Вінницький національний технічний університет
Факультет інтелектуальних інформаційних технологій та автоматизації
Кафедра комп'ютерних наук

Бакалаврська дипломна робота на тему:

ПРОГРАМНИЙ МОДУЛЬ WEB-ДОДАТКА ОРГАНІЗАЦІЇ РОБОТИ МЕДИЧНИХ ЗАКЛАДІВ

Виконав: студент 4 курсу, групи 2КН-20б
спеціальності 122 – Комп'ютерні науки
(шифр і назва напряму підготовки, спеціальності)
Русначенко Б.В.
(прізвище та ініціали)

Керівник: д.т.н., проф. каф. КН
Іванчук Я. В.
(прізвище та ініціали)
«07» 06 2024 р.

Рецензент: д.т.н., проф., зав. каф. КСУ
Ковтун В. В.
(прізвище та ініціали)

«07» 06 2024 р.

Допущено до захисту
Зав. кафедри проф., д.т.н. Яровий А.А.
«07» 06 2024 р.

Вінниця ВНТУ - 2024 рік

Вінницький національний технічний університет
Факультет інтелектуальних інформаційних технологій та автоматизації
Кафедра комп'ютерних наук
Рівень вищої освіти перший (бакалаврський)
Галузь знань – 12 Інформаційні технології
Спеціальність – 122 Комп'ютерні науки
Освітньо - професійна програма – Комп'ютерні науки

ЗАТВЕРДЖУЮ
Завідувач кафедри КН
проф., д.т.н. Яровий А.А.
«07» 03 2024р

ЗАВДАННЯ
НА БАКАЛАВРСЬКУ ДИПЛОМНУ РОБОТУ СТУДЕНТУ
Русначенку Богдану Валентиновичу

1. Тема роботи: Програмний модуль WEB-додатка організації роботи медичних закладів.
Керівник роботи: д.т.н., професор Іванчук Я. В. затверджені наказом вищого навчального закладу "11" 03 2024 року №80
2. Строк подання студентом роботи 07 05 2024 року
3. Вихідні дані до роботи : API для взаємодії WEB-додатку та сервера - 1 од.; WEB-додаток для організації роботи медичних закладів - 1 од.; дані про співробітників та пацієнтів; кількість сутностей в базі даних - 5 од.; мова програмування JS, TS.
4. Зміст текстової частини (перелік питань, які потрібно розробити): вступ; аналіз предметної області, огляд існуючих систем для організації роботи медичних систем; проектування програмних модулів WEB-додатку організації роботи медичних закладів; програмна реалізація модулів WEB-додатку організації роботи медичних закладів, тестування роботи програмного модулю WEB-додатку організації роботи медичних закладів, висновки, спислк використаної літератури, додатки.
5. Перелік ілюстративного матеріалу (з точним зазначенням обов'язкових креслень): схема алгоритму роботи взаємодії розроблених модулів; функції роботи WEB-додатку; UML-діаграма класів WEB-додатку, UML-діаграма послідовностей до модуля створення операцій; UML-діаграма класів DynamoDB; типова схема архітектури front-end частини WEB-додатку.

6. Консультанти розділів проекту (роботи)

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
1-3	Іванчук Я. В., д.т.н., проф.. каф. КН		

7. Дата видачі завдання 07.03.2024

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів бакалаврської дипломної роботи	Термін виконання		Примітка
		Початок	Закінчення	
1	Аналіз предметної області, існуючих систем для організації роботи медичних систем	06.05.24	10.05.24	
2	Проектування програмного модулю WEB-додатка організації роботи медичних закладів	11.05.24	17.05.24	
3	Програмна реалізація модулю WEB-додатка організації роботи медичних закладів	18.05.24	24.05.24	
4	Розробка інструкції користувача.	25.05.24	30.05.24	
5	Оформлення пояснювальної записки	31.05.24	04.06.24	
6	Оформлення матеріалів до захисту БДР.	05.06.24	06.06.24	

Студент _____ Русначенко Б.В.
(підпис) (прізвище та ініціали)

Керівник роботи _____ Іванчук Я. В.
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

Бакалаврська дипломна робота складається 84 сторінок формату А4, на яких є 22 рисунки, 1 таблиця, список використаної літератури містить 32 найменування.

Ця бакалаврська робота присвячена розробці та використанню WEB-додатку для організації роботи медичних закладів. Головною метою роботи є розробка простого, зручного та ефективного WEB-додатку, що зможе забезпечити медичний заклад простою та водночас ефективною системою керування персоналу, медичних втручань та їх аналіз. Було розроблено структурну схему WEB-додатку для організації роботи медичних закладів. Для розробки front-end частини додатку було використано React та його сателіти. Для back-end використовуються AWS Lambda Functions, що забезпечують безперебійну та швидку роботу.

Результатом роботи є зручний та надійний програмний модуль, що дозволяє ефективно виконувати поставленні задачі організації роботи медичних закладів.

Ключові слова: медичний заклад, WEB-додаток, організація роботи.

ABSTRACT

The bachelor thesis consists of 84 pages of A4 format, on which there are 22 figures, 1 table, the list of used sources contains 32 titles.

This bachelor's thesis is devoted to the development and use of a WEB application for the organization of the work of medical institutions. The main goal of the work is the development of a simple, convenient and effective WEB application that can provide a medical institution with a simple and at the same time effective system for managing personnel, medical interventions and their analysis. A structural diagram of a WEB application for organizing the work of medical institutions was developed. React and its satellites were used to develop the front-end part of the application. AWS Lambda Functions are used for the back-end, which ensure smooth and fast work.

The result of the work is a convenient and reliable software module that allows you to effectively perform the tasks of organizing the work of medical institutions.

Keywords: medical institution, WEB application, work organization.

ЗМІСТ

ВСТУП

Актуальність. Система охорони здоров'я є однією з найважливіших галузей, від ефективного функціонування якої залежить добробут населення та розвиток суспільства в цілому. Медичні заклади постають ключовими організаціями, які забезпечують надання якісних медичних послуг, проведення профілактичних заходів та лікування хвороб. Проте, сучасні виклики, такі як зростаючий попит на медичну допомогу, необхідність оптимізації використання ресурсів та підвищення ефективності роботи, вимагають впровадження новітніх інформаційних технологій та автоматизованих систем управління [1].

Одним з перспективних напрямків підвищення ефективності діяльності медичних закладів є розробка та впровадження WEB-додатків, які дозволяють організувати та оптимізувати широкий спектр процесів та операцій. WEB-додатки забезпечують централізоване управління даними, зручний доступ до інформації з будь-якого пристрою, підключеного до мережі Інтернет, та можливість інтеграції з іншими системами і сервісами [2].

Програмний модуль WEB-додатку організації роботи медичних закладів є комплексним рішенням, що охоплює ключові аспекти діяльності закладу, такі як:

1. Управління потоком пацієнтів та записом на прийом до лікарів різних спеціалізацій.
2. Ведення електронних медичних карт пацієнтів, історій хвороб та результатів діагностичних обстежень.
3. Моніторинг наявності лікарських препаратів, медичних матеріалів та обладнання на складі закладу.
4. Планування та організація робочих графіків медичного персоналу з урахуванням навантаження та специфіки роботи.
5. Аналіз діяльності закладу, формування звітності та статистичних даних для прийняття управлінських рішень [3].

Ефективний програмний модуль WEB-додатку повинен забезпечувати зручний та інтуїтивно зрозумілий інтерфейс для взаємодії користувачів (медичного персоналу, адміністраторів та пацієнтів) з системою, гарантувати безпеку та конфіденційність конфіденційних даних, а також бути масштабованим та адаптивним до змін у вимогах та специфіці роботи медичного закладу. Крім того, важливим аспектом є інтеграція з іншими інформаційними системами та сервісами, такими як системи електронних платежів, обміну медичними даними тощо.

Впровадження програмного модуля WEB-додатку організації роботи медичних закладів може значно підвищити ефективність використання ресурсів, скоротити час на виконання рутинних операцій, поліпшити координацію між різними підрозділами та спеціалістами, а також забезпечити більш якісне та своєчасне обслуговування пацієнтів. Це, в свою чергу, сприятиме підвищенню рівня надання медичних послуг та загальному покращенню системи охорони здоров'я.

Метою дослідження є підвищення рівня автоматизації обліку процесу роботи надання медичних послуг за допомогою розробки доступного програмного модуля WEB-додатку для організації роботи медичних закладів, що дозволить підвищити рівень охорони здоров’я.

Об’єктом дослідження є процес автоматизованого обліку роботи медичних закладів за допомогою WEB-додатку.

Предметом дослідження є програмні засоби WEB-додатку для організації роботи медичних закладів.

Задачі подальшого дослідження:

1. Провести аналіз предметної області, методів та засобів організації роботи медичних закладів.
2. Розробити структуру програмного забезпечення для організації роботи медичних закладів.
3. Розробити методи для взаємодії із базою даних системи організації роботи медичних закладів.

4. Розробити алгоритми для процесів обліку і керування роботою для організації роботи медичних закладів.

5. Виконати програмну реалізацію системи організації роботи медичних закладів з урахуванням вимог галузі охорони здоров'я.

6. Провести тестування роботи програмного модуля WEB-додатку для організації роботи медичних закладів

7. Розробити інструкцію користувача.

Апробація роботи. Робота апробувалася на конференції «LIII Всеукраїнська науково-технічна конференція факультету інтелектуальних інформаційних технологій та автоматизації (2024)» [4].

Публікації. Електронні тези на тему «Перспективи розробки програмного модуля ВЕБ-додатку організації роботи медичних закладів» [4].

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Аналіз проблеми впровадження WEB-додатку організації роботи медичних закладів

Впровадження WEB-додатку для організації роботи медичних закладів є важливим кроком для покращення ефективності та якості надання медичних послуг. Однак цей процес також пов'язаний з певними проблемами та викликами, які потрібно ретельно проаналізувати та вирішити.

Однією з головних проблем є інтеграція нового WEB-додатку з існуючими системами медичних закладів для збереження та обробки даних пацієнтів, призначень, медичних записів тощо. Ця інтеграція може бути складною та потребувати ретельного планування й координації, щоб уникнути втрати цінної інформації або дублювання даних. Необхідно забезпечити безперебійний обмін даними між різними системами та мінімізувати будь-які затримки або перебої в роботі.

Безпека та конфіденційність даних є надзвичайно важливими аспектами у медичній сфері. WEB-додаток повинен забезпечувати найвищий рівень захисту даних пацієнтів від несанкціонованого доступу, витоку або зловживань. Це вимагає використання сучасних методів шифрування, аутентифікації, контролю доступу та регулярного оновлення систем безпеки. Крім того, необхідно дотримуватися суворих правил захисту конфіденційності даних відповідно до місцевих законів та нормативних актів, таких як стандарти HIPAA та GDPR [5].

Ще однією проблемою є необхідність навчання медичного персоналу, адміністраторів та іншого залученого персоналу щодо використання нового WEB-додатку. Це може бути складним завданням, особливо для великих медичних установ з різним рівнем технічних знань співробітників. Розробка ефективних навчальних програм, наочних інструкцій та постійної підтримки

користувачів є вкрай важливими для забезпечення правильного використання системи та мінімізації помилок.

Масштабованість та продуктивність WEB-додатку також є ключовими вимогами, оскільки кількість пацієнтів, обсяг даних та навантаження на систему можуть значно зрости з часом. Архітектура додатку та використана інфраструктура повинні бути спроектовані з урахуванням майбутнього зростання, щоб уникнути сповільнення роботи або обмежень у функціональності.

Зручність використання та доступність інтерфейсу WEB-додатку для користувачів різного віку та рівня технічних навичок є важливим фактором для успішного впровадження системи. Незручний або складний інтерфейс може відштовхнути користувачів та знизити ефективність роботи. Тому необхідно забезпечити інтуїтивно зрозумілий та зручний дизайн, який враховує потреби різних груп користувачів.

Після впровадження WEB-додатку необхідно забезпечити належну технічну підтримку, вирішення проблем та регулярне оновлення програмного забезпечення для усунення вразливостей у безпеці, виправлення помилок та додавання нових функцій. Це вимагає наявності кваліфікованої команди розробників, адміністраторів та служби підтримки користувачів.

Також проблемою є значні витрати та необхідність ретельного планування бюджету на розробку, впровадження та підтримку WEB-додатку. Недостатнє фінансування може призвести до компромісів у функціональності, безпеці, продуктивності або масштабованості системи, що може зашкодити загальній ефективності та задоволеності користувачів.

Отже, впровадження WEB-додатку для організації роботи медичних закладів є складним процесом, який вимагає ретельного планування, аналізу та вирішення численних проблем, пов'язаних з інтеграцією, безпекою, навчанням персоналу, масштабованістю, зручністю використання, технічною підтримкою, витратами та дотриманням правових вимог. Лише ретельний аналіз цих проблем та вжиття необхідних заходів дозволить забезпечити

успішне впровадження WEB-додатку та отримати максимальну віддачу від інвестицій у нову систему, підвищити ефективність, якість та безпеку надання медичних послуг.

1.2 Принципи побудови систем організації роботи медичних закладів

Системи організації роботи медичних закладів мають свої унікальні особливостей, які відрізняють їх від інших типів систем. Ці особливості зумовлені специфікою галузі охорони здоров'я, необхідністю забезпечення високої якості медичних послуг, безпеки пацієнтів та ефективного управління ресурсами. Розглянемо детальніше особливості побудови та функціонування таких систем.

Однією з особливостей систем організації роботи медичних закладів є необхідність інтеграції з різними медичними пристроями та інформаційними системами. Вони повинні мати можливість безперешкодно взаємодіяти з електронними медичними картками (ЕМК), системами візуалізації (PACS), лабораторними інформаційними системами, системами електронного призначення ліків, а також з різноманітними діагностичними та лікувальними пристроями, такими як апарати УЗД, рентгенівські апарати, апарати МРТ тощо, відповідно до стандартів інтероперабельності HL7 [6].

Інтеграція з такими системами та пристроями дозволяє створити єдиний інформаційний простір, де всі дані про пацієнта, включаючи його медичну історію, результати обстежень, призначення та лікування, доступні в режимі реального часу. Це значно підвищує ефективність та якість медичного обслуговування, оскільки лікарі мають доступ до повної та актуальної інформації про стан здоров'я пацієнта, що дозволяє приймати більш обґрунтовані клінічні рішення.

Крім того, інтеграція з медичними пристроями дозволяє автоматизувати збір та передачу даних, зменшуючи ризик помилок та заощаджуючи час

медичного персоналу. Наприклад, результати лабораторних аналізів можуть автоматично передаватися в ЕМК пацієнта, а показники життєво важливих функцій, отримані з моніторингових пристроїв, можуть відображатися в режимі реального часу в системі організації роботи медичного закладу.

Іншою важливою особливістю систем організації роботи медичних закладів є необхідність забезпечення високого рівня безпеки та конфіденційності даних пацієнтів. Медичні дані є однією з найбільш чутливих категорій персональних даних, і будь-які порушення їх конфіденційності можуть мати серйозні наслідки для пацієнтів, включаючи дискримінацію, втрату репутації та навіть загрозу життю.

Тому такі системи повинні використовувати найсучасніші методи захисту інформації, такі як шифрування даних при передачі та зберіганні, багатофакторну аутентифікацію користувачів, суворий контроль доступу на основі ролей та регулярні оновлення безпеки для усунення можливих вразливостей.

Системи організації роботи медичних закладів повинні враховувати різноманітність користувачів та їхніх потреб. Користувачами таких систем можуть бути лікарі різних спеціальностей, медсестри, адміністративний персонал, пацієнти та їхні родичі. Кожна з цих груп має свої специфічні вимоги до функціональності та інтерфейсу системи.

Наприклад, лікарям потрібен швидкий та зручний доступ до медичних записів пацієнтів, можливість призначення обстежень та лікування, а також інструменти для прийняття клінічних рішень, такі як системи підтримки прийняття рішень (СППР) та бази знань. Медсестрам важливо мати можливість відстежувати виконання призначень, вести записи про спостереження за пацієнтами та взаємодіяти з іншими членами медичної команди.

З іншого боку, пацієнтам та їхнім родичам потрібен зручний доступ до інформації про стан здоров'я, можливість запису на прийом до лікаря,

отримання результатів обстежень та доступ до освітніх матеріалів про здоровий спосіб життя та профілактику захворювань.

Тому системи організації роботи медичних закладів повинні мати гнучкий та адаптивний користувацький інтерфейс, який може бути налаштований відповідно до потреб та преференцій кожної групи користувачів. Це може включати створення окремих модулів або портальних рішень для лікарів, медсестер, адміністраторів та пацієнтів, які забезпечують оптимальний досвід користування та підвищують ефективність роботи з системою.

Також масштабованість та продуктивність є важливими особливостями систем організації роботи медичних закладів. Зі зростанням обсягів медичних даних, збільшенням кількості користувачів та підвищенням складності медичних процесів, ці системи повинні бути здатними ефективно обробляти великі обсяги інформації та забезпечувати швидкий доступ до неї в будь-який час.

Для досягнення високої продуктивності системи повинні використовувати потужні сервери та оптимізовані бази даних, які можуть швидко обробляти запити та надавати необхідну інформацію. Крім того, застосування технологій розподілених обчислень, таких як хмарні сервіси та розподілені бази даних, дозволяє розподілити навантаження та забезпечити безперебійну роботу системи навіть в умовах пікових навантажень.

Масштабованість системи передбачає можливість легкого розширення її функціональності та обчислювальних потужностей зі зростанням потреб медичного закладу. Це може включати додавання нових модулів, інтеграцію з новими пристроями та системами, а також збільшення кількості користувачів без втрати продуктивності. Для забезпечення масштабованості системи повинні бути побудовані на основі модульної архітектури та використовувати стандартизовані протоколи обміну даними.

Галузь охорони здоров'я є однією з найбільш динамічних та інноваційних, з постійною появою нових технологій, методів лікування та

нормативних вимог. Тому системи організації роботи медичних закладів повинні бути орієнтовані на постійне вдосконалення та оновлення, щоб відповідати сучасним викликам та потребам галузі.

Це передбачає регулярне оновлення програмного забезпечення, впровадження нових функціональних можливостей та інтеграцію з передовими технологіями, такими як штучний інтелект, машинне навчання та аналітика великих даних. Ці технології дозволяють автоматизувати рутинні завдання, підтримувати прийняття клінічних рішень та виявляти приховані закономірності в медичних даних, що може значно покращити якість медичного обслуговування та оптимізувати використання ресурсів.

Крім того, системи організації роботи медичних закладів повинні бути гнучкими та адаптивними, здатними швидко реагувати на зміни в галузевих стандартах та нормативних вимогах. Це може включати оновлення протоколів безпеки, впровадження нових стандартів обміну даними та забезпечення відповідності новим законодавчим вимогам щодо захисту персональних даних та якості медичного обслуговування, як рекомендовано у блозі HIMSS про кібербезпеку [7].

Безперервне вдосконалення та оновлення систем організації роботи медичних закладів дозволяє медичним установам залишатися на передовій медичних інновацій, надавати пацієнтам найсучасніші методи діагностики та лікування, а також забезпечувати високу ефективність та якість медичного обслуговування.

Отже, особливості побудови та функціонування систем організації роботи медичних закладів зумовлені специфікою галузі охорони здоров'я та необхідністю забезпечення високої якості медичних послуг, безпеки пацієнтів та ефективного управління ресурсами. Ці системи повинні забезпечувати інтеграцію з широким спектром медичних пристроїв та інформаційних систем, гарантувати безпеку та конфіденційність даних пацієнтів, адаптуватися до потреб різних груп користувачів, демонструвати високу

масштабованість та продуктивність, а також бути орієнтованими на безперервне вдосконалення та оновлення.

Врахування цих особливостей при розробці та впровадженні систем організації роботи медичних закладів дозволяє створити потужні та надійні інструменти для управління медичними процесами, підвищення якості медичного обслуговування та оптимізації використання ресурсів. Такі системи стають невід'ємною частиною сучасної медицини, забезпечуючи безперервність та координацію медичної допомоги на всіх етапах взаємодії пацієнта з медичним закладом, від профілактики та діагностики до лікування та реабілітації.

1.3 Аналітичний огляд відомих систем організації роботи медичних закладів

На сьогодні кількість різних CRM систем дуже велика. Обираючи собі систему для керування медичною установою, лікарі повинні чітко розуміти свої потреби, адже від цього буде залежати кінцева вартість підключення продукту.

У цьому розділі розглянуто найбільші та найпопулярніші системи в світі: Practice Fusion EHR, athenahealth, Epic Systems, Cerner та eClinicalWorks.

Practice Fusion EHR є хмарною CRM-системою, що пропонує базову версію безкоштовно, роблячи її привабливою для невеликих клінік та приватних практик з обмеженим бюджетом. Вона доступна з будь-якого пристрою з підключенням до Інтернету та має мобільні додатки для Android та iOS, що дозволяє лікарям переглядати дані пацієнтів зі смартфонів або планшетів. Система інтегрується з клінічними системами, такими як лабораторії, аптеки та системи візуалізації, а також має функцію електронного призначення рецептів [8].

З іншого боку, безкоштовна версія Practice Fusion EHR має обмежений функціонал, і для отримання додаткових функцій, таких як біллінг або

розширена звітність, потрібно платити за преміум-пакети. Деякі користувачі висловлювали занепокоєння щодо конфіденційності та безпеки даних пацієнтів у хмарі.

Також повідомлялося про проблеми з інтеграцією з певними системами або пристроями та недостатню технічну підтримку чи складність налаштування системи без додаткової допомоги. Крім того, Practice Fusion EHR не має певних функцій, таких як управління аптекою або інвентаризацією, які можуть бути важливими для деяких медичних закладів.

Користувацький інтерфейс застосунку Practice Fusion EHR зображено на рисунку 1.1.

