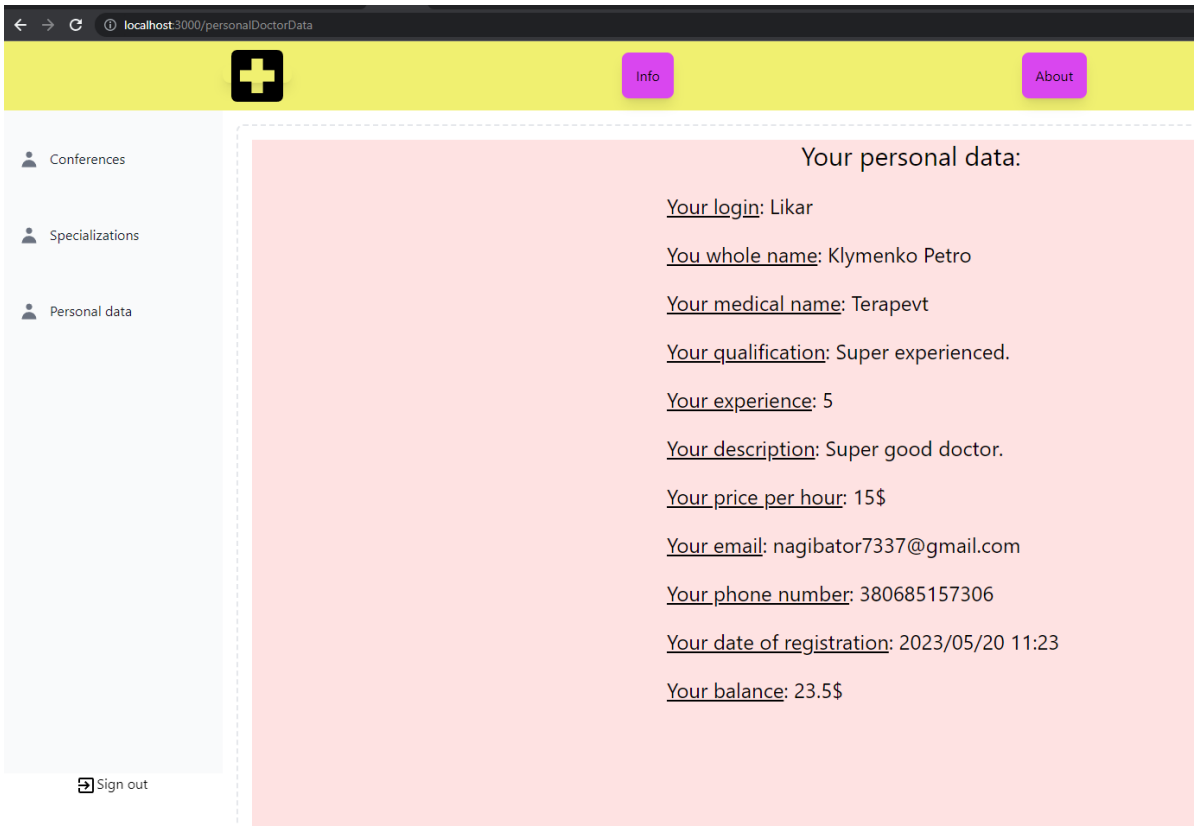

Рисунок 3.9 – Сторінка персональних даних персонального кабінету лікаря

Кожен лікар може подивитись свої персональні данні, для прикладу, яка ціна за годину консультації встановлена, скільки всього коштів він заробив за час роботи в КУН системі. Приклад персональних даних користувача можемо бачити на рис. 3.9. Баланс лікаря збільшується після кожної консультації.

3.1.3 Огляд інтерфейсу кабінету пацієнта

Відразу після введення правильних авторизаційних даних від кабінету клієнта нас буде перенаправлено на стартову сторінку. Тут користувачу запропоновані деякі інші пункти меню, а саме:

– конференції;

– хвороби;

– лікарі;

– лікар-штучний інтелект;

– особисті дані.

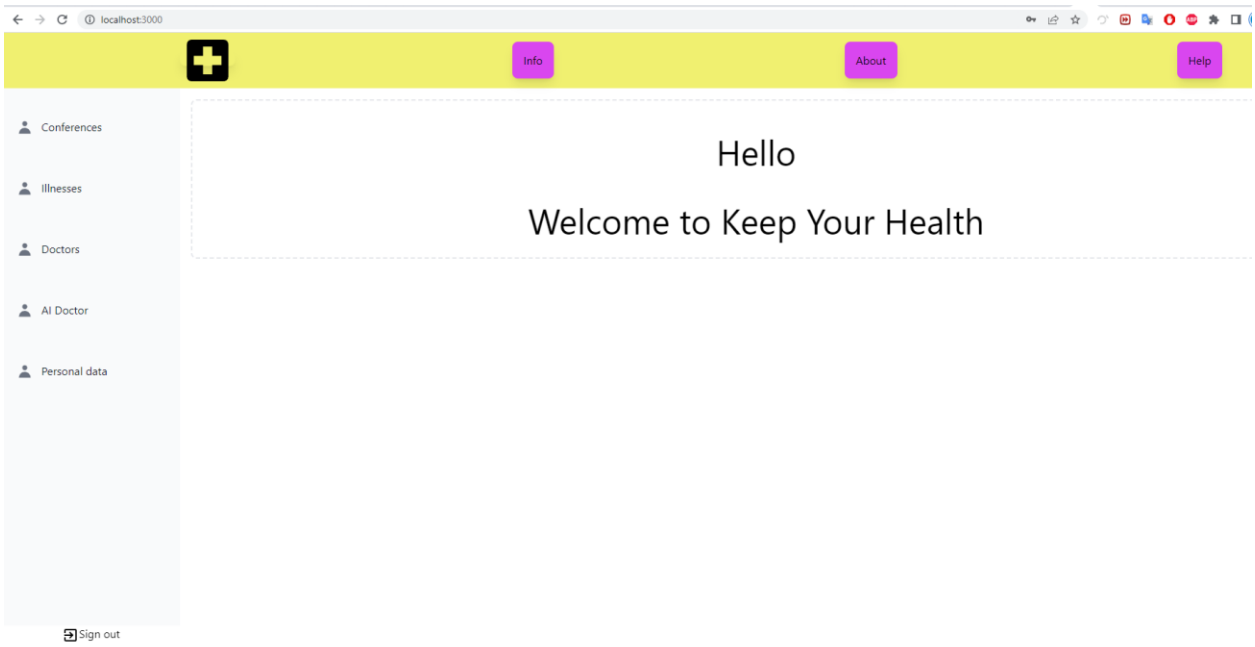

Рисунок 3.10 – Стартова сторінка клієнта.

Приклад стартової сторінки можемо побачити на рис. 3.10.

У клієнта доступні усі ті самі сторінки в верхній панелі. Відразу після входу нас буде вітати стартова сторінка. Як видно на рис. 3.11 у нас введено в полі введення число 12, це означає, що коли ми натиснемо клавішу “Top up” ці гроші автоматично будуть додані на рахунок, де наразі є 5.25$.

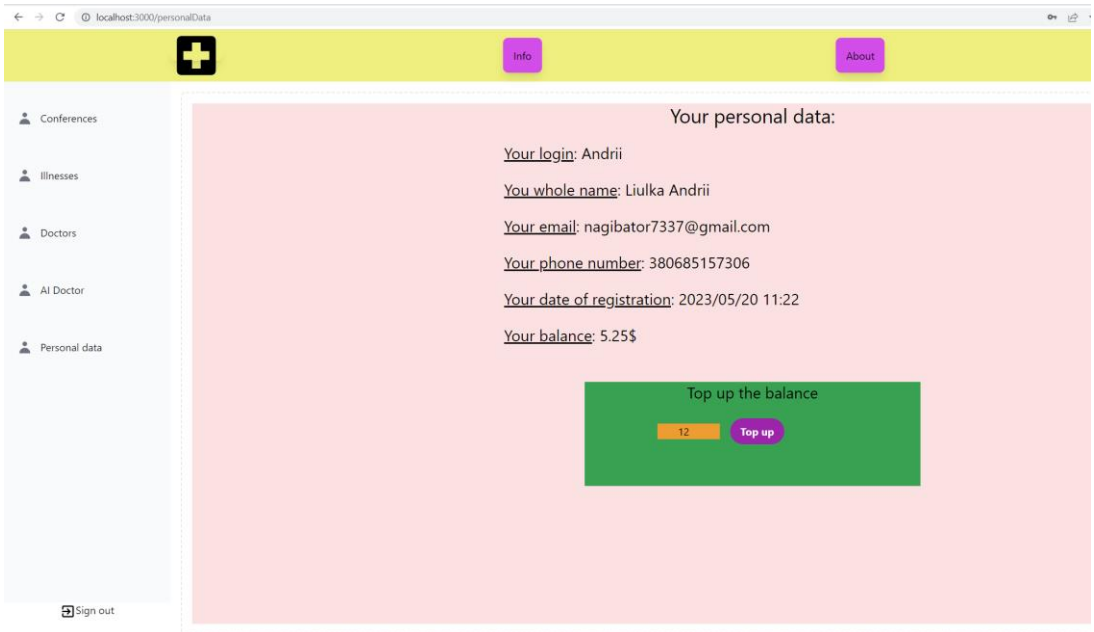

Рисунок 3.11 – Сторінка з персональними даними

На останньому пункті ми можемо бачити пункт для перегляду наших персональних даних. Тут також є пункт для того щоб ми могли поповнити наш рахунок, з якого будуть у подальшому зніматись кошти як оплата консультації. Цей спосіб оплати був створений для тестового середовища додатку, оскільки на реальній версії, тут реалізований функціонал оплати консультації за справжні гроші.

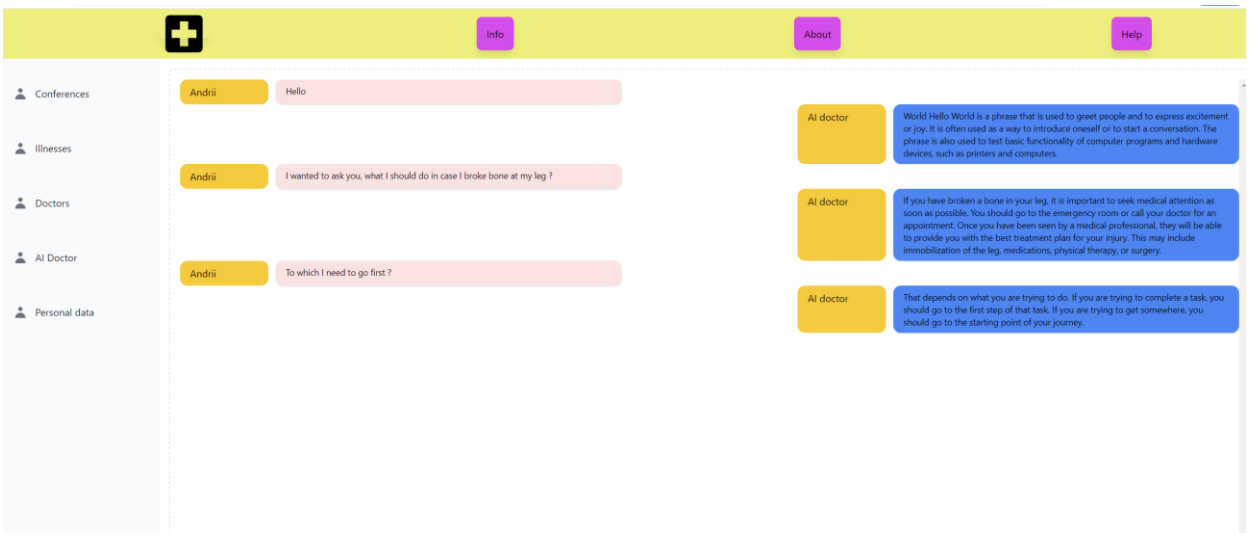

Рисунок 3.12 – Сторінка для консультації з штучним інтелектом

Люди які не мають змоги оплатити консультацію можуть безплатно поспілкуватись з моделлю штучного інтелекту створеного на основі відкритого API ChatGPT. У більшості випадків модель всеодно рекомендує звернутись до справжнього клаліфікованого лікаря для того щоб отримати більш правильні рекомендації. Модель може тільки надати базові поради щодо лікування цієї проблеми або допомогти з обиранням лікаря до якого варто звернутись. Як ми бачимо на рис. 3.12, штучний інтелект дає загальні рекомендації, як виявити що нога є зломана, але все таки найперше що він пропонує, це звернутись до медпрацівників.

Така можливість спілкуватись з штучним інтелектом була створена як додаткова можливість і не є головним функціоналом.

