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 медичних закладів «LifeLine». Front-end
(тема)

Виконав:
здобувач четвертого року навчання
групи ПЗПІ-21-6

Дарина БУРАВКОВА
(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. кафедри ПІ Віктор КАУК
(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
(код і повна назва)
Тип програми Освітньо-професійна
Освітня програма Програмна Інженерія
(шифр і назва)

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

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

здобувачеві Буравковій Дарині Андріївні
(прізвище, ім'я, по батькові)

1. Тема роботи Програмна система для медичних закладів «LifeLine». Front-end
затверджена наказом по університету від 19.05.2025р. № 397 Ст
2. Термін подання студентом роботи до екзаменаційної комісії 16.06.2025
3. Вихідні дані до роботи Розробити клієнтську частину веб-застосунку медичної інформаційної системи «LifeLine», що забезпечує інтерфейс для пацієнтів, лікарів та адміністративного персоналу, з використанням мови програмування JavaScript, бібліотеки React та з інтеграцією REST API.
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, висновки, додатки.

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів роботи	Термін виконання етапів роботи	Примітка
1	Аналіз предметної галузі	08.04.2025	*виконано*
2	Створення специфікації ПЗ	11.04.2025	*виконано*
3	Проектування ПЗ	16.04.2025	*виконано*
4	Розробка ПЗ	21.04.2025	*виконано*
5	Тестування ПЗ	24.05.2025	*виконано*
6	Оформлення пояснювальної записки	26.05.2025	*виконано*
7	Підготовка презентації та доповіді	06.06.2025	*виконано*
8	Попередній захист	13.06.2025	*виконано*
9	Нормоконтроль, рецензування	13.06.2025	*виконано*
10	Здача роботи у електронний архів	13.06.2025	*виконано*
11	Допуск до захисту у зав. кафедри	15.06.2025	*виконано*

Дата видачі завдання « 8 » « квітня » 2025р.

Здобувач _____
(підпис)

Керівник роботи _____ доц. кафедри ПІ Віктор КАУК
(підпис) (посада, Власне ім’я, ПРІЗВИЩЕ)

РЕФЕРАТ

Пояснювальна записка до кваліфікаційної роботи бакалавра, 100 стор., 12 рис., 12 джерел, 5 додатків.

АВТОМАТИЗАЦІЯ, АДМІНІСТРАТОР, ВЕБ-РОЗРОБКА, ЕЛЕКТРОННА МЕДИЧНА КАРТКА, ЛІКАРНЯ, МЕДИЧНА СИСТЕМА, ПАЦІЄНТ, ПРОТОТИП, FRONT-END, JAVASCRIPT, REACT, REST API, VISUAL STUDIO CODE.

Об’єкт розробки – клієнтська частина програмної системи для медичних закладів «LifeLine», що забезпечує взаємодію користувачів із системою через зручний веб-інтерфейс.

Мета розробки – створення функціонального, інтуїтивно зрозумілого та безпечного інтерфейсу для ефективної роботи лікарів, пацієнтів та адміністраторів.

Метод рішення – середовище розробки Visual Studio Code з використанням мови JavaScript і бібліотеки React для реалізації модульної архітектури. Для верстки застосовувалися HTML5 і CSS3, а UI/UX дизайн створено згідно з сучасними принципами доступності та адаптивності.

У результаті розробки створено повнофункціональну клієнтську частину застосунку, що включає модулі створення акаунтів та авторизації, запису на прийом, перегляду електронної медичної картки, замовлення аналізів, генерації звітів, а також забезпечує адаптивність до різних пристроїв, високу швидкодію та базові механізми захисту даних.

ABSTRACT

AUTOMATION, ADMINISTRATOR, WEB DEVELOPMENT, ELECTRONIC MEDICAL RECORD, HOSPITAL, MEDICAL SYSTEM, PATIENT, PROTOTYPE, FRONT-END, JAVASCRIPT, REACT, REST API, VISUAL STUDIO CODE.

The object of development is the client part of the LifeLine software system for medical institutions, which provides user interaction with the system through a convenient web interface.

The purpose of the work is to create a functional, intuitive and secure interface for the efficient work of doctors, patients and administrators.

Solution method – Visual Studio Code development environment using the JavaScript language and the React library to implement a modular architecture. HTML5 and CSS3 were used for the layout, and the UI/UX design was created in accordance with modern principles of accessibility and adaptability.

As a result of the development, a fully functional client part of the application was created, including modules for creating accounts and authorisation, making an appointment, viewing an electronic medical record, ordering tests, generating reports, and ensuring adaptability to different devices, high performance and basic data protection mechanisms.

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ

REST – Representational State Transfer

API – Application Programming Interface

HTTP – HyperText Transfer Protocol

HTTPS – HyperText Transfer Protocol Secure

UI – User Interface

UX – User Experience

SPA – Single Page Application

JWT – JSON Web Token

HTML – HyperText Markup Language

CSS – Cascading Style Sheets

JSON – JavaScript Object Notation

DOM – Document Object Model

URL – Uniform Resource Locator

ВСТУП

Сучасні інформаційні технології значною мірою змінюють підхід до організації роботи у різних галузях діяльності, зокрема у сфері охорони здоров’я. Застосування цифрових рішень у медицині дозволяє підвищити якість обслуговування пацієнтів, оптимізувати роботу персоналу та забезпечити швидкий доступ до актуальної інформації. В умовах високих навантажень на медичні заклади, особливо у періоди епідемій чи кризових ситуацій, ефективність управління ресурсами та процесами стає критично важливою. Саме тому розробка сучасних програмних систем для медичних закладів є надзвичайно актуальною.

Однією з ключових проблем медичних установ є застаріле або недостатньо ефективне програмне забезпечення, яке не враховує потреби користувачів у зручності, швидкодії, безпеці даних та мобільності доступу. Пацієнти прагнуть мати можливість швидко записатися до лікаря, переглянути результати аналізів, отримати нагадування про візити, а медичний персонал потребує інструментів для зручного керування прийомами, ведення електронних медичних карток, аналітики та фінансової звітності. Тому створення функціональної, безпечної та ефективної клієнтської частини системи «LifeLine» є актуальним та своєчасним кроком у напрямку цифровізації медицини.

Розробка клієнтської частини цієї системи є критично важливою складовою, адже саме через інтерфейс користувачі взаємодіють із сервісом. Зручність, швидкодія, безпека, адаптивність та інтуїтивна зрозумілість інтерфейсу безпосередньо впливають на успішність впровадження системи у реальних умовах.

Мета роботи полягає у розробці фронтенд-частини програмної системи для медичних закладів «LifeLine», яка забезпечить інтерактивний інтерфейс для взаємодії пацієнтів, лікарів та адміністративного персоналу. Основна увага

приділяється створенню інтуїтивно зрозумілого, адаптивного та безпечного веб-інтерфейсу з використанням сучасних технологій.

Розроблена клієнтська частина може бути використана у приватних та державних медичних установах різного рівня для автоматизації процесів обслуговування пацієнтів, обліку медичної діяльності та оптимізації внутрішніх операційних процесів. Система «LifeLine» сприяє зменшенню навантаження на адміністративний персонал, підвищенню прозорості роботи медичного закладу та покращенню рівня обслуговування пацієнтів за рахунок швидкого доступу до актуальної інформації.

1 АНАЛІЗ ПРЕДМЕТНОЇ ГАЛУЗІ

1.1 Аналіз предметної галузі

Сфера охорони здоров’я є однією з найважливіших галузей для суспільства, оскільки безпосередньо пов'язана із забезпеченням здоров’я та добробуту людей. З розвитком інформаційних технологій медичні установи все активніше інтегрують цифрові рішення для автоматизації бізнес-процесів, оптимізації роботи персоналу, підвищення якості надання медичних послуг та покращення взаємодії з пацієнтами.

У сучасних реаліях потреба у впровадженні електронних систем управління медичними послугами значно зросла. Традиційні паперові облікові системи застаріли і не відповідають вимогам ефективності, швидкості та безпеки даних. Впровадження інформаційних систем дозволяє не тільки мінімізувати ці недоліки, але й забезпечити прозорість усіх процесів, підвищити якість медичних послуг та зменшити навантаження на адміністраторів медичних закладів.

Серед головних функцій таких систем – централізоване зберігання інформації про пацієнтів, інтеграція з лабораторіями та аптеками, моніторинг ефективності роботи лікарів, можливість дистанційного запису на прийом, автоматичне нагадування пацієнтам про заплановані візити та обробка статистичних даних для аналітичних звітів.

При аналізі предметної області необхідно враховувати особливі вимоги до захисту персональних даних відповідно до міжнародних та національних стандартів безпеки інформації, регламентування медичної документації, специфіку обробки чутливої інформації медичних даних пацієнтів, а також вимоги до доступності та безперервності роботи системи.

Така система може бути актуальною для різних типів медичних установ, включаючи державні та приватні клініки, діагностичні центри, стоматологічні кабінети, лабораторії, а також для систем телемедицини. Використання такого

інструменту сприятиме підвищенню якості медичних послуг, зниженню витрат часу на адміністративні процеси, покращенню досвіду пацієнтів та забезпеченню прозорості у взаємодії з медичним персоналом.

Розуміння потреб кінцевих користувачів, як медичного персоналу, так і пацієнтів – є ключовим для розробки зручного та високоефективного програмного забезпечення. Інтерфейс системи має бути інтуїтивно зрозумілим, швидким у роботі та адаптивним до різних пристроїв, враховуючи високе навантаження, різні сценарії використання та можливу обмежену цифрову грамотність частини користувачів.

1.2 Виявлення та вирішення проблем

Провевши аналіз предметної галузі, було виявлено низку ключових проблем, характерних для сучасних медичних закладів, які суттєво впливають на якість обслуговування пацієнтів, ефективність роботи лікарів та адміністративний менеджмент.

Однією з найбільш поширених проблем є неефективна організація процесу запису на прийом. Багато закладів досі використовують телефонні дзвінки або живі черги для реєстрації пацієнтів, що призводить до черг, плутанини у розкладі лікарів, незадоволення клієнтів та втрати часу як пацієнтів, так і медичного персоналу. Програмна система вирішує цю проблему шляхом впровадження онлайн-запису на прийом із можливістю вибору лікаря, дати та часу візиту.

Ще одна важлива проблема – відсутність централізованого обліку медичних даних пацієнтів. Традиційні паперові системи обліку є малоефективними: вони потребують значних затрат часу, ресурсів та є вразливими до людських помилок. Пошук необхідної інформації в паперових архівах займає багато часу, що негативно впливає на оперативність прийняття рішень лікарями. Програмна система вирішує це завдяки впровадженню

електронних медичних карток, які забезпечують зручне зберігання, оновлення та доступ до історії хвороб кожного пацієнта.

Низька якість комунікації між лікарями та пацієнтами також є суттєвою проблемою. Часто пацієнти забувають про заплановані прийоми або не мають актуальної інформації щодо результатів аналізів. Для вирішення цієї проблеми програмна система реалізує зручний функціонал, який дозволяє пацієнтам у власному кабінеті переглядати інформацію про візити, оновлення статусу замовлених послуг та результати аналізів безпосередньо через особистий кабінет на веб-сайті. Таким чином, у будь-який момент користувачі мають доступ до актуальної інформації онлайн у зручному та зрозумілому інтерфейсі.

Адміністративне навантаження на персонал медичних закладів є ще однією суттєвою проблемою. Велика кількість рутинних операцій, таких як складання графіків роботи лікарів, облік відвідувань та підготовка фінансової звітності, суттєво зменшує час, який можна витратити на безпосереднє лікування пацієнтів. Для вирішення цієї проблеми система пропонує повну автоматизацію відповідних процесів. Управління графіками, облік фінансових надходжень та витрат, а також генерація аналітичних і фінансових звітів здійснюються через інтуїтивно зрозумілий веб-інтерфейс. Адміністратори можуть переглядати результати у вигляді графіків, діаграм і таблиць, що забезпечує швидке сприйняття інформації та полегшує ухвалення управлінських рішень.

Розробка клієнтського застосунку безпосередньо спрямована на вирішення найважливіших проблем сучасних медичних закладів: оптимізацію процесу обліку та обслуговування пацієнтів, централізацію інформації, автоматизацію адміністративних задач, підвищення якості комунікації та забезпечення ефективного управління установою на основі аналітичних даних.