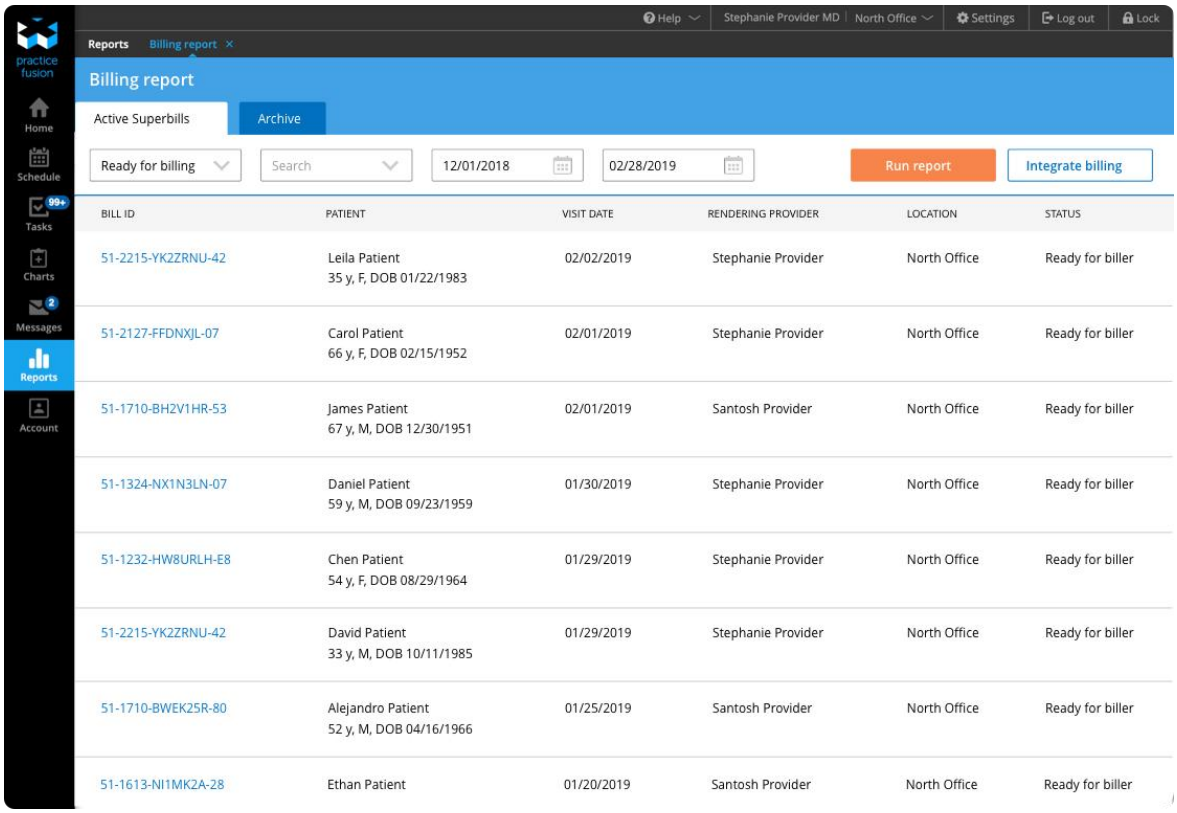

Рисунок 1.1 – Загальний вигляд інтерфейсного вікна програми Practice Fusion EHR

Athenahealth - потужна хмарна CRM, що пропонує широкий набір функцій для медичних закладів. Окрім електронних медичних записів, практичного менеджменту, управління пацієнтами та комунікації з третіми

сторонами, вона забезпечує аналітику для відстеження показників роботи та покращення результатів.

Athenahealth відома високим ступенем масштабованості та доступністю у хмарі, що робить її привабливою для закладів різних розмірів. Також вона інтегрується з багатьма лабораторними системами для полегшення обміну даними [9].

Однак деякі клініки можуть вважати athenahealth дорогою, особливо невеликі практики з обмеженим бюджетом. Крім того, процес впровадження може виявитися складним через необхідність навчання персоналу роботі з багатофункціональною системою.

Користувацький інтерфейс athenahealth зображено на рисунку 1.2.

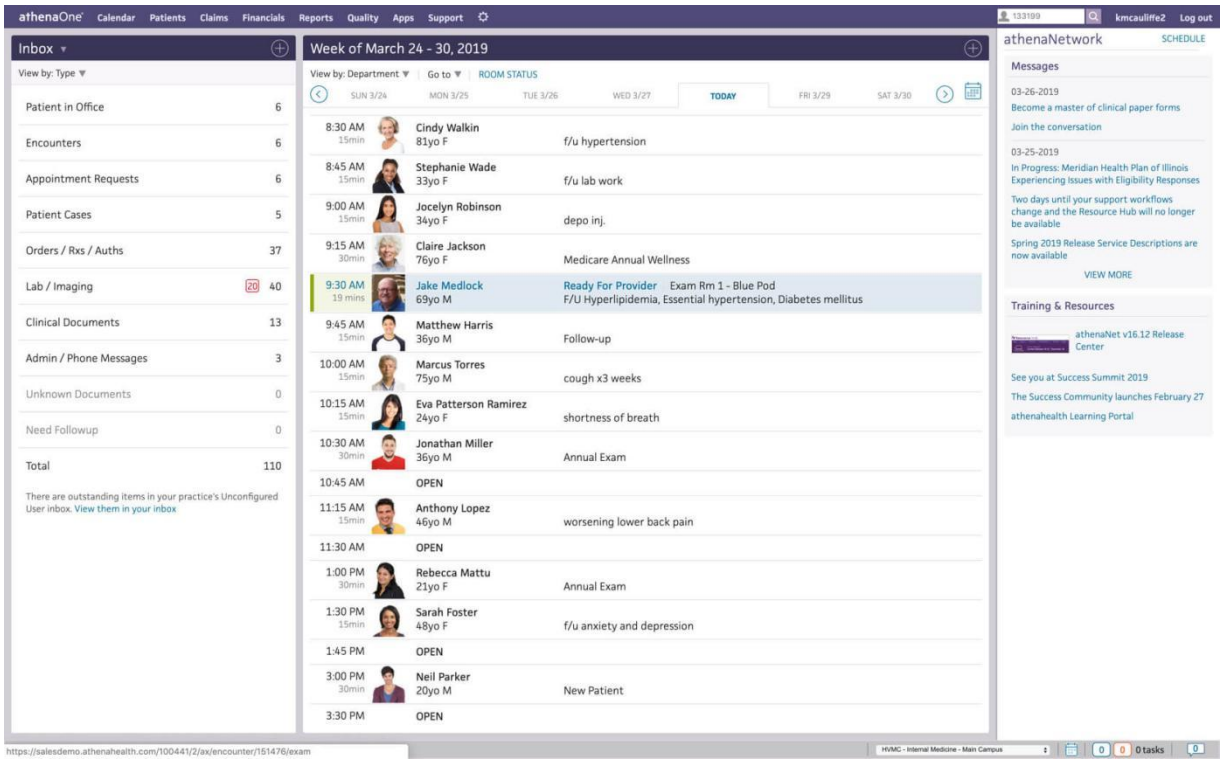

Рисунок 1.2 – Загальний вигляд інтерфейсного вікна програми athenahealth

Epic Systems вважається однією з найбільш комплексних та потужних CRM для великих медичних центрів і госпітальних систем. Вона покриває практично всі аспекти роботи - клінічну документацію, фінанси, реєстрацію пацієнтів, управління якістю, аналітику та інші [10].

Epic відома високими стандартами безпеки та можливістю інтеграції з різними системами охорони здоров'я завдяки відкритій архітектурі. Але через масштабність функціоналу, впровадження Epic може бути складним і дорогим процесом, що вимагає значних ресурсів для налаштування, інтеграції та ретельного навчання персоналу.

Користувацький інтерфейс Epic Systems зображено на рисунку 1.3.

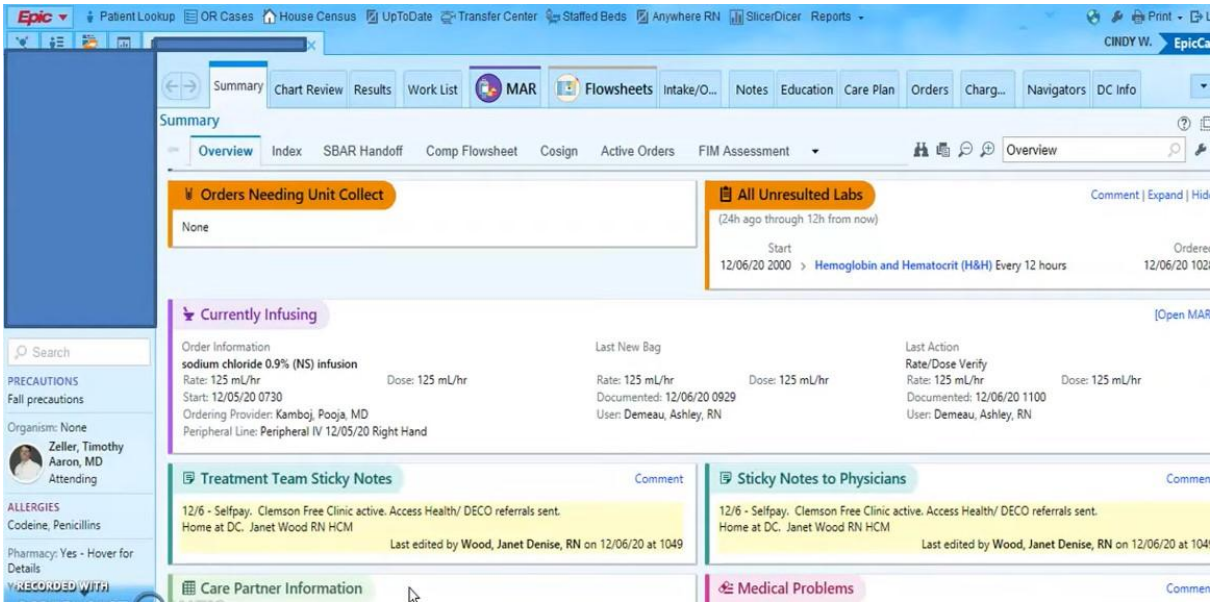

Рисунок 1.3 – Загальний вигляд інтерфейсного вікна програми Epic Systems

Cerner - визнаний лідер серед постачальників CRM-рішень для лікарень та великих медичних організацій. Її система забезпечує комплексне управління електронними медичними картками пацієнтів, фінансами, реєстрацією, аптечним відділенням і аналітикою показників. Cerner також підтримує мобільність та інтеграцію з діагностичними пристроями різних виробників [11].

Безпечна архітектура та відповідність галузевим стандартам роблять Cerner надійним вибором для великих закладів. Проте користувачі іноді скаржаться на громіздкість та складність інтерфейсу, що вимагає ретельного навчання персоналу для продуктивної роботи.

Користувацький інтерфейс застосунку Cerner зображено на рисунку 1.4.

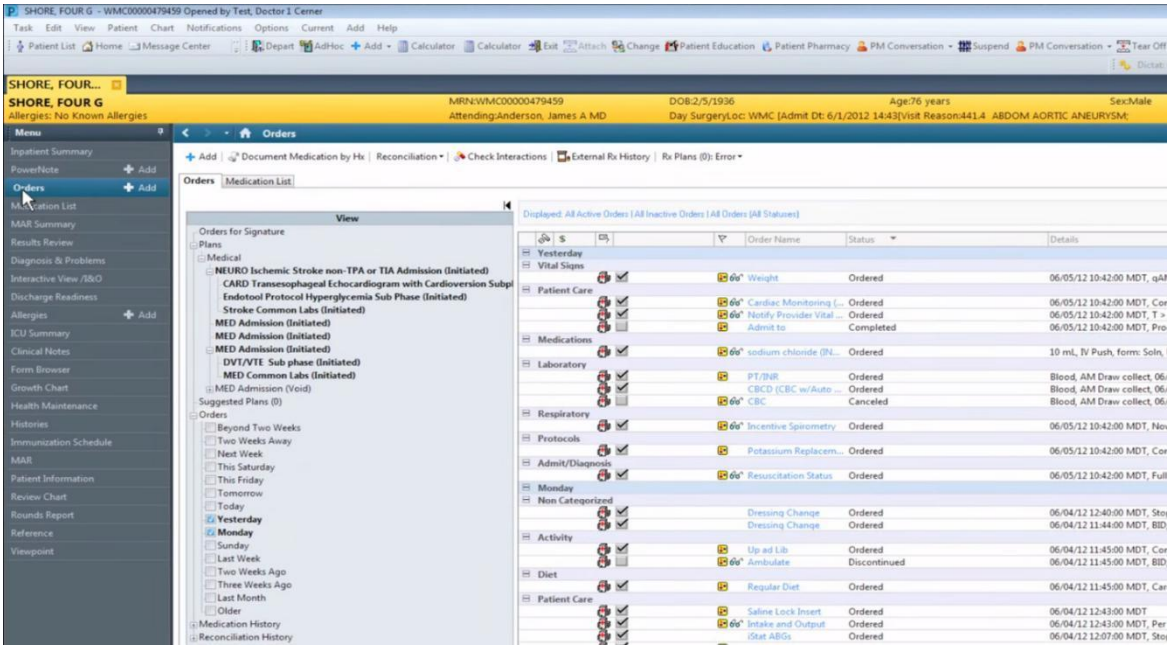

Рисунок 1.4 – Загальний вигляд інтерфейсного вікна програми Cerner

EClinicalWorks - хмарна CRM, розроблена спеціально для амбулаторних клінік та приватних медичних офісів. Вона пропонує відносно простий та інтуїтивний інтерфейс для ведення електронних карток пацієнтів, керування рецептами, біллінгу й генерування звітів. eClinicalWorks також включає функції телемедицини [12].

Завдяки хмарній моделі та орієнтації на менші практики, ця система є більш доступною ціною та простішою у впровадженні порівняно з масштабними CRM. Однак її функціональність може виявитися обмеженою для великих медичних установ із більш складними та специфічними вимогами.

Користувацький інтерфейс застосунку eClinicalWorks зображено на рисунку 1.5.

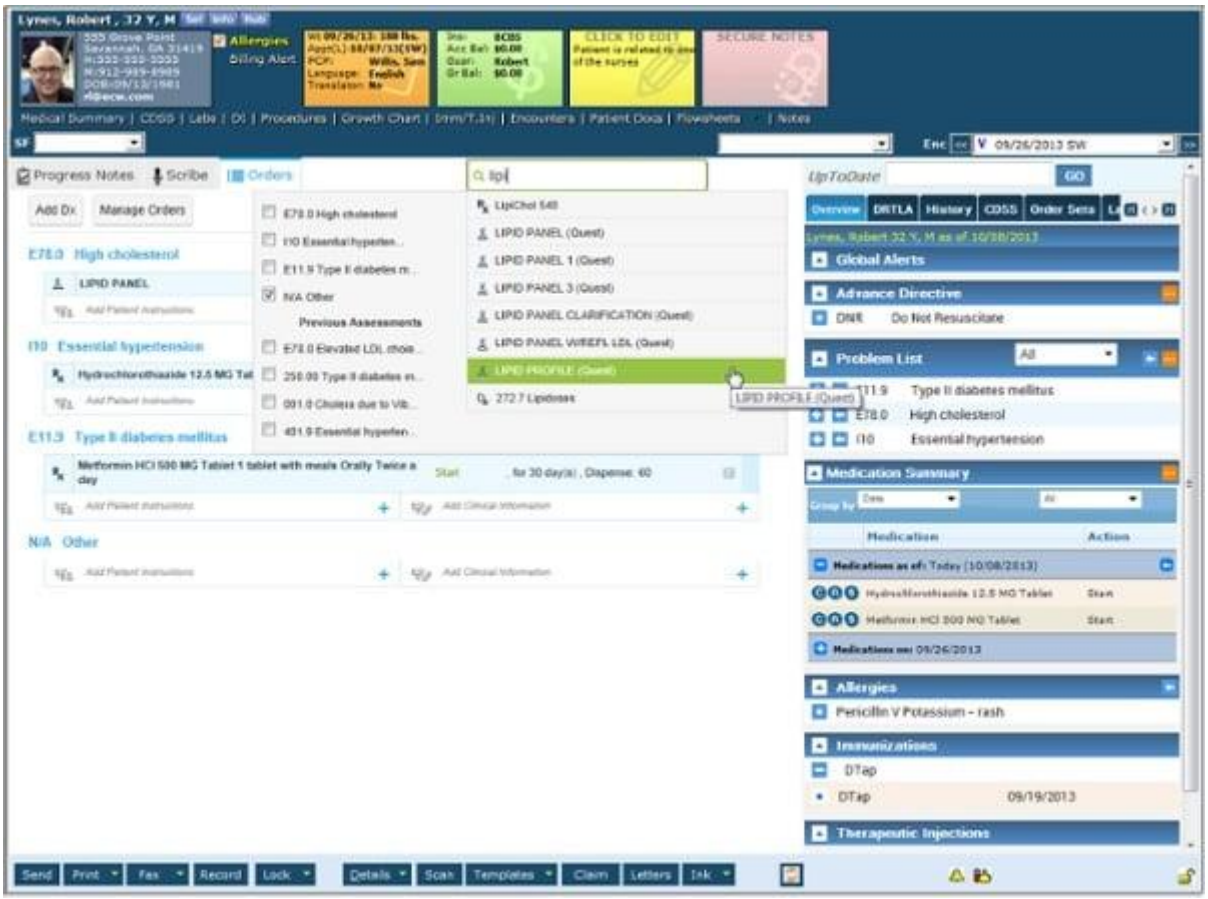

Рисунок 1.5 – Загальний вигляд інтерфейсного вікна програми eClinicalWorks

Отже, на основі проведеної порівняльної характеристики сучасних систем для організації роботи медичних засобів можна зробити висновок, що є потреба в розробці програмного забезпечення, яке об’єднуватиме усі необхідні функції та компенсує зазначені недоліки.

1.4 Висновок до розділу 1

Отже, впровадження WEB-додатку для організації роботи медичних закладів є складним процесом, який вимагає ретельного планування та врахування низки ключових проблем і принципів. Основними викликами є безпечна інтеграція з існуючими системами, забезпечення конфіденційності даних, навчання персоналу, масштабованість, зручність використання, технічна підтримка, витрати та дотримання правових норм.

На ринку присутні різні CRM-системи, такі як Practice Fusion, athenahealth, Epic Systems, Cerner та eClinicalWorks, які мають свої переваги та недоліки залежно від розміру закладу та специфічних потреб. Однак для успішного впровадження необхідна розробка комплексної системи, яка об'єднуватиме ключові функції та принципи, включно з модульністю, високим рівнем безпеки, інтероперабельністю, зручним інтерфейсом, масштабованістю та можливістю безперервного вдосконалення.

Дотримання принципів модульної побудови, забезпечення безпеки даних, інтеграції з іншими системами, зручності використання, продуктивності та постійного вдосконалення є критично важливим для успішної розробки та впровадження сучасної CRM-системи, здатної підвищити ефективність, якість та безпеку надання медичних послуг.

2 ПРОЕКТУВАННЯ ПРОГРАМНОГО МОДУЛЮ WEB-ДОДАТКУ ОРГАНІЗАЦІЇ РОБОТИ МЕДИЧНИХ ЗАКЛАДІВ

2.1 Розробка структури роботи WEB-додатку організації роботи медичних закладів

Перевіривши вхідні дані для WEB-додатку організації роботи медичних закладів, та дослідивши дані, що будуть зберігатися, розроблено структуру додатку, яка зображена на рисунку 2.1.

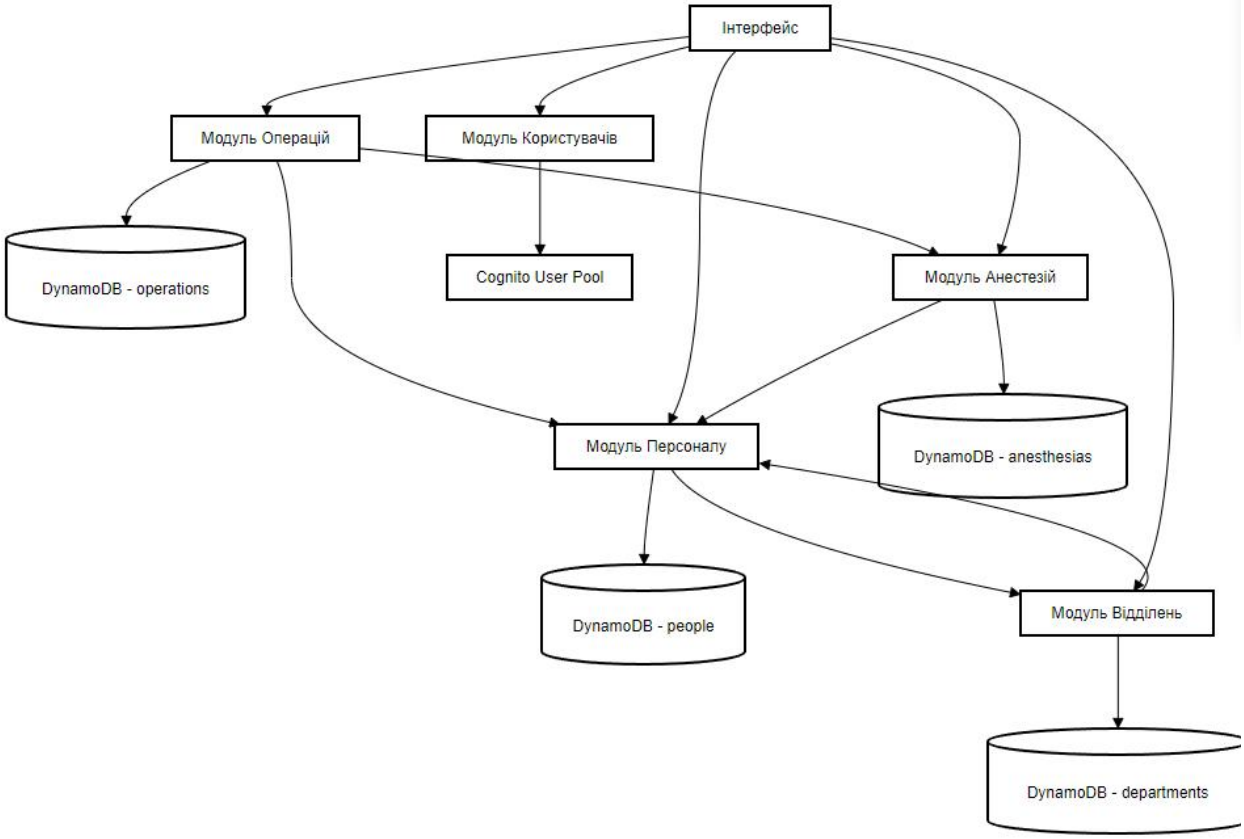

Рисунок 2.1 – Структурна схема WEB-додатку для організації роботи медичних закладів

Точкою входу в програму є користувацьке меню, з якого починається навігація програмою по інших її складових.