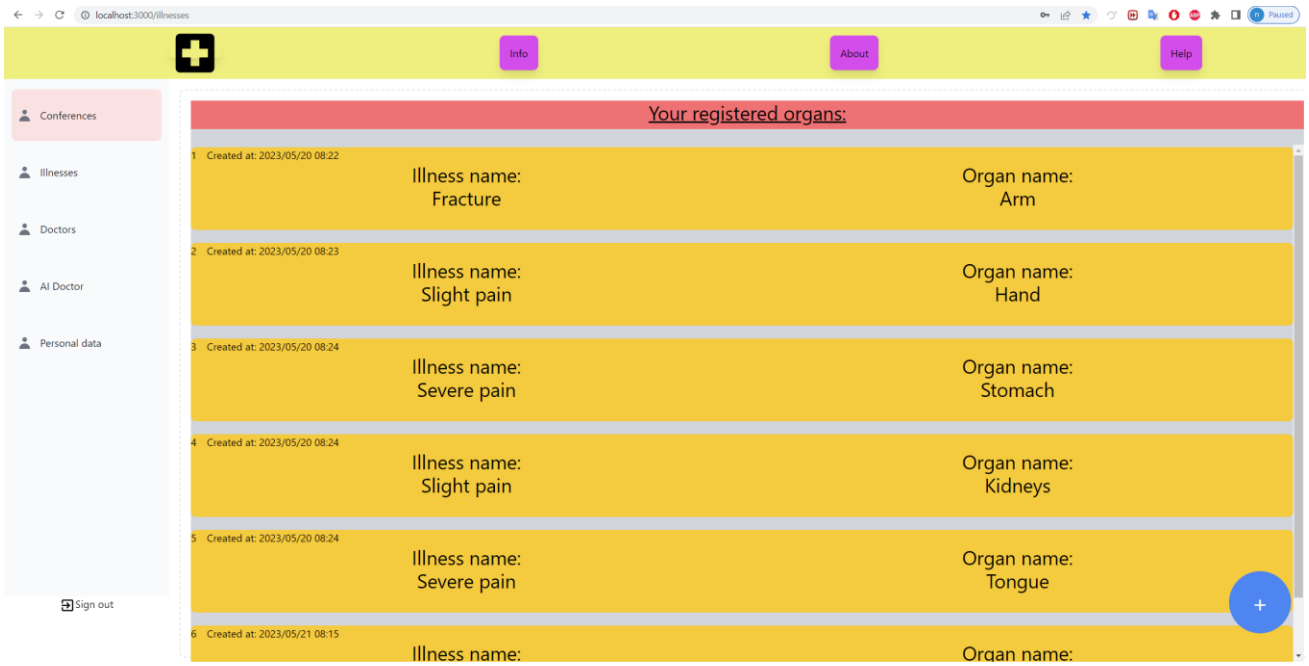

Рисунок 3.13 - Вміст сторінки “illnesses” попередньо зареєстрованих хвороб

У системі є сутність “Illness”, тобто це хвороба яка реєструється з допомогою пацієнта, і містить в собі орган або частину тіла та назву хвороби що турбує пацієнта. Якщо ми перейдемо на пункт у меню “illnesses” ми зможемо побачити весь список зареєстрованих попередньо як показано на рис. 3.13.

Для того щоб зареєструвати цю сутність (illness) нам потрібно натиснути на синій круглий круг в нижньому правому куті сторінки “illnesses”, ми можемо побачити цю кнопку на рис. 3.13.

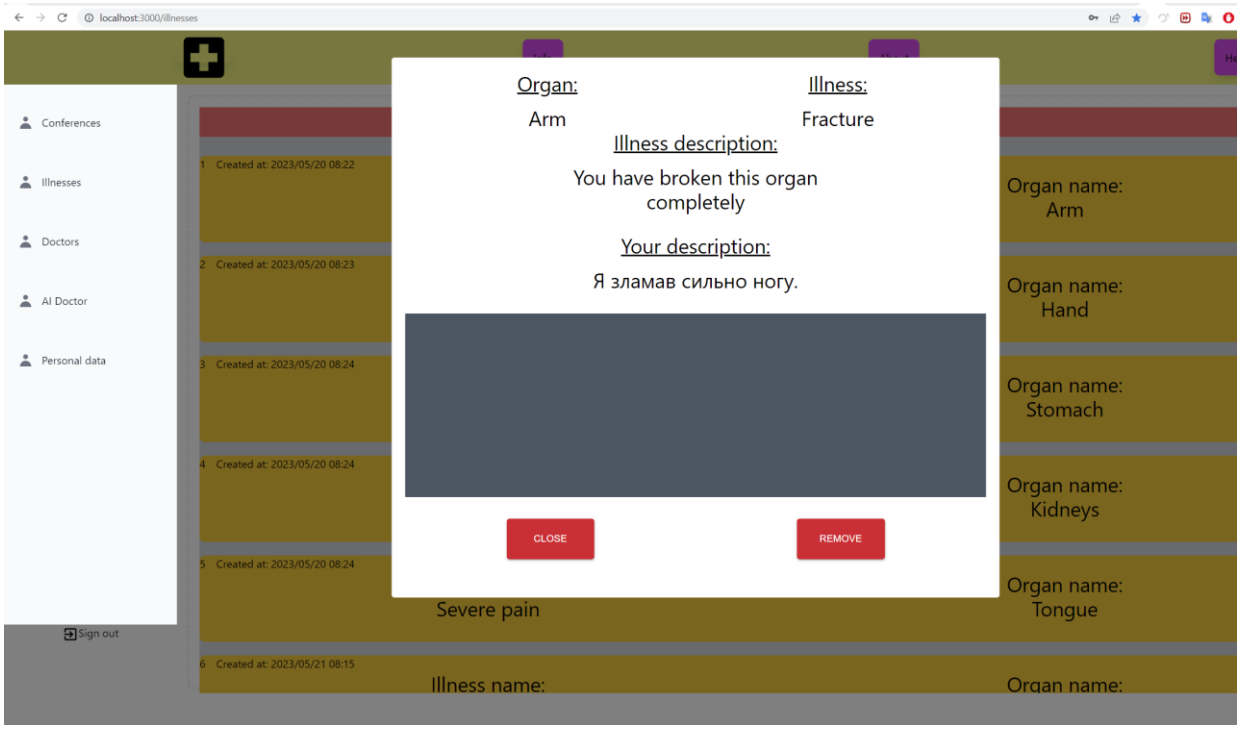

Рисунок 3.14 - Вміст модального вікна детального перегляду зареєстрованої хвороби на сторінці “illnesses”

Кожна хвороба містить більш детальну інформацію, цю інформацію можна переглянути натиснувши на будь яку з доступних хвороб. Приклад огляду детальної інформації про конкретну хворобу можна побачити на рис. 3.14. У цьому вікні для нас є доступні дві дії: “close” та “remove”. Перша кнопка просто поверне нас на загальний список зареєстрованих хвороб, друга видалить цю хворобу зі списку.

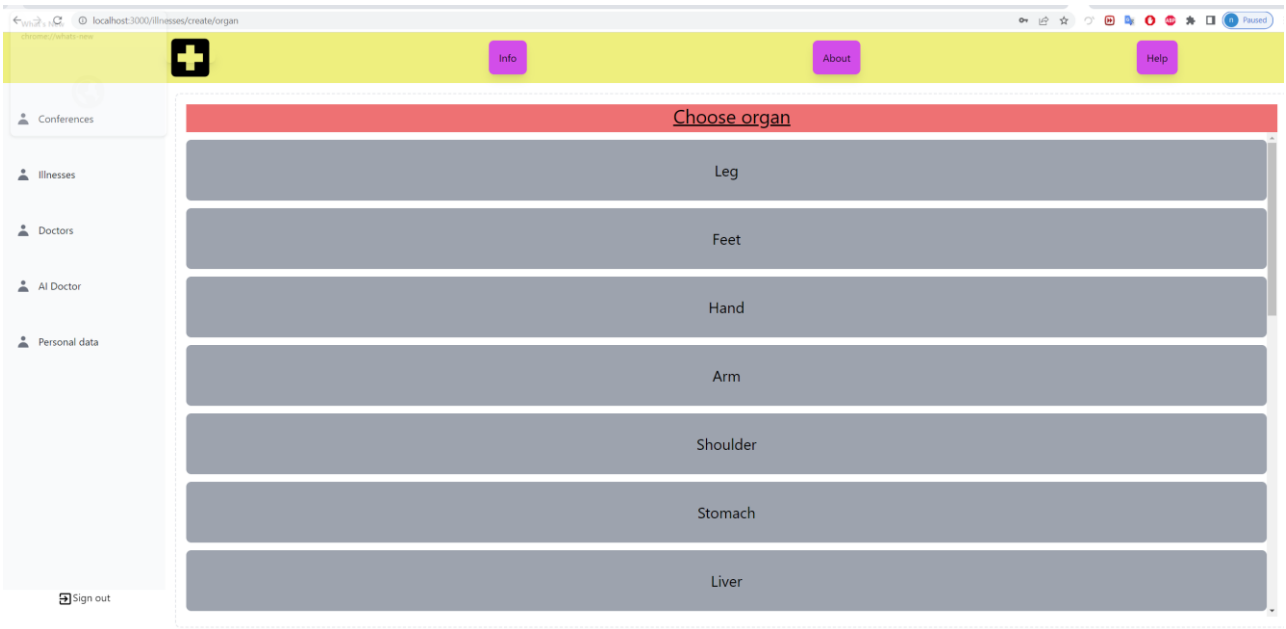

Рисунок 3.15 – Список всіх доступних частин тіла та органів для створення сутності “illness”

Список всіх доступних для вибору хвороб можуть бути добавлені в системі тільки адміністратором системи.

Хвороби, до конкретної частини тіла підбираються відразу після натискання на обрану хворобу. Приклад списку хвороб можна побачити на рис. 3.15. Після натискання на орган, ми отримаємо такий самий список хвороб які виникають в цьому органі чи частині тіла.

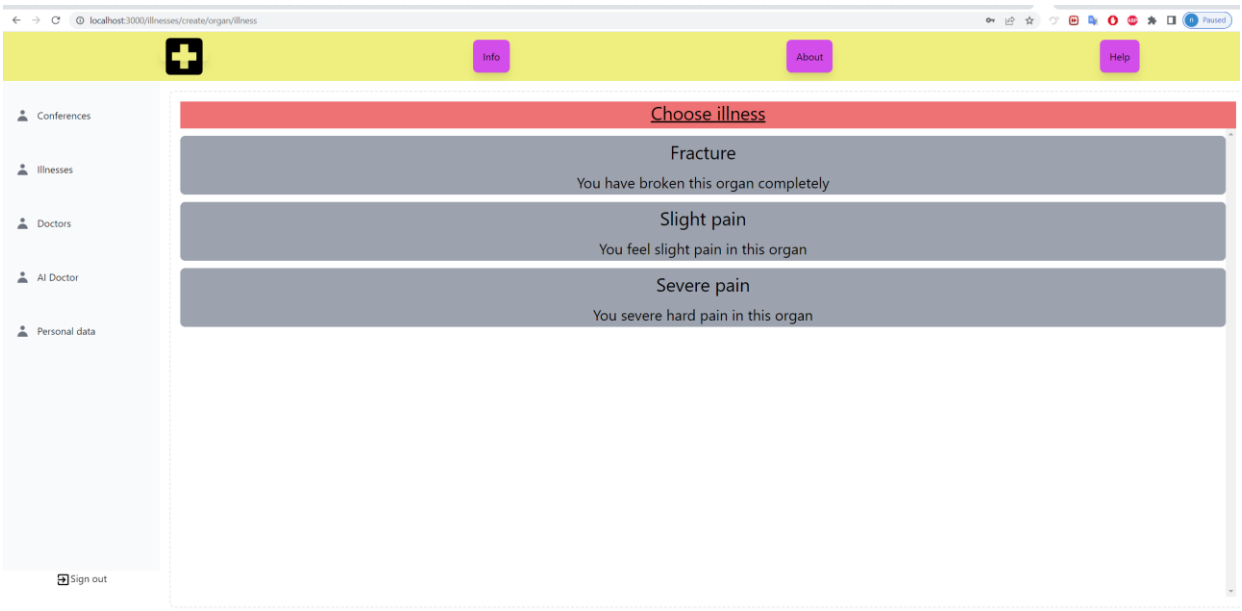

Рисунок 3.16 – Список хвороб для частини тіла “стопа”

Після обирання потрібного органу, ми отримаємо список доступних хвороб, наприклад на рис. 3.16 зображено список хвороб, які можуть статися з органом “стопа”.