1.3 Аналіз існуючих аналогів

На сьогодні ринок програмного забезпечення для медичних закладів активно розвивається, зокрема і в Україні. Існує ряд рішень, які частково або повністю автоматизують процеси медичних установ. Проте більшість існуючих систем мають обмеження в частині користувацького інтерфейсу та зручності роботи саме через веб-платформу.

Одним із найпоширеніших українських продуктів є система Helsi.me (див. рис. 1.1). Вона забезпечує електронний запис до лікаря, доступ до електронних медичних карток пацієнтів, електронні рецепти та повідомлення про результати аналізів. Основними перевагами Helsi.me є широке охоплення державних закладів охорони здоров'я та наявність базових електронних сервісів для пацієнтів. Проте система має обмежені можливості для приватних клінік, недостатньо гнучкий інтерфейс та мінімальні можливості візуалізації аналітичних даних.

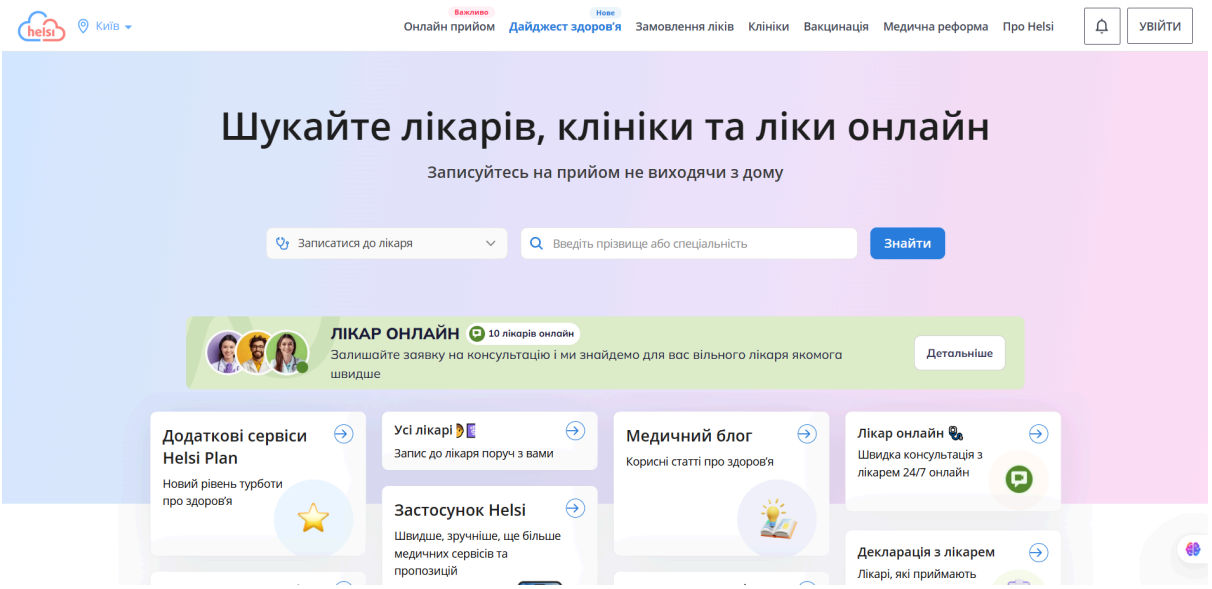

Рисунок 1.1 – Веб-сервіс Helsi.me (за даними [1])

Іншим прикладом є система Medics (див. рис. 1.2), яка орієнтована переважно на лікарів і адміністративний персонал. Вона дозволяє вести

електронні медичні записи, формувати електронні рецепти, автоматизувати подачу звітності до Міністерства охорони здоров'я. Важливою перевагою Medics є інтеграція з державними стандартами та спрощення документообігу для закладів, що працюють із eHealth. Проте Medics має досить складний процес реєстрації та авторизації, що ускладнює його швидке впровадження в медичну практику. Крім того, для пацієнтів функціональність залишається обмеженою: відсутня зручна аналітика, обмежені можливості управління записами, недостатня адаптивність для мобільних та веб-платформ.

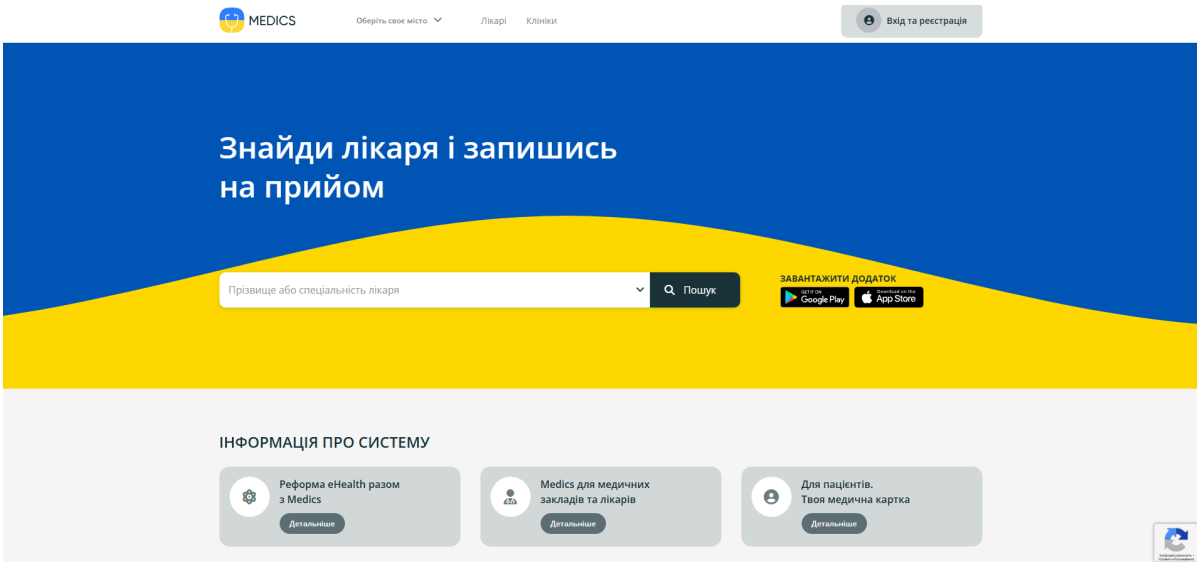

Рисунок 1.2 – Веб-сервіс Medics (за даними [2])

Попри наявність таких рішень, існують суттєві обмеження щодо сучасності інтерфейсу, інтерактивності взаємодії та розширених можливостей для аналізу даних як для медичних працівників, так і для керівництва клінік.

Основними перевагами розроблюваного програмного продукту над існуючими аналогами є сучасний адаптивний веб-інтерфейс, доступ до всіх функцій через зручний сайт без необхідності встановлення мобільного додатку, розширені можливості аналітики із візуалізацією даних у вигляді графіків і діаграм, зручний електронний запис до лікаря, автоматичні нагадування та

повідомлення, а також гнучке налаштування системи під потреби як державних, так і приватних медичних установ.

1.4 Монетизація

Для забезпечення стійкого розвитку програмної системи та її подальшої підтримки передбачено кілька моделей монетизації, які враховують специфіку використання в різних типах медичних закладів та різні рівні доступу користувачів до функціональності системи.

Основним джерелом доходу буде щомісячна або щорічна підписка для медичних закладів. Залежно від обраного пакету, клієнти отримуватимуть доступ до базових або розширених функцій. Базовий пакет включатиме основні модулі: електронний запис до лікаря, автоматичні нагадування, управління графіками, базову аналітику. Розширені пакети надаватимуть доступ до додаткових функцій: розгорнута аналітика, фінансові звіти, модулі замовлення лабораторних досліджень, персоналізовані повідомлення тощо.

Для деяких функцій системи може бути застосована модель оплати за фактичне використання. Кожен медичний заклад має свої особливості організації роботи, тому в подальшому передбачається можливість гнучкого налаштування інтерфейсу, додавання специфічних модулів або інтеграцій із внутрішніми системами обліку та державними реєстрами. Ці послуги будуть оплачуватися окремо на основі погоджених технічних завдань.

Ще одним можливим елементом монетизації є платні додаткові функції для пацієнтів, такі як доступ до особистої медичної аналітики, персоналізовані рекомендації щодо здоров’я, можливість запису до найкращих лікарів без черги, розширені профілі обліку стану здоров’я. Базовий функціонал залишатиметься безкоштовним, що забезпечить залучення широкої аудиторії користувачів.

Окремим напрямом може стати партнерство з лабораторіями та аптечними мережами, які за додаткову плату отримуватимуть можливість інтегрувати свої послуги безпосередньо у веб-платформу. Це дасть змогу

пацієнтам швидше замовляти аналізи чи отримувати рецептурні препарати безпосередньо через систему.

1.5 Постановка задачі

Виходячи з аналізу предметної галузі, виявлених проблем і оцінки існуючих аналогів, формулюється задача розробки програмної системи для медичних закладів. Головною метою є створення сучасного веб-застосунку, що забезпечує зручну, швидку та безпечну взаємодію пацієнтів, лікарів і адміністративного персоналу через веб-інтерфейс.

Проект має комплексний характер і виконується у складі команди з двох розробників. Перший розробник відповідає за реалізацію серверної частини (backend), розробленою за допомогою Node.js з використанням Express та бази даних PostgreSQL. Серверна частина відповідає за зберігання, обробку та валідацію всіх медичних даних, забезпечуючи створення механізмів авторизації, контролю доступу на основі існуючих ролей в системі, розробку та налаштування API, а також впровадження логіки бізнес-процесів для медичних установ.

Другий розробник, в межах поданої частини кваліфікаційної роботи, відповідає за розробку клієнтської частини (frontend), що охоплює створення візуального інтерфейсу та логіки взаємодії користувачів із системою. Головним завданням клієнтської частини є реалізація інтерактивного веб-застосунку, який отримує, відображає та надсилає дані до серверної частини через API-запити відповідно до дій користувачів у системі. Таким чином, обидві частини тісно взаємодіють між собою.

Початковим етапом розробки клієнтської частини є формування правильної архітектури застосунку. Необхідно визначити основні модулі системи, спроектувати їхню взаємодію, обрати технології для реалізації. Вибір технологічного стеку є критично важливим: було обрано використання React як

основного інструменту для розробки клієнтської частини завдяки його гнучкості, модульності та високій продуктивності.

Другою ключовою задачею є створення користувацького інтерфейсу (UI/UX), який забезпечить простоту навігації, адаптивність для різних пристроїв, таких як комп’ютери, планшети, смартфони, та швидкий доступ до основних функцій системи. Для цього буде розроблено набір макетів і прототипів, що відображають головні сценарії використання: авторизацію користувачів, запис на прийом чи обстеження, перегляд медичних записів, взаємодію з фінансовими та аналітичними модулями.

Особливу увагу необхідно приділити інтеграції фронтенду із зовнішніми сервісами та API бекенд-системи. Для цього необхідно розробити механізми безпечного обміну даними, автентифікації користувачів, обробки помилок і захисту переданої інформації. Усі запити до сервера повинні бути захищені за допомогою HTTPS, а доступ до даних – обмежений відповідно до ролі користувача в системі.

Крім основної функціональності, важливим аспектом є забезпечення продуктивності застосунку. Оптимізація швидкості завантаження сторінок, мінімізація обсягу передаваних даних, використання асинхронних запитів та оптимальне кешування інформації мають бути враховані вже на етапі проєктування архітектури.

Постановка зазначених задач дозволить створити повноцінний інструмент для автоматизації основних процесів у медичних закладах, забезпечить комфорт пацієнтам та лікарям, оптимізує адміністративну діяльність і підвищить загальну ефективність функціонування медичних установ.

2 ФОРМУВАННЯ ВИМОГ ДО ПРОГРАМНОЇ СИСТЕМИ

2.1 Окреслення концепції

Програмна система має на меті створити комплексне рішення для автоматизації основних процесів у медичних закладах різного рівня – від приватних клінік до державних лікарень. Система повинна забезпечити зручну та безпечну взаємодію пацієнтів, лікарів та адміністративного персоналу через сучасний веб-інтерфейс.

В основі концепції цієї системи лежить прагнення створити універсальне, інтуїтивно зрозуміле та доступне середовище для взаємодії всіх учасників медичного процесу. Ключовою ідеєю є максимальна простота користування незалежно від рівня технічної підготовки користувача. Для пацієнтів система забезпечує можливість швидкого і зручного доступу до медичних послуг. Для лікарів створено функціональні інструменти управління власним графіком роботи, ведення електронних медичних карток пацієнтів, призначення ліків та формування медичних звітів. Адміністративний персонал отримує засоби для ефективного контролю над фінансовими потоками закладу, автоматизованого складання графіків лікарів та генерації аналітичних звітів для оцінки ефективності роботи медичного закладу.

