Міністерство освіти і науки України
Національний технічний університет
«Дніпровська політехніка»

Навчально-науковий
інститут електроенергетики
(навчально-науковий інститут)
Факультет інформаційних технологій
(факультет)
Кафедра інформаційних технологій та комп'ютерної інженерії
(повна назва)

ПОЯСНЮВАЛЬНА ЗАПИСКА
кваліфікаційної роботи ступеня магістра

здобувача вищої освіти Соколовського Дмитра Олександровича
(ПІБ)
академічної групи 123м-23-1
(шифр)
спеціальності 123 Комп'ютерна інженерія
(код і назва спеціальності)
за освітньо-професійною програмою «Комп'ютерна інженерія»
(офіційна назва)

на тему "Обґрунтування структури комп'ютерної системи стоматологічної клініки «Amel Dental Clinic» із урахуванням методів рендерингу CSR та SSR вбудованого в бот веб-додатку"
(назва за наказом ректора)

Керівники	**Прізвище, ініціали**	**Оцінка за шкалою**		**Підпис**
		рейтинговою	**інституційною**	
кваліфікаційної роботи	проф. Цвіркун Л.І.			
розділів:				
синтез системи	доц. Ткаченко С.М.			
розроблення програмного забезпечення	доц. Бешта Л.В.			

Рецензент				

Нормоконтролер	проф. Цвіркун Л.І.			

Дніпро
2024

ЗАТВЕРДЖЕНО:
завідувач кафедри
інформаційних технологій
та комп'ютерної інженерії
(повна назва)
_____ В.В. Гнатушенко
(підпис) (ініціали, прізвище)

«__ ___» _____ 2024 року

ЗАВДАННЯ
на кваліфікаційну роботу
ступеня магістра
(бакалавра, магістра)

здобувачу вищої освіти Соколовського Д.О. **академічної групи** 123м-23-1
(прізвище та ініціали) (шифр)
спеціальності 123 Комп'ютерна інженерія
за освітньо-професійною програмою «Комп'ютерна інженерія»
(офіційна назва)
на тему «Обґрунтування структури комп'ютерної системи стоматологічної клініки «Amel Dental Clinic» із урахуванням методів рендерингу CSR та SSR вбудованого в бот веб-додатку», затверджену наказом ректора НТУ «Дніпровська політехніка» від 17 жовтня 2024 р. №1388-с

Розділ	Зміст	Термін виконання
Стан питання та постановка завдання	На основі матеріалів практик, інших науково-технічних джерел сформулювати наукове завдання, конкретизувати предмет та мету досліджень	11.10.2024
Теоретичний	Обґрунтувати теоретичну базу системи для запису на прийом до клініки "Amel Dental Clinic"	25.10.2024
Синтез системи	Розробка комп'ютерної системи для запису на прийом до клініки "Amel Dental Clinic"	15.11.2024
Розроблення програмного забезпечення	Розробка програмного забезпечення системи для запису на прийом до клініки "Amel Dental Clinic"	29.11.2024
Експериментальний розділ	Проведення і обробка результатів експериментів порівняння методів рендерингу CSR та SSR вбудованого в чат-бот веб-додатку	06.12.2024

Завдання видано _____ проф. Л.І. Цвіркун
(підпис керівника) (ініціали, прізвище)

Дата видачі 06 вересня 2024 р.

Дата подання до екзаменаційної комісії 10.12.2024 р.

Прийнято до виконання _____ Д.О. Соколовський
(підпис здобувача вищої освіти) (ініціали, прізвище)

РЕФЕРАТ

ПОЯСНЮВАЛЬНА ЗАПИСКА 127 с., 30 рис., 7 табл., 2 дод., 17 джерел.

КОМП’ЮТЕРНА СИСТЕМА, TELEGRAM, API, БАЗА ДАНИХ, ВЕБ-ДОДАТОК, JAVASCRIPT, FRONTEND, BACKEND, SSR, CSR, NODE, REACT, NEXT, EXPRESS, MONGODB, ЧАТ-БОТ

Об’єкт розробки − комп'ютерна система медичної клініки “Amel Dental Clinic” з чат-ботом в Telegram для запису на прийом.

Мета роботи − розробка чат-боту для управління записами до медичної клініку, який включає дослідження методів рендерингу CSR і SSR веб-додатку, інтегрованому в чат-бот в Telegram.

Методи дослідження – застосовувались методи порівняльного аналізу показників швидкості завантаження, оцінок та метрик продуктивності з використанням інструментів Chrome DevTools та Lighthouse.

У розділі «Стан питання і постановка завдання» виконано аналіз існуючих інформаційних систем в медичній сфері, описано їх проблеми та загальні тенденції, а також сформульовано завдання дослідження.

У розділі «Теоретичний розділ» обрано та обґрунтовано методи експериментального дослідження і технології для реалізації системи, порівняно види веб-додатків, чат-ботів та різних методів рендерингу.

У розділі «Синтез системи» сформулювано цілі системи, технічні вимоги до системи, розроблено схему функціональної структури системи, а також специфікацію апаратних засобів.

У розділі «Розробка програмного забезпечення системи телеграм-бота для запису до клініки» проведено розробку чат-бота з вбудованим веб-додатком для запису на прийом до клініки, опис розробленої програми та її тестування.

У розділі «Експериментальний розділ» проведено експеримент з порівняння ефективності та продуктивності методів рендерингу веб-додатку.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

AJAX – Asynchronous JavaScript and XML (Асинхронний JavaScript і XML).

AI – Artificial Intelligence (Штучний інтелект).

API – Application Programming Interface (Інтерфейс програмування додатків).

BSON – Binary JSON (Бінарний формат JSON).

CLS – Cumulative Layout Shift (Кумулятивне зміщення макета).

CORS – Cross-Origin Resource Sharing (Спільне використання ресурсів між джерелами).

CRUD – Create, Read, Update, Delete (Створити, прочитати, оновити, видалити).

CSR – Client-Side Rendering (Клієнтське рендеринг).

CSRF – Cross-Site Request Forgery (Міжсайтове підроблення запиту).

CSS – Cascading Style Sheets (Каскадні таблиці стилів).

DDR4 – Double Data Rate 4 (Подвійна швидкість передачі даних 4-го покоління).

DOM – Document Object Model (Об’єктна модель документа).

FCP – First Contentful Paint (Перше відображення контенту).

HTML – HyperText Markup Language (Мова розмітки гіпертексту).

HTTP(S) – HyperText Transfer Protocol (Secure) (Протокол передавання гіпертексту (з шифруванням)).

IOS – iPhone Operating System (Операційна система для iPhone).

ISR – Incremental Static Regeneration (Інкрементальне статичне оновлення).

JSON – JavaScript Object Notation (Нотація об’єктів JavaScript).

JWT – JSON Web Token (Веб-токен у форматі JSON).

LCP – Largest Contentful Paint (Найбільше відображення контенту).

MPA – Multi-Page Application (Багатосторінковий додаток).

MQL – MongoDB Query Language (Мова запитів MongoDB).

MTP – Mobile Transport Protocol (Мобільний транспортний протокол).

NLP – Natural Language Processing (Обробка природної мови).

PWA – Progressive Web App (Прогресивний веб-додаток).

SEO – Search Engine Optimization (Пошукова оптимізація).

SI – Speed Index (Індекс швидкості).

SPA – Single Page Application (Односторінковий додаток).

SQL – Structured Query Language (Мова структурованих запитів).

SSG – Static Site Generation (Статичне генерування сайтів).

SSD – Solid State Drive (Твердотільний накопичувач).

SSL – Secure Sockets Layer (Рівень захищених сокетів).

SSR – Server-Side Rendering (Серверне рендеринг).

TTI – Time to Interactive (Час до інтерктивності).

TLS – Transport Layer Security (Транспортний рівень захисту).

UI – User Interface (Інтерфейс користувача).

URL – Uniform Resource Locator (Уніфікований локатор ресурсів).

UTF – Unicode Transformation Format (Формат трансформації Юнікоду).

UX – User Experience (Досвід користувача).

XSS – Cross-Site Scripting (Міжсайтовий скриптинг).

XML – Extensible Markup Language (Розширювана мова розмітки).

JSX – JavaScript XML (Розширення синтаксису для JavaScript).

ПІБ – Прізвище, ім'я, по батькові.

ДБЖ – джерело безперебійного живлення.

ВСТУП

Розробка інтерактивних систем для обміну інформацією через месенджери є однією з ключових сучасних тенденцій в області програмної інженерії. Чат-боти, як автоматизовані системи взаємодії з користувачами, дозволяють ефективно оптимізувати робочі процеси та забезпечувати доступ до інформації цілодобово, виконуючи безліч функцій, від підтримки користувачів до управління бізнес-процесами.

Вивчення сучасних розробок у сфері чат-ботів, зокрема для медичних закладів, показує значний прогрес у технологіях штучного інтелекту та автоматизації. Вітчизняні та зарубіжні дослідники активно досліджують різноманітні аспекти розробки чат-ботів, зокрема їх ефективність, адаптивність та вплив на якість обслуговування пацієнтів.

Попри значний прогрес у розвитку чат-ботів, все ще існують ряд проблем, які потребують вирішення: технічні обмеження, недостатня інтеграція з існуючими системами, конфіденційність і безпека даних, обмежена інтерактивність та користувацький досвід.

Актуальність роботи полягає в зростаючій ролі чат-ботів у сучасному бізнес-середовищі та їхньому впливі на покращення обслуговування клієнтів. У сфері медичних послуг, зокрема, ефективна організація запису пацієнтів є критично важливою для забезпечення високої якості обслуговування, зменшення черг та оптимізації робочого часу медичного персоналу.

Мета дослідження – дослідження методів рендерингу веб-додатків у середовищі Telegram, зокрема клієнтського (CSR) та серверного рендерингу (SSR), для підвищення якості користувацького досвіду, продуктивності та стабільності інтерфейсу медичного додатку.

Завдання дослідження:

– проаналізувати особливості клієнтського та серверного рендерингу, їх переваги та недоліки в контексті інтеграції з Telegram;

– провести серію експериментів із застосуванням CSR та SSR для веб-додатку медичної клініки за показників швидкості завантаження, оцінок та метрик продуктивності з використанням інструментів Chrome DevTools та Lighthouse;

– визначити найоптимальніший метод рендерингу для веб-додатку в Telegram медичної клініки та обґрунтувати рекомендації щодо його використання;

– здійснити порівняльний аналіз результатів з метою обґрунтування впливу обраного методу рендерингу на продуктивність і досвід користувача.

Об’єкт дослідження – процес інтеграції веб-додатків із платформою Telegram у рамках медичних інформаційних систем.

Предмет дослідження – методи рендерингу (SSR та CSR) та їх вплив на продуктивність, швидкість завантаження, стабільність інтерфейсу та користувацький досвід у середовищі Telegram.

Для досягнення поставленої мети використовувалися методи веб-розробки (SSR та CSR рендеринг), методи порівняльного аналізу показників швидкості завантаження, оцінок та метрик продуктивності з використанням інструментів Chrome DevTools та Lighthouse.

Наукові положення:

1. Встановлено, що метод SSR є більш оптимальним для медичних додатків, вбудованих у Telegram, оскільки він забезпечує стабільність інтерфейсу (низький CLS) і кращу видимість контенту за рахунок швидшого завантаження готових сторінок.

2. Виявлено, що CSR підходить для додатків з потребою у швидкому оновленні контенту та плавному переході між сторінками, проте поступається SSR за показниками FCP та Speed Index, що може впливати на користувацький досвід при низькій швидкості мережі.

Наукові результати:

1. Розроблено рекомендації для вибору оптимального методу рендерингу веб-додатків у Telegram на основі показників продуктивності та стабільності інтерфейсу.

2. Одержані експериментальні результати підтвердили, що SSR забезпечує більш ефективну роботу веб-додатків медичних клінік завдяки стабільності інтерфейсу та зручному перегляду контенту.

Обґрунтованість та достовірність наукових положень, висновків та рекомендацій підтверджуються використанням апробованих методів аналізу продуктивності з інструментами Chrome DevTools та Lighthouse, актуальними даними щодо роботи Telegram, а також експериментальним підтвердженням результатів теоретичних досліджень.

Практичне значення отриманих результатів полягає у створенні рекомендацій для інтеграції веб-додатків у середовище Telegram та вибору методу рендерингу, що забезпечує ефективну роботу медичних додатків. Це дозволить створювати додатки, які працюють стабільно та швидко, що є важливим для задоволення потреб користувачів та забезпечення кращого досвіду взаємодії з медичними сервісами.

1 СТАН ПИТАННЯ І ПОСТАНОВКА ЗАВДАННЯ

1.1 Загальні тенденції у розвитку та цифровізації галузі охорони здоров’я

Медична галузь є однією з найважливіших складових соціальної інфраструктури, яка відіграє ключову роль у забезпеченні здоров’я та добробуту населення. У сучасному світі медицина охоплює широкий спектр напрямків – від профілактики та діагностики захворювань до лікування і реабілітації пацієнтів. Розвиток цієї галузі значною мірою визначається рівнем наукових досліджень, технологічним прогресом та доступністю якісних медичних послуг.

Значний вплив на медицину має глобалізація, яка сприяє поширенню передових методик лікування та інноваційних технологій. Завдяки цьому медичні установи у всьому світі поступово впроваджують новітні інструменти для покращення діагностики, лікування та управління пацієнтами. Одним із ключових напрямків розвитку є використання інформаційних технологій для автоматизації робочих процесів, покращення взаємодії між лікарями та пацієнтами, а також забезпечення доступності медичних послуг у віддалених регіонах.

Розширення можливостей медичної галузі супроводжується розвитком таких сфер, як:

1) діагностика – сучасні методи, зокрема використання штучного інтелекту, дозволяють швидше і точніше визначати захворювання;

2) терапія – інноваційні підходи до лікування забезпечують ефективніші результати та мінімізують ризики для пацієнтів;

3) телемедицина – дистанційне консультування стає дедалі популярнішим завдяки його зручності для пацієнтів і лікарів;

4) профілактика – інформаційні кампанії та моніторинг здоров'я сприяють зменшенню ризику виникнення захворювань.

Значення медичної галузі для суспільства неможливо переоцінити. Розвиток медицини є одним із ключових чинників, що впливають на якість життя, середню тривалість життя та загальний рівень добробуту населення.

1.2 Характеристика об’єкту впровадження

Об’єкт впровадження – стоматологічна клініка «Amel Dental Clinic».

Стоматологічна клініка «Amel Dental Clinic», заснована у 2017 році, спеціалізується на наданні стоматологічних послуг, включаючи лікування зубів, ясен, профілактику та догляд за ротовою порожниною. Серед запропонованих послуг – лікування карієсу, кореневих каналів, захворювань ясен, видалення зубів, імплантація, протезування, встановлення коронок і брекетів. Також клініка пропонує загальнотерапевтичну допомогу пацієнтам.

Клініка використовує функціональну організаційну структуру управління, яка забезпечує поділ обов'язків між спеціалізованими підрозділами. Керівники цих підрозділів відповідають за виконання завдань і ефективність роботи своїх відділів.

Адміністративний відділ включає IT-відділ, бухгалтерію, відділ кадрів та відділ маркетингу. Медичний відділ представлений різними спеціалізованими підрозділами, такими як пародонтологічний, терапевтичний, ортопедичний, ортодонтичний, хірургічний, анестезіологічний, дитячий та загальнотерапевтичний.

Медичний директор координує роботу всієї клініки, тоді як адміністративний відділ забезпечує виконання організаційних функцій. Така структура дозволяє ефективно взаємодіяти адміністративному та медичному персоналу.

Бухгалтерія займається нарахуванням зарплат, веденням фінансового обліку, контролем за оплатою послуг пацієнтами та підготовкою податкової звітності.

IT-відділ забезпечує функціонування комп’ютерних систем і підтримку користувачів у разі технічних проблем з обладнанням чи програмами.

Відділ кадрів відповідає за пошук і прийом на роботу кваліфікованих фахівців. Кандидати проходять співбесіду з керівниками підрозділів та медичним директором.

Відділ маркетингу працює над просуванням послуг клініки та залученням нових клієнтів.

Медичні підрозділи спеціалізуються на діагностиці та лікуванні пацієнтів відповідно до своєї сфери компетенції.

Організаційну структуру клініки «Amel Dental Clinic» наведено на рисунку 1.1

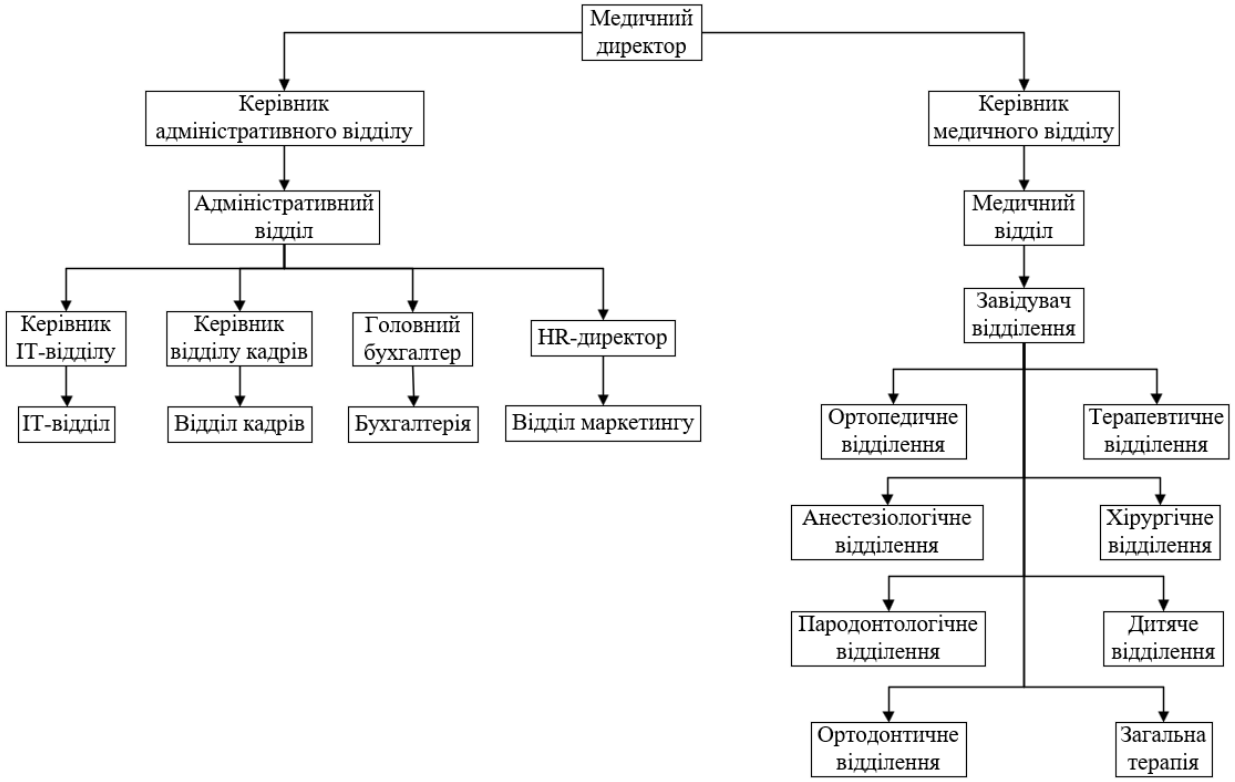

Рисунок 1.1 – Схема організаційної структури стоматологічної клініки «Amel Dental Clinic»

Структурні підрозділи стоматологічної клініки «Amel Dental Clinic» розташовані у двох окремих будівлях, де надаються однакові послуги.

Перша локація клініки знаходиться за адресою: 49000, Україна, Дніпропетровська область, м. Дніпро, бул. Катеринославський, 2, у торгово-діловому комплексі "Босфор". Тут орендовані приміщення на другому та п’ятому поверхах (рисунок 1.2 А).

Друга локація розташована за адресою: 49000, Україна, Дніпропетровська область, м. Дніпро, бул. Слави, 2Б. Тут клініка має власну двоповерхову будівлю (рисунок 1.2 Б).

Відстань між цими будівлями становить 5300 метрів у прямій лінії.

Топографічна схема розміщення структурних підрозділів клініки показана на рисунку 1.2

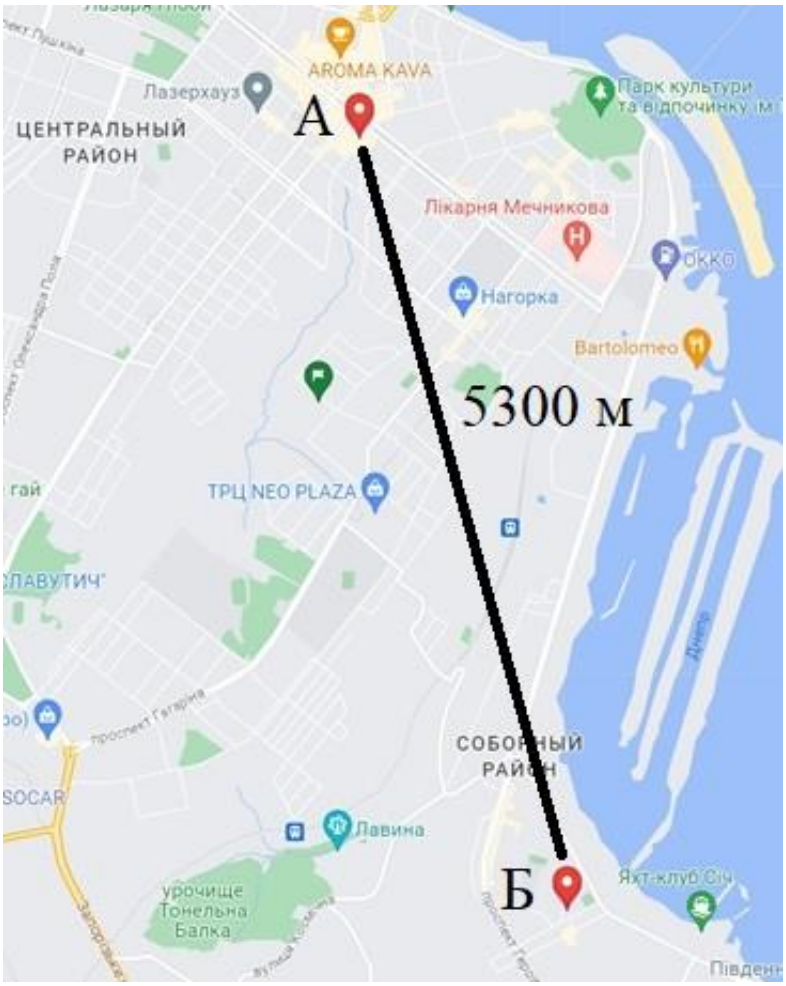

Рисунок 1.2 – Топографічна схема розміщення структурних підрозділів стоматологічної клініки «Amel Dental Clinic»

1.3 Проблеми сучасних систем

Медичні інформаційні системи відіграють ключову роль у забезпеченні ефективної роботи клінік та медичних центрів. Проте, попри стрімкий розвиток

технологій, існує низка проблем, які обмежують їх ефективність та вплив на якість медичних послуг.

1) складність інтеграції – багато існуючих систем є автономними і не підтримують інтеграцію з іншими платформами, такими як месенджери, CRM-системи чи державні бази даних. Це ускладнює обмін даними між різними підрозділами клініки або з іншими установами;

2) відсутність персоналізованого підходу – багато систем не враховують специфіку конкретної клініки, що призводить до обмеженої функціональності або необхідності доопрацювань. Наприклад, у стоматологічних клініках часто потрібні спеціалізовані модулі для обліку стоматологічних процедур та витратних матеріалів, яких у стандартних рішеннях може не бути;

3) проблеми з безпекою даних – медична інформація є конфіденційною і потребує високого рівня захисту. Проте деякі системи не забезпечують належного рівня безпеки, що може призвести до витоку даних або несанкціонованого доступу до них;

4) низька доступність мобільних платформ – популярність мобільних пристроїв вимагає адаптації систем для їх використання. Багато медичних додатків або веб-сервісів мають недостатньо оптимізовані мобільні версії, що ускладнює доступ до інформації для пацієнтів та лікарів;

5) обмежені можливості аналітики та автоматизації – сучасні системи часто не мають розвинених модулів для аналітики даних та автоматизації процесів, таких як складання графіків прийому лікарів, прогнозування потреб у витратних матеріалах або автоматичне нагадування пацієнтам про прийоми;

6) проблеми з масштабованістю – багато систем не пристосовані до роботи з великими обсягами даних або до зростання клініки;

7) обмежений функціонал для взаємодії з пацієнтами – більшість систем пропонують базовий набір функцій, таких як запис на прийом чи оплата послуг. Проте відсутність інтерактивних функцій, наприклад, інтеграції з популярними

месенджерами для спілкування з пацієнтами, значно обмежує зручність та ефективність взаємодії.

Отже, сучасні медичні системи стикаються з багатьма викликами, які потребують вирішення для підвищення якості обслуговування пацієнтів та оптимізації роботи медичних закладів. Інтеграція з популярними платформами, зокрема месенджерами, створення оптимізованих мобільних версій та впровадження розширених аналітичних інструментів можуть стати вирішенням багатьох з перелічених проблем.

1.4 Постановка завдання дослідження

Основною метою даного дослідження є розробка чат-боту для медичної клініки з використанням платформи Telegram, який повинен дозволяти пацієнтам зручно записуватися на прийом, а лікарям клініки переглядати свій графік роботи. Особлива увага приділяється дослідженню методів рендерингу (CSR та SSR) веб-додатку в контексті його продуктивності та взаємодії з платформою Telegram.

Для досягнення поставленої мети необхідно вирішити наступні завдання:

– проаналізувати існуючі системи та платформи для чат-ботів, а також видів веб-додатків та методів їхнього рендерингу;

– провести огляд технологій та інструментів для створення чат-ботів;

– обрати технології та інструменти для створення чат-боту;

– провести аналіз інструментів для дослідження;

– обґрунтувати і обрати методи експериментального дослідження;

– визначити цілі впровадження системи;

– визначити вимоги до реалізації системи;

– визначити функції, які повинна виконувати система;

– визначити вимоги до видів забезпечення;

– розробити схему функціональної структури;

– обрати та обґрунтувати серверне обладнання;

– розробити програмне забезпечення системи;

– описати розроблену програму;

– визначити техніко-економічні показники;

– провести тестування розробленої програми;

– визначити мету та завдання експерименту;

– визначити методику експерименту;

– визначити вимоги до експерименту;

– провести експеримент;

– визначити результати есперименту.

2 ТЕОРЕТИЧНИЙ РОЗДІЛ

2.1 Чат-боти

За останні роки чат-боти стали важливою складовою цифрової трансформації в багатьох сферах життя, таких як бізнес, медицина, освіта, банківські послуги і роздрібна торгівля. Широке впровадження чат-ботів зумовлене потребою в автоматизації обслуговування клієнтів, підвищенні ефективності бізнес-процесів і зростаючими вимогами користувачів до швидкої і зручної взаємодії з сервісами.

Чат-боти почали активно розвиватися з появою сучасних месенджерів, таких як Telegram, WhatsApp, Facebook Messenger та інших платформ для обміну повідомленнями, які стали основними каналами комунікації для багатьох користувачів. Завдяки інтерфейсу, що схожий на живе спілкування, чат-боти дозволяють користувачам отримувати необхідну інформацію та виконувати певні дії, не виходячи з улюбленого месенджера. За останні кілька років технологія досягла рівня, коли чат-боти можуть розуміти й обробляти складні запити, виконувати функції підтримки, запису на прийом, обробляти фінансові транзакції, допомагати у виборі товарів тощо.

Застосування чат-ботів дозволяє бізнесу скорочувати витрати на обслуговування клієнтів, оскільки автоматизовані системи можуть працювати цілодобово без залучення великого штату співробітників. Крім того, чат-боти забезпечують персоналізований підхід до кожного клієнта, оскільки можуть збирати і зберігати інформацію про користувачів, аналізувати їхні вподобання і попередні взаємодії. Це сприяє підвищенню рівня задоволеності клієнтів і лояльності до бренду.

Чат-боти знайшли широке застосування у різних галузях:

1) медицина – чат-боти допомагають пацієнтам записуватись на прийом до лікаря, отримувати медичні рекомендації, нагадування про прийом ліків;

2) банківська справа – банківські чат-боти можуть інформувати клієнтів про баланс рахунку, надавати консультації з фінансових питань, пропонувати нові продукти;

3) освіта – чат-боти підтримують навчання, проводячи тестування, допомагаючи з матеріалами або відповідаючи на питання студентів;

4) роздрібна торгівля – боти обробляють запити клієнтів, допомагають з вибором товарів, відстежують замовлення та надають інформацію про акції.