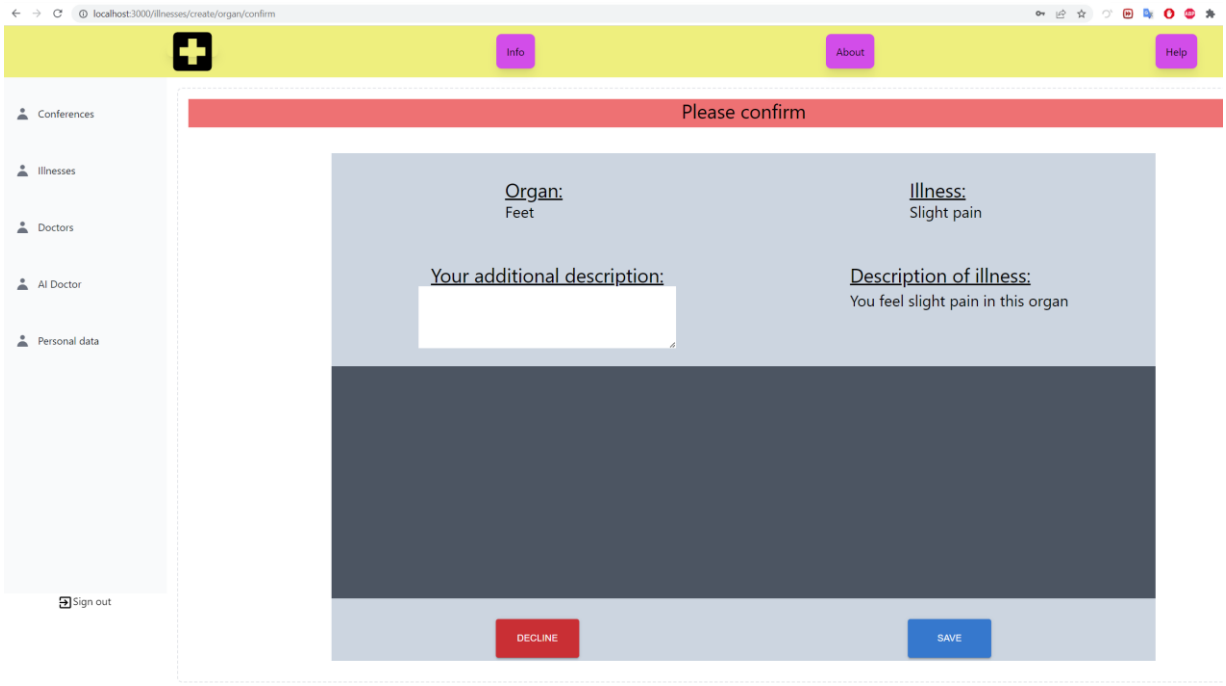

Рисунок 3.17 – Сторінка для підтвердження реєстрації хвороби

Сторінка з підтвердженням реєстрації хвороби зображена на рис. 3.17.

Останній крок який повинен зробити користувач щоб зареєструвати хворобу, це заповнити останню сторінку підтвердження, та натиснути кнопку “Save”, після чого хвороба буде успішно зареєстрована. Для користувача також доступна для натискання інша кнопка “Decline”, після натискання якої всі обрані попередньо сутності скасуються і ми повернемося до списку “Illnesses”.

Не менш важливою одиницею системи є сутність “Conference”. Конференція є основною одиницею системи, оскільки саме по конференції утворюється відео консультація з лікарем, а також тут збережена вся необхідна інформація про стан консультації з лікарем.

Рисунок 3.18 – Сторінка зі списком конференцій

Кожна конференція може мати один з 4-ох статусів:

– scheduled;

– finished;

– canceled;

– inProgress.

Кожен статус має своє спеціальне призначення як ми можемо бачити на рис. 3.18. Scheduled – статус присвоюється будь якій консультації відразу після її створення. Коли консультація починається, вона автоматично отримує статус

InProgress. Якщо при буль яких обставинах клієнт вирішив скасувати консультацію, то їй буде присвоєно статус Canceled. Відповідно коли консультація завершилась, їй присвоюється статус Finished. Ці статуси дуже важливі, оскільки відносно них йде відображення лікарю тільки певних консультацій. Також кожна консультація має статус “IsPaid” для відображення, чи пацієнт оплатив консультацію чи ні. Відповідно, лікарю тільки відображаються консультації, які знаходяться в статусі “InProgress” або “Scheduled”, а також ці консультації обов’язково мають бути оплачені. Оскільки, лікар отримує гроші за цю консультацію після її проведення, вона точно має бути оплачена.

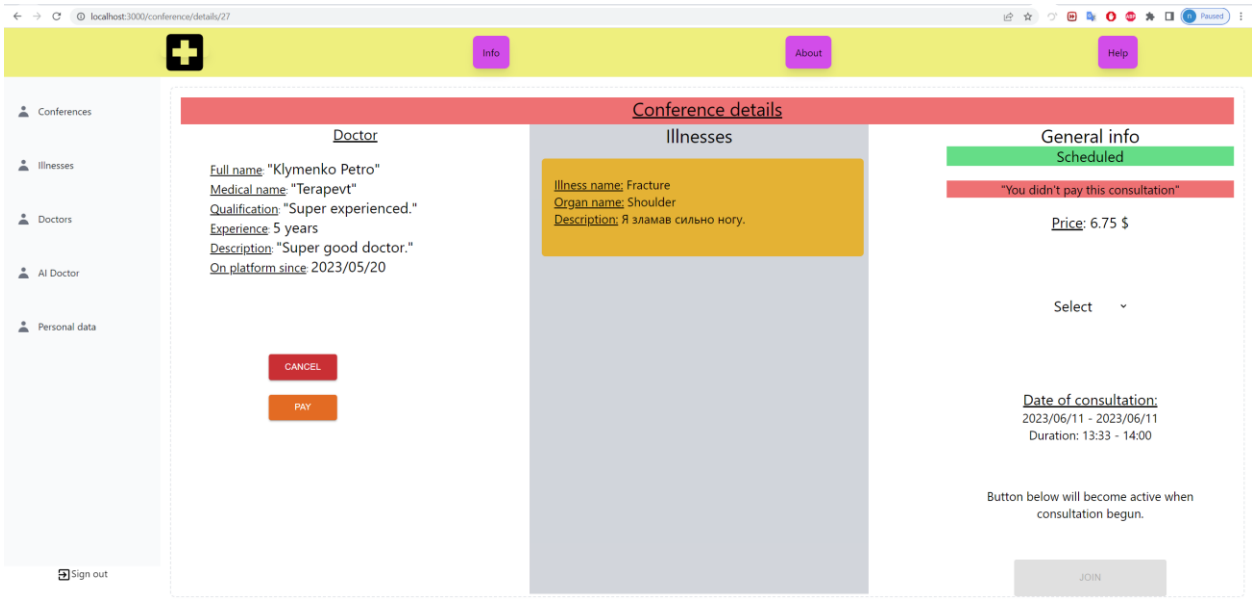

Рисунок 3.19 – Сторінка детальної інформації про конференцію

Для того щоб оплатити конференцію, потрібно зайти на сторінку “conferences”, та натиснути на потрібну консультацію, після чого нас буде перенапрямлено на сторінку деталей конференції, приклад сторінки деталей конференції можна побачити на рис. 3.19. Далі потрібно натиснути кнопку “Pay”, якщо на вашому рахунку, який можна побачити на сторінці персональних даних, недостатньо коштів, ви отримаєте відповідне повідомлення про помилку виконання оплати конференції. Якщо коштів достатньо то операція буде успішно виконання, і консультація буде оплачена.

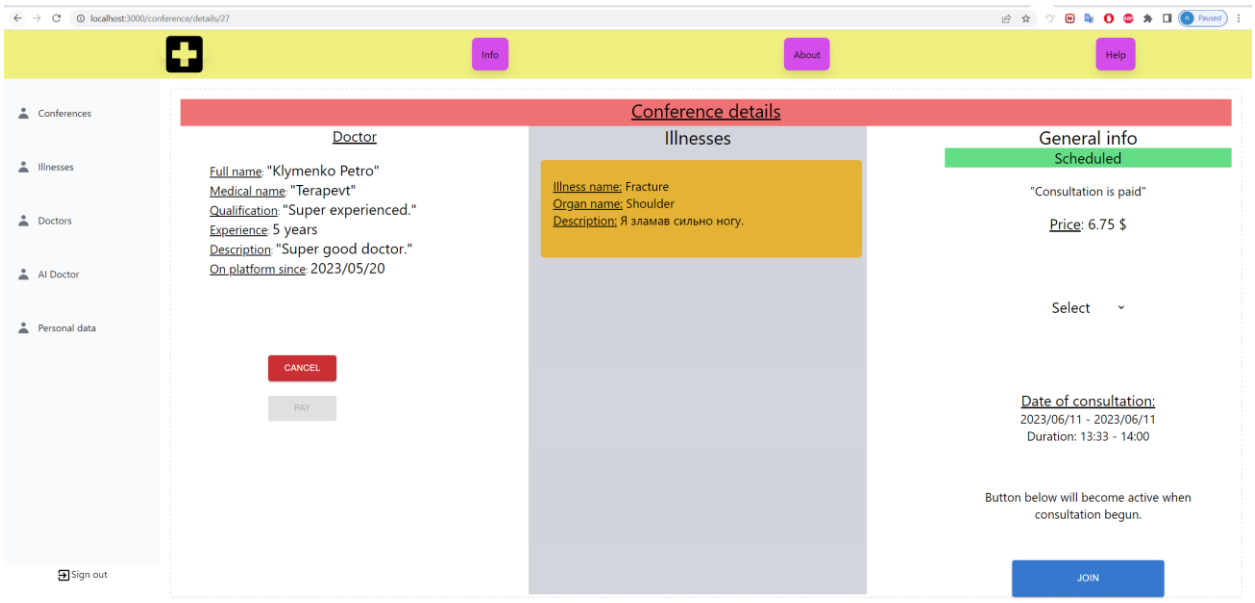

Рисунок 3.20 – Сторінка детальної інформації про оплачену конференцію

Для того щоб приєднатись на конференцію, вона повинна бути оплачена та час поточний повинен співпадати з діапазоном часу запланованої конференції. Як ми бачимо на рис. 3.20, час наразі співпадає і консультація оплачена, тому кнопка “Join” активна. Для того щоб приєднатись до консультації, нам потрібно натиснути на цю кнопку.

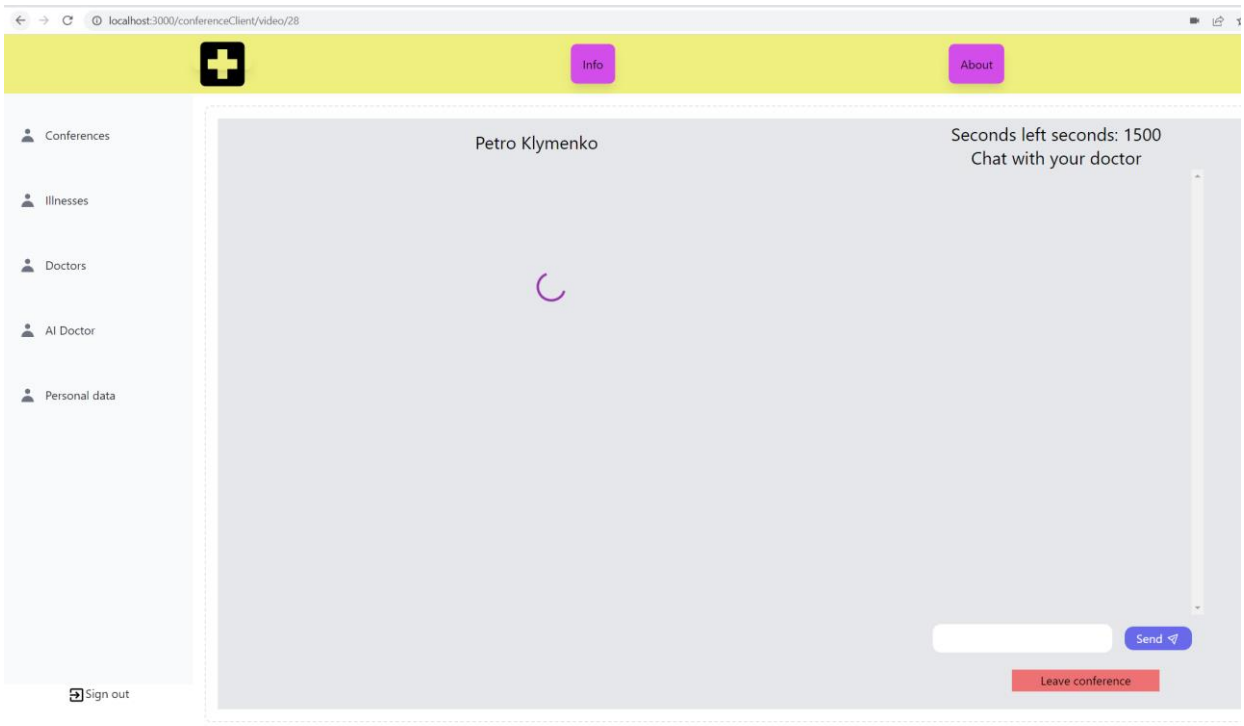

Рисунок 3.21 – Сторінка конференції в очікуванні на приєднання лікаря

Якщо один член конференції приєднався і очікує на приєднання іншого, то замість відео він буде бачити іконку завантаження. Відповідно коли інший член конференції приєднався, то відео буде включено і консультація розпочнеться. Як показано на рис. 3.21 у верхній правій частині ми бачимо кількість секунд яка залишилась у цій конференції, якщо цей час закінчиться, то консультація автоматично завершиться і нас буде знову перенаправлено на стартову сторінку. Після завершення консультації, гроші за неї будуть надіслані на рахунок лікаря, що її проводив.