Клієнтська частина системи розробляється з урахуванням найкращих практик веб-розробки, використовуючи сучасні технології та орієнтуючись на створення адаптивного і динамічного інтерфейсу. Вся логіка взаємодії користувача з системою буде побудована за принципами простоти, наочності та мінімізації кількості дій, необхідних для досягнення цільового результату.

Важливою складовою концепції є питання безпеки персональних даних користувачів. Передбачено обов’язкове шифрування переданої та збереженої інформації, автентифікацію користувачів та контроль доступу до даних відповідно до призначених ролей.

Реалізація цієї системи не тільки змінить підхід до управління медичними закладами, а й дозволить значно підвищити ефективність їхньої роботи. Завдяки об’єднанню адміністративних, медичних та комунікаційних процесів в єдиному веб-застосунку, клініки зможуть забезпечити пацієнтам швидкий і зручний доступ до медичних послуг, покращити якість обслуговування, а також оптимізувати внутрішню організацію своєї діяльності. Створення такої цифрової платформи відкриває можливості для оперативного прийняття рішень, підвищення прозорості процесів та кращого контролю за роботою персоналу, що є важливими перевагами в умовах сучасних вимог до медичної сфери.

2.2 Концептуальне моделювання

Розглянемо ключові моменти щодо функціонування та структури системи з точки зору взаємодії з користувачами, потоків даних і базових сценаріїв використання.

Розроблена програмна система передбачає поділ користувачів за ролями відповідно до їхніх функціональних обов’язків та рівня доступу до даних. Це дозволяє організувати ефективну і безпечну роботу з системою, розмежувати доступ до інформації та оптимізувати робочі процеси. Загалом у системі визначено такі основні типи користувачів: пацієнт, лікар-медичний персонал та адміністратор.

Кожна з ролей має свої специфічні права доступу та сценарії взаємодії із системою. Для кожної ролі користувача створено відповідні Use-Case діаграми, які наочно демонструють основні взаємодії користувача із системою.

Роль пацієнта у системі полягає у взаємодії з медичними послугами через особистий кабінет. Для пацієнта (див. рис. 2.1) діаграма включає сценарії запису на прийом, перегляд особистої медичної картки, результатів аналізів, призначень лікарів, перегляд історії консультацій та лікувань, отримання електронних рецептів і направлень та редагування особистої інформації.

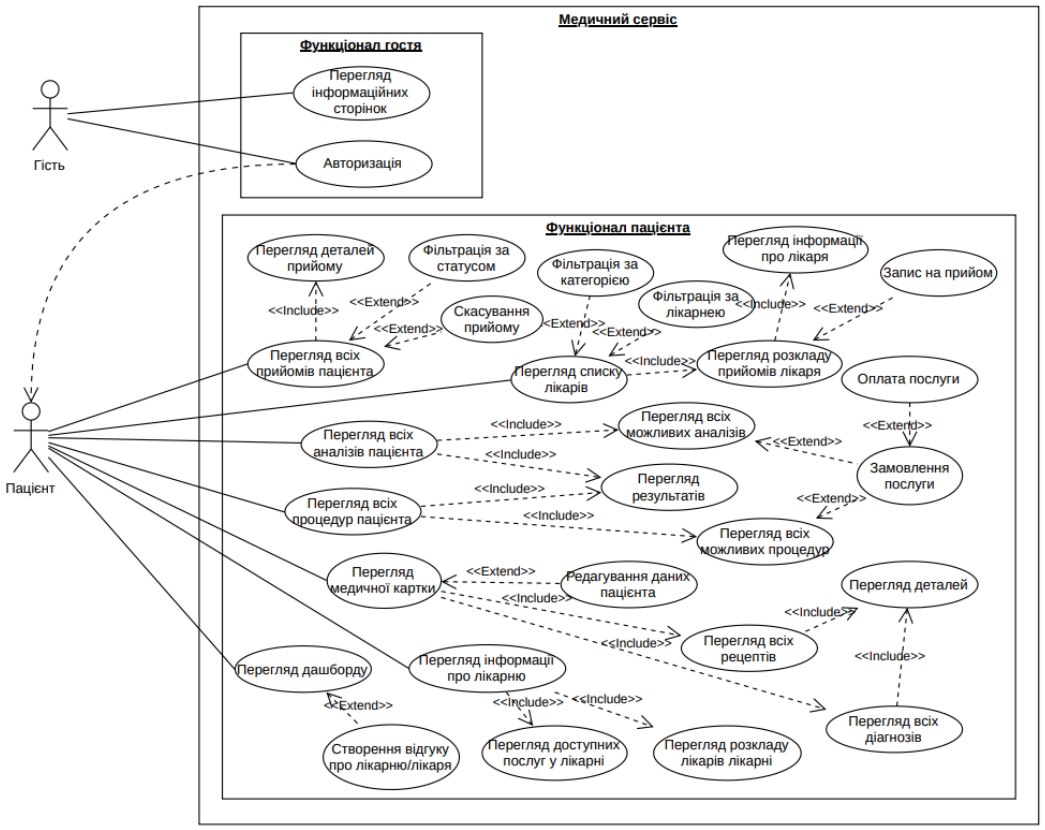

Рисунок 2.1 – Use-case діаграма для пацієнта (рисунок виконаний самостійно)

Лікар є основним користувачем системи з доступом до професійних інструментів роботи. Він має доступ лише до інформації своїх пацієнтів і не може змінювати дані інших лікарів або адмініструвати загальні налаштування системи. Для лікаря (див. рис. 2.2) відображено сценарії керування пацієнтами, призначення ліків, створення електронних рецептів, внесення результатів аналізів та оновлення медичних карток. Для полегшення роботи, лікар повинен мати інструменти пошуку пацієнтів за їх персональними даними, а також фільтрацію для зручного пошуку прийомів. Для медичного персоналу діаграма ідентична лікарській.

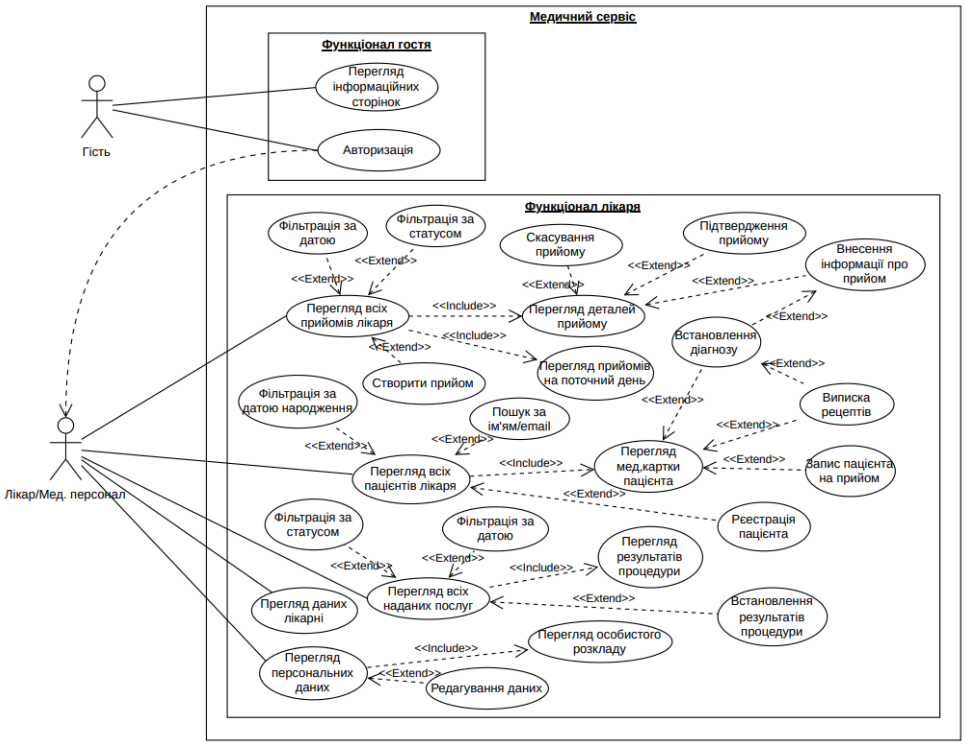

Рисунок 2.2 – Use-case діаграма для лікаря (рисунок виконаний самостійно)

Адміністратор має найбільший рівень доступу до системи і відповідає за її загальне функціонування. Для адміністратора (див. рис. 2.3) діаграма включає керування обліковими записами користувачів, контроль графіків роботи лікарів, моніторинг різноманітної аналітики та адміністрування системи, моніторинг всіх наданих послуг у лікарні. Крім основних потреб, адміністратор системи повинен мати доступ до ефективних інструментів пошуку та фільтрації даних. Це критично важливо для швидкої взаємодії з великою кількістю записів та забезпечення високої продуктивності роботи. Адміністратор потребує розвиненої системи пошуку облікових записів за ім’ям користувача, електронною поштою чи роллю. Також необхідна фільтрація всіх прийомів та наданих послуг.

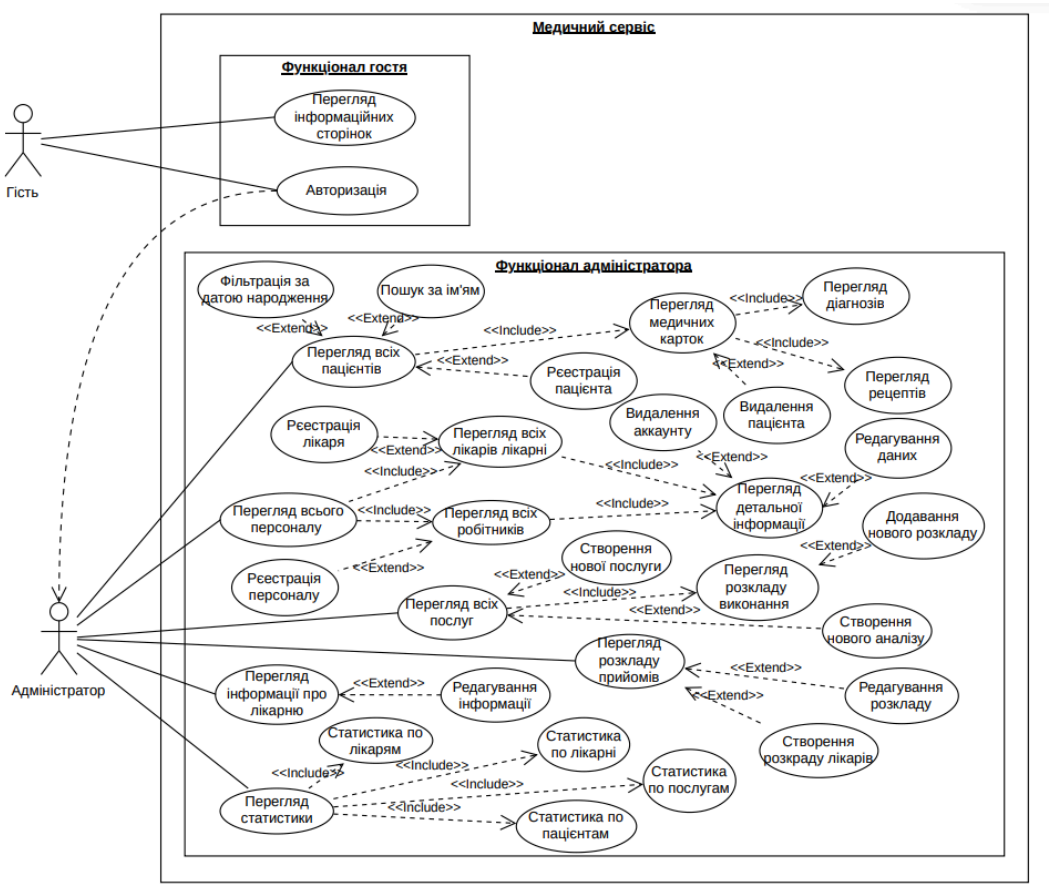

Рисунок 2.3 – Use-case діаграма для адміністратора (рисунок виконаний самостійно)

Система передбачає повний набір функціональностей для кожної категорії користувачів, забезпечуючи швидкий доступ до потрібної інформації та підтримуючи прозорість і зручність взаємодії у межах єдиної медичної інформаційної платформи.

