Міністерство освіти і науки України
Харківський національний університет радіоелектроніки

Факультет комп'ютерних наук
(повна назва)

Кафедра програмної інженерії
(повна назва)

КВАЛІФІКАЦІЙНА РОБОТА
Пояснювальна записка

рівень вищої освіти перший (бакалаврський)

Програмна система для обліку проведення щеплень

(тема)

Виконав:
здобувач 4 року навчання
групи ПЗПІ-21-10

Нікіта ЯКУШЕВ
(Власне ім'я, ПРІЗВИЩЕ)

Спеціальність 121 – Інженерія програмного забезпечення
(код і повна назва спеціальності)
Тип програми освітньо-професійна
Освітня програма Програмна інженерія
(повна назва освітньої програми)

Керівник доц. Роксана МЕЛЬНІКОВА
(посада, Власне ім'я, ПРІЗВИЩЕ)

Допускається до захисту
Зав. кафедри _____ Кирило СМЕЛЯКОВ
(підпис) (Власне ім'я, ПРІЗВИЩЕ)

2025 р.

Харківський національний університет радіоелектроніки

Факультет комп’ютерних наук
Кафедра програмної інженерії
Рівень вищої освіти перший (бакалаврський)
Спеціальність 121 – Інженерія програмного забезпечення
Тип програми Освітньо-професійна
Освітня програма Програмна Інженерія
(шифр і назва)

ЗАТВЕРДЖУЮ:
Зав. кафедри _____
(підпис)
«___» _____ 2025 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

здобувачеві Якушеву Нікіті Олеговичу
(прізвище, ім’я, по батькові)

1. Тема роботи Програмна система для обліку проведення щеплень
Затверджена наказом по університету від 397Ст від 19.05.2025
2. Термін подання студентом роботи до екзаменаційної комісії 12.06.2025
3. Вихідні дані до роботи Розробити програмну систему для обліку проведення щеплень, яка вирішує проблеми медичних закладів, лікарів та пацієнтів, пов’язані з проведенням та обліком щеплень.
4. Перелік питань, що потрібно опрацювати в роботі
Вступ, аналіз предметної галузі, формування вимог до програмної системи, архітектура та проектування програмного забезпечення, опис прийнятих програмних рішень, тестування розробленого програмного забезпечення, впровадження програмного забезпечення, висновки.

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів роботи	Термін виконання етапів роботи	Примітка
1	Аналіз предметної галузі	20.05.2025	*виконано*
2	Створення специфікації ПЗ	21.05.2025	*виконано*
3	Проектування ПЗ	23.05.2025	*виконано*
4	Розробка ПЗ	26.05.2025	*виконано*
5	Тестування ПЗ	28.05.2025	*виконано*
6	Оформлення пояснювальної записки	30.05.2025	*виконано*
7	Підготовка презентації та доповіді	01.06.2025	*виконано*
8	Попередній захист	02.06.2025	*виконано*
9	Нормоконтроль, рецензування	08.06.2025	*виконано*
10	Здача роботи у електронний архів	10.06.2025	*виконано*
11	Допуск до захисту у зав. кафедри	10.06.2025	*виконано*

Дата видачі завдання «20» « 05 » 2025р.

Здобувач _____
(підпис)

Керівник роботи _____ доц. Роксана МЕЛЬНІКОВА
(підпис) (посада, Власне ім’я, ПРІЗВИЩЕ)

РЕФЕРАТ

Пояснювальна записка до кваліфікаційної роботи бакалавра, 69 сторінок, 21 рисунок, 2 таблиці, 12 джерел посилань.

ВАКЦИНА, ДОВІДКА ПРО ЩЕПЛЕННЯ, КЛІЄНТСЬКА ЧАСТИНА, ЛІКАР, МЕДИЧНА ІНФОРМАЦІЙНА СИСТЕМА, МЕДИЧНИЙ ЗАКЛАД, МОБІЛЬНИЙ ЗАСТОСУНОК, ПАЦІЄНТ, ПУНКТ ВАКЦИНАЦІЇ, СЕРВЕРНА ЧАСТИНА, ХМАРНА БАЗА ДАНИХ, ЩЕПЛЕННЯ, JAVASCRIPT, KOTLIN, MICROSOFT AZURE, SQL SERVER.

Об’єкт розробки – програмна система для обліку проведення щеплень.

Мета розробки – створення програмної системи для обліку проведення щеплень, яка складається з серверної частини, клієнтської частини та мобільного програмного застосунку.

Метод рішення – середовища розробки IntelliJ IDEA, Android Studio, мови програмування JavaScript, Kotlin, SQL Server база даних.

В результаті розробки створено програмну систему для обліку проведення щеплень, яка складається з серверної частини, клієнтської частини та мобільного програмного застосунку, та яка автоматизує ключові процеси вакцинації, зменшує вплив людського фактору, спрощує взаємодію пацієнтів і лікарів.

ABSTRACT

VACCINE, VACCINATION CERTIFICATE, CLIENT-SIDE, DOCTOR, MEDICAL INFORMATION SYSTEM, MEDICAL INSTITUTION, MOBILE APPLICATION, PATIENT, VACCINATION POINT, SERVER-SIDE, CLOUD DATABASE, VACCINATION, JAVASCRIPT, KOTLIN, MICROSOFT AZURE, SQL SERVER.

Object of Development – a software system for vaccination record management.

Purpose of Development – to create a software system for vaccination record management, which consists of a server-side component, a client-side component, and a mobile application.

Method of Implementation – development environments IntelliJ IDEA and Android Studio, programming languages JavaScript and Kotlin, and SQL Server database.

As a result of the development, a software system for managing vaccination records was created. It consists of a server-side application, a client-side application, and a mobile application, and it automates key vaccination processes, reduces the impact of the human factor, and simplifies interaction between patients and doctors.

ЗМІСТ

ВСТУП

Поштовхом для виконання даної роботи стали проблеми та потреби людей та медичних закладів в сфері вакцинації. Головною проблемою в даній галузі є облік проведення щеплень. Перед нами поставлена задача в розробці актуальної та функціональної програмної системи, яка вирішує такі потреби користувачів, які ще не були вирішені конкурентами у вищезгаданій області.

Дана система відноситься до медичних інформаційних систем (МІС) [1] та пов’язана зі сферою медицини.

Програмна система складається з серверної, клієнтської частин та мобільного програмного застосунку. Для реалізації серверної частини обрано фреймворк Node.js [2] та мову програмування JavaScript. Для клієнтської частини обрано бібліотеку React.js та мову програмування JavaScript. Та для створення мобільного програмного застосунку обрано мову програмування Kotlin [3].

Для зберігання даних системи обрано реляційну модель бази даних Microsoft SQL Server. Використовується саме хмарна база даних Azure SQL Database.

Отже, метою даної роботи є створення програмної системи для обліку проведення щеплень, яка вирішує актуальні проблеми пацієнтів та медичних закладів, складається з серверної, клієнтської частин та мобільного застосунку.

1 АНАЛІЗ ПРЕДМЕТНОЇ ГАЛУЗІ

1.1 Аналіз предметної галузі

В сучасному світі вакцинація є дуже важливою для забезпечення здоров’я людини. Після пандемії COVID-19 люди зрозуміли, що щеплення та вакцини необхідні для боротьби з такими серйозними захворюваннями.

Щеплення – це процес створення штучного імунітету в людини, шляхом введення їй спеціального антигенного матеріалу. Вакцинація може проводитися різними шляхами: підшкірно, внутрішньом’язово, оральним способом тощо. [4]

Процес щеплення проводиться в процедурних кабінетах у лікарнях або в спеціальних пунктах щеплення. Після вакцинації в медичну картку пацієнта заносяться дані про виконану вакцину. При вже згаданій пандемії COVID-19 ми стикнулися із системою електронних сертифікатів про щеплення. Таким чином, в першу чергу інформація про вакцину заноситься в електронну базу даних, звідки вже можна отримати паперову довідку про вакцинацію.

Зберігання даних у форматі паперових довідок та карток не є дуже зручним, адже дані можуть втратитися або пошкодитися. В такому випадку сучасна система електронних довідок про щеплення є дуже зручною, адже таким чином можна завжди подивитися свої щеплення та отримати електронний витяг з бази.

На сьогоднішній день існують програмні системи та сервіси в галузі охорони здоров’я. Такі системи мають назву «медична інформаційна система» (МІС). Зазвичай, такі програми надають функціонал для декількох типів користувачів.

Наприклад, для звичайних користувачів сервіси надають функціонал для стеження за показниками здоров’я та формування порад для його покращення, для пошуку медичних закладів та запису на прийом до лікаря, власне для обліку проведення вакцинації тощо.

Також програмами можуть скористуватися як клініки, для автоматизації процесу роботи медичного закладу, так і приватні лікарі, для пошуку клієнтів,

проведення консультацій з ними в дистанційному форматі та запису на прийом онлайн.

Загалом, можна описати процес роботи цих програмних систем наступним чином. Спочатку користувач-пацієнт шукає заклад охорони здоров’я, куди він хоче прийти на прийом та до якого лікаря. Система надає користувачу можливі варіанти лікарів, які можуть його прийняти, а також інформацію про найближчу вільну дату та час. В цей момент до електронної реєстратури закладу охорони здоров’я заносяться дані про пацієнта, який записаний на прийом через програмну систему. В результаті пацієнт прибуває на прийом у вказаний день та час, а в його електронну медичну картку автоматично заноситься інформація про прийом. Дана система є дуже зручною як для пацієнтів, так і для клінік та лікарів, бо економить багато зусиль та часу. А електронна система забезпечить той факт, що дані та записи точно не будуть втрачені або пошкоджені, бо вони зберігаються в електронному форматі.

Розглянемо один із таких українських сервісів, як Helsi [5] (див. рис. 1.1).

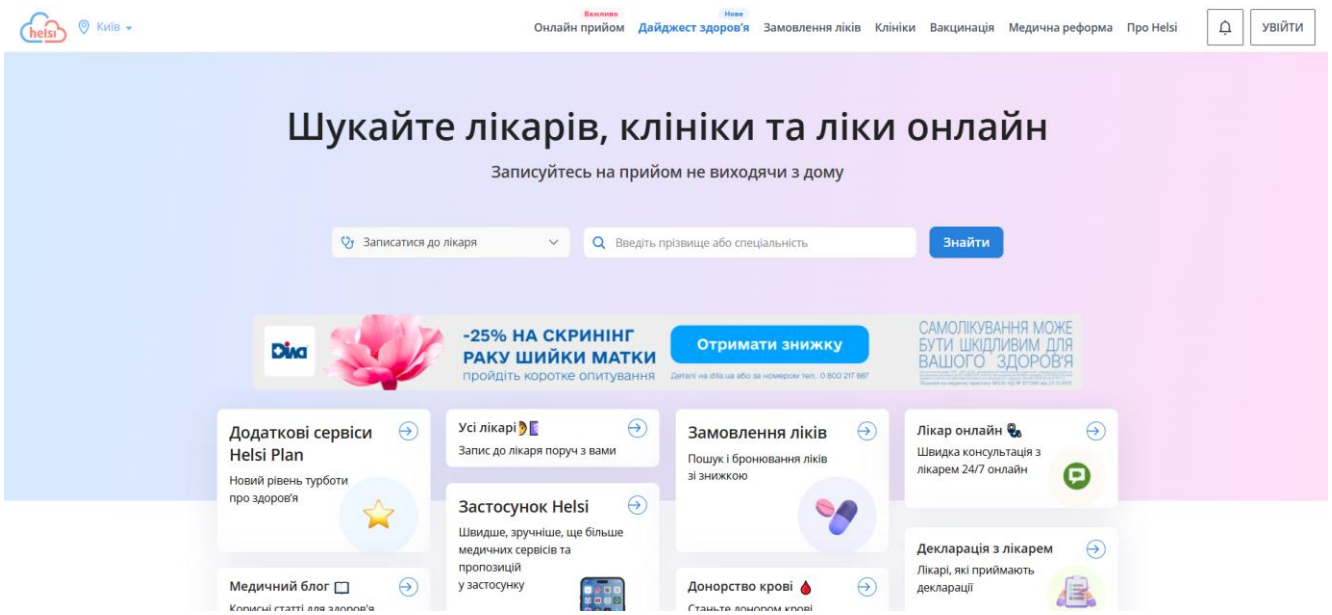

Рисунок 1.1 – Головна веб-сторінка сервісу Helsi (рисунок створений самостійно)

Даний сервіс надає можливості запису на прийом до лікаря, перегляду доступних клінік, замовлення ліків, зв’язку з лікарями та обліку проведення щеплень.

Helsi надає свої послуги для таких типів користувачів:

– пацієнти;
– приватні лікарі;
– медичні заклади.

Для пацієнтів надаються можливості запису на прийом, вибір лікаря, доступ до електронної медичної картки. Для приватних лікарів – особистий кабінет для ведення прийому пацієнта, доступ до даних діагностики та аналізів пацієнта. Та для медичних закладів надаються можливості повної автоматизації роботи закладу, формування звітності та статистики.

При аналізі даної програмної системи виділили такі переваги:

– доступ до центральної бази даних – офіційної державної бази даних в сфері охорони здоров’я;
– зручний процес авторизації користувачів в системі за номером телефону;
– інтуїтивно зрозумілий інтерфейс користувача;
– швидкий доступ до особистих даних в електронному кабінеті як для пацієнтів, так і для лікарів;
– повна автоматизація процесу роботи закладу охорони здоров’я зі сторони пацієнтів, лікарів та закладу охорони здоров’я вцілому.

І також були виділені такі недоліки сервісу:

– відсутність в системі закладів охорони здоров’я невеликих міст або селищ;
– обмежена робота з електронними довідками – система не надає можливість електронних довідок про прийом, про період непрацездатності або про зроблену вакцину.

Розглянемо функціонал, який стосується вакцинації більш детально. Helsi надає можливості запису на проведення щеплень, нагадування про вакцинацію

через спеціальний календар та перегляду вже зроблених щеплень. Пацієнт має доступ до інформації про вже зроблені щеплення, але він може тільки її переглядати. Як вже були вказано вище, в користувача немає можливості отримати електронну довідку або сертифікат про щеплення. Для отримання довідки в такому випадку користувачу потрібно звернутися до лікаря для отримання фізичної довідки про вакцинацію, що не є зручним.

Далі розглянемо іншу систему охорони здоров'я – eHealth [6] (див. рис. 1.2).