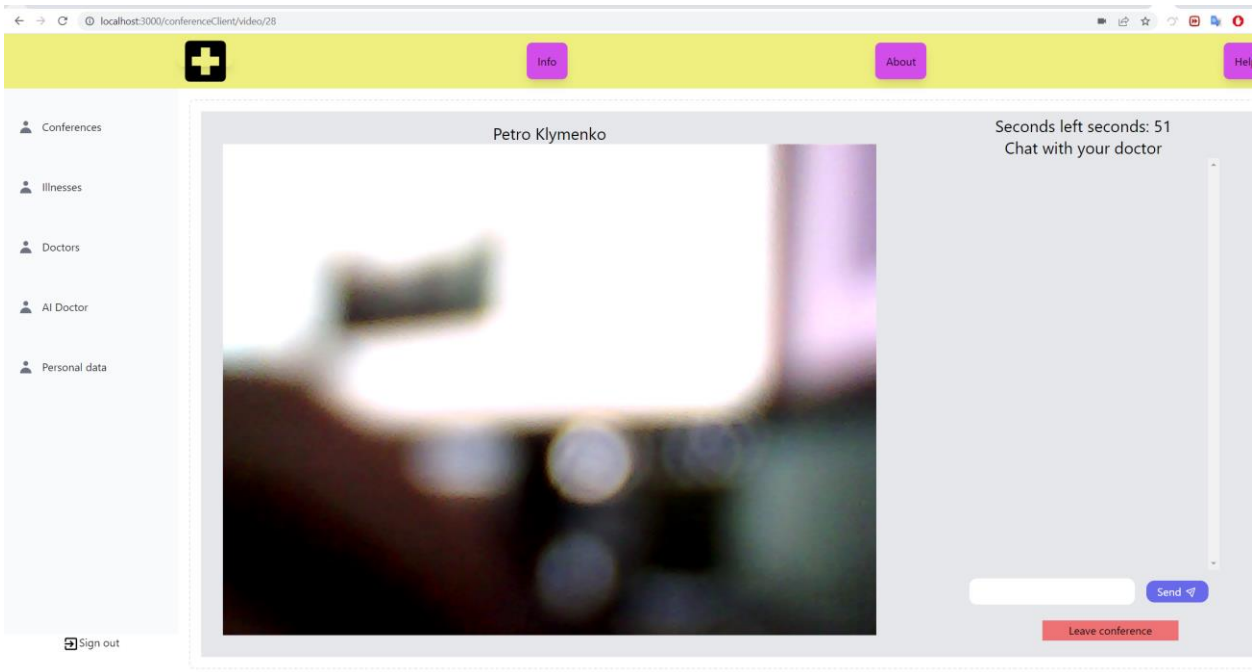

Рисунок 3.22 – Сторінка конференції коли обидва учасники приєдналися

Вся комунікація між пацієнтом та лікарем відбувається з використанням чату, який розміщений у правій частині сторінки консультації як ми можемо бачити на рисунку на рис. 3.22.

3.2 Розробка користувацької частини

Для написання FrontEnd частини системи була використана технологія React на мові Typescript. Весь користувацький інтерфейс системи складається з різних компонентів. Комбінація різних компонентів в одному компоненті дає змогу створювати цілі сторінки, які працюють динамічно і не потребують перезавантаження у випадку зміни вмісту сторінки.

Основний сервіс системи це “MainIntermediate” який базовим для всіх інших, і тут прописана логіка вибору потрібного меню яке є різне для адміністратора, лікаря та клієнта. Тут була прописана вся основна логіка системи, весь обмін данними, всі дані яки повернаються користувачу та обробка основних запитів відбувається через данний сервіс. У ньому налаштована безпека, тобто з кожним запитом відбувається перевірка токена доступу.

У лістингу 3.1 можна побачити код компонента “MainIntermediate”.

Лістинг 3.1 – Користувацький інтерфейс базовий компонент програми

```
export default function MainIntermediate(){
    const [authorized,setAuthorized] = useState(false);
    const [role,setRole] = useState('');
    const navigate = useNavigate();
    const logout = () => {
        setAuthorized(false);
        ClearLocalStorage();
    };
    useEffect(()=>{

        if(isAuthorized()){
            setRole(GetUserRole());
            setAuthorized(true);
        }
        else{
            navigate(PrjRoutes.login);
            setAuthorized(false);
        }},[]);
    return (
        <>
            {!authorized &&
                <IdentityRoutings login={login}/>
            }
            {authorized &&
                <>
                    {role === AccountNames.ClientAccount &&
                        <ClientAccount logout={logout}/>
                    }
                    {role === AccountNames.DoctorAccount &&
                        <DoctorAccount logout={logout}/>
                    }</>}</>)}
```

3.3 Розробка серверної частини

Серверна частина системи написана на C#, з використанням технології ASP .NET Core де загальний принцип роботи Web Api відбувається через Middlewares, які знаходяться в класі Startup, який описаний в лістингу 3.2. Тут є два метода, “ConfigureServices” та “Configure”, у першому налаштовані всі сервіси, які будуть добавлені в Dependency Injection контейнер, а в другому знаходяться ті самі Middlewares які будуть викликатись в певному порядку і модифікувати вхідний HttpContext об’єкт.

Лістинг 3.2 – Код налаштування MainApi сервісу

```
public class Startup
{
     public Startup(IConfiguration configuration)
     {
          Configuration = configuration;
     }

     public IConfiguration Configuration { get; }

     // This method gets called by the runtime. Use this method to
add services to the container.
     public void ConfigureServices(IServiceCollection services)
     {
          services.AddMainApiDIServices();

          services.AddSqlDatabase(Configuration);

        services
                .AddControllers()
                .AddJsonOptions(x =>
                    x.JsonSerializerOptions.ReferenceHandler =
ReferenceHandler.IgnoreCycles);

          services.AddSwaggerGen(c =>
          {
               c.SwaggerDoc("v1", new OpenApiInfo { Title =
"Example", Version = "v1" });
          });

          services.AddHealthChecks();

     public void Configure(IApplicationBuilder app,
IWebHostEnvironment env)
```

```
    {
        app.UseHttpsRedirection();

        app.UsePathBase(GlobalRoutePrefix.GetPrefix);
        app.UseRouting();

        app.UseMiddleware<ExceptionHandlingMiddleware>();

        app.UseCors(Pocilies.AppPolicy);

        app.UseAuthentication();
        app.UseAuthorization();

        app.UseEndpoints(endpoints =>
        {
            endpoints
                .MapControllers()
                .RequireAuthorization();
            endpoints.MapHealthChecks("/health");
        });
    }
}
```

Також для відправляння запиту з кінцевої точки “endpoint” до сервісу в якому прописана бізнес логіка була використана бібліотека MediatR, для того щоб чітко розділити код на команди і запити “CQRS”. У лістингу 3.2 знаходиться код підключення цієї бібліотеки в DI контейнер, де ми зчитуємо метадані всієї збірки, знаходимо базовий обробник, через метадані знаходимо метадані похідних класів, і підключаємо їх в DI контейнер.

Лістинг 3.3 – Підключення бібліотеки MediatR в DI контейнер

```
services.AddMediatR(x=>x.RegisterServicesFromAssembly(typeof(BaseHandler).GetTypeInfo().Assembly));
```

Як ви можемо бачити з лістингу 3.3, робота бібліотеки MediatR була налаштована з використанням метаданих збірки.

Тестування системи було здійснено в браузері Google Chrome для клієнта та в браузері Microsoft Edge для лікаря.

РОЗДІЛ 4 БЕЗПЕКА ЖИТТЄДІЯЛЬНОСТІ, ОСНОВИ ОХОРОНИ ПРАЦІ

4.1 Долікарська допомога при ураженні електричним струмом

Дуже широке та різноманітне використання електроенергетики вимагає відповідно обережного та правильного поводження з нею з дотриманням чітко визначених правил, порушення яких може призвести до важкої та інколи навіть до смертельної травми. Напруга 40 В є цілком безпечною для людини, проте струм вище 50 В може викликати тепловий і електролітичний ефект.

У більшості випадків ураження виникає внаслідок недотримання правил техніки безпеки при роботі з електропристроями як у побуті так і на виробництві.

Людина яка перебуває під ураженням електричного струму не може самостійно допомогти собі або покликати на допомогу. Дотик до предмету який перебуває під великою напругою призводить до судом м'язів, які частково або повністю обмежують людину і будь яких діях чи спілкуванні. У більшості випадків для того щоб зрозуміти що людина у небезпеці свідчить несподіване падіння посеред вулиці, раптове відкидання від джерела високої напруги без втручання сторонніх предметів, несподівана втрата свідомості, чітко виражене раптове скорочення м'язів або сильні опіку на тілі і особливо в місці ураження струмом.

Потрібно обов'язково попередити інших людей про небезпеку ураження електричним струмом, і попросити когось викликати швидку допомогу за той час поки ви намагаєтесь звільнити потерпілого від дії високої напруги.

Для того щоб звільнити потерпілого від дії електричного струму, у перш чергу вам потрібно знеструмити джерело високої напруги, це може бути обладнання або оголений провід, які внаслідок поломки стали причиною ураження людини електричним струмом. Для цього можна використати декілька

способів: вимикання рубильника, витягнути пробки на електричному щитку, знайти розетку та вимкнути джерело ураження струмом. Будь-яке зволікання при наданні долікарської допомоги нерідко призводить до летальних наслідків. Якщо неможливо уникнути прямого контакту із електричним струмом, використовуйте: сухий одяг, дерев'яну дошку або шмат резини.

Штучне дихання також є дуже важливою частиною серцево легеневої реанімації щоб забезпечити організм киснем. Потрібно приставити свої губи до губ потерпілого через стерильну серветку для уникнення потрапляння мікроб в ротову порожнину закривши йому ніс або закрити потерпілому рот та прикласти свій рот до носа потерпілого вдихаючи повітря у легені як показано на рис. 4.1. Потрібно робити 2 вдихи та 30 натискань непрямого масажу серця, якщо неможливо зробити штучне дихання, потрібно щонайменьше робити непрямий масаж серця.

Відразу після того як потерпілий був звільнений від дії електричного струму, потрібно негайно надати йому долікарську медичну допомогу, попередньо викликавши швидку допомогу за номером 103. Якщо людина не подає ознак життя, потрібно почати серцево-легеневу реанімацію.

Рисунок 4.1 – штучне дихання з використанням стерильної серветки

Непрямий масаж серця потрібно почати виконувати якомога швидше, потрібно виконувати натискання у нижній частині грудини, по центрі грудкої клітки, натискати потрібно в глибину не меньше 5 см але і не більше 6 см, із швидкістю 100-130 натисків за хвилину з якомога меньшою кількістю переривань, це може робити декілька людей по черзі, щоб дати людині перепочити. Натискання потрібно робити на твердій поверхні, після кожного натискання потрібно дати грудній клітці повністю випрямитись.

Потерпілого можна відтягнути від струмопровідних частин тримаючи його за сухий одяг і уникаючи дотику до будь яких струмопровідних предметів і оголених участків тіла потірпілого. Якщо ви тягнете потерпілого за ноги не доторкайтеся до взуття, оскільки воно може стати сирим і почати проводити електричний струм. Особа яка надає допомогу обов'язково повинна обмотати руки сухою тканиною чи одягнути діелектричні рукавиці. Існує ще один спосіб ізольувати себе, ставши на гумовий килим або дерев'янну суху дошку.

Після прибуття швидкої допомоги потрібно розказати про всю ситуацію лікарям у деталях якщо є можливість і на в якому разі не відпускати потерпілого навіть якщо він немає видимих наслідків ураження електричним струмом. Лікаря потрібно у будь якому разі незалежно від того чи притомний потерпілий чи знаходиться без свідомості. Навіть якщо до приїзду швидкої допомоги потерпілий прийшов до тями, його не слід відпускати додому. Над ним повинні встановити спостереження лікарі, оскільки наслідки ураження електричним струмом можуть проявлятись не відразу а через певний час і призвести до більш важких наслідків якщо не буде надана необхідна медична допомога.

4.2 Особливості заходів електробезпеки на підприємствах, які використовують хмарні сервіси

Існує три системи забезпечення електробезпеки:

– система технічних засобів і заходів;

– система електрозахисних засобів;

– система організаційно-технічних заходів і засобів.

Технічні заходи електробезпеки встановлюються у конструкцію певної електроустановки ще на етапі розробки. Технічні засоби забезпечення електробезпеки за функціями поділяються на дві групи:

– технічні заходи електробезпеки при нормальному режимі роботи електроустановки;
– технічні заходи електробезпеки при аварійній режимі роботи електроустановки.

Основні заходи забезпечення електробезпеки при нормальному режимі роботи електроустановок включають:

– ізоляцію струмопровідних частин;
– захисне розділення електричних мереж;
– компенсацію ємнісних струмів замикання на землю;
– недоступність струмовідних частин;
– вирівнювання потенціалів;
– блоківки безпеки;
– засоби орієнтації в електроустановках;
– виконання електроустановок, ізольованих від землі.

Для того щоб підвищити безпеку в електроустановах зазвичай використовують одночасно більшість з перерахованих технічних засобів.

Ізоляція струмопровідних частин покращує технічну працездатність установок, зменьшує ймовірність ураження працівника електричним струмом, замикань на землю і на корпус електроустановок, а також зменьшує струм через людину при дотику до незаізольованих струмо-провідних частин в електроустановах. Згідно ДСТУ існують 4 типи ізоляції [14]:

– робоча – забезпечує нормальну роботу електроустановокі створює захист від ураження високою напругою;
– додаткова – забезпечує захист від ураження високою напругою на випадок пошкодження ізоляції робочої ізоляції;
– подвійна – складається одночасно з додаткової та робочої;

– підсилена – поліпшена робоча ізоляція, який створює такий самий рівень захисту як і подвійна.