2.3 Головна функціональність

Головна функціональність клієнтської частини програмної системи охоплює ключові можливості, необхідні для ефективної роботи медичного закладу. Основні функції системи включають:

- створення адміністратором облікових записів для пацієнтів, лікарів та медичного персоналу;

- авторизація користувачів із розмежуванням доступу за ролями;
- планування та управління графіками роботи лікарів і медичного персоналу із можливістю перегляду доступних прийомів;
- запис пацієнта на прийом до обраного лікаря;
- формування, ведення та перегляд електронних медичних карток пацієнтів лікарями та медичним персоналом;
- призначення лікарями медикаментів та формування електронних рецептів;
- замовлення лабораторних аналізів та перегляд результатів досліджень через особистий кабінет пацієнта і лікаря;
- генерація аналітичних медичних та фінансових звітів із можливістю перегляду інформації у вигляді графіків, діаграм та таблиць;
- пошук та фільтрація даних за різними критеріями;
- перегляд та редагування особистих даних профілю для всіх категорій користувачів.

3 АРХІТЕКТУРА ТА ПРОЄКТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

3.1 Загальні принципи побудови системи

Клієнтська частина програмної системи реалізована з використанням сучасного технологічного стеку для побудови динамічного, масштабованого та ефективного веб-застосунку.

Основною технологією для фронтенд-розробки обрано React – одну з найпопулярніших JavaScript-бібліотек для створення інтерактивних користувацьких інтерфейсів. Оскільки React дозволяє будувати ієрархії компонентів, архітектура на стороні клієнта може включати такі аспекти, як управління станами (за допомогою MobX), маршрутизація (React Router) та інтеграція з ресурсом (на стороні сервера) через REST API. React дозволяє організувати розробку у вигляді набору незалежних, перевикористовуваних компонентів, що забезпечує гнучкість та спрощує підтримку проекту.

Основною технологією обміну даними між клієнтською та серверною частинами реалізується через REST API, що забезпечує обмін у форматі JSON. Завдяки цьому користувачі отримують швидкий доступ до актуальної інформації без необхідності перезавантаження сторінок. Усі запити до серверної частини системи виконуються через стандартні HTTP-методи.

Комунікація з сервером реалізована за допомогою бібліотеки Axios, яка забезпечує зручний механізм для роботи з HTTP-запитами (див. рис. 3.1), обробкою відповідей та обробкою помилок мережевої взаємодії.

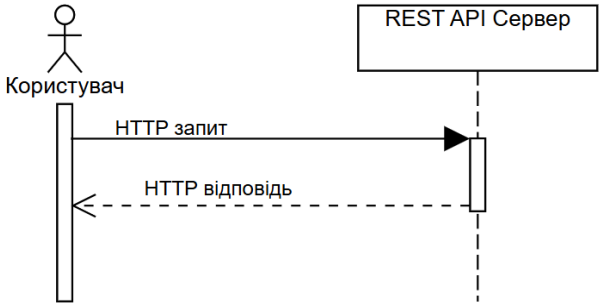

Рисунок 3.1 – Діаграма зв’язку клієнта з REST API сервером (рисунок виконаний самостійно)

Для організації навігації між сторінками застосунку використовується бібліотека React Router DOM, яка забезпечує маршрутизацію у межах єдиної сторінки додатку (SPA), без необхідності перезавантаження сторінки. Це дозволяє покращити користувацький досвід і забезпечити безшовний перехід між різними розділами системи.

Керування станом застосунку реалізовано за допомогою бібліотеки MobX. Вона забезпечує простий та ефективний механізм реактивного оновлення інтерфейсу при зміні стану даних, що особливо важливо для великих проектів зі складною структурою взаємодії компонентів.

Аутентифікація та авторизація користувачів реалізується через роботу з JSON Web Tokens. Для розшифрування токенів та витягування даних про користувача застосовується утиліта, яка дозволяє зчитувати інформацію про роль користувача після входу в систему та обмежувати або надавати доступ до певних функціональностей інтерфейсу відповідно до призначених прав доступу. Також визначається час життя токена та інші параметри без необхідності додаткових запитів до сервера.

Взаємодія між клієнтом та сервером організована таким чином, що після аутентифікації користувача на клієнті зберігається токен авторизації у локальному сховищі браузера. При кожному запиті до API цей токен автоматично передається в заголовках запиту для верифікації сесії. Сервер

перевіряє дійсність токена та у разі успіху виконує відповідні операції. Така організація обміну забезпечує безпечну та контрольовану роботу системи. Нижче на рисунку 3.2 наведена спрощена діаграма розгортання програмної системи, яка відображає основні компоненти та взаємозв'язки між ними.

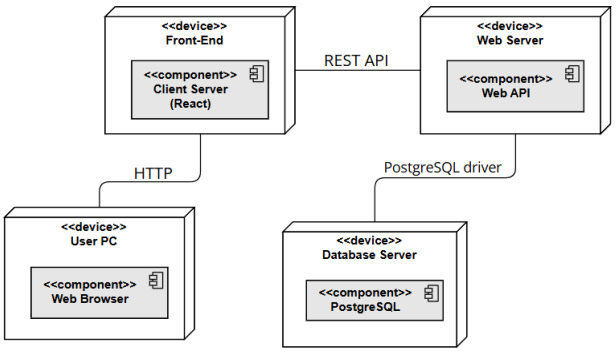

Рисунок 3.2 – Діаграма розгортання програмної системи (рисунок виконаний самостійно)

На даній діаграмі видно, що користувачі взаємодіють із фронтенд-застосунком, який працює у браузері та здійснює обмін даними із сервером через безпечні API-запити. Сервер обробляє запити, взаємодіє з базою даних та повертає оброблену інформацію клієнту для відображення.

Загальні принципи побудови системи орієнтовані на створення швидкого, масштабованого, безпечного веб-застосунку, який можна легко розширювати та підтримувати в майбутньому.

3.2 Проєктування архітектури ПЗ

При розробці клієнтського застосунку було використано модульну архітектуру. За принципом цієї архітектури додаток розділяється на логічні модулі або розділи, які відповідають за певні частини функціональності. Такий підхід дозволяє легко розширювати, підтримувати і масштабувати проєкт без необхідності суттєвого перероблення існуючого коду [3].

Кожен модуль може містити в собі свою власну логіку, обробники подій, стилі, шаблони та інші ресурси, які використовуються в цьому модулі. Це забезпечує високу модульність та ізоляцію коду, що полегшує підтримку та повторне використання компонентів. Використання CSS-модулів або styled-components дозволяє кожному модулю мати свої власні стилі, які не будуть конфліктувати з іншими частинами додатку.

Фізична структура застосунку організована логічно відповідно до функціонального призначення файлів та директорій. На рисунку 3.3 наведено основні елементи структури.

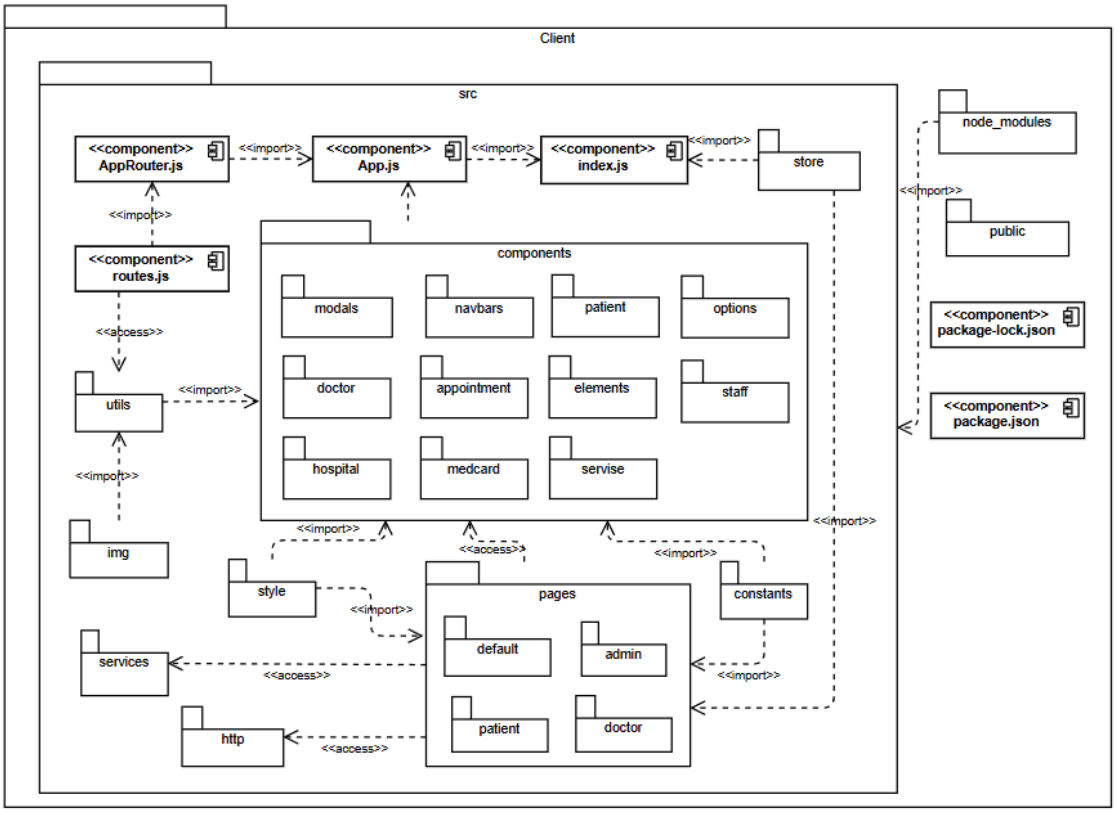

Рисунок 3.3 – Діаграма пакетів клієнтської частини (рисунок виконаний самостійно)

Директорія components об’єднує всі візуальні елементи інтерфейсу. Сюди входять як загальні компоненти для різних сторінок, які також поділено по

окремим папкам, так і спеціалізовані модальні вікна (modals), навігаційні панелі (navbars) та елементи інтерфейсу (elements).

Директорія http містить функціонал для роботи з API сервера через HTTP-запити. Тут централізовано налаштовані всі взаємодії фронтенду з бекендом, використовуючи бібліотеку Axios.

Директорія img відповідає за зберігання всіх графічних ресурсів: іконок, зображень та логотипів, які використовуються в інтерфейсі застосунку.

Директорія pages містить повноцінні сторінки, які відповідають за представлення основних функцій системи. Сторінки згруповані за ролями користувачів.

Директорія store реалізує зберігання тимчасових даних застосунку за допомогою станів MobX. Тут керується глобальний стан системи, наприклад, дані користувача після входу або інформація про поточні сесії.

Директорія style містить усі файли стилізації, що відповідають за візуальний вигляд компонентів та сторінок.

Директорія utils містить допоміжні константи та утиліти, зокрема константи для іменування маршрутів у системі.

Директорія services містить всю бізнес-логіку, що забезпечує взаємодію з backend-частиною, а також інші утиліти, пов’язані з функціональною логікою додатка, які не належать до частини інтерфейсу користувача.

Директорія constants використовується для зберігання статичних значень, які не змінюються під час виконання програми.

Окрім цього, у кореневій папці розміщені важливі файли для ініціалізації та запуску застосунку:

- App.js – головний компонент, що визначає базову структуру сторінок і обгортає їх відповідною маршрутизацією.
- index.js – стартова точка застосунку, де відбувається ініціалізація рендерингу в DOM та підключення провайдерів для стану застосунку.

- routes.js – файл, де оголошені маршрути, розділені за ролями користувачів, що забезпечує контроль доступу до окремих частин інтерфейсу.

Застосована модульна архітектура забезпечує програмній системі гнучкість, масштабованість, зручність підтримки та розширення функціональності в майбутньому. Чітка організація структури коду та використання стандартних підходів до проектування дозволяють ефективно управляти складністю системи при її розвитку.

3.3 Створення UI / UX дизайну системи

У рамках розробки клієнтської частини програмної системи велика увага приділялася створенню якісного UI/UX дизайну. Основна мета цього етапу полягала у розробці зручного, інтуїтивно зрозумілого та естетично привабливого інтерфейсу користувача, що відповідає сучасним вимогам до медичних інформаційних систем.

Для розробки прототипу інтерфейсу було використано спеціалізоване дизайнерське середовище Figma, яке дозволяє створювати інтерактивні макети, прототипи та забезпечує зручну співпрацю між членами команди [4].