Модуль користувацького інтерфейсу є основним компонентом, що забезпечує взаємодію з користувачем та представлення інформації у зручній та інтуїтивно зрозумілій формі. Його головна роль полягає у створенні візуального середовища, в якому користувач може легко взаємодіяти з функціональністю додатку.

Цей модуль відповідає за розробку та реалізацію різноманітних графічних елементів, таких як меню, кнопки, поля вводу, списки, діалогові вікна та інші компоненти інтерфейсу користувача. Ці елементи дозволяють користувачеві здійснювати вибір опцій, вводити дані, ініціювати дії та отримувати зворотний зв'язок від додатку.

Модуль користувацького інтерфейсу забезпечує зручну навігацію, яка дозволяє користувачеві легко переміщатися між різними розділами та функціями додатку. Він організовує та представляє інформацію у логічній та привабливій формі, враховуючи принципи юзабіліті та зручності використання.

Крім того, цей модуль відповідає за обробку вхідних даних від користувача, валідацію введених даних та передачу їх до відповідних модулів додатку для подальшої обробки. Він також отримує результати операцій від інших модулів та відображає їх користувачеві у зрозумілому вигляді.

Він тісно взаємодіє з іншими модулями додатку, такими як модуль операцій, користувачів, медичного персоналу, медичною процедурою типу анестезія та відділень. Він забезпечує зв'язок між користувачем та бізнес-логікою додатку, дозволяючи користувачеві легко виконувати різноманітні дії та отримувати необхідну інформацію.

Ефективний та зручний користувацький інтерфейс є вирішальним фактором для успіху додатку, оскільки він забезпечує задоволеність користувачів та полегшує взаємодію з функціональністю додатку. Модуль користувацького інтерфейсу відіграє центральну роль у забезпеченні позитивного досвіду користувача та підвищенні продуктивності роботи з додатком.

Модуль управління операціями відповідає за управління даними про операції, що проводяться в медичному закладі. Він містить наступні функції:

1. CreateOperation: Ця функція дозволяє створювати нові записи про операції в таблиці OperationsTable. Вона приймає дані про операцію (ім'я пацієнта, тип операції, дата, відділення, залучений персонал тощо) і зберігає їх у таблиці.

2. GetAllOperations: Ця функція повертає список усіх операцій, збережених у таблиці OperationsTable. Вона може використовуватися для отримання загального переліку операцій або для відображення їх в інтерфейсі користувача.

3. GetOperationById: Ця функція дозволяє отримати деталі конкретної операції за її ідентифікатором (id). Вона може використовуватися для перегляду деталей операції або для оновлення/видалення існуючої операції.

4. GetOperationsByDepartment: Ця функція повертає список операцій, згрупованих за відділенням. Вона може бути корисною для аналізу навантаження на різні відділення або для фільтрування операцій за відділенням.

5. GetOperationsByPerson: Ця функція повертає список операцій, згрупованих за конкретним співробітником медичного персоналу. Вона може використовуватися для відстеження роботи окремих лікарів, медсестер або іншого персоналу.

6. DeleteOperation: Ця функція дозволяє видаляти існуючі записи про операції з таблиці OperationsTable за їхнім ідентифікатором.

7. UpdateOperation: Ця функція дозволяє оновлювати деталі існуючої операції в таблиці OperationsTable. Вона може використовуватися для внесення змін у дані про операцію, наприклад, при зміні дати або залученого персоналу.

Усі ці функції використовують таблицю OperationsTable для зберігання та отримання даних про операції. Вони також взаємодіють з іншими модулями, такими як person та department, для отримання додаткової

інформації про персонал та відділення, пов'язані з операціями. Головна робота модуля зображена на рисунку 2.2.

Рисунок 2.2 – Типова схема модуля управління хірургічними операціями

Модуль управління користувачами відповідає за управління користувачами системи, включаючи створення нових користувачів, перегляд інформації про користувачів та видалення користувачів. Він містить наступні функції:

1. GetUserInfo: Ця функція дозволяє отримати інформацію про поточного авторизованого користувача, такі як його ім'я, електронну пошту та інші деталі профілю.

2. CreateUser: Ця функція дозволяє створювати нових користувачів у пулі користувачів . Вона приймає дані про користувача (ім'я, електронну пошту, пароль тощо) і додає його до пулу користувачів. За замовчуванням, новостворені користувачі додаються до групи "Nurse".

3. GetAllUsers: Ця функція повертає список усіх користувачів, які є в пулі користувачів. Вона може використовуватися для перегляду та управління списком користувачів системи.

4. RemoveUser: Ця функція дозволяє видаляти існуючих користувачів з пулу користувачів за їхнім ідентифікатором.

Цей модуль тісно пов'язаний з конфігурацією пулом користувачів, яка використовується для управління автентифікацією та авторизацією користувачів у WEB-додатку. Він також взаємодіє з іншими модулями, такими як operation та person, для забезпечення належного доступу

користувачів до відповідних даних та функціональності. Головна робота модуля зображена на рисунку 2.3.

Рисунок 2.3 – Типова схема модуля управління користувачами

Модуль управління персоналом відповідає за управління даними про медичний персонал, який працює у закладі. Він містить наступні функції:

1. CreatePerson: Ця функція дозволяє створювати нові записи про співробітників у таблиці PeopleTable. Вона приймає дані про особу (ім'я, посада, відділення, контактна інформація тощо) і зберігає їх у таблиці.

2. GetPeople: Ця функція повертає список усіх співробітників, збережених у таблиці PeopleTable. Вона може використовуватися для отримання переліку всього персоналу або для відображення його в інтерфейсі користувача.

3. GetPeopleByDepartment: Ця функція повертає список співробітників, згрупованих за відділенням. Вона може бути корисною для перегляду персоналу, який працює в певному відділенні, або для фільтрування співробітників за відділенням.

4. UpdatePerson: Ця функція дозволяє оновлювати деталі існуючого запису про співробітника в таблиці PeopleTable. Вона може використовуватися для внесення змін у дані про співробітника, наприклад, при переведенні до іншого відділення або зміні контактної інформації.

5. DeletePerson: Ця функція дозволяє видаляти існуючі записи про співробітників з таблиці PeopleTable за їхнім ідентифікатором.

Модуль управління персоналом тісно взаємодіє з іншими модулями, такими як operation та department. Наприклад, при створенні нової операції

необхідно вказати співробітників, які беруть участь, тому дані про персонал повинні бути доступними. Крім того, при видаленні відділення потрібно оновити інформацію про співробітників, які працюють у цьому відділенні. Головна робота модуля зображена на рисунку 2.4.

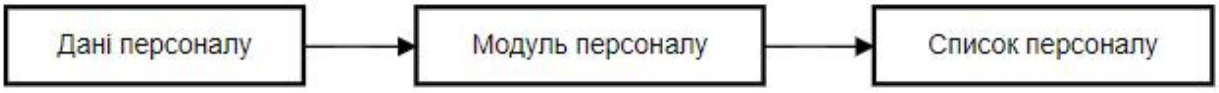

Рисунок 2.4 – Типова схема модуля управління медичним персоналом

Модуль управління анестезією відповідає за управління даними про анестезії, що проводяться під час операцій. Він містить наступні функції:

1. CreateAnesthesia: Ця функція дозволяє створювати нові записи про анестезії в таблиці AnesthesiasTable. Вона приймає дані про анестезію (тип анестезії, дози препаратів, час початку та закінчення, відповідальний анестезіолог тощо) і зберігає їх у таблиці. Ці дані можуть бути пов'язані з конкретною операцією з модуля operation.

2. GetAllAnesthesias: Ця функція повертає список усіх анестезій, збережених у таблиці AnesthesiasTable. Вона може використовуватися для отримання загального переліку анестезій або для відображення їх в інтерфейсі користувача.

3. DeleteAnesthesia: Ця функція дозволяє видаляти існуючі записи про анестезії з таблиці AnesthesiasTable за їхнім ідентифікатором.

Модуль управління анестезією взаємодіє з модулями operation та person, оскільки дані про анестезію пов'язані з конкретною операцією та включають інформацію про анестезіолога, який проводив анестезію. Він забезпечує зберігання та доступ до важливої інформації про анестезії, що проводилися під час операцій. Головна робота модуля зображена на рисунку 2.5.

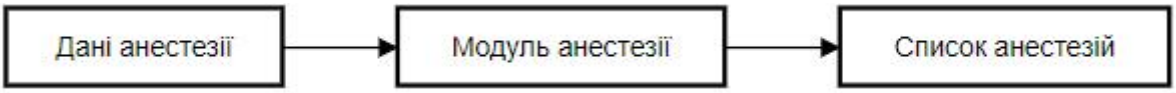

Рисунок 2.5 – Типова схема модуля управління медичною процедурою типу анестезія

Модуль управління відділеннями відповідає за управління даними про відділення медичного закладу. Він містить наступні функції:

1. CreateDepartment: Ця функція дозволяє створювати нові записи про відділення в таблиці DepartmentsTable. Вона приймає дані про відділення (назва, спеціалізація, керівник відділення тощо) і зберігає їх у таблиці.

2. GetAllDepartments: Ця функція повертає список усіх відділень, збережених у таблиці DepartmentsTable. Вона може використовуватися для отримання переліку всіх відділень або для відображення їх в інтерфейсі користувача.

3. DeleteDepartment: Ця функція дозволяє видаляти існуючі записи про відділення з таблиці DepartmentsTable за їхнім ідентифікатором. При видаленні відділення також оновлюються дані про співробітників, які працюють у цьому відділенні, в таблиці PeopleTable.

Модуль управління відділеннями тісно взаємодіє з модулями person та operation. Відділення пов'язані зі співробітниками, які в них працюють, і з операціями, що проводяться у відповідних відділеннях. Тому під час створення, оновлення чи видалення відділень необхідно відповідно оновлювати пов'язані дані в інших модулях. Головна робота модуля зображена на рисунку 2.6.

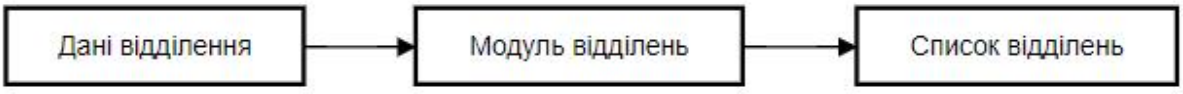

Рисунок 2.6 – Типова схема модуля управління структурними підрозділами

Наступний рисунок 2.7 показує детальніше зв’язки між модулями та їх функції.

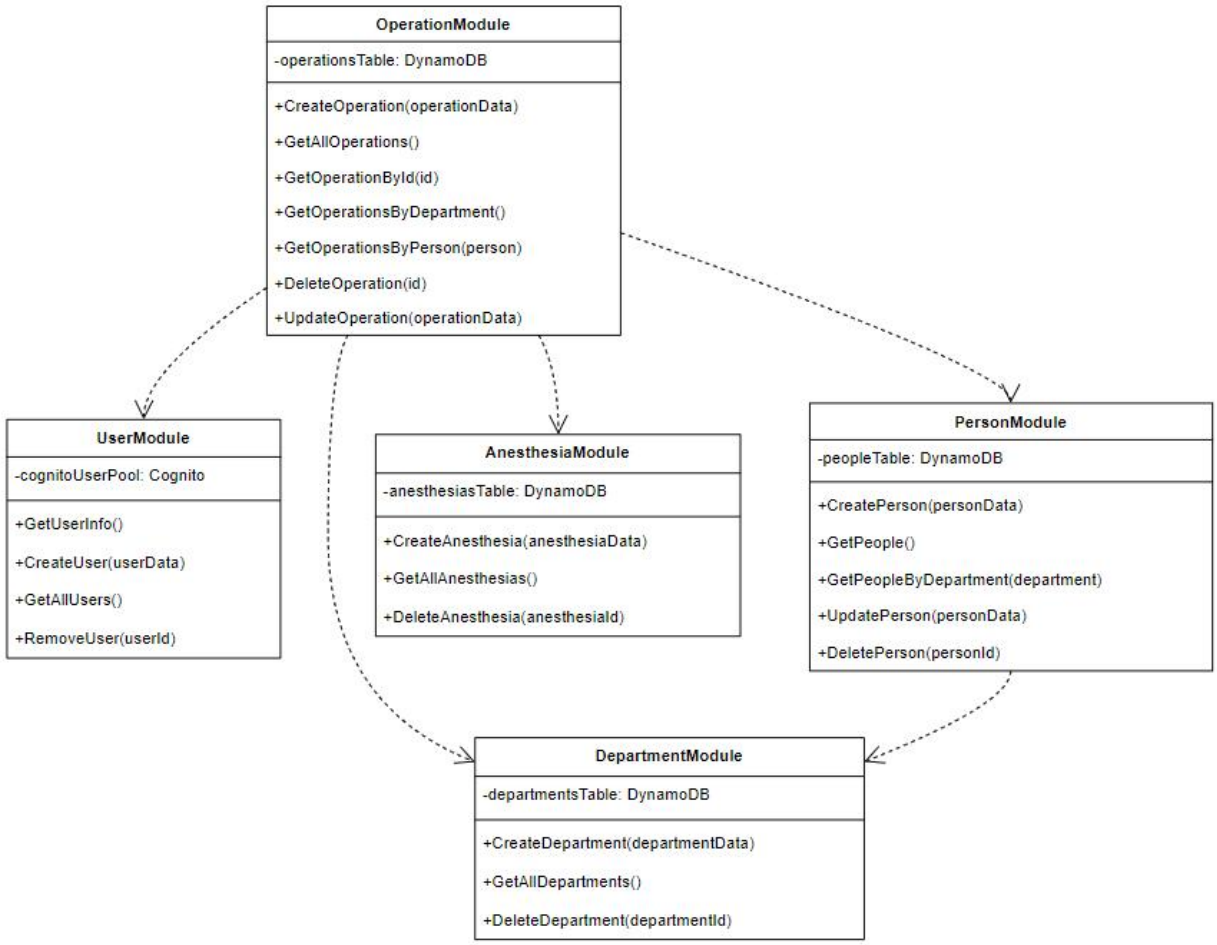

Рисунок 2.7 – UML-діаграма класів WEB-додатку

У цій діаграмі класів відображені основні класи (модулі) та їхні методи. Також показані зв'язки між модулями за допомогою ліній з пунктирними стрілками. Наприклад, OperationModule взаємодіє з PersonModule, DepartmentModule та AnesthesiaModule, оскільки операції пов'язані з персоналом, відділеннями та анестезіями.

Кожен модуль містить відповідну таблицю, з якою він взаємодіє для зберігання та отримання даних. Наприклад, OperationModule використовує таблицю operationsTable для роботи з даними про операції.

Ця діаграма класів демонструє загальну структуру системи та взаємозв'язки між її компонентами на високому рівні.

Розроблена структура WEB-додатку для організації роботи медичних закладів забезпечує ефективне управління різними аспектами діяльності медичного закладу. Модульний підхід дозволяє розділити функціональність на окремі компоненти, такі як управління операціями, користувачами, персоналом, анестезіями та відділеннями. Кожен модуль відповідає за свою специфічну область і надає відповідні функції для створення, отримання, оновлення та видалення даних.

Взаємодія між модулями забезпечує цілісність та узгодженість даних у системі. Наприклад, при створенні нової операції використовуються дані про персонал та відділення, а при видаленні відділення оновлюється інформація про співробітників, які працюють у цьому відділенні. Така тісна інтеграція дозволяє підтримувати актуальність та достовірність інформації в системі.

Така структура є гнучкою та розширюваною. У майбутньому можна додавати нові модулі та функціональність для задоволення зростаючих потреб медичного закладу. Крім того, завдяки модульності, можна легко модифікувати або замінювати окремі компоненти без впливу на інші частини системи.

Загалом, представлена структура WEB-додатку забезпечує надійну основу для ефективного управління роботою медичних закладів. Вона дозволяє автоматизувати та оптимізувати різні процеси, покращити якість обслуговування пацієнтів та полегшити роботу медичного персоналу.

2.2 Розробка алгоритму функціонування програмного модулю створення нового запису про операцію

Модуль CreateOperation відповідає за функціональність створення нової операції в системі. Цей модуль взаємодіє з користувачем, сховищем Redux та

API для збору необхідних даних, відправки запиту на створення операції та оновлення стану системи після успішного створення.

Діаграма, яка ілюструє роботу модуля CreateOperation, є діаграмою послідовності (рис. 2.8). Вона показує взаємодію між компонентами системи в часі, демонструючи послідовність дій та обмін повідомленнями між ними.

Ось детальний опис роботи модуля CreateOperation:

1. Користувач відкриває сторінку створення операції.
2. Сторінка створення операції запитує у сховища Redux дані, необхідні для заповнення форми створення операції, такі як список відділень, лікарів, доступних анестезій тощо.
3. Сховище Redux повертає запитані дані на сторінку створення операції.
4. Користувач заповнює форму створення операції, вводячи всі необхідні деталі, такі як дата операції, пацієнт, лікар, анестезія та інші параметри. Після заповнення форми користувач натискає кнопку "Створити".
5. Сторінка створення операції відправляє введені користувачем дані операції до сховища Redux.
6. Сховище Redux, у свою чергу, відправляє запит на створення нової операції до API, передаючи отримані дані.
7. API обробляє запит, створює нову операцію в базі даних та повертає результат (успішно створена операція або повідомлення про помилку) назад до сховища Redux.
8. Сховище Redux оновлює стан системи, додаючи нову операцію до списку операцій, і передає оновлений стан назад на сторінку створення операції.
9. Сторінка створення операції відображає повідомлення про успішне створення операції користувачеві.

Така послідовність дій забезпечує збір необхідних даних від користувача, відправку цих даних на сервер для створення нової операції та оновлення стану системи після успішного створення. Використання сховища

Redux дозволяє централізовано керувати станом додатку та забезпечує консистентність даних у всіх компонентах системи.

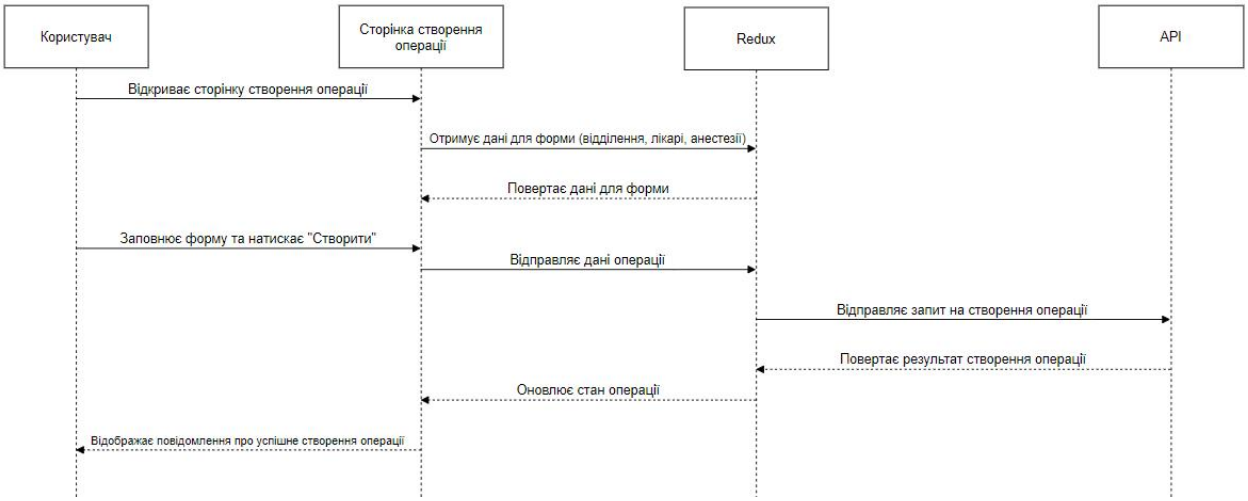

Рисунок 2.8 – UML-діаграма послідовностей до модуля створення операцій

Діаграма послідовності наочно демонструє цю взаємодію між компонентами в часі, показуючи послідовність подій та обмін даними між ними.

2.3 Проектування структури бази даних програмного модулю WEB-додатку організації роботи медичних закладів

При виборі бази даних для WEB-додатку організації роботи медичних закладів було розглянуто кілька популярних варіантів, включаючи реляційні бази даних (наприклад, MySQL, PostgreSQL) та NoSQL бази даних (наприклад, MongoDB, Cassandra).

Реляційні бази даних, такі як MySQL та PostgreSQL, мають жорстку схему даних та підтримують транзакції ACID (Atomicity, Consistency, Isolation, Durability). Вони добре підходять для структурованих даних та забезпечують цілісність даних за допомогою зовнішніх ключів та обмежень. Однак, вони можуть бути менш гнучкими при роботі зі змінними схемами даних та можуть мати обмеження масштабованості при збільшенні обсягів даних [13].

З іншого боку, NoSQL бази даних, такі як MongoDB та Cassandra, пропонують гнучкість схеми даних та горизонтальну масштабованість. Вони дозволяють зберігати неструктуровані та напівструктуровані дані, що може бути корисним для WEB-додатків з мінливими вимогами до даних. Однак, вони можуть не мати такої ж підтримки транзакцій ACID та цілісності даних, як реляційні бази даних [14].

Після аналізу вимог нашого WEB-додатку та врахування переваг і недоліків різних баз даних, ми вирішили використовувати Amazon DynamoDB. DynamoDB - це повністю керована NoSQL база даних, яка пропонує високу продуктивність, масштабованість та надійність.

Обґрунтування вибору Amazon DynamoDB:

1. Масштабованість: DynamoDB автоматично розподіляє дані та трафік на достатню кількість серверів, щоб обробляти вимоги до пропускної здатності та розміру зберігання. Це дозволяє нашому WEB-додатку масштабуватися горизонтально без необхідності ручного налаштування апаратного забезпечення [15].

2. Продуктивність: DynamoDB забезпечує стабільну продуктивність з низькою латентністю на будь-якому масштабі. Вона використовує SSD-накопичувачі та може обробляти мільйони запитів у секунду, що важливо для нашого WEB-додатку з потенційно великою кількістю одночасних користувачів [15].