Зростання попиту на чат-боти стимулювало розвиток технологій, таких як обробка природної мови (NLP) і штучний інтелект (AI). Ці технології дозволяють ботам аналізувати текст користувачів, розуміти контекст і давати відповідь, що найбільше відповідає запиту. Розвиток API для чат-ботів на платформах Telegram, Facebook Messenger, а також спеціальних сервісів, таких як Dialogflow і IBM Watson, спростили розробку ботів і надали додаткові функції для взаємодії з користувачами.

Чат-боти стали невід’ємною частиною сучасної інфраструктури взаємодії з користувачами, що забезпечує швидкість, зручність та ефективність. З урахуванням швидкого розвитку технологій, чат-боти продовжуватимуть вдосконалюватися, стаючи все більш інтерактивними, інтелектуальними та адаптивними до потреб користувачів.

2.2 Порівняння месенджерів

Telegram є однією з найбільш розвинених платформ для створення чат-ботів. Платформа пропонує відкритий API для розробки ботів і підтримує різні методи взаємодії, такі як Webhook та Long polling. Telegram дозволяє ботам обробляти повідомлення, файли, зображення, голосові повідомлення та команди, а також забезпечує інтеграцію з іншими сервісами через механізми обміну даними (API).

Основні переваги Telegram:

1) Mini Apps – можливість інтеграції веб-додатків у чат-боти, що дає можливість використовувати сучасні технології розробки фронтенду, такі як React або Next.js;

2) гнучкі налаштування взаємодії – підтримка inline-кнопок, кастомних клавіатур та меню, що дозволяє значно покращити UX (досвід користувача);

3) швидке виконання – за допомогою Webhook боти отримують повідомлення майже миттєво, що підвищує швидкість відповіді;

4) підтримка великих даних – Telegram дозволяє надсилати та отримувати великі файли (до 2 ГБ), що є перевагою у порівнянні з іншими платформами.

Facebook Messenger також пропонує широкі можливості для розробки чат-ботів. Платформа дозволяє автоматизувати взаємодію з користувачами через Facebook і Instagram, що відкриває можливості для маркетингових кампаній, обробки запитів користувачів, надання підтримки клієнтам. Facebook API надає доступ до таких функцій, як відправка та отримання повідомлень, підтримка інтеграції з іншими сервісами, налаштування кастомних меню та кнопок.

Однак, у порівнянні з Telegram, платформа Facebook Messenger має деякі обмеження:

– обмежена підтримка великих файлів (до 25 МБ);

– більш складний процес налаштування та отримання доступу до API;

– потреба в додаткових кроках для підтвердження бізнес-аккаунтів.

Viber надає можливість створювати публічні акаунти для бізнесів і дозволяє автоматизувати спілкування з клієнтами через чат-боти. Хоча Viber має певні обмеження щодо функціональності, зокрема відсутність підтримки веб-додатків (Mini Apps), платформа добре підходить для невеликих бізнесів, які потребують простих сценаріїв взаємодії.

WhatsApp підтримує розробку ботів через WhatsApp Business API. Ця платформа призначена переважно для великих компаній, які мають потребу в автоматизації підтримки клієнтів та обслуговуванні запитів. Однією з ключових відмінностей WhatsApp є більш жорсткі обмеження щодо доступу до API, які

вимагають спеціального ліцензування. Проте, WhatsApp пропонує високу безпеку та надійність, що робить її привабливою для корпоративного сектору.

Порівняння технічних обмежень сучасних месенджерів показано в таблиці 2.1

Таблиця 2.1 – Порівняння технічних обмежень сучасних месенджерів

Платформа	Підтримка Webhook	Підтримка Long polling	Максимальний розмір файлу	Підтримка Mini Apps	Налаштування інтеграцій
Telegram	Так	Так	До 2 ГБ	Так	Відкрите API
Facebook Messenger	Так	Ні	До 25 МБ	Ні	API доступ через Facebook
Viber	Так	Ні	До 200 МБ	Ні	Через публічий акаунт
WhatsApp	Так	Ні	До 100 МБ	Ні	Потрібен ліцензований доступ до API

2.3 Види веб-додатків

Веб-додатки є важливою частиною сучасних інформаційних систем і можуть бути реалізовані з використанням різних підходів до архітектури та рендерингу. Основні види веб-додатків поділяються на SPA (Single Page Application) і MPA (Multi Page Application), кожен з яких має свої переваги та недоліки, що впливають на зручність користування, продуктивність та оптимізацію для пошукових систем [17].

SPA (односторінковий додаток) – це веб-додаток, який завантажує одну HTML-сторінку і динамічно оновлює контент цієї сторінки при взаємодії

користувача. Замість завантаження нових сторінок з сервера під час переходу між різними секціями додатка, SPA взаємодіє з сервером через AJAX або інші асинхронні методи, що дозволяє оновлювати контент без перезавантаження сторінки.

Переваги SPA:

1) швидкість – після першого завантаження програми всі наступні дії виконуються без необхідності повторного завантаження сторінок, що суттєво скорочує час відгуку;

2) плавний UX – користувач отримує безперервний досвід взаємодії з додатком, оскільки немає постійного перезавантаження сторінок;

3) мінімізація трафіку – SPA зменшує кількість запитів до сервера, передаючи лише необхідні дані у форматі JSON чи іншим чином.

Недоліки SPA:

1) SEO-проблеми – пошукові системи часто мають труднощі з індексацією SPA через те, що вміст генерується динамічно за допомогою JavaScript;

2) продуктивність при першому завантаженні – перший рендер додатка може бути повільнішим, оскільки браузеру потрібно завантажити великий обсяг JavaScript для повного рендерингу сторінки;

SPA зазвичай використовують фреймворки, такі як React, Vue або Angular, які дозволяють керувати станом додатка на клієнтській стороні та забезпечують динамічне оновлення контенту без необхідності перезавантаження сторінки.

MPA (багатосторінковий додаток) – це класична архітектура веб-додатків, де кожен перехід між сторінками спричиняє повне перезавантаження сторінки з сервера. Кожна сторінка має власний HTML-файл, який сервер надсилає користувачеві.

Переваги MPA:

1) SEO – оскільки кожна сторінка MPA має власний HTML-код, пошукові системи легше індексують контент таких додатків;

2) масштабованість – MPA дозволяють легше керувати великими сайтами з численними розділами або секціями, адже кожна сторінка є окремим модулем.

Недоліки MPA:

1) час завантаження – оскільки кожен перехід між сторінками призводить до повного перезавантаження, користувач може помітити затримку в роботі додатка;

2) ускладнення фронтенд-розробки – для складних інтерфейсів і взаємодій може виникнути необхідність у використанні додаткових технологій на кшталт AJAX, що ускладнює розробку.

MPA добре підходить для великих корпоративних сайтів або інформаційних порталів, де важливо забезпечити надійний SEO і необхідно обробляти великий обсяг статичного контенту.

Порівняння SPA та MPA показано в таблиці 2.2

Таблиця 2.2 – Порівняння SPA та MPA

Параметр	SPA	MPA
SEO	Проблеми з індексацією	Гарна підтримка SEO
Користувацький досвід	Плавна взаємодія без перезавантажень	Затримка при завантаженні нових сторінок
Швидкість	Швидко після першого завантаження	Кожен перехід вимагає нових запитів
Підтримка масштабованості	Краще для невеликих додатків, інтерактивних сервісів	Краще для великих проєктів, контентних сайтів

Для оптимізації роботи веб-додатків застосовуються різні підходи до рендерингу: CSR (Client-Side Rendering), SSR (Server-Side Rendering), SSG (Static Site Generation) та ISR (Incremental Static Regeneration). Кожен із цих методів має

свої особливості та впливає на продуктивність, зручність розробки й індексацію пошуковими системами [12].

CSR – рендеринг виконується на стороні клієнта за допомогою JavaScript. Використовується в SPA і забезпечує швидке оновлення контенту після першого завантаження.

SSR – рендеринг виконується на сервері перед відправкою сторінки користувачеві, що забезпечує кращий SEO та швидший початковий рендер.

SSG – статичне генерування сторінок на етапі збірки, що робить сайт швидким і зручним для SEO.

ISR – дозволяє поєднувати SSG із динамічним оновленням контенту після публікації.

2.4 Порівняння клієнтського та серверного методів рендерингу веб-додатків

Клієнтський (CSR) та серверний рендеринг (SSR) є двома основними підходами для обробки та відображення контенту у веб-додатках. Вони мають значні відмінності, що визначають їх ефективність, продуктивність та придатність до використання в різних сценаріях.

а) принципи роботи:

1) клієнтський рендеринг (CSR) передбачає, що браузер користувача отримує мінімальний HTML-код, і весь процес рендерингу контенту виконується на стороні клієнта. Основні обчислювальні ресурси використовуються для обробки JavaScript, який формує інтерфейс додатка безпосередньо у браузері;
2) серверний рендеринг (SSR), у свою чергу, здійснює формування та рендеринг HTML-коду на сервері перед його надсиланням клієнту. У результаті клієнт отримує повністю згенерований HTML-код, готовий для відображення в браузері, що прискорює завантаження вмісту, особливо для першого відвідування сторінки.

b) продуктивність та швидкість завантаження:
    1) CSR зазвичай повільніше обробляє перше завантаження сторінки, оскільки браузер повинен отримати та обробити усі необхідні файли JavaScript перед відображенням контенту. Проте подальші переходи між сторінками часто швидші, оскільки відбуваються на рівні клієнта без необхідності повторного завантаження всієї сторінки;
    2) SSR забезпечує швидке відображення сторінки під час першого завантаження, оскільки клієнт отримує готовий HTML-код. Це підходить для додатків, де швидкість першого завантаження є критичною, як, наприклад, у контентних сайтах чи додатках з високими вимогами до SEO.
c) SEO та доступність контенту:
    1) CSR має обмеження для SEO, оскільки пошукові системи можуть мати труднощі з індексацією контенту, який завантажується динамічно на клієнтській стороні. Хоча сучасні пошукові системи частково підтримують індексацію JavaScript-контенту, результат не завжди стабільний;
    2) SSR сприяє кращій SEO-оптимізації, оскільки пошукові боти отримують готовий HTML-контент, що спрощує процес індексації. Це робить SSR кращим вибором для додатків, орієнтованих на SEO.
d) навантаження на сервер та клієнт:
    1) CSR переносить значну частину обчислювального навантаження на клієнт, що дозволяє серверу виконувати мінімальні операції та зменшує навантаження на серверні ресурси. Однак це може призвести до проблем із продуктивністю на старих або малопотужних пристроях;
    2) SSR збільшує навантаження на сервер, оскільки сервер повинен виконувати рендеринг кожного запиту. Це може стати проблемою

під час високих навантажень, особливо якщо додаток має велику кількість одночасних користувачів.

e) реактивність інтерфейсу:
   1) CSR дозволяє створювати динамічні, інтерактивні інтерфейси, оскільки рендеринг відбувається на клієнті. Це підходить для складних SPA-додатків (Single Page Applications), де взаємодія користувача з додатком є більш динамічною;
   2) SSR іноді потребує додаткових рішень для забезпечення інтерактивності, оскільки перше завантаження є статичним. Після завантаження сторінки JavaScript також виконується на клієнті для забезпечення взаємодії, але це може викликати так званий "ефект миготіння" при переході між сторінками.
f) підходящі випадки використання:
   1) CSR ідеально підходить для інтерфейсів з високою взаємодією та додатків, де швидкість переходу між сторінками має значення. Це включає соціальні мережі, адмінпанелі та інтерактивні інструменти;
   2) SSR найкраще підходить для додатків, де важливий швидкий перший рендеринг сторінки та SEO, як-от інформаційні сайти, новинні портали, блоги або інтернет-магазини.

Таким чином, вибір між CSR і SSR залежить від вимог до продуктивності, SEO, навантаження на сервер і інтерактивності. Кожен підхід має свої переваги та обмеження, які варто враховувати при розробці веб-додатків із специфічними вимогами до користувацького досвіду та пошукової оптимізації.

2.5 Огляд технологій та інструментів

2.5.1 Мови програмування для створення чат-ботів

Для створення чат-ботів можна використовувати різні мови програмування, кожна з яких має свої переваги та особливості.

JavaScript є однією з найпопулярніших мов програмування для створення чат-ботів, особливо завдяки своєму використанню в середовищі Node.js. Ця мова дозволяє розробникам створювати асинхронні, неблокуючі програми, що робить її ідеальною для обробки запитів у реальному часі. За допомогою бібліотек, таких як Telegraf і Botpress, розробники можуть легко створювати боти для Telegram, Facebook Messenger та інших платформ. JavaScript також забезпечує можливість використання однієї мови на клієнтській та серверній стороні.

Python є однією з найпопулярніших мов програмування у світі, і це не випадково. Його простота та зрозумілість роблять Python ідеальним вибором для розробників, які хочуть швидко створити чат-бота. Python має безліч бібліотек для роботи з API чат-ботів, таких як python-telegram-bot для Telegram і Flask для створення веб-серверів.

C# — це мова програмування, яка активно використовується для розробки чат-ботів, особливо у середовищі Microsoft. За допомогою Microsoft Bot Framework, розробники можуть створювати ботів, які можуть взаємодіяти з різними платформами, такими як Skype, Facebook Messenger і Teams. C# надає розробникам потужні можливості для створення складних бізнес-логік та інтеграції з різними службами.

2.5.2 Види фреймворків та причини їхньої популярності

Фреймворки є невід'ємною частиною сучасної веб-розробки, оскільки вони спрощують створення, підтримку та масштабування веб-додатків. Існує кілька типів фреймворків, які класифікуються за різними аспектами:

Фреймворки для фронтенду:

1) React.js – бібліотека для створення компонентів інтерфейсу користувача з можливістю повторного використання. Популярний завдяки простоті, продуктивності та великій спільноті розробників;

2) Vue.js – легкий фреймворк для побудови користувацьких інтерфейсів. Відзначається своєю простотою у використанні і низьким порогом входження;

3) Angular – повнофункціональний фреймворк від Google для створення великих, складних додатків з можливостями управління станом і потужним інструментарієм для тестування;

4) Next.js – фреймворк, побудований на базі React.js, що дозволяє реалізовувати як серверний, так і клієнтський рендеринг. Next.js особливо популярний завдяки можливостям автоматичного налаштування серверного рендерингу (SSR) і статичного генерування сторінок (SSG);

5) Nuxt.js – фреймворк на основі Vue.js, аналогічний Next.js, з підтримкою серверного рендерингу та статичної генерації сторінок.

Фреймворки для бекенду:

1) Express.js – мінімалістичний фреймворк для Node.js, який дозволяє швидко створювати API та веб-додатки. Популярний завдяки своїй простоті, гнучкості та легкості у налаштуванні;

2) Django – веб-фреймворк на Python, орієнтований на швидку розробку та простоту. Пропонує вбудовані засоби для безпеки, управління базами даних і користувацьких інтерфейсів;

3) Ruby on Rails – фреймворк для швидкої розробки на Ruby, популярний у стартапах завдяки тому, що дозволяє швидко запускати MVP (Minimum Viable Product) продукти (продукт, що володіє мінімальними, але достатніми для задоволення перших споживачів функціями).

Причини популярності фреймворків:

1) швидкість розробки – фреймворки дозволяють швидше створювати додатки завдяки використанню готових компонентів і інструментів;

2) повторне використання коду – можливість створювати повторно використовувані компоненти зменшує обсяг коду та покращує підтримку проєкту;

3) велика спільнота та підтримка – популярні фреймворки мають активну спільноту розробників, що забезпечує постійну підтримку, доступ до навчальних матеріалів і прикладів;

4) підтримка масштабованості – використання фреймворків дозволяє легко розширювати додатки, додаючи нові функціональні можливості без порушення існуючого коду;

5) забезпечення безпеки – багато сучасних фреймворків мають вбудовані інструменти для забезпечення безпеки веб-додатків, що знижує ризик вразливостей.

2.5.3 Порівняння реляційних та NoSQL баз даних

При розробці сучасних інформаційних систем, зокрема таких, як чат-боти для медичних установ, вибір бази даних є ключовим фактором для забезпечення надійного зберігання, обробки та доступу до даних. Реляційні та NoSQL бази даних є двома основними типами сховищ, кожен з яких має свої переваги і недоліки залежно від специфіки проєкту. У цьому підпункті розглянемо основні характеристики обох типів баз даних та їх доцільність для певних задач.

a) Структура даних:
   1) Реляційні бази даних (SQL) використовують таблиці з чітко визначеними схемами (структурами), де кожна таблиця складається з рядків і стовпців, а кожен запис у рядку чітко відповідає структурі. Схема визначається наперед і строго дотримується, що забезпечує високий рівень організації та цілісності даних;
   2) NoSQL бази даних забезпечують більш гнучке зберігання даних, де структура може бути гнучкою або зовсім відсутньою. Дані можуть зберігатися у вигляді документів, ключ-значення пар, графів або колонок, що дозволяє адаптуватися до змін структури без необхідності складних міграцій.

b) Масштабованість:
   1) Реляційні бази даних краще підтримують вертикальне масштабування (додавання ресурсів до одного сервера). Вони

мають складні механізми блокувань і транзакцій, що ускладнює горизонтальне масштабування (додавання нових серверів);

2) NoSQL бази даних, на відміну від реляційних, спроєктовані для горизонтального масштабування, що є однією з їх основних переваг. Це дозволяє легко розширювати систему за рахунок додавання нових серверів, що є критично важливим для великих проєктів, таких як глобальні чат-боти.

c) Гнучкість:

1) SQL бази вимагають попереднього чіткого визначення схеми, тому зміни в структурі бази потребують складних процесів міграції. Це може бути обмеженням для динамічних проєктів, де структура даних може змінюватися в процесі розвитку;

2) NoSQL бази надають більшу гнучкість, оскільки дані можуть зберігатися у довільній структурі або навіть без чіткої схеми. Це дозволяє швидко адаптуватися до змін без необхідності переписування значної частини коду.

d) Запити та мова взаємодії:

1) Реляційні бази даних використовують мову SQL (Structured Query Language) для взаємодії з даними, яка дозволяє виконувати складні запити за допомогою JOIN, GROUP BY та інших операцій. Це робить SQL дуже потужним інструментом для аналізу і маніпуляцій з даними;

2) NoSQL бази часто використовують запити на основі ключ-значення або документо-орієнтовані запити, що не завжди підтримують складні операції, такі як JOIN. Однак, це компенсується швидкістю і простотою доступу до даних в реальних сценаріях використання.

e) Цілісність даних:

1) Реляційні бази даних гарантують високу цілісність даних за допомогою підтримки транзакцій, ключових обмежень (FOREIGN KEY), та

реляцій між таблицями. Це забезпечує суворі правила щодо зв'язків між різними типами даних;

2) NoSQL бази менш суворі в цьому аспекті і, як правило, не підтримують транзакції або гарантують їх лише на рівні одного документа чи операції. Це дозволяє підвищити продуктивність за рахунок зниження складності підтримки цілісності даних.

f) Продуктивність:

1) Реляційні бази даних забезпечують високу продуктивність для складних аналітичних запитів завдяки оптимізації запитів і індексації. Однак, вони можуть виявитися менш ефективними при роботі з великими обсягами даних або високим навантаженням;
2) NoSQL бази даних, навпаки, забезпечують високу продуктивність при роботі з великими обсягами даних і великим навантаженням завдяки простішій структурі і можливостям горизонтального масштабування.

g) Застосування:

1) Реляційні бази даних ідеально підходять для проєктів, де критично важлива структура даних і їх взаємозв'язки, наприклад, фінансові системи або системи управління ресурсами;
2) NoSQL бази даних використовуються у випадках, коли необхідна гнучкість, швидке масштабування і обробка великого обсягу неструктурованих даних, як у випадку з чат-ботами, соціальними мережами, інтернет-магазинами та аналітичними платформами.

2.5.4 Аналіз інструментів для дослідження

Інструменти Chrome DevTools і Lighthouse є основними засобами для веб-розробників, які забезпечують можливість детальної діагностики та покращення продуктивності, доступності й зручності веб-додатків. Ці інструменти дозво ляють розробникам отримати глибоке розуміння процесів, які відбуваються під час

завантаження і роботи веб-сторінок, аналізувати різні аспекти роботи додатків і швидко вирішувати виявлені проблеми [13, 15].

Chrome DevTools – це набір інструментів, вбудований у браузер Google Chrome. Він забезпечує доступ до потужних функцій для налагодження, аналізу і профілювання веб-сторінок. Основні можливості Chrome DevTools включають:

1) Elements – у вкладці Elements розробники можуть переглядати і змінювати структуру HTML, CSS, і DOM елементи в режимі реального часу. Це дозволяє швидко тестувати зміни в стилях, налаштуваннях макета та динамічних елементах;

2) Console – Console – інструмент для виконання JavaScript-коду на сторінці, а також для перегляду повідомлень про помилки й відстеження подій. Console корисний для виведення діагностичних повідомлень, аналізу API-запитів і роботи з об’єктами JavaScript;

3) Network – вкладка Network дозволяє стежити за всіма запитами до сервера, включно з отриманням даних, завантаженням ресурсів, та кешуванням. Інструмент також дозволяє імітувати різні швидкості мережі (наприклад, 3G або офлайн-режим), що корисно для оцінки часу завантаження сторінки за умов повільного з’єднання;

4) Performance – інструмент Performance дозволяє записувати та аналізувати продуктивність веб-додатка. Зокрема, розробники можуть відстежувати процеси завантаження сторінки, скрипти JavaScript, розгортання ресурсів і процеси відтворення графіки. Це допомагає оптимізувати швидкість та усувати "вузькі місця" в продуктивності;

5) Memory – за допомогою вкладки Memory можна виявляти проблеми з управлінням пам'яттю, зокрема, знаходити та усувати "витоки пам’яті" (memory leaks), які виникають у складних JavaScript-додатках. Memory дозволяє переглядати розподіл об’єктів у пам’яті і проводити профілювання використання пам’яті з метою підвищення продуктивності;

6) Application – вкладка Application дозволяє керувати кешем, локальним і сеансовим сховищем, кукі та іншими ресурсами. Вона також містить інструменти для роботи з Service Workers, IndexedDB, та Web SQL, що є важливими для розробки прогресивних веб-додатків;

7) Security – Security забезпечує можливість аналізу та налагодження проблем із сертифікатами, підключеннями HTTPS і перевіркою походження контенту. Це корисно для забезпечення безпечної взаємодії користувачів із веб-додатком;

8) Lighthouse – Lighthouse доступний у Chrome DevTools і забезпечує комплексний аудит сторінки за різними метриками. Це дозволяє одразу переглядати результати аудиту і вносити зміни безпосередньо у DevTools.

Lighthouse – це автоматизований інструмент аудиту веб-сторінок, який забезпечує оцінку роботи додатків за такими критеріями:

1) Performance (Продуктивність) – Lighthouse вимірює ключові показники швидкості завантаження сторінки, такі як First Contentful Paint (FCP), Largest Contentful Paint (LCP), Speed Index (SI), та Time to Interactive (TTI). Завдяки цим метрикам розробники можуть виявити та оптимізувати найбільш "важкі" елементи сторінки;

2) Accessibility (Доступність) – Lighthouse перевіряє доступність контенту для користувачів з обмеженими можливостями. Він виявляє проблеми зі структурою заголовків, описами для зображень, контрастом кольорів, що дозволяє поліпшити доступність додатка;

3) Best Practices (Найкращі практики) – оцінка відповідності найкращим практикам включає перевірку роботи додатка з точки зору безпеки (відсутність застарілих API), продуктивності (оптимальне використання медіафайлів), та загальних рекомендацій щодо якості коду;

4) SEO – Lighthouse аналізує основні фактори, що впливають на SEO, такі як наявність мета-тегів, відповідні заголовки, атрибути зображень, що дозволяє покращити видимість сайту в пошукових системах;

5) Progressive Web App (PWA) – Lighthouse перевіряє додаток на відповідність стандартам PWA, таким як доступність в офлайн-режимі, наявність сервісного воркера та маніфесту додатка. Це корисно для перетворення сайту на прогресивний веб-додаток, який може бути збережений і використаний як автономний.

2.6 Вибір технологій для реалізації системи

2.6.1 Javascript

JavaScript – це універсальна мова програмування, яка є основною для створення інтерактивних веб-додатків і чат-ботів. Вона підтримується усіма сучасними веб-браузерами і використовується як на клієнтській, так і на серверній стороні. У цьому підпункті розглянемо ключові особливості JavaScript, його переваги для створення чат-ботів та роль у сучасних веб-технологіях.

JavaScript є об'єктно-орієнтованою мовою, що підтримує функціональне і імперативне програмування. Основні концепції JavaScript включають:

1) змінні та типи даних – JavaScript має динамічну типізацію, що дозволяє використовувати різні типи даних (числа, рядки, масиви, об'єкти) без явного визначення;

2) функції – мова підтримує функціональність вищого порядку, що дозволяє передавати функції як аргументи і повертати їх з інших функцій;

3) об'єкти – JavaScript дозволяє створювати та маніпулювати об'єктами, що є основою для організації коду та повторного використання.

JavaScript ідеально підходить для інтеграції з HTML і CSS, що дозволяє розробникам створювати динамічні веб-сторінки. З його допомогою можна:

1) додавати інтерактивні елементи на веб-сторінки, такі як кнопки, модальні вікна та спливаючі повідомлення;

2) виконувати валідацію форм на стороні клієнта, що покращує взаємодію з користувачем.

JavaScript може виконуватись не лише в браузері, а й на сервері за допомогою таких платформ, як Node.js. Це дозволяє розробникам використовувати одну мову для обох частин додатку (клієнтської та серверної), що спрощує розробку та зменшує час на навчання.

JavaScript має величезну кількість бібліотек і фреймворків, які спрощують процес розробки. Серед них:

1) React – використовується для створення користувацьких інтерфейсів;
2) Express – забезпечує просту і ефективну розробку серверних додатків.

Ці інструменти дозволяють зосередитися на логіці додатка, зменшуючи обсяги рутинної роботи.

JavaScript підтримує асинхронне програмування, що дозволяє виконувати операції без блокування основного потоку виконання. Це особливо важливо для чат-ботів, які потребують обробки численних запитів та взаємодії з API в режимі реального часу. Для реалізації асинхронних операцій використовуються:

– callback-функції;

– promises;

– async/await.

JavaScript забезпечує прості способи для роботи з API через AJAX-запити, Fetch API та WebSocket, що дозволяє чат-ботам отримувати та надсилати дані до зовнішніх сервісів. Це особливо корисно для інтеграції з різними платформами та сервісами.

JavaScript є основною мовою програмування для створення чат-ботів завдяки своїй універсальності, асинхронному виконанню та зручній інтеграції з іншими технологіями. Його можливості дозволяють розробникам створювати інтерактивні, швидкі та функціональні боти, які можуть працювати на різних платформах і задовольняти потреби користувачів.

2.6.2 Клієнтська частина

2.6.2.1 React.js

React.js – це популярна JavaScript-бібліотека для створення користувацьких інтерфейсів, розроблена компанією Facebook у 2011 році. Вона дозволяє розробникам будувати складні, інтерактивні веб-додатки, що складаються з незалежних компонентів, які можуть бути повторно використані. Це сприяє підвищенню продуктивності та зручності підтримки коду [10].

Основні особливості React.js:

1) компонентний підхід – у React все побудовано навколо компонентів — самостійних частин коду, які відповідають за певну частину інтерфейсу. Кожен компонент має свою логіку, структуру (розмітку) та стиль. Це дозволяє ефективно організовувати проект та повторно використовувати компоненти в різних частинах програми;

2) віртуальний DOM – React використовує концепцію віртуального DOM (Document Object Model). Це означає, що при зміні стану компонента React спочатку оновлює віртуальний DOM, а потім проводить порівняння з реальним DOM. Завдяки цьому зменшується кількість змін, які потрібно зробити в реальному DOM, що підвищує продуктивність і швидкість відображення інтерфейсу;

3) однонаправлений потік даних – у React дані передаються від батьківських компонентів до дочірніх. Це дозволяє легше відстежувати зміни в даних і робить структуру програми більш зрозумілою;

4) JSX (JavaScript XML) – JSX — це синтаксичне розширення JavaScript, яке дозволяє писати HTML-подібний код у JavaScript. Це робить код більш читабельним і зрозумілим, полегшуючи створення та управління компонентами.