На початковому етапі розробки макету було проведено аналіз основних сценаріїв взаємодії користувачів із системою: пацієнтів, лікарів, медичного персоналу та адміністраторів. Це дозволило чітко визначити, які елементи інтерфейсу повинні бути доступні для кожної ролі, які функції найчастіше використовуються і як краще організувати навігацію.

На основі зібраних вимог був створений макет системи у Figma, який зображено у додатку В. Він охоплює основні сторінки й екрани: вхід, особисті кабінети користувачів різних ролей (пацієнта, лікаря, адміністратора), сторінки запису на прийом, перегляду медичних карток, управління фінансами, перегляду результатів аналізів, управління графіками тощо.

Макет був виконаний у відповідності до принципів чистого,

мінімалістичного дизайну з акцентом на простоту навігації та легкість сприйняття інформації [5].

Навігаційна структура системи була побудована таким чином, щоб користувачі могли максимально швидко дістатися потрібних функцій: через бічні або верхні навігаційні панелі для лікарів та адміністраторів, а також через спрощене меню для пацієнтів.

Для системи обрано спокійну кольорову палітру, характерну для медичних застосунків, що підкреслює професійність і довіру. Типографіка підбиралася з урахуванням доступності – великі кеглі, чіткі шрифти для легкої читабельності навіть на невеликих екранах.

Було розроблено набір базових UI-компонентів, таких як кнопки, поля введення, списки, картки записів, модальні вікна, навігаційні панелі тощо, що використовуються по всьому застосунку для уніфікації вигляду інтерфейсу.

Розробка макету у Figma та комплексне опрацювання UI/UX дизайну дозволили сформувати базу для побудови якісного, сучасного і ефективного інтерфейсу клієнтської частини системи. Завдяки цьому користувачі зможуть легко взаємодіяти із системою, виконувати необхідні дії швидко й без помилок, а сам медичний заклад отримає зручний інструмент для автоматизації роботи та підвищення якості обслуговування пацієнтів.

4 ОПИС ПРИЙНЯТИХ ПРОГРАМНИХ РІШЕНЬ

4.1 Реалізація маршрутизації користувачів

Одним із ключових аспектів побудови фронтенд-архітектури програмної системи є реалізація ефективної маршрутизації між сторінками. Для реалізації маршрутизації у проекті використовується бібліотека React Router, яка дозволяє легко керувати переходами в односторінковому застосунку (SPA), додавати динамічні та вкладені маршрути, а також реалізовувати захищені маршрути [6], що є обов’язковим критерієм при створені медичного застосунку.

У програмній системі передбачено три основні ролі: адміністратор, лікар та пацієнт. Кожна з ролей має доступ до свого набору сторінок, а також до унікального функціоналу. Для реалізації розмежування доступу була розроблена система приватних та публічних маршрутів.

Всі маршрути організуються в єдиному централізованому файлі, де вони поділені за типами доступу: публічні (publicRoutes), адміністративні (adminRoutes), для лікарів (doctorRoutes) і для пацієнтів (patientRoutes).

Нижче наведено програмний код оголошення маршрутів пацієнта:

```
export const patientRoutes = [
  { path: PATIENT_PANEL_ROUTE, Component: PatientDashboard },
  { path: PATIENT_MEDCARD_ROUTE, Component: PatientMedCard },
  { path: PATIENT_ANALYSIS_ROUTE, Component: PatientAnalys },
  // ...
];
```

Усі маршрути передаються до головного маршрутизатора AppRouter, який визначає, які з них повинні бути активними відповідно до поточної ролі користувача. Визначення ролі ґрунтується на основі збережених даних у глобальному стані через MobX. Після авторизації користувача та визначення його ролі система автоматично перенаправляє його на відповідну панель – адміністратора, лікаря або пацієнта.

Нижче наведено функцію, яка повертає відповідний масив маршрутів згідно з роллю поточного користувача:

```
const renderPrivateRoutes = () => {
    switch (user.role) {
      case 'Admin': return adminRoutes;
      case 'Doctor': return doctorRoutes;
      case 'Patient': return patientRoutes;
      default: return [];
    }
  };
```

Важливою частиною реалізації стало забезпечення захисту приватних маршрутів. Якщо користувач не авторизований або його роль не визначена, доступ до захищених сторінок буде заборонено. Усі спроби переходу на такі сторінки будуть автоматично перенаправлені на головну сторінку. Завдяки такому розмежуванню прав доступу система гарантує повну безпеку даних.

4.2 Реалізація аутентифікації та авторизації

Аутентифікація та авторизація відіграють ключову роль у забезпеченні захисту будь-якої інформаційної системи, особливо коли мова йде про обробку персональних даних у медичній системі. В клієнтській частини застосунку реалізовано логіку безпечного входу користувача, перевірки його ролі, збереження токена та організацію перенаправлення до відповідного інтерфейсу.

Процес аутентифікації реалізовано у вигляді форми входу, де користувач вводить свої облікові дані – електронну пошту та пароль (див. рис. 4.1). Після натискання кнопки «Увійти в систему», на стороні клієнта відправляється запит до серверного API, який виконує валідацію введених даних. У разі успішного входу система виконує додатковий запит до API для перевірки користувача, який повертає інформацію про користувача, зокрема його роль. Програмну реалізацію обробки авторизації наведено у додатку Г.1.

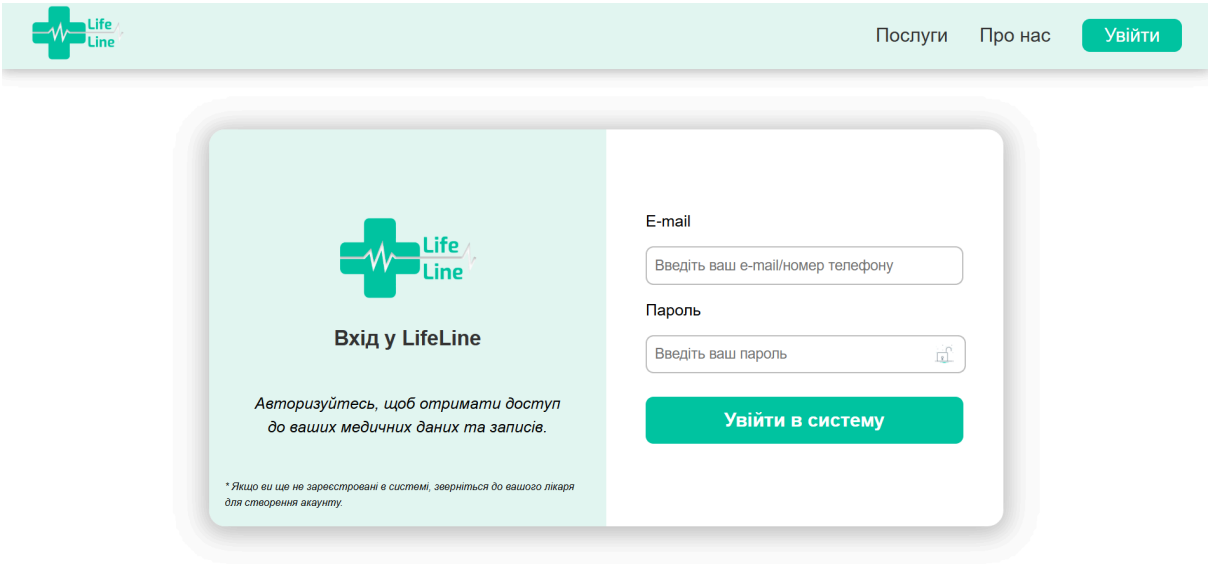

Рисунок 4.1 – Форма авторизації

Для збереження сесії та ідентифікації користувача при подальшому використанні застосунку, система використовує токен доступу (JWT). Отриманий токен зберігається у локальному сховищі браузера разом із роллю користувача, що дозволяє зберігати стан авторизації навіть після перезавантаження сторінки. На основі ролі користувача відбувається автоматичне перенаправлення до відповідної панелі системи.

Завдяки такій реалізації система забезпечує як зручність, так і безпеку користувацького входу, дозволяючи гнучко контролювати доступ до критично важливої інформації на основі ролей. Це є невід'ємною частиною надійного та масштабованого створюваного медичного веб-застосунку.

4.3 Взаємодія з API

Ефективна взаємодія між клієнтською та серверною частиною програмної системи забезпечується через стандартизовану технологію REST API. У клієнтській частині для обміну даними із сервером використовується бібліотека Axios – один із найпопулярніших HTTP-клієнтів у JavaScript-середовищі.

Для зручності та повторного використання коду була побудована централізована система запитів, яка базується на створенні двох екземплярів клієнта Axios:

- $host – використовується для публічних запитів, які не потребують авторизації;
- $authHost – використовується для авторизованих запитів, які передбачають обов’язкову передачу токена в заголовках.

Цей поділ дозволяє чітко розмежувати запити, які можна надсилати без автентифікації, від запитів, які потребують підтвердження особи користувача.

Кожен авторизований запит при зверенені до API автоматично доповнюється токеном авторизації завдяки налаштованому інтерсептору [7]. Це гарантує безпечну та стабільну передачу обмежених даних між клієнтом і сервером без дублювання логіки у кожному окремому запиті. Код інтерсептору наведено нижче:

```
const authInterceptor = config => {
    config.headers.authorization = `Bearer
${localStorage.getItem('token')}`
    return config
}

$authHost.interceptors.request.use(authInterceptor)
```

Уся логіка роботи із сервером винесена в окремий модуль. Такий підхід дозволяє централізовано керувати API-запитами, швидко знаходити потрібні функції та легко масштабувати код із додаванням нових маршрутів.

Кожна функція запиту реалізована як асинхронна функція, яка виконує HTTP-запит до певного API і повертає лише дані з відповіді сервера. Функція запиту, яка отримує медичні записи пацієнта наведена нижче:

```
export const fetchMedicalRecordsByPatientId = async (patientId) =>
{
  try {
```

```
    const { data } = await
$authHost.get(`api/medical-records/patient/${patientId}`);
    return data;
  } catch (error) {
    console.error("Помилка при отриманні медичних записів:",
error);
    throw error;
  }
};
```

Усі запити обгорнуті в конструкцію try...catch, яка дозволяє обробляти помилки та відображати користувачеві відповідні повідомлення з серверу. Це підвищує надійність взаємодії та допомагає запобігати раптовим збоям у роботі інтерфейсу.

4.4 Управління станом застосунку

У клієнтській частині програмної системи не менш важливою складовою роботи є ефективне управління станом. Для реалізації цієї задачі використовується бібліотека MobX, яка забезпечує реактивну модель керування даними на стороні фронтенду. Її головна перевага полягає в тому, що будь-які зміни у стані автоматично призводять до оновлення інтерфейсу, без необхідності ручного перерендерингу або складної синхронізації компонентів.

На відміну від більш громіздких підходів, таких як Redux, MobX надає більш просту та гнучку модель, засновану на спостережуваних об’єктах, автоматичних реакціях та обчислювальних значеннях. Це робить MobX ідеальним вибором для масштабованих SPA-застосунків із великою кількістю станів.

У рамках проекту реалізовано кілька сховищ, кожен із яких відповідає за окрему ділянку функціональності. Найважливішими серед них є UserStore, який зберігає інформацію про поточного користувача, стан авторизації та його роль. Завдяки цьому інші компоненти можуть реагувати на зміну авторизаційного стану. Програмний код цього сховища наведено нижче:

```
export default class User {
  constructor() {
    this._isAuth = !!localStorage.getItem("token");
    this._user = {};
    this._role = localStorage.getItem("role") || "";
    makeAutoObservable(this);
  }

  setIsAuth(bool) {
    this._isAuth = bool;
  }

  setUser(user) {
    this._user = user;
  }

  get isAuth() {
    return this._isAuth;
  }

  get user() {
    return this._user;
  }
}
```

Усі сховища створюються за допомогою конструктора та певного методу, який автоматично робить усі властивості реактивними [8]. На прикладі UserStore видно, що кожне поле має геттери та сеттери, які дозволяють контролювати доступ до внутрішнього стану. Це означає, що будь-яка зміна значення викличе оновлення компонентів, які на них підписані. Така модель ідеально підходить для медичних інформаційних систем, де важливо забезпечити точність, швидкість оновлення та зручність у відображенні персоналізованих даних.