3. Гнучкість схеми даних: DynamoDB дозволяє зберігати дані з гнучкою схемою, що означає, що кожен елемент може мати власний унікальний набір атрибутів. Це дає нам можливість адаптувати структуру даних до потреб нашого WEB-додатку без необхідності модифікувати схему бази даних [16].

4. Інтеграція з екосистемою AWS: DynamoDB легко інтегрується з іншими сервісами AWS, такими як AWS Lambda, Amazon S3, Amazon Cognito тощо. Це дозволяє нам створювати безсерверні архітектури та використовувати переваги інших сервісів AWS для нашого WEB-додатку [17].

5. Надійність та доступність: DynamoDB забезпечує вбудовану відмовостійкість та автоматичне резервне копіювання. Вона автоматично реплікує дані на кілька серверів у різних зонах доступності, що гарантує високу доступність та стійкість до збоїв [17].

6. Керованість: Як повністю керована служба, DynamoDB позбавляє нас необхідності управляти інфраструктурою бази даних, оновленнями програмного забезпечення та налаштуваннями масштабування. Це дозволяє нам зосередитися на розробці та функціональності нашого WEB-додатку, а не на управлінні базою даних [15].

Структура бази даних розроблена з урахуванням специфіки предметної області та функціональних вимог додатку. Вона складається з кількох таблиць, кожна з яких відповідає за зберігання даних для певного модуля системи.

Розглянемо детальніше кожну з таблиць:

1. OperationsTable - таблиця для зберігання даних про операції. Кожен запис у таблиці містить наступні атрибути:

- id (PartitionKey) - унікальний ідентифікатор операції (рядок, сформований за допомогою функції nanoid(8)).
- department - відділення, в якому проводиться операція.
- headOfDepartment - керівник відділення.
- tableNumber - номер операційного столу.
- dateStart - дата початку операції.
- dateEnd - дата завершення операції.
- actualDate - фактична дата проведення операції (пусте значення за замовчуванням).
- createdDate - дата створення запису про операцію.
- updatedDate - дата оновлення запису про операцію (пусте значення за замовчуванням).
- patient - ім'я пацієнта.
- medicalChamber - номер медичної палати.
- diagnosis - діагноз пацієнта.

- hospitalizationType - тип госпіталізації.
- attendingDoctor - лікар, який веде пацієнта.
- operatingDoctor - лікар, який проводить операцію.
- operation - назва операції.
- complexityOperation - складність операції.
- assistants - список асистентів (множина рядків).
- anesthesia - тип анестезії.
- nurse - ім'я медсестри.
- nurseId - ідентифікатор медсестри.
- operationStatus - статус операції (значення за замовчуванням).

2. AnesthesiasTable - таблиця для зберігання даних про анестезії. Кожен запис у таблиці містить наступні атрибути:
   - id (PartitionKey) - унікальний ідентифікатор анестезії (рядок, сформований за допомогою функції nanoid(8)).
   - name - назва анестезії.
3. DepartmentsTable - таблиця для зберігання даних про відділення. Кожен запис у таблиці містить наступні атрибути:
   - id (PartitionKey) - унікальний ідентифікатор відділення (рядок, сформований за допомогою функції nanoid(8)).
   - name - назва відділення.
4. PeopleTable - таблиця для зберігання даних про медичний персонал. Кожен запис у таблиці містить наступні атрибути:
   - id (PartitionKey) - унікальний ідентифікатор співробітника (рядок, сформований за допомогою функції nanoid(8)).
   - firstName - ім'я співробітника.
   - secondName - по-батькові співробітника.
   - thirdName - прізвище співробітника.
   - department - відділення, в якому працює співробітник.
   - departmentId - ідентифікатор відділення, в якому працює співробітник.

- isHeadOfDepartment - булеве значення, яке вказує, чи є співробітник керівником відділення.
- staffType - тип персоналу (лікар, медсестра тощо).

На рисунку 2.9 наведено UML-діаграму класів DynamoDB.

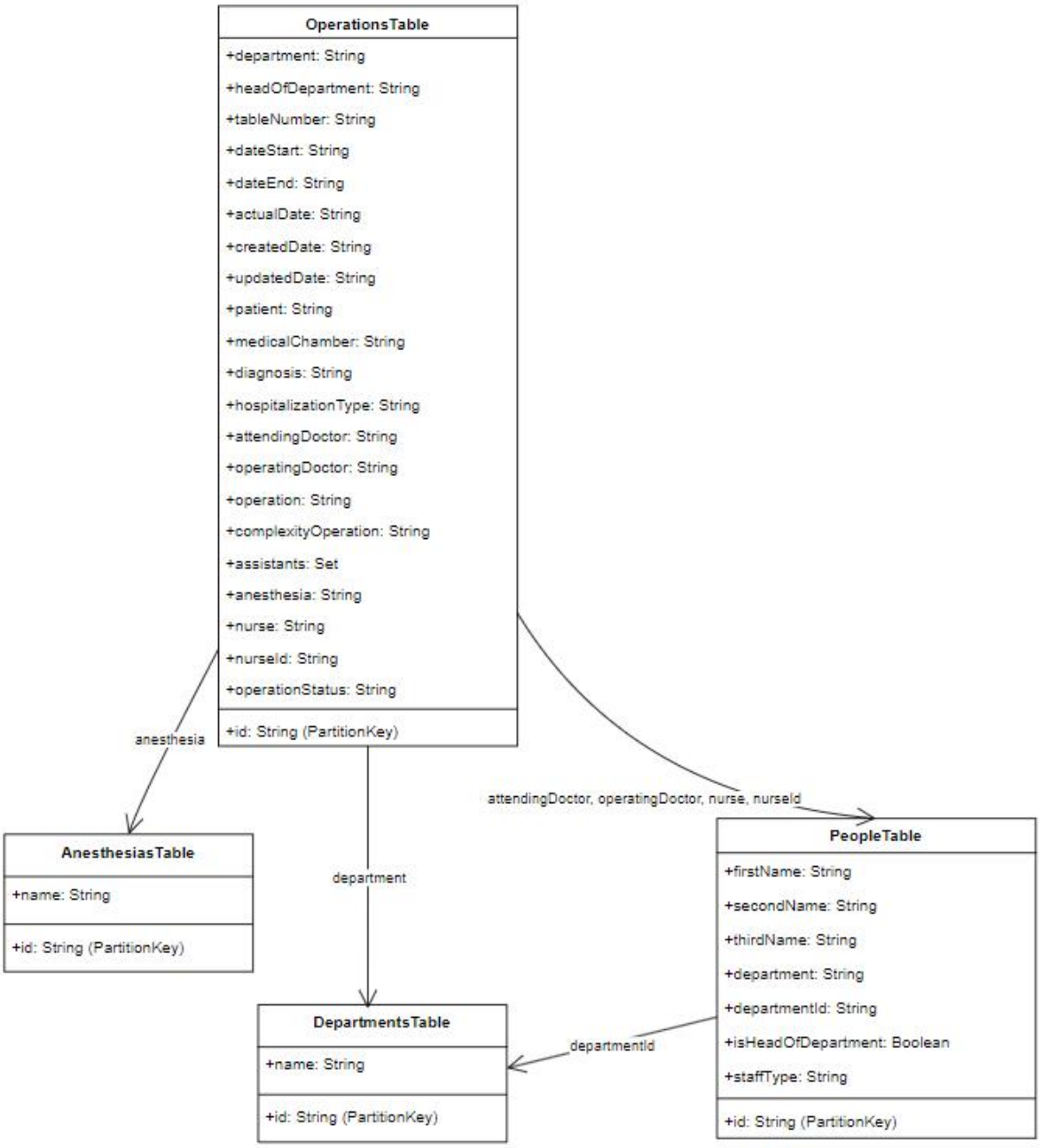

Рисунок 2.9 – UML-діаграма класів DynamoDB

Зв'язки між таблицями реалізуються за допомогою зовнішніх ключів (foreign keys). Наприклад, атрибут departmentId в таблиці PeopleTable посилається на id в таблиці DepartmentsTable, що дозволяє встановити зв'язок між співробітником та відділенням, в якому він працює. Аналогічно, атрибут anesthesiaId в таблиці OperationsTable посилається на id в таблиці AnesthesiasTable, встановлюючи зв'язок між операцією та використаною анестезією.

Для забезпечення ефективного доступу до даних, кожна таблиця має первинний ключ (primary key), який складається з партиціонного ключа (partition key) id. Це дозволяє швидко отримувати записи за їхнім унікальним ідентифікатором.

Для керування користувачами та їх автентифікацією використовується сервіс Amazon Cognito. Cognito дозволяє легко додавати можливості реєстрації, входу та керування користувачами до WEB-додатків [18]. Він забезпечує безпечне зберігання даних про користувачів, включаючи їхні облікові записи, паролі та додаткову інформацію.

З Amazon Cognito створюється User Pool - централізоване сховище для даних про користувачів. User Pool містить інформацію про зареєстрованих користувачів, їхні атрибути та групи, до яких вони належать. Кожен користувач має унікальний ідентифікатор (sub), який використовується для зв'язку з іншими даними в таблицях DynamoDB.

Для авторизації доступу до ресурсів та функцій WEB-додатку використовуються групи користувачів у Cognito. Наприклад, група "Admin" може мати повний доступ до всіх функцій системи, тоді як група "Nurse" матиме обмежений доступ лише до певних розділів та дій.

Інтеграція Amazon DynamoDB та Amazon Cognito забезпечує надійне та безпечне зберігання даних, а також гнучке керування доступом до них. DynamoDB відповідає за зберігання основних даних про операції, персонал, відділення та анестезії, тоді як Cognito забезпечує автентифікацію та авторизацію користувачів.

Запити до бази даних виконуються через відповідні API, які розроблені для кожного модуля системи. Ці API забезпечують абстракцію над базою даних та дозволяють виконувати операції створення, читання, оновлення та видалення (CRUD) записів у таблицях DynamoDB [19].

Нижче на рисунку 2.10 наведено приклад взаємодії з DynamoDB через API з офіційної документації [20].

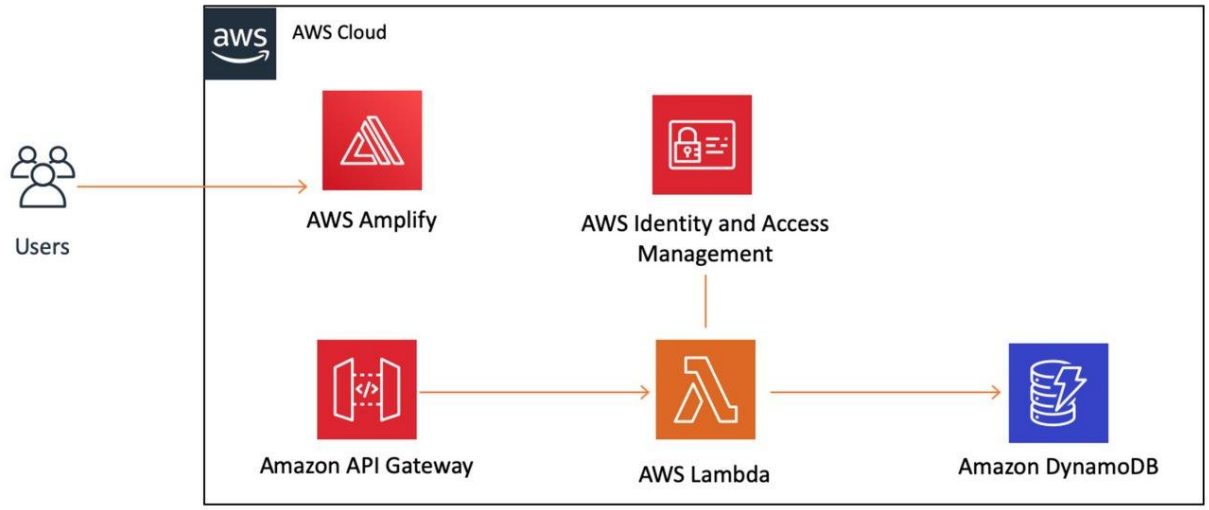

Рисунок 2.10 – Типова схема взаємодії з DynamoDB через API

Для забезпечення безпеки даних, всі запити до бази даних проходять через механізми автентифікації та авторизації. Користувачі повинні пройти автентифікацію через Cognito, щоб отримати доступ до захищених ресурсів. Крім того, на рівні API перевіряються права доступу користувача до конкретних дій та ресурсів на основі його групи та ролі.

Така структура бази даних та інтеграція з Amazon Cognito забезпечує ефективне зберігання, доступ та керування даними в WEB-додатку організації роботи медичних закладів. Вона дозволяє масштабувати систему відповідно до зростаючих потреб, забезпечує високу доступність та продуктивність, а також гарантує безпеку та конфіденційність даних.

2.4 Проєктування архітектури для розробки front-end частини WEB-додатку

При розробці WEB-додатків важливо обрати відповідну архітектуру, яка забезпечить масштабованість, можливість технічної підтримки, що підвищить ефективність розробки у цілому. У цьому розділі буде розглянуто декілька поширених архітектурних підходів для WEB-додатків та обґрунтуємо вибір найкращої архітектури для нашого проекту.

Монолітна архітектура - це традиційний підхід до розробки WEB-додатків, при якому вся функціональність реалізована в єдиному модулі або програмі. У цій архітектурі сервер обробляє всі запити, виконує бізнес-логіку та генерує HTML-сторінки для відправки клієнту. Монолітна архітектура проста в розробці та розгортанні, оскільки вся система знаходиться в одному місці [21].

Переваги монолітної архітектури:

- Простота розробки та розгортання.
- Легка комунікація між компонентами.
- Швидка продуктивність завдяки спільному використанню ресурсів.

Недоліки монолітної архітектури:

- Складність масштабування окремих компонентів.
- Жорстка зв'язаність компонентів, що ускладнює внесення змін.
- Обмежена гнучкість у виборі технологій.

Сервіс-орієнтована архітектура (SOA) розбиває додаток на набір незалежних сервісів, які взаємодіють між собою через мережеві протоколи, такі як HTTP або SOAP. Кожен сервіс відповідає за певну бізнес-функцію і може бути розроблений, розгорнутий та масштабований незалежно від інших сервісів. SOA забезпечує більшу гнучкість та можливість повторного використання сервісів у різних додатках [22].

Переваги SOA:

- Модульність та незалежність сервісів.

- Можливість повторного використання сервісів.
- Гнучкість у виборі технологій для кожного сервісу.

Недоліки SOA:

- Складність координації та управління великою кількістю сервісів.
- Збільшення мережевих затримок через комунікацію між сервісами.
- Необхідність узгодження контрактів та протоколів взаємодії між сервісами.

Мікросервісна архітектура є еволюцією SOA, при якій додаток розбивається на ще менші та незалежніші сервіси. Кожен мікросервіс відповідає за одну конкретну функцію і може бути розроблений, розгорнутий та масштабований окремо. Мікросервіси комунікують між собою через легковагі протоколи, такі як HTTP або messaging. Ця архітектура надає високу гнучкість, масштабованість та можливість використання різних технологій для кожного мікросервісу [23].

Переваги мікросервісної архітектури:

- Незалежність розробки, розгортання та масштабування кожного мікросервісу.
- Гнучкість у виборі технологій для кожного мікросервісу.
- Підвищена відмовостійкість завдяки ізоляції збоїв.

Недоліки мікросервісної архітектури:

- Складність управління та моніторингу великої кількості мікросервісів.
- Підвищена складність розробки через розподіленість системи.
- Необхідність реалізації міжсервісної комунікації та узгодженості даних.

Враховуючи вимоги та особливості нашого WEB-додатку для організації роботи медичних закладів, ми вирішили використовувати модульну архітектуру на основі компонентів з використанням бібліотеки React.

Ось переваги такого підходу:

1. Модульність та повторне використання: Розбиття додатку на незалежні компоненти дозволяє легко розділяти відповідальність та повторно використовувати компоненти в різних частинах додатку. Це покращує підтримуваність коду та прискорює розробку.

2. Інкапсуляція та незалежність: Кожен компонент інкапсулює свою власну логіку та стан, що робить їх незалежними та легкими для тестування та модифікації. Зміни в одному компоненті не впливають на інші, що зменшує ризик виникнення помилок.

3. Гнучкість та масштабованість: Модульна архітектура дозволяє легко додавати, змінювати або видаляти компоненти в міру розвитку додатку. Це забезпечує гнучкість та масштабованість, оскільки можна розширювати функціональність без впливу на існуючі компоненти.

4. Ефективність розробки: Використання бібліотеки React з її компонентним підходом та віртуальним DOM прискорює розробку та покращує продуктивність додатку. React дозволяє ефективно оновлювати інтерфейс користувача та мінімізує маніпуляції з реальним DOM.

Запропонована структура проекту відображає модульну архітектуру з чітким розділенням відповідальності:

- Директорія `components` містить реакт-компоненти, які інкапсулюють логіку та представлення для різних частин інтерфейсу користувача.
- Директорія `pages` містить компоненти сторінок, які комбінують компоненти з директорії `components` для створення повноцінних сторінок додатку.
- Директорія `redux` містить код для управління станом додатку за допомогою бібліотеки Redux, яка доповнює компонентну архітектуру та забезпечує централізоване сховище даних.
- Директорія `hooks` містить користувацькі хуки, які інкапсулюють логіку та стан, що використовуються в різних компонентах.

- Директорія `configure` містить конфігураційні файли та утиліти, що використовуються в додатку.

Така структура (рис. 2.11) забезпечує чітку організацію коду, розділення відповідальності та можливість повторного використання компонентів, що відповідає принципам модульної архітектури.

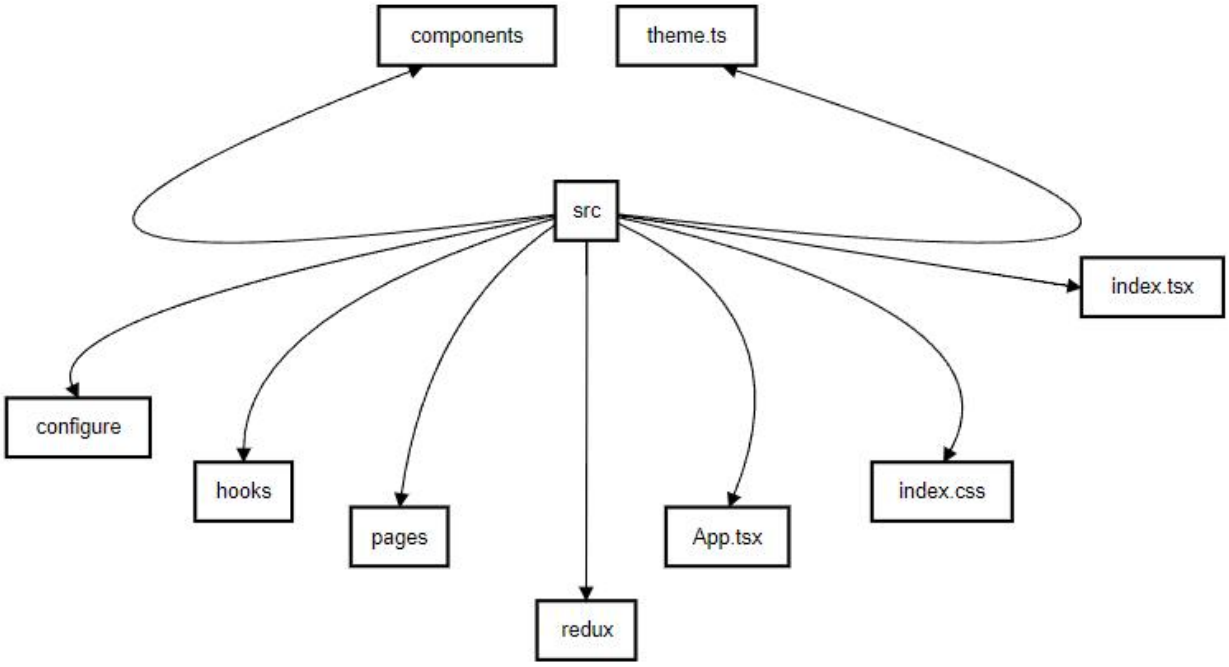

Рисунок 2.11 – Типова схема архітектури front-end частини WEB-додатку

Отже, обрана модульна архітектура на основі компонентів з використанням React є найбільш підходящою для нашого WEB-додатку організації роботи медичних закладів. Вона забезпечує гнучкість, масштабованість та ефективність розробки, дозволяючи створювати високоякісний та підтримуваний код.

2.5 Висновок до розділу 2

У другому розділі було детально розглянуто процес проектування WEB-додатку для організації роботи медичних закладів. Розроблена

структура додатку базується на модульному підході, що забезпечує ефективне управління різними аспектами діяльності медичного закладу. Для зберігання даних обрано Amazon DynamoDB, яка відповідає вимогам проекту завдяки своїй масштабованості, гнучкості та інтеграції з сервісами AWS.

Запропонована структура бази даних та інтеграція з сервісом автентифікації Amazon Cognito забезпечують високий рівень безпеки та конфіденційності даних, що є критично важливим для медичної галузі. Це дозволяє захистити чутливу інформацію про пацієнтів та персонал від несанкціонованого доступу.

При виборі архітектури для розробки front-end частини було обрано модульну архітектуру на основі компонентів з використанням бібліотеки React. Така архітектура забезпечує модульність, повторне використання компонентів, інкапсуляцію та ефективність розробки. Запропонована структура проекту відображає обрану архітектуру та гарантує чітку організацію коду.

Проведений аналіз та проектування створюють надійний фундамент для розробки WEB-додатку, який не лише автоматизує та оптимізує процеси в медичному закладі, але й відповідає високим стандартам якості, безпеки та продуктивності, що висуваються до програмного забезпечення в галузі охорони здоров'я.

3 ПРОГРАМНА РЕАЛІЗАЦІЯ МОДУЛЮ WEB-ДОДАТКУ ОРГАНІЗАЦІЇ РОБОТИ МЕДИЧНИХ ЗАКЛАДІВ

3.1 Вибір інструментів розробки front-end частини програмного модуля

При розробці Front-end частини WEB-додатку організації роботи медичних закладів важливо обрати відповідні інструменти та фреймворки, які забезпечать ефективність розробки, гнучкість та масштабованість інтерфейсу користувача. Тому важливо розглянути кілька інструментів, щоб вибрати найкращий.