React компоненти мають життєвий цикл, який можна розділити на три основні стадії:

1) монтування – коли компонент створюється та додається в DOM. Це включає методи, такі як componentDidMount, які викликаються після того, як компонент буде відображено;

2) оновлення – коли стан або властивості компонента змінюються. У цей момент викликаються методи, такі як shouldComponentUpdate та componentDidUpdate;

3) вилучення – коли компонент видаляється з DOM. У цей момент викликається метод componentWillUnmount, який використовується для очищення ресурсів.

State (стан) – це внутрішні дані компонента, які можуть змінюватися в процесі його роботи. Коли стан змінюється, компонент повторно рендериться, що дозволяє відображати актуальну інформацію.

Props (властивості) – це параметри, які передаються від одного компонента до іншого. Вони є незмінними, тобто компонент не може змінити свої props. Це дозволяє створювати динамічні компоненти, які можуть отримувати дані з батьківських компонентів.

React підтримує обробку подій, що дозволяє взаємодіяти з користувачем. Коли користувач взаємодіє з компонентом (наприклад, натискає кнопку), React може реагувати на ці дії, виконуючи відповідні функції. Обробники подій можуть бути прив'язані до елементів у JSX, що робить управління подіями простим і зрозумілим.

React.js є ідеальним вибором для створення односторінкових веб-додатків (SPA), де необхідно забезпечити швидку реакцію інтерфейсу без перезавантаження сторінок. Його активно використовують у різних сферах, включаючи соціальні мережі, електронну комерцію та медичні системи.

Переваги React.js:

1) продуктивність – завдяки віртуальному DOM React забезпечує високу продуктивність, навіть у складних інтерфейсах;

2) гнучкість – розробники можуть використовувати різні підходи для управління станом і маршрутизації, що дозволяє легко інтегрувати нові технології;

3) підтримка SEO – хоча React є бібліотекою для клієнтського рендерингу, за допомогою Next.js або інших інструментів можна реалізувати серверний рендеринг, що покращує SEO.

2.6.2.2 Next.js

Next.js – це популярний фреймворк на базі React, який спрощує створення повнофункціональних веб-додатків з підтримкою різних стратегій рендерингу. Next.js є відмінним вибором для побудови сучасних веб-додатків, які потребують гарної продуктивності та SEO-оптимізації [8].

Основні особливості Next.js:

1) серверний рендеринг (SSR) – Next.js дозволяє виконувати рендеринг компонентів на сервері до того, як сторінка буде відправлена на клієнт. Це підвищує швидкість завантаження сторінки та покращує SEO, оскільки пошукові системи можуть проіндексувати повністю завантажений контент;

2) статична генерація (SSG) – Next.js може генерувати статичні сторінки під час збірки, що забезпечує швидше завантаження на боці клієнта. Це корисно для сайтів, контент яких рідко змінюється, але має велике значення для продуктивності;

3) інкрементальне статичне оновлення (ISR) – Next.js підтримує автоматичне оновлення застарілих статичних сторінок після їх першого запиту. Це дозволяє зберігати переваги статичної генерації і водночас забезпечувати актуальність даних;

4) роутинг на базі файлів – Next.js використовує файлову систему для організації маршрутизації. Кожен файл у директорії pages відповідає за окремий маршрут у додатку. Це спрощує налаштування роутів і усуває необхідність вручну створювати маршрути;

5) API маршрути – Next.js дозволяє легко створювати серверні API всередині того ж проекту за допомогою папки pages/api. Це зручно для малих проектів, де серверна і клієнтська частина знаходяться в одному додатку;

6) підтримка TypeScript – Next.js зручно інтегрується з TypeScript, що додає переваги типізації і робить код більш безпечним і надійним.

Готові до використання функції для оптимізації продуктивності: Next.js має вбудовані інструменти для оптимізації зображень, автоматичне розбиття коду та попереднє завантаження сторінок, що покращує продуктивність веб-додатка.

Переваги Next.js:

1) SEO-оптимізація – завдяки серверному рендерингу Next.js дозволяє пошуковим системам легко індексувати контент, що підвищує видимість сайтів;

2) продуктивність – завдяки інструментам оптимізації та підтримці статичної генерації, Next.js дозволяє створювати дуже швидкі додатки;

3) простота у використанні – чітка організація роутингу і вбудована підтримка API спрощують розробку та скорочують час на створення функціоналу;

4) гнучкість у рендерингу – Next.js дозволяє вибирати найкращу стратегію рендерингу для різних частин додатка, що забезпечує оптимальне поєднання продуктивності і динамічності.

2.6.3 Серверна частина

2.6.3.1 Express.js

Express.js – це популярний фреймворк для Node.js, який дозволяє розробникам створювати веб-додатки та API з максимальною простотою і ефективністю. Завдяки своїй легкості та гнучкості, Express.js став одним з найбільш використовуваних фреймворків у JavaScript-середовищі.

Основні характеристики Express.js:

1) простота використання – Express.js надає простий і зрозумілий інтерфейс для створення веб-додатків. Його архітектура дозволяє швидко налаштувати сервер і реалізувати маршрутизацію, що значно скорочує час розробки;

2) широкі можливості маршрутизації – Express.js підтримує різноманітні методи HTTP (GET, POST, PUT, DELETE тощо) і дозволяє легко створювати

маршрути для різних URL-адрес. Це забезпечує простоту у визначенні обробників запитів і налаштуванні поведінки сервера;

3) модульність – Express.js використовує концепцію middleware, що дозволяє розробникам додавати функціонал у вигляді модулів. Це означає, що можна підключати різні бібліотеки для обробки запитів, логування, аутентифікації, обробки помилок та інших задач, не порушуючи основну структуру програми;

4) сумісність із різними базами даних – Express.js може працювати з різними системами управління базами даних, такими як MongoDB, MySQL, PostgreSQL та інші, що робить його універсальним рішенням для створення бекенд-частини додатка;

5) підтримка RESTful сервісів – Express.js дозволяє легко реалізувати RESTful API, що робить його ідеальним для створення серверної частини веб-додатків, які взаємодіють із клієнтськими додатками, такими як мобільні або веб-додатки;

6) велика спільнота та екосистема – завдяки широкій популярності Express.js, існує велика кількість плагінів, бібліотек та інструментів, які можуть бути використані для розширення його функціональності.

Переваги використання Express.js:

1) швидкість розробки – завдяки простоті та зручності Express.js, розробка веб-додатків відбувається швидше, що є важливим фактором для стартапів та проектів з обмеженими термінами;

2) гнучкість – розробники мають можливість налаштувати сервер під свої потреби, використовуючи різні middleware і бібліотеки;

3) продуктивність – Express.js має малу вагу та високу продуктивність, що дозволяє обробляти велику кількість запитів одночасно.

2.6.3.2 REST API

REST (Representational State Transfer) – це архітектурний стиль, який використовує стандартні методи HTTP для створення та взаємодії з веб-сервісами.

REST API є основою для побудови сучасних веб-додатків, дозволяючи їм взаємодіяти один з одним через інтерфейс програмування. Основною ідеєю REST є представлення ресурсів (даних) у вигляді URL-адрес, що забезпечує простоту використання та розширюваність системи [9].

Основні принципи REST:

a) статусний – всі дані, з якими працює REST API, розглядаються як ресурси. Кожен ресурс має свій унікальний ідентифікатор, зазвичай у вигляді URL (наприклад, /patients для пацієнтів);

b) використання стандартних HTTP-методів – REST API використовує стандартні HTTP-методи для виконання CRUD-операцій:
   1) GET: отримання даних з сервера;
   2) POST: створення нового ресурсу;
   3) PUT: оновлення існуючого ресурсу;
   4) DELETE: видалення ресурсу.

c) статус коду відповіді – REST API використовує стандартні HTTP-коди відповіді для повідомлення про статус виконання запиту. Наприклад:
   1) 200 OK: запит виконано успішно;
   2) 201 Created: ресурс створено;
   3) 404 Not Found: ресурс не знайдено.

d) безстанова архітектура – REST API не зберігає стан сесії між запитами. Кожен запит від клієнта повинен містити всю необхідну інформацію для його обробки. Це спрощує масштабування і підвищує надійність сервісу;

e) формати обміну даними – REST API підтримує різні формати даних для обміну, але найпоширенішими є JSON (JavaScript Object Notation) і XML. JSON є особливо популярним завдяки своїй легкості та простоті у використанні.

Переваги використання REST API:

1) простота – завдяки використанню стандартних методів HTTP та URL-адрес для ресурсів, REST API легко зрозуміти та реалізувати;
2) масштабованість – безстанова архітектура дозволяє легко масштабувати додатки, оскільки сервери не зберігають стан сесії;
3) взаємодія між платформами – REST API забезпечує зручний спосіб взаємодії між різними системами та платформами, що робить його ідеальним для мікросервісної архітектури;
4) гнучкість – REST API підтримує різні формати даних, що дозволяє легко адаптувати API до потреб клієнтів.

2.6.3.3 Node.js

Node.js – це потужна платформа для серверного програмування, яка базується на JavaScript і працює на рушії V8 від Google. Вона забезпечує асинхронне, неблокуюче виконання коду, що робить її ідеальним вибором для розробки чат-ботів, які повинні обробляти численні запити в реальному часі. Нижче наведено детальний огляд основних характеристик Node.js, а також його переваг і особливостей у контексті створення ботів.

Однією з ключових переваг Node.js є його асинхронна архітектура, яка дозволяє виконувати кілька запитів одночасно. Це досягається завдяки подіям та колбек-функціям, які забезпечують обробку запитів без блокування основного потоку виконання. Такий підхід особливо корисний для чат-ботів, які повинні швидко реагувати на повідомлення від користувачів і обробляти численні запити одночасно.

Node.js має велику екосистему пакетів, доступну через npm (Node Package Manager). Ця бібліотека містить тисячі модулів, які можуть значно спростити розробку ботів. Серед популярних бібліотек для створення ботів можна виділити:

1) Telegraf – бібліотека для створення ботів у Telegram, яка забезпечує простий API для взаємодії з Telegram Bot API;

2) Botpress – платформа для створення чат-ботів з відкритим вихідним кодом, яка надає інтерфейс для розробки, управління та впровадження ботів.

Node.js дозволяє розробникам використовувати JavaScript як на клієнтській, так і на серверній стороні, що спрощує розробку та зменшує кількість необхідних технологій для освоєння. Це дозволяє розробникам, які вже знайомі з JavaScript, легко переходити до створення серверних рішень.

Node.js підходить для створення масштабованих веб-додатків завдяки своїй архітектурі, що підтримує обробку великої кількості одночасних з'єднань. Це робить його особливо корисним для чат-ботів, які повинні працювати в умовах високих навантажень.

Node.js забезпечує зручні засоби для роботи з RESTful API, що є важливим аспектом у розробці ботів, які взаємодіють з зовнішніми сервісами. Завдяки бібліотекам, таким як Axios або Fetch, розробники можуть легко виконувати HTTP-запити для отримання та надсилання даних.

Node.js має вбудовану підтримку WebSocket, що дозволяє реалізувати двостороннє спілкування між клієнтом і сервером. Це особливо важливо для чат-ботів, які повинні надавати користувачам миттєві відповіді та реалізовувати функціональність реального часу.

2.6.3.4 MongoDB

MongoDB – це популярна NoSQL база даних, яка використовується для зберігання і управління великими обсягами даних у нереляційних структурах. На відміну від традиційних реляційних баз даних, таких як MySQL або PostgreSQL, MongoDB зберігає дані у вигляді документів формату BSON (бінарний JSON), що забезпечує велику гнучкість у роботі з неструктурованими даними. Ця база даних ідеально підходить для розробки динамічних і масштабованих додатків, таких як чат-боти, веб-додатки та мобільні сервіси.

Основні характеристики MongoDB:

1) документо-орієнтована структура – MongoDB використовує документи для зберігання даних, кожен з яких є аналогом запису в традиційних базах даних, але з гнучкою структурою. Документи зберігаються у колекціях і можуть мати різну структуру, що дає змогу адаптувати дані під потреби системи без необхідності суворого дотримання єдиної схеми;

2) гнучкість структури даних – однією з ключових переваг MongoDB є можливість зберігати документи різної структури в одній колекції. Це дозволяє легко масштабувати і модифікувати систему, додаючи нові поля до документів без необхідності змінювати загальну схему бази даних. Така гнучкість дозволяє швидко реагувати на зміни в вимогах до зберігання інформації;

3) масштабованість – MongoDB забезпечує горизонтальне масштабування, що робить її ідеальним рішенням для систем з великим обсягом даних. За допомогою техніки шардування (sharding), MongoDB може розподіляти дані між кількома серверами, що значно підвищує продуктивність і дозволяє легко обробляти великі обсяги запитів і даних;

4) висока продуктивність – MongoDB оптимізована для високої продуктивності, особливо при роботі з великими даними. Завдяки простим запитам і індексації на основі ключів, ця база даних дозволяє здійснювати швидкий пошук і маніпулювання даними, що є критично важливим для систем, що працюють в режимі реального часу, таких як чат-боти або сервіси бронювання;

5) підтримка реплікації – MongoDB має вбудовані можливості для реплікації даних, що підвищує надійність і доступність системи. Реплікація дозволяє створювати кілька копій бази даних на різних серверах, що забезпечує безперервний доступ до даних навіть у випадку збою одного з серверів;

6) інтеграція з Node.js – MongoDB має відмінну інтеграцію з Node.js через бібліотеку mongodb, яка дозволяє легко здійснювати операції з базою даних. Це робить MongoDB чудовим вибором для розробників, які використовують Node.js у своїх проєктах, оскільки можна зручно реалізовувати функціонал для роботи з даними без додаткових накладних витрат на адаптацію бази.

2.6.4 Інструменти взаємодії та інтеграції

2.6.4.1 Telegram API

Telegram API – це набір інструментів та інтерфейсів для взаємодії сторонніх розробників із платформою Telegram. Використовуючи Telegram API, розробники можуть створювати власні чат-боти, інтеграції з веб- та мобільними додатками, а також автоматизувати різні процеси. Ця технологія надає широкий спектр можливостей для розробників, дозволяючи створювати як прості, так і складні системи для автоматизації завдань, взаємодії з користувачами та управління контентом у чатах [11].

Основні функції Telegram API:

1) отримання та обробка повідомлень – боти можуть отримувати повідомлення від користувачів або інших ботів. Це може бути текст, медіа-файли (зображення, відео), геолокація, контакти або інші типи даних. Наприклад, в медичному боті користувачі можуть надсилати свої дані, щоб записатися на прийом до лікаря;

2) надсилання відповідей – боти можуть автоматично надсилати повідомлення або відповіді користувачам, обробляти команди або надавати додаткову інформацію на запит. Це особливо корисно для нагадувань про майбутні прийоми, надання інформації про графік лікарів та бронювання;

3) кастомізовані клавіатури та кнопки – Telegram API дозволяє створювати інтерактивні елементи, такі як кастомізовані клавіатури або кнопки, які роблять взаємодію з ботом зручнішою. Наприклад, користувачі можуть вибирати лікаря або час прийому за допомогою інтерактивних елементів;

4) обробка подій – Telegram API підтримує отримання подій через вебхуки або Long polling. Це дозволяє ботам реагувати на дії користувачів у реальному часі. Вебхуки надають можливість автоматично отримувати повідомлення про нові події, що дозволяє боту миттєво відповідати на запити.

Переваги використання Telegram API:

1) масштабованість – Telegram API дозволяє створювати ботів, здатних обробляти велику кількість користувачів і запитів. Це особливо корисно для медичних чат-ботів, де може бути необхідність обробки сотень або тисяч запитів одночасно;

2) безпека – Telegram API підтримує шифрування та забезпечує захист даних, що передаються між користувачами і ботом. Це є важливим аспектом у медичних проєктах, де необхідно захищати конфіденційну інформацію про пацієнтів;

3) легкість інтеграції – Bot API має простий та інтуїтивний інтерфейс, що дозволяє швидко створювати та налаштовувати ботів. Інтеграція з такими платформами, як Node.js і Express.js, полегшує процес розробки серверної частини та взаємодії з базою даних;

4) можливість багатофункціональності – Telegram API дозволяє створювати багатофункціональних ботів, які можуть вирішувати широкий спектр завдань: від автоматизації запису пацієнтів до надання інформації про лікарів та розклади.

2.6.4.2 Telegram Mini Apps

Telegram Mini Apps – це функціональність, що дозволяє інтегрувати веб-додатки всередині Telegram через спеціальний інтерфейс. Ця технологія дозволяє створювати динамічні та інтерактивні веб-застосунки, які можуть працювати в межах самого додатку Telegram, надаючи користувачам швидкий доступ до необхідних функцій без необхідності залишати чат.

Telegram Mini Apps будується за принципом взаємодії між клієнтською частиною та сервером, де клієнтський інтерфейс надається через веб-технології (HTML, CSS, JavaScript). Бекенд частина обробляє запити, керує бізнес-логікою і забезпечує взаємодію з базою даних.

Telegram Mini Apps працює в тісній інтеграції з Telegram API. Через Mini Apps можна передавати дані боту, який обробляє запити та забезпечує взаємодію з іншими компонентами системи, наприклад, з базою даних.

Основні переваги Telegram Mini Apps:

1) інтеграція веб-додатків у середовищі Telegram – веб-додатки можуть бути відкриті безпосередньо всередині чату за допомогою кнопок або спеціальних команд. Це дозволяє користувачам взаємодіяти з веб-застосунками, такими як онлайн-опитування, форми для бронювання, каталоги товарів або інші сервіси, не покидаючи інтерфейс Telegram;

2) доступ через Telegram клавіатуру або кнопки – Mini Apps може бути відкритий за допомогою кастомізованих кнопок у чаті або вбудованої клавіатури. Наприклад, у чат-боті медичної клініки можна додати кнопку для бронювання прийому до лікаря, яка відкриє форму в Mini Apps для введення необхідної інформації;

3) сучасний UX/UI – оскільки Telegram Mini Apps дозволяє розробляти веб-застосунки, які працюють всередині Telegram, розробники можуть використовувати сучасні фреймворки та технології (React.js, Next.js), щоб створити зручний і швидкий користувацький інтерфейс. Це підвищує якість взаємодії користувачів із системою;

4) передача даних через Mini Apps – Telegram Mini Apps дозволяє передавати дані між чат-ботом і веб-застосунком. Наприклад, під час відкриття Mini Apps бот може передати ідентифікатор користувача або інші параметри (наприклад, інформацію про лікаря чи попередні записи пацієнта). Це забезпечує безшовну інтеграцію між ботом і веб-додатком.

2.7 Обґрунтування і вибір методів експериментальних досліджень

Експериментальні дослідження є важливою частиною виконання наукової роботи, оскільки вони дозволяють на практиці перевірити обґрунтованість обраних рішень, технологій та моделей, а також порівняти їх ефективність у різних умовах.

Для досягнення поставлених цілей було обрано кілька методів експериментальних досліджень, що дозволяють провести кількісну та якісну оцінку розробленої системи:

– метод порівняльного аналізу показників швидкості завантаження (DOMContentLoaded, Load);

– метод порівняльного аналізу оцінок продуктивності (Performance, Accessibility, Best Practices, SEO, PWA);

– метод порівняльного аналізу метрик продуктивності (FCP, TTFB, Speed Index, LCP, CLS).

При проведенні експериментальних досліджень можуть виникнути похибки через різні фактори, які слід враховувати:

– нестабільність мережі – швидкість завантаження контенту залежить від пропускної здатності та стабільності мережі;

– навантаження на систему – продуктивність може залежати від завантаженості сервера або клієнтського пристрою, що впливає на обчислення часу завантаження та відображення контенту.

Щоб зменшити вплив випадкових похибок, кожен експеримент буде проводитись кілька разів, після чого буде використовуватись середнє арифметичне значення показників. Це забезпечить більш точний результат та допоможе мінімізувати випадкові коливання.

2.8 Очікувані результати досліджень

Очікувані результати досліджень полягають у визначенні відмінностей у швидкості завантаження та ефективності рендерингу між серверним (SSR) та клієнтським рендерингом (CSR) для різних умов мережі та платформ (Desktop і Mobile). Основні очікувані результати включають наступне:

а) час завантаження HTML-документа:

1) CSR буде демонструвати менший час завантаження HTML-документа та вмісту сторінки у швидких умовах мережі, таких як

без затримок і Fast 3G, за рахунок завантаження на клієнтській стороні.

b) метрики продуктивності:
   1) FCP (First Contentful Paint) – SSR покаже нижчі значення FCP, особливо у повільних мережевих умовах, що дозволить користувачам швидше побачити початковий контент;
   2) TTFB (Time to First Byte) – TTFB для обох методів буде приблизно однаковим, оскільки залежить від відповіді сервера, а не від способу рендерингу сторінки;
   3) SI (Speed Index) – SSR матиме нижчий SI, що дозволить швидше відображати більшу частину вмісту сторінки;
   4) LCP (Largest Contentful Paint) – CSR продемонструє нижчий LCP, що позитивно вплине на сприйняття швидкості програми;
   5) CLS (Cumulative Layout Shift) – очікується, що SSR забезпечить значно нижчий показник CLS, що підвищить стабільність відображення інтерфейсу.

c) оцінки продуктивності:
   1) Performance – очікується, що SSR матиме вищі оцінки продуктивності порівняно з CSR;
   2) Accessibility та Best Practice – обидва підходи покажуть високі результати. Зокрема, Best Practice для обох методів, ймовірно, матиме однакову оцінку, що вказує на дотримання сучасних стандартів коду. Це свідчить про те, що обидва методи відповідають найкращим практикам безпеки, оптимізації ресурсів і зручності для користувача;
   3) SEO – очікується, що SSR матиме вищі оцінки SEO порівняно з CSR;
   4) PWA – результати будуть ідентичними для обох методів.

2.9 Висновок за розділом

У розділі 2 представлено основну структуру досліджуваного об'єкта, а саме комп'ютерної системи для управління записами до лікаря через Telegram-чат-бот. Виділено клієнтську частину, серверну частину та базу даних.

Наведено огляд технологій, зокрема популярні мови програмування для створення чат-ботів, підходи до розробки веб-додатків, способи рендерингу веб-додатків, види фреймворків.

Також в цьому розділі обгрунтовано вибір технологій для реалізації системи, зокрема основну мову програмування, фреймворки для реалізації клієнтської частини, фреймворки та платформу для побудови серверної частини. Також розглянуто архітектурний стиль для побудови API.

В кінці розглянуто Telegram API, Telegram Mini Apps, обгрунтовано і вибрано методи експериментального дослідження, а також описано очікувані результати досліджень.

3 СИНТЕЗ СИСТЕМИ

3.1 Цілі впровадження системи

Впровадження системи має на меті забезпечити зручний та ефективний сервіс для пацієнтів та лікарів медичної клініки, що дозволяє:

a) пацієнтам:
   1) переглядати інформацію про лікарів клініки;
   2) записуватися на прийом;
   3) переглядати записи;
   4) скасовувати записи.

b) лікарям:
   1) переглядати свій розклад роботи.

3.2 Формулювання технічних вимог до системи

3.2.1 Вимоги до реалізації системи

Система повинна складатись з клієнтської (див.п.2.6.2) та серверної частини (див.п.2.6.3).

Клієнтська частина повинна представляти собою веб-додаток, написаний для двох варіантів рендерингу – клієнтського CSR на React.js (див.п.2.6.2.1) та серверного SSR на Next.js (див.п.2.6.2.2), а також сам чат-бот, написаний на Node.js (див.п.2.6.3.3).

Серверна частина повинна представляти собою базу даних на основі нереляційної MongoDB (див.п.2.6.3.4) для зберігання інформації про лікарів та записи пацієнтів, а також API на Express.js (див.п.2.6.3.1) для обробки запитів і роботи з базою даних.

Також повинні бути використані Telegram Mini Apps (див.п.2.6.4.2) та Telegram API (див.п.2.6.4.1) для механізмів взаємодії та інтеграції.

Реалізація системи повинна відповідати наступним вимогам:

1) кросплатформність – система повинна працювати на наступних операційних системах: Android, iOS, Windows, macOS, Linux. Також система повинна процювати на наступних платформах: мобільний додаток Telegram на Android та iOS, десктопний додаток Telegram для Windows, macOS, Linux, а також через веб-версію Telegram;

2) висока продуктивність та стабільність – система повинна бути оптимізована для швидкої обробки запитів, навіть при великій кількості одночасних користувачів. Час відповіді на запит не повинен перевищувати 1 с. Система повинна забезпечувати обробку мінімум 1000 запитів на секунду при навантаженні до 1000 одночасних користувачів без значного зниження продуктивності. Під час пік-завантаження (до 5000 одночасних користувачів) система повинна залишатися стабільною, з рівнем помилок не більше 0.1%;

3) безперервність роботи – система повинна підтримувати безперебійний доступ до чат-бота, з мінімізацією часу простою. Потрібно мати ДБЖ для забезпечення мінімум 8 годин роботи за відсутності електропостачання.

3.2.2 Вимоги до функцій виконуваних системою

Для забезпечення ефективної роботи системи управління чат-ботом у медичній клініці, вона повинна виконувати наступні функції:

1) надавання можливості перегляду списку лікарів та інформації про конкретного лікаря;

2) запис пацієнтів на прийом – система повинна надавати можливість пацієнтам записуватися на прийом до лікарів через чат-бот. Система повинна надавати можливість пацієнтам обрати спеціалізацію лікаря, конкретного лікаря та доступний час для запису, а також надати свої контактні дані (ім'я, номер телефону);

3) управління розкладом лікарям – система повинна надавати можливість лікарям переглядати свій робочий графік. В разі вводу лікарем спеціальної команди

чат-бот повинен надсилати дати, на які є записи. Після вибору дати, чат-бот повинен надсилати список записів на обрану дату;

4) скасування запису – система повинна надавати пацієнтам можливість скасувати запис на прийом через чат-бот;

5) повідомлення в чат-бот – система повинна надсилати повідомлення лікарям та пацієнтам в чат-бот при записі на прийом та при скасуванні запису.

3.2.3 Вимоги до видів забезпечення

3.2.3.1 Вимоги до інформаційного забезпечення

Для забезпечення коректної та безперебійної роботи системи управління медичними записами клініки «Amel Dental Clinic» необхідно визначити основні вимоги до інформаційного забезпечення, які стосуються організації даних, інформаційної сумісності та структури процесів обробки даних. Нижче наведені ключові вимоги до інформаційного забезпечення:

Склад, структура і способи організації даних у Системі:

1) лікарі – ПІБ, спеціалізація, номер телефону, фото, освіта, інформація, графік роботи, Telegram ID;
2) записи на прийом – ПІБ пацієнта, Telegram ID лікаря, спеціалізація, дата, час, Telegram ID пацієнта, ПІБ пацієнта та його номер телефону.

Інформаційний обмін між компонентами системи відбувається через API, побудованому на основі архітектури REST (див.п.2.6.3.2). Основні принципи інформаційного обміну:

1) запити між клієнтською (див.п.2.6.2) і серверною частиною (див.п.2.6.3) через HTTP-запити;
2) використання CRUD-операцій для взаємодії з базою даних (створення, читання, оновлення, видалення записів);
3) взаємодія з Telegram Mini Apps (див.п.2.6.4.2) через його API на Express.js (див.п.2.6.3.1) для обробки запитів від користувачів (пацієнтів та лікарів);

4) забезпечення шифрування переданих даних (через HTTPS) для захисту конфіденційної інформації;

Система повинна використовувати базу даних MongoDB (див.п.2.6.3.4) для зберігання структурованої інформації про пацієнтів, лікарів і медичні записи.

Процес збору даних починається з введення інформації пацієнтами та лікарями через Telegram Mini Apps або через веб-додаток у браузері. Обробка даних здійснюється серверною частиною системи, яка керує записами і зберігає їх у базі даних. Основні етапи процесу:

1) збір даних – введення пацієнтами своїх даних через форми у веб-додатку або Telegram Mini Apps;
2) обробка даних – сервер виконує валідацію даних і обробку запитів (наприклад, запис на прийом або відміна запису);
3) передача даних – передача інформації між клієнтськими додатками і сервером через REST API;
4) представлення даних – для лікарів у вигляді графіків прийому, для пацієнтів у вигляді інформації про лікарів, їх спеціалізацію, доступні дати і години для прийому, а також заплановані записи.

3.2.3.2 Вимоги до технічного забезпечення

Для роботи системи потрібен сервер.

Для стабільної обробки запитів у реальному часі потрібен процесор із достатньою кількістю ядер та тактовою частотою. Чат-бот, API та веб-додаток можуть одночасно обробляти запити від сотень користувачів.