4.5 Реалізація сторінок для різних ролей користувачів

Програмна система реалізовує рольову модель доступу, відповідно до якої кожна категорія користувачів має окремий набір інтерфейсних сторінок, що

відповідають її функціональним потребам. Сторінки розділені за ролям і мають окремі директорії в структурі проекту.

Кожна сторінка реалізована як самостійний React-компонент, що динамічно завантажує дані через API, керує станом, реагує на зміни й виводить контент згідно з призначенням. Компоненти реалізовані адаптивно та максимально універсально, щоб забезпечити повторне використання в інших частинах системи [9]. Приклад використання таких компонентів на одній зі сторінок наведено у додатку Г.2.

4.5.1 Сторінки пацієнта

Однією з ключових категорій користувачів є пацієнти, для яких створено повноцінну індивідуальну панель з доступом до персональних медичних даних та функцій. Сторінки, пов’язані з пацієнтом, розміщені у відповідному каталозі, і об’єднані в навігаційну структуру через рольову маршрутизацію.

Після успішної авторизації пацієнт потрапляє на головну сторінку особистого кабінету, яка виконує роль інформаційної панелі (див. рис. Д.1 у додатку Д). На цій сторінці користувач може побачити декілька кнопок для переходу до найбільш затребуваних розділів. Нижче них виводиться список майбутніх прийомів пацієнта, а також є модуль для створення відгуку про лікаря або лікарню, для цього створено окреме модальне вікно з формою створення відгуку (див. рис. Д.2 у додатку Д). Для кожної картки майбутнього прийому є можливість перегляду детальної інформації (див. рис. Д.3 у додатку Д). Зліва розташоване бічне меню з навігаційними пунктами: «Запис до лікаря», «Мої прийоми», «Аналізи», «Послуги», «Медична картка» та «Моя лікарня». Ця структура дозволяє пацієнту легко зорієнтуватися у доступному функціоналі і з легкістю перейти на необхідну сторінку.

Сторінка «Запис до лікаря» (див. рис. Д.4 у додатку Д) надає користувачу можливість переглядати список всіх лікарів, незалежно від лікарні, з фільтрами за категорією лікаря, лікарнею та містом. Для відображення списку лікарів

створена окрема картка, яка відображає ПІБ лікаря, його фото та деякі дані. В правому верхньому куті картки є кнопка, при натисканні на яку відкривається модальне вікно з повною інформацією про лікаря (див. рис. Д.5 у додатку Д). Щоб переглянути розклад прийомів лікаря є відповідна кнопка «Переглянути розклад», при натисканні на яку, відбувається перехід на сторінку з розкладом лікаря на тиждень (див. рис. Д.6 у додатку Д). З цієї сторінки пацієнт може безпосередньо забронювати прийом, вибравши вільний час з представлених часових слотів для відповідного дня (див. рис. Д.7 у додатку Д). Вся інформація оновлюється в реальному часі через виклик API. При натисканні на відповідний слот з'являється модальне вікно з підтвердженням створення запису, в якому надано повну інформацію про майбутній прийом.

У розділі «Мої прийоми» (див. рис. Д.8 у додатку Д) пацієнт переглядає всі свої прийоми, а саме: майбутні, минулі та скасовані. Кожен прийом відображається на сторінці як окрема картка, яка містить такі дані: статус прийому, ПІБ лікаря, дату та час прийому, а також місце прийому. Також кожна картка містить кнопку «Переглянути деталі» (див. рис. Д.3 у додатку Д), яка відкриває модальне вікно з повною інформацією про прийом. Для запланованих прийомів також є кнопка «Скасувати», яка відкриває модальне вікно, в якому необхідно вказати причину скасування (див. рис. Д.9 у додатку Д). В разі скасування прийому, запит відправляється на сервер і статус змінюється на «Скасовано». Для більш зручного перегляду списку прийомів, на сторінці реалізовано фільтр для сортування прийомів за їх статусами.

Сторінки «Аналізи» та «Послуги» (див. рис. Д.10 у додатку Д) є схожими по функціональності. На цих сторінках користувач може побачити всі свої виконані лабораторні аналізи та медичні послуги. Вони представлені як окремі картки, де кожен запис містить назву процедури, дату виконання та поточний статус («Очікується» або «Виконано»). Після того, як лікар вносить результати та позначає процедуру готовою, сторінка автоматично отримує оновлені дані через виклик API і статус змінюється на «Виконано», а кнопка «Переглянути

результати» стає активною. При натисканні на цю кнопку відкривається сторінка детального звіту з результатами (див. рис. Д.11 у додатку Д), які можливо переглянути у форматі PDF, та коментарями лікаря. Також в горі сторінок з аналізами пацієнта міститься кнопка «Замовити аналіз/послугу», яка перенаправляє користувача на сторінку зі списком всіх можливих аналізів та послуг в лікарнях (див. рис. Д.12 у додатку Д). Для кожного елементу списку зазначено його назву, дані лікарні та ціну. Поруч з кожним елементом є кнопка, яка відкриває модальне вікно для бронювання та оплати послуги (див. рис. Д.13 у додатку Д). Для цього необхідно обрати дату та час виконання, і після цього натиснути кнопку сплатити, яка відкриває вікно для сплати через PayPal. В разі успішної оплати на сторінці прийомів пацієнта з'являється відповідний прийом на виконання послуги, якщо ж оплата не пройшла, то користувач отримує повідомлення про це. Для перегляду детальної інформації про послугу чи аналіз необхідно клікнути на назву процедури, після чого відкриється модальне вікно з усіма даними.

Пункт «Медична картка» відкриває сторінку, на якій зібрано всю інформацію про пацієнта: його персональні дані, діагнози та призначені ліки (див. рис. Д.14 у додатку Д). В центрі цієї сторінки створена картка з персональними та контактними даними пацієнта. Якщо пацієнту потрібно оновити свої особисті дані, використовується сторінка «Редагування особистої інформації». На ній розташована форма з відповідними полями для введення даних (див. рис. Д.15 у додатку Д). Після внесення змін користувач натискає кнопку «Зберегти», і дані оновлюються через відповідний API-запит. У разі успіху з'являється невелике сповіщення про успішне збереження, а у разі помилки – повідомлення про необхідність виправити невалідні поля. Також пацієнт має можливість змінити свой пароль, для цього створено окреме модальне вікно яке відкривається при натисканні кнопки «Змінити пароль» (див. рис. Д.16 у додатку Д).

Нижче самої картки з даними є дві колонки, які відображають останні встановлені діагнози та виписані рецепти. Аби переглянути повний список діагнозів або рецептів, під кожною колонкою є кнопка, яка перенаправляє користувача на відповідні сторінки які містять повний список записів пацієнта. На сторінці зі всіма діагнозами (див. рис. Д.17 у додатку Д) пацієнт може переглядати свої записи з прийомів від різних лікарів. Кожен запис відображає дату візиту, короткий діагноз та кнопку «Детальніше», яка відкриває сторінку з повним описом хвороби (див. рис. Д.18 у додатку Д). На цій сторінці виводяться всі дані: місце прийому та відповідаючий лікар, записи лікаря стосовно хвороби, список призначених препаратів для лікування хвороби.

Сторінка «Мої рецепти» (див. рис. Д.19 у додатку Д) демонструє всі електронні рецепти, виписані лікарями. Вони відображаються у виді списку, в якому вказано назву, дату, коли був виписаний рецепт, та дату закінчення дії рецепту. Для кожного рецепту наведено назви препаратів, дозування, інструкції, частота прийому та особливі вказівки (див. рис. Д.20 у додатку Д). Є можливість завантажити його у PDF-форматі для зручного друку (див. рис. Д.21 у додатку Д).

Нарешті, «Моя лікарня» – це сторінка з детальною інформацією про медичну установу, в якій пацієнт стоїть на обліку (див. рис. Д.22 у додатку Д). Вгорі сторінки є компонент який відображає такі дані лікарні, як назва, адреса та контактні дані. Нижче наведено два списки, які відображають повні списки послуг чи аналізів, що проводяться у лікарні. При натисканні на назву послуг відкривається модальне вікно з повною інформацією, а при натисканні кнопки «Замовити» відкривається модалка для бронювання та оплати послуги (див. рис. Д.13 у додатку Д). Також внизу є список всіх працюючих лікарів в відповідній лікарні, з можливістю перегляду повної інформації про них та швидким доступом до сторінки запису на прийом через кнопку «Записатися». Щоб переглянути розлад прийому всіх лікарів необхідно натиснути на кнопку

«Розклад прийому лікарів», після чого відкривається сторінка з таблицею часів прийому лікарів на тиждень (див. рис. Д.23 у додатку Д).

Комплексна реалізація сторінок для пацієнта охоплює всі етапи взаємодії користувача з системою. Завдяки єдиній навігації та узгодженому UI/UX дизайну пацієнт може інтуїтивно знаходити необхідні розділи, бачити актуальні дані в режимі реального часу та оперативно виконувати потрібні дії в межах свого кабінету.

4.5.2 Сторінки лікаря

У медичній інформаційній системі роль лікаря є центральною в частині обробки даних пацієнтів, включаючи всі необхідні маніпуляції з медичними даними. Для забезпечення повного робочого процесу лікаря у системі реалізовано низку сторінок, об’єднаних у панелі лікаря, яка доступна після авторизації.

Користувач із роллю лікаря після входу в систему автоматично перенаправляється на головну сторінку панелі лікаря (див. рис. Д.24 у додатку Д). Ця панель містить кнопки для швидкого переходу до найбільш затребуваних сторінок або елементів. Під цими кнопками відображено дані про лікаря у виді картки, аналогічної до картки лікарів на сторінках пацієнта, а нижче відображено розклад лікаря на тиждень. Зліва розташоване бічне меню з розділами: «Пацієнти», «Прийоми», «Послуги», «Моя лікарня». Структура інтерфейсу побудована таким чином, щоб лікар міг швидко отримати доступ до потрібних розділів і працювати з пацієнтами без зайвих переходів і затримок.

Однією з основних сторінок є «Пацієнти», де відображається перелік усіх пацієнтів, закріплених за конкретним лікарем (див. рис. Д.25 у додатку Д). Вгорі сторінки є кнопка яка відкриває модальне вікно для створення нового облікового запису пацієнта (див. рис. Д.26 у додатку Д). При такому створені, пацієнт автоматично буде стояти на обліку у лікаря, який його зареєстрував. На сторінці реалізована система пошуку за іменем і датою народження, що

дозволяє швидко знайти потрібного пацієнта. Кожен запис у списку містить наступну інформацію: ПІБ пацієнта, дату народження та електронну адресу пацієнта. Кожен елемент списку містить кнопку «Переглянути медичну картку», яка відкриває розгорнутий профіль з персональною інформацією пацієнта, а також даними про здоров'я (див. рис. Д.27 у додатку Д). Лікар має можливість редагувати дані пацієнта, за виключенням фото профілю, натиснувши кнопку «Редагувати дані» на картці пацієнта. Також у лікаря є доступ до медичної картки, яка включає історію хвороб та електронні рецепти. Ці дані відображаються як окремі компоненти у картці пацієнта і дають можливість переглядати повні дані натиснувши відповідну кнопку. З метою забезпечення зручного доступу до медичної інформації було впроваджено функцію пошукового поля. У разі, коли лікарю необхідно записати пацієнта на прийом, є відповідна кнопка вгорі сторінки, для створення прийому пацієнта, яка відкриває модальне вікно запису на прийом (див. рис. Д.28 у додатку Д).

Розділ «Прийоми» надає лікарю інформацію про прийоми на поточний день (див. рис. Д.29 у додатку Д). Прийоми відображаються аналогічно до картка у пацієнта, але з необхідними даними для лікаря. Є можливість скасування прийому. Для того, щоб переглянути всі прийоми лікаря за весь час необхідно натиснути кнопку «Всі прийоми». Інтерфейс цієї сторінки дозволяє лікарю бачити як поточні прийоми на сьогодні, так і переглядати історію візитів за попередні дні або заплановані прийоми (див. рис. Д.30 у додатку Д). За допомогою фільтру діапазону дат, лікар може обрати будь-яку дату або проміжок дат і отримати список пацієнтів на цей період. На цій сторінці кожен запис прийому містить інформацію про пацієнта, час візиту, а також статус. Натиснувши на будь-який прийом, лікар переходить на сторінку детального перегляду інформації, де можна залишити коментар (див. рис. Д.31 у додатку Д), встановити новий діагноз, за допомогою відкритого модального вікна при натискані кнопки «Додати» поруч з заголовком секції діагнозів (див. рис. Д.32 у додатку Д). При створені нового діагноза лікар може додавати необхідні

препарати, для цього реалізовано інше модальне вікно (див. рис. Д.33 у додатку Д). Також передбачена можливість змінити статус візиту, наприклад, відзначити як «Завершено» або «Скасовано».