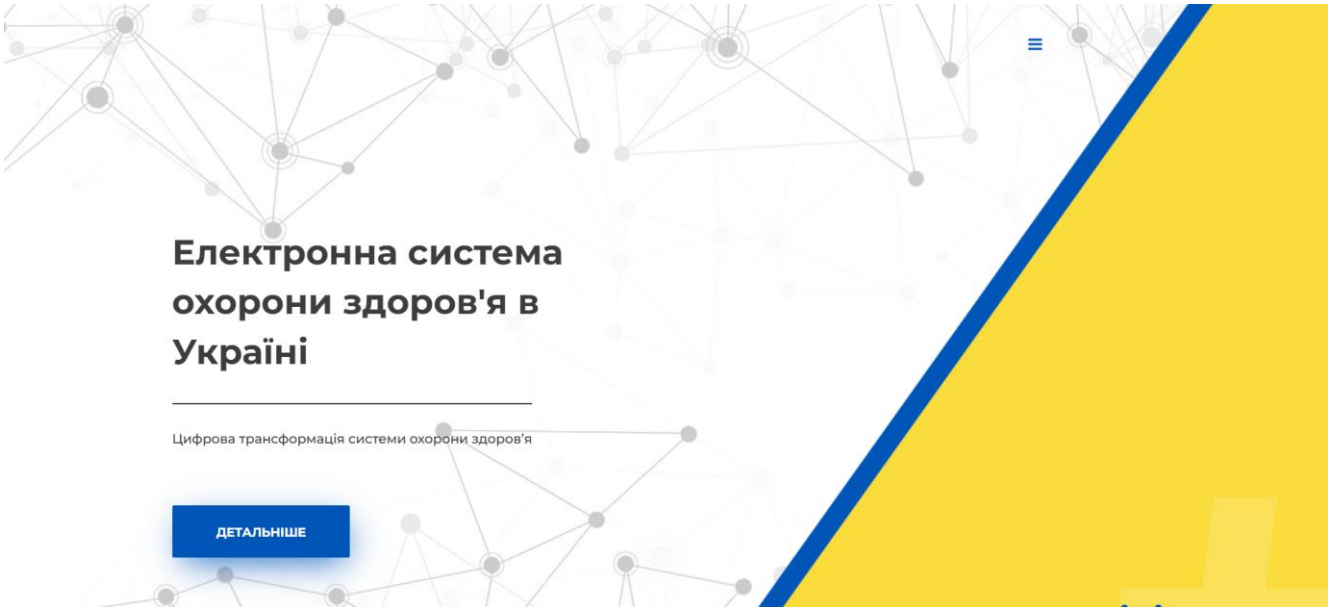

Рисунок 1.2 – Головна веб-сторінка системи eHealth (рисунок створений самостійно)

Дана програмна система надає таку ж функціональність, як й інші медичні інформаційні системи, а саме можливість інтеграції клінік з програмною системою для їх автоматизації. Також, як і система Helsi, надає доступ до електронних медичних реєстрів.

Але на відміну від Helsi, дана система також підтримує інтеграцію з аптечними закладами.

Головною перевагою даної системи є те, що вона надає більш гнучке налаштування для закладів охорони здоров'я. Якщо клініка скористається послугами eHealth, то вона отримає окрему платформу для свого закладу, яким

можна керувати, та налаштовувати самостійно, а платформа буде допомагати та сприяти цьому.

Дана перевага може стати й недоліком системи, адже, на відміні від Helsi, тут немає єдиного сервісу з реєстрацією користувачів, пошуком закладів охорони здоров'я, особистим кабінетом пацієнта або лікаря тощо.

Також треба звернути увагу на те, що дана система не сприяє роботі з електронними довідками та сертифікатами. Тому можна стверджувати, що система eHealth не може вести облік щеплень.

1.2 Виявлення та вирішення проблем

Так як в Україні та багатьох інших країнах існує проблема освіченості людей щодо щеплень, є необхідним створення програмного рішення для підвищення рівня знань щодо вакцинації. Дуже часто в дорослому віці людина може не знати, які щеплення вже були зроблені, тому такий програмний продукт необхідний для збору усієї інформації про вакцинацію в одному місці.

Однією з проблем та потреб пацієнтів є особистий час, адже вони не хочуть записуватися на прийом в медичний заклад фізично або чекати прийом в чергах. Але дана проблема вже має рішення в будь якій медичній інформаційній системі. Будь-яка така програмна система надає користувачам можливість онлайн запису на прийом до лікарні, на здачу аналізів, на процедури або на вакцинацію.

Паперові документи також на сьогоднішній день є проблемою, так як вони можуть зіпсуватися, пошкодитися, втратитися. Перехід на електронні документи, в тому числі й медичні, дуже спрощує життя людям. Медичні програмні системи вже перейшли на електронні медичні документи, так що пацієнти можуть подивитися свою медичну картку зі смартфона в будь який момент, переглянути результати аналізів або історію прийомів в лікарні.

Але не всі медичні документи на сьогоднішній день існують в електронному вигляді. Ті ж самі довідки про вакцинації або картки щеплень існують тільки у фізичному вигляді. І якщо людина загубила свою картку

щеплень, то їй потрібно звернутися до сімейного лікаря, який випишить нову картку, користуючись архівними даними та медичною карткою пацієнта. Це є проблемою та потребує програмного рішення.

Програмні системи також вже спрощують роботу й медичним закладам, адже тепер сучасні лікарні вже працюють здебільшого тільки на комп'ютерах та з електронними документами. Також такі програмні рішення допомагають частково або повністю автоматизувати роботу закладу охорони здоров'я.

Але загалом, проаналізувавши медичні інформаційні системи, можна помітити, що ні одна не забезпечує роботу з процесом вакцинації пацієнтів в повному обсязі, відповідно до проблем пацієнтів

Виділимо основні проблеми та потреби, які стосуються саме вакцинації:

- відсутність високого рівня освіченості людей, щодо вакцинації;
- паперові документи, сертифікати та довідки про щеплення;
- відсутність електронного помічника або порадника, який допомагав би з питань вакцинації;
- відсутність одного програмного рішення, яке в сукупності вирішило б усі потреби у сфері вакцинації.

Дані проблеми та потреби користувачів мають бути вирішені програмних шляхом.

1.3 Постановка задачі

Відповідно до виявлених проблем та потреб в сфері вакцинації, можна зробити висновок, що на сьогоднішній день необхідно розробити програмне рішення, яке б змогло автоматизувати процедуру вакцинації як зі сторони пацієнтів, так і зі сторони медичних закладів.

Необхідно пацієнтам надати доступ до програмного рішення як з комп'ютерів, так і зі смартфонів. Тому, необхідно розробити веб-застосунок та мобільний застосунок.

Програмна система має надавати можливості таким типам користувачів:

– користувачі-пацієнти;
– лікарі закладів охорони здоров’я та пунктів вакцинації, які будуть надавати послуги пацієнтам;
– адміністратори системи, які будуть працювати з даними система та загалом керувати нею.

Для користувача-пацієнта система має надавати наступний функціонал:

– особистий кабінет користувача, в якому він зможе переглядати та редагуввати особисті дані;
– запис на щеплення до обраного медичного закладу;
– перегляд історії прийомів на щеплення в медичних закладах;
– перегляд історії щеплень;
– формування електронної довідки про виконане щеплення;
– електронний помічник, інтегрований з нейромережею, який зможе відповідати на будь які питання, які пов’язані з вакцинацією.

Для лікарів медичних закладів програмна система надає такий функціонал:

– перегляд пацієнтів, записаних на вакцинацію;
– внесення даних про виконане щеплення пацієнта в систему;
– перегляд статистики в графічному вигляді.

Для адміністратора системи надається такий функціонал:

– реєстрація медичних закладів в системі, редагування, перегляд, видалення даних про них;
– реєстрація лікарів медичних закладів;
– перегляд статистики в графічному вигляді;
– перегляд даних системи.

Серверна частина реалізовує основну логіку роботи програми, а саме:

– робота з базою даних: вибірка, редагування, створення та видалення даних;
– прийом та обробка запитів від клієнтської та мобільної частини;
– відправлення запитів на клієнтську та мобільну частини;

– реалізація бізнес логіки.

Клієнтська частина надає веб-інтерфейс користувачам для роботи із системою. Мобільній застосунок надає інтерфейс для смартфонів з операційною системою Android.

Клієнтська та мобільна частини не будуть дублювати функціональність повністю. Зазвичай, від веб-версій програмних систем користувачі очікують більш широку функціональність, на відміну від мобільних застосунків, де надаються тільки окремі основі можливості сервісу.

Тому у клієнтській частині реалізовано функціональність в повному обсязі для усіх типів користувачів системи. Клієнтська частина надає веб-інтерфейс для таких функцій програмної системи:

– авторизація та реєстрація усіх типів користувачів;
– запис на прийом на щеплення для пацієнтів;
– перегляд записів на прийом на щеплення лікарями;
– можливість лікарям, як відповідальним особам, вносити дані про виконане щеплення в систему;
– перегляд історії виконаних щеплень пацієнтами;
– перегляд історії прийомів на щеплення пацієнтами;
– перегляд звітності та аналітики роботи медичного закладу лікарями;
– формування електронних довідок про щеплення пацієнтами;
– електронний помічник для пацієнтів для отримання відповідей на запитання щодо вакцинації;
– внесення, читання, редагування та видалення даних системи адміністраторами;
– реєстрація в системі медичних закладів та лікарів адміністраторами;
– доступ до статистики роботи програмної системи адміністраторами.

Мобільний застосунок створений тільки для використання користувачами-пацієнтами та надавати інтерфейс для таких можливостей:

– авторизація в системі;

– запис на прийом на щеплення в обраний заклад охорони здоров’я;

– формування електронної довідки про щеплення;

– перегляд історії записів на щеплення.

Отже, поставлена задача розробити програмну систему для обліку проведення щеплень, яка включає веб-застосунок та мобільний застосунок. Дана система вирішує головні проблеми та потреби людей у сфері вакцинації та надає функціональність для пацієнтів та лікарів медичних закладів.

2 ФОРМУВАННЯ ВИМОГ ДО ПРОГРАМНОЇ СИСТЕМИ

2.1 Функціональні вимоги

Як вже вказано в попередніх розділах, програмна система надає функціональні можливості для 3 типі користувачів: користувачі-пацієнти, лікарі закладів охорони здоров'я та адміністратори. Виділимо функціональні вимоги для кожного типу користувачів.

Функціональні вимоги програмної системи для користувачів пацієнтів:

- авторизація та реєстрація за допомогою електронної пошти та пароля;
- внесення та редагування особистих даних в електронний кабінет користувача;
- пошук та перегляд закладів охорони здоров'я;
- вибір закладу охорони здоров'я для запису на щеплення в доступну дату та час;
- перегляд історії прийомів на щеплення до закладів охорони здоров'я;
- перегляд історії проведених щеплень у вигляді списку;
- формування електронних довідок про щеплення;
- налаштування та отримання нагадувань при запис на щеплення електронною поштою;
- використання електронного помічника, інтегрованого з нейромережею для отримання відповідей на запитання щодо вакцинації.

Функціональні вимоги програмної системи для користувачів-лікарів:

- авторизація за допомогою електронної пошти та пароля;
- ведення індивідуального графіку прийому на щеплення пацієнтів;
- перегляд записаних на прийом на щеплення пацієнтів;
- виставлення відмітки про проведений прийом на щеплення пацієнта;
- внесення в базу даних інформації про проведене пацієнту щеплення;
- формування звітності про кількість прийнятих пацієнтів у вигляді графіку.

Функціональні вимоги програмної системи для адміністраторів:

– авторизація в системі за допомогою електронної пошти та паролю;
– реєстрація закладів охорони здоров’я;
– реєстрація лікарів та закріплення їх за закладом охорони здоров’я;
– перегляд, редагування та видалення даних про вакцини, лікарів та закладів охорони здоров’я;
– перегляд статистичних даних у вигляді графіків – кількість зареєстрованих користувачів за обраний проміжок часу.

2.2 Нефункціональні вимоги

Виділимо такі нефункціональні вимоги для програмної системи:

– забезпечення безпеки даних користувачів шифрування переданої інформації та захисту бази даних;
– забезпечення безперебійної роботи;
– мінімальний час відповіді системи на запити;
– доступність системи через усі сучасні браузери та мобільні пристрої на операційній системі Android.

2.3 Вимоги до середовища розробки та виконання

Виділимо вимоги щодо середовища розробки та виконання:

– для розробки серверної частини використовувати мову програмування JavaScript, фреймворк Node.JS та середовище розробки IntelliJ IDEA;
– для розробки клієнтської частини використовувати мову програмування JavaScript, бібліотеку React.JS та середовище розробки IntelliJ IDEA;
– для розробки мобільної частини використовувати мову програмування Kotlin та середовище розробки Android Studio;
– для зберігання даних системи використовувати реляційну модель бази даних з розгортання за допомогою хмарного сервісу Azure SQL Database;

– для розгортання та впровадження застосунку використовувати контейнери з використанням хмарного сервісу Azure.

2.4 Обмеження

Виділимо такі обмеження в роботі програмної системи:

– постійне підключення до мережі Інтернет для працездатності системи;
– підтримка роботи мобільного застосунку тільки для операційної системи Android версії 9.0 та вище;
– робота програмної системи тільки на території України на перших версіях.

3 АРХІТЕКТУРА ТА ПРОЄКТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

3.1 UML проєктування ПЗ

Відповідно до вимог ПЗ розробимо Use-case діаграми.

Відповідно до типів користувачів системи, виділимо таких акторів на діаграмі:

– пацієнт;

– лікар;

– адміністратор.

Створена Use-case діаграма для актора-пацієнта наведена на рисунку 3.1.