Angular - це фреймворк з відкритим кодом, розроблений компанією Google. Він використовує мову TypeScript та підтримує концепцію односторінкових додатків (SPA). Angular має потужну екосистему та надає широкий набір інструментів для розробки, таких як Angular CLI, шаблони, директиви та сервіси. Він також має вбудовану систему впровадження залежностей та підтримує реактивне програмування за допомогою бібліотеки RxJS [24].

Vue.js - це прогресивний фреймворк для побудови інтерфейсів користувача. Він відрізняється своєю простотою та гнучкістю. Vue.js дозволяє поступово впроваджувати його в існуючі проекти, що робить його легким для вивчення та інтеграції. Він має реактивну систему прив'язки даних, що дозволяє ефективно оновлювати інтерфейс при зміні стану додатку. Vue.js також має офіційні інструменти, такі як Vue CLI та Vue Router, які спрощують процес розробки [25].

React - це бібліотека JavaScript з відкритим кодом, розроблена компанією Facebook. Вона використовує компонентний підхід до побудови інтерфейсів користувача. React дозволяє створювати багаторазові компоненти з власним станом та методами життєвого циклу. Він використовує віртуальний DOM для ефективного оновлення інтерфейсу та мінімізації

маніпуляцій з реальним DOM. React має велику екосистему та підтримується активною спільнотою розробників [26].

Зважаючи на велике ком’юніті React та його детально описану документацію, ми вирішили використовувати його для розробки Front-end нашого WEB-додатку. Ось основні причини цього вибору:

− Компонентна архітектура: React заохочує створення багаторазових компонентів, які інкапсулюють свій стан та логіку. Це дозволяє розбивати інтерфейс на менші, незалежні частини, що покращує читабельність, підтримуваність та тестованість коду. Компонентний підхід також дозволяє легко комбінувати та повторно використовувати компоненти в різних частинах додатку [27].

− Віртуальний DOM: React використовує концепцію віртуального DOM для ефективного оновлення інтерфейсу. Замість прямих маніпуляцій з реальним DOM, React створює віртуальне представлення DOM в пам'яті. При зміні стану компонента, React обчислює різницю між попереднім та новим віртуальним DOM (різницеве порівняння) і застосовує лише необхідні зміни до реального DOM. Це мінімізує кількість маніпуляцій з DOM і покращує продуктивність додатку [28].

− Багата екосистема: React має велику та активну спільноту розробників, яка постійно створює та підтримує різноманітні бібліотеки, інструменти та розширення. Це дозволяє легко знаходити готові рішення для поширених завдань, таких як маршрутизація, управління станом, візуалізація даних тощо. Наявність широкого вибору бібліотек та інструментів прискорює процес розробки та дозволяє зосередитися на бізнес-логіці додатку [29].

− Продуктивність розробки: React має простий та інтуїтивно зрозумілий API, що дозволяє швидко почати розробку. Він використовує декларативний підхід до опису інтерфейсу, де розробник описує, як компонент повинен виглядати на основі його стану. Це зменшує кількість коду, необхідного для побудови інтерфейсу, та покращує читабельність. React

також має потужні інструменти розробки, такі як React Developer Tools, які спрощують налагодження та профілювання додатку [30].

Для прискорення розробки інтерфейсу користувача та забезпечення узгодженості дизайну, можна використовувати бібліотеку Material-UI (MUI) разом з React. MUI - це популярна бібліотека компонентів React, яка реалізує принципи дизайну Material Design від Google [31].

Ось деякі переваги використання MUI:

- Готові компоненти: MUI надає широкий набір готових компонентів, таких як кнопки, форми, меню, таблиці, модальні вікна тощо. Ці компоненти мають привабливий та сучасний вигляд і можуть бути легко налаштовані відповідно до потреб проекту. Використання готових компонентів прискорює процес розробки та забезпечує узгодженість дизайну у всьому додатку.
- Адаптивний дизайн: Компоненти MUI розроблені з урахуванням адаптивного дизайну та можуть легко адаптуватися до різних розмірів екрану та пристроїв. Це дозволяє створювати WEB-додатки, які добре виглядають і функціонують на настільних комп'ютерах, планшетах та мобільних пристроях без необхідності написання додаткового коду для кожного типу пристрою.
- Теми та кастомізація: MUI підтримує концепцію тем, що дозволяє легко змінювати зовнішній вигляд компонентів відповідно до фірмового стилю або дизайну проекту. Розробники можуть створювати власні теми, змінювати кольори, шрифти, відступи тощо, щоб відповідати вимогам дизайну. Це забезпечує гнучкість та узгодженість у візуальному оформленні додатку.

Використання MUI разом з React дозволить нам швидко створити привабливий та функціональний інтерфейс користувача для нашого WEB-додатку організації роботи медичних закладів. Готові компоненти, адаптивний дизайн та можливості кастомізації зроблять процес розробки ефективнішим та забезпечать високу якість користувацького досвіду.

Також важливо обрати зручне середовище розробки. Після розгляду кількох популярних варіантів середовищ програмування, таких як Visual Studio Code, Sublime Text та WebStorm, ми обрали WebStorm [32] як основне середовище для розробки нашого WEB-додатку.

WebStorm - це потужне інтегроване середовище розробки (IDE) від компанії JetBrains, яке спеціально створене для розробки WEB-додатків з використанням JavaScript, TypeScript та інших WEB-технологій.

Основні переваги WebStorm включають:

- Інтелектуальне авто-доповнення коду.
- Вбудований відладчик.
- Інтеграцію з фреймворками та бібліотеками.
- Можливості рефакторингу коду.
- Інтеграцію з системами контролю версій.
- Можливості налаштування та розширення.

Завдяки своїм потужним функціям, зручності використання та інтеграції з популярними WEB-технологіями, WebStorm став нашим основним вибором для ефективної та якісної розробки WEB-додатку організації роботи медичних закладів.

3.2 Програмна реалізація модулів WEB-додатку

Головна функція є асинхронною та приймає параметр event, який містить дані запиту. Вона починається з отримання інформації про авторизованого користувача з контексту авторизації, зокрема nurseId, який буде використовуватися для зв'язку операції з медсестрою:

```
const info = event.requestContext.authorizer.claims;
const nurseId = info['sub'].
```

Далі створюється новий клієнт DynamoDB з використанням опцій ddbOptions. З тіла запиту event.body парситься JSON, щоб отримати дані про

операцію. Встановлюється значення за замовчуванням для статусу операції як "Created":

```
const client = new AWS.DynamoDB(ddbOptions);
const operationState = JSON.parse(event.body);
const defaultOperationStatus = "Created".
```

Формується об'єкт params, який містить параметри для збереження операції в таблиці DynamoDB. Кожен атрибут операції представлений у форматі { S: значення } для рядкових значень або { SS: значення } для масиву рядків:

```
const params: AWS.DynamoDB.Types.PutItemInput = {
   TableName: process.env.TABLE || table,
   Item: {
      id: { S: nanoid(8) },
      department: { S: department },
      headOfDepartment: { S: headOfDepartment },
      // ...
      assistants: { SS: assistants },
      anesthesia: { S: anesthesia },
      nurse: { S: nurse },
      nurseId: { S: nurseId },
      operationStatus: { S: defaultOperationStatus },
   },
}.
```

Після цього викликається метод client.putItem(params).promise() для збереження операції в таблиці DynamoDB. Це асинхронна операція, яка повертає проміс:

```
await client.putItem(params).promise().
```

У модулі редагування операцій спочатку створюється клієнт DynamoDB з відповідними опціями:

```
const client = new AWS.DynamoDB.DocumentClient(ddbOptions).
```

З тіла запиту парситься JSON, щоб отримати нові дані операції:

```
const operationState = JSON.parse(event.body).
```

Формується об'єкт params з параметрами для оновлення операції в таблиці DynamoDB:

```
const params: AWS.DynamoDB.Types.UpdateItemInput = {
TableName: process.env.TABLE || table,
Key: {id: event.pathParameters.id,
},
UpdateExpression: `SET department = :department, headOfDepartment = :headOfDepartment, tableNumber = :tableNumber, dateStart = :dateStart, dateEnd = :dateEnd, actualDate = :actualDate, patient = :patient, medicalChamber = :medicalChamber, diagnosis = :diagnosis, hospitalizationType = :hospitalizationType, attendingDoctor = :attendingDoctor, operatingDoctor = :operatingDoctor, operation = :operation, complexityOperation = :complexityOperation, assistants = :assistants, anesthesia = :anesthesia, nurse = :nurse, operationStatus = :operationStatus`,
ExpressionAttributeValues: {
":department": department,
":headOfDepartment": headOfDepartment,
":tableNumber": tableNumber,
":dateStart": dateStart,
":dateEnd": dateEnd,
":actualDate": actualDate,
":patient": patient,
":medicalChamber": medicalChamber,
":diagnosis": diagnosis,
":hospitalizationType": hospitalizationType,
":attendingDoctor": attendingDoctor,
":operatingDoctor": operatingDoctor,
":operation": operation,
":complexityOperation": complexityOperation,
":assistants": client.createSet(assistants) as any,
":anesthesia": anesthesia,
":nurse": nurse,
":operationStatus": operationStatus,},
```

```
ReturnValues: "UPDATED_OLD",}.
```

Викликається метод client.update(params).promise() для оновлення операції.

Модуль пошуку всіх користувачів, що мають доступ до системи, створює клієнт Cognito з відповідними опціями:

```
const CognitoIdentityServiceProvider = AWS.CognitoIdentityServiceProvider;
const client = new CognitoIdentityServiceProvider({ apiVersion: "2016-04-19" }).
```

Формується об'єкт params з ідентифікатором пулу користувачів (UserPoolId):

```
const params = {UserPoolId: process.env.UserPoolId}.
```

Викликається метод для отримання списку користувачів:

```
const userList = await client.listUsers(params as any).promise().
```

Модуль додавання нових користувачів також створює клієнт Cognito з відповідними опціями.

З тіла запиту парситься JSON, щоб отримати дані нового користувача:

```
const state = JSON.parse(event.body);
const { username, temporaryPassword, email, firstName, lastName } = state.
```

Формується об'єкт params з параметрами для створення нового користувача в Cognito:

```
const params = {
UserPoolId: process.env.UserPoolId,
Username: username,
TemporaryPassword: temporaryPassword,
UserAttributes: [
{ Name: "email", Value: email },
{ Name: "given_name", Value: firstName },
{ Name: "family_name", Value: lastName },
{ Name: "email_verified", Value: "true" },
],}.
```

Викликається метод для створення користувача:

```
await client.adminCreateUser(params as any).promise().
```

Формується об'єкт з параметрами для додавання користувача до групи:

```
const addUserToGroupParams = {
GroupName: process.env.GroupName,
UserPoolId: process.env.UserPoolId,
Username: username,
}.
```

Викликається метод для додавання користувача до групи:

```
await client.adminAddUserToGroup(addUserToGroupParams as any).promise().
```

Вхідна точка проєкту (App) є головним компонентом React-додатку, який відповідає за маршрутизацію та відображення відповідних компонентів залежно від поточного маршруту.

Спочатку компонент використовує хук useToast для створення функцій successToast та errorToast, які будуть використовуватися для відображення повідомлень про успіх та помилки відповідно:

```
const [successToast, errorToast] = useToast().
```

Далі, за допомогою хука useSelector, компонент отримує поточний стан Toast з Redux-сховища. Цей стан містить інформацію про повідомлення, які потрібно відобразити:

```
const toast = useSelector(selectToastState).
```

Компонент також використовує хук useLocation з react-router-dom для отримання поточного маршруту:

```
const el = useLocation().
```

Потім, на основі поточного маршруту, компонент визначає, чи потрібно відображати Header. Якщо маршрут не є "login", "change-password" або "forgot-password", то Header відображається:

```
const isHeader = ["login", "change-password", "forgot-password"].some(el => el === location)
{ !isHeader && <Header/> }.
```

Далі, за допомогою компонента Routes з react-router-dom, визначаються маршрути та відповідні компоненти, які будуть відображатися для кожного маршруту:

```
      <Routes>
        <Route path="/login" element={<LoginForm />} />
        <Route                                  path="/change-password"
element={ <ChangePassword/> }/>
        <Route                                  path="/forgot-password"
element={ <ForgotPassword/> }/>
        <Route path="/" element={<Outlet />}>
          <Route
            path="/"
            element={
              <PrivateRoute
                roles={[Role.Nurse, Role.Admin]}
                elementsMapping={{
                  Nurse: <ListPage />,
                  Admin: <Settings />,
                }}
              />
            }
          />
          <Route
            path="/edit-operation/:id"
            element={
              <PrivateRoute roles={[Role.Nurse, Role.Admin]}>
                <EditOperationPage />
              </PrivateRoute>
            }
          />
          <Route
            path="/add-operation"
            element={
              <PrivateRoute roles={[Role.Nurse, Role.Admin]}>
                <CreateOperationPage />
```

```
          </PrivateRoute>
        }
      />
      <Route
        path="/settings"
        element={
          <PrivateRoute roles={[Role.Admin]}>
            <Settings />
          </PrivateRoute>
        }
      />
      <Route
        path="/statistics"
        element={
          <PrivateRoute roles={[Role.Nurse, Role.Admin]}>
            <StatisticsPage />
          </PrivateRoute>
        }
      />
      <Route
        path="/list"
        element={
          <PrivateRoute roles={[Role.Nurse, Role.Admin]}>
            <ListPage />
          </PrivateRoute>
        }
      />
      <Route path="/tv/:department" element={<TVList />} />
      <Route path="/tv" element={<TVSelect />} />
    </Route>
  </Routes>.
```

Для захисту маршрутів, які вимагають автентифікації та певних ролей користувача, використовується компонент PrivateRoute. Він перевіряє, чи

користувач має необхідні ролі (Nurse або Admin) для доступу до відповідного компонента:

```
<PrivateRoute roles={[Role.Nurse, Role.Admin]}>
<EditOperationPage />
</PrivateRoute>.
```

Компонент App також використовує інші компоненти, такі як Account, Toast, Modal, для забезпечення загальної функціональності та відображення додаткових елементів інтерфейсу.

Таким чином, вхідна точка проєкту (App) є центральним компонентом, який керує маршрутизацією, відображенням компонентів та забезпечує загальну структуру та функціональність React-додатку.

3.3 Тестування роботи програмного модулю WEB-додатку організації роботи медичних закладів

Під час тестування програмного модулю потрібно було перевірити коректність введення та відображення даних. Також перевірити систему авторизації та кожного окремого модуля.

Для того, щоб запустити наш WEB-додаток, можна використовувати будь-який редактор коду, у нас це буде WebStorm. За допомогою команди npm і встановимо всі необхідні пакети та запустивши команду npm run dev запустимо наш додаток.

Відкривши посилання http://localhost:3000, нас зустріне форма авторизації. Одразу можна перевірити нашу систему авторизації. Як бачимо на рисунку 3.1, ввівши неправильні дані, ви не зможете потрапити до системи. Одразу потрібно зазначити, що до системи мають доступ лише ті люди, що були раніше зареєстровані адміністратором. Такий метод управління може вберегти систему від несанкціонованих користувачів та їх шкідливого впливу на неї.

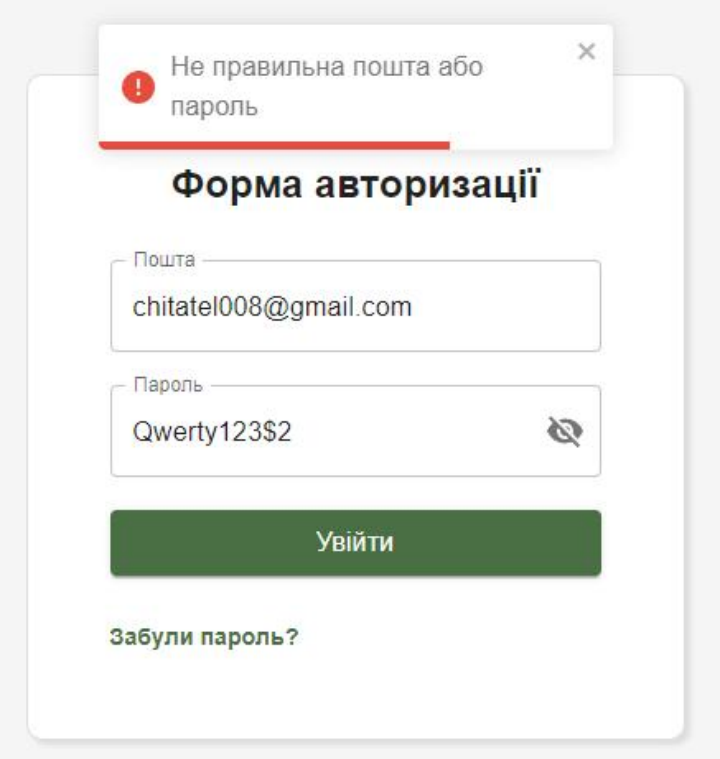

Рисунок 3.1 – Загальний вигляд інтерфейсного вікна форми авторизації

У разі успішної авторизації ми побачимо нашу систему, та може переглянути її детальніше.

Якщо заходити з доступом звичайного користувача, то ви одразу побачите весь список операцій(рис. 3.2), але якщо ви будете адміністратором, то ви побачите функціонал для управління персоналом та системою загалом(рис 3.3).

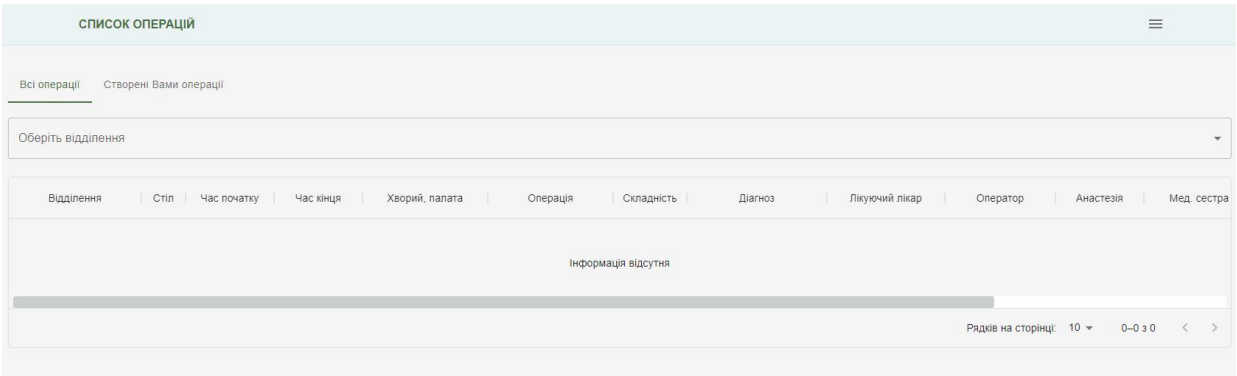

Рисунок 3.2 – Загальний вигляд інтерфейсного вікна зі списком операцій

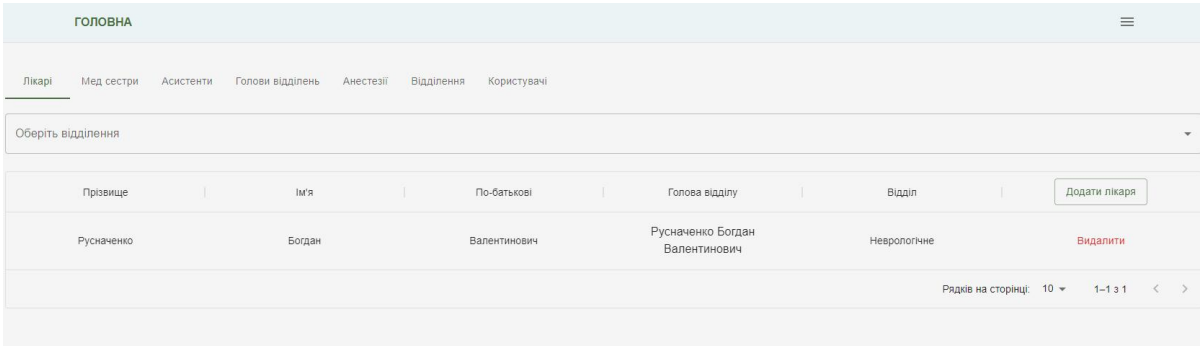

Рисунок 3.3 – Загальний вигляд інтерфейсного вікна для редагування працівників медичного закладу

Після авторизації від імені адміністратора перейдемо до вкладки “Користувачі”, де ми зможемо побачити всіх користувачів, які мають доступ до користування нашою системою на рисунку 3.4.

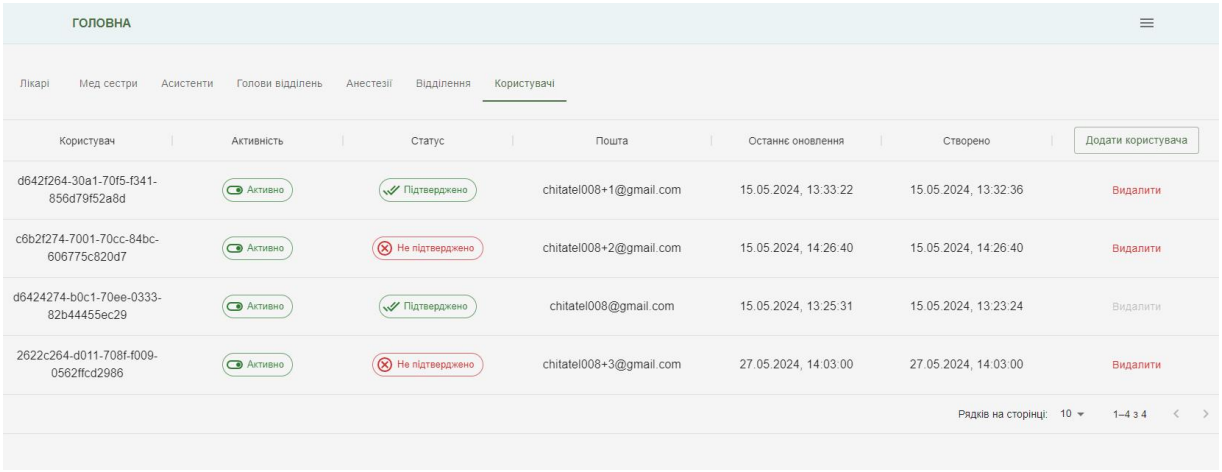

Рисунок 3.4 – Загальний вигляд інтерфейсного вікна зі списком довірених користувачів

Щоб створити нову операцію потрібно в меню справа вибрати відповідну опцію, після чого вас перенаправить до відповідної форми, що зображена на рисунку 3.5.