Протягом розробки електроустановок опір ізоляції приймається в межах 1кОм/В, якщо технічними умовами не передбачео більш жорсткі вимоги згідно до чинних актів. Для забезпечення безпеки і працездатності електричних установок, а також для безпечної їх експлуатації та підприємствах повинний буде постійни контроль стану ізоляції, який вимірює електричну міцність ізоляції, її електричним опором та діелектричними втратами. На підприємствах де є електричні пристрої з високою напругою, регулярнор проводять всі види випробовувань ізоляції, проте при низьких напругах проводиться контроль тільки електричної міцності і електричного опору. Існують також приймально-задавальні випробовування, міжремонтні і післяремонтні(після капітального ремонту), встановлені чинними нормативами в залежності від типу цієї електроустановки і умов її уксплуатації. Так опір електросвітильників, що живляться від електромережі,контролюється кожні 6 місяців, зварювальне обладнання – кожні 12 місяців. Опір ізоляції повинен бути не меньше 0.5 МОм, в той час я для електрофікованого інструменту – 1 Мом. [http://opcb.kpi.ua/]

Серед людей уражених електричним струмом в Україні, більшість електротравм є пов’язаними з дотиком до незаізольовних струмопровідних елеметів установок, які знаходяться під високою напругою (більше 50%). В той час як у пристроїв до 1000 В, небезпека електротравм пов’язана, в більшості випадків з доторканням до незаізольованих струмопровідних частин електроустановок, то при напрузі вище 1000 В електро-травми можливі при дотику навіть до заізольованих частин струмопровідних частин електроустановки. Тому струмопровідні частини електроустановок роблять зазвичай, за захисними огорожами, у недосяжному для людині місці(під землею чи на великій висоті) або у закритих комутаційних апаратах.

Блокування безпеки використовують в електроустановах де потрібен періодичний доступ до огороджених струмопровідних частин, комутаційних апаратах, помилки в оперативних переключеннях яких можуть спричинити аварії

і нещасні випадки.

Від протяжності електричної мережі та її розгалуженості залежить кінцевий опір ізоляції її проводів відносно змелів, а також ємкісна складова струму замикання на змелю. Тому при розділенні мережі використовують трансформатири з різним коефіцієнтом трансформації.

Вирівнювання потенціалів використовується для зниження можливих напруг дотику, які можуть виникнути протягом експлуатації електроустановок або потраплянні людини під дію цієї напруги за за інших обставин. Цього ефекту можна досягти за рахунок підвищення потенціалу опорної поверхні, на якій буде стояти людина, до рівня потенціалу струмопровідних частин, до яких ця людина буде доторкатись або також шляхом зменьшення перепаду потенціалів на площі де буде стояти працівник.

Для того щоб працівники були проінформовані про потенційно небезпечні місця на електроустановках, де є загроза отримання удару електричним струмом, завжди застосовуються засоби орієнтації в електроустановках, які дають можливість персоналу чітко орієнтуватись при монтажі, виконанні ремонтних робіт чи екплуатації цих пристроїв. До засобів орієнтації в електроустановах належать: попереджувальні знаки, знаки високої напруги, комутаційні схеми, таблички, написи, ізоляції, попереджувальні сигнали, кольорові рішення неозольованих струмопровідних частин, бирки на проводах, маркування частин електрообладнання, проводів та струмопроводів. Величина струму яка пройде через людину визначається опором ізоляції фаз відносно поверхні землі, і становить не меньше 1К Ом, при однофазному підключенні людини під напругу і відсутності пошкодження ізоляції інших фаз. Відповідно, виконання електричних мереж, сильно обмежує величину струму яка пройде через людину за рахунок опору ізоляції фаз відносно землі за умови забезпечення потрібного стану ізоляції. Якщо є наявність фаз пошкоджених ізоляцією і дотику людини до фізного проводу з непошкодженою ізоляцією сила струму яка пройде через людину зростає. Тому застосування мереж, ізольованих від землі потребують обов’язкового контролю опору ізоляції.

ВИСНОВКИ

Робота присвячена створенню системи електронного консультування пацієнта для чого було проведено огляд предметної області та виокремлено ключові сутності, які були відображені у програмній системі.

В процесі розроблення системи проведення відеоконсультацій з лікарем було розглянуту та обґрунтовано вибір технологій, а саме: бібліотек для створення програмного забезпечення (WEB RTC, MaterialUI, React, ASP .NET Core,Entity Framework Core,SignalR, FluentValidator, Automapper, Identity Server4,MadiatR) та хмарних сервісів у яких воно буде розгорнуто(Kubernetes, Blob storage, Key Vault, Configuration storage). Це дало змогу знизити затрати на створення та супровід системи.

У роботі пояснено підходи, які використано для створення програмного забезпечення, а саме Domain Driven Design, Command Query , Microservices architecture, багаторівнева архітектура та технології об’єктно-реляційних відображень для взаємодії з базами даних. Використані в ході створення системи мови програмування: TypeScript (фронтенд), C# (бекенд).

Під час проектування програмного забезпечення було розглянуто логічну структуру системи, а також програмні складові сервісів з користувацьким інтерфейсом.

Розроблено алгоритми роботи відповідних модулів, які використовуються основними сервісами системи. Спроектовано інтерфейс користувача з урахування функційних вимог та зручності використання. Запропоновані шляхи інтеграції з системами штучного інтелекту, на прикладі Chat GPT, як електронного лікаря.

Проведено тестування роботи системи й перевірено працездатність системи в цілому та всіх її модулів.

У роботі також розглянуто питання охорони праці. У відповідному розділі було здійснено огляд долікарської допомоги при уражені електричним стурмом та були пояснені особливості заходів електробезпеки на підприємствах, які використовують хмарні сервіси.

ПЕРЕЛІК ПОСИЛАНЬ

1. Макмілан М. Data Structures and Algorithms with JavaScript. Київ, 2017. 138 с.

2. Троелсон А. Принципи і практики C# 10 with .NET 6. Київ, 2019. 321 с.

3. Стефанов С. React: Up & Running: Building Web Applications, Київ, 2018. 186с.

4. Технічна документація MediatR. URL: https://github.com/jbogard/MediatR (дата звернення 15.04.2023)

5. Технічна документація IdentityServer4. URL: https://identityserver4.readthedocs.io/ (дата звернення 24.03.2023)

6. Технічна документація FluentValidator. URL: https://docs.fluentvalidation.net/ (дата звернення 20.05.2023)

7. Технічна документація EntityFramework. URL: https://www.learnentityframeworkcore.com/ (дата звернення 10.04.2023)

8. Тезхнічна документація Visual Studio Enterprise 2022. URL: https://visualstudio.microsoft.com/ru/vs/ (дата звернення 15.05.2023)

9. Jeffrey Richter CLR via C#. Київ, 2012. 273 с.

10. Роберт Мартін. Чиста Архітектура. Київ, 2020. 135с.

11. Роберт Мартін Чистий Код. Харків, 2021. 84с.

12. Марк Д. C# 11 .NET 7 Фундаментальні основи кросплатформенної роборобки. Київ, 2023р. 131с.

13. Ушкалом О. БЖД. Київ, 2007р. 150с.

14. Атаманчук П. Безпека життєдіяльності. Київ, 2015р. 134с.

15. Роберт М. Чистий Agile. Київ, 2019р. 212с.

16. Луцків А. Імітаційне моделювання циклічних випадкових процесів. Львів, 2006р. 210с.

17. Луцків А. Оцінювання ймовірнісних характеристик динамічно введеного підпису для задач аутентифікації особи в інформаційних системах. Київ, 2006р. 152с.

18. Осухівська Г.М., Тиш Є.В., Луцик Н.С., Паламар А.М. Методичні вказівки до виконання кваліфікаційних робіт здобувачів першого (бакалаврського) рівня вищої освіти спеціальності 123 «Комп'ютерна інженерія» усіх форм навчання. Тернопіль, ТНТУ. 2022. 32 с.

Додаток А

Технічне завдання

Тернопільський національний технічний університет імені Івана Пулюя

Факультет комп’ютерно-інформаційних систем і програмної інженерії

Кафедра комп’ютерних систем та мереж

“Затверджую”
Завідувач кафедри КС
_____ Осухівська Г.М.
“16” Березня 2023 р

КОМП’ЮТЕРИЗОВАНА СИСТЕМА ЕЛЕКТРОННОГО КОНСУЛЬТУВАННЯ ПАЦІЄНТІВ З ВИКОРИСТАННЯМ ХМАРНИХ СЕРВІСІВ.

ТЕХНІЧНЕ ЗАВДАННЯ
на _9_ листках

Вид робіт: Кваліфікаційна робота

На здобуття освітнього ступеня «Бакалавр»

Спеціальність 123 «Комп’ютерна інженерія»

«УЗГОДЖЕНО»	«ВИКОНАВЕЦЬ»
Керівник кваліфікаційної роботи	Студент групи СІ-41
_____ к.т.н., доц. Луцків А.М.	_____ Люлька А.В.
«____» _____ 2023 р.	«____» _____ 2023 р.

Тернопіль 2023

1 Загальні відомості

1.1 Повна назва та її умовне позначення

Повна назва теми кваліфікаційної роботи: «Комп’ютерна система IP-відеоспостереження супермаркету Наш край».

Умовне позначення кваліфікаційної роботи: КСКП 123.046.00.00 ПЗ

1.2 Виконавець

Студент групи СІ-41, факультету комп’ютерно-інформаційних систем і програмної інженерії, кафедри комп’ютерних систем та мереж, Тернопільського національного технічного університету імені Івана Пулюя, Люлька Андрій Вікторович.

1.3 Підстава для виконання роботи

Підставою для виконання кваліфікаційної роботи є наказ по університету (№4/7-180 від 23.03.2022 р.)

1.4 Планові терміни початку та завершення роботи

Плановий термін початку виконання кваліфікаційної роботи – 23.03.2022 р.

Плановий термін завершення виконання кваліфікаційної роботи – 26.06.2022 р.

1.5 Порядок оформлення та пред’явлення результатів роботи

Порядок оформлення пояснювальної записки та графічного матеріалу здійснюється у відповідності до чинних норм та правил ІСО, ГОСТ, ЕСКД, ЕСПД та ДСТУ.

Пред’явлення проміжних результатів роботи з виконання кваліфікаційної роботи здійснюється у відповідності до графіку, затвердженого керівником роботи. Попередній захист кваліфікаційної роботи відбувається при готовності роботи на 90% , наявності пояснювальної записки та графічного матеріалу.

Пред’явлення результатів кваліфікаційної роботи відбувається шляхом захисту на відповідному засіданні ЕК, ілюстрацією основних досягнень за допомогою графічного матеріалу.

2 Призначення і цілі створення системи

2.1 Призначення системи

Система консультації що розробляється призначена для:

– Реєстрації хвороби,
– Знаходження потрібного лікаря,
– Організація консультації.

2.2 Мета створення системи

Метою кваліфікаційної роботи є розробка комп’ютеризованої системи електронного консультування пацієнтів з використанням хмарних сервісів.

2.3 Характеристика об'єкту

Об'єкт, для якого розробляється дана комп’ютеризована система є хмарне середовище дата центру Azure. У хмарному середовищі ця система буде розгорнута та налаштована так, щоб пацієнти мали змогу організовувати онлайн консультації з лікарями по всьому світу.

3 Вимоги до системи

3.1 Вимоги до системи в цілому

3.1.1 Вимоги до структури та функціонування системи

Комп’ютерна система відеонгляду повинна складатись з:

- Сервера;
- Відеокамери;
-Окремого комп’ютера для кожного користувача.

3.1.2 Вимоги до способів та засобів зв’язку між компонентами системи

Основна вимога, яка ставиться до способів та засобів інформаційного обміну – це їх узгодженість.

3.1.3 Вимоги по діагностуванню системи

Діагностування комп’ютеризованої системи проводиться самим дата центром відповідно до їх регламенту.

3.1.4 Вимоги до режимів функціонування системи

Для системи визначено два режими функціонування:

– нормальний режим функціонування;
– аварійний режим функціонування.

Основним режимом функціонування є нормальний режим.

Для забезпечення нормального режиму функціонування системи необхідно виконувати вимоги і дотримуватись умов експлуатації програмного забезпечення і комплексу технічних засобів системи, вказані у

відповідних технічних документах (технічна документація, інструкції з експлуатації і т. д.).

Аварійний режим функціонування системи характеризується відмовою одного або декількох компонент програмного і (або) технічного забезпечення. При цьому функції роботи системи продовжують підтримувати роботу комп’ютеризованої системи організації консультацій в межах базових налаштувань.

3.1.5 Вимоги по діагностуванню системи

Для діагностування системи консультації використовуються інструменти діагностування основних процесів системи, які вмонтовані в операційну систему сервера.

Інструменти повинні забезпечувати зручний інтерфейс для можливості перегляду діагностичних подій, моніторингу процесу виконання програм.

3.1.6 Перспективи розвитку, проектування системи

Дана система може бути розширена оскільки спроектована архітектурно правильно, тому туди легко може бути дописаний додатковий функціонал.

3.2 Показники призначення

Система повинна передбачати можливість масштабування. Можливості масштабування повинні забезпечуватися засобами використовуваного базового програмного і технічного забезпечення.

3.2.1 Вимоги до надійності

Система повинна забезпечувати працездатність та відновлення своїх функцій при виникненні наступних ситуацій:

– при збоях в системі електропостачання апаратної частини;

– при помилках в роботі апаратних засобів;

– при помилках, пов'язаних з програмним забезпеченням (ОС і драйвери пристроїв).