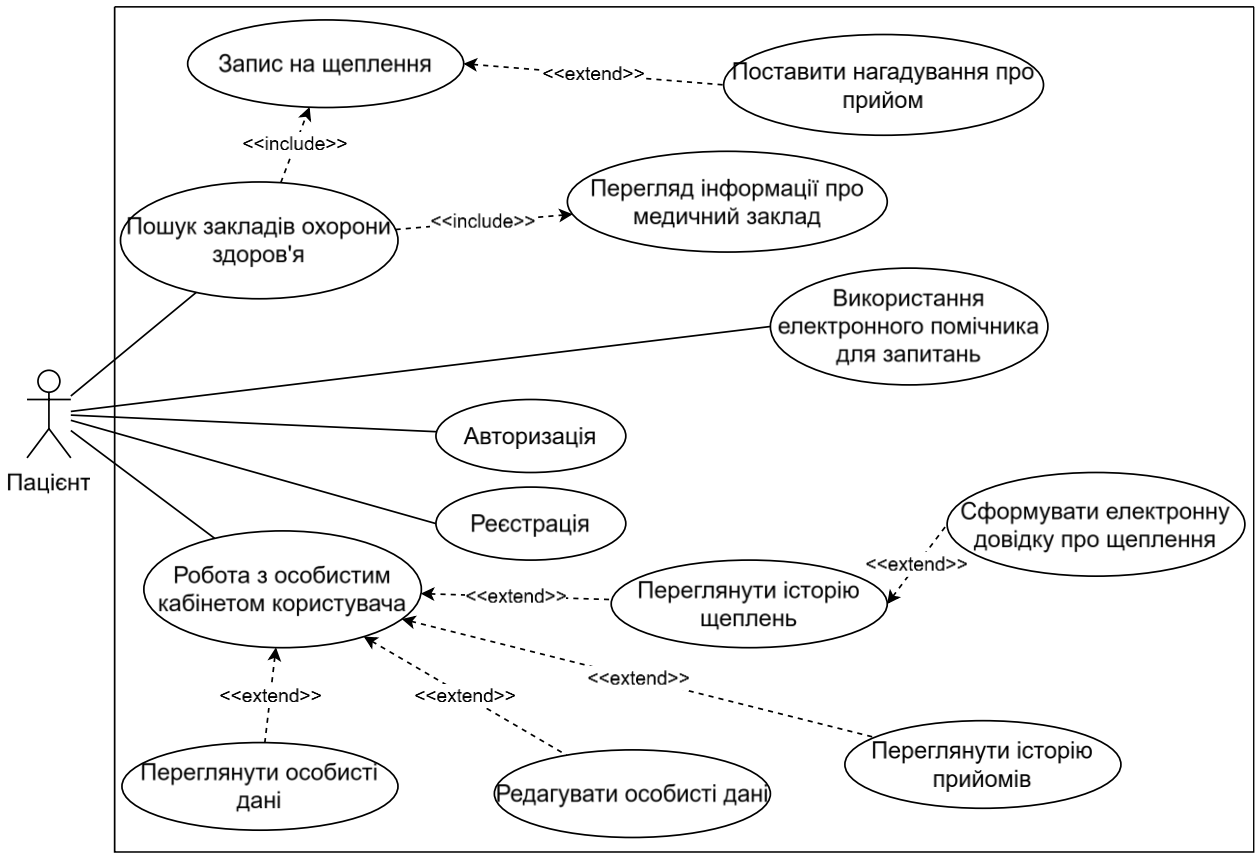

Рисунок 3.1 – Use-case діаграма для актора пацієнта (рисунок створений самостійно)

Далі на рисунку 3.2 наведено Use-case діаграму для актора-лікаря.

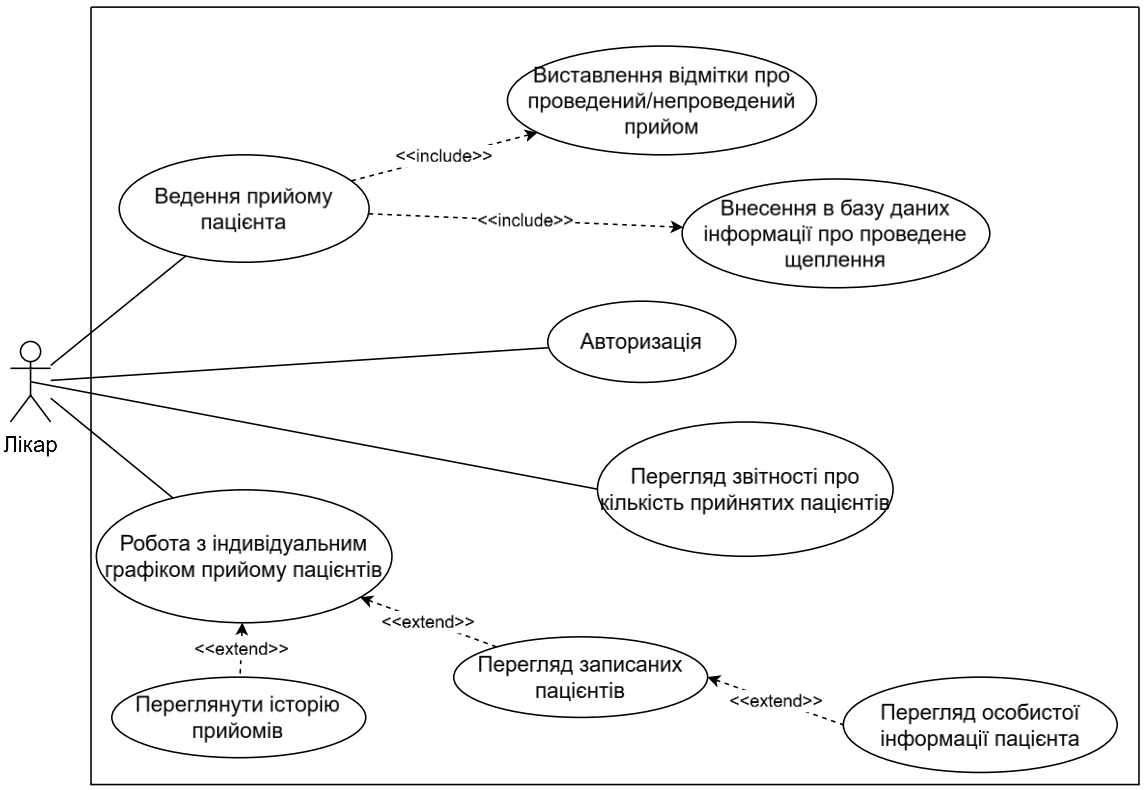

Рисунок 3.2 – Use-case діаграма для актора-лікаря (рисунок створений самостійно)

Далі на рисунку 3.3 наведено Use-case діаграму для актора-адміністратора.

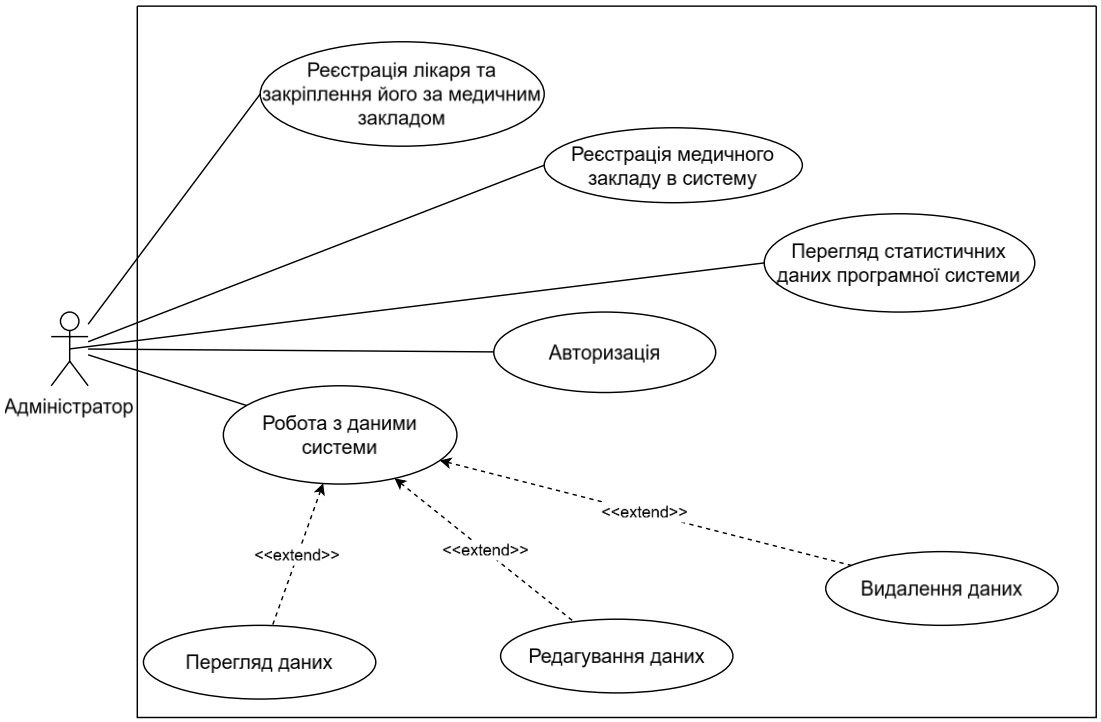

Рисунок 3.3 – Use-case діаграма для актора-адміністратора (рисунок створений самостійно)

Отже, ми побудували Use-case діаграми, які показують, як користувачі взаємодіють з програмною системою для обліку щеплень.

3.2 Проєктування архітектури ПЗ

Відповідно до вимог, необхідно розробити серверну, клієнтську частини та мобільний застосунок. Для зберігання даних використовується реляційна база даних.

Саме тому в якості архітектури програмної системи обрана багаторівнева клієнт-серверна архітектура.

Дана архітектура складається з трьох шарів:

– клієнтський шар;

– шар бізнес-логіки;

– шар зберігання даних.

Клієнтський шар представляє собою те, що бачить користувач – інтерфейс користувача. В нашому програмному забезпеченні в якості клієнтського шару виступають веб-застосунок та мобільний застосунок. Клієнтський шар напряму пов’язаний із шаром бізнес логіки. Клієнт відправляє запити на сервер та отримує від нього відповіді.

Шар бізнес логіки представляє серверну частину нашої програмної системи. Саме на цьому рівні відбувається обробка запитів від клієнта, звертання до бази даних для читання, запису, редагування та видалення.

Шар зберігання даних представляє собою власне базу даних, де зберігається інформація.

Отже, дана архітектура працює таким чином, що клієнт надсилає запит на сервер, де він оброблюється. Потім сервер звертається до бази даних та виконує запит. І в результаті сервер відправляє його на клієнта з результатами запиту, що виконується.

Взаємодія між рівнями програмної системи здійснюється за допомогою REST API, реалізованого у серверній частині. Веб-застосунок та мобільний

застосунок формують HTTP-запити (методи GET, POST, PUT, DELETE тощо) до серверного застосунку, який обробляє запити, виконує перевірку вхідних даних, реалізує відповідну бізнес-логіку та звертається до бази даних. У відповідь клієнти отримують структуровані дані у форматі JSON, що дозволяє ефективно відображати інформацію на клієнтській стороні.

Такий підхід дозволяє чітко розділити функціональність між рівнями системи, що відповідає сучасним принципах розробки програмного забезпечення.

REST-архітектура є незалежною від клієнтської реалізації, що забезпечує гнучкість веб-клієнт, мобільний застосунок або навіть сторонні сервіси можуть працювати з єдиним сервером через API, що спрощує підтримку та розвиток системи.

Клієнт-серверна архітектурна модель забезпечує високу модульність, адже кожен компонент (веб-, мобільний застосунок, сервер, база даних) може розвиватися незалежно. Завдяки масштабованості, у разі зростання кількості користувачів або обсягу даних, можливе горизонтальне або вертикальне масштабування серверної частини чи бази даних.

Зручність у супроводі та модернізації досягається через чітке розділення обов'язків: клієнт відповідає за інтерфейс користувача, сервер за логіку та обробку запитів, а база даних – за збереження інформації. Це дозволяє легко оновлювати або замінювати окремі частини системи без впливу на інші.

Завдяки застосуванню такої архітектури програмна система зберігає високу стійкість до збоїв, надійність у роботі навіть при зростанні навантаження, а також забезпечує легкість масштабування та модифікації функціоналу. Це робить систему придатною до подальшого розвитку, впровадження нових можливостей та безперешкодної інтеграції з зовнішніми сервісами або платформами в майбутньому.

Відобразимо взаємодію архітектурних частин програмної системи за допомогою діаграми розгортання [7], наведеній на рисунку 3.4.

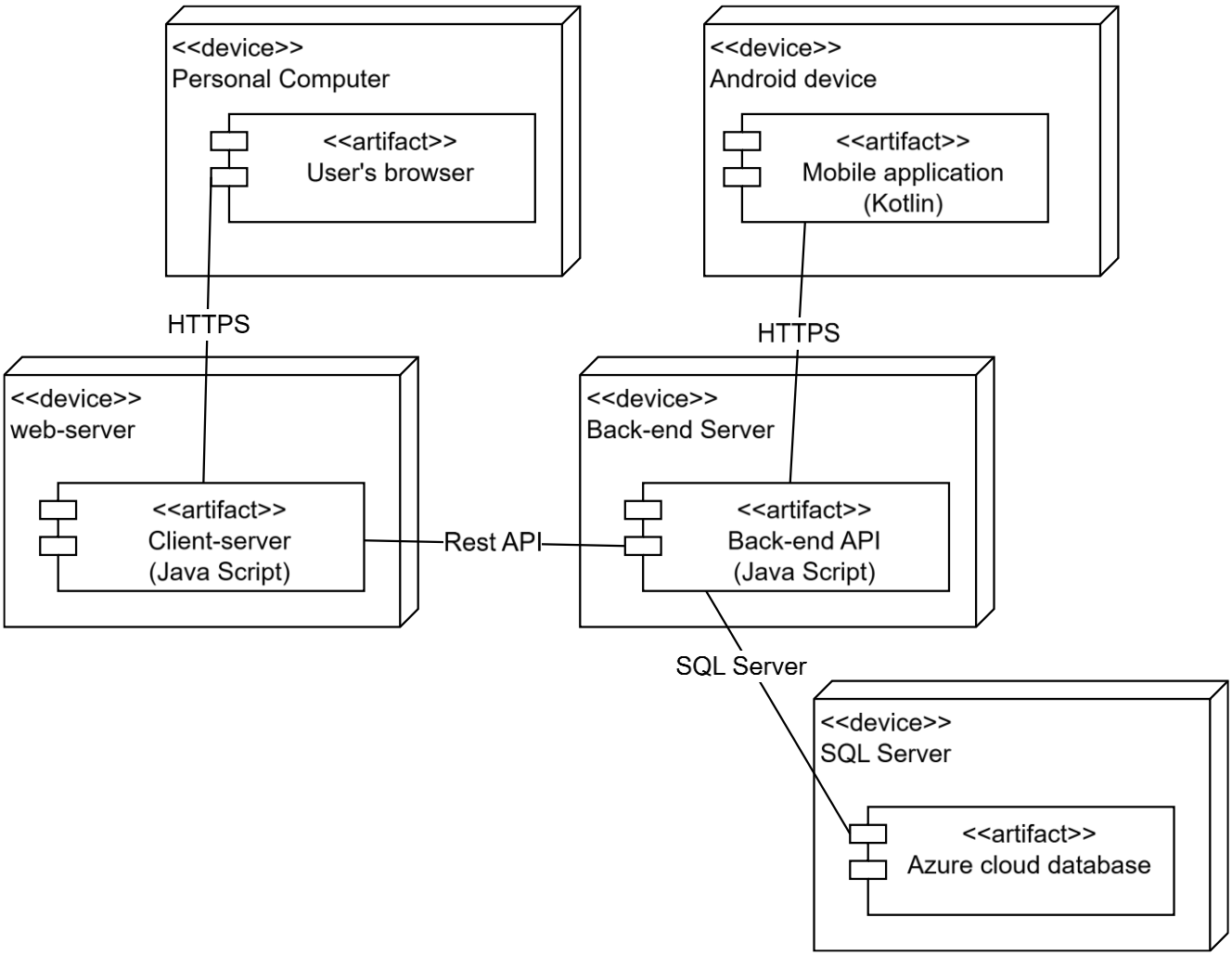

Рисунок 3.4 – Діаграма розгортання (рисунок створений самостійно)

Отже, спроєктовано та описано архітектуру програмної системи для обліку проведення щеплень.