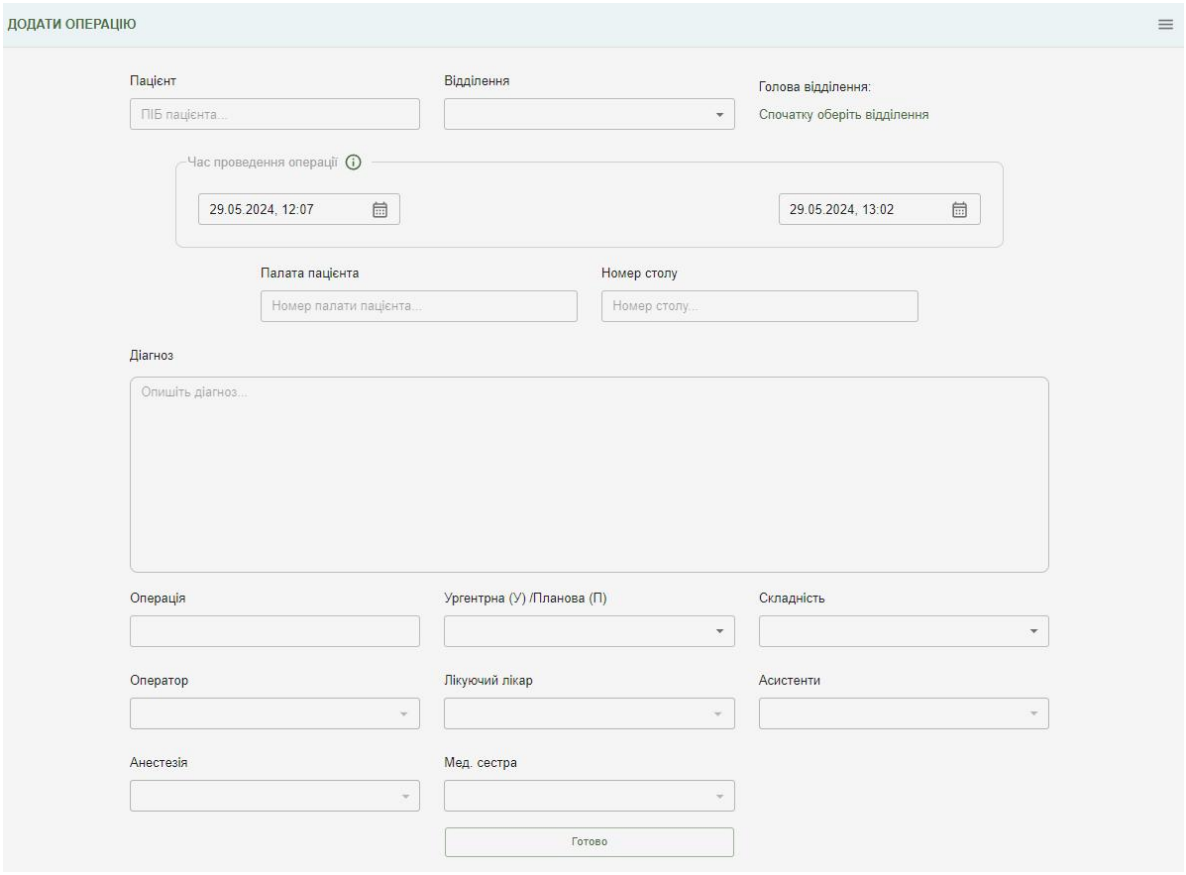

Рисунок 3.5 – Загальний вигляд інтерфейсного форми для створення операції

Після заповнення всіх полів, натискаємо Готово і операція створена. Вона одразу буде додана до бази даних і ми зможемо взаємодіяти з нею(рис. 3.6).

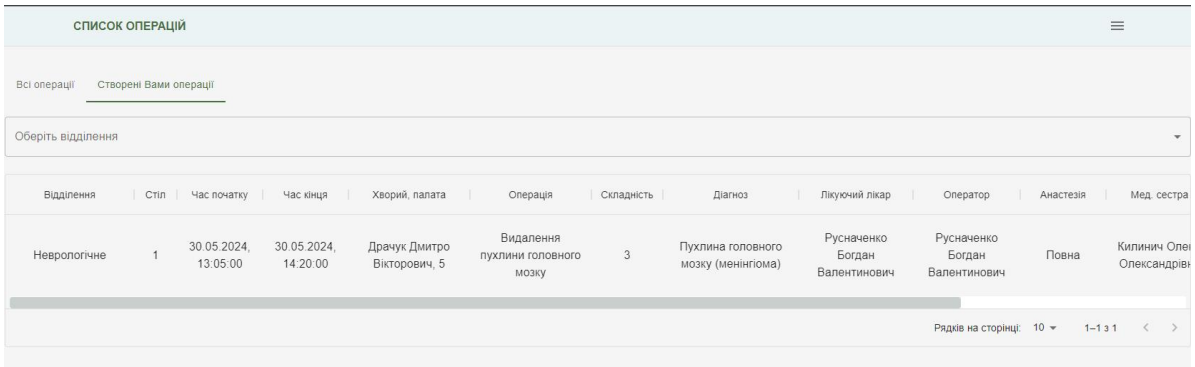

Рисунок 3.6 – Загальний вигляд інтерфейсного вікна списку операцій після додавання нової

Проаналізувавши існуючі рішення для організації роботи медичних закладів, у таблиці 3.1 наведено порівняння основних цінових політик компаній з нашою.

Таблиця 3.1 – Порівняльний аналіз сучасних засобів та розробленої системи

	Наша система	Practice Fusion EHR	Athenah ealth	Epic Systems	Cerner	eClinical-Works
Середній час впровадження (місяці)	2	4	4	6	5	4
Частота оновлень (на рік)	4	8	10	6	8	9
Ціна ($ на місяць за користувача)	50	100	200	500	400	250
Початкова вартість впровадження(тис. $)	5	15	20	50	40	25

3.4 Висновок до розділу 3

У третьому розділі було розглянуто вибір інструментів розробки front-end частини WEB-додатку, обґрунтовано середовище програмування та описано програмну реалізацію модуля створення операцій.

Для розробки front-end було обрано бібліотеку React завдяки її компонентній архітектурі, віртуальному DOM, багатій екосистемі та продуктивності розробки. Також вирішено використовувати бібліотеку

компонентів Material-UI (MUI) для прискорення розробки інтерфейсу користувача та забезпечення узгодженості дизайну.

Як середовище програмування було обрано WebStorm завдяки його потужним функціям, зручності використання та інтеграції з популярними WEB-технологіями.

Детально описано процес створення нової операції з використанням AWS Lambda та DynamoDB. Розглянуто отримання даних про операцію з запиту, збереження операції в таблиці DynamoDB та повернення відповідного результату або обробку помилок.

Порівняльний аналіз розробленої системи з існуючими рішеннями показав, що наша система має переваги з точки зору швидкості впровадження (2 місяці проти 4-6 місяців у конкурентів), нижчої ціни ($50 на місяць за користувача проти $100-500) та меншої початкової вартості впровадження ($5 тис. проти $15-50 тис.).

Таким чином, вибрані інструменти та середовище програмування, а також реалізований модуль створення операцій забезпечують ефективну та якісну розробку WEB-додатку організації роботи медичних закладів.

ВИСНОВКИ

Поставлені перед бакалаврською кваліфікаційною роботою задачі виконано в повному обсязі, а саме:

- Проведено аналіз предметної області, методів та засобів організації роботи медичних закладів.
- Розглянуто сучасні системи для організації роботи медичних закладів, виявлено їх переваги та недоліки.
- Розроблено структуру програмного забезпечення для організації роботи медичних закладів.
- Виконано програмну реалізацію системи організації роботи медичних закладів.
- Проведено тестування роботи програмного модуля WEB-додатку. Визначено, що розроблена система має переваги з точки зору швидкості впровадження (2 місяці проти 4-6 місяців у конкурентів) та меншої початкової вартості впровадження ($5 тис. проти $15-50 тис.).

У ході виконання бакалаврської роботи було проаналізовано основні проблеми розробки WEB-додатку для організації роботи медичних закладів. Розроблено структуру програмного модуля та UML-діаграму.

Відповідно з завданням до бакалаврської роботи було обґрунтовано вибір мов програмування JS/TS для програмної реалізації WEB-додатку для організації роботи медичних закладів, а також обґрунтовано вибір середовища розробки. Також було проведено тестування.

Мета роботи - підвищення рівня автоматизації обліку процесу роботи надання медичних послуг за допомогою розробки програмного модуля WEB-додатку для організації роботи медичних закладів - досягнута за рахунок реалізації функціональних можливостей системи, які дозволяють ефективно керувати та здійснювати облік роботи медичних закладів, а також за рахунок економічних переваг розробленої системи у порівнянні з існуючими аналогами.

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. New Information Technologies, Simulation and Automation, [Електронний ресурс]. Режим доступу: https://www.researchgate.net/publication/362014317_New_Information_Technologies_Simulation_and_Automation .

2. Web-Based Applications in Healthcare, [Електронний ресурс]. Режим доступу: https://www.researchgate.net/publication/251225200_WebBased_Applications_in_Healthcare.

3. Advantages of Electronic Health Records, [Електронний ресурс]. Режим доступу: https://www.healthit.gov/faq/what-are-advantages-electronic-health-records.

4. Русначенко Б.В., Іванчук Я. В. "Перспективи розробки програмного модуля веб-додатку організації роботи медичних закладів" в Матеріалах конференції «LIII Всеукраїнська науково-технічна конференція факультету інтелектуальних інформаційних технологій та автоматизації (2024)», Вінниця, 2024. [Електронний ресурс]. Режим доступу: https://conferences.vntu.edu.ua/index.php/all-fksa/all-fksa-2024/paper/view/21026.

5. Health Information Privacy, [Електронний ресурс]. Режим доступу: https://www.hhs.gov/hipaa/index.html.

6. Introduction to HL7 Standards, [Електронний ресурс]. Режим доступу: https://www.hl7.org/implement/standards/.

7. Cybersecurity in Healthcare, [Електронний ресурс]. Режим доступу: https://www.himss.org/resources/cybersecurity-healthcare.

8. Practice Fusion EHR, [Електронний ресурс]. Режим доступу: https://www.practicefusion.com/.

9. Athenahealth, [Електронний ресурс]. Режим доступу: https://www.athenahealth.com/solutions/athenaone.

10. Epic Systems, [Електронний ресурс]. Режим доступу: https://www.emrfinder.com/epic-ehr-software/.

11. Cerner for Electronic Medical Records Software, [Електронний ресурс]. Режим доступу: https://startupstash.com/tools/cerner/.

12. eClinicalWorks EMR Platform, [Електронний ресурс]. Режим доступу: https://www.revelemd.com/services/eclinicalworks/emr-0.

13. SQL vs. NoSQL Databases: What's the Difference?, [Електронний ресурс]. Режим доступу: https://www.ibm.com/blog/sql-vs-nosql/.

14. What is NoSQL?, [Електронний ресурс]. Режим доступу: https://www.mongodb.com/resources/basics/databases/nosql-explained.

15. Amazon DynamoDB, [Електронний ресурс]. Режим доступу: https://aws.amazon.com/dynamodb/.

16. Core components of Amazon DynamoDB, [Електронний ресурс]. Режим доступу: https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/HowItWorks.CoreComponents.html.

17. What is Amazon DynamoDB, [Електронний ресурс]. Режим доступу: https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/Integrations.html.

18. Amazon Cognito Documentation, [Електронний ресурс]. Режим доступу: https://docs.aws.amazon.com/cognito/.

19. AWS SDK for JavaScript Documentation , [Електронний ресурс]. Режим доступу: https://docs.aws.amazon.com/sdk-for-javascript/.

20. Build a Basic Web Application TUTORIAL, [Електронний ресурс]. Режим доступу: https://aws.amazon.com/getting-started/hands-on/build-web-app-s3-lambda-api-gateway-dynamodb/module-four/?nc1=h_ls.

21. Monolithic Architecture, [Електронний ресурс]. Режим доступу: https://www.ibm.com/cloud/learn/monolithic-architecture.

22. Service-Oriented Architecture (SOA), [Електронний ресурс]. Режим доступу: https://www.ibm.com/cloud/learn/soa.

23. Microservices Architecture, [Електронний ресурс]. Режим доступу: https://www.ibm.com/cloud/learn/microservices.

24. Angular, [Електронний ресурс]. Режим доступу: https://angular.io/.

25. Vue.js, [Електронний ресурс]. Режим доступу: https://vuejs.org/.

26. React, [Електронний ресурс]. Режим доступу: https://reactjs.org/.

27. React Components, [Електронний ресурс]. Режим доступу: https://reactjs.org/docs/components-and-props.html.

28. React Virtual DOM, [Електронний ресурс]. Режим доступу: https://reactjs.org/docs/faq-internals.html.

29. React Ecosystem, [Електронний ресурс]. Режим доступу: https://reactjs.org/community/support.html.

30. React Developer Tools, [Електронний ресурс]. Режим доступу: https://reactjs.org/blog/2015/09/02/new-react-developer-tools.html.

31. Material-UI, [Електронний ресурс]. Режим доступу: https://mui.com/.

32. WebStorm The JavaScript and TypeScript IDE, [Електронний ресурс]. Режим доступу: https://www.jetbrains.com/webstorm/.

ДОДАТОК А
ПРОТОКОЛ ПЕРЕВІРКИ КВАЛІФІКАЦІЙНОЇ РОБОТИ НА НАЯВНІСТЬ ТЕКСТОВИХ ЗАПОЗИЧЕНЬ

Назва роботи: Програмний модуль WEB-додатка організації роботи медичних закладів

Тип роботи: бакалаврська дипломна робота
(БДР, МКР)

Підрозділ кафедра комп'ютерних наук, ФІІТА
(кафедра, факультет)

Показники звіту подібності Unicheck

Оригінальність 97,84% Схожість 2,16%

Аналіз звіту подібності (відмітити потрібне):

☑ Запозичення, виявлення у роботі, оформлені коректно і не містять ознак плагіату.

□ Виявлені у роботі запозичення не мають ознак плагіату, але їх надмірна кількість викликає сумніви щодо цінності роботи і відсутності самостійності її виконання автором. Роботу направити на розгляд експертної комісії кафедри.

□ Виявлені у роботі запозичення є недобросовісними і мають ознаки плагіату та/або в ній містяться навмисні спотворення тексту, що вказують на спроби приховування недобросовісних запозичень.

Особа, відповідальна за перевірку _____ Озеранський В.С.

Ознайомлені з повним звітом подібності, який був згенерований системою Unicheck щодо роботи.

Автор роботи _____ Русначенко Б.В.

Керівник роботи _____ Іванчук Я.В.

ДОДАТОК Б

Інструкція користувача

1. Відкриваємо редактор коду.
2. За допомогою команди npm і встановимо всі необхідні пакети
3. Запустивши команду npm run dev запустимо наш додаток..
4. Відкриваємо посилання localhost http://127.0.0.1:8000/ та користуємось програмою.

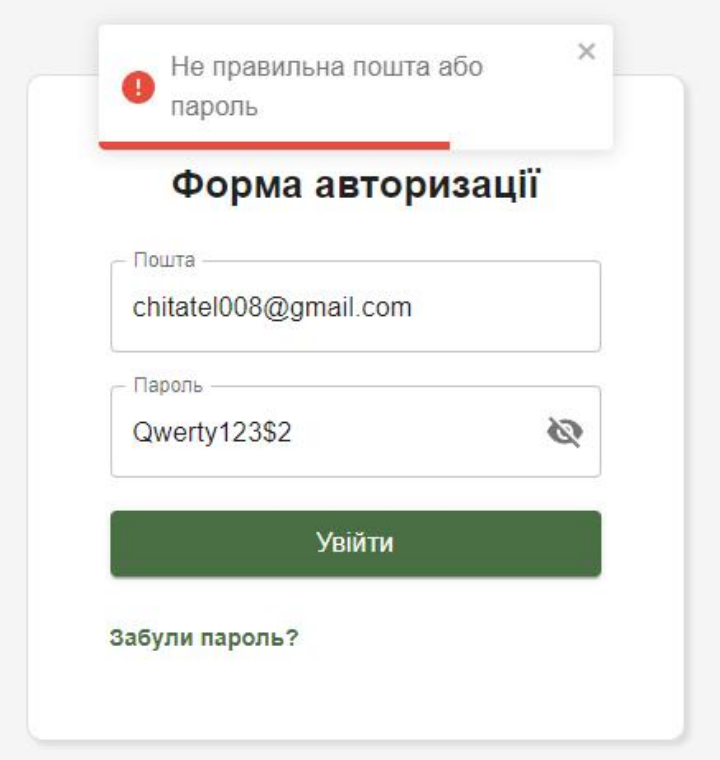

Рисунок Б.1 – Загальний вигляд інтерфейсного вікна форми авторизації

5. Переглядаємо список операцій, додаємо, редагуємо чи видаляємо за необхідності.

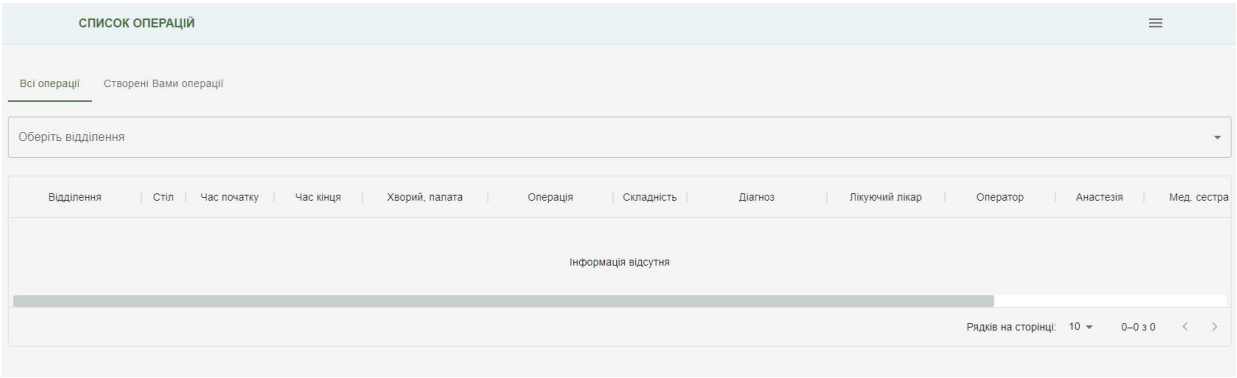

Рисунок Б.2 – Загальний вигляд інтерфейсного вікна зі списком операцій

6. Натиснувши кнопку створити нас перенаправить до відповідної форми.