Для захисту апаратури від стрибків напруги і комутаційних завад повинні застосовуватися мережні фільтри.

3.3 Вимоги до безпеки

Зовнішні елементи технічних засобів системи, що перебувають під напругою, повинні мати захист від випадкового дотику, а самі технічні засоби мати занулення або захисне заземлення .

Система електроживлення повинна забезпечувати захисне вимикання при перевантаженнях і коротких замиканнях в колах навантаження, а також аварійне ручне вимикання.

Загальні вимоги пожежної безпеки повинні відповідати нормам на побутове електрообладнання. У разі пожежі не має виділятися отруйних газів і димів. Після зняття електроживлення має бути доступне застосування будь-яких засобів пожежогасіння.

Шкідливі фактори не повинні перевищувати норм СанПиН 2.2.2./2.4.1340-03 від 03.06.2003 р.

3.3.1 Вимоги до експлуатації, технічного обслуговування, ремонту і зберігання компонентів системи

Мікроклімат в приміщеннях повинен відповідати нормам виробничого мікроклімату по ГОСТ 15150-69,УХЛ 4,1.

– температуру повітря в межах від +10□С до +35□С;

– відносну вологість повітря при 25□С в межах від 30% до 80%;

– атмосферний тиск 760□25 мм рт. ст.

Періодичне технічне обслуговування використовуваних технічних засобів має проводитися відповідно до вимог технічної документації, але не рідше ніж один раз на рік.

Періодичне технічне обслуговування і тестування технічних засобів повинні включати обслуговування і тестування всіх використовуваних засобів, датчики, контроллери, системи передачі даних, пристрої безперебійного живлення.

На підставі результатів тестування технічних засобів повинні проводитися аналіз причин виникнення виявлених дефектів і прийматися заходи по їх ліквідації.

3.4 Вимоги до захисту інформації від несанкціонованого доступу

Система повинна забезпечувати захист від несанкціонованого доступу на рівні не нижче встановленого вимогами, що пред'являються до категорії 1Д по класифікації документа, що діє, “Автоматизовані системи. Захист від несанкціонованого доступу до інформації. Класифікація автоматизованих систем”.

Компоненти підсистеми захисту від НСД повинні забезпечувати:

– ідентифікацію користувача;
– перевірку повноважень користувача при роботі з системою;
– розмежування доступу користувачів.

Рівень захищеності від несанкціонованого доступу засобів обчислювальної техніки, що здійснюють обробку конфіденційної інформації, повинен відповідати вимогам класу захищеності згідно вимогам документу “Засоби обчислювальної техніки. Захист від несанкціонованого доступу до інформації. Показники захищеності від несанкціонованого доступу до інформації”.

3.4.1 Вимоги по збереженню інформації при аваріях

Інформація, при виникненні аварійних ситуацій повинна бути збережена на резервних носіях.

3.4.2 Вимоги по стандартизації і уніфікації

Система повинна відповідати вимогам ергономіки і зручності користування за умови комплектування високоякісним обладнанням (ЕОМ, монітор і інше обладнання), що має необхідні сертифікати відповідності і безпеки.

3.4.3 Вимоги до функцій (завдань), що виконуються системою:

– забезпечення якісного зображення;
– забезпечення якісного обміну інформацією;
– зручність та легкість користування;
– забезпечення високої швидкодії.

4 Вимоги до документації

Документація повинна відповідати вимогам ЄСКД та ДСТУ

Комплект документації повинен складатись з:

– пояснювальної записки;
– графічного матеріалу:

*Примітка: У комплект документації можуть вноситися міни та доповнення в процесі розробки.

5 Стадії та етапи проектування

Таблиця 1 – Стадії та етапи виконання кваліфікаційної роботи бакалавра

№ етапу	Назва етапу виконання кваліфікаційної роботи	Термін виконання
1	Ознайомлення з завданням до кваліфікаційної роботи	3.03-25.03.2023
2	Провести огляд літературних джерел по темі випускної бакалаврської роботи	26.03-01.04.2023
3	Провести порівняльну характеристику програмних продуктів	02.04-16.04.2023
4	Розробити функціональну схему роботи об’єкта проектування. Розробка моделі бази даних	17.04-04.05.2023
5	Практична реалізація об’єкта проектування	05.05-10.05.2023
6	Описати та розробити організаційні заходи спрямовані на поліпшення стану охорони праці	11.05-29.05.2023
7	Виконання завдання до підрозділу «Безпека життєдіяльності»	30.05-05.06.2023
8	Виконання завдання до підрозділу «Основи хорони праці»	06.06-10.06.2023
9	Оформлення кваліфікаційної роботи	11.06-15.06.2023
10	Перевірка на плагіат	14.06-15.06.2023
11	Нормоконтроль	19.06-20.06.2023

6 Додаткові умови виконання кваліфікаційної роботи

Під час виконання кваліфікаційної роботи у дане технічне завдання можуть вноситися зміни та доповнення.