3.3 Проєктування структури зберігання даних

Програмна система для обліку проведення щеплень працює з великою кількістю різноманітних даних. Система має оброблювати особисті дані користувачів, такі як ім’я, прізвище, дата народження тощо. Також система працює з даними для авторизації користувачів в системі – пацієнтів, лікарів, адміністраторів. Для цього необхідна електронна пошта та пароль користувача. На даному етапі треба забезпечити безпеку даних користувачів, щоб

адміністратор або розробник програмної системи не зміг отримати доступ до паролів користувачів.

Також система має працювати з даними про клініки. Для медичних закладів необхідно зберігати інформацію про назву медичного закладу та його розташування. Потрібно забезпечити прив’язку лікарів до медичних закладів.

Необхідно забезпечити зберігання та обробку інформації про записи на щеплення. Для записів необхідно зберігати дату, статус, назву вакцини, а також забезпечити зв’язки між пацієнтами, лікарями та відповідними їм записами на прийоми.

Також користувач повинен мати можливість ознайомитися з доступними вакцинами. Тому потрібно зберігати інформацію про різноманітні щеплення – назву, дозування, опис тощо.

Отже, в якості способу для зберігання даних системи обрано реляційну модель бази даних SQL [8]. Такий вибір є обґрунтованим з урахуванням особливостей предметної галузі, вимог до структури даних та безпеки.

Реляційні бази даних краще підходять для інформаційних систем, що мають чітку структуру даних, зв'язки між сутностями та потребу у забезпеченні цілісності даних. У програмній системі для обліку щеплень чітко визначені сутності: пацієнти, лікарі, заклади охорони здоров'я, записи на щеплення, вакцини, нагадування. Між цими об'єктами існують однозначні зв'язки «один-до-багатьох» або «багато-до-одного», які дуже ефективно реалізуються саме в реляційній моделі.

Додатковим аргументом є наявність стандартної мови запитів SQL, яка дозволяє легко реалізовувати складні вибірки, об'єднання, фільтрацію та агрегацію даних для формування звітності, історій щеплень та аналітики що є важливою функціональною частиною системи.

Також прийнято рішення використовувати саме хмарну базу даних Azure SQL Database [9]. Хмарна база даних забезпечує високу доступність сервісу, автоматичне резервне копіювання, оновлення, а також можливість масштабування в майбутньому.

В якості СУБД використано Microsoft SQL Server Management Studio, яка дуже добре працює з SQL Server базою даних. Дана СУБД має розвинені механізмі забезпечення:

– цілісності даних (через первинні та зовнішні ключі);

– безпеки (рольова модель доступу);

– надійності (транзакції, журнали змін);

– масштабованості та продуктивності (особливо у хмарному середовищі, як-от Azure SQL Database).

Отже, обраний підхід базується на поєднанні чіткої логічної структури системи, вимог до надійності, безпеки та доступності даних, що робить реляційну модель бази даних найбільш доцільною для реалізації нашої програмної системи.

Спроєктуємо та розробимо схему бази даних. Виділимо такі сутності в нашій програмній системі:

– patient (пацієнт);

– doctor (лікар);

– admin (адміністратор);

– appointment (прийом);

– clinic (медичний заклад);

– vaccine (вакцина);

– inoculation (щеплення);

– reminder (нагадування).

Виділимо атрибути для кожної сутності.

Сутність «пацієнт» має такі атрибути:

– id (ідентифікатор);

– name (ім’я);

– lastname (прізвище);

– phone_number (номер телефону);

– email (електронна пошта);

– password (пароль);

– birthday (дата народження);
– gender (стать);
– registation_date (дата реєстрації).

Сутність «лікар» має такі атрибути:

– id (ідентифікатор);
– name (ім’я);
– lastname (прізвище);
– phone_number (номер телефону);
– email (електронна пошта);
– password (пароль);
– registration_date (дата реєстрації);
– office_number (номер кабінету лікаря).
– clinic_id (ідентифікатор медичного закладу, до якого належить лікар).

Сутність «адміністратор» має такі атрибути:

– id (ідентифікатор);
– name (ім’я);
– lastname (прізвище);
– email (електронна);
– password (пароль).

Сутність «прийом» має такі атрибути:

– id (ідентифікатор);
– patient_id (ідентифікатор пацієнта, що записаний);
– doctor_id (ідентифікатор лікаря, до якого записаний пацієнт);
– datetime (дата і час запланованого прийому);
– status (статус запису: заплановано, скасовано, проведено).

Сутність «медичний заклад» має такі атрибути:

– id (ідентифікатор);
– name (назва);
– city (місто, в якому розташований заклад);

– address (адреса медичного закладу);
– description (додатковий опис щодо закладу).

Сутність «вакцина» має такі атрибути:

– id (ідентифікатор);
– name (назва);
– repetitions_number (кількість доз, які потрібно ввести);
– description (опис вакцини).

Сутність «щеплення» має такі атрибути:

– id (ідентифікатор);
– appointment_id (ідентифікатор прийому, на якому буде проведено щеплення);
– vaccine_id (ідентифікатор вакцини, яка буде застосована);
– dose_number (номер дози в рамках курсу вакцинації — наприклад, 1-ша чи 2-га);
– status (статус щеплення: виконано/не виконано).

Сутність «нагадування» має такі атрибути:

– id (ідентифікатор нагадування);
– appointment_id (ідентифікатор пов’язаного запису на щеплення);
– datetime (дата та час, коли має бути відправлено нагадування);
– status (статус нагадування: виконано, невиконано).

Між зазначеними сутностями встановлюються різного типу зв’язки, які відображають логіку роботи системи. Наприклад, один пацієнт може мати багато записів на прийом, а кожен запис пов’язаний із конкретним лікарем – це реалізується через зв’язки типу "багато до одного" Аналогічно, один медичний заклад може мати декількох лікарів, що також відображається як "багато до одного". Такі зв’язки дозволяють забезпечити цілісність даних, уникнути дублювання інформації та ефективно реалізувати запити до бази даних.

Відобразимо дані сутності, їх атрибути та зв’язки на ER-діаграмі бази даних (див. рис. 3.5).

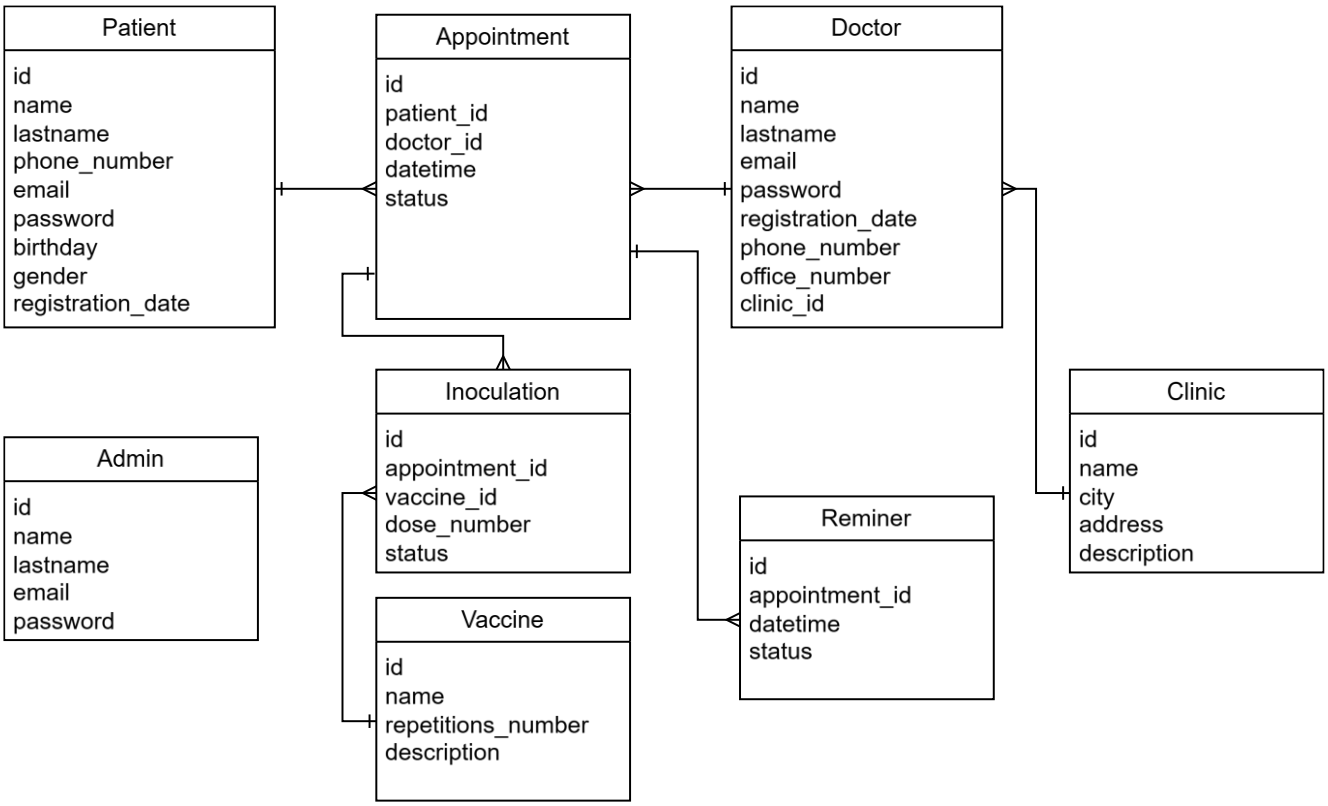

Рисунок 3.5 – ER-діаграма бази даних (рисунок створений самостійно)

Отже, спроєктовано та розроблено ER-діаграму бази даних, яка показує взаємодію між сутностями та атрибутами бази даних.

3.4 Приклади найцікавіших алгоритмів та методів

Наведемо опис трьох ключових логічних процесів, реалізованих в системі. Алгоритми реалізовані на серверній та клієнтський частині з використанням деяких бібліотек мови програмування JavaScript.

Алгоритм формування електронної довідки про виконане щеплення має такі кроки для виконання:

- користувач запитує створення довідки в особистому кабінеті;
- система перевіряє, чи є виконані щеплення у таблиці «Вакцинація»;
- вибирається останній (або конкретний) запис щеплення з усіма даними (пацієнт, вакцина, лікар, дата);
- на основі цих даних формується PDF-файл із довідкою;

– файл автоматично завантажується на пристрій користувача.

Для реалізації алгоритму застосовується бібліотека «pdfkit» для генерації PDF-документу та вбудовані в Node.JS модулі «fs», «promises» для збереження довідки.

Наступний алгоритм для перевірки доступності дати та часу запису до лікаря на прийом має такі кроки:

– коли пацієнт обирає лікаря, дату й час, система виконує перевірку;

– у таблиці «Прийом» виконується SQL-запит: чи існує запис з обраним ідентифікатором лікаря та датою з часом, у якого статус "заплановано"?;

– якщо запис знайдено – повертається повідомлення, що час зайнятий;

– якщо ні – система дозволяє зберегти новий запис.

Для реалізації використовується бібліотека «Express.js» для оброки запитів та «Sequelize» для запитів до бази даних.

Програмний код даного методу наведено далі.

```
const getAvailableSlots = async (req, res) => {
    const { date, doctor_id } = req.query;

    try {
        const allSlots = [];
        const baseDate = moment.utc(date).startOf('day').hour(9);
        for (let i = 0; i <= 11; i++) {
            const slot = baseDate.clone().add(i * 30, 'minutes');
            allSlots.push(slot.format('YYYY-MM-DD HH:mm:ss'));
        }

        const appointments = await Appointment.findAll({
            where: {
                doctor_id,
                status: 'заплановано',
                [Op.and]: [
                    where(fn('CONVERT', literal('date'), col('datetime')),
'=', date)
                ]
            }
        });

        const takenSlots = new Set(
            appointments.map(appt =>
                moment.utc(appt.datetime).format('YYYY-MM-DD HH:mm:ss')
            )
        );
```

```
        const availableSlots = allSlots.filter(slot =>
!takenSlots.has(slot));

        res.status(200).json(availableSlots);
    } catch (error) {
        console.error('Помилка при отриманні вільних слотів:', error);
        res.status(500).json({ error: 'Внутрішня помилка сервера' });
    }
};
```

Та третім цікавим алгоритмом є алгоритм надсилання електронного нагадування пацієнту. Даний алгоритм має такі кроки:

– система періодично перевіряє таблицю «Нагадування» на наявність записів, дата й час яких відповідає поточному;
– виконується перевірка статусу нагадування – надсилалося чи ні;
– якщо повідомлення ще не було відправлено, система формує листа з даними про майбутній прийом на щеплення: дата, час, адреса клініки, ім’я лікаря;
– лист надсилається на електронну пошту користувача;
– після успішної відправки статус нагадування оновлюється на "виконано".

Для реалізації цього алгоритму використовується служба Azure Communication Services, яка дозволяє надсилати електронні листи. Також застосовуються модулі Node.js cron для періодичного запуску перевірки та бібліотека «azure/communication-email» для роботи з email API.