У розділі «Послуги» лікар бачить список медичних процедур, що необхідно виконати, або які вже були виконані (див. рис. Д.34 у додатку Д). З метою спрощення процесу пошуку необхідних процедур, сторінка містить пошукове поле, яке здійснює пошук за даними пацієнта, та фільтри за датою та типом процедури (аналіз чи медична послуга). Для виконаних послуг лікар повинен внести результати, якщо ж цього ще не було зроблено. Ті послуги, які ще не мають внесенного результату, відображаються у списку з кнопкою «Внести дані», при натисканні на яку відкривається сторінка для внесення результату (див. рис. Д.35 у додатку Д). Також лікар може залишити коментар або висновок, що буде автоматично доступний пацієнту у його кабінеті. Вже внесені результати лікар може переглянути, та в разі необхідності змінити.

Сторінка «Моя лікарня» (див. рис. Д.36 у додатку Д) містить інформацію про медичну установу, в якій працює лікар. Тут він може переглядати дані лікарні, всі процедури що проводяться у лікарні, та весь медичний персонал. Для зручної комунікації між лікарями певної медичної установи, реалізовано кнопку «Написати» у картці кожного лікаря, яка відкриває електронну пошту для зв’язку з лікарем. Лист відправляється на ту електронну адресу, яка вказана в контактній інформації лікаря.

Загалом, інтерфейс лікаря побудований таким чином, щоб забезпечити швидкий доступ до всієї медичної та організаційної інформації, мінімізувати час на заповнення даних та забезпечити повну електронну взаємодію з пацієнтами. Усі сторінки працюють через API-запити до серверної частини, що забезпечує динамічне оновлення даних, високу продуктивність та точність.

4.5.3 Сторінки адміністратора

Адміністративна панель програмної системи є одним із найфункціональніших і найвідповідальніших модулів, оскільки забезпечує повний контроль за всіма процесам, користувачами та даними медичного закладу. Інтерфейс адміністратора розроблений для управляючого персоналу медичних установ, який відповідає за організаційні процеси, керування персоналом, фінансову звітність і аналітику.

Після входу в систему адміністратор автоматично потрапляє на головну сторінку – панель адміністратора (див. рис. Д.37 у додатку Д). Як і у двох попередніх панелях, на цій містяться картки для швидкого доступу до деяких функції. Панель містить дані про адміністратора у виді картки, такої самої, як у лікаря, але є додаткова можливість редагувати власні дані. Для цього створено окреме модальне вікно (див. рис. Д.38 у додатку Д). Зліва розташоване бічне меню зі всіма необхідними розділами: «Лікарі», «Пацієнти», «Прийоми», «Послуги», «Розклад», «Аналітика», «Лікарня». Меню містить всі необхідні розділи для зручної навігації адміністратора між певними структурованими даними. Це дозволяє швидко перемикатися між сторінками та знаходити необхідну інформацію.

При натисканні на бічній панелі кнопки «Лікарі» відкривається сторінка керування лікарями та медичним персоналом (див. рис. Д.39 у додатку Д). Вона включає список усіх зареєстрованих лікарів та працівників із можливістю сортування за спеціальністю та пошуком по імені лікаря. Для більш зручного відображення даних, сторінка поділена на дві вкладки «Лікарі» та «Медичний персонал». На цій сторінці адміністратор може створити нового працівника, використовуючи модальне вікно що відкривається (див. рис. Д.40 у додатку Д). Також біля кожного поля з даними працівника є кнопка для редагування даних, з можливістю повного видалення даних працівника з системи (див. рис. Д.41 у додатку Д).

Сторінка «Пацієнти» надає адміністратору список всіх зареєстрованих пацієнтів у лікарні (див. рис. Д.42 у додатку Д). Кожен профіль відкривається у вигляді електронної картки, з можливістю переглянути повну інформацію про пацієнта. При потребі адміністратор може змінити персональні дані користувача або видалити профіль. Додатково, з профілю пацієнта є доступ до перегляду медичних даних, які відображаються у форматі, подібному до лікарського, але в режимі читання.

Для перегляду всіх прийомів у лікарні на поточний день необхідно відкрити «Прийоми». На цій сторінці адміністратор може бачити всі створені пацієнтами або лікарями записи, з можливістю перегляду повної інформації або ж скасування прийому. Інтерфейс цієї сторінки ідентичний до інтерфейсу сторінки у панелі лікаря. Для того, щоб переглянути повний список всіх майбутніх, минувших або скасованих прийомів, необхідно перейти на сторінку всіх прийомів (див. рис. Д.43 у додатку Д) натиснувши кнопку «Всі прийоми». Для зручного перегляду даних реалізовано фільтри та поле пошуку за певними параметрами. Також додано можливість створення прийому. Форма для створення прийому структурно подібна до форми для лікаря, але містить додаткове поле, яке дозволяє обрати відповідного спеціаліста, до якого потрібно записати пацієнта на прийом (див. рис. Д.44 у додатку Д).

Сторінка «Послуги» предназначена для керування послугами. На цій сторінці можна переглянути всі надані послуги у лікарні (див. рис. Д.45 у додатку Д) з можливістю перегляду результатів тих процедур, які позначені як виконані. Також адміністратор має можливість створювати, редагувати або видаляти медичні послуги, які надаються в установі (див. рис. Д.46 у додатку Д). Адміністратор встановлює процедуру, доступні години для прийому та лікарів, які можуть їх проводити (див. рис. Д.47 у додатку Д). Ці дані синхронізуються з іншими модулями, щоб пацієнти бачили лише актуальні й доступні послуги.

Для моніторингу та керування розкладом прийому лікарів, створено відповідну сторінку «Розклад» (див. рис. Д.48 у додатку Д). При відкриті цієї сторінки адміністратор може побачити загальний графік прийомів лікарів на тиждень. Щоб створити новий розклад для лікаря, достатньо натиснути на кнопку «Створити новий розклад» вгорі сторінки після чого відкриється модальне вікно з шаблоном встановлення часу прийому на кожен день тижня для обраного лікаря зі списку (див. рис. Д.49 у додатку Д). Для того, щоб переглянути дані розкладу конкретного лікаря необхідно натиснути на ім’я лікаря у таблиці і тоді відбувається перехід на сторінку з графіком лікаря, де можна побачити скільки вільних та зайнятих місць залишилось (див. рис. Д.50 у додатку Д). Якщо натиснути на слот, який позначено як зайнятий, можна побачити дані пацієнта який записан на прийом. При натисканні на вільний слот можна видалити його з розкладу.

Ще одним ключовим компонентом є сторінка «Аналітика», яка відображає графіки та таблиці зі сформованими даними (див. рис. Д.51 у додатку Д). Адміністратор може переглядати медичну статистику таку як: топ лікарів, кількість відвідувачів за тиждень/місяць, найбільш затребувані медичні послуги, а також фінансові звіти. Ці дані автоматично оновлюються на основі активності в системі.

Окремим інтерфейсом реалізовано управління лікарнею на сторінці «Лікарня» (див. рис. Д.52 у додатку Д). Тут адміністратор може редагувати дані лікарні, наприклад, контактну інформацію, графік роботи лікарні або ж навіть її назву (див. рис. Д.53 у додатку Д). Додатково на цій сторінці можна переглядати відгуки користувачів про лікарню.

Загалом панель адміністратора є невід’ємним інструментом для централізованого керування всім функціоналом медичного закладу. Вона дозволяє гнучко управляти даними, автоматизувати адміністративні процеси та оперативно приймати управлінські рішення на основі аналітики й актуальної інформації.

4.6 Адаптивна верстка інтерфейсу

Однією з важливих вимог до сучасного веб-застосунку є забезпечення адаптивності інтерфейсу. Це особливо важливо у випадку системи «LifeLine», коли користувачі можуть працювати з платформою як із комп’ютерів, так і з планшетів чи смартфонів. Усі сторінки клієнтської частини було спроектовано з урахуванням адаптивного дизайну, що дозволяє зберігати зручність користування незалежно від типу пристрою.

Для досягнення цієї мети у проекті використано комбінацію CSS Flexbox та CSS Grid, які забезпечують гнучке компонування блоків інтерфейсу, а також медіа-запити, що дозволяють застосовувати різні стилі залежно від ширини екрана [10]. Також у певних компонентах застосовується адаптивна стилізація безпосередньо в JavaScript за допомогою логіки, що визначає розмір екрана та змінює стилі на льоту, використовуючи змінні та об’єднання базових стилів із адаптивними.

Кожна сторінка, кожен блок або елемент мають адаптивне масштабування, тобто змінюють розміри, відступи, шрифти або розташування залежно від ширини вікна браузера. Це дає змогу зберігати єдиний вигляд і функціональність застосунку на всіх пристроях. До прикладу, для елементів із фіксованою шириною на комп’ютері передбачено зменшення ширини на планшетах або мобільних пристроях, що дозволяє покращити сприйняття контенту.

Для прикладу, в одному з CSS-файлів модального вікна застосовано таке адаптивне масштабування:

```
@media (max-width: 1024px) {
  .modal {
    width: 70%;
    max-width: 90%;
  }
  .form {
    margin: 20px 20px;
  }
}
```

Також у багатьох компонентах використовуються умовні стилі всередині JavaScript-файлів, через визначення ширини екрана за допомогою хука або ж змінної. Це дозволяє динамічно застосовувати різні стилі без необхідності писати окремий CSS-файл.

```
const responsiveStyles = {
  headerBox: {
    ...baseStyles.headerBox,
    padding: isSmallScreen ? '16px 20px' :
baseStyles.headerBox.padding,
  },
  };
```

Це підхід зручний у випадках, коли елемент отримує стилі з обчислюваного об’єкта style в React, особливо для елементів інтерфейсу із динамічним вмістом.

Перевірити, наскільки адаптивною є клієнтська частина застосунку, можна за допомогою інструментів розробника, доступних у будь-якому браузері. Ці інструменти дозволяють побачити, як одна й та сама сторінка відображатиметься на різних пристроях. Використання цих інструментів дозволяє діагностувати проблеми з адаптивністю та швидко редагувати сторінки, що допомагає створювати якісніші застосунки. Для наглядного прикладу було обрано сторінку медичної картки пацієнта, яку протестовано на пристроях з різним розміром єкрану: смартфон (див. рис. 4.2), планшет (див. рис. 4.3) та комп’ютер (див. рис. 4.4).