Наприклад, Node.js ефективно працює на багатоядерних процесорах, оскільки дозволяє виконувати асинхронні операції, але для забезпечення швидкого виконання блокуючих операцій потрібна висока тактова частота;

Для підтримки кількох служб одночасно сервер повинен мати достатньо пам’яті для роботи програм, обробки запитів і роботи бази даних MongoDB.

MongoDB використовує оперативну пам’ять для кешування даних, що значно пришвидшує доступ до них. Об’єм у 8 ГБ дозволяє серверу зберігати активний кеш і виконувати запити без затримок.

SSD-диски забезпечують швидкий доступ до даних і зменшують час очікування запитів, що є критично важливим для високої продуктивності серверів.

Для збереження логів серверу потрібно швидке сховище. 256 ГБ достатньо для журналів роботи сервера та інших файлів.

Сучасні сервери із SSD та ефективними багатоядерними процесорами споживають невелику кількість енергії. Однак блок живлення повинен забезпечувати стабільну роботу серверу та його компонентів навіть у пікових навантаженнях.

Мінімальні технічні вимоги до сервера:

1) процесор – будь-який сучасний чотирьохядерний процесор з тактовою частотою не менше 3 ГГц;
2) оперативна пам'ять – від 8 ГБ DDR4;
3) накопичувач – від 256 ГБ SSD;
4) блок живлення – від 200 Вт.

3.2.3.3 Вимоги до захисту інформації

Для забезпечення захисту інформації в системі управління медичними записами «Amel Dental Clinic», яка використовує Telegram Mini Apps, необхідно дотримуватися наступних вимог:

1) захист від XSS (cross-site scripting) атак та CSRF (cross-site request forgery);
2) HTTPS (SSL та TLS) – вся передача даних між сервером та клієнтом повинна здійснюватися через захищені з'єднання HTTPS, що використовують протоколи SSL (Secure Sockets Layer) та TLS (Transport Layer Security). Це гарантує шифрування даних і запобігає перехопленню чи зміні інформації під час її передачі між користувачами та сервером;

3) CORS (Cross-Origin Resource Sharing) – для забезпечення безпеки та запобігання атакам через запити з інших доменів (наприклад, атакам типу «cross-origin»), необхідно правильно налаштувати політику CORS на сервері. Це дозволить контролювати доступ до ресурсів системи лише з довірених джерел та запобігати небажаним запитам з інших сайтів;

4) шифрування даних – система повинна використовувати технології шифрування для захисту даних під час передачі між ботом і користувачем. Для цього Telegram має використовувати протокол MTProto, що забезпечує високий рівень захисту. Це гарантує, що медичні записи та інша конфіденційна інформація надійно захищені під час обміну даними між користувачами та сервером через Telegram;

5) авторизація – система повинна використовувати авторизацію через Telegram для ідентифікації пацієнтів та лікарів. Це забезпечує підтвердження облікових записів та знижує ризик несанкціонованого доступу до системи;

6) контроль доступу до інформації – важливо, щоб дані були доступні лише тим користувачам, які мають на це право. Пацієнти не повинні бачити графіки роботи лікарів. Кожен користувач повинен мати відповідні права доступу в залежності від його ролі в системі (пацієнт, лікар). Це забезпечується через налаштування рівнів доступу і перевірку прав кожного користувача;

7) відповідність законодавству – система повинна відповідати чинним нормам і вимогам законодавства України, що стосуються захисту персональних даних, таких як Закон України "Про захист персональних даних". Це включає дотримання конфіденційності, захист прав користувачів, правильне зберігання і обробку персональних даних пацієнтів, а також виконання вимог законів у разі обробки медичних записів.

3.2.3.4 Вимоги до ергономіки системи

Ергономічні вимоги до системи є важливим аспектом розробки, оскільки вони впливають на зручність використання системи кінцевими користувачами —

пацієнтами, лікарями та адміністративним персоналом медичної клініки. Основні вимоги до ергономіки системи включають такі аспекти:

a) зручний та інтуїтивний інтерфейс:
   1) простота навігації – інтерфейс системи повинен бути логічно структурованим, з чіткими переходами між основними;
   2) інтуїтивність – використання зрозумілих елементів інтерфейсу (кнопки, форми) з підказками та інструкціями сприятиме зниженню кількості помилок з боку користувачів та поліпшить їх взаємодію із системою.
b) адаптивність інтерфейсу:
   1) мобільна версія – оскільки система інтегрована з Telegram Mini Apps, вона повинна коректно відображатися та функціонувати на мобільних пристроях. Інтерфейс повинен бути адаптованим до різних розмірів екранів та типів пристроївю;
   2) відповідність вимогам доступності – для забезпечення доступу до системи для людей з обмеженими можливостями слід дотримуватися принципів доступності (WCAG). Наприклад, підтримка екранних читачів, великі іконки та високий контраст кольорів, навігація за допомогою кнопки TAB.
c) візуальна привабливість:
   1) кольорова схема – вибір кольорів має бути нейтральним і заспокійливим для користувачів, уникаючи різких контрастів, які можуть викликати втомленість очей;
   2) світла та темна тема – в залежності від налаштувань в додатку Telegram, вбудований в бот веб-додаток повинен відкриватися зі світлою або темною темою.

3.2.4 Розробка схеми функціональної структури

Розробляємо схему функціональної структури системи:

a) додаток Telegram:
    1) опис – мобільний або десктопний додаток Telegram, який забезпечує інтерфейс для взаємодії користувача з чат-ботом та веб-додатком;
    2) входи – команди користувача, натискання кнопок;
    3) виходи – відповіді чат-бота, запуск веб-додатку.
b) Telegram API:
    1) опис – програмовий інтерфейс Telegram, який дозволяє розробникам створювати ботів та інтегрувати їх з додатком Telegram;
    2) входи – запити від чат-бота (наприклад, відправка повідомлень, отримання оновлень);
    3) виходи – відповіді на запити чат-бота (наприклад, оновлення про нові повідомлення, дані про користувачів).
c) чат-бот:
    1) опис – програма, яка працює на основі Telegram API та взаємодіє з користувачем в режимі чату;
    2) входи – команди користувача, оновлення від Telegram API;
    3) виходи – повідомлення користувачу, запити до API для отримання даних з бази даних;
    4) технології – Node.js.
d) веб-додаток:
    1) опис – повноцінний веб-додаток, який може бути запущений прямо в додатку Telegram;
    2) входи – взаємодія користувача з елементами інтерфейсу;
    3) виходи – запити до API для виконання певних дій;
    4) технології – React.js, Next.js.
e) API:
    1) опис – програмовий інтерфейс, який забезпечує доступ до функціональності бекенду системи;

2) входи – запити від чат-бота та веб-додатку;
3) виходи – дані з бази даних, результати виконання операцій;
4) технології – Express.js.

f) база даних:
1) опис – зберігає інформацію про лікарів, записи на прийом. Система використовує MongoDB як основну базу даних;
2) входи – запити від API на читання або запис даних;
3) виходи – дані, які відповідають запиту;
4) колекції – doctors (інформація про лікарів), records (інформація про записи).

g) взаємодія компонентів:
1) користувач взаємодіє з чатом-ботом або веб-додатком;
2) чат-бот і веб-додаток відправляють запити до API;
3) API взаємодіє з базою даних MongoDB для отримання або збереження даних;
4) Telegram API забезпечує зв'язок між додатком Telegram і чат-ботом.

Схема функціональної структури показана на рисунку 3.1

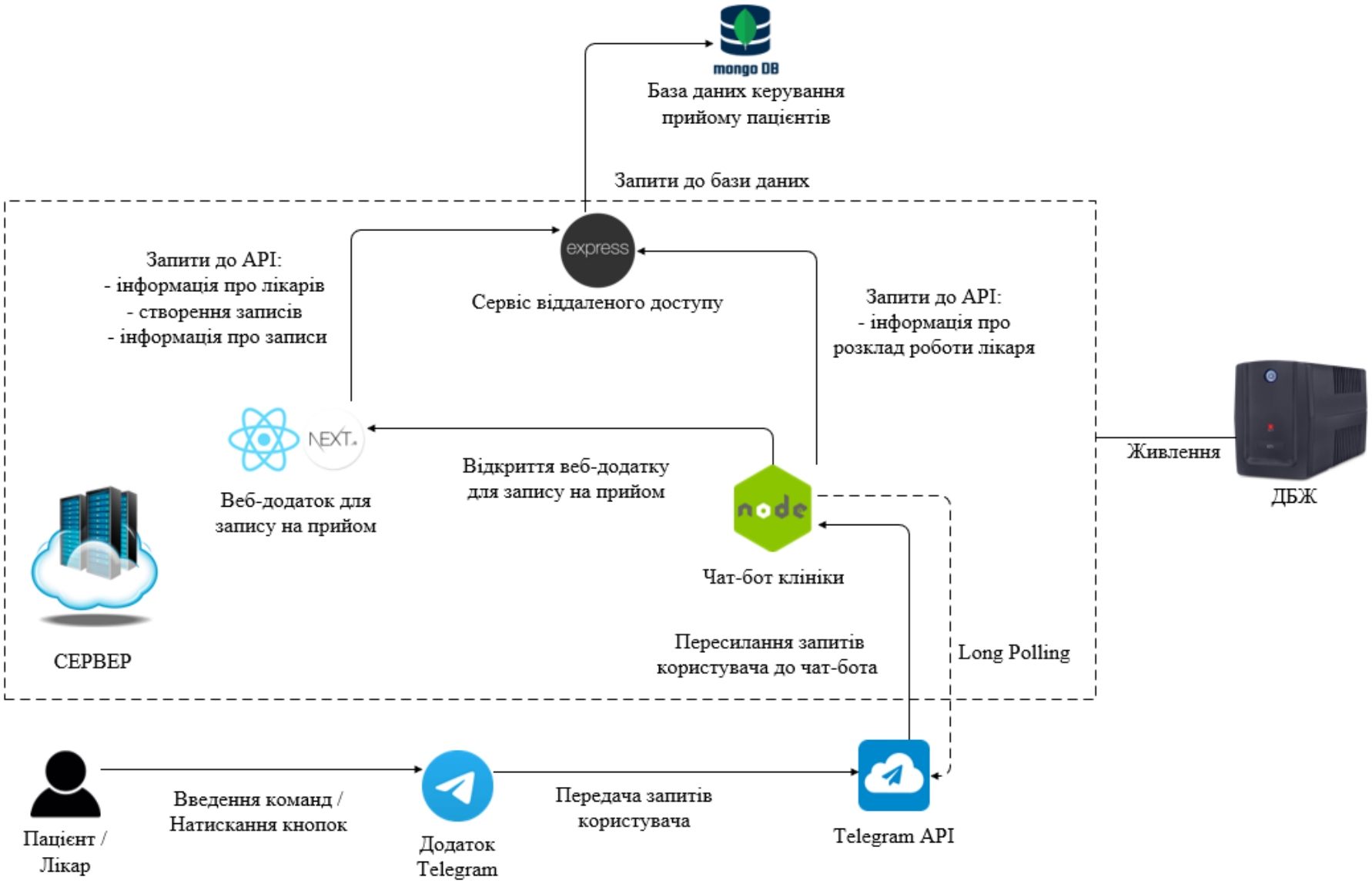

Рисунок 3.1 – Схема функціональної структури системи

3.3 Стислі відомості про комп’ютерну мережу клініки

Комп’ютерна мережа клініки складається з п’яти локальних мереж (далі — підсистеми).

Локальна мережа LAN_1 обслуговує 58 вузлів.

LAN_2 забезпечує взаємодію між 93 вузлами.

LAN_3, яка є віддаленою, включає 80 вузлів.

LAN_4 об’єднує 9 вузлів і включає два сервери: HTTP для обслуговування веб-сторінок і DNS для трансляції доменних імен у IP-адреси.

LAN_5 підтримує 91 вузол і містить сервер TFTP, призначений для передачі файлів.

Загальна кількість вузлів у мережі головного офісу становить 251, а у віддаленій мережі — 80.

У LAN_1 застосовується технологія EtherChannel, яка дозволяє об’єднувати кілька фізичних з’єднань між комутаторами в одне логічне, що підвищує як пропускну здатність, так і надійність.

LAN_5 використовує VLAN — метод логічного розділення фізичної мережі на віртуальні сегменти.

Віддалена мережа LAN_3 підключається до основної через VPN, що забезпечує безпечне з’єднання.

У мережі впроваджено протокол динамічної маршрутизації OSPF для автоматичного визначення та встановлення маршрутів між підмережами. Підключення до Інтернету здійснюється через провайдера за допомогою технології NAT.

Для внутрішніх з’єднань у локальних мережах використовується витий кабель, тоді як об’єднання мереж реалізовано за допомогою оптоволоконного кабелю.

На рисунку 3.2 представлена топологічна схема корпоративної мережі.

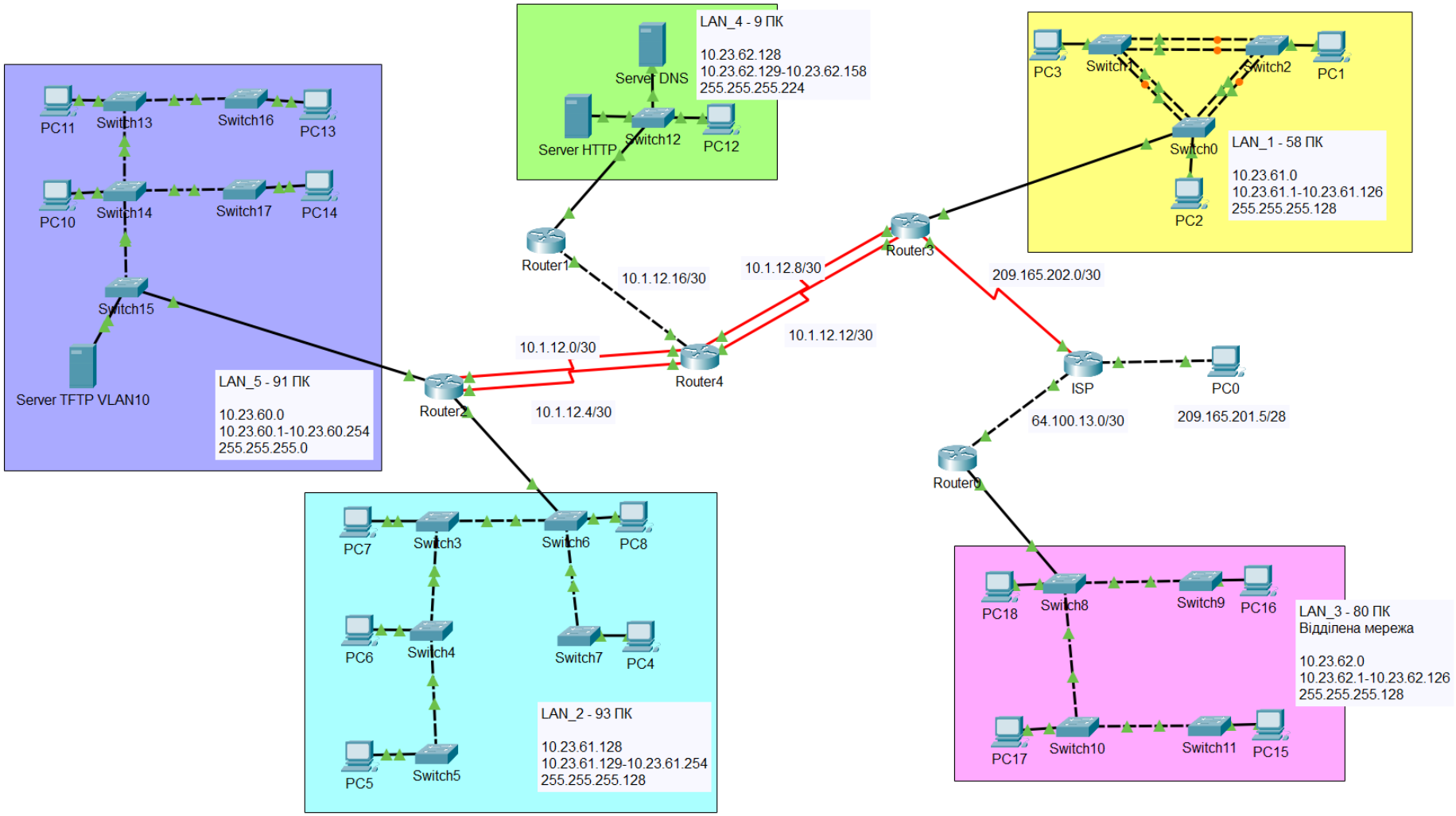

Рисунок 3.2 – Топологічна схема корпоративної мережі стоматологічної клініки «Amel Dental Clinic

3.4 Специфікації апаратних засобів системи

Оберемо сервер Cisco UCS C220 M3 LFF для розгортання Telegram-бота, API та веб-додатка, обґрунтувавши це рішення наступними факторами:

1) достатня потужність процесорів та ядер – 20 ядер (40 потоків) дозволять обробляти запити до Telegram-бота, API та веб-додатка навіть за високих навантажень;
2) стабільність роботи – підтримка гарячої заміни жорстких дисків і віддаленого управління (CIMC) підвищує надійність;
3) енергоефективність – процесори Intel Xeon E5-2650L v2 мають низьке енергоспоживання (70 Вт кожен), що знижує витрати на ДБЖ;
4) масштабованість – є можливість додати оперативну пам'ять (до 16 GB або більше) та збільшити обсяг сховища.

Розрахунок енергоспоживання:

– потужність блоку живлення сервера: 650 Вт;

– реальне споживання: ~300–350 Вт (при середньому навантаженні);

– мережеве обладнання (роутер і комутатор): 50 Вт;

– загальна потужність: 350 Вт + 50 Вт = 400 Вт;

– енергія для 8 годин роботи: 400 Вт × 8 годин = 3200 Вт\год.

В якості ДБЖ оберемо APC Smart-UPS 1000VA (SMT1000RMI2UC) через його високу надійність та відповідність вимогам для забезпечення безперебійного живлення серверів та чутливого обладнання. Це лінійно-інтерактивний ДБЖ з вихідною потужністю 1000 ВА (700 Вт), який забезпечує чисту синусоїду на виході, що важливо для серверів і комп'ютерної техніки.

Даний ДБЖ має достатню кількість розеток (4 IEC), підтримує холодний старт, має вбудовану свинцево-кислотну батарею та дозволяє моніторинг через дисплей.

Час роботи на батареї при навантаженні 400 Вт становить приблизно 1,5-2 години, що є достатнім для короткочасних перебоїв з електроживленням. Для забезпечення 8 годин роботи необхідно додатково підключити модулі батарей. Без

цього часу автономної роботи не вистачить для тривалої роботи при такому навантаженні.

Таким чином, вибір цього ДБЖ є оптимальним для умов, коли потрібна стабільна і безпечна робота серверів та обладнання при нормальних навантаженнях, але для досягнення 8 годин автономної роботи треба буде додати батареї.

Специфікація обладнання для комп'ютерної системи наведена в таблиці 3.1.

Таблиця 3.1 – Специфікація обладнання

Позиція	Найменування і технічна характеристика	Тип, марка, позначення документа, опитувального листа	Одиниці виміру	Кількість	Примітки
1	**2**	**3**	**4**	**5**	**6**
1.	Маршрутизатор серії Cisco 2901: 2 onboard GE, 4 EHWIC slots, 2 DSP slots, 1 ISM slot, 256MB CF default, 512MB DRAM default, IP Base	Cisco 2901/K9	од.	6	За структурною схемою КТЗ: Router 0-4 Детальні характеристики: https://stack-systems.com.ua/marshrutizator-cisco-2901-k9
2.	Комутатор серії 2960: 24 10/100 + 2 1000BT LAN Base Image	WS-C2960-24TT-L	од.	18	За структурною схемою КТЗ: Switch 0-6 Детальні характеристики: https://stack-systems.com.ua/kommutator-cisco-ws-c2960-24tt-l

Кінець таблиці 2.1

3.	Сервер: 2 шт x Intel Xeon E5-2650L v2 (1.70-2.10 GHz), 8 GB DDR3, 2x порта 1 Gb Ethernet, Cisco Integrated Management Controller (CIMC)	Cisco UCS C220 M3 LFF	од.	3	За структурною схемою КТЗ: Сервер HTTP, DNS, TFTP Детальні характеристики: http://surl.li/hnuad
4.	Комп'ютер: AMD Ryzen 5 5600G (3.9 — 4.4 ГГц), 16 ГБ DDR4, 480 ГБ SSD, AMD Radeon Vega 7, Windows 11 Pro	ARTLINE Business B38v08Win	од.	331	За структурною схемою КТЗ: ПК Детальні характеристики: https://comfy.ua/ua/nettop-artline-business-b38-b38v08win.html
5.	ДБЖ: лінійно-інтерактивний, 1000 ВА, 700 Вт, SLA, 4 розетки IEC, чиста синусоїда, холодний старт.	APC Smart-UPS 1000VA (SMT1000RMI2UC)	од.	1	За схемою функціональної структури: ДБЖ Детальні характеристики: http://surl.li/vrhpdf

Все мережеве обладнання було обрано від компанії Cisco, що гарантує сумісність компонентів системи та виключає можливі проблеми інтеграції.

3.5 Висновки до розділу

Розділ "Синтез системи" детально описує процес створення комплексного рішення для автоматизації записів на прийом до медичної клініки за допомогою чат-бота. Були визначені основні цілі системи, вимоги до реалізації системи та функції, які повинна виконувати система.

Далі, в розділі було сформульовано детальні технічні вимоги до системи, включаючи архітектуру, функціональність, вимоги до видів забезпечення, включаючи вимоги до інформаційного та технічного забезпечення, захисту інформації та ергономіки системи.

Була розроблена детальна функціональна структура системи, яка описує взаємодію між компонентами.

Також було надано стислі відомості про комп’ютерну мережу клініки, включно з її топологією.

Крім того, в розділі було обрано та обґрунтовано необхідне обладнання, яке забезпечить надійну роботу системи, а також розроблена структурна схема системи.

4 РОЗРОБКА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ СИСТЕМИ ТЕЛЕГРАМ-БОТА ДЛЯ ЗАПИСУ ДО КЛІНІКИ

4.1 Призначення й область застосування програмного забезпечення

Програмне забезпечення системи телеграм-бота для запису до клініки призначене для автоматизації процесу запису пацієнтів на прийом до лікаря.

Система телеграм-бота знаходить своє застосування в сфері охорони здоров'я, зокрема в медичних клініках, де відбувається регулярний прийом пацієнтів. Вона забезпечує ефективний метод організації запису пацієнтів у медичні установи, що підвищує рівень обслуговування клієнтів і знижує навантаження на адміністративний персонал.

4.2 Постановка завдання на розробку програми

Основним завданням при розробці програмного забезпечення телеграм-бота для запису до клініки є створення автоматизованої системи, яка дозволить пацієнтам записуватися на прийом до лікаря без необхідності особистого контакту з адміністрацією клініки. Бот повинен забезпечувати користувачів можливістю запису на прийом до лікаря, обравши спеціалізацію, лікаря, дату та час прийому. Також пацієнти повинні мати можливість скасовувати записи та переглядати інформацію про лікарів, їхню освіту, фотографію, графік роботи. Бот повинен надавати лікарям та пацієнтам зворотний зв’язок щодо нових записів або їх скасування та можливість лікарям переглядати їхній графік роботи. Для реалізації цієї функціональності використовуються математичні методи обробки даних і побудови структур взаємодії з базою даних, що дозволяє організовувати та зберігати інформацію про записи пацієнтів і лікарів.

4.3 Обґрунтування технічних характеристик програм

Для розробки чат-бота системи запису на прийом до клініки було обрано сучасні технології, що забезпечують продуктивність, надійність та

масштабованість. Чат-бот має забезпечувати стабільну роботу з обробки запитів пацієнтів і лікарів, зручний інтерфейс для взаємодії з системою запису, а також високу продуктивність при обробці даних.

Серверна частина повинна бути реалізована на платформі Node.js з використанням фреймворку Express для ефективного оброблення HTTP-запитів. Node.js, забезпечуючи високу продуктивність завдяки своїй подієвій моделі, що дозволяє виконувати JavaScript на сервері та обробляти великий обсяг одночасних запитів. Express.js спрощує побудову API, що полегшує обробку запитів від користувачів і Telegram API, необхідного для інтеграції з месенджером.

Для управління даними обрано MongoDB як базу даних, що надає гнучкі можливості для зберігання й обробки структурованих та неструктурованих даних. MongoDB, яка добре інтегрується з Node.js, зберігає записи пацієнтів та лікарів у вигляді документів, що спрощує масштабування й дозволяє швидко обробляти запити до бази даних.

Клієнтська частина чат-бота повинна бути розроблена за допомогою бібліотеки React для забезпечення зручного та динамічного інтерфейсу. React дозволяє реалізувати реактивний підхід, що полегшує створення адаптивних веб-інтерфейсів, які відповідають вимогам сучасних користувачів. Це забезпечує зручну взаємодію користувачів із системою через Telegram Mini Apps.

Next.js повинен використовуватися для реалізації серверного рендерингу (SSR), що поліпшує швидкість завантаження сторінок та оптимізує SEO для веб-доступу. SSR забезпечує передачу готової сторінки з сервера, що сприяє швидкому відображенню інтерфейсу і знижує навантаження на клієнтський пристрій.

Для взаємодії клієнтської частини з серверною API повинна використовуватися бібліотека Axios, що спрощує процеси HTTP-запитів до серверної частини, дозволяючи легко обробляти дані й забезпечувати передачу необхідної інформації між клієнтською та серверною частинами.

Для розробки чат-бота у Telegram використано Telegram Bot API та бібліотеку node-telegram-bot-api, що дозволяє ефективно взаємодіяти з Telegram

сервером, відправляти й отримувати повідомлення від користувачів. Завдяки цьому API чат-бот може автоматично відповідати на запити пацієнтів і лікарів, обробляти інформацію про записи та надавати доступ до записів на прийом.

Для розгортання компонентів програми обрано надійний сервер достатньої потужності.

4.4 Розробка програми

4.4.1 Розробка клієнтської частини

Розробку клієнтської частини проєкту на React.js зазвичай починають зі створення початкової структури. У React.js це можна зробити, виконавши в терміналі команду “npx create-react-app .”. Команда спрацює успішно за умови, що встановлено Node.js, а назва директорії, в якій створюється проєкт, написана латиницею в нижньому регістрі.

Після успішного створення «скелету» проєкту в папці src додаємо додаткові директорії для більшої зручності структури:

1) api – для зберігання файлів із запитами до api;
2) components – для зберігання файлів компонентів;
3) context – для зберігання контекстів (глобальних змінних, доступних у будь-якій частині проєкту);
4) fonts – для зберігання шрифтів;
5) hooks – для зберігання користувацьких хуків – функції для повторного використання логіки в різних компонентах. Вони використовують вбудовані хуки (наприклад, useState, useEffect) і починається з префікса use;
6) images – для зберігання картинок.

В папці public знаходиться index.html файл, який є основним HTML-документом. У цьому файлі важливо зауважити, що весь контент React-додатку рендериться в один елемент з атрибутом id="root". В файлі додаємо мета теги (теги в HTML, які містяться в секції <head> і надають метадані про веб-сторінку: опис, ключові слова, авторство, інструкції для пошукових систем і соціальних

мереж. Вони не відображаються на сторінці, але допомагають SEO та покращують взаємодію з платформами), іконку для відображення на вкладках браузера, яку попередньо додано в папку public.

В папці api створюємо файл api.js, де створюємо екземпляр axios (JavaScript-бібліотека для виконання HTTP-запитів із браузера або Node.js) з базовою URL адресою api. Створюємо файли registration-api.js, doctors-api.js, records-api.js, де створюємо об’єкти для роботи із запитами.

В папці components створюємо папку common, де створюємо папку Toast, де створюємо однойменні .jsx та .scss файли для відображення повідомлень про помилки. Для того, щоб підказка була фіксована відносно <body> використовуємо функцію createPortal з react-dom, яка дозволяє рендерити дочірні елементи в інше місце в DOM-структурі, відмінне від батьківського компонента.

В папці components створюємо папку Doctors, де створюємо однойменні .jsx та .scss файли для відображення сторінки зі списком лікарів клініки. В тій же папці створюємо папку DoctorsDetails, де створюємо однойменні .jsx та .scss файли для відображення сторінки з інформацією про конкретного лікаря. Для отримання параметра doctorId використовуємо хук useParams з бібіліотеки react-router-dom.