3.5 Створення дизайну програмної системи

Під час проєктування інтерфейсу користувача враховано необхідність забезпечення зручної, інтуїтивно зрозумілої та адаптивної взаємодії з програмною системою для всіх типів користувачів: пацієнтів, лікарів та адміністраторів.

В клієнтський частині інтерфейс користувача реалізовано за допомогою мови розмітки HTML та стилів CSS. В мобільному застосунку інтерфейс створено за допомогою конструктора в Android Studio та мови розмітки XML.

У структурі інтерфейсу виділено кілька основних модулів:

– робоче середовище пацієнта має посилання на усі доступні йому функції;
– особистий кабінет містить дані про пацієнта, можливість редагування;
– сторінка запису на щеплення дає змогу знайти та обрати медичний заклад, вакцину, дату та час прийому на щеплення;
– інтерфейс лікаря містить список записаних пацієнтів, функцію фіксації проведених щеплень;
– інтерфейс адміністратора – дозволяє управляти медичними закладами, лікарями, переглядати статистику та модифікувати дані системи.

В якості кольорів інтерфейсу обрано білий та відтінки блакитного, в якості основних елементів інтерфейсу. Саме дані кольори добре підходять для медичної програмної системи, адже вони асоціюються в користувача з чистотою, безпекою та порядком.

Кольорову схему інтерфейсу наведено на рисунку 3.6.

Рисунок 3.6 – Кольорова схема інтерфейсу (рисунок створений самостійно)

Кольором шрифту обрано чорний, тому що він добре контрастує з відтінками блакитного.

Для кнопок, які відображають негативний стан та для інформування про помилки обрано червоний колір, так як він є класичним для таких елементів інтерфейсу.

Таким чином, був спроєктований дизайн інтерфейсу програмної системи з дотриманням стандартних вимог відповідно до розробки інтерфейсу користувача [10].

4 ОПИС ПРИЙНЯТИХ ПРОГРАМНИХ РІШЕНЬ

4.1 Вибір технологій та середовища розробки

Програмна система для обліку проведення щеплень складається з клієнтської, серверної частини та мобільного застосунку для операційної системи Android.

Так як клієнтська частина представляє собою веб-застосунок, то для розробки доцільно використовувати мову програмування JavaScript. Дана мова програмування дуже підходить саме для веб-розробки, надає велику кількість фреймворків та бібліотек для реалізації будь який програмних рішень. JavaScript підтримує розробку як веб-застосунків, так і серверних застосунків.

В якості фреймворків для розробки використано Node.JS для серверної частини та React.JS для клієнтської частини. Дані фреймворки добре працюють разом при розробці програмний систем.

Для створення веб-сторінок та їх оформлення використано мову гіпертекстової розмітки HTML та каскадні таблиці стилів CSS. Використання даних інструментів є обов’язковим при розробці веб-сторінок.

Для реалізації мобільного застосунку використано мову програмування Kotlin для написання програмного коду та мову розмітки XML для створення зовнішнього вигляду мобільного інтерфейсу.

В якості середовища для розробки клієнтської та серверної частин використано IntelliJ IDEA. Дане середовище розробки підтримує роботу майже з будь якими мовами програмування, в тому числі й JavaScript, та з великою кількістю фреймворків та бібліотек.

Для розробки мобільного застосунку для операційної системи Android використано Android Studio. Дане середовище розробки на сьогоднішній день є найкращим для розробки Android-застосунків.

Як вже описано в попередньому розділі, для зберігання даних програмної системи використано хмарну реляційну базу даних Azure SQL Server. В якості системою управління базою даних використано Microsoft SQL Server Management

Studio. Дана СУБД добре працює з реляційними базами даних типу SQL Server, має зручний інтерфейс для створення, перегляду та редагування таблиць та виконання запитів до БД.

4.2 Робота з базою даних

Розробка програмної системи для обліку проведення щеплень було почато зі створення сховища зберігання даних. В моєму обліковому записі сервісу Microsoft Azure створено SQL сервер, де розгортається база даних (див. рис. 4.1).

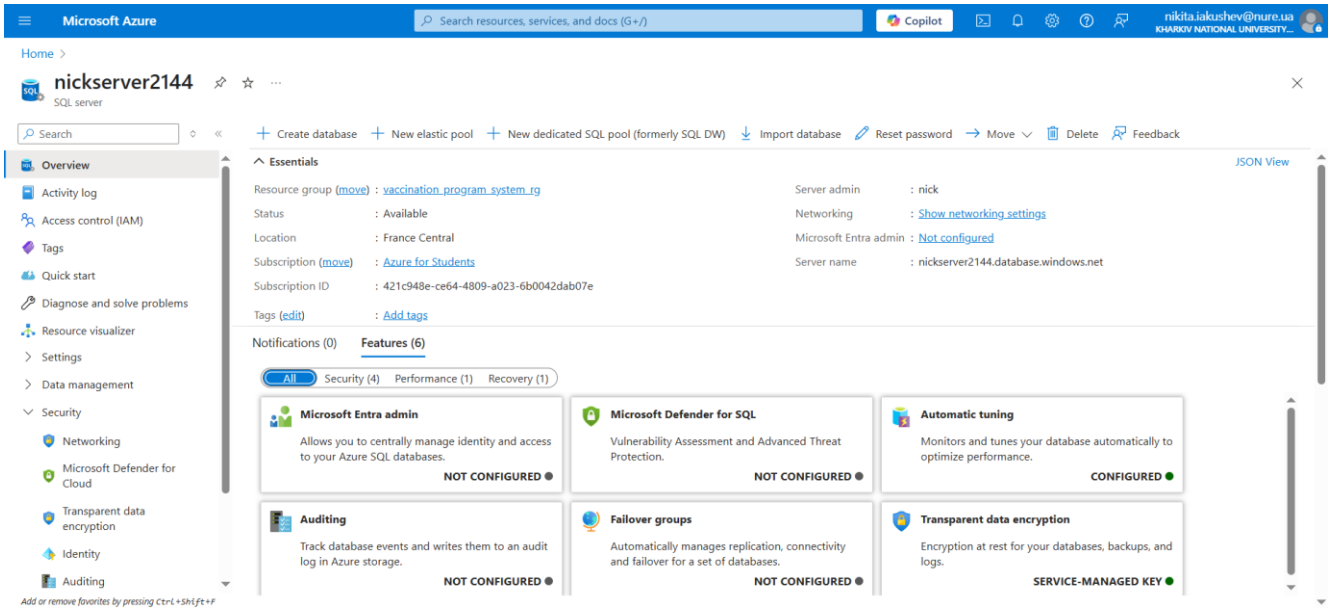

Рисунок 4.1 – Створений сервер бази даних (рисунок створений самостійно)

Створена база даних SQL, яка розміщена на даному сервері, наведена на рисунку 4.2.

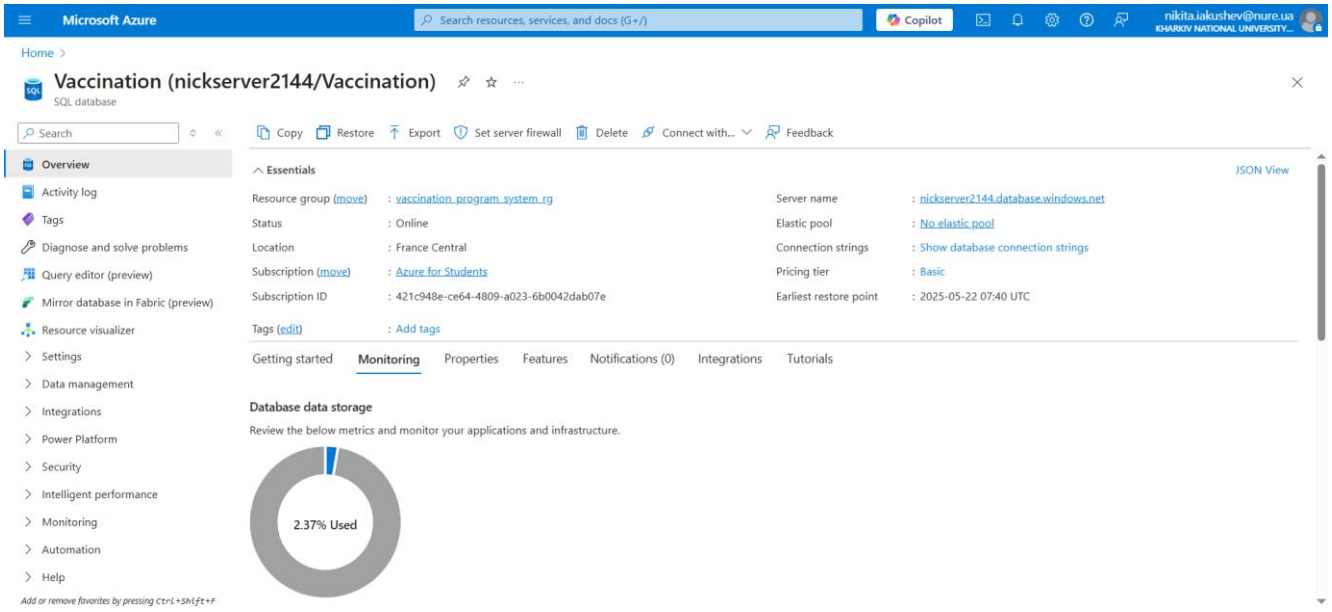

Рисунок 4.2 – Створена база даних SQL (рисунок створений самостійно)

Далі за допомогою СУБД Microsoft SQL Management Studio створено таблиці бази даних та зв’язки між ними (див. рис. 4.3).

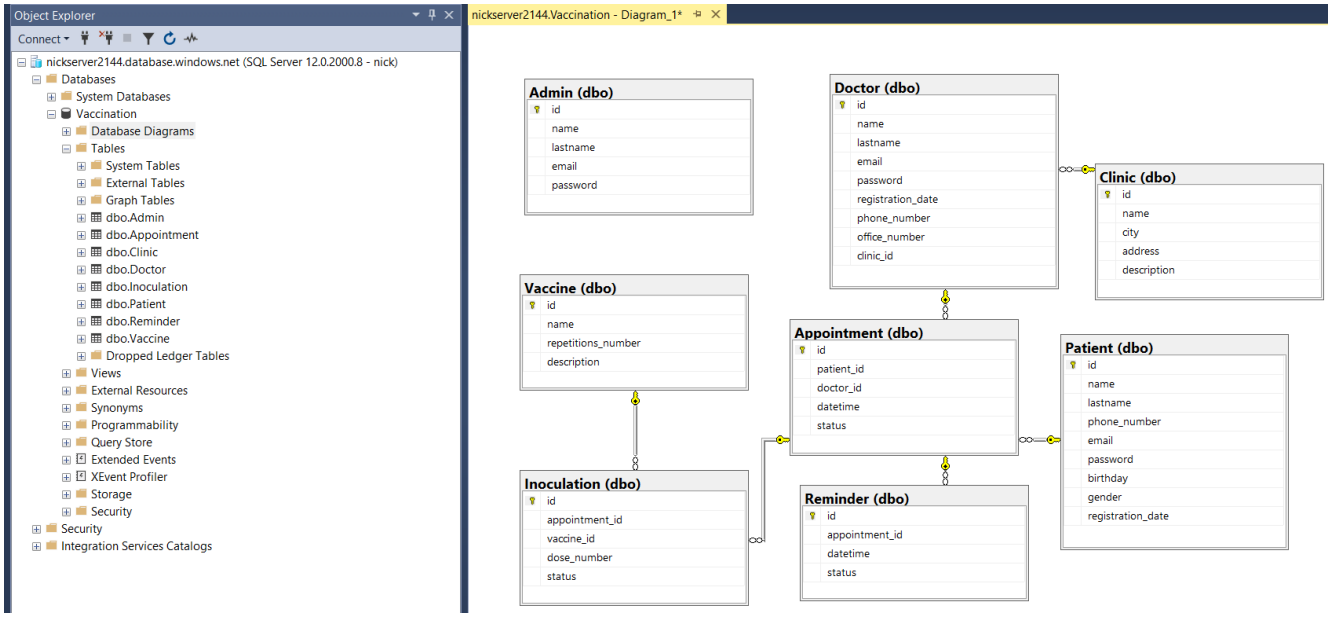

Рисунок 4.3 – Створені таблиці бази даних та схема зв’язків (рисунок створений самостійно)

Таким чином, реалізовано сховище зберігання даних програмної системи для обліку проведення щеплень.

4.3 Реалізація серверної частини

Для реалізації серверної частини створено застосунок на фреймворку Node.JS. Файлову структуру проєкту розділено за ролями (див. рис. 4.4).

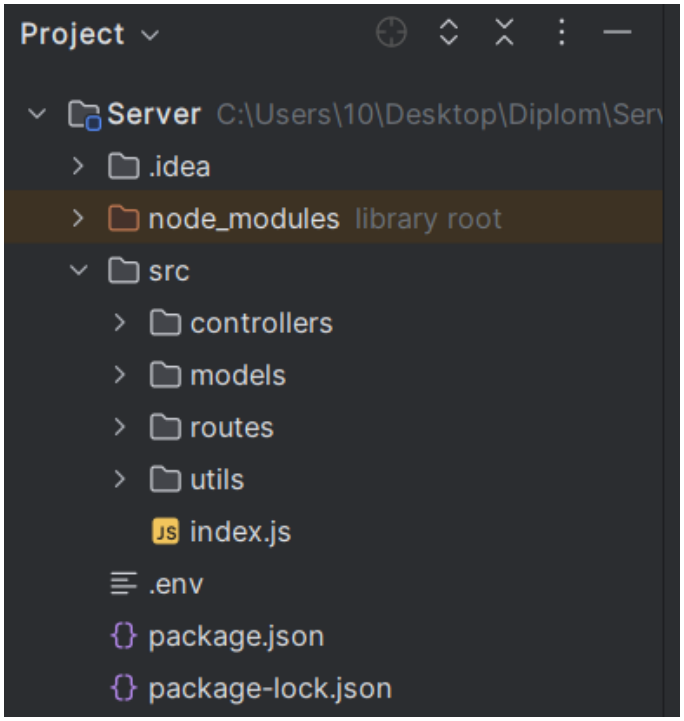

Рисунок 4.4 – Файлова структура серверного застосунку (рисунок створений самостійно)

В даній структурі маємо головний файл, який запускає сервер, – «index.js». Даний файл знаходиться в корені проєкту.