Додаток Б

Код частини користувацького інтерфейсу системи.

```
export default function ClientAccount({logout} : LogOutCallback){
    return(
        <>
            <Header/>
            <Routings logout={logout}/>
            <Menu logout={logout}/>
        </>
    );
}
export function Header() {
    return(
        <div className=" w-full mainElementsBgColor h-[80px] grid grid-cols-4">
            <HeaderNavigation/>
        </div>
    )
}
export function HeaderNavigation() {
    return(
        <>
            <div className='place-items-center rounded-lg ml-64 h-full'>
                <Link className='rounded-lg shadow-lg h-[50px] w-[50px] mt-10px'
to={PrjRoutes.startPage}>
                    <LocalHospitalIcon className='min-h-[80px] min-w-[80px]'/>
                </Link>
            </div>
            <div className = 'place-items-center rounded-lg ml-64 mt-7 '>
                <Link className='p-4 rounded-lg shadow-lg bg-fuchsia-500'
to={PrjRoutes.infoPage}>Info</Link>
            </div>
            <div className='place-items-center rounded-lg ml-64 mt-7 '>
                <Link className='p-4 rounded-lg shadow-lg bg-fuchsia-500'
to={PrjRoutes.aboutPage}>About</Link>
            </div>
            <div className='place-items-center rounded-lg ml-64 mt-7 '>
                <Link className='p-4 rounded-lg shadow-lg bg-fuchsia-500'
to={PrjRoutes.helpPage}>Help</Link>
            </div>
        </>
    )
}

export default function LoginPage({login} : LogInCallback){
    const navigate = useNavigate();
    const [error, setError] = useState<ErrorModel>({
        isError: false,
        errorMessage: ''
    });

    const [loginData, setLoginData] = useState<LoginModel>({
        login: "",
        password: ""});

    const loginClick = () => {
        Login(loginData.login, loginData.password)
            .then(response=> {
                if(isResultSuccess(response)){
                    SetAccessToken(response.data.access_token);
                    SetRefreshToken(response.data.refresh_token);
```

```
                    setError({
                        isError: false,
                        errorMessage: ""
                    });
                    login();
                    navigate(PrjRoutes.startPage);
                }
            })
            .catch(error => {
                const errorResponse = error.response.data.error_description;
                setError({
                    isError: true,
                    errorMessage: errorResponse
                });
            });
    }
    const handleChange = (event: React.ChangeEvent<HTMLInputElement>) => {
        const target = event.target;
        const value = target.value;
        const name = target.name;
        setLoginData((prev) => {
            return {...prev, [name]: value}
        })
    }
    return(
        <>
            {error!.isError &&
                <Error errorMessage={error.errorMessage}></Error>
            }
            <div className="mt-[14em] h-full ">
                <div className="mx-auto container w-200 h-800">
                    <Box textAlign='center'>
                        <Link to={PrjRoutes.registration}>
                            Registration <ArrowForwardIcon/>
                        </Link>
                    </Box>
                    <br/>
                    <Box textAlign='center'>
                        <TextField label="Login" name='login'
onChange={handleChange}></TextField>
                    </Box>
                    <br/>
                    <Box textAlign='center'>
                        <TextField type='password' label="Password"
name='password' onChange={handleChange}></TextField>
                    </Box>
                    <br/>
                    <Box textAlign='center'>
                        <Button type="submit" variant="contained"
className="button bg-cyan-700 w-[130px] h-[60px] text-3xl" onClick={loginClick}>
                            Login
                        </Button>
                    </Box>
                </div>
            </div>
        </>
    );
}
export default function IdentityIntermediate({login} : LogInCallback){
    return(
        <>
            <Routes>
                <Route path={PrjRoutes.login} element={
                    <IdentityWrapper>
                        <LoginPage login={login}/>
```

```
                    </IdentityWrapper>}>
                </Route>
                <Route path={PrjRoutes.registration} element={
                    <IdentityWrapper>
                        <RegisterPage login={login}/>
                    </IdentityWrapper>}>
                </Route>
                <Route path={PrjRoutes.doctorRegistration} element={
                    <IdentityWrapper>
                        <RegisterDoctorPage login={login}/>
                    </IdentityWrapper>}>
                </Route>
            </Routes>
        </>
    );
}
export default function IdentityWrapper({children} : {children :
React.ReactNode}){
    const [authorized,setAuthorized] = useState(false);
    const navigate = useNavigate();

    useEffect(()=>{
        if(isAuthorized()){
            setAuthorized(true);
            navigate(PrjRoutes.startPage);
        }
        else{
            setAuthorized(false);
        }
    },[]);

    return(
        <>
            {!authorized &&
                <>
                    <Box textAlign='center' className='w-full pt-10'>
                        <h1 className='text-4xl'>K Y H</h1>
                    </Box>
                    {children}
                </>
            }
        </>
    )
}
interface Props{
    allMessages : MessageItem[]
}
export function MessageItems({allMessages} : Props){

    return(
        <>
           {allMessages.map((item,index)=>
               <li key={index}>
                   <div className='grid grid-cols-12'>
                       {!item.IsAi
                           ?
                           (<>
                               <div className='col-start-1 col-span-1 bg-yellow-
400 py-2 px-4 rounded-xl'>
                                   {item.Author}
                               </div>
                               <div className='col-start-2 col-span-4 relative
ml-3 text-sm bg-red-100 py-2 px-4 shadow rounded-xl'>
                                   {item.Message}
                               </div>
```

```
                            </>)
                            :
                            (<>
                                <div className='col-start-8 col-span-1 bg-yellow-
400 py-2 px-4 rounded-xl'>
                                    {item.Author}
                                </div>
                                <div className='col-start-9 col-span-4 relative
ml-3 text-sm bg-blue-500 py-2 px-4 shadow rounded-xl'>
                                    {item.Message}
                                </div>
                            </>)
                        }
                    </div>
                </li>
            )}
        </>
    );
}
export default function ConferenceDetailsPage(){
    const [startDateReal, setStartDateReal] = useState<Date | null>(null);
    const [endDateReal, setEndDateReal] = useState<Date | null>(null);

    const [conferenceButtonActive,setConferenceButtonActive] =
useState<boolean>(false);
    const [conferenceLink,setConferenceLink] = useState<string>("");
    const [currentConference, setCurrentConference] = useState<Consultation>();
    const { id } = useParams();

    const [accessibleCameras,setAccessibleCameras] =
useState<MediaDeviceInfo[]>();
    const [mySelectedDeviceId, setMySelectedDeviceId] = useState<string>("");
    const [errorMessage, setErrorMessage] = useState<string | null>(null);

    const statusObject = GetStatusObject(currentConference?.status ?? 10);

    async function SetCurrentConsultation(){
        const response = await GetConference(id);

        setCurrentConference(response);
setConferenceLink(PrjRoutes.conferenceClientVideo.replace(":id",`${response.id}`)
);
    }
    useEffect(() => {
        GetVideoSources()
        .then(response => {
            setAccessibleCameras(response);
        })
        .catch(error => console.log(error));

        SetCurrentConsultation()
        .then()
        .catch(error => console.log(error));
    },[]);

    useEffect(() => {
        const startDateTime = new Date(currentConference?.startDateTime+"Z" ??
"");
        const endDateTime = new
Date(Date.parse(currentConference?.endDateTime+"Z" ?? ""));
        const currentDateTime = new Date();
        setStartDateReal(startDateTime);
        setEndDateReal(endDateTime);
        if(startDateTime && endDateTime){
```

```
            const conferenceActive =  (startDateTime?.valueOf() <=
currentDateTime.valueOf()) && (currentDateTime.valueOf() <=
endDateTime?.valueOf());
            setConferenceButtonActive(conferenceActive);
        }
    },[currentConference]);
    const payConsultationClick = () => {
        if(id){
            PayConsultation(id)
                .then(response => {
                    SetCurrentConsultation()
                        .then(response => {
                            setErrorMessage(null);
                        })
                        .catch(error => console.log(error));
                })
                .catch(error => {
                    setErrorMessage(error.response.data.resultMessage);
                    console.log(error);
                });
        }
    };
    const cancelConsultationClick = () => {
        if(id){
            CancelConsultation(id)
            .then(response => {
                SetCurrentConsultation()
                    .then(response => {
                        setErrorMessage(null);
                    })
                    .catch(error => console.log(error));
            })
            .catch(error => {
                setErrorMessage(error.response.data.resultMessage);
                console.log(error);
            });
        }
    };
    return(
        <>
            <div className="w-full h-[50em]">
                <div className='bg-red-400 text-5xl h-11'>
                    <h1 className='text-center text-3xl underline'>Conference
details</h1>
                </div>
                <div className="w-full h-full flex">
                    <div className="w-[34%]">
                        <div className="text-center">
                            <p className="underline text-2xl">Doctor</p>
                        </div>
                        <br/>
                        <div className="pl-[3em]">
                            <span className="underline text-xl">Full name</span>
                            <span>:  </span>
                            <span className='text-
2xl'>"{currentConference?.doctor.user.lastName}
{currentConference?.doctor.user.firstName}"</span>
                        </div>
                        <div className="pl-[3em]">
                            <span className="underline text-xl">Medical
name</span>
                            <span>:  </span>
                            <span className='text-
2xl'>"{currentConference?.doctor.name}"</span>
                        </div>
```

```
                        <div className="pl-[3em]">
                            <span className="underline text-
xl">Qualification</span>
                            <span>:  </span>
                            <span className='text-
2xl'>"{currentConference?.doctor.qualification}"</span>
                        </div>
                        <div className="pl-[3em]">
                            <span className="underline text-xl">Experience</span>
                            <span>:  </span>
                            <span className='text-
2xl'>{currentConference?.doctor.experience} years</span>
                        </div>
                        <div className="pl-[3em]">
                            <span className="underline text-
xl">Description</span>
                            <span>:  </span>
                            <span className='text-
2xl'>"{currentConference?.doctor.description}"</span>
                        </div>
                        <div className="pl-[3em]">
                            <span className="underline text-xl">On platform
since</span>
                            <span>:  </span>
                            <span className='text-
2xl'>{moment(currentConference?.doctor.user.registrationDate ??
"").format("YYYY/MM/DD")}</span>
                        </div>
                        <p className="text-red-500 text-center text-
2xl">{errorMessage ?? ""}</p>
                        <div className="w-full ml-[9em] mt-[8em]">
                            <Button type="submit" color="error"
variant="contained" disabled={CancelButtonDisabled(currentConference)}
onClick={cancelConsultationClick} className="text-2xl bg-red-600 w-[8em] h-[3em]
mx-auto">Cancel</Button>
                            <br/>
                            <br/>
                            <Button type="submit" color="warning"
variant="contained" disabled={PayButtonDisabled(currentConference)}
onClick={payConsultationClick} className="text-2xl bg-orange-500 w-[8em] h-[3em]
mx-auto">Pay</Button>
                        </div>
                    </div>
                    <div className="w-[33%]">
                        <ul className="w-full h-[48em] bg-gray-300">
                            <p className="text-center text-3xl">Illnesses</p>

{currentConference?.consultation_ClientIllnessOrgans?.map((item,index)=>
                                <ConferenceDetailsIllnessSubItem index={index}
illness={item.client_IllnessOrgan}/>)}
                        </ul>
                    </div>
                    <div className="w-[28%] ml-[8em] text-center">
                        <div className="text-3xl">
                            <p>General info</p>
                        </div>
                        <div className={`text-center
${statusObject.statusColor}`}>
                            <span className="text-
2xl">{statusObject.statusName}</span>
                        </div>
                        <br/>
                        {currentConference?.isPaid ?
                            <p className="bg-greed-300 text-xl">"Consultation is
paid"</p> :
```

```
                            <p className="bg-red-400 text-xl">"You didn't pay
this consultation"</p>}
                        <div className="w-full h-[37em]">
                            <div className="w-full h-[27%]">
                                <br/>
                                <p className="text-center text-2xl">
                                    <span className="underline">Price</span>
                                    <span>: </span>
                                    <span>{currentConference?.totalPrice}
$</span>
                                </p>
                            </div>
                            <div className="w-full h-[26%]">
                                <select className="text-2xl" onChange={(e) => {
SetLocalStorageValue<string>(LocalStorageKeys.chosenCameraId,e.target.value);
                                    setMySelectedDeviceId(e.target.value);
                                }}>
                                    <option>Select</option>
                                    {accessibleCameras?.map((item,index)=>
                                    <CameraOption index={index} camera={item}/>)}
                                </select>
                            </div>
                            <div className="w-full h-[27%]">
                                <p className="underline text-center text-
2xl">Date of consultation:</p>
                                <p className="text-center text-
xl">{`${moment(startDateReal?.toString() ?? "").format("YYYY/MM/DD")} -
${moment(endDateReal?.toString() ?? "").format("YYYY/MM/DD")}`}</p>
                                <p className="text-center text-xl">
                                    <span>Duration: </span>
                                    <span>{moment(startDateReal?.toString() ??
"").format("HH:mm")}</span>
                                    <span> - </span>
                                    <span>{moment(endDateReal?.toString() ??
"").format("HH:mm")}</span>
                                </p>
                            </div>
                            <div className="w-full h-[13%] text-xl">
                                <p className="text center">Button below will
become active when</p>
                                <p className="text center">consultation
begun.</p>
                            </div>
                        </div>
                        <Button disabled={(!conferenceButtonActive ||
mySelectedDeviceId.includes("Select") || !currentConference?.isPaid ||
IsScheduled(currentConference?.status)
                        )}  variant="contained" size="large"
href={conferenceLink} className="w-[250px] h-[60px] text-4xl">Join</Button>
                    </div>
                </div>
            </div>
        </>
    );
}
export default function ConferenceVideoPage(){
    const navigate = useNavigate();
    const { id } = useParams();
    const [socketConnection, setSocketConnection] = useState<HubConnection>();
    const [currentConferenceData, setCurrentConferenceData] =
useState<Consultation>();

    const [stream, setStream] = useState<MediaStream>();

    const [currentUserId, setCurrentId] = useState<string | null>(null);
```

```
    const myVideo = useRef<HTMLVideoElement>(null);

    const mySelectedDeviceId : string =
GetLocalStorageValue<string>(LocalStorageKeys.chosenCameraId);

    //in reality must be false
    const [doctorJoined, setDoctorJoined] = useState<boolean>(false);

    const videoCameraId : string = "videoCameraId";

    const [videoElemId, setVideoElemId] = useState<string>(videoCameraId);

    const [messagesList, setMessagesList] =
useState<ConferenceMessageItem[]>([]);

    const [currentUserName, setCurrentUserName] = useState<string>();

    const [currentChatText, setCurrentChatText] = useState<string | null>("");

    const [startEndTimeDifference, setStartEndTimeDifference] =
useState<TimerData | null>(null);
    const [isTimerActive, setIsTimerActive] = useState<boolean>(true);

    useEffect(() => {
        CheckAccess(Number(id))
        .then(response => {
            if(!response.isSuccess && response.isFailed){
                setDoctorJoined(false);
                socketConnection?.invoke("Leave",id);
                navigate(PrjRoutes.startPage);
            }
            MakeConsultationInProgress(id)
            .then(response => {
                GetConference(id)
                .then(response => {
                    setCurrentConferenceData(response);
                    const timeDifference = new Date(new
Date(response.endDateTime).valueOf() - new Date().valueOf());
                    setStartEndTimeDifference({
                        hours : timeDifference.getHours(),
                        minutes : timeDifference.getMinutes(),
                        seconds : timeDifference.getSeconds()
                    });

                    setCurrentUserName(`${response.client.firstName}
${response.client.lastName}`);

                    const connection = GetConfiguredSocket();
                    connection.then(finalConnection => {
                        finalConnection.start()
                        .then(() => {
                            setSocketConnection(finalConnection);
                        })
                        .catch(error => console.log(error));
                    });
                })
                .catch(error => console.log(error));
            })
            .catch(error => console.log(error));
        })
        .catch(error => {
            console.log(error);
            setDoctorJoined(false);
            socketConnection?.invoke("Leave",id);
```

```
            navigate(PrjRoutes.startPage);
        });

        return () => {
            if(socketConnection){
                socketConnection.invoke("Leave",id);
            }
            stream?.getTracks().forEach((track) => {
                track.stop();
            });
            myVideo.current?.pause();
            setDoctorJoined(false);
        };
    }, []);
    useEffect(() => {
        if(socketConnection){
            socketConnection.on("Hub",(userLogin, message)=> {
                console.log(`"${userLogin}": ${message}`);
            });
            socketConnection.on("UserJoined",()=> {
                console.log("UserJoined");
                setDoctorJoined(true);
            });
            socketConnection.on("OponentLeft",()=> {
                console.log("OponentLeft");
                setDoctorJoined(false);
            });
            if(currentConferenceData && id){
                socketConnection.invoke("Join",id)
                .then((myConnectionId) => {
                    setCurrentId(myConnectionId);
                    socketConnection.invoke("SendQuestion",id);
                    socketConnection.on("Question",() => {
                        socketConnection.invoke("SendAnswer",id);
                        //window.location.reload();
                    });
                    socketConnection.on("Answer",() => {
                        setDoctorJoined(true);
                        setVideoElemId(videoCameraId);
                    });
                    socketConnection.on("ChatHub", (userLogin : string, message :
string) => {
                        const myMessageItem : ConferenceMessageItem = new
ConferenceMessageItem(false, userLogin, message);

                        setMessagesList(previous => [...previous,
myMessageItem]);
                    });
                })
                .catch(error => console.log(error));

                navigator.mediaDevices.getUserMedia(
                    {video: {
                        deviceId: mySelectedDeviceId
                    },
                    audio: {
                        deviceId: mySelectedDeviceId
                    }})
                    .then((stream) =>{
                        setStream(stream);
                        if(myVideo.current){
                            myVideo.current.srcObject = stream;
                            myVideo.current.play();
                        }
                    })
```

```
                    .catch(error => console.log(error));
            }
        }
    }, [socketConnection]);

    function handleInputChange(event: React.ChangeEvent<HTMLInputElement>){
        setCurrentChatText(event.target.value)
    }

    const onTimeUpCallback = () => {
        console.log("On callback after time interval works");
        MakeConsultationFinished(id)
        .then(response => {
            setDoctorJoined(false);
            socketConnection?.invoke("Leave",id);
            navigate(PrjRoutes.startPage);
            window.location.reload();
        })
        .catch(error => console.log(error));
        setIsTimerActive(false);
    };
    const  addMessage = (author : string, message : string) => {
        const myMessageItem : ConferenceMessageItem = new
ConferenceMessageItem(true, author, message);
        if(currentChatText){
            setMessagesList(previous => [...previous, myMessageItem]);
            socketConnection?.invoke("Send", id,
`${currentConferenceData?.client.firstName}
${currentConferenceData?.client.lastName}`, currentChatText);
            setCurrentChatText(null);
        }
    }
    return (
        <>
            <div className="w-full h-[50em] flex bg-gray-200">
                <div className="h-[45em] pt-[1em] pl-[2em]">
                    <h1 className="text-2xl text-
center">{`${currentConferenceData?.doctor.user.firstName ?? ""}
${currentConferenceData?.doctor.user.lastName ?? ""}`}</h1>
                    {doctorJoined ?
                        <video
                            id={videoElemId}
                            autoPlay
                            playsInline
                            muted
                            className="h-[45em]"
                            ref={myVideo}>
                        </video>
                        :
                        <div className="h-[45em] w-[50em]">
                            <CircularProgress className="mx-auto mt-[10em] ml-
[25em]" color="secondary" />
                        </div>
                    }
                </div>
                    {currentUserName &&
                        <div className="h-[38em] pt-[5px] pl-[6.5em]">
                            <p className="text-2xl text-center">
                                {startEndTimeDifference &&
                                    <span>Seconds left <Timer
isActive={isTimerActive} timerData={startEndTimeDifference}
callback={onTimeUpCallback}/></span>
                                }
                            </p>
```

```
                            <p className="text-center text-2xl">Chat with your
doctor</p>
                            <ul className='w-[400px] h-full  overflow-y-scroll'>
                                <ConferenceMessageItems allMessages =
{messagesList}/>
                            </ul>
                            <ConferenceTextInput
                                AddMessageCallback = {(author,message) =>
addMessage(author,message)}
                                InputChangeEvent = {handleInputChange}
                                Author = {currentUserName}
                                Message = {currentChatText ?? ""}/>
                            <button
                                onClick={() => {
                                    socketConnection?.invoke("Leave",id);
                                    navigate(PrjRoutes.confPage);
                                }}
                                className="ml-[35%] w-[200px] h-[30px] bg-red-400
mt-[10px]">
                                Leave conference
                            </button>
                        </div>
                    }
            </div>
        </>
    );
};
```