В папці components створюємо папку Error, де створюємо однойменні .jsx та .scss файли для відображення сторінки в разі, якщо користувач перейшов на неіснуючу сторінку.

В папці components створюємо папку Error, де створюємо однойменні .jsx та .scss файли для відображення сторінки в разі, якщо користувач перейшов на неіснуючу сторінку.

В папці components створюємо папки Footer та Header, де створюємо однойменні .jsx та .scss файли для відображення нижньої частини сторінки, так званого “підвалу”, (де показано режим роботи, контакти клініки та мапа її місцезнаходження), та верхньої частини для відображення навігаційного меню і логотипу клініки.

В папці components створюємо папки Modal, де створюємо однойменні .jsx та .scss файли для відображення модального вікна для підтвердження видалення запису до лікаря. Тут також використовується функція createPortal з react-dom, як і в компоненті Toast.jsx

В папці components створюємо папки Preloader, де створюємо однойменні .jsx та .scss файли для відображення індикатора завантаження під час очікування завантаження сторінок. Тут також використовується функція createPortal з react-dom, як і в компоненті Toast.jsx і Modal.jsx

В папці components створюємо папку Records, де створюємо однойменні .jsx та .scss файли для відображення сторінки із записами пацієнта клініки. В тій же папці створюємо папки RecordsList та RecordsItem, де створюємо однойменні .jsx та .scss файли для відображення списку записів та конкретного запису.

В папці components створюємо папку Registration, де створюємо однойменні .jsx та .scss файли для відображення сторінки для запису до лікаря. В тій же папці створюємо папки PatientInfo, SpecializationList, DoctorList, DateList, TimeSlotList, де створюємо однойменні .jsx та .scss файли для відображення форми для вводу ПІБ та номеру телефона, списку спеціалізацій, лікарів за обраною спеціалізацією, доступні дати та години для запису до обраного лікаря.

В папці context створюємо однойменний .js файл, в якому реалізуємо контекст для зберігання ідентифікатора (Telergam ID) користувача.

В папці hools створюємо файл useTelegram.js , в якому реалізуємо кастомний хук для роботи з Telegram Mini Apps API. Цей хук надає зручний інтерфейс для взаємодії з API, що дозволяє з легкістю управляти основною кнопкою, відправляти дані, перевіряти активну тему і таке подібне.

У папці src створюємо файл App.jsx, в якому реалізуємо основний компонент застосунку. У повернутому JSX-коді використовуємо HashRouter для визначення маршрутизації, відображаємо Header, основний контент з маршрутизацією, а також Footer. Для відображення Preloader, поки завантажуються ліниво імпортовані

компоненти, обертаємо маршрути в компонент Suspense. Також в компоненті блокуємо прокрутку сторінки при відкритті меню.

Маршрути для відображення:

1) / та /doctors – компонент Doctors;
2) / doctors/:doctorId – компонент DoctorsDetails;
3) /registration – компонент Registration;
4) /records – компонент Records;
5) неіснуюча сторінка – компонент Error.

У папці src створюємо файл index.jsx, в якому реалізуємо основну точку входу для React-додатку. У цьому компоненті імплементуємо контекст для зберігання userId, що дозволяє передавати дані про користувача через компоненти, не передаючи їх вручну через пропси. Використовуємо хук useEffect, щоб отримати параметри Telegram з sessionStorage, якщо вони є, та, якщо там є user_id, зберігаємо його в localStorage і оновлюємо стан userId. Якщо параметри Telegram не знайдено, ми намагаємося отримати userId з localStorage. Далі обгортаємо компонент App в UserIdContext.Provider, передаючи значення userId, що дозволяє всім дочірнім компонентам отримати доступ до цього значення. У кінці, компонент рендеримо в DOM у з ідентифікатором root, використовуючи функцію createRoot з ReactDOM.

В Next.js основна структура та ж сама, але є деякі відмінності: відсутність папки src, замість index.jsx та App.jsx файл _app.js, маршрути створені за допомогою папок в папці pages. Створено проект за допомогою команди аналогічної до React.js, а саме “npx create-next-app .”.

В обох проектах в файлах .env знаходиться строка з токеном чат-боту для надсилання повідомлення лікарям. Це потрібно для забезпечення безпеки, оскільки токен — це унікальний ідентифікатор, який дозволяє авторизувати запити до Telegram API. Зберігання токену в .env файлі запобігає його випадковому поширенню у відкритих репозиторіях і дозволяє легко змінювати його при необхідності, не змінюючи код програми.

Дерево компонентів клієнтської частини веб-додатку, написаної на React.js та Next.js показано на рисунку 4.1

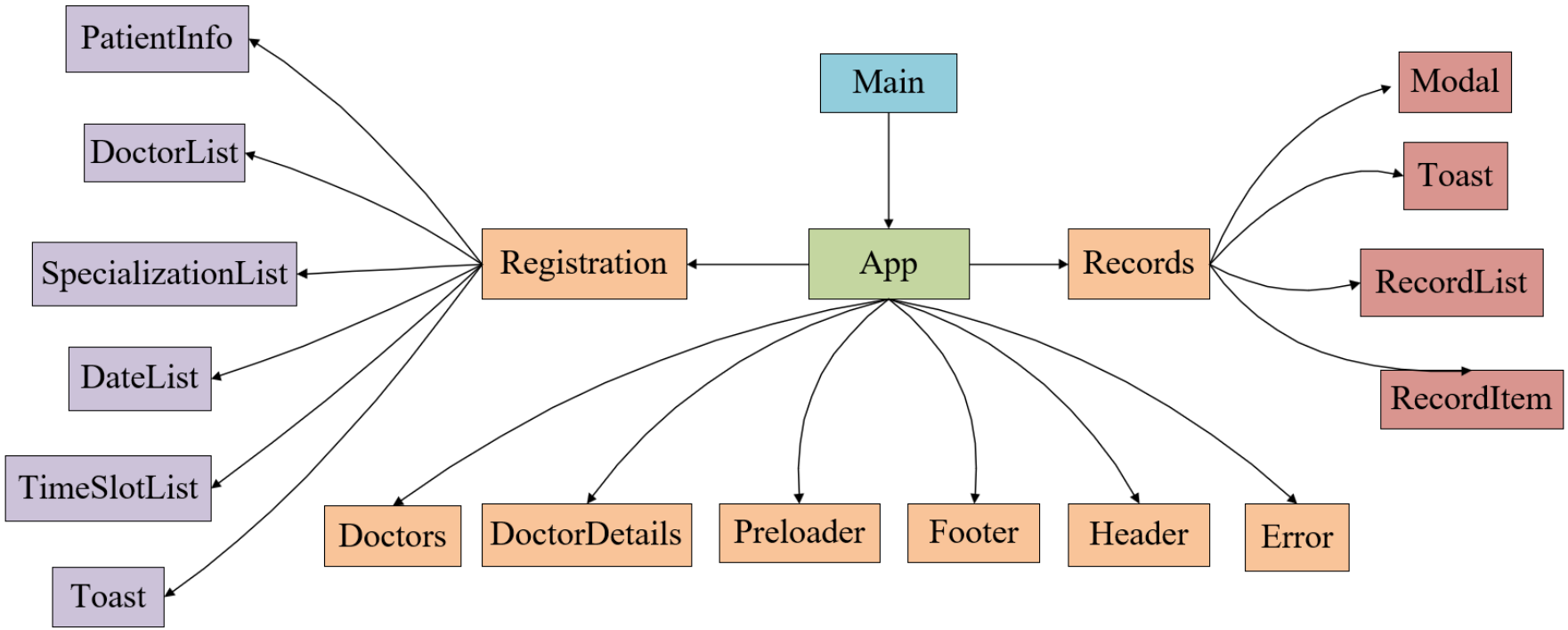

Рисунок 4.1 – Дерево компонентів клієнтської частини веб-додатку

Після завершення розробки клієнтської частини проект компілюється (збирається в готовий варіант) за допомогою команди npm run build. Ця команда запускає скрипт, вказаний у файлі package.json, який за замовчуванням в React виконує react-scripts build, а в Next – next build. Готова версія програми збирається у папку build. Для її успішного запуску на сервері достатньо перемістити файли з цієї папки в кореневу директорію, до якої звертається сервер.

4.4.2 Розробка серверної частини

4.4.2.1 API

API створено за допомогою фреймворку Express.js і працює за принципом "запит – відповідь": клієнт надсилає запит із параметрами, сервер обробляє дані та завжди повертає результат.

У корені проекту створюємо файл index.js, у якому налаштовуємо основні параметри сервера, зокрема, підключення необхідних модулів і налаштування CORS для обмеження доступу до API тільки для певних доменів. Також у цьому файлі запускаємо сервер, задаємо основні маршрути API, такі як /api/doctor, /api/doctors та /api/records, які відповідають за обробку запитів, пов'язаних з управлінням даними лікарів та записами пацієнтів.

У корені проекту створюємо папку config, а в ній файл database.js, в якому налаштовується підключення до бази даних MongoDB. Для цього імпортуємо MongoClient з пакету mongodb, що дозволяє здійснювати підключення до бази. Рядок підключення зберігається в змінних середовища (файл .env в корні проекту) для безпечного доступу, а також визначаємо назву бази даних, яка використовується для зберігання медичних записів.

У файлі database.js експортуються дві основні функції: connectToDatabase та getDatabase. Перша функція встановлює з'єднання з базою даних, зберігаючи посилання на неї, і при успішному підключенні виводить повідомлення в консоль. Друга функція повертає активне підключення до бази даних для використання в

інших частинах програми, що дає можливість звертатися до бази з різних модулів додатку.

У корені проекту створюємо папку routes з файлами doctorRoutes.js, doctorsRoutes.js, recordRoutes.js, в яких пишемо маршрути для взаємодії з відповідними ресурсами. Опис всіх маршрутів API для кожного з роутерів показано в таблиці 4.1

Таблиця 4.1 – Опис всіх маршрутів API

HTTP метод	Маршрут	Опис
Маршрут doctor		
GET	/api/doctor/:telegram_id	Отримання даних лікаря
GET	/api/doctor/:id/schedule	Отримання графіку роботи лікаря
GET	/api/doctor/:id/available-dates	Отримання доступних дат для запису до лікаря
GET	/api/doctor/:id/records/:date	Отримання записів до лікаря на конкретну дату
Маршрут doctors		
GET	/api/doctors/	Отримання списку всіх лікарів
GET	/api/doctors/:specialization	Отримання лікарів за спеціалізацією
GET	/api/doctors/specializations/all	Отримання списку унікальних спеціалізацій лікарів
Маршрут records		
POST	/api/records/	Додавання нового запису
GET	/api/records/:user_id	Отримуємо записи користувача
DELETE	/api/records/:id	Видалення запису

У корені проекту створюємо папку controllers з файлами doctorController.js, doctorsController.js, recordController.js, в яких пишемо логіку для обробки запитів до відповідних ресурсів.

У корені проекту створюємо папку models з файлом recordModel.js, де оголошуємо схему валідації даних для запису на прийом пацієнта за допомогою бібіліотеки Joi.

4.4.2.2 Чат-бот

Чат-бот створено за допомогою Node.js.

В корні проекту створюємо файл index.js, який є точкою входу в програму.

В корні проекту створюємо папку bot з файлом bot.js, в якому ініціалізуємо Telegram-бот за допомогою бібліотеки node-telegram-bot-api, обробляються вхідні повідомлення та callback-запити. Токен чат-боту зберігаємо в змінних середовища (файл .env в корні проекту) для безпечного доступу.

В папці bot створюємо папку handlers з файлами callbackQueryHandler.js та messageHandler.js, в яких реалізуємо обробники для управління вхідними повідомленнями та callback-запитами від користувачів. Ці файли містять логіку для обробки різних команд, отримання доступних дат для запису, а також формування відповідних повідомлень для пацієнтів та лікарів.

В папці bot створюємо папку config з файлом constants.js, в якому містяться різні константи та повідомлення, що використовуються в чат-боті, такі як привітання, повідомлення про успішні записи та скасування, а також заголовки та тіла повідомлень про записи до лікаря. Створюємо файл env.js, який містить змінні середовища, деякі з яких завантажуються з .env файлу. Вони включають токен бота, URL веб-додатку та URL API для взаємодії з бекендом.

В папці bot створюємо папку utils з файлом api.js, в якому містяться функції для роботи з API. Створюємо файл telegram.js, який містить функцію для отримання URL веб-додатку з параметром user_id. Функція getWebAppUrlWithUserId формує URL для веб-додатку, додаючи до нього user_id як параметр запиту. Це дозволяє легко генерувати посилання на веб-додаток, передаючи ідентифікатор користувача для подальшої обробки в додатку.

4.5 Опис розробленої програми

4.5.1 Загальні відомості

Розроблена програма має назву "Телеграм-бот для запису до клініки 'Amel Dental Clinic'".

Увесь додаток можна описати як клієнт-серверне програмне забезпечення, написане на JavaScript з використанням додаткових зовнішніх бібліотек для обох частин.

4.5.2 Функціональне призначення

Програма "Телеграм-бот для запису до клініки 'Amel Smart Clinic'" призначена для автоматизації процесу запису пацієнтів на прийом до лікарів клініки та для спрощення комунікації між пацієнтами та клінікою.

Класи розв'язуваних завдань та призначення програми:

1) перегляд інформації про лікарів − програма дозволяє пацієнтам переглядати список лікарів та інформацію про конкретного лікаря;

2) запис пацієнтів на прийом − програма дозволяє пацієнтам обирати спеціалізацію, лікаря, доступні дати та часові слоти для запису на прийом. Збереження інформації про запис у базі даних MongoDB, щоб забезпечити доступність інформації як для лікарів, так і для адміністрації клініки;

3) надсилання повідомлень лікарям − через Telegram API програма надсилає повідомлення лікарям про нові записи пацієнтів, інформуючи про час і контактні дані пацієнта. Ця функція дозволяє лікарям оперативно отримувати інформацію про прийоми без необхідності додаткових систем управління;

4) управління записами пацієнтів − пацієнти можуть переглядати свої записи, що зберігаються в базі даних, а також скасовувати їх за потреби. Лікарі також мають доступ до інформації про свої прийоми для зручного планування робочого часу.

Функціональні обмеження на застосування:

1) доступність до записів лише через Телеграм − лише користувачі, які відкрили веб-застосунок через телеграм-бот мають можливість переглядати свої записи та скасовувати їх за потреби. В веб-застосунку, відкритому в браузері користувачі можуть можливість лише переглядати інформацію про лікарів та записуватись на прийом;

2) обмеження взаємодії з лікарями − повідомлення надсилаються лише тим лікарям, які використовували Telegram-бота і мають ідентифікатор Telegram ID. Без цього ідентифікатора надсилання повідомлень не відбувається;

3) вимоги до підключення до інтернету − програма працює в реальному часі та потребує стабільного підключення до інтернету для взаємодії з базою даних та Telegram API;

4) обмеження на використання через Telegram API − програма залежить від політик і обмежень Telegram API, зокрема щодо відправки повідомлень лише користувачам, які попередньо взаємодіяли з ботом, та обмежень щодо частоти запитів до API.

4.5.3 Зв'язок програми з іншими програмами

Програма інтегрується з Telegram API для обробки запитів користувачів у Telegram і з API бази даних MongoDB для зберігання записів та даних лікарів. Цей підхід дозволяє забезпечити надійну і швидку обробку даних, що надходять.

4.5.4 Використовувані технічні засоби

Для роботи програмного забезпечення Telegram-бота для запису до клініки використовуються такі технічні засоби:

1) сервер для розгортання всіх компонентів програми, характеристики якого написано в пункті 3.3;

2) користувацькі пристрої (смартфони, планшети, ПК) − пацієнти й лікарі використовують власні пристрої для доступу до Telegram-бота та інтегрованого

веб-додатку. Програма працює на мобільних платформах (Android, iOS), а також на десктопних операційних системах (Windows, macOS).

4.5.5 Виклик і завантаження

Спосіб виклику програми з відповідного носія даних:

1) веб-додаток (React і Next.js) – додаток розгорнуто на звичайному сервері, де користувачі можуть отримати доступ до інтерфейсу через веб-браузер. Виклик додатку здійснюється шляхом відкриття веб-додатку в телеграм-боті з URL, що перенаправляє на сервер, який віддає відповідні сторінки для рендерингу;

2) API та Telegram-бот – запускаються на тому ж сервері, обробляючи HTTP-запити через виділені порти. Telegram-бот отримує події через Long polling, забезпечуючи інтерактивну комунікацію з користувачами безпосередньо в Telegram.

Вхідні точки в програму:

1) веб-додаток (React і Next.js) – вхідною точкою є головна сторінка веб-додатку, яка ініціює завантаження всіх необхідних компонентів та забезпечує рендеринг залежно від URL-запиту;

2) API – вхідна точка для API розміщена на конкретному маршруті (наприклад, /api/records), що обробляє запити від веб-додатку і Telegram-бота, дозволяючи інтеграцію з базою даних;

3) Telegram-бот – вхідною точкою є серверна функція, яка отримує запити від Telegram через Long polling.

Адреси завантаження, використання оперативної пам'яті, обсяг програми:

1) веб-додаток (React і Next.js) – завантажується безпосередньо з сервера. Рендеринг здійснюється як на сервері (SSR), так і на клієнті (CSR) залежно від налаштувань Next.js, оптимізуючи використання пам'яті на клієнтському пристрої;

2) API та Telegram-бот – завантажуються на сервері, який обробляє всі виклики, використовуючи оперативну пам’ять і процесорні ресурси залежно від навантаження.

4.5.6 Вхідні дані

Характер, організація і попередня підготовка вхідних даних:

1) дані пацієнтів – ім'я пацієнта, номер телефону, спеціалізація лікаря, ім’я лікаря, дата та час прийому. Ці дані вводяться користувачем у веб-додатку та передаються через Telegram-бот;

2) команди телеграм-бота /start для початку роботи, /schedule для відображення дат, на які є записи до лікаря;

3) кнопка для відкриття веб-додатку та кнопки для вибору дати для перегляду графіка лікаря у вибраний день.

Формат, опис і спосіб кодування вхідних даних:

1) формат даних – всі дані передаються у форматі JSON для зручності обробки і сумісності між клієнтською та серверною частинами;

2) кодування даних – текстові дані (ім'я, спеціалізація) зберігаються у форматі UTF-8, а номер телефону зберігається як числове значення;

3) валідація – перед передачею даних здійснюється їх валідація для запобігання некоректним запитам, що включає перевірку введеного номера телефону, а також наявності всіх обов’язкових полів.

4.5.7 Вихідні дані

Характер, організація і попередня підготовка вихідних даних:

1) підтвердження дії для пацієнта – після успішного створення або скасування запису веб-додаток і Telegram-бот надсилають пацієнту підтвердження запису;
2) повідомлення лікарю – повідомлення про новий запис або скасування запису надсилається обраному лікарю через Telegram.

Формат, опис і спосіб кодування вихідних даних:

1) формат даних – вихідні дані надсилаються як повідомлення у форматі JSON (для взаємодії з API) або тексту (у випадку повідомлення);

2) кодування даних – повідомлення для лікаря кодується у форматі UTF-8 і містить інформацію у вигляді тексту з emoji для покращення сприйняття. JSON-відповіді API містять ключі та значення у форматі, що відповідає структурам даних MongoDB.

4.5.8 Опис логічної структури програми

На рисунку 4.2 показано схему алгоритму програми клієнтської частини програми на React.js та Next.js

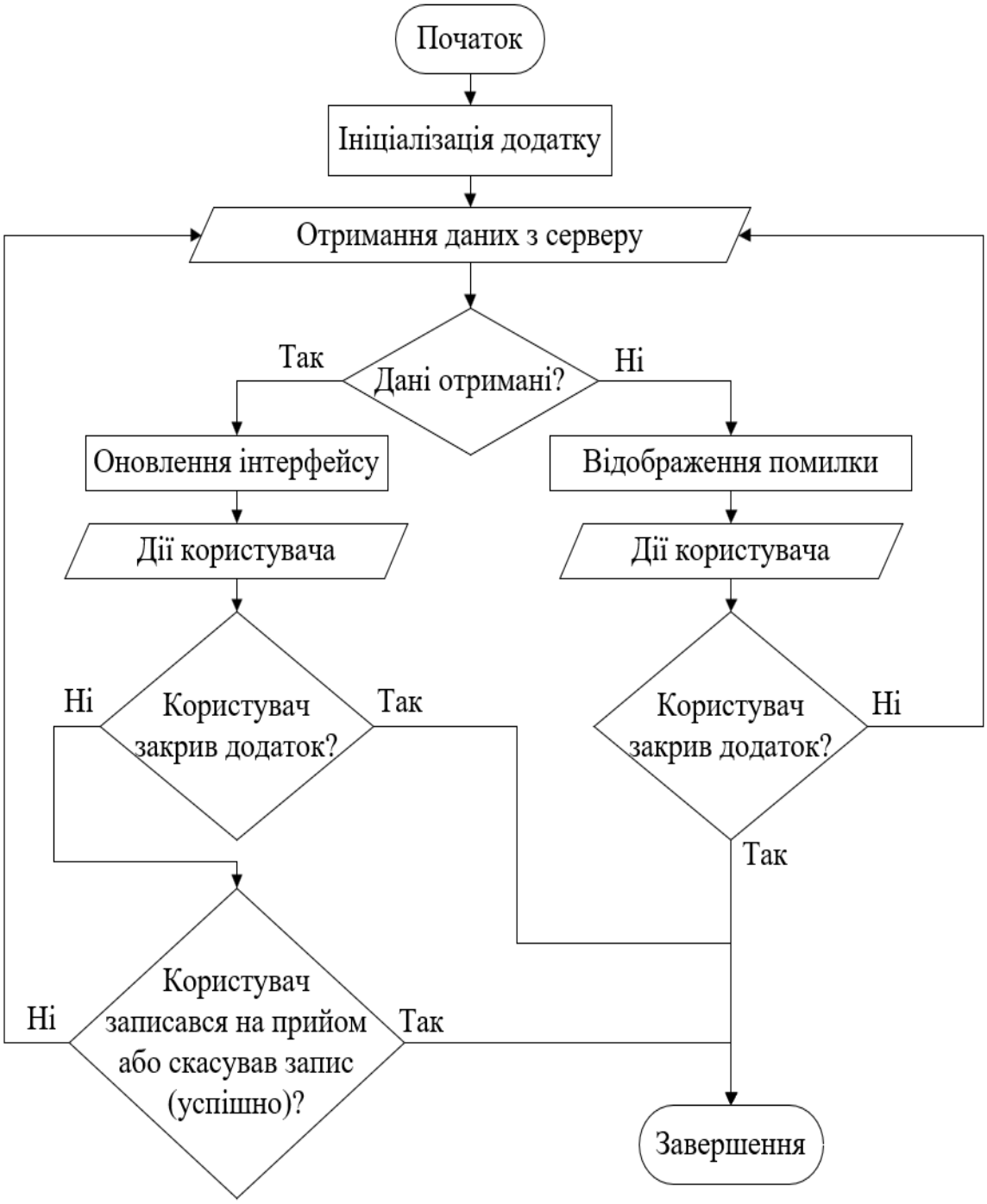

Рисунок 4.2 − Схема алгоритму програми клієнтської частини

На рисунку 4.3 показано схему алгоритму програми API.

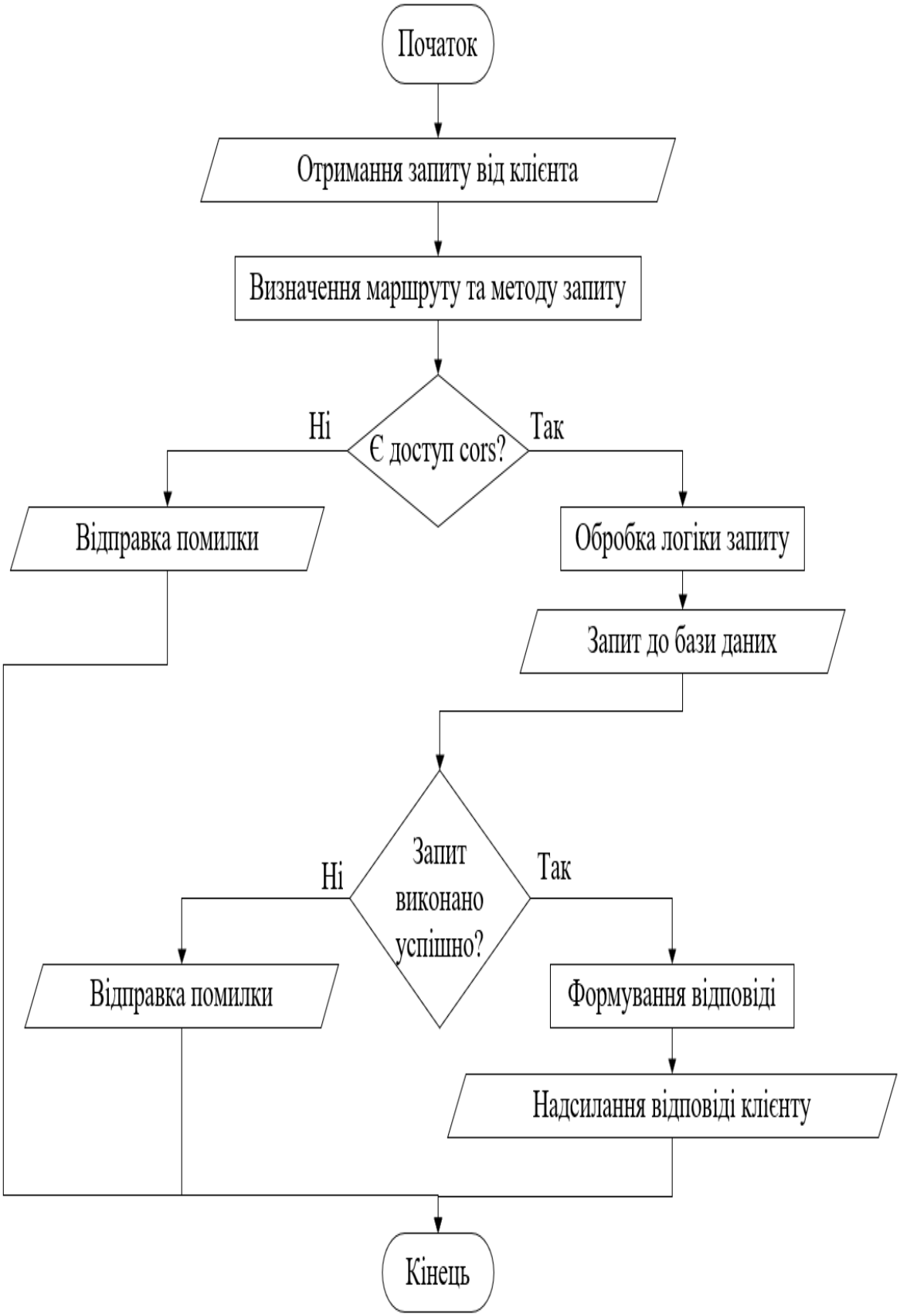

Рисунок 4.3 – Схема алгоритму програми API

На рисунку 4.4 показано схему алгоритму програми чат-боту.