Маємо такі папки в структурі:

– «controllers» – контролери, які відповідають за бізнес логіку серверної частини;
– «models» – моделі, відповідають за сутності бази даних та відношення між ними;
– «routes» – маршрути, відповідають за маршрутизацію в серверній частині;

– «utils» – додаткові функції та налаштування в застосунку, наприклад підключення до бази даних або налаштування відправлення електронних листів.

Дана файлова структура серверної частини забезпечує чіткий поділ відповідальностей та зручність у підтримці коду. Контролери містять бізнес-логіку, що робить її ізольованою від деталей реалізації маршрутизації та моделювання даних. Моделі відповідають за структуру бази даних і дозволяють зручно взаємодіяти з нею за допомогою ORM. Маршрути організовують обробку HTTPS-запитів та забезпечують зрозумілу навігацію між частинами API. Додаткові утиліти виносять окрему технічну логіку, як-от підключення до бази даних або конфігурації. Такий підхід робить серверну частину масштабованою, зрозумілою та легкою для розширення.

Для підключення до хмарної бази даних використовується бібліотека «sequelize». Для цього створено окремий файл db.js, в якому ми вказуємо сервер, до якого створюється підключення, ім’я користувача та пароль та інші налаштування для підключення. Програмний код файлу db.js наведено далі.

```
const { Sequelize } = require('sequelize');

const sequelize = new Sequelize(
    process.env.DB_NAME,
    process.env.DB_USER,
    process.env.DB_PASSWORD,
    {
        host: process.env.DB_SERVER,
        dialect: 'mssql',
        timezone: '+03:00',
        dialectOptions: {
            encrypt: true
        },
        logging: false
    }
);

module.exports = sequelize;
```

Для правильного використання даної бібліотеки також необхідно визначити моделі, в яких треба прописати усі атрибути та типи даних, які є у відповідній

таблиці бази даних. Приклад програмного коду моделі Appointment, яка визначає сутність запису на прийом наведено далі.

```
const { DataTypes, Model } = require('sequelize');
const sequelize = require('../utils/db');

class Appointment extends Model {}

Appointment.init({
    id: {
        type: DataTypes.INTEGER,
        primaryKey: true,
        autoIncrement: true
    },
    patient_id: {
        type: DataTypes.INTEGER,
    },
    doctor_id: {
        type: DataTypes.INTEGER,
    },
    datetime: {
        type: DataTypes.DATE,
    },
    status: {
        type: DataTypes.STRING,
        validate: {
            isIn: [['заплановано', 'скасовано', 'проведено']]
        }
    }
}, {
    sequelize,
    modelName: 'Appointment',
    tableName: 'Appointment',
    timestamps: false
});

module.exports = Appointment;
```

Далі дані моделі використовуються в контролерах для створення, вибірки, редагування чи видалення інформації. Далі наведемо функцію getAll з файлу «appointmentController» для вибірки усіх записів про прийом.

```
const getAll = async (req, res) => {
    try {
        const appointments = await Appointment.findAll({
            include: [
                {
                    model: Patient,
                    as: 'patient'
                },
```

```
                {
                    model: Doctor,
                    as: 'doctor'
                }
            ]
        });
        res.json(appointments);
    } catch (err) {
        console.error('Помилка при отриманні прийомів:', err);
        res.status(500).send('Помилка сервера');
    }
};
```

Далі функції з контролерів використовуються для створення кінцевих точок, до яких може звертатись клієнтська або мобільна частини.

Приклад звернення до кінцевої точки «/appointments» за методом GET та результат його виконання наведено на рисунку 4.5.

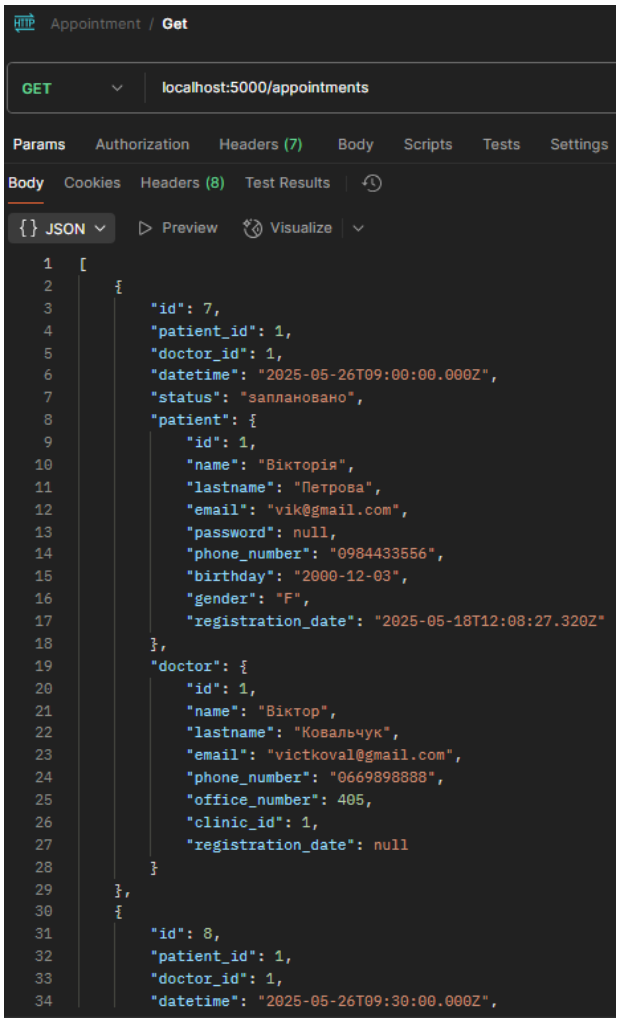

Рисунок 4.5 – Результат отримання усіх записів на прийом (рисунок створений самостійно)

Таким чином реалізовано усі кінцеві точки для виконання запитів GET, POST, PATCH, DELETE.

4.4 Реалізація клієнтської частини

Для реалізації клієнтської частини створено проєкт за допомогою бібліотеки React.JS. Використано типову файлову структуру React-проєкту, яка розділяє його на компоненти, сторінки, та стилі CSS (див. рис. 4.6).

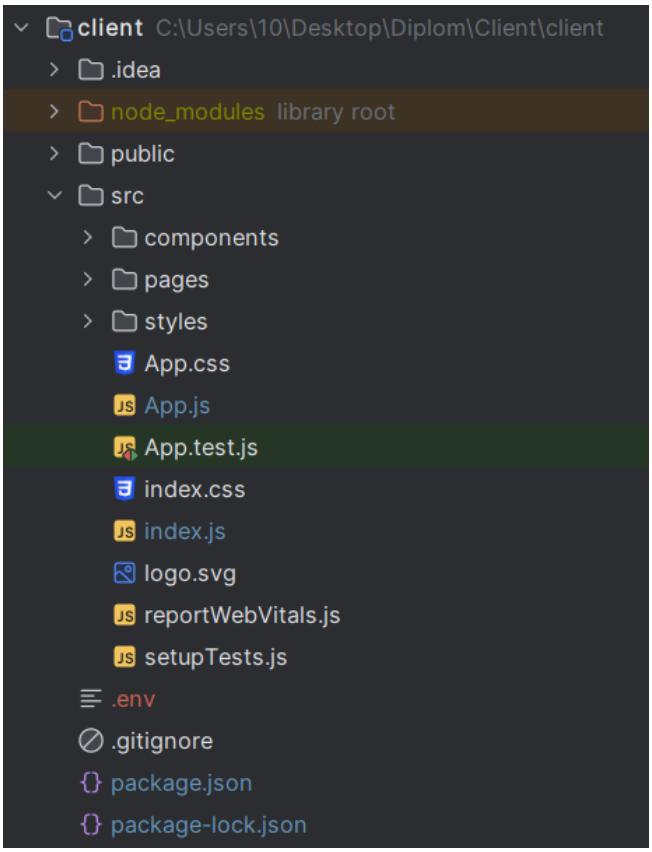

Рисунок 4.6 – Файлова структура клієнтської частини (рисунок створений самостійно)

В даній файловій структурі виділено наступні папки:

– «components» – програмні елементи інтерфейсу, які використовуються більше одного разу на веб-сторінці;

– «pages» – самостійні веб-сторінки, які бачить користувач;

– «styles» – стилі CSS для покращення зовнішнього вигляду веб-сторінок.

Обрана структура клієнтської частини на основі React забезпечує зручність розробки завдяки компонентному підходу, який дозволяє повторно використовувати елементи інтерфейсу. Розділення проєкту на папки components, pages та styles сприяє кращій організації коду та легшій навігації в межах застосунку. Такий підхід спрощує масштабування проєкту та пришвидшує розробку нових функцій. Винесення стилів до окремої папки забезпечує централізоване керування зовнішнім виглядом сторінок. Загалом, це дозволяє покращити підтримку коду та забезпечити більшу гнучкість у розробці інтерфейсу користувача.

Усі реалізовані сторінки та компоненти складаються з двох частин. Перша частина представляє собою стрілкову функцію, в якій реалізується звернення до серверу та обробка запитів. Тобто дана функція й представляє собою React-компонент. Приклад програмного коду для компонента «LoginPage.js» наведено далі.

```
const LoginPage = () => {
    const [email, setEmail] = useState("");
    const [password, setPassword] = useState("");
    const [error, setError] = useState("");

    const handleLogin = async (e) => {
        e.preventDefault();
        setError("");

        try {
            const response = await
fetch(`${process.env.REACT_APP_BASE_URL}/patients/login`, {
                method: "POST",
                headers: {
                    "Content-Type": "application/json",
                },
                body: JSON.stringify({ email, password }),
            });

            if (!response.ok) {
                const message = await response.text();
                throw new Error(message);
            }

            const data = await response.json();
            const { patient, token } = data;
```

```
            localStorage.setItem("token", token);
            localStorage.setItem("userId", patient.id);

            toast.success("Успішна авторизація!", {
                position: "top-right",
                autoClose: 2000,
                hideProgressBar: false,
                closeOnClick: true,
                pauseOnHover: false,
                draggable: true,
                progress: undefined,
            });

            setTimeout(() => {
                window.location.href = "/dashboard";
            }, 2500);

        } catch (err) {
            setError(err.message || "Помилка при вході");
            toast.error("Помилка при вході", {
                position: "top-right",
                autoClose: 3000,
            });
        }
    };
```

Та друга частина представляє собою JSX розмітку, тобто то, що побачить користувач у браузері. Дана частина представляє собою код на мові HTML, в якому можна використовувати JavaScript змінні. Приклад такого коду з компонента «LoginPage.js» наведено далі.

```
return (
        <div className="login-container">
            <ToastContainer />
            <form className="login-form" onSubmit={handleLogin}>
                <h2>Авторизація</h2>
                {error && <p style={{ color: "red" }}>{error}</p>}
                <input
                    type="email"
                    placeholder="Електронна пошта"
                    value={email}
                    onChange={(e) => setEmail(e.target.value)}
                    required
                />
                <input
                    type="password"
                    placeholder="Пароль"
                    value={password}
                    onChange={(e) => setPassword(e.target.value)}
                    required
                />
```

```
                    <button type="submit">Увійти</button>
                    <p className="register-text">
                        Ще не маєте акаунту? <a
href="/patient/registration">Зареєструватися</a>
                    </p>
                </form>
            </div>
        );
};
```

В результаті виконання даного коду користувач побачить в браузері сторінку авторизації (див. рис. 4.7).

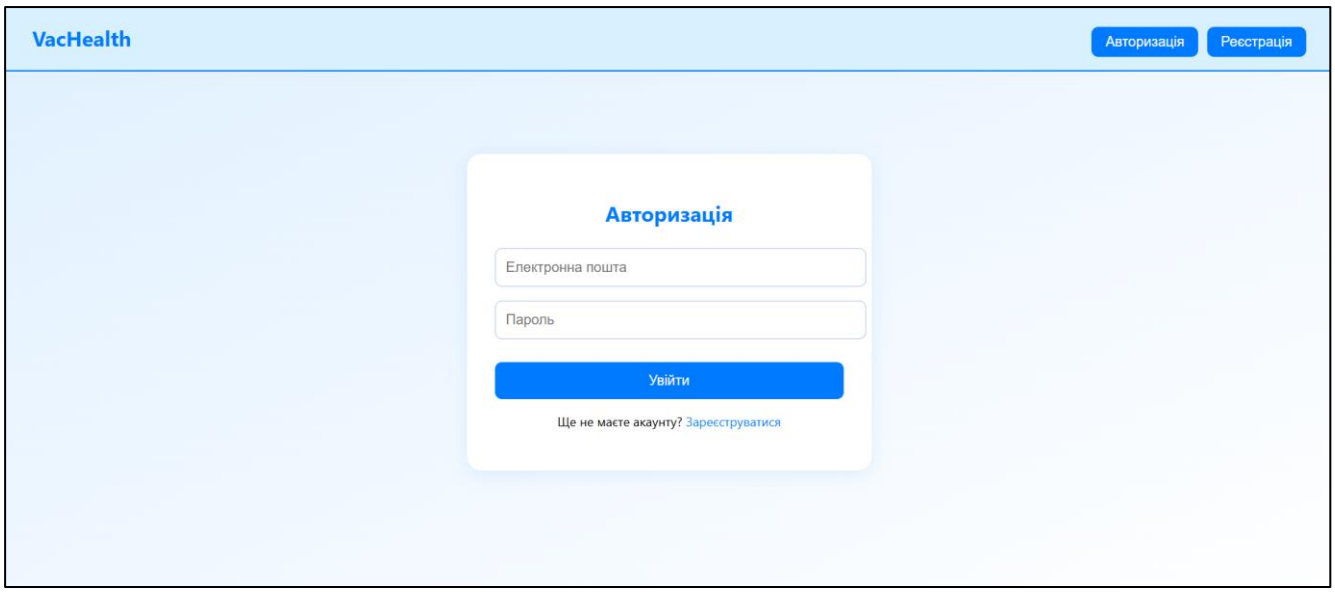

Рисунок 4.7 – Зовнішній вигляд сторінки авторизації (рисунок створений самостійно)

Саме таким чином було реалізовано всі інші веб-сторінки клієнтської частини.

4.5 Реалізація мобільного застосунку

Для реалізації мобільного Android застосунку обрано мову програмування Kotlin для написання програмного коду. Мову розмітки XML для створення елементів інтерфейсу та його зовнішнього вигляду. Та в якості середовища розробки та виконання обрано Android Studio.