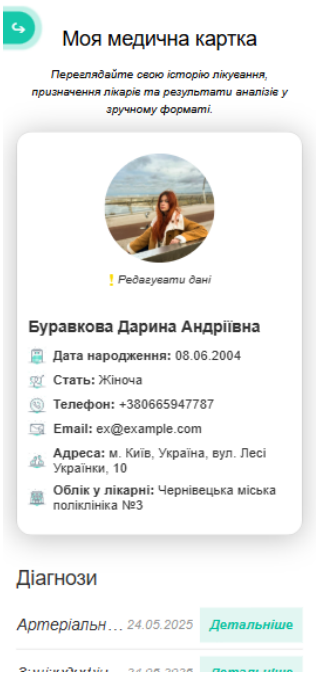

Рисунок 4.2 – Перегляд медичної картки на мобільному девайсі «iPhon 14 Pro Max»

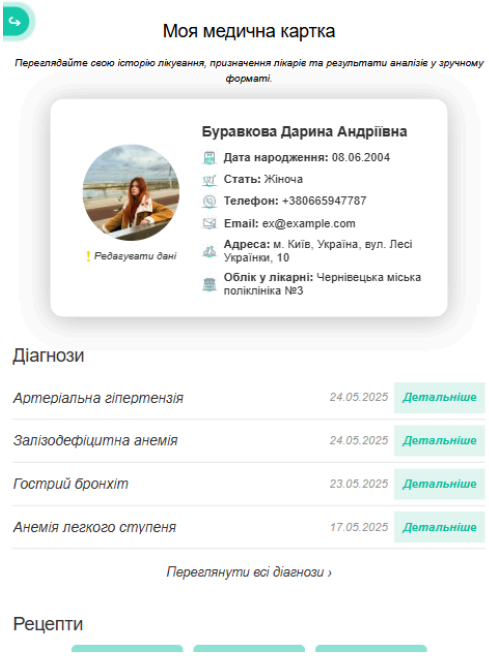

Рисунок 4.3 – Перегляд медичної картки на планшетному девайсі «iPad Mini»

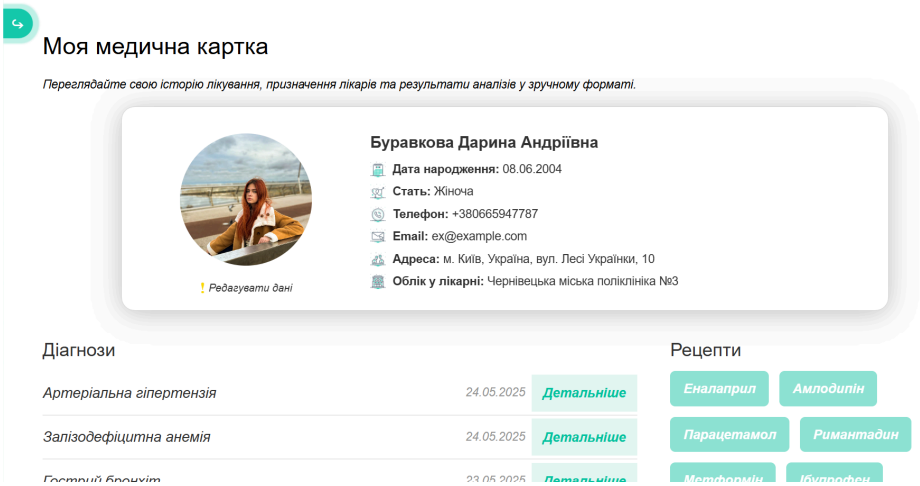

Рисунок 4.4 – Перегляд медичної картки на екрані комп’ютера

Завдяки використанню сучасних методів компонування, інтерфейс у версії для великих екранів зберігає широку структуру з панелями навігації, великими таблицями та зручними полями введення. У мобільній версії ці елементи

автоматично перетворюються у вертикальні блоки, спрощується доступ до меню через «hamburger menu», а великі таблиці перетворюються на списки або ж таблиці, що прокручуються по горизонталі. Поля вводу, кнопки та модальні вікна масштабуються для комфортної взаємодії на сенсорних пристроях.

5 ТЕСТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

5.1 Функціональне тестування

Важливу роль в розробці інформаційної системи відіграє тестування функціональності веб-застосунку, оскільки саме в результаті тестування можна виявити вразливі місця, усунути помилки у його ключових функціях та забезпечити високу якості веб-додатку. Функціональне тестування є ідеальним вибором для тестування застосунку, адже його основна мета – переконатися, що всі сторінки, елементи інтерфейсу та логіка роботи системи відповідають вимогам, викладеним у технічному завданні та документації до проекту.

У рамках функціонального тестування було здійснено покрокову перевірку сценаріїв взаємодії для кожної з ролі користувачів. Було протестовано кожен функціональний маршрут, починаючи з авторизації користувача, і закінчуючи взаємодією з медичними даними, створенням записів, реєстрацією користувачів та адміністративними модулями керування.

Тестування програмної системи проводилось мануально. Для кожної основної сторінки та ключового сценарію взаємодії були сформовані тест-кейси, складені на основі технічної специфікації. Кожен тест-кейс включав набір дій, очікувану поведінку інтерфейсу та результат тесту. Завдяки цьому вдалось систематизувати процес перевірки та забезпечити повне покриття функціональності.

Для кожної сторінки перевірялися базові функції: відображення компонентів, реакція на дії користувача, правильність переходів між сторінками, коректність обробки введених даних, а також обробка помилок і повідомлень. Наприклад, на сторінці авторизації перевірялась валідація полів, повідомлення про неіснуючий акаунт, невірний пароль або неправильний формат пошти.

На панелі пацієнта було протестовано: перегляд електронної медичної картки, перегляд медичних записів, запис до лікаря, скасування прийому, перегляд результатів аналізів/медичних послуг, редагування особистих даних та

замовлення медичних послуг. Усі ці функції були виконані з дотриманням заданого сценарію та відповідними повідомленнями користувачу про успішну або невдалу дію.

У панелі лікаря тестувались: перегляд пацієнтів, робота з медичними картками, створення медичних записів, створення рецептів, внесення результатів послуг та перегляд особистого графіку. Також перевірялась валідація всіх взаємодій з формами вводу та виведення відповідних повідомлень при різноманітних маніпуляціях у системі.

Для адміністратора перевірялися всі розділи керування – від додавання нових користувачів до створення графіків прийомів. Було протестовано створення нових лікарів, редагування даних пацієнтів, оновлення даних користувачів, перегляд усіх прийомів у системі та навігація між статистичними модулями.

В разі виявлення помилок при проходженні тесту, все фіксувалося у тест-кейсах та в подальшому виносилося для доопрацювання функціональності елементу. Такий підхід дозволив швидко усувати недоліки у реалізації та забезпечити високу якість готового інтерфейсу.

В результаті проведення повного тестування системи було визначено, що клієнтська частина системи реалізована відповідно до функціональних вимог, забезпечує стабільну роботу інтерфейсу для кожної ролі користувача та має надійну логіку взаємодії між компонентами. Детальне тестування забезпечило узгодження між очікуваною поведінкою користувача та фактичною реакцією системи, що є критично важливим для медичного програмного забезпечення, де точність і надійність відіграють ключову роль.

5.2 Інтеграційне та системне тестування

Після перевірки кожного компонента окремо було проведено інтеграційне тестування, яке дозволило перевірити взаємодію між модулями, сторінками та компонентами. На цьому етапі тестування акцент було зроблено на

послідовність та узгодженість роботи системи як єдиного цілого. Зокрема, перевірялося, як відображаються та передаються дані між компонентами після виконання певних дій користувачем, наскільки узгоджено працюють маршрутизація, стан додатку та асинхронна обробка запитів.

Особлива увага приділялася правильності навігації між сторінками в залежності від ролі користувача. Наприклад, після входу в систему користувача з роллю «Пацієнт» автоматично перенаправляється на свою панель, де перевіряється доступність усіх функцій, таких як перегляд медичної картки, запис до лікаря або замовлення аналізів. Усі переходи між цими сторінками відбуваються з передачею контексту користувача, збереженням стану авторизації, відповідною URL-навігацією, а також з коректним оновленням вмісту сторінок.

Оскільки велика частина сторінок приймати дані через параметри з динамічних маршрутів, необхідно було перевірити коректність даних які передаються. Прикладом таких сторінок є медичні записи в панелі лікаря або пацієнта, так як дані кожного елемента передавалися через URL із ідентифікаційним номером. Для таких сторінок важливо, щоб система не лише правильно відображала компонент, а й щоб дані коректно завантажувалися з API на основі переданого параметра, і оновлювали інтерфейс у режимі реального часу.

Паралельно з інтеграційним тестуванням було проведено системне (End-to-End) тестування, яке охопило повний життєвий цикл користувача в межах системи. Кожен сценарій тестування описував типовий маршрут користувача починаючи з авторизації та закінчуючи будь-якою можливою функціональною дією на панелі користувача. Наприклад, для адміністратора це може бути маршрут від авторизації до створення розкладу прийомів лікаря, де для досягнення цілі йому необхідно відкрити сторінку з розкладом, відкрити модальне вікно для створення розкладу, внести відповідні дані, зберегти їх та одразу побачити в таблиці розкладу щойно створені дані. На кожному з етапів

перевірялося, чи правильно обробляється запит до API, чи зберігається інформація між переходами, та чи не виникає логічних помилок у сценарії.

Іншим прикладом системного сценарію виступає взаємодія користувача з роллю «Лікар». Після авторизації лікар відкриває список пацієнтів, обирає конкретного пацієнта, відкриває його медичну картку, додає дані хвороби, і в результаті пацієнт бачить ці дані у своєму інтерфейсі. Таке тестування дозволяє переконатися, що система коректно обробляє взаємодію між ролями та даними, які передаються через API.

На основі визначених помилок за результатам тестування було внесено кілька виправлень: оновлено логіку обробки маршрутів, покращено відображення помилок та обробку некоректних переходів. Також були виявлені та усунені проблеми з асинхронністю даних при одночасному оновленні кількох станів.

Після завершення інтеграційного та системного тестування було підтверджено, що клієнтська частина системи працює як єдине узгоджене ціле, всі зв’язки між модулями коректно реалізовані, а користувачі можуть безперешкодно виконувати повний набір дій, передбачених функціональністю програмної системи.

ВИСНОВКИ

У ході виконання кваліфікаційної роботи було повністю реалізовано клієнтську частину медичної інформаційної системи «LifeLine», головною метою якої є автоматизування ключових процесів у медичних закладах. Процес розробки інформаційної системи охоплював всі аспекти: від аналізу предметної галузі й постановки задачі до проектування програмного забезпечення, реалізації та тестування розробленого застосунку.

У результаті роботи отримано повнофункціональну клієнтську частину медичної інформаційної системи, як розроблена з використанням сучасного стеку технологій: мови програмування JavaScript, бібліотеки React, бібліотеки управління станом застосунку MobX, управління маршрутами через React Router, та HTTP-клієнта Axios для відправки запитів на сервер через REST API. Структура проекту була розроблена за принципами модульної архітектури, що забезпечує масштабованість, за рахунок легкого впровадження нових модулів, зручність у підтримці та перевикористання компонентів. Для кожної ролі користувача було створено окремі набори сторінок, які забезпечують виконання всіх необхідні дії згідно з встановленими функціональними вимогами.

Візуальна частина інтерфейсу розроблена на основі створених макетів, з дотриманням сучасних принципів UI/UX-дизайну, доступності, зручності та логіки взаємодії. Особливу увагу приділено адаптивній верстці, завдяки чому сайт чудово адаптується до всіх типів пристроїв: комп’ютерів, планшетів та смартфонів.

Особливу увагу було приділено безпеці доступу. Реалізовано надійний механізм авторизації, за рахунок збереження токена, перевірки ролі та обмеження прав доступу до сторінок системи залежно від ролі користувача.

Результати виконаної роботи свідчать про готовність клієнтської частини системи «LifeLine» до впровадження у медичних закладах. Створена інформаційна система відповідає встановленим вимогам і очікуванням, та

сприяє підвищенню ефективності роботи медичного персоналу, якості обслуговування пацієнтів та прозорості управління процесами у сфері охорони здоров’я, тим самим задовільняючи потреби всіх типів користувачів.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Медична інформаційна система. Helsi.me. URL: https://helsi.me/ (дата звернення: 02.05.2025).

2. Медична соціальна платформа. Medics. URL: https://medics.ua/ (дата звернення: 02.05.2025).

3. Чиста архітектура: Мистецтво розроблення програмного забезпечення / пер. з англ. І. Бондар-Терещенко. – Харків : Вид-во «Ранок» : Фабула, 2019. – 368 с.

4. Figma Learn – Product documentation. *Figma Design*. URL: https://help.figma.com/hc/en-us/categories/360002042553-Figma-Design (дата звернення: 23.04.2025).

5. Gothelf J., Seiden J. Lean UX: Creating Great Products with Agile Teams. – O'Reilly Media, Incorporated, 2021. – 256 p.

6. Шайтан В., Овчаренко І. Роутинг у React. *Hillel blog*. URL: https://blog.ithillel.ua/articles/routing-in-react (дата звернення: 28.05.2025).

7. Interceptors | Axios Docs. *Axios*. URL: https://axios-http.com/docs/interceptors (дата звернення: 29.05.2025).

8. React integration · MobX. *MobX*. URL: https://mobx.js.org/react-integration.html (дата звернення: 29.05.2025).

9. Wieruch R. The Road to React: Your journey to master plain yet pragmatic React.js. Independently published, 2020. 226 p.

10. Meyer E. A., Weyl E. CSS : The Definitive Guide: Web Layout and Presentation. O'Reilly Media, Incorporated, 2023. 1128 p.

11. Chernova A., Buravkova D. *Medical Information Management System: Software Requirements Specification. Version 1*. 01.10.2024. URL: https://docs.google.com/document/d/17D2pk9acoPfpkmW1FZbBB4s4OWIakL0B/edit (дата звернення: 01.10.2024).

12. Github.NureBuravkovaDaryna/2025_B_PI_PZPI-21-6_Buravkova_D_A URL: https://github.com/NureBuravkovaDaryna/2025_B_PI_PZPI-21-6_Buravkova_D_A