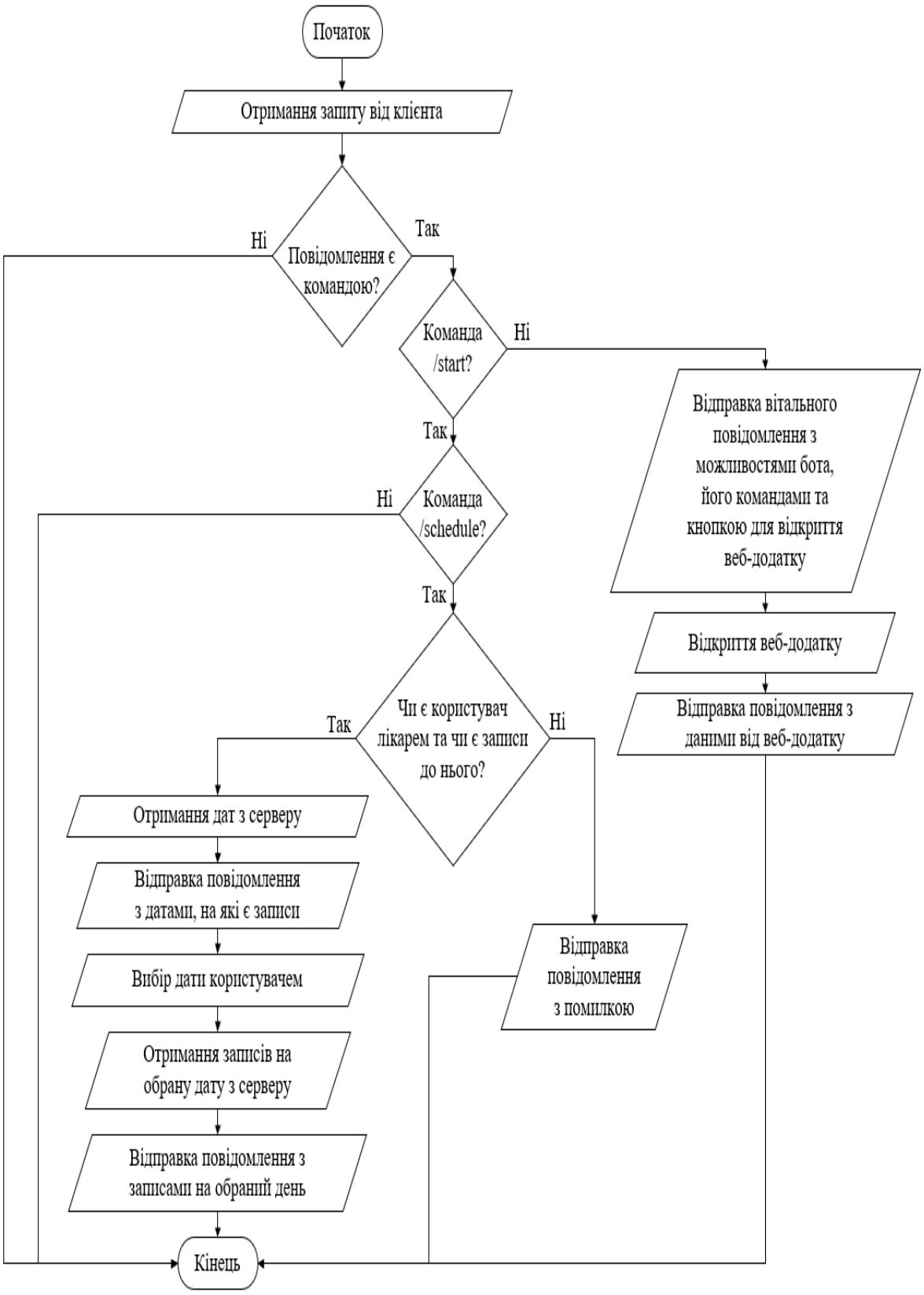

Рисунок 4.4 – Схема алгоритму програми чат-боту

4.5.9 Розробка структурної схеми

Загальна структура програмного забезпечення (рисунок 4.5) виглядає наступним чином: користувач відкриває чат-бот клініки в додатку Telegram на телефоні, планшеті або комп’ютері. Після вводу команди /start чат-бот надсилає початкове повідомлення з можливостями бота. Пацієнт натискає кнопку для відкриття веб-додатку, в якому він може переглянути інформацію про лікарів клініки, записатися на прийом до лікаря, переглянути свої записи та скасувати їх за необхідності. При записі або видаленні запису чат-бот надсилає повідомлення як пацієнтам так і лікарям. Лікарі можуть ввести команду /schedule, де можуть переглянути записи на обрану дату.

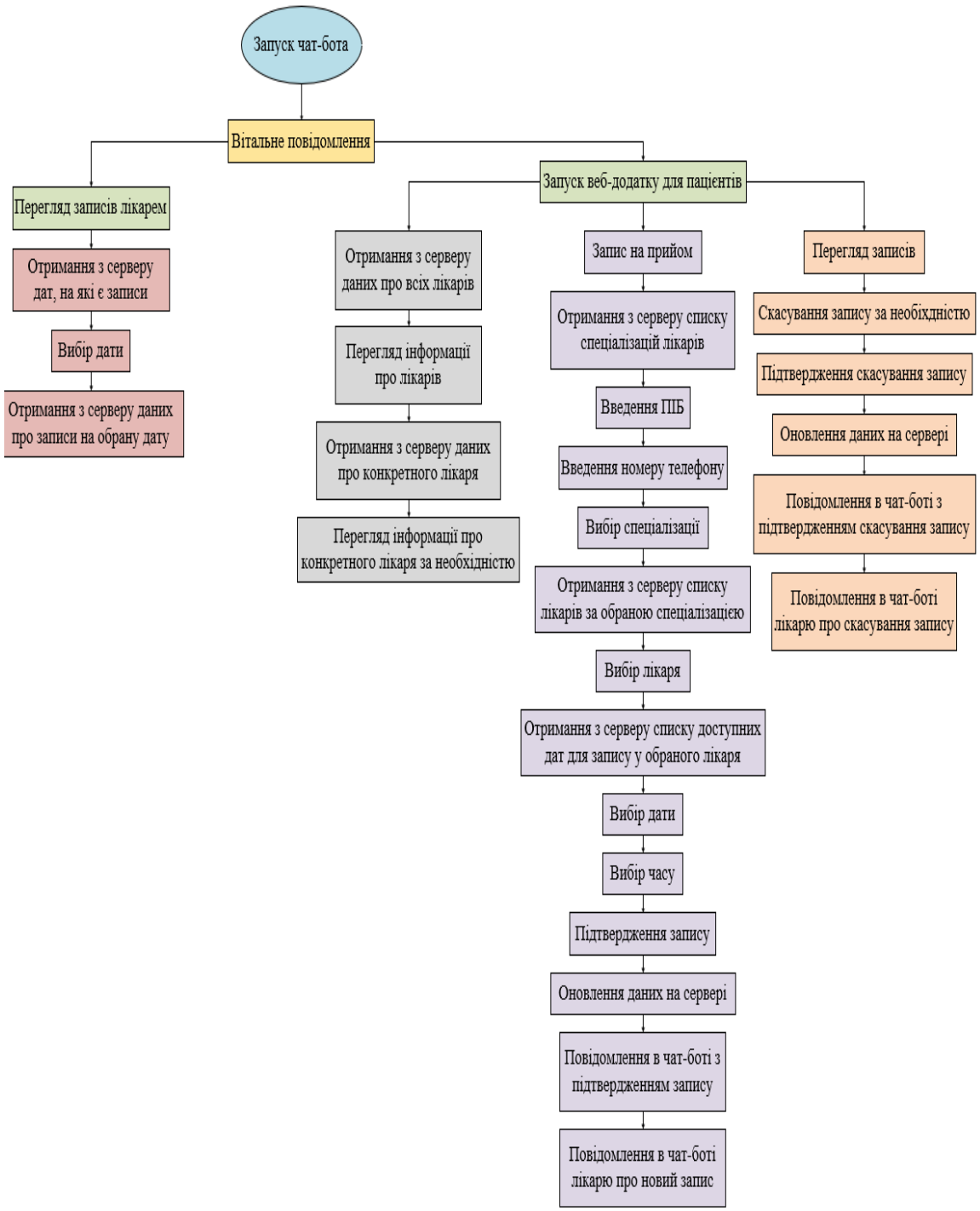

Рисунок 4.5 – Загальна структура програмного забезпечення

Структура бази даних, яка має назву hospital, складається з двох колекцій:

– doctors: для даних лікарів;

– records: для записів пацієнтів.

Структура бази даних показана на рисунку 4.6

Рисунок 4.6 – Структура бази даних

4.6 Очікувані техніко-економічні показники

Розроблений Telegram-бот для запису до медичної клініки забезпечує низку переваг у порівнянні з традиційними методами організації записів пацієнтів та розподілу ресурсів медичного закладу. Обраний варіант технічного рішення, який включає використання Next.js для серверного рендерингу веб-додатка, Telegram Mini Apps для інтеграції з платформою Telegram, а також API, реалізоване на Node.js з використанням MongoDB, дозволяє досягти таких техніко-економічних показників:

1) скорочення часу обслуговування пацієнтів – запис на прийом займає мінімум часу як для пацієнта, так і для адміністративного персоналу клініки. Бот автоматично фіксує дані пацієнта, спеціалізацію, обраного лікаря та зручний час, що знижує необхідність ручного ведення записів і телефонного обслуговування;

2) оптимізація використання часу лікарів – миттєві сповіщення, які лікарі отримують через Telegram, дозволяють швидше реагувати на записи та їх скасування. В результаті підвищується коефіцієнт зайнятості лікарів та ефективність їх роботи;

3) зменшення витрат на адміністрування – впровадження автоматизованої системи запису скорочує витрати, пов’язані з роботою адміністративного персоналу, який зазвичай займається реєстрацією та плануванням візитів. Це також знижує потребу в телефонному зв'язку з пацієнтами, що позитивно впливає на загальну вартість підтримки процесу запису;

4) гнучкість і масштабованість – вибране програмне забезпечення, включаючи Next.js і MongoDB, забезпечує високу продуктивність та масштабованість системи. Це дозволяє легко адаптувати та розширювати функціонал Telegram-бота при розширенні клініки;

5) оперативність – реалізація серверного рендерингу (SSR) з Next.js знижує час завантаження сторінок, що підвищує зручність користувача, особливо в Telegram Mini Apps. Швидка обробка запитів також зменшує час очікування, забезпечуючи позитивний користувацький досвід;

6) економія на розробці та підтримці – обрані технології (Node.js, Next.js, React.js, Express.js, MongoDB) є open-source і широко використовуються, що забезпечує низьку вартість розробки та надійну підтримку. Завдяки цьому економляться кошти на ліцензійні збори та забезпечується простота у підтримці системи;

7) безпека та конфіденційність – інтеграція з Telegram гарантує високий рівень безпеки передачі даних між пацієнтом і клінікою. Використання сертифікатів SSL та захищених каналів зв’язку дозволяє зберегти конфіденційність пацієнтських даних, що є критично важливим у медичній сфері.

4.7 Тестування розробленої програми

Запускаємо Telegram-бота. Бот вітає користувача та пропонує натиснути кнопки “Запустити”. Початкове зображення та текст налаштовано за допомогою головного боту Telegram @BotFather. Це показано на рисунку 4.7. За допомогою @BotFather і взагалі створено бота.

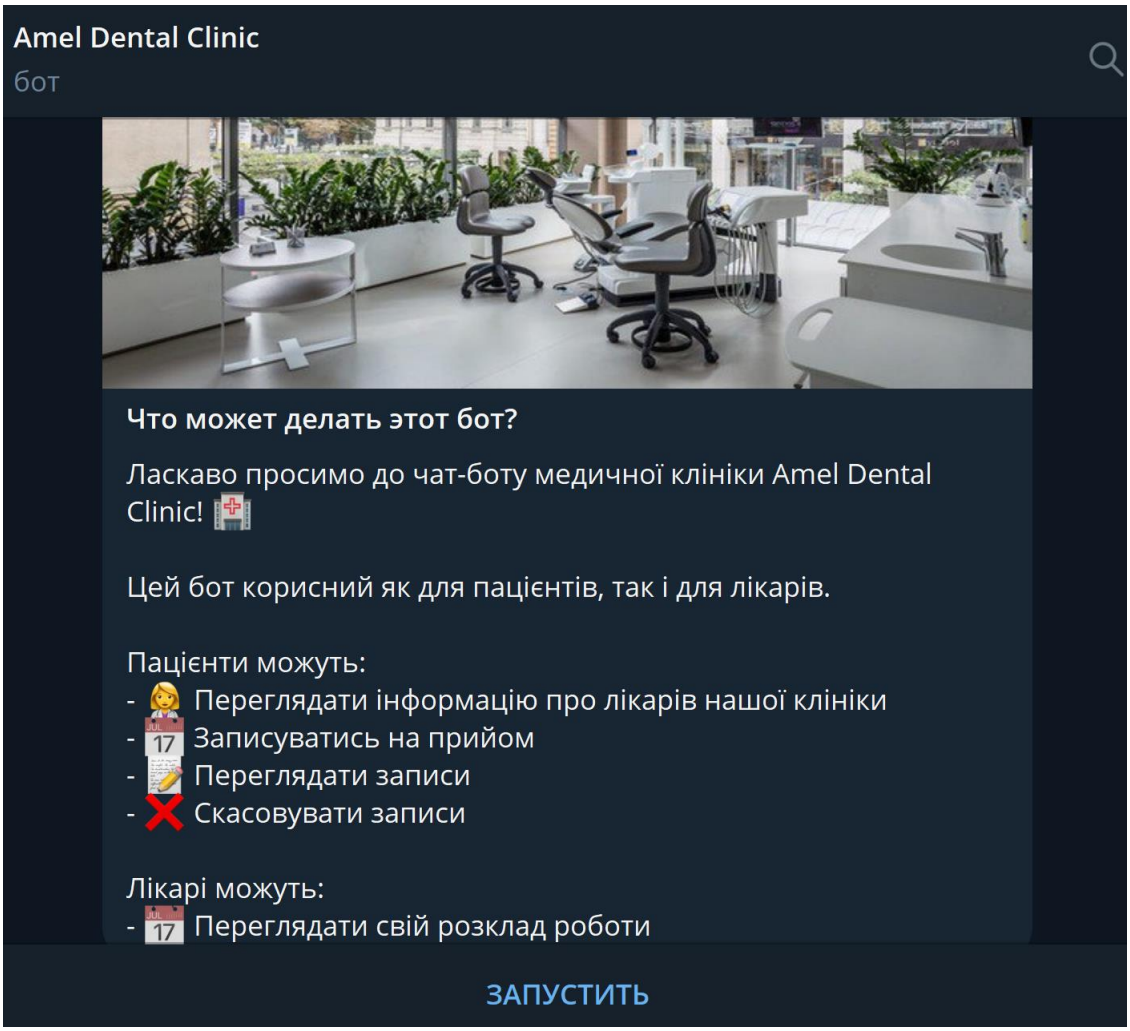

Рисунок 4.7 – Запуск Telegram-бота

Telegram-бот має зображення, назву та опис, що теж налаштовано за допомогою головного боту Telegram @BotFather. Це показано на рисунку 4.8

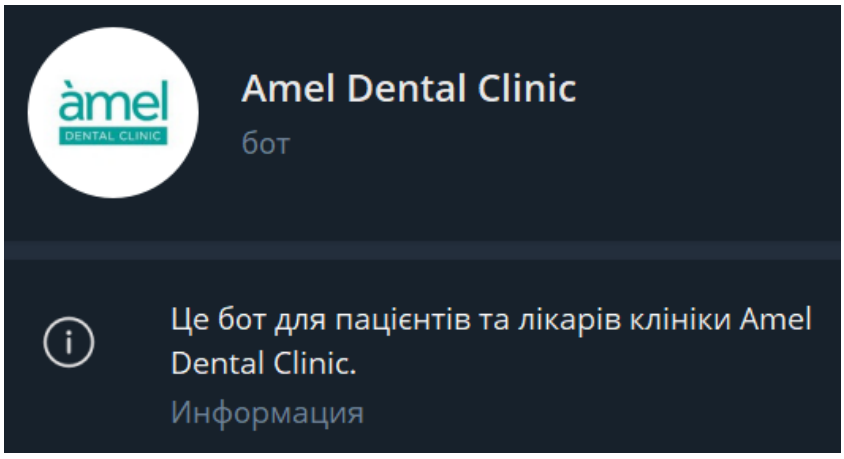

Рисунок 4.8 – Зображення, назва та опис Telegram-бота

Натискання кнопки “Запустити” рівносильно вводу команди /start, яка виводить початкове повідомлення з описом можливостей бота і кнопку “Для пацієнтів”. Це показано на рисунку 4.9

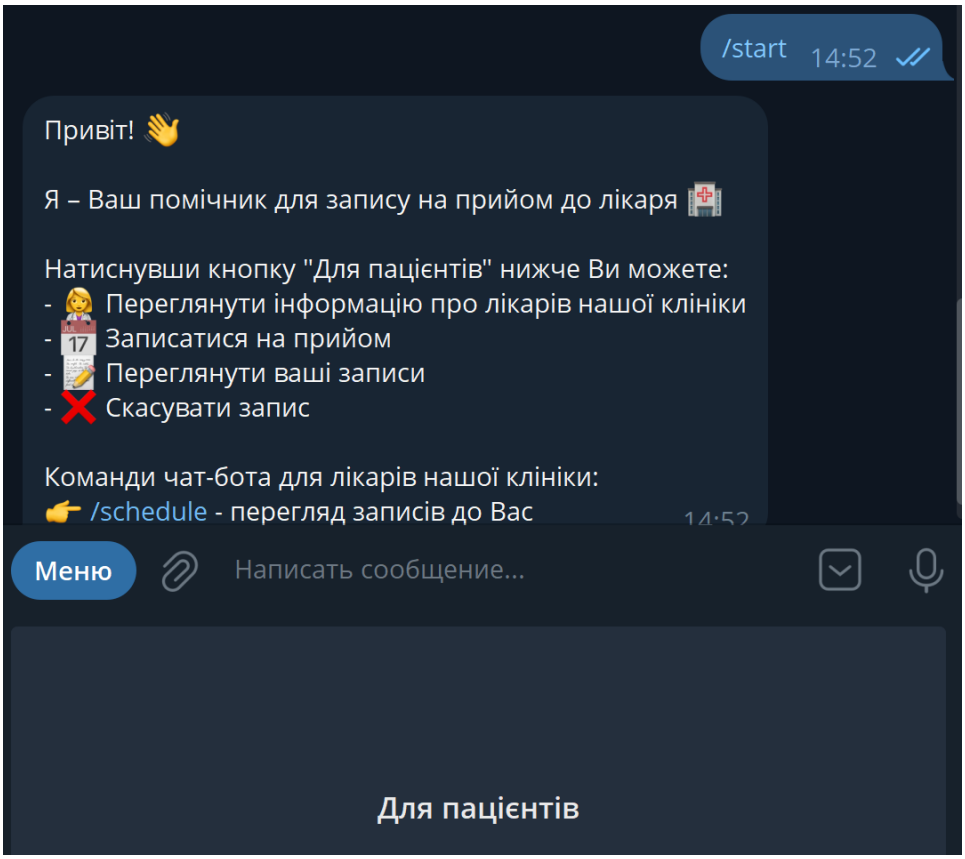

Рисунок 4.9 – Початкове повідомлення бота

Зліва від поля вводу повідомлення є кнопка “Меню”, натиснувши на яку можна побачити доступні команди бота, що теж налаштовано за допомогою головного боту Telegram @BotFather. Це показано на рисунку 4.10

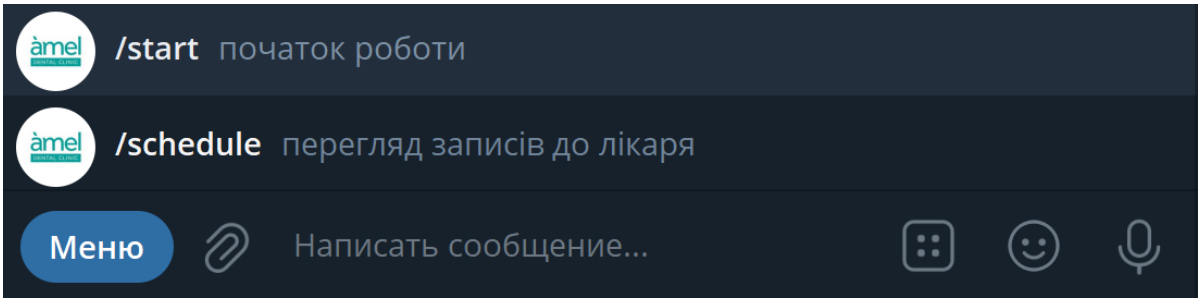

Рисунок 4.10 – Доступні команди бота

Натиснувши на кнопку “Для пацієнтів” відкривається веб-додаток, а саме сторінка з лікарями як показано на рисунку 4.11

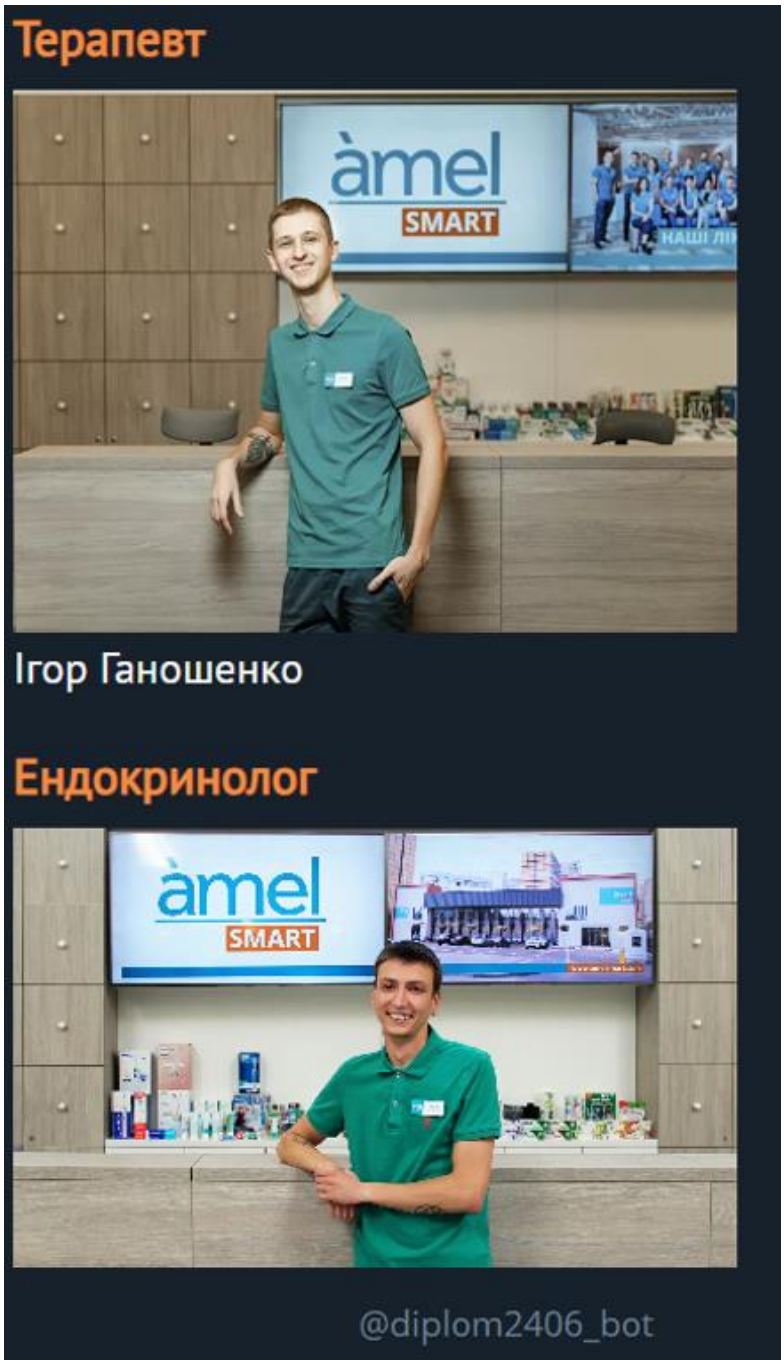

Рисунок 4.11 – Сторінка з лікарями

Натиснувши на фотографію лікаря відкривається сторінка з інформацією про лікаря як показано на рисунку 4.12

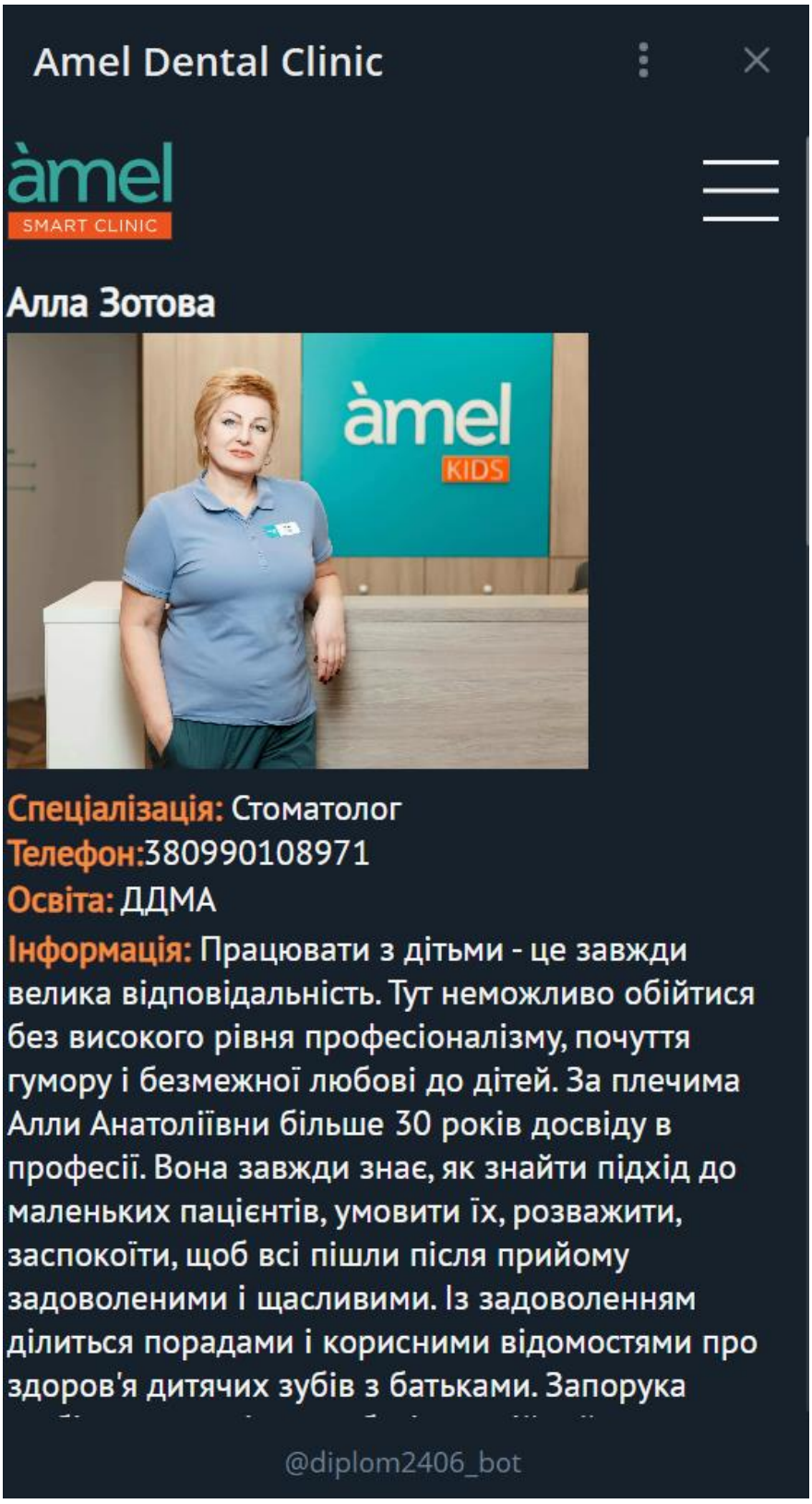

Рисунок 4.12 – Сторінка з інформацією про лікаря

Також показується графік роботи лікаря та внизу сторінки показано контакти клініки, режим роботи та мапа, де показується місцезнаходження клініки. Це показано на рисунку 4.13