В якості архітектури обрано MVC архітектуру. Вона полягає в розділені логіки роботи програми на окремі частини:

– «models» – моделі, сутності, дані, які відображаються користувачу та з якими працює програма;

– «view» – інтерфейс користувача, те, що він бачить та з чим може взаємодіяти;

– «controller» – обробка запитів від серверу.

Хоча в традиційній MVC архітектурі контролери відповідають за бізнес логіку та обробку даних, проте уся логіка вже реалізована на серверній частині, а в мобільному застосунку ми тільки звертаємось до серверу для відправки та отримання даних.

Дана архітектура проєкту наведена на рисунку 5.8.

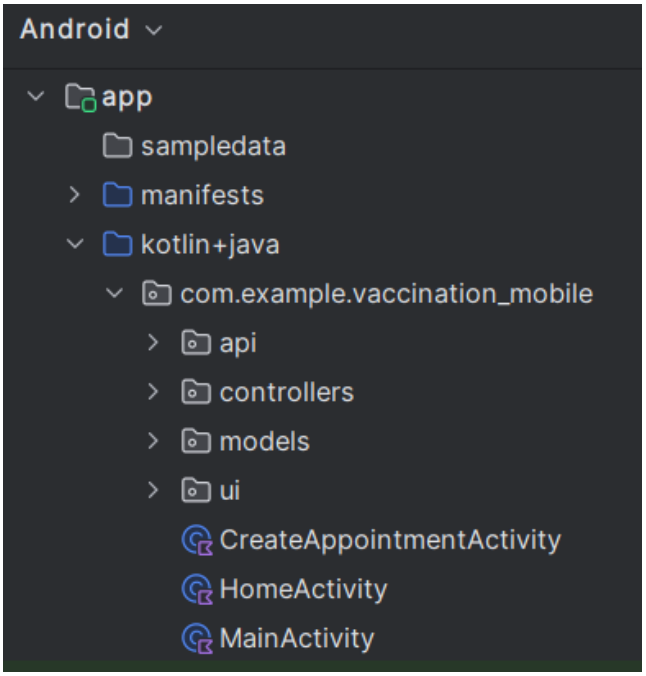

Рисунок 5.8 – Архітектура мобільної частини програмної системи (рисунок створений самостійно)

В даній архітектурі розділення обов’язків таким чином є головною перевагою, адже кожен компонент відповідає за свою частину логіки, що спрощує

розробку, тестування та подальшу підтримку програмного коду. Таким чином можна додавати нові функції в програму, ну чіпаючи інші компоненти.

Мобільна частина звертається до серверної частини для отримання або відправки даних. Для цього використовується бібліотека «retrofit.2». Налаштування звернення до серверу вказано у файлі «RetrofitInstance.kt», програмний код якого наведемо далі.

```
package com.example.vaccination_mobile.api

import com.example.vaccination_mobile.api.ApiService
import retrofit2.Retrofit
import retrofit2.converter.gson.GsonConverterFactory

object RetrofitInstance {
    private const val BASE_URL = "https://vacserver-
echbdrgpb4h3a0bm.francecentral-01.azurewebsites.net/"

    val api: ApiService by lazy {
        Retrofit.Builder()
            .baseUrl(BASE_URL)
            .addConverterFactory(GsonConverterFactory.create())
            .build()
            .create(ApiService::class.java)
    }
}
```

У файлі «ApiService.kt» вказуються кінцеві точки серверу, до яких звертається мобільна частина. Наведемо приклад кінцевої точки для отримання медичних закладів за назвою міста.

```
@GET("clinics/by-city")
    fun getClinicsByCity(@Query("city") city: String):
Call<List<Clinic>>
```

Далі необхідно визначити модель – сутність Clinic та її атрибути. Програмний код наведено нижче.

```
package com.example.vaccination_mobile.models

data class Clinic(
    val id: Int,
    val name: String,
```

```
    val address: String,
    val city: String,
    val description: String?
)
```

Й обробка запиту відбувається в контролері, наведемо як приклад «ClinicController.kt» з методом отримання даних.

```
package com.example.vaccination_mobile.controllers

import com.example.vaccination_mobile.api.RetrofitInstance
import com.example.vaccination_mobile.models.Clinic
import retrofit2.Call
import retrofit2.Callback
import retrofit2.Response

object ClinicController {
    fun getClinicsByCity(city: String, onResult: (List<Clinic>?) ->
Unit) {
        val call = RetrofitInstance.api.getClinicsByCity(city)

        call.enqueue(object : Callback<List<Clinic>> {
            override fun onResponse(call: Call<List<Clinic>>,
response: Response<List<Clinic>>) {
                if (response.isSuccessful) {
                    onResult(response.body())
                } else {
                    onResult(null)
                }
            }

            override fun onFailure(call: Call<List<Clinic>>, t:
Throwable) {
                onResult(null)
            }
        })
    }
}
```

Та результати запиту використовуються для відображення на елементах інтерфейсу користувачу. Результат пошуку медичних закладів за назвою міста відображено на рисунку 4.9.

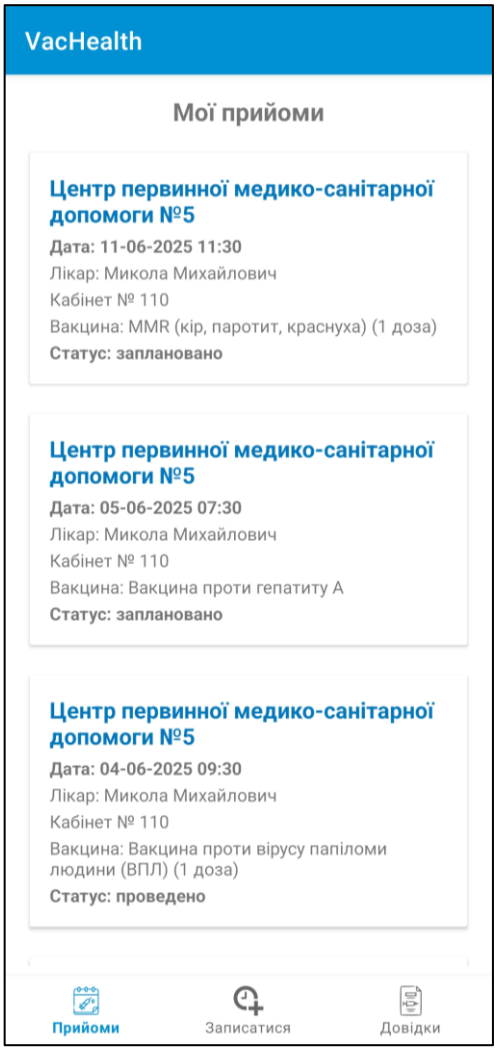

Рисунок 4.9 – Відображення результатів пошуку медичних закладів за назвою міста (рисунок створений самостійно)

Отже, саме таким чином реалізовано мобільну частину програмної системи для обліку проведення щеплень.

5 ТЕСТУВАННЯ РОЗРОБЛЕНОГО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Для перевірки працездатності, коректної обробки даних та відповідності функціоналу вимогам програмної системи для обліку проведення щеплень було проведено тестування.

Для тестування API серверної частини використано утиліту Postman. За допомогою неї перевірено:

– коректність обробки запитів GET, POST, PATCH, DELETE;

– валідацію вхідних даних;

– обробку помилок;

– повну роботи програмної системи з точки зору серверної частини.

Для кожного контролеру серверної частини створена колекція, в якій створено усі можливі запити до серверу (див. рис. 5.1).

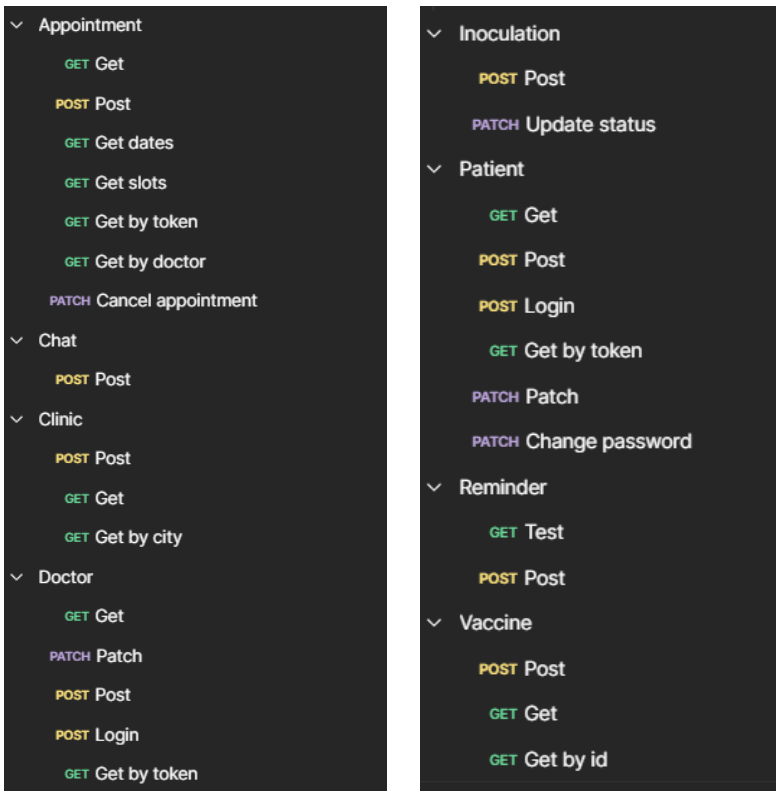

Рисунок 5.1 – Папки з усіма маршрутами до серверу в Postman (рисунок створений самостійно)

Таким чином, перевірена уся бізнес логіка та робота серверної частини. Тестування пройшло успішно.

Для тестування клієнтської частини використано мануальне тестування. Були створені тест-кейси. Кожен тест-кейс описував умови, очікувану поведінку системи та фактичний результат.

Результати тестування клієнтської частини наведено в таблиці 5.1.

Таблиця 5.1 – Тест-кейси клієнтської частини (таблиця виконана самостійно)

№	Тест-кейс	Вхідні дані	Очікуваний результат	Результат тесту
1	Авторизація пацієнта	Електронна пошта та пароль пацієнта.	Повідомлення про успішну авторизацію та перехід на робочу панель пацієнт	Пройдено
2	Спроба авторизації з помилковими даними	Невірна електронна пошта або пароль пацієнта.	Повідомлення про неправильну пошту або пароль.	Пройдено
3	Реєстрація пацієнта	Електронна пошта, пароль, ім’я, прізвище, номер телефону, день народження, стать.	Повідомлення про успішну реєстрація та перехід на сторінку авторизації.	Пройдено

Продовження таблиці 5.1

№	Тест-кейс	Вхідні дані	Очікуваний результат	Результат тесту
4	Редагування особистого профілю пацієнта.	Електронна пошта, пароль, ім’я, прізвище, номер телефону, день народження, стать.	Повідомлення про успішне збереження профілю пацієнта.	Пройдено
5	Зміна паролю пацієнта	Старий пароль, двічі новий пароль.	Повідомлення про успішну зміну паролю.	Пройдено
6	Запис на прийом на щеплення	Ідентифікатор лікаря, ідентифікатор вакцини, дата та час запису.	Повідомлення про успішне створення запису на прийом та перехід на сторінку перегляду прийомів.	Пройдено
7	Використання чату для питань	Питання	З’являється відповідь на питання від нейромережі	Пройдено
8	Скасування запису на прийом на щеплення	Ідентифікатор прийому.	Повідомлення про те, що прийом успішно скасовано.	Пройдено
9	Відмітка про виконаний прийом	Ідентифікатор прийому.	Повідомлення, що прийом завершено.	Пройдено

Кінець таблиці 5.1

№	Тест-кейс	Вхідні дані	Очікуваний результат	Результат тесту
10	Реєстрація нового доктора в системі	Ім’я, прізвище, номер телефону, електронна пошта, пароль, ідентифікатор клініки, номер кабінету.	Повідомлення про те, що доктора зареєстровано та листа з реєстраційними даними відправлено на пошту.	Пройдено
11	Додавання в систему нового медичного закладу	Назва, місто, адреса, опис.	Повідомлення про те, що медичний заклад додано успішно додано.	Пройдено
12	Формування медичної довідки	Ідентифікатор запису на прийом.	Повідомлення, що довідку завантажено.	Пройдено

Для тестування функціональності мобільного застосунку обрано також мануальне тестування. Тест-кейси та результати тестування наведено в таблиці 5.2.

Таблиця 5.2 – Тест-кейси мобільної частини (таблиця виконана самостійно)

№	Тест-кейс	Вхідні дані	Очікуваний результат	Результат тесту
1	Авторизація пацієнта	Електронна пошта та пароль пацієнта.	Повідомлення, що авторизація успішна, перехід на екран з прийомами.	Пройдено

Кінець таблиці 5.2

№	Тест-кейс	Вхідні дані	Очікуваний результат	Результат тесту
2	Спроба авторизації з помилковими даними	Невірна електронна пошта або пароль пацієнта.	Повідомлення про неправильну пошту або пароль.	Пройдено
3	Перегляд усіх записів на прийом пацієнта	Токен авторизації пацієнта.	Список записів на прийом, які пов’язані саме з даним пацієнтом.	Пройдено
4	Пошук медичних закладів за назвою міста	Назва міста.	Список медичних закладів, які знаходяться саме в даному місті.	Пройдено
5	Запис на прийом на щеплення	Ідентифікатор лікаря, ідентифікатор вакцини, дата та час запису.	Повідомлення про успішне створення запису на прийом та перехід на екран перегляду прийомів.	Пройдено
6	Формування довідки про вакцинацію	Ідентифікатор щеплення	Завантаження та відкриття довідки	Пройдено

Таким чином був протестований функціонал та інтерфейс користувача програмної системи для обробки проведення щеплень. Усі тести успішно виконано, що означає, що система відповідає вимогам та очікуванням користувачів та готова до впровадження.

6 ВПРОВАДЖЕННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Після етапів проєктування, розробки та тестування програмне забезпечення готове до впровадження. Для впровадження програмної системи необхідно надати доступ до нього для користувачів на будь яких пристроях. Для цього необхідно обрати метод розгортання. Мною обрано розгортання програмної системи в хмарному сервісі Microsoft Azure.