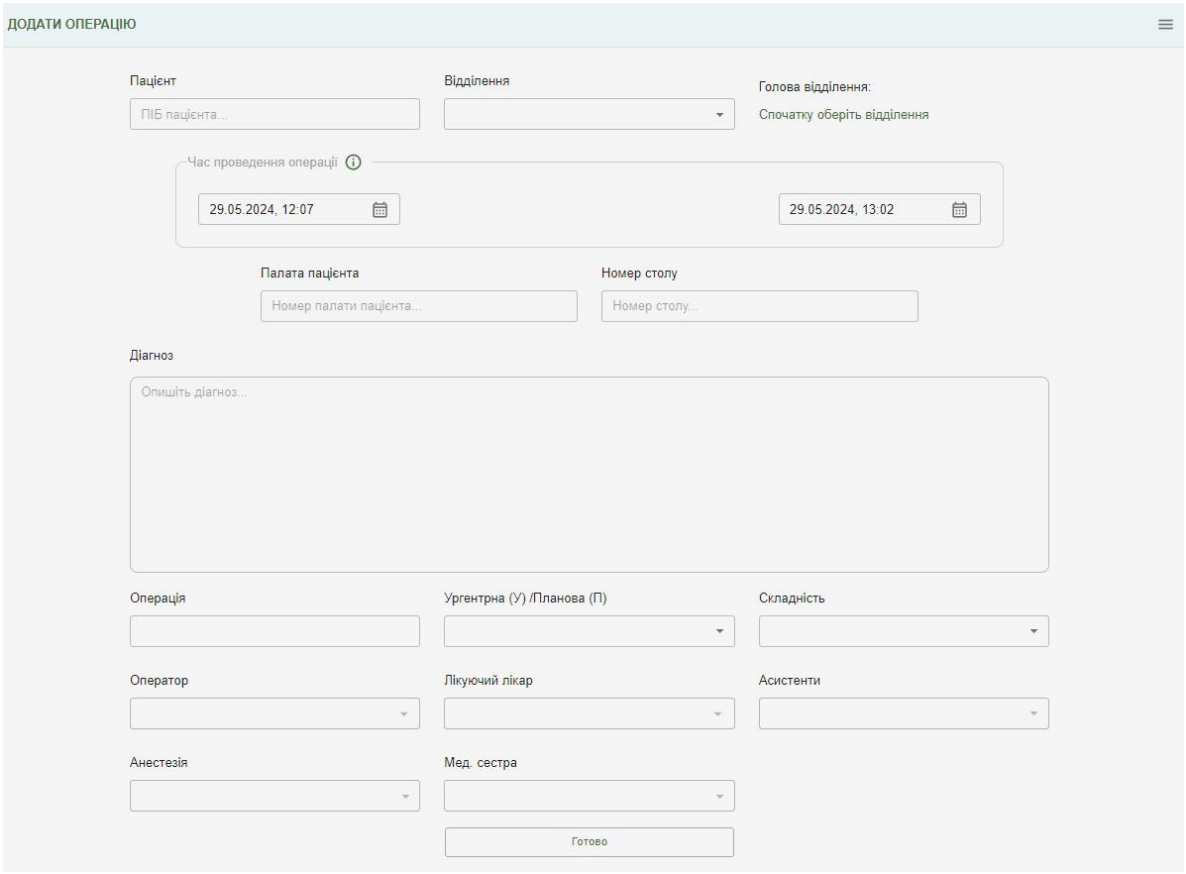

Рисунок Б.3 – Загальний вигляд інтерфейсного вікна форми для створення операції

ДОДАТОК В

Лістинг програми

Модуль створення нових операцій

```
export const CreateOperation = async (event: any): Promise<APIGatewayProxyResult> => {
    let response: APIGatewayProxyResult;
    try {
        const info = event.requestContext.authorizer.claims;
        const nurseId = info['sub'];
        const client = new AWS.DynamoDB(ddbOptions);
        const operationState = JSON.parse(event.body);
        const defaultOperationStatus = "Created";
        const {
            department,
            headOfDepartment,
            tableNumber,
            dateStart,
            dateEnd,
            createdDate,
            patient,
            medicalChamber,
            diagnosis,
            hospitalizationType,
            attendingDoctor,
            operatingDoctor,
            operation,
            complexityOperation,
            assistants,
            anesthesia,
            nurse,
        } = operationState;
        const params: AWS.DynamoDB.Types.PutItemInput = {
            TableName: process.env.TABLE || table,
            Item: {
                id: { S: nanoid(8) },
```

```
                department: { S: department },
                headOfDepartment: { S: headOfDepartment },
                tableNumber: { S: tableNumber },
                dateStart: { S: dateStart },
                dateEnd: { S: dateEnd },
                actualDate: { S: "" },
                createdDate: { S: createdDate },
                updatedDate: { S: "" },
                patient: { S: patient },
                medicalChamber: { S: medicalChamber },
                diagnosis: { S: diagnosis },
                hospitalizationType: { S: hospitalizationType },
                attendingDoctor: { S: attendingDoctor },
                operatingDoctor: { S: operatingDoctor },
                operation: { S: operation },
                complexityOperation: { S: complexityOperation },
                assistants: { SS: assistants },
                anesthesia: { S: anesthesia },
                nurse: { S: nurse },
                nurseId: { S: nurseId },
                operationStatus: { S: defaultOperationStatus },
            },
        };
        await client.putItem(params).promise();
        response = {
            statusCode: 200,
            headers: {
                "Content-Type": "application/json",
                "Access-Control-Allow-Origin": "*",
                "Access-Control-Allow-Headers": "*",
            },
            body: JSON.stringify({
                message: "The operation was successfully created!",
            }),
        };
    } catch (err: any) {
```

```
        console.log(err);
        response = {
            statusCode: err.statusCode,
            body: JSON.stringify({
                message: err ? err.message : "some error happened",
            }),
        };
    }
    return response;
};
```

Модуль редагування операцій

```
import { APIGatewayProxyResult } from "aws-lambda";
import * as AWSCore from "aws-sdk";
import * as AWSXRay from "aws-xray-sdk-core";
let AWS: typeof AWSCore;
const table = "operations";
const ddbOptions: AWSCore.DynamoDB.Types.ClientConfiguration = {
    apiVersion: "2012-08-10",
};
if (process.env.AWS_SAM_LOCAL) {
    AWS = AWSCore;
    ddbOptions.endpoint = "http://dynamodb:8000";
} else {
    AWS = AWSXRay.captureAWS(AWSCore);
}
export const UpdateOperation = async (event: any): Promise<APIGatewayProxyResult> => {
    let response: APIGatewayProxyResult;
    try {
        const client = new AWS.DynamoDB.DocumentClient(ddbOptions);
        const operationState = JSON.parse(event.body);
        const {
            department,
            headOfDepartment,
            tableNumber,
```

```
            dateStart,
            dateEnd,
            actualDate,
            patient,
            medicalChamber,
            diagnosis,
            hospitalizationType,
            attendingDoctor,
            operatingDoctor,
            operation,
            complexityOperation,
            assistants,
            anesthesia,
            nurse,
            operationStatus,
        } = operationState;
        const params: AWS.DynamoDB.Types.UpdateItemInput = {
            TableName: process.env.TABLE || table,
            Key: {
                id: event.pathParameters.id,
            },
            UpdateExpression: `SET department = :department, headOfDepartment
= :headOfDepartment, tableNumber = :tableNumber, dateStart = :dateStart, dateEnd = :dateEnd,
actualDate = :actualDate, patient = :patient, medicalChamber = :medicalChamber, diagnosis = :diagnosis,
hospitalizationType = :hospitalizationType, attendingDoctor = :attendingDoctor, operatingDoctor
= :operatingDoctor, operation = :operation, complexityOperation = :complexityOperation, assistants
= :assistants, anesthesia = :anesthesia, nurse = :nurse, operationStatus = :operationStatus`,
            ExpressionAttributeValues: {
                ":department": department,
                ":headOfDepartment": headOfDepartment,
                ":tableNumber": tableNumber,
                ":dateStart": dateStart,
                ":dateEnd": dateEnd,
                ":actualDate": actualDate,
                ":patient": patient,
                ":medicalChamber": medicalChamber,
```

```
                    ":diagnosis": diagnosis,
                    ":hospitalizationType": hospitalizationType,
                    ":attendingDoctor": attendingDoctor,
                    ":operatingDoctor": operatingDoctor,
                    ":operation": operation,
                    ":complexityOperation": complexityOperation,
                    ":assistants": client.createSet(assistants) as any,
                    ":anesthesia": anesthesia,
                    ":nurse": nurse,
                    ":operationStatus": operationStatus,
                },
                ReturnValues: "UPDATED_OLD",
            };
            await client.update(params).promise();
            response = {
                statusCode: 200,
                headers: {
                    "Content-Type": "application/json",
                    "Access-Control-Allow-Origin": "*",
                    "Access-Control-Allow-Headers": "*",
                },
                body: JSON.stringify({
                    message: "The operation was successfully updated!",
                }),
            };
        } catch (err: any) {
            console.log(err);
            response = {
                statusCode: err.statusCode,
                body: JSON.stringify({
                    message: err instanceof Error ? err.message : "some error happened",
                }),
            };
        }
        return response;
};
```

Модуль пошуку всіх користувачі, що мають доступ до системи

```
import { APIGatewayProxyResult } from "aws-lambda";
import AWS from "aws-sdk";
export const GetAllUsers = async (event: any): Promise<APIGatewayProxyResult> => {
    const CognitoIdentityServiceProvider = AWS.CognitoIdentityServiceProvider;
    const client = new CognitoIdentityServiceProvider({ apiVersion: "2016-04-19" });
    let response: APIGatewayProxyResult;
    try {
        const params = {
            UserPoolId: process.env.UserPoolId
        };
        // create the user in Cognito
        const userList = await client.listUsers(params as any).promise();
        response = {
            statusCode: 200,
            headers: {
                "Content-Type": "application/json",
                "Access-Control-Allow-Origin": "*",
                "Access-Control-Allow-Headers": "*",
            },
            body: JSON.stringify(userList),
        };
    } catch (err: any) {
        response = {
            statusCode: err.statusCode,
            headers: {
                "Content-Type": "application/json",
                "Access-Control-Allow-Origin": "*",
                "Access-Control-Allow-Headers": "*",
            },
            body: JSON.stringify({
                message: err.message,
            }),
        };
```

```
    }
    return response;
};
```

Модуль додавання нових користувачів

```
import { APIGatewayProxyResult } from "aws-lambda";
import AWS from "aws-sdk";
export const CreateUser = async (event: any): Promise<APIGatewayProxyResult> => {
    const CognitoIdentityServiceProvider = AWS.CognitoIdentityServiceProvider;
    const client = new CognitoIdentityServiceProvider({ apiVersion: "2016-04-19" });
    let response: APIGatewayProxyResult;
    try {
        const state = JSON.parse(event.body);
        const { username, temporaryPassword, email, firstName, lastName } = state;
        const params = {
            UserPoolId: process.env.UserPoolId,
            Username: username,
            TemporaryPassword: temporaryPassword,
            UserAttributes: [
                { Name: "email", Value: email },
                { Name: "given_name", Value: firstName },
                { Name: "family_name", Value: lastName },
                { Name: "email_verified", Value: "true" },
            ],
        };
        const addUserToGroupParams = {
            GroupName: process.env.GroupName,
            UserPoolId: process.env.UserPoolId,
            Username: username,
        };
        // create the user in Cognito
        await client.adminCreateUser(params as any).promise();
        await client.adminAddUserToGroup(addUserToGroupParams as any).promise()
        response = {
            statusCode: 200,
```

```
                headers: {
                    "Content-Type": "application/json",
                    "Access-Control-Allow-Origin": "*",
                    "Access-Control-Allow-Headers": "*",
                },
                body: JSON.stringify({
                    message: "The user was successfully created!",
                }),
            };
        } catch (err: any) {
            response = {
                statusCode: err.statusCode,
                headers: {
                    "Content-Type": "application/json",
                    "Access-Control-Allow-Origin": "*",
                    "Access-Control-Allow-Headers": "*",
                },
                body: JSON.stringify({
                    message: err.message,
                }),
            };
        }
        return response;
};
```

Вхідна точка проєкту

```
import React from "react";
import { Outlet, Route, Routes, useLocation } from "react-router-dom";
import { useSelector } from "react-redux";
import Container from "@mui/material/Container";
import {
  CreateOperationPage,
  EditOperationPage,
  ListPage,
  Settings,
```

```
    StatisticsPage,
    TVList,
    TVSelect,
} from "./pages";
import { Header, Modal, PrivateRoute, Toast } from "./components";
import { Role } from "./components/interfaces/auth/IAuth";
import { useToast } from "./hooks/toast/useToast";
import { closeToast, selectToastState } from "./redux/slices/toast";
import { ToastType } from "./redux/types/toast";
import { appDispatch } from "./redux/store";
import { LoginForm } from "./components/cognito/signIn/Login";
import { Account } from "./components/cognito/account/Account";
import { ChangePassword } from "./components/cognito/change-password/ChangePassword";
import { ForgotPassword } from "./components/cognito/forgot-password/ForgotPassword";
function App() {
    const [successToast, errorToast] = useToast();
    const toast = useSelector(selectToastState);
    const el = useLocation();
    const [location, setLocation] = React.useState("");
    const isHeader = ["login", "change-password", "forgot-password"].some(el => el === location)
    React.useEffect(() => {
        setLocation(el.pathname.split("/")[1]);
    }, [el.pathname.split("/")]);
    React.useEffect(() => {
        if (toast.message) {
            switch (toast.toastType) {
                case ToastType.Error:
                    errorToast(toast.message);
                    break;
                case ToastType.Success:
                    successToast(toast.message);
                    break;
            }
            appDispatch(closeToast());
        }
    }, [toast?.message, toast?.toastType]);
```

```
return (
  <>
    <Account>
      <Toast />
      <Modal />
      { !isHeader && <Header/> }
      <Container
        style={{
          padding: location === "tv" ? "0 0 20px" : "0 20px 20px",
          maxWidth:
            ["", "list", "settings"].includes(location)
              ? "1700px"
              : location === "tv"
              ? "initial"
              : "",
        }}
      >
        <Routes>
          <Route path="/login" element={<LoginForm />} />
          <Route path="/change-password" element={ <ChangePassword/> }/>
          <Route path="/forgot-password" element={ <ForgotPassword/> }/>
          <Route path="/" element={<Outlet />}>
            <Route
              path="/"
              element={
                <PrivateRoute
                  roles={[Role.Nurse, Role.Admin]}
                  elementsMapping={{
                    Nurse: <ListPage />,
                    Admin: <Settings />,
                  }}
                />
              }
            />
            <Route
              path="/edit-operation/:id"
```

```
    element={
      <PrivateRoute roles={[Role.Nurse, Role.Admin]}>
        <EditOperationPage />
      </PrivateRoute>
    }
  />
  <Route
    path="/add-operation"
    element={
      <PrivateRoute roles={[Role.Nurse, Role.Admin]}>
        <CreateOperationPage />
      </PrivateRoute>
    }
  />
  <Route
    path="/settings"
    element={
      <PrivateRoute roles={[Role.Admin]}>
        <Settings />
      </PrivateRoute>
    }
  />
  <Route
    path="/statistics"
    element={
      <PrivateRoute roles={[Role.Nurse, Role.Admin]}>
        <StatisticsPage />
      </PrivateRoute>
    }
  />
  <Route
    path="/list"
    element={
      <PrivateRoute roles={[Role.Nurse, Role.Admin]}>
        <ListPage />
      </PrivateRoute>
```

```
                        }
                      />
                      <Route path="/tv/:department" element={<TVList />} />
                      <Route path="/tv" element={<TVSelect />} />
                    </Route>
                  </Routes>
                </Container>
              </Account>
            </>
        );
}

export default App;
```

ДОДАТОК Г

ІЛЮСТРАТИВНА ЧАСТИНА

«ПРОГРАМНИЙ МОДУЛЬ WEB-ДОДАТКА ОРГАНІЗАЦІЇ РОБОТИ МЕДИЧНИХ ЗАКЛАДІВ»

Виконав: студент 4 курсу,групи 2КН-20б
спеціальності 122 – Комп'ютерні науки
(шифр і назва напряму підготовки, спеціальності)

Русначенко Б.В.
(прізвище та ініціали)

Керівник: д.т.н., проф. каф. КН
Іванчук Я. В ..
(прізвище та ініціали)

« 07 » 06 2024 р.

Вінниця ВНТУ - 2024 рік

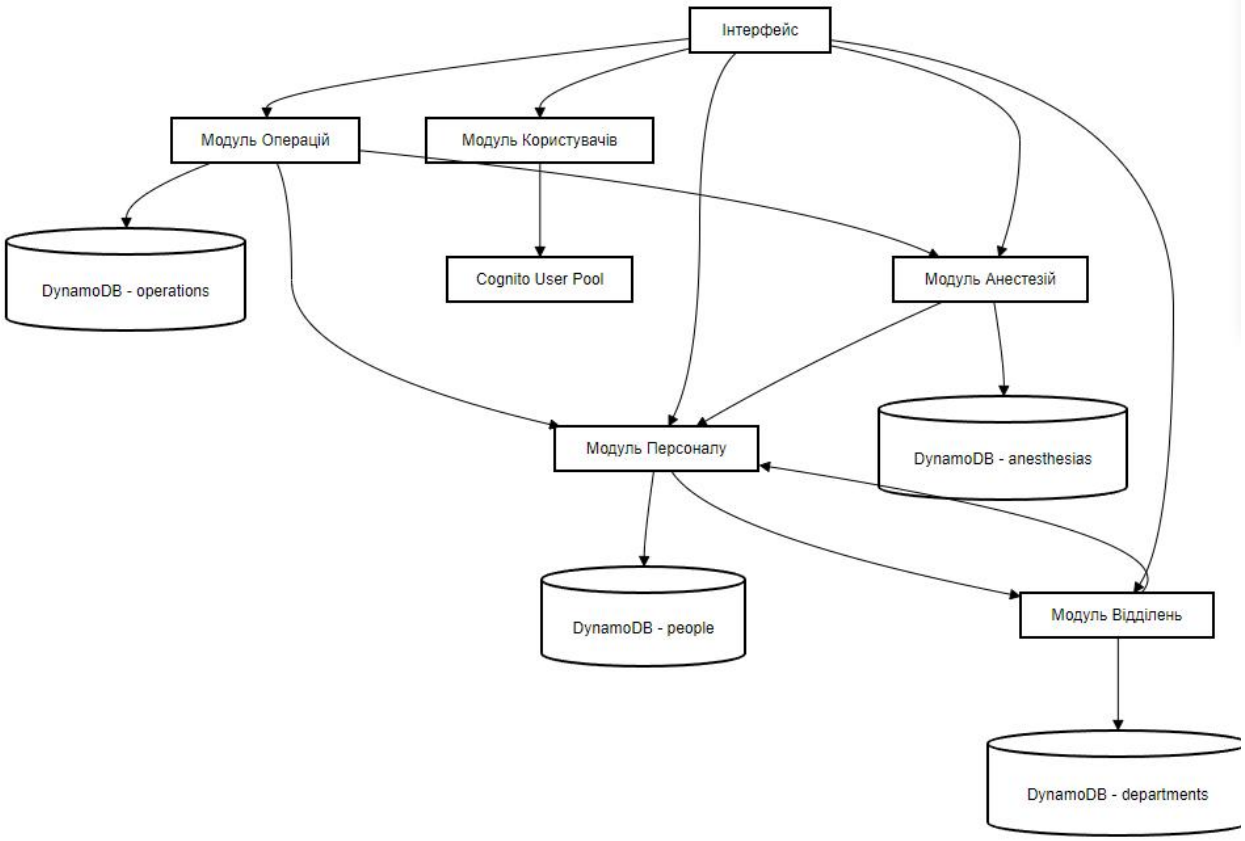

Рисунок Г.1 – Структурна схема WEB-додатку для організації роботи медичних закладів