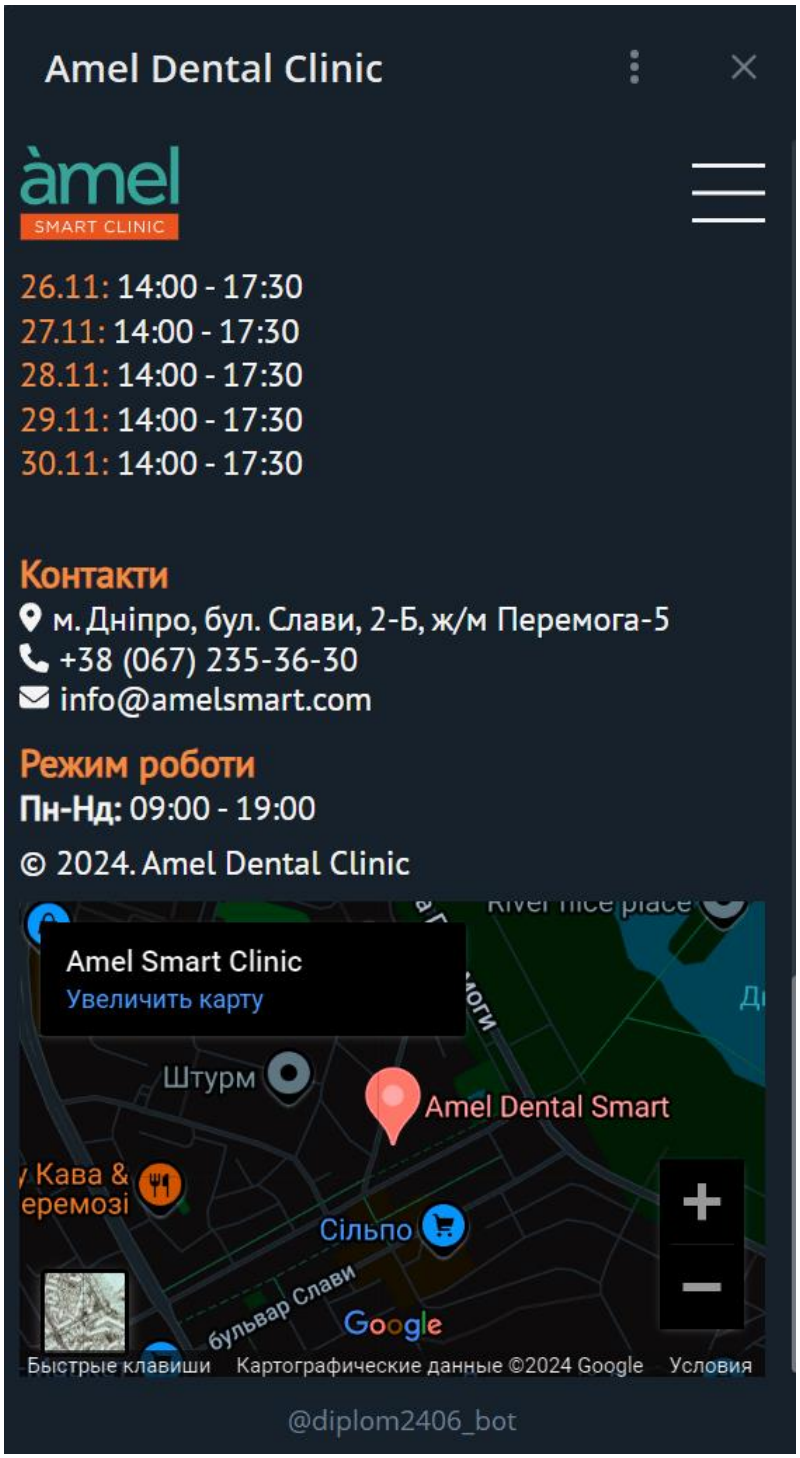

Рисунок 4.13 – Графік роботи лікаря, контакти клініки, режим роботи та мапа

Натиснувши на 3 лінії справа зверху (меню-бургер), показується навігаційне меню як показано на рисунку 4.14

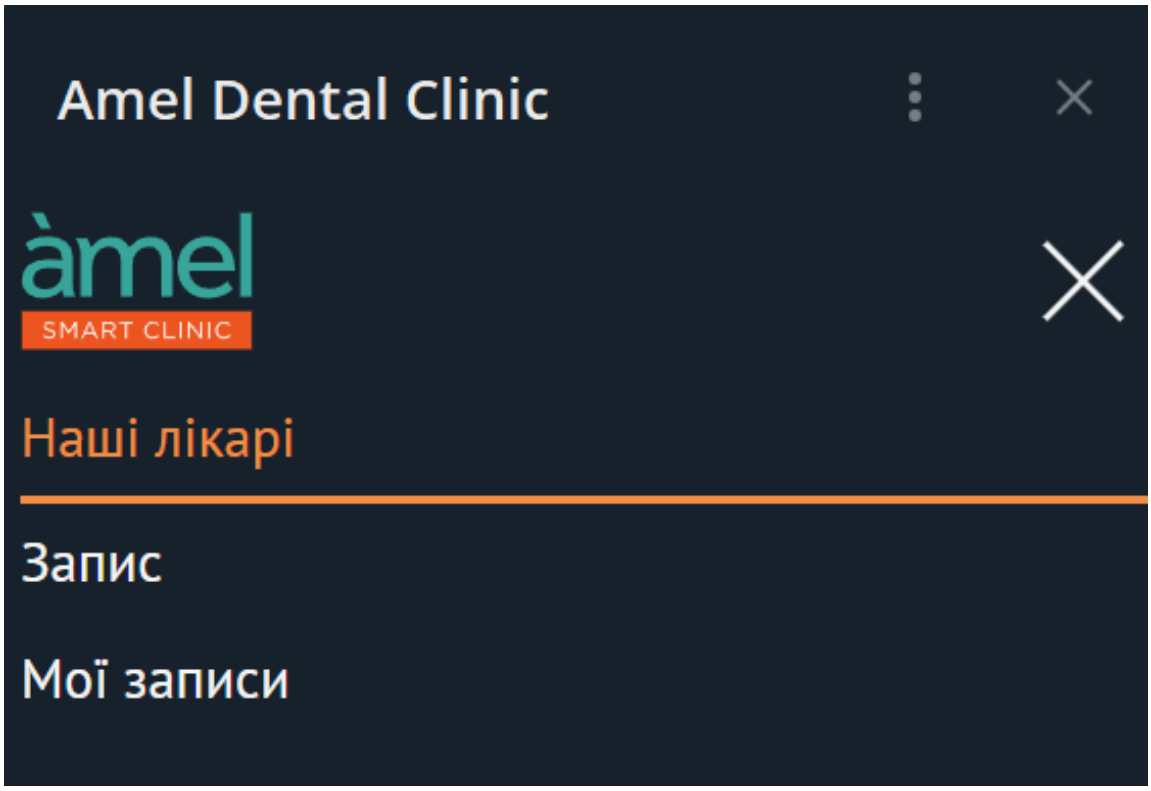

Рисунок 4.14 – Навігаційне меню

Натиснувши на “Запис”, відкривається сторінка для запису.

Поля ПІБ та номер телефону обов’язкові до заповнення. Номер телефону повинен бути введений у форматі 380XXXXXXXXX. Якщо номер не починається з 380, містить будь-які інші символи окрім цифр або містить меньше або більше 12 цифр, то він вважається невірним. Ввівши ПІБ та номер телефону у правильному форматі, обираємо спеціалізацію. Тепер показуються лікарі за обраною спеціалізацією. Вибравши лікаря показуються доступні дати для запису до обраного лікаря як показано на рисунку 4.15

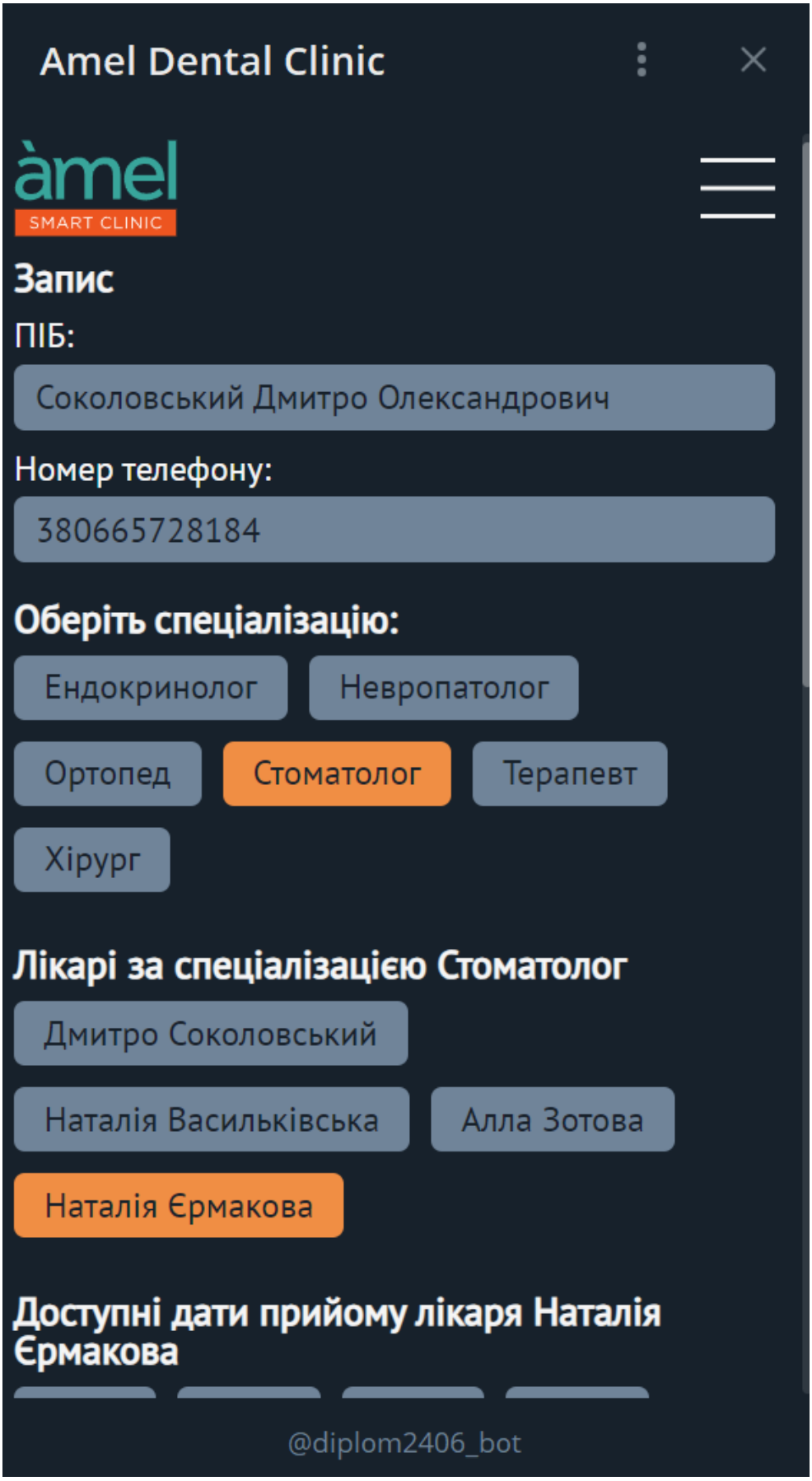

Рисунок 4.15 – Ввод ПІБ та номер телефону, вибір спеціалізації та лікаря

Вибравши лікаря показуються доступні дати для запису. Записатися можна тільки від завтрашнього дня та максимально на місяць вперед. Вибравши дату показуються доступні години. Вибравши час, відображається кнопка “Записатися” як показано на рисунку 4.16. Кнопка відображається тільки в разі коректного вводу номера телефону, ПІБ та вибору часу прийому.

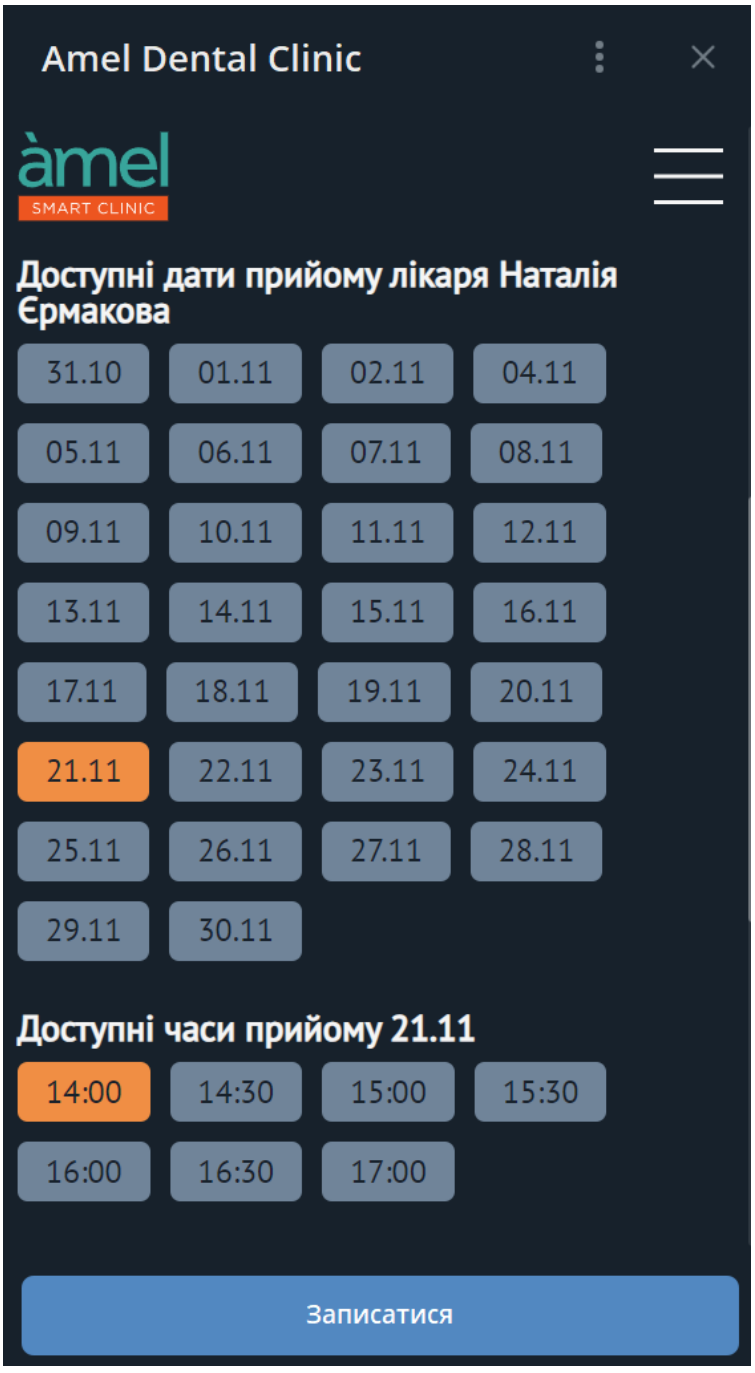

Рисунок 4.16 – Вибір дати та часу прийому

Натиснувши кнопку “Записатися”, в разі успішності чат-бот надсилає повідомлення з інформацією про запис як показано на рисунку 4.17

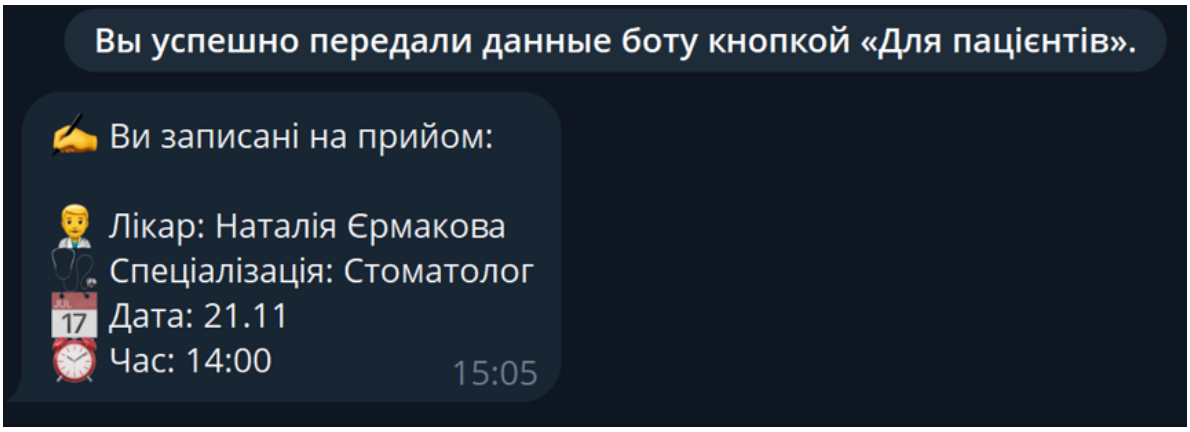

Рисунок 4.17 – Повідомлення пацієнту про успішність запису

Лікарю приходить повідомлення в той же телеграм-бот про запис до нього на прийом з даними пацієнта, дати та часу прийому як показано на рисунку 4.18

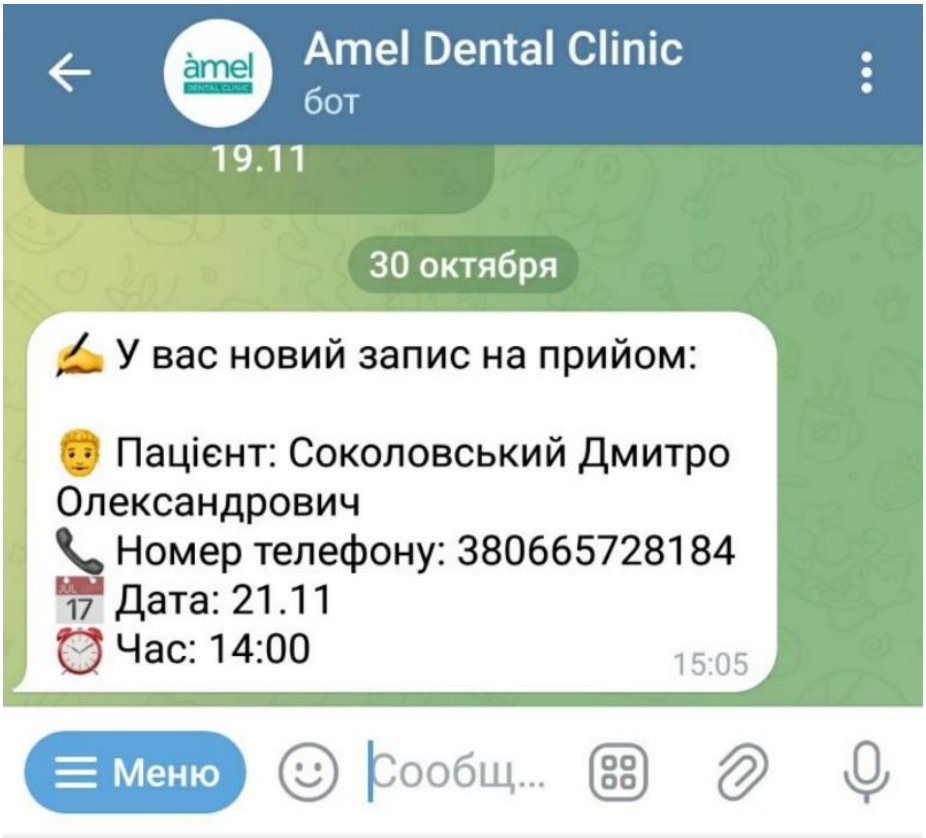

Рисунок 4.18 – Повідомлення лікарю про успішність запису

Після успішного запису більше неможливо записатися на ту же дату і в той же час.

Введемо команду /schedule в акаунті лікаря. Показується відсортовані дати. Дати відсортовані від меншого до більшого та показуються тільки від сьогоднішнього дня і на місяць вперед. Натиснувши на дату, показуються записи на обрану дату. Це показано на рисунку 4.19

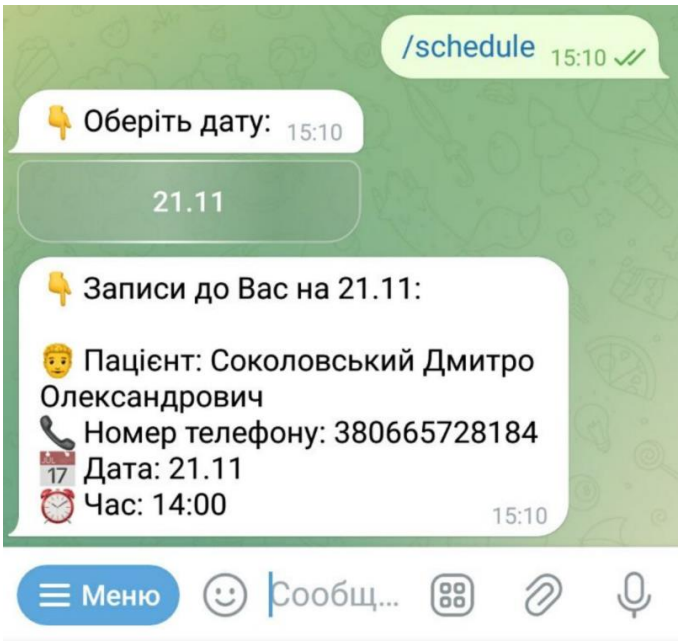

Рисунок 4.19 – Перегляд графіку роботи лікаря

Якщо виникає помилка при отриманні графіку роботи лікаря, користувач не є лікарем або записів нема, то чат-бот надсилає повідомлення з помилкою як показано на рисунку 4.20

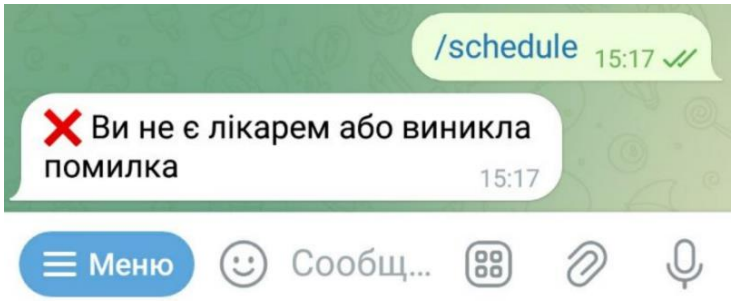

Рисунок 4.20 – Помилка при отриманні графіку роботи лікаря

Натиснувши на “Мої записи” в навігаційному меню, відкривається сторінка із записами пацієнта як показано на рисунку 4.21. Якщо записів нема, то буде відповідний надпис про це.

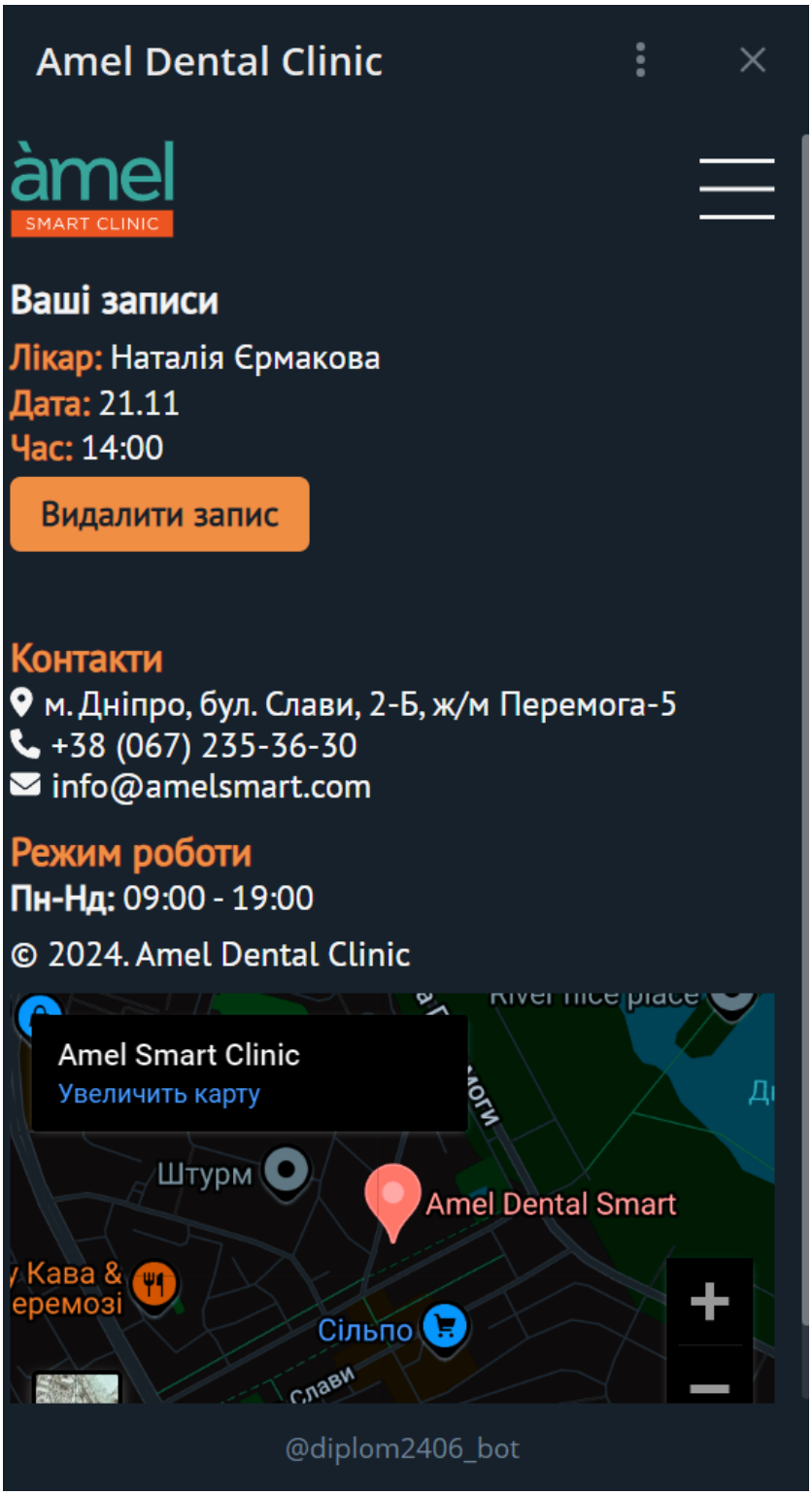

Рисунок 4.21 – Сторінка із записами пацієнта

Натиснувши на кнопку “Видалити запис”, відображається модальне вікно з проханням підтвердити дію як показано на рисунку 4.22

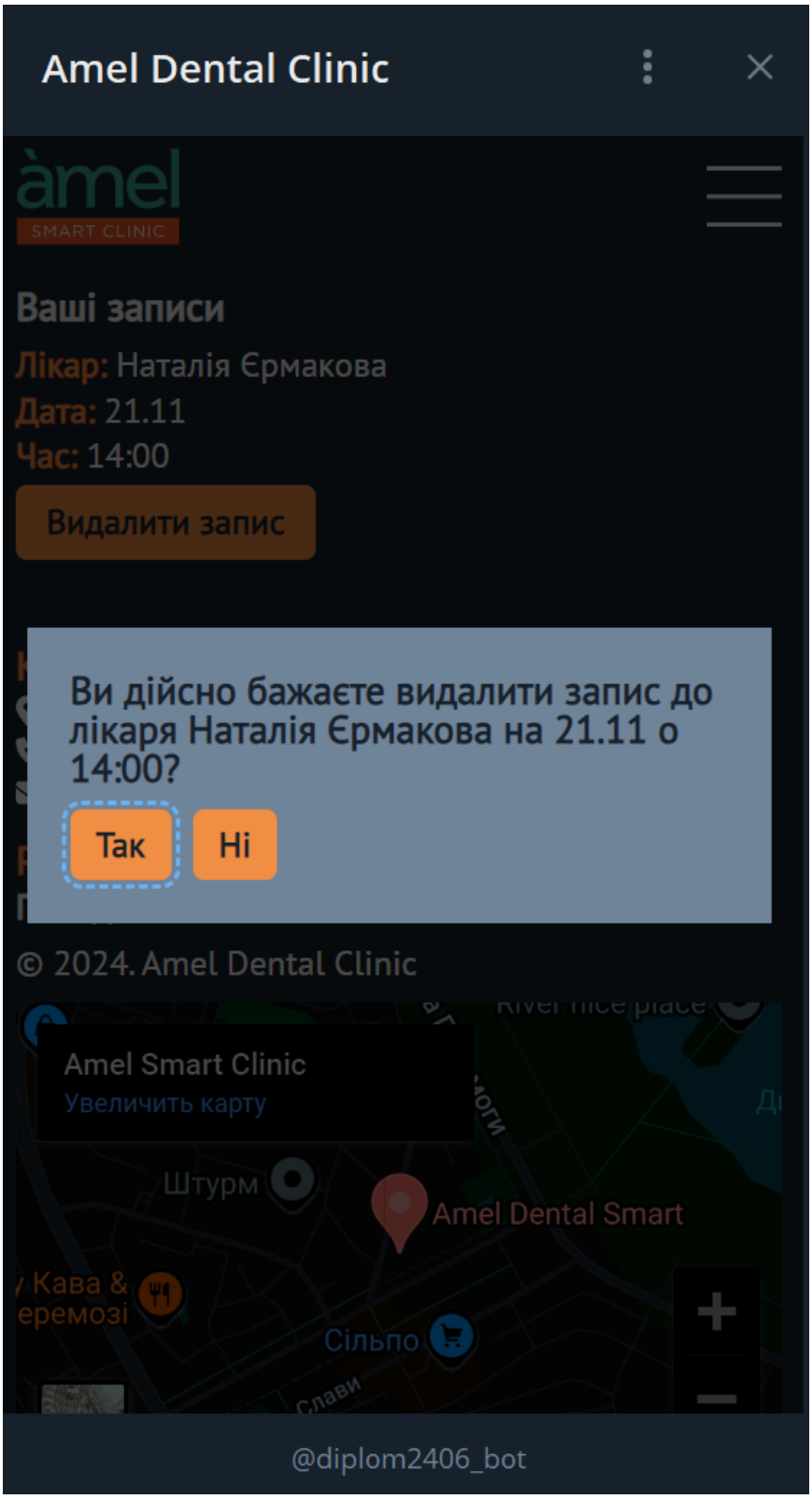

Рисунок 4.22 – Модальне вікно

Якщо натиснути “Ні”, то модальне вікно закриється і запис не буде скасовано. Натиснувши “Так”, запис скасовується. В разі успішного скасування запису чат-бот надсилає повідомлення з інформацією про скасований запис як показано на рисунку 4.23