Загалом, хмарні сервіси мають низку переваг порівняно з традиційними методами розгортання. По-перше, вони забезпечують високу масштабованість, що дозволяє автоматично або вручну збільшувати ресурси при зростанні навантаження. По-друге, доступність з будь-якого місця та з різних пристроїв забезпечує зручність для кінцевих користувачів. Крім того, хмарні платформи, зокрема Microsoft Azure, надають інтегровані засоби безпеки, резервного копіювання та моніторингу, що підвищує надійність і стабільність роботи програмної системи. Також хмарне розгортання зменшує витрати на підтримку фізичної інфраструктури.

Microsoft Azure надає дуже багато можливостей та ресурсів для розгортання застосунків. По перше, вже використовується хмарну базу даних Azure SQL. Таким чином можна отримати доступ до даних з будь якого пристрою в будь який час, ніж зберігати дані локально на комп’ютері.

Для розгортання серверної та клієнтської частини обрано метод контейнеризації [11]. Контейнеризація дозволяє упаковувати програму разом з усіма її залежностями в ізольоване середовище, що називається контейнером. Це забезпечує відтворюваність, портативність та незалежність від середовища, у якому буде запускатись система. Найпоширенішим інструментом для створення контейнерів є Docker, який використано для контейнеризації як серверної частини, так і клієнтської частини.

Azure надає хмарні контейнери, які мають назву Container registry. Обрано саме цей тип хмарного контейнеру, адже в нього легко завантажити образи прямо з Docker.

Для запуску даних контейнерів обрано Azure Web App. Даний ресурс надає домен для веб-застосунку. Таким чином користувачі з будь який пристроїв зможуть отримати доступ до програмної системи через браузер.

Першим етапом перед розгортанням у хмарі, треба зібрати образи клієнтської та серверної частин в Docker.

Для цього треба створити окремо на в кожному проєкті Dockerfile. Він описує правила збору та запуску проєкту. Й далі використана команда «docker build» для збору образу. Дані команди для серверної та клієнтської частин відповідно наведені нижче.

```
docker build -t vaccinationcontainerregistry1.azurecr.io/vacserver .
docker build -t vaccinationcontainerregistry1.azurecr.io/vacclient .
```

Далі необхідно створити Container registry в Microsoft Azure (див. рис. 6.1).

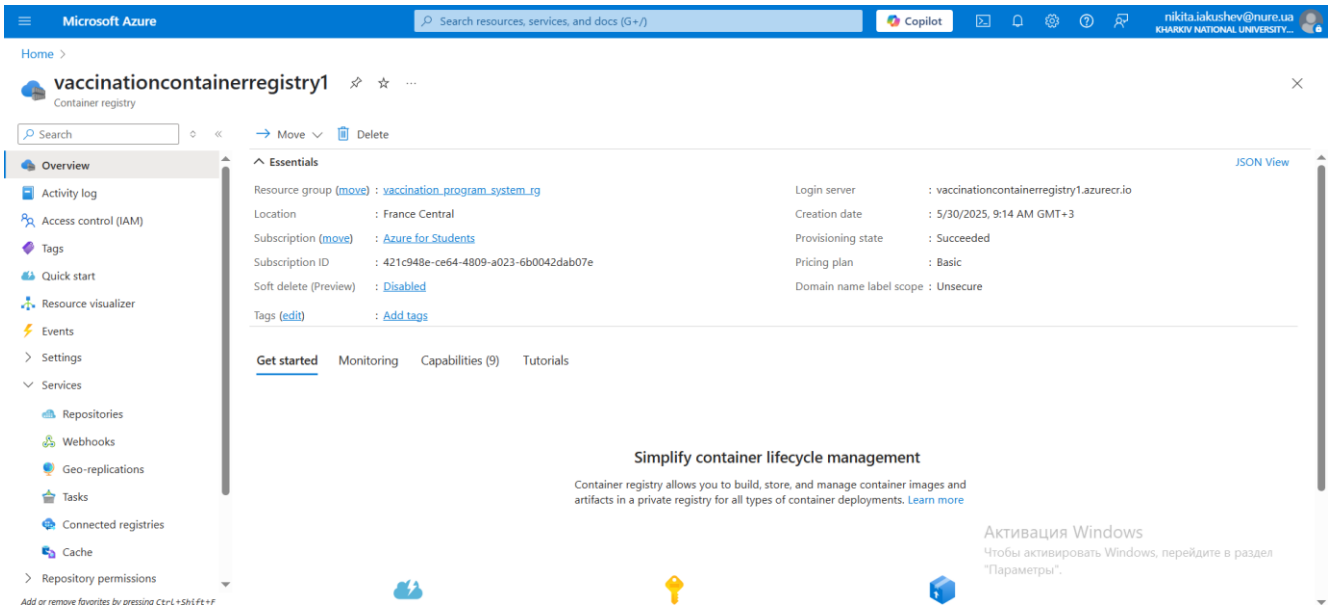

Рисунок 6.1 – Container registry для контейнеризації застосунків (рисунок створений самостійно)

Після створення хмарного контейнеру необхідно відправити наші образи застосунків у хмару за допомогою команди «docker push».

Далі наведено дані команди для серверної та клієнтської частин відповідно.

```
docker push vaccinationcontainerregistry1.azurecr.io/vacserver
docker push vaccinationcontainerregistry1.azurecr.io/vacclient
```

Останнім кроком необхідно створити Azure Web App окремо для запуску серверної та клієнтської частин. Створені Azure Web App для серверної та клієнтської частин наведені на рисунках 6.2 та 6.3 відповідно.

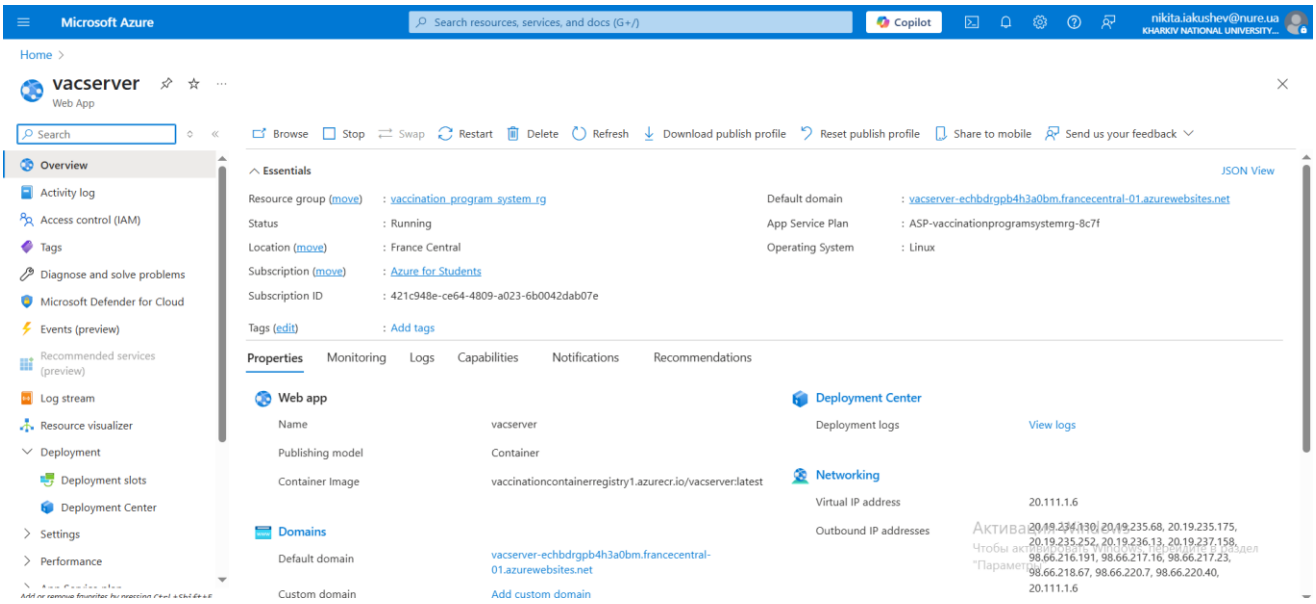

Рисунок 6.2 – Azure Web App для розгортання серверної частини (рисунок створений самостійно)

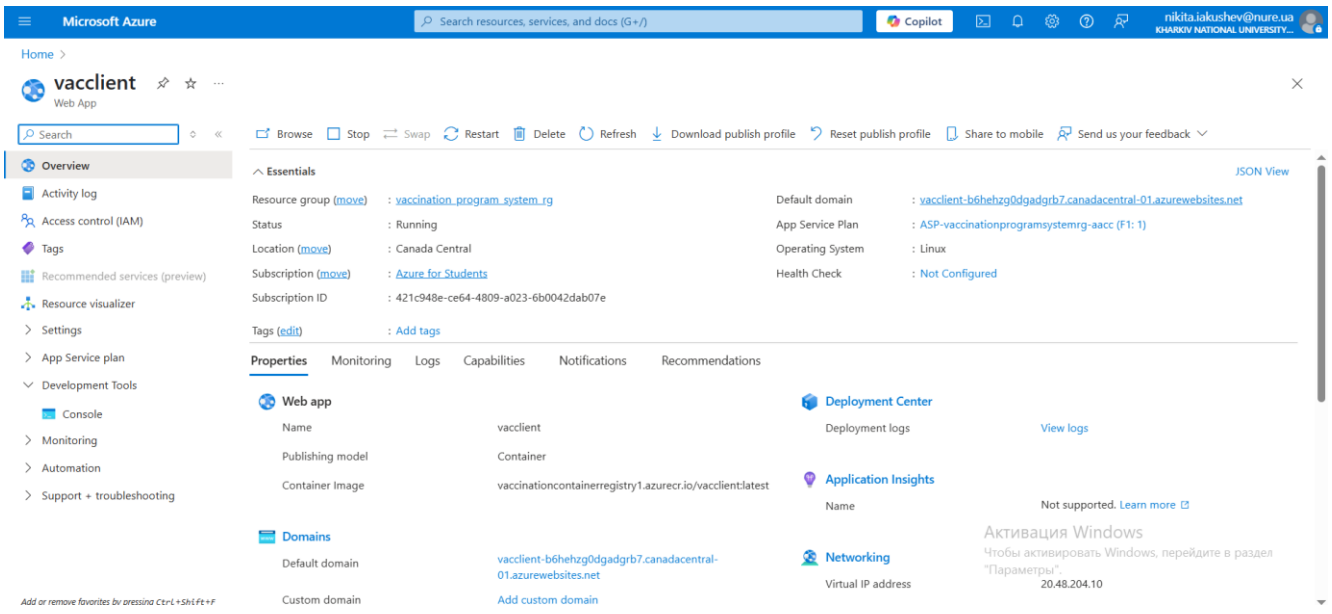

Рисунок 6.3 – Azure Web App для розгортання клієнтської частини (рисунок створений самостійно)

Таким чином, ми можемо перейти за посиланням https://vacclient-b6hehzg0dgadgrb7.canadacentral-01.azurewebsites.net/ та почати користуватися програмною системою для обліку проведення щеплень.

ВИСНОВКИ

У процесі виконання кваліфікаційної роботи бакалавра було проведено повноцінний цикл розробки програмної системи для обліку проведення щеплень – від аналізу предметної області до реалізації, тестування та підготовки до впровадження. Розроблена система дозволяє автоматизувати процеси запису пацієнтів на щеплення, ведення історії вакцинацій, обліку роботи лікарів, управління медичними закладами.

На етапі аналізу предметної області були визначені основні проблеми існуючих рішень, зокрема відсутність централізованої бази даних, складнощі з отриманням довідок, висока ймовірність помилок під час ручного обліку та обмежений доступ пацієнтів до історії щеплень. На основі цього було сформульовано завдання та визначено цілі проєкту.

Сформовані функціональні та нефункціональні вимоги стали основою для подальшого проєктування. У роботі було побудовано Use-case діаграми для трьох основних категорій користувачів – пацієнтів, лікарів та адміністраторів.

Було спроєктовано архітектуру програмного забезпечення, структуру бази даних, ключові алгоритми та логіку взаємодії між компонентами.

Були описані прийняті програмні рішення, обґрунтовано вибір технологій, описано реалізацію серверної, клієнтської та мобільної частин, забезпечено взаємодію з базою даних та реалізовано функціональність, необхідну для основного сценарію використання.

Також було проведено тестування програмної системи, що дозволило виявити й усунути помилки, перевірити відповідність реалізації заданим вимогам та забезпечити надійність і стабільність роботи.

Розглянуто питання впровадження системи, включаючи сценарії розгортання та подальшої підтримки.

Таким чином, у межах виконання кваліфікаційної роботи бакалавра розроблено повноцінну програмну систему для обліку проведення щеплень.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Медичні інформаційні системи [Електронний ресурс] – URL: https://ehealth.moz.gov.ua/electronic-healthcare/mis/ (дата звернення: 15.04.2025).

2. Node.js v23.11.0 documentation [Електронний ресурс] – URL: https://nodejs.org/docs/latest/api/ (дата звернення: 20.04.2025).

3. Develop Android apps with Kotlin [Електронний ресурс] – URL: https://developer.android.com/kotlin (дата звернення: 21.04.2025).

4. Загальна інформація | Центр громадського здоров’я. Центр громадського здоров’я України | МОЗ [Електронний ресурс] – URL: https://phc.org.ua/kontrol-zakhvoryuvan/imunizaciya/zagalna-informaciya (дата звернення: 16.04.2025).

5. Helsi.me [Електронний ресурс] – URL: https://helsi.me/ (дата звернення: 17.04.2025).

6. Електронна система охорони здоров'я в Україні [Електронний ресурс] – URL: https://ehealth.gov.ua/ (дата звернення: 17.04.2025).

7. Deployment diagram [Електронний ресурс] – URL: https://c4model.com/diagrams/deployment (дата звернення: 21.04.2025).

8. Bock D. B., Bordoloi B. SQL for SQL Server. Prentice Hall, 2003. – 416 с.

9. Klein S., Roggero H. Pro SQL Azure. Apress, 2011. – 312 с.

10. Xia J. UI/UX Design. Artpower International Publish Company, Limited, 2016. 256 с.

11. Kravets N., Tseshkovskyi N., Liutova K. Containers and virtual machines in Microsoft Azure //Science, society, education: topical issues and development prospects. Abstracts of the 7th International scientific and practical conference. SPC “Sciconf. com. ua”. Kharkiv, Ukraine. – 2020. – С. 278-281.

12. Посилання на GitHub: https://github.com/NureIakushevNikita/2025_B_PI_PZPI-21-10_Iakushev_N_O