Рисунок Г.2 – Типова схема модуля управління хірургічними операціями

Рисунок Г.3 – Типова схема модуля управління користувачами

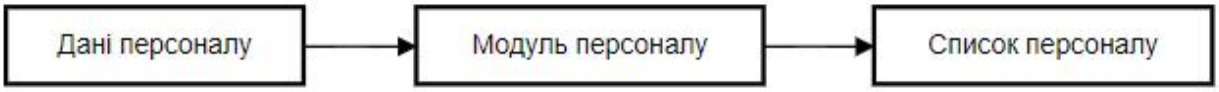

Рисунок Г.4 – Типова схема модуля управління медичним персоналом

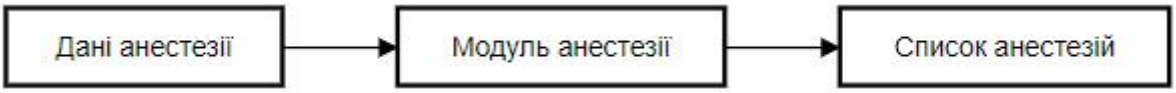

Рисунок Г.5 – Типова схема модуля управління медичною процедурою типу анестезія

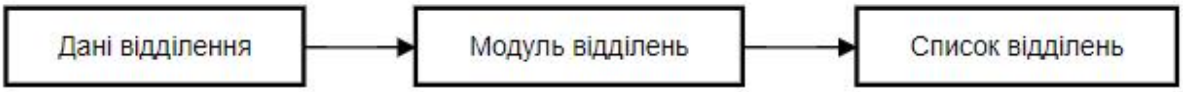

Рисунок Г.6 – Типова схема модуля управління структурними підрозділами

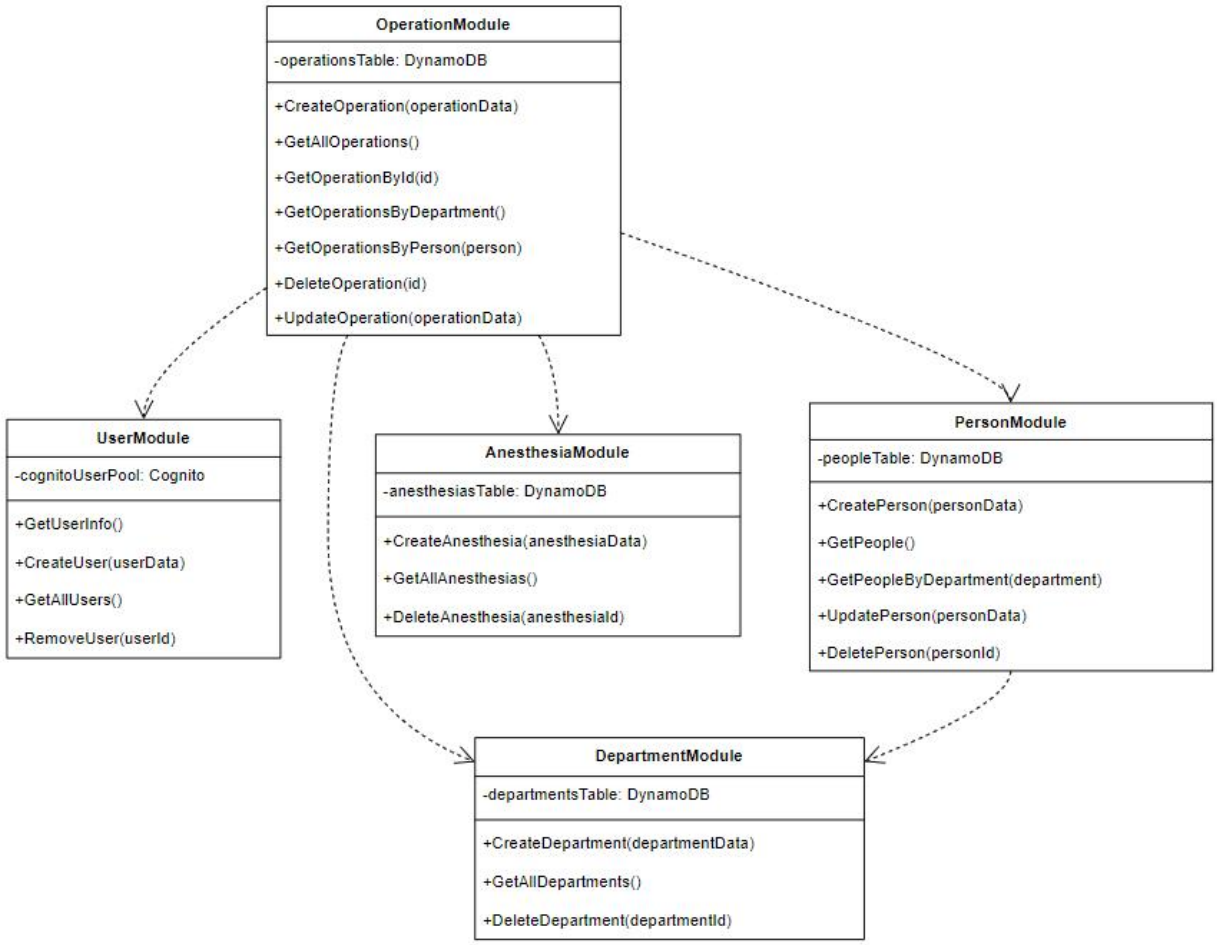

Рисунок Г.7 – UML - діаграма класів WEB-додатку

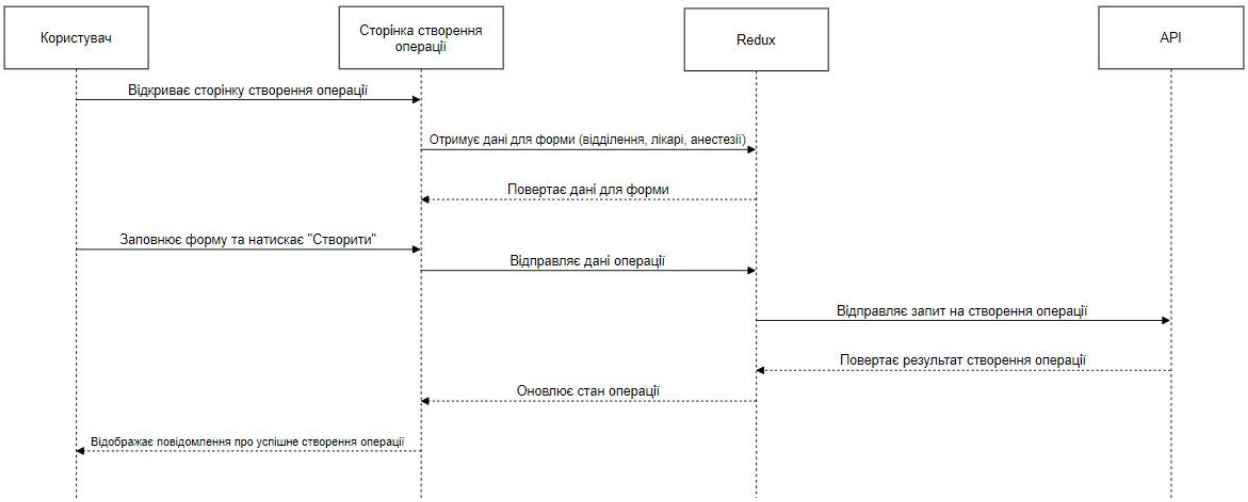

Рисунок Г.8 – UML-діаграма послідовностей до модуля створення операцій

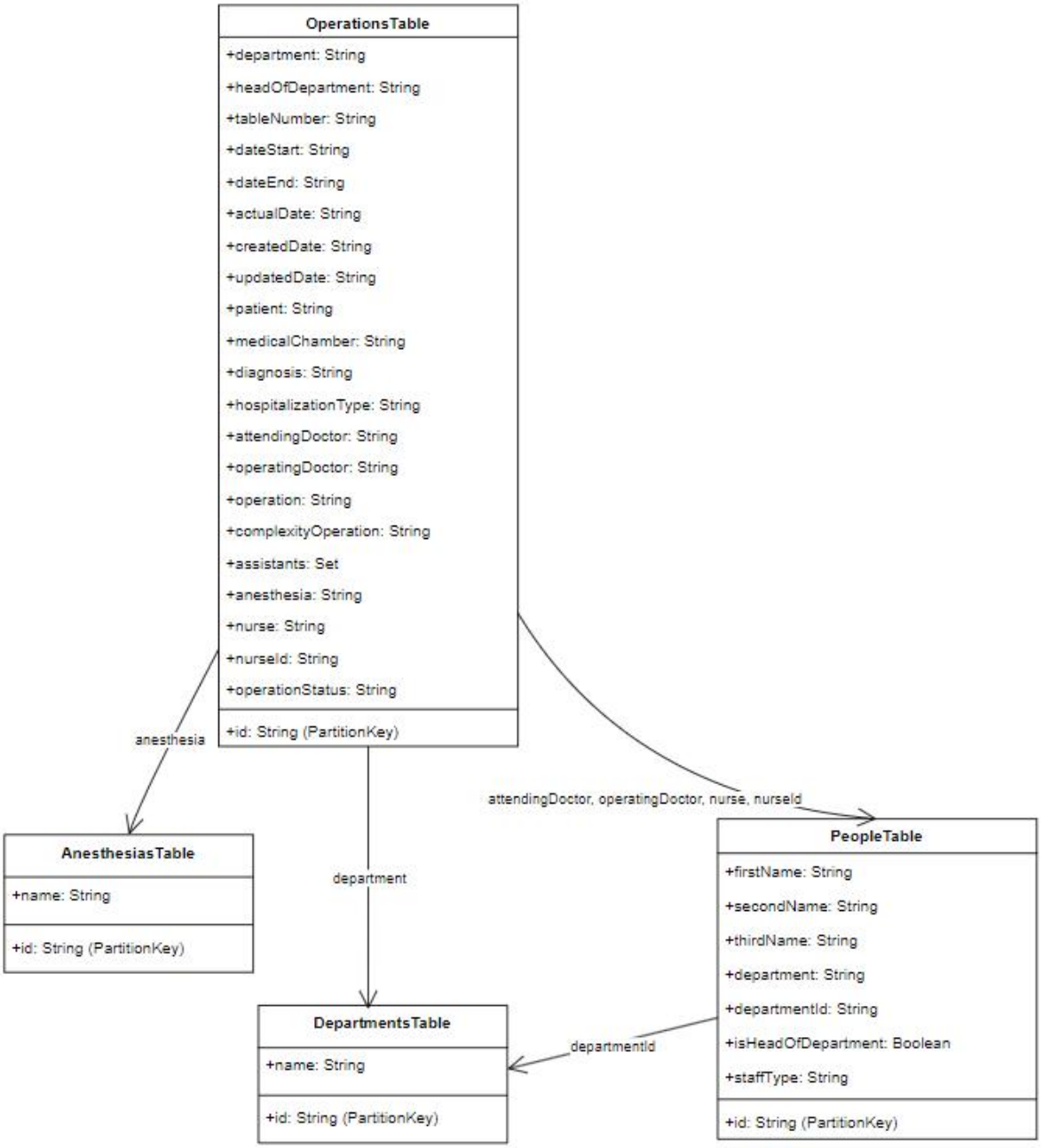

Рисунок Г.9 – UML-діаграма класів DynamoDB

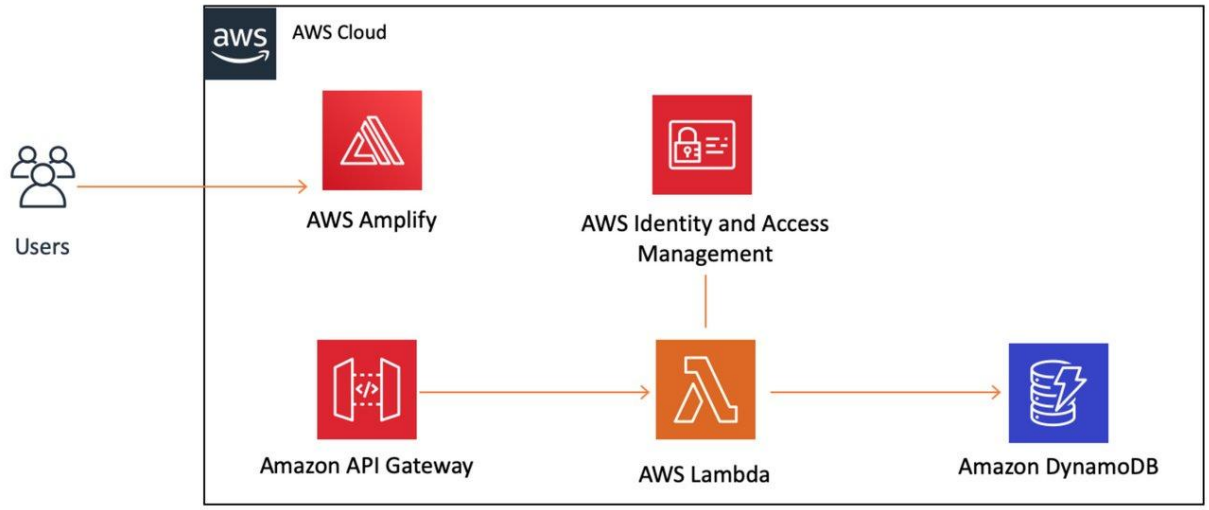

Рисунок Г.10 – Типова схема взаємодії з DynamoDB через API

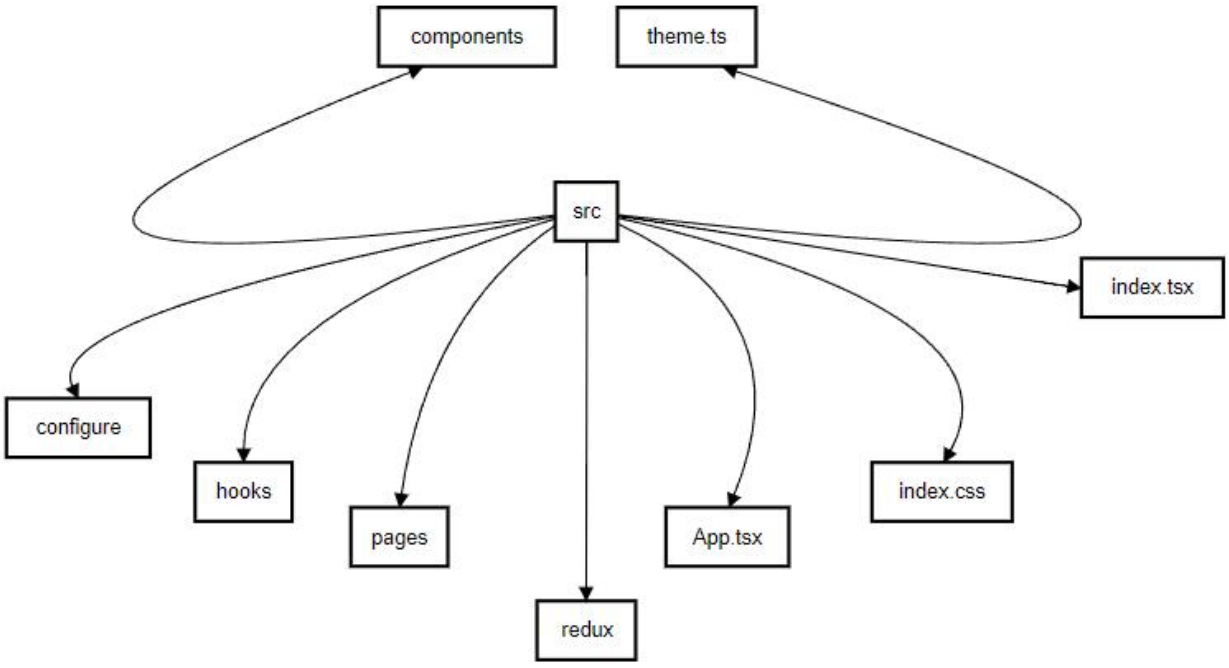

Рисунок Г.11 – Типова схема архітектури front-end частини WEB-додатку

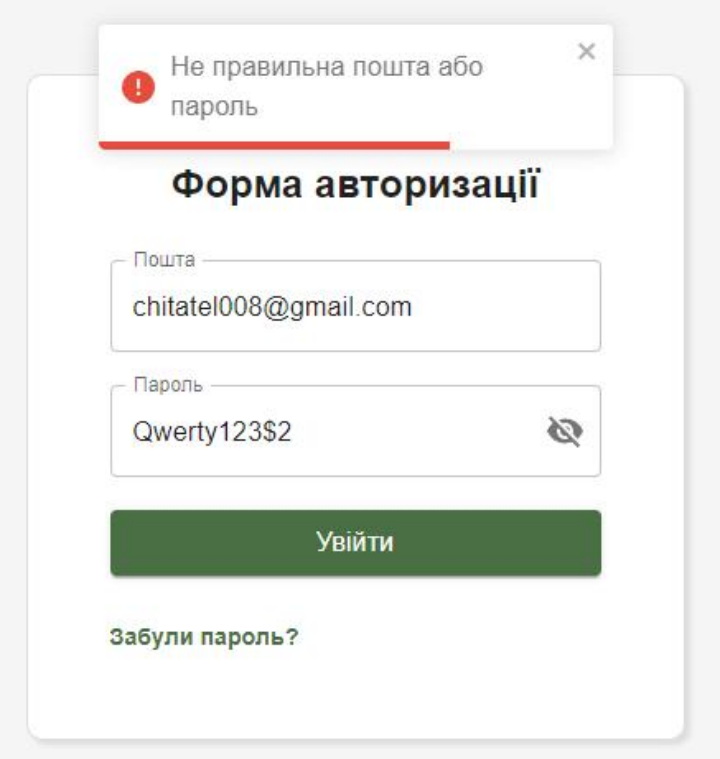

Рисунок Г.12 – Загальний вигляд інтерфейсного вікна форми авторизації

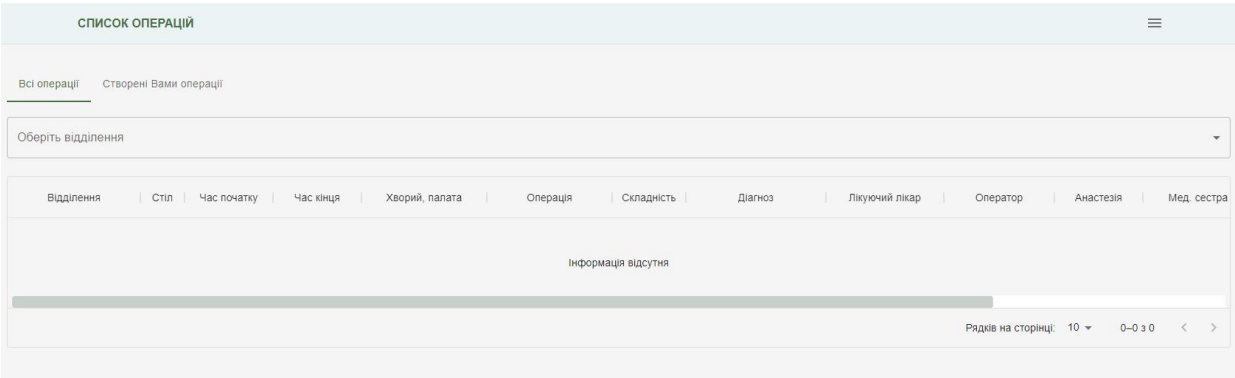

Рисунок Г.13 – Загальний вигляд інтерфейсного вікна зі списком операцій

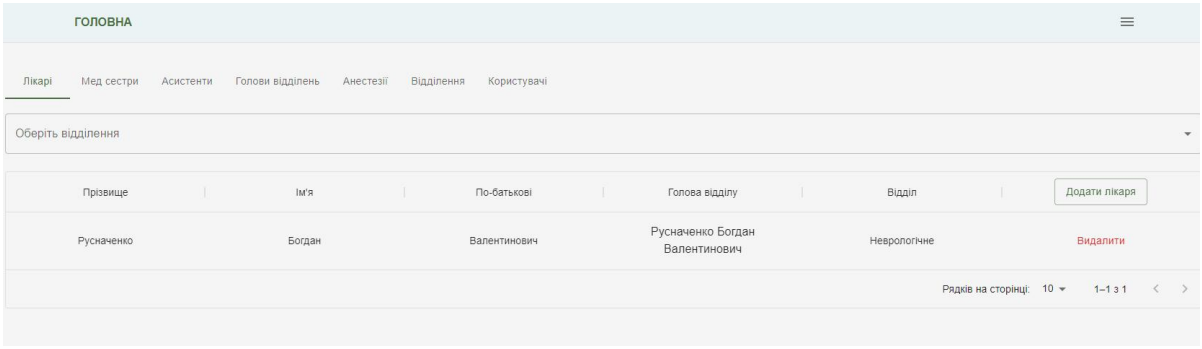

Рисунок Г.14 – Загальний вигляд інтерфейсного вікна для редагування працівників медичного закладу

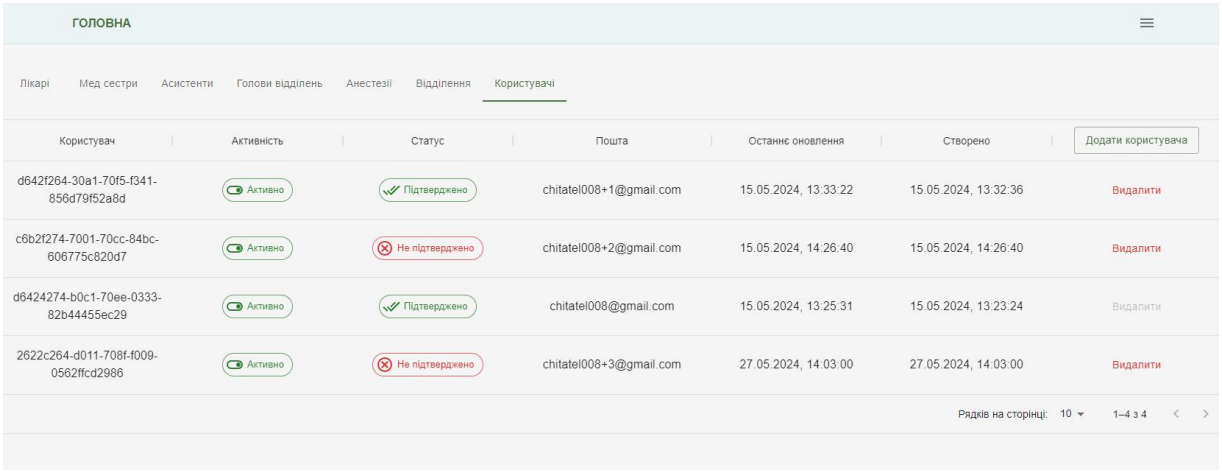

Рисунок Г.15 – Загальний вигляд інтерфейсного вікна зі списком довірених користувачів

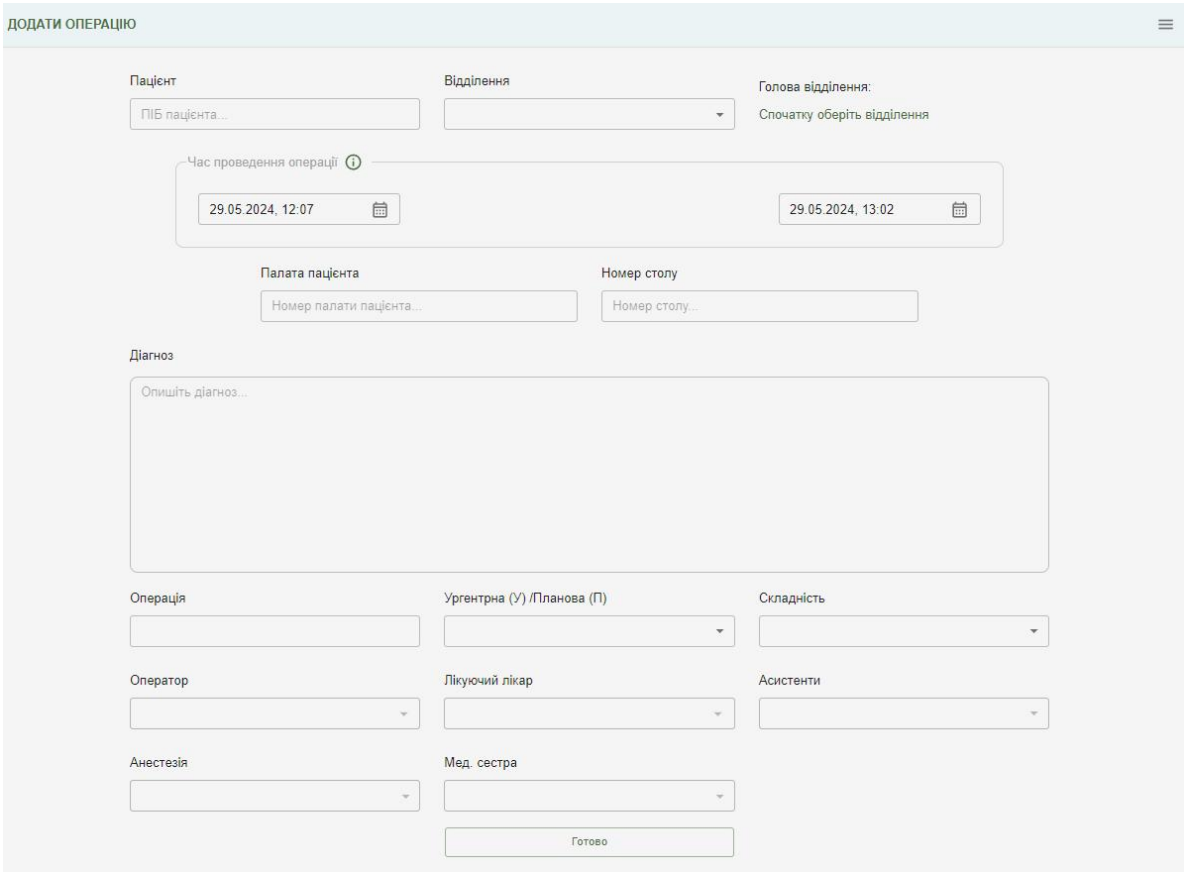

Рисунок Г.16 – Загальний вигляд інтерфейсного форми для створення операції

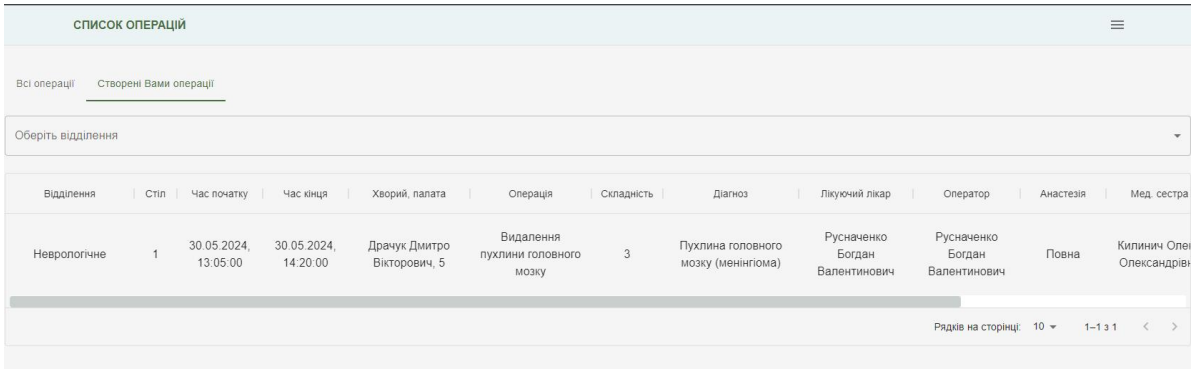

Рисунок Г.17 – Загальний вигляд інтерфейсного вікна списку операцій після додавання нової