Додаток В

Код серверної частини системи

```
namespace KYH_Online_Clinic.IdentityService.Controllers;

[ApiController]
public abstract class BaseController : ControllerBase
{
                                        protected readonly IMediator _mediator;
                                        public BaseController(IMediator mediator)
                                        {
                                            _mediator = mediator;
                                        }

                                        public async Task<T>
SendRequest<T>(IRequest<T> command, CancellationToken cancellationToken =
default)
                                        {
                                            return await _mediator.Send(command,
cancellationToken);
                                        }
}
namespace KYH_Online_Clinic.MainApiService.Controllers;

[ApiController]
[Route("[controller]/")]
public class Client_IllnessOrganController : BaseController
{
                                        public
Client_IllnessOrganController(IMediator mediator) : base(mediator)
                                        {
                                        }

                                        #region Commands

                                        [HttpPost("save")]
                                        public async Task<IActionResult>
SaveClientIllnessOrgan([FromBody] SaveClientIllnessOrganCommand cio,
CancellationToken cancellationToken = default)
                                        {
                                            var response = await
base.SendRequest(cio, cancellationToken);

                                            if (response.IsSucess)
                                            {
                                                  return Ok(response);
                                            }

                                            return BadRequest(response);
                                        }

                                        [HttpDelete("remove")]
                                        public async Task<IActionResult>
RemoveClientIllnessOrgan([FromQuery] RemoveClientIllnessOrganCommand cio,
CancellationToken cancellationToken = default)
                                        {
                                            var response = await
base.SendRequest(cio, cancellationToken);

                                            if (response.IsSucess)
                                            {
                                                  return Ok(response);
```

```
                                        }

                                        return BadRequest(response);
                                    }
                                    #endregion

                                    #region Queries

                                    [HttpGet("get")]
                                    public async Task<IActionResult>
GetCio([FromQuery] GetAllCioQuery cio, CancellationToken cancellationToken =
default)
                                    {
                                        var response = await
base.SendRequest(cio, cancellationToken);

                                        if (response.IsSucess)
                                        {
                                              return Ok(response);
                                        }

                                        return BadRequest(response);
                                    }

                                    [HttpGet("get-all")]
                                    public async Task<IActionResult>
GetAllCio([FromQuery] GetAllCioWithoutPaginationQuery cio, CancellationToken
cancellationToken = default)
                                    {
                                        var response = await
base.SendRequest(cio, cancellationToken);

                                        if (response.IsSucess)
                                        {
                                              return Ok(response);
                                        }

                                        return BadRequest(response);
                                    }

                                    #endregion

}
namespace KYH_Online_Clinic.MainApiService.Controllers;

[ApiController]
[Route("[controller]/")]
public class ConsultationController : BaseController
{
                                    public ConsultationController(IMediator
mediator) : base(mediator)
                                    {
                                    }
                                    #region Commands
                                    [HttpPost("create-consultation")]
                                    public async Task<IActionResult>
CreateConsultation([FromBody] CreateConsultationCommand command,
CancellationToken cancellationToken = default)
                                    {
                                        var response = await
base.SendRequest(command, cancellationToken);

                                        if (response.IsSucess)
                                        {
                                              return Ok(response);
```

```
                                        }

                                        return BadRequest(response);
                                    }

                                    [HttpPost("pay")]
                                    public async Task<IActionResult>
Pay([FromBody] PayConsultationCommand command, CancellationToken
cancellationToken = default)
                                    {
                                        var response = await
base.SendRequest(command, cancellationToken);

                                        if (response.IsSucess)
                                        {
                                              return Ok(response);
                                        }

                                        return BadRequest(response);
                                    }

                                    [HttpPost("cancel")]
                                    public async Task<IActionResult>
Cancel([FromBody] CancelConsultationCommand command, CancellationToken
cancellationToken = default)
                                    {
                                        var response = await
base.SendRequest(command, cancellationToken);

                                        if (response.IsSucess)
                                        {
                                              return Ok(response);
                                        }

                                        return BadRequest(response);
                                    }

                                    [HttpPost("make-in-progress")]
                                    public async Task<IActionResult>
MakeInProgress([FromBody] MakeInProgressCommand command, CancellationToken
cancellationToken = default)
                                    {
                                        var response = await
base.SendRequest(command, cancellationToken);

                                        if (response.IsSucess)
                                        {
                                              return Ok(response);
                                        }

                                        return BadRequest(response);
                                    }

                                    [HttpPost("make-finished")]
                                    public async Task<IActionResult>
MakeFinished([FromBody] MakeFinishedCommand command, CancellationToken
cancellationToken = default)
                                    {
                                        var response = await
base.SendRequest(command, cancellationToken);

                                        if (response.IsSucess)
                                        {
                                              return Ok(response);
                                        }
```

```
                                        return BadRequest(response);
                                    }
                                    #endregion

                                    #region Queries
                                    [HttpGet("get")]
                                    public async Task<IActionResult>
GetConsultations([FromQuery] GetConsultationsQuery query, CancellationToken
cancellationToken = default)
                                    {
                                        var response = await
base.SendRequest(query, cancellationToken);

                                        if (response.IsSucess)
                                        {
                                              return Ok(response);
                                        }

                                        return BadRequest(response);
                                    }

                                    [HttpGet("get-for-doctor")]
                                    public async Task<IActionResult>
GetConsultations([FromQuery] GetConsultationsForDoctorQuery query,
CancellationToken cancellationToken = default)
                                    {
                                        var response = await
base.SendRequest(query, cancellationToken);

                                        if (response.IsSucess)
                                        {
                                              return Ok(response);
                                        }

                                        return BadRequest(response);
                                    }

                                    [HttpGet("get-one")]
                                    public async Task<IActionResult>
GetConsultation([FromQuery] GetConsultationQuery query, CancellationToken
cancellationToken = default)
                                    {
                                        var response = await
base.SendRequest(query, cancellationToken);

                                        if (response.IsSucess)
                                        {
                                              return Ok(response);
                                        }

                                        return BadRequest(response);
                                    }

                                    [HttpGet("get-one-for-doctor")]
                                    public async Task<IActionResult>
GetConsultationForDoctor([FromQuery] GetConsultationForDoctorQuery query,
CancellationToken cancellationToken = default)
                                    {
                                        var response = await
base.SendRequest(query, cancellationToken);

                                        if (response.IsSucess)
                                        {
                                              return Ok(response);
```

```
                                        }

                                        return BadRequest(response);
                                    }

                                    [HttpGet("check-access")]
                                    public async Task<IActionResult>
CheckAccess([FromQuery] CheckAccessQuery query, CancellationToken
cancellationToken = default)
                                    {
                                        var response = await
base.SendRequest(query, cancellationToken);

                                        return Ok(response);
                                    }
                                    #endregion
}

namespace KYH_Online_Clinic.Services.Handlers.Commands;

public class MakeFinishedCommandHandler : BaseHandler,
IRequestHandler<MakeFinishedCommand, Response>
{
                                    private readonly IConsultationRepository
_consultationRepository;
                                    private readonly IUserRepository
_userRepository;
                                    private readonly
IUserIdentityProfileProvider _identityProfileProvider;
                                    private readonly
IValidationBuilder<MakeFinishedCommand> _validationBuilder;

                                    public
MakeFinishedCommandHandler(IConsultationRepository consultationRepository,
IUserIdentityProfileProvider identityProfileProvider,
IValidationBuilder<MakeFinishedCommand> validationBuilder, IUserRepository
userRepository)
                                    {
                                        _consultationRepository =
consultationRepository;
                                        _identityProfileProvider =
identityProfileProvider;
                                        _validationBuilder = validationBuilder;
                                        _userRepository = userRepository;
                                    }
                                    public async Task<Response>
Handle(MakeFinishedCommand command, CancellationToken cancellationToken)
                                    {
                                        var currentUserLogin =
_identityProfileProvider.UserName;

                                        var neededConsultation = await
_consultationRepository.GetConsultation(command.ConsultationId,
cancellationToken);
                                        var allowLogins = new List<string?> {
neededConsultation?.Doctor_Login, neededConsultation?.Client_Login };

                                        var currentUser = await
_userRepository.GetUserByLogin(currentUserLogin, cancellationToken);

                                        var validationResult =
_validationBuilder
                                              .RegisterValidableObject(command)
                                              .RuleFor(x => neededConsultation
is not null, "Such consultation doesn't exist!")
```

```
                                                .RuleFor(x =>
allowLogins.Contains(currentUserLogin), EnglishTranslation.ForbiddenForThisUser)
                                                .RuleFor(x =>
neededConsultation!.Status == Consultation_Status.Finished ||
neededConsultation!.Status == Consultation_Status.InProggress, "Consultation
cannot be put to finished!")
                                                .Build();

                                          if (!validationResult.Item1)
                                          {
                                                return new Response
                                                {
                                                      IsFailed = true,
                                                      ResultMessage =
validationResult.Item2
                                                };
                                          }

                                          neededConsultation!.Status =
Consultation_Status.Finished;

                                          await _consultationRepository.Save();

                                          return new Response
                                          {
                                                IsSucess = true,
                                                ResultMessage = "Your
consultation was made finished successfully!"
                                          };
                                    }
}

namespace KYH_Online_Clinic.Services.Handlers.Commands;

public class PayDoctorForConsultationCommandHandler : BaseHandler,
IRequestHandler<PayDoctorForConsultationCommand, Response>
{
                                    private readonly IConsultationRepository
_consultationRepository;
                                    private readonly IUserRepository
_userRepository;
                                    private readonly
IUserIdentityProfileProvider _identityProfileProvider;
                                    private readonly
IValidationBuilder<PayDoctorForConsultationCommand> _validationBuilder;
                                    public
PayDoctorForConsultationCommandHandler(IConsultationRepository
consultationRepository, IUserIdentityProfileProvider identityProfileProvider,
IValidationBuilder<PayDoctorForConsultationCommand> validationBuilder,
IUserRepository userRepository)
                                    {
                                       _consultationRepository =
consultationRepository;
                                       _identityProfileProvider =
identityProfileProvider;
                                       _validationBuilder = validationBuilder;
                                       _userRepository = userRepository;
                                    }
                                    public async Task<Response>
Handle(PayDoctorForConsultationCommand command, CancellationToken
cancellationToken)
                                    {
                                       var currentUserLogin =
_identityProfileProvider.UserName;
```

```
                                        var neededConsultation = await
_consultationRepository.GetConsultation(command.ConsultationId,
cancellationToken);

                                        var currentUser = await
_userRepository.GetUserByLogin(currentUserLogin, cancellationToken);

                                        var validationResult =
_validationBuilder
                                              .RegisterValidableObject(command)
                                              .RuleFor(x => neededConsultation
is not null, "Such consultation doesn't exist!")
                                              .RuleFor(x =>
neededConsultation!.Status == Consultation_Status.Finished, "Money for this
consultation cannot be transferred")
                                              .RuleFor(x =>
neededConsultation!.EndDateTime < DateTime.UtcNow, "Consultation is not finished
to be cancelled")
                                              .RuleFor(x =>
neededConsultation!.Doctor_Login == currentUserLogin, "This doctor didn't conduct
this consultation")
                                              .RuleFor(x =>
neededConsultation!.IsPaid, "This consultation is not paid to be invoiced")
                                              .RuleFor(x =>
!neededConsultation!.IsInvoiced, "This consultation is already invoiced")
                                              .RuleFor(x => currentUser.Role ==
Role.Doctor , "Only doctor can get payment for this consultation")
                                              .Build();

                                        if (!validationResult.Item1)
                                        {
                                              return new Response
                                              {
                                                    IsFailed = true,
                                                    ResultMessage =
validationResult.Item2
                                              };
                                        }

                                        await using var transaction = await
_consultationRepository.CreateTransaction(cancellationToken);

                                        try
                                        {
                                              var consultationPrice =
neededConsultation!.TotalPrice;

                                              neededConsultation!.IsInvoiced =
true;

                                              currentUser.Balance +=
consultationPrice;

                                              await
transaction.CommitAsync(cancellationToken);
                                              await
_userRepository.Save(cancellationToken);
                                        }
                                        catch (Exception e)
                                        {
                                              await
transaction.RollbackAsync(cancellationToken);
                                              throw;
                                        }
```

```
                                        return new Response
                                        {
                                              IsSucess = true,
                                              ResultMessage = "Money for
consultation were transferred to the doctor successfully!"
                                        };
                                    }
}

namespace KYH_Online_Clinic.Services.Handlers.Commands;

public class TopUpBalanceCommandHandler : BaseHandler,
IRequestHandler<TopUpBalanceCommand, Response>
{
                                    private readonly IUserRepository
_userRepository;
                                    private readonly
IUserIdentityProfileProvider _identityProfileProvider;
                                    private readonly
IValidationBuilder<TopUpBalanceCommand> _validationBuilder;
                                    public
TopUpBalanceCommandHandler(IUserRepository userRepository,
IUserIdentityProfileProvider identityProfileProvider,
IValidationBuilder<TopUpBalanceCommand> validationBuilder)
                                    {
                                       _userRepository = userRepository;
                                       _identityProfileProvider =
identityProfileProvider;
                                       _validationBuilder = validationBuilder;
                                    }
                                    public async Task<Response>
Handle(TopUpBalanceCommand command, CancellationToken cancellationToken)
                                    {
                                       var currentUserLogin =
_identityProfileProvider.UserName;

                                       var validationResult =
_validationBuilder
                                              .RegisterValidableObject(command)
                                              .RuleFor(x => x.Amount > 0,
EnglishTranslation.IncorrectAmountToTopUpBalance)
                                              .Build();

                                       if (!validationResult.Item1)
                                       {
                                              return new Response
                                              {
                                                    IsFailed = true,
                                                    ResultMessage =
validationResult.Item2
                                              };
                                       }

                                       await
_userRepository.TopUpBalance(currentUserLogin, command.Amount,
cancellationToken);

                                       await
_userRepository.Save(cancellationToken);

                                       return new Response
                                       {
                                              IsSucess = true,
                                              ResultMessage = "Balance was
topped up successfully!"
```

```
                };
            }
        }
```