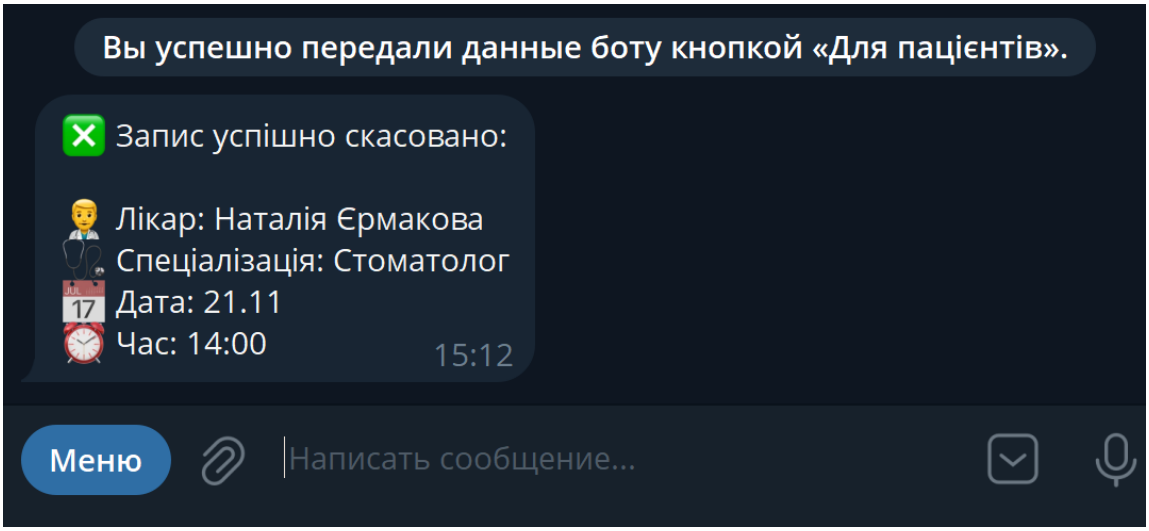

Рисунок 4.23 – Повідомлення пацієнту про успішно скасований запис

Лікарю приходить повідомлення в той же телеграм-бот про скасування запису до нього з даними пацієнта, дати та часу прийому як показано на рисунку 4.24

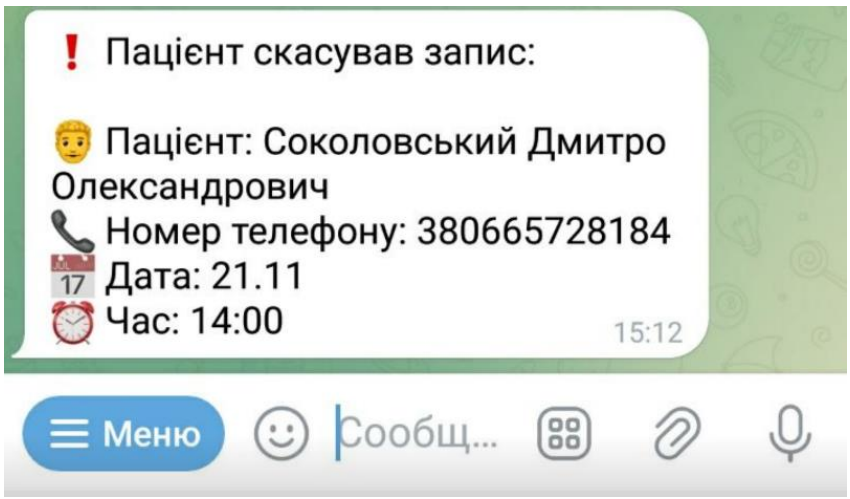

Рисунок 4.24 – Повідомлення лікарю про скасування запису

Веб-додаток для пацієнтів є адаптивним, тобто дизайн автоматично підлаштовується під розмір екрана пристрою для зручного перегляду.

Веб-додаток також є доступним, тобто люди з різними можливостями, зокрема з інвалідністю, можуть легко ним користуватися. Завдяки ретельно продуманій структурі та навігації, користувачі можуть зручно переміщуватися по сайту, просто натискаючи клавішу Tab. Крім того, сайт оптимізований для роботи зі скрінрідерами, які здатні правильно озвучувати вміст і полегшувати взаємодію для людей із порушеннями зору.

При виникненні будь-якої помилки, її текст відображається у вигляді спливаючого вікна.

4.8 Висновки до розділу

У четвертому розділі було детально розглянуто процес розробки програмного забезпечення для системи Telegram-бота, призначеного для запису пацієнтів до медичної клініки. Спочатку визначено призначення програми та обґрунтовано вибір технічних характеристик, які забезпечують ефективне виконання завдань, пов'язаних із автоматизацією процесу запису на прийом до лікаря.

На основі вимог до функціональності було реалізовано логічну структуру програми з використанням сучасних інструментів, таких як Node.js, Next.js, React.js, Express.js, MongoDB, а також створено зручний інтерфейс для взаємодії з пацієнтами та лікарями через Telegram Mini Apps. Завдяки застосованим технологіям досягнуто оптимізації процесів запису, скорочено час обслуговування пацієнтів і підвищено ефективність роботи лікарів.

Розглянуто структуру програми, описано алгоритми її роботи, взаємодію між різними компонентами та спосіб організації вхідних і вихідних даних. Було також описано технічні засоби, необхідні для функціонування програми, зокрема типи серверів і засоби розгортання.

Таким чином, розроблена система відповідає поставленим вимогам і забезпечує ефективне рішення для автоматизації запису на прийом до лікаря, значно підвищуючи якість обслуговування пацієнтів та ефективність роботи клініки.

5 ЕКСПЕРИМЕНТАЛЬНИЙ РОЗДІЛ

5.1 Мета і завдання експерименту з порівняння ефективності та продуктивності методів рендерингу

Метою даного експерименту є порівняння ефективності та продуктивності методів рендерингу на основі підходів CSR (Client-Side Rendering) та SSR (Server-Side Rendering) у контексті веб-дІодатку в Telegram. Дослідження спрямоване на оцінку того, який метод рендерингу забезпечує кращу швидкість завантаження, продуктивність, доступність і відповідність рекомендаціям щодо SEO як на мобільних пристроях, так і на настільних комп’ютерах.

Завдання експерименту:

– вимірювання та порівняння показників швидкості завантаження;

– вимірювання та порівняння ключових метрик та оцінок продуктивності;

– аналіз відповідності теоретичних та експериментальних досліджень;

– характеристика новизни отриманих результатів.

5.2 Методика експерименту

Для досягнення мети та виконання завдань експерименту розроблено методику, яка передбачає етапи вимірювання ключових показників продуктивності, аналізу завантаження та оцінки відповідності стандартам.

Експериментальне середовище буде складатися з набору пристроїв та умов мережевого з’єднання, обраних для коректного порівняння показників продуктивності. Тестування буде проводитись на двох пристроях – десктопному комп’ютері та мобільному телефоні, що дозволить відобразити специфіку роботи додатка на різних платформах. Мережеві умови будуть представлені трьома варіантами з’єднання: без затримок (No throttling), що імітує високошвидкісне з'єднання; Fast 3G, що відповідає стандартному підключенню швидкого 3G; та Slow 3G, для моделювання низькошвидкісного з'єднання. Це забезпечить реалістичність

умов, у яких функціонує веб-додаток у користувачів з різним рівнем доступу до Інтернету.

Для вимірювання ключових метрик продуктивності буде використано інструмент для аудиту веб-сторінок Lighthouse, який дозволить оцінити:

1) Performance – продуктивність;
2) Accessibility – доступність;
3) Best Practice – відповідність найкращим практикам;
4) SEO – доступність;
5) Progressive Web Application (PWA) – відповідність прогресивним веб-додаткам.

Крім того, він дозволить оцінити конкретні метрики продуктивності, а саме:

1) First Contentful Paint (FCP) – перше відтворення вмісту: час, за який користувач бачить перший елемент вмісту на сторінці;
2) Time to First Byte (TTFB) – час до першого байта: час, за який перший байт даних передається з сервера до браузера;
3) Speed Index (SI) – індекс швидкості: показник того, як швидко весь вміст сторінки стає видимим користувачу;
4) Largest Contentful Paint (LCP) – найбільше відтворення вмісту: час, за який найбільший елемент вмісту візуально з’являється на екрані;
5) Cumulative Layout Shift (CLS) – кумулятивне зміщення макета: вимірює всі зміни макета, які відбуваються під час завантаження сторінки і призводять до зміщення видимих елементів.

Також буде застосовуватись інструменти Chrome DevTools, що емулюють різні мережеві умови та збирають дані щодо швидкості завантаження додатка, зокрема час DOMContentLoaded, коли браузер завершує завантаження HTML-документа та будує DOM, та час Load, коли всі ресурси сторінки повністю завантажені.

Процедура тестування включатиме кілька етапів:

1) відкриття CSR-версії додатка у середовищі без затримок;

2) аудит веб-сторінки;

3) вимірювання швидкості завантаження додатка при різних мережевих умовах;

4) ті ж самі дії для SSR-версії;

5) занесення зібраних даних в таблиці.

Отримані результати буде порівняно з теоретичними очікуваннями. Це дозволить проаналізувати, наскільки реальні показники відповідають передбаченням та теоретичним припущенням, та виявити, де саме існують розбіжності.

5.3 Вимоги до експерименту

Під час аналізу враховуватимуться можливі похибки та обмеження експерименту. Серед них – вплив мережевих умов, апаратних характеристик пристроїв та точності вимірювання інструментів. Умови емуляції мережі можуть не повністю відображати реальні умови користування. Для підвищення достовірності результатів визначаємо основні вимоги до експерименту:

– використання ідентичної бази даних, чат-боту, API та структури компонентів для обох версій веб-додатку, щоб виключити вплив різних наборів даних та логіки на продуктивність;

– ідентичні пристрої для тестування, щоб мінімізувати вплив апаратних засобів;

– повторюваність вимірювань, щоб мінімізувати випадкові похибки;

– стандартизоване збереження результатів у таблицях і графіках, що сприятиме виявленню тенденцій та відмінностей між методами рендерингу.

5.4 Результати експерименту

5.4.1 Сутність експерименту

Експеримент був спрямований на порівняння двох методів рендерингу веб-додатка, інтегрованого в Telegram: клієнтського рендерингу (CSR) і серверного

рендерингу (SSR). Метою експерименту було визначити, який з підходів є більш ефективним з точки зору швидкості завантаження, взаємодії з користувачем та стабільності роботи, особливо в умовах різних швидкостей інтернет-з'єднання.

Основним об'єктом дослідження є веб-додаток для запису пацієнтів до медичної клініки через чат-бота в Telegram. Цей додаток використовує API (на основі Express.js) для взаємодії з базою даних MongoDB (для зберігання інформації про лікарів і записи пацієнтів), а також Telegram API для передачі даних між ботом і веб-інтерфейсом. У ході експерименту тестувалися дві версії додатка:

– CSR-версія (на основі React) рендерила компоненти та дані повністю на стороні клієнта після завантаження базової HTML-сторінки;

– SSR-версія (на основі Next.js) генерувала HTML-код на сервері, щоб передати користувачеві попередньо сформовану сторінку, що мало скоротити час початкового завантаження контенту.

Для оцінки продуктивності кожної з версій було визначено кілька ключових показників, таких як FCP (First Contentful Paint), LCP (Largest Contentful Paint), CLS (Cumulative Layout Shift), Speed Index, та TTFB (Time to First Byte). Ці метрики дозволяють отримати всебічне уявлення про швидкість завантаження сторінок, стабільність відображення контенту та зручність використання додатка.

Завдання експерименту написано в пункті 5.1, методика – в пункті 5.2, а вимоги до експерименту – в пункті 5.3

5.4.2 Результати експерименту в цифрах і фактах

Порівняння швидкості завантаження між CSR і SSR при різних мережевих умовах (без затримок, Fast 3G, Slow 3G) показано в таблиці 5.1

Таблиця 5.1 – Порівняння швидкості завантаження між CSR і SSR при різних мережевих умовах

Метод рендерингу	CSR		SSR	
Подія завантаження	DOMContentLoaded	Load	DOMContentLoaded	Load
Без затримок				
Час	596 мс	597 мс	1.44 с	2.07 с
	539 мс	539 мс	1.50 с	2.05 с
	343 мс	346 мс	1.46 с	2.35 с
	333 мс	336 мс	1.38 с	2.29 с
	321 мс	324 мс	1.28 с	2.03 с
	321 мс	324 мс	1.52 с	2.36 с
	195 мс	199 мс	1.44 с	1.90 с
	202 мс	205 мс	1.47 с	2.03 с
	205 мс	208 мс	1.52 с	2.37 с
	206 мс	206 мс	1.45 с	2.18 с
Fast 3G				
Час	875 мс	878 мс	1.43 с	1.45 с
	897 мс	900 мс	1.36 с	1.38 с
	732 мс	735 мс	1.48 с	1.50 с
	866 мс	867 мс	1.38 с	1.39 с
	864 мс	867 мс	1.45 с	1.47 с
	909 мс	912 мс	1.61 с	1.62 с
	865 мс	868 мс	1.44 с	1.45 с
	822 мс	825 мс	1.71 с	1.73 с
	727 мс	731 мс	1.55 с	1.56 с
	730 мс	735 мс	1.47 с	1.48 с

Кінець таблиці 5.1

Slow 3G				
Час	2.30 с	2.31 с	2.54 с	3.62 с
	2.39 с	2.39 с	2.53 с	3.55 с
	2.32 с	2.32 с	2.58 с	3.29 с
	2.30 с	2.30 с	2.53 с	3.11 с
	2.34 с	2.34 с	2.57 с	3.17 с
	2.32 с	2.32 с	2.51 с	3.07 с
	2.32 с	2.32 с	2.54 с	3.21 с
	2.32 с	2.32 с	2.59 с	2.36 с
	2.33 с	2.33 с	2.54 с	3.54 с
	2.32 с	2.33 с	2.63 с	3.52 с

Копія екрану під час порівняння швидкості завантаження при Fast 3G показано на рисунку 5.1

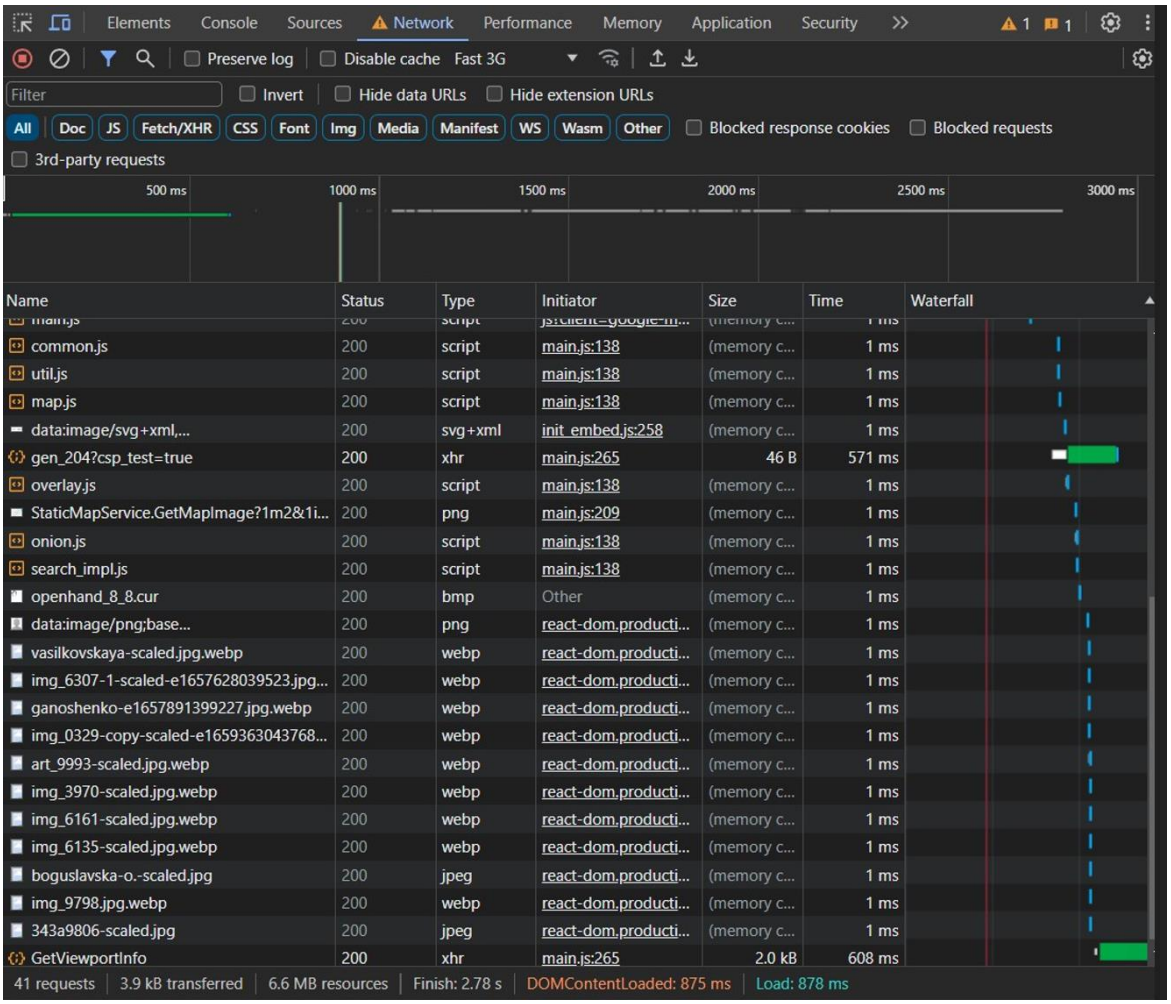

Рисунок 5.1 – Копія екрану під час порівняння швидкості завантаження при Fast 3G

Порівняння метрик продуктивності (FCP, TTFB, SI, LCP, CLS) між CSR і SSR на різних пристроях показано в таблиці 5.2

Таблиця 5.2 – Порівняння метрик продуктивності між SPA і SSR на різних пристроях

Метод рендерингу	CSR					SSR				
Метрика	FCP	TTFB	SI	LCP	CLS	FCP	TTFB	SI	LCP	CLS
Mode – Navigation, Device - Desktop										
Час (с)	0.8	0	1.2	0.9	0.739	0.4	0	1.3	1.4	0.009

Кінець таблиці 5.2

	0.7	0	1.2	0.9	0.737	0.4	0	1.2	1.2	0.009
	0.8	0	1.2	0.9	0.737	0.3	0	1.1	1.4	0.009
	0.8	0	1.3	0.9	0.737	0.4	0	1.1	1.2	0.009
	0.8	0	1.2	1.0	0.737	0.3	0	1.0	1.2	0.009
Mode – Navigation, Device - Mobile										
Час (с)	2.4	0	3.5	3.2	0.383	1.7	0	3.8	4.4	0.013
	2.4	0	3.4	3.2	0.383	1.7	0	3.0	6.2	0.013
	1.9	0	2.9	3.2	0.408	1.7	0	3.0	6.2	0.013
	2.4	0	3.4	3.2	0.383	1.7	0	3.0	6.8	0.013
	2.5	0	3.4	3.9	0.383	1.7	0	3.1	6.2	0.013

Порівняння оцінки продуктивності між CSR і SSR на різних пристроях показано в таблиці 5.3

Таблиця 5.3 – Порівняння оцінки продуктивності (Performance, Accessibility, Best Practice, SEO, PWA) між CSR і SSR на різних пристроях

Метод рендерингу	CSR					SSR				
Показник	Performance	Accessibility	Best Practice	SEO	PWA	Performance	Accessibility	Best Practice	SEO	PWA
Mode – Navigation, Device - Desktop										
Оцінка	74	92	95	100	29	95	87	95	100	29
	75	92	95	100	29	97	87	95	100	29
	75	92	95	100	29	95	87	95	100	29
	74	92	95	100	29	97	87	95	100	29
	74	92	95	100	29	97	87	95	100	29

Кінець таблиці 5.3

Mode – Navigation, Device - Mobile										
Оцінка	70	92	95	98	38	83	87	95	98	38
	71	92	95	98	38	76	87	95	98	38
	72	92	95	98	38	77	87	95	98	38
	71	92	95	98	38	75	97	95	98	38
	70	92	95	98	38	78	87	95	98	38

Копія екрану під час порівняння метрик та оцінок продуктивності показано на рисунку 5.2

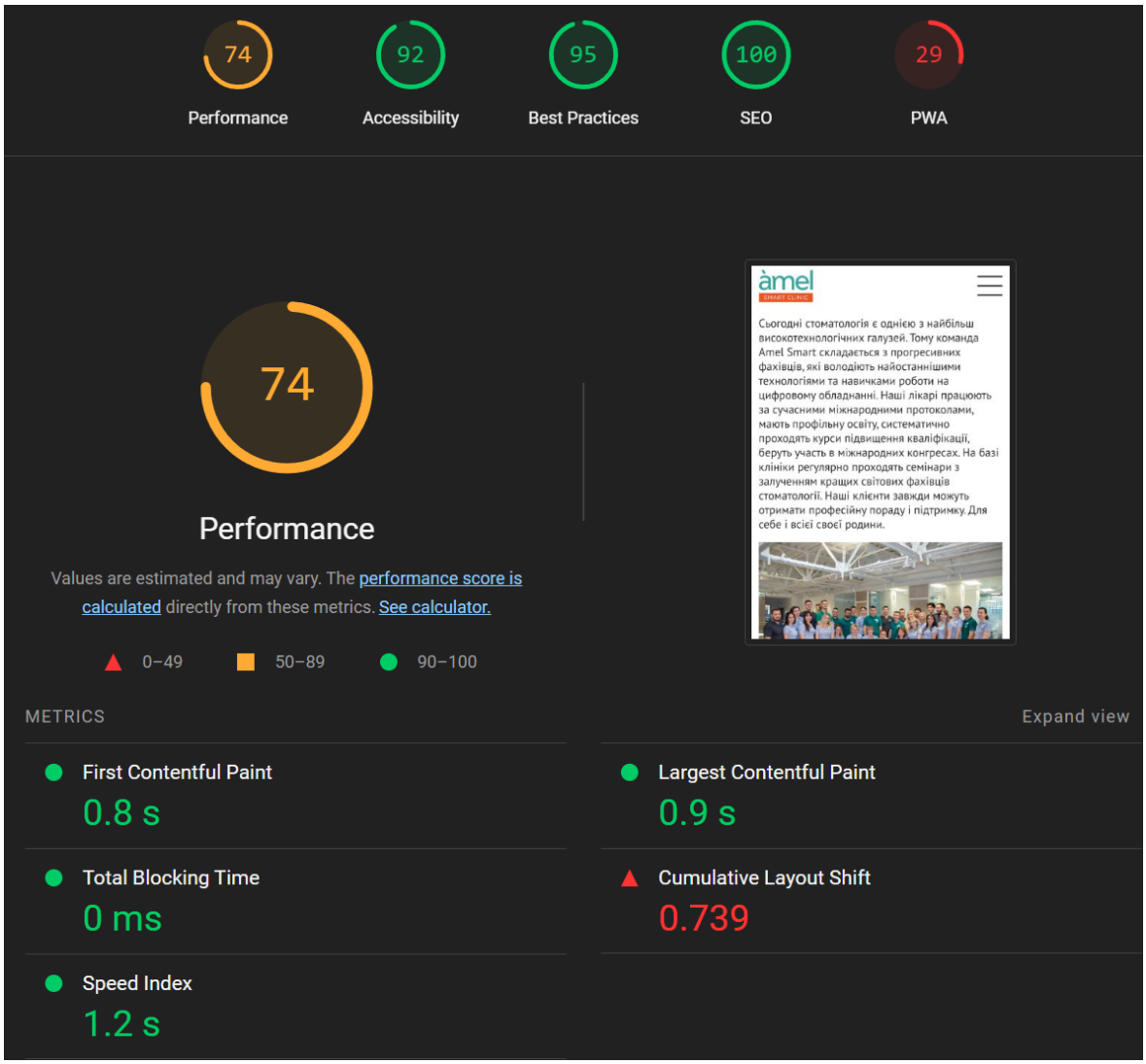

Рисунок 5.2 – Копія екрану під час порівняння метрик та оцінок продуктивності

Середні значення швидкості завантаження:

1) без затримок – CSR ≈ 326 мс для DOMContentLoaded та ≈ 328 мс для Load, тоді як SSR ≈ 1.45 с для DOMContentLoaded та ≈ 2.16 с для Load;

2) Fast 3G – CSR ≈ 828 мс для DOMContentLoaded та ≈ 832 мс для Load, тоді як SSR ≈ 1.49 с для DOMContentLoaded та ≈ 1.5 с для Load;

3) Slow 3G – CSR ≈ 2.33 с для DOMContentLoaded та ≈ 2.33 с для Load, тоді як SSR ≈ 2.56 с для DOMContentLoaded та ≈ 3.24 с для Load.

CSR виявився кращим у швидкості завантаження базової сторінки (HTML-документ) та всього вмісту сторінки, включно з CSS, зображеннями та скриптами. Особливо це видно у швидких мережевих умовах (без затримок або Fast 3G). Це обумовлено тим, що CSR завантажує сторінку та вміст з клієнтської сторони.

Середні значення метрик:

1) FCP – CSR ≈ 0.78 с для Desktop та ≈ 2.32 с для Mobile, тоді як SSR ≈ 0.36 с для Desktop та ≈ 1.7 с для Mobile;

2) TTFB – CSR ≈ 0 с для Desktop та ≈ 0 с для Mobile, тоді як SSR ≈ 0 с для Desktop та ≈ 0 с для Mobile;

3) SI – CSR ≈ 1.22 с для Desktop та ≈ 3.32 с для Mobile, тоді як SSR ≈ 1.14 с для Desktop та ≈ 3.18 с для Mobile;

4) LCP – CSR ≈ 0.92 с для Desktop та ≈ 3.34 с для Mobile, тоді як SSR ≈ 1.28 с для Desktop та ≈ 5.96 с для Mobile;

5) CLS – CSR ≈ 0.74 с для Desktop та ≈ 0.39 с для Mobile, тоді як SSR ≈ 0.009 с для Desktop та ≈ 0.013 с для Mobile.

Висновки по значенням метрик:

– SSR забезпечує нижчий First Contentful Paint (FCP), що дозволяє користувачам побачити початковий контент швидше, особливо на мобільних пристроях. У SSR сторінка завантажується вже попередньо відрендерена на сервері, тому користувач отримує повністю готову HTML-структуру, яка може відразу показувати контент, тому FCP нижчий в SSR, особливо при слабкому

підключенні, де повний оброблений контент завантажується швидше, ніж якщо все необхідно обробляти на стороні клієнта;

– Time to First Byte (TTFB) ідентичний, оскільки обидва залежать від відповіді сервера;

– SSR забезпечує трішки нижчий Speed Index (SI), що дозволяє користувачам швидше побачити весь вміст сторінки. Цей показник має тенденцію бути кращим у SSR, оскільки сторінка надходить у більш заповненому стані. Це дозволяє браузеру відображати значну частину сторінки відразу, тоді як CSR може завантажувати контент частинами;

– CSR забезпечує нижчий Largest Contentful Paint (LCP), особливо на мобільних пристроях, що поліпшує сприйняття швидкості додатка, роблячи його зручнішим для користувачів;

– SSR має значно нижчий Cumulative Layout Shift (CLS) як на мобільних, так і на десктопних пристроях, що забезпечує кращу стабільність візуального відображення інтерфейсу. Це важливо для додатків, де велика кількість інформації повинна відображатися швидко та без зміщень, таких як інформаційні панелі.

Середні значення оцінок продуктивності:

1) Performance – CSR ≈ 74.4 на Desktop та ≈ 70.8 на Mobile, тоді як SSR ≈ 96.2 на Desktop та ≈ 77.8 на Mobile;

2) Accessibility – CSR ≈ 92 на Desktop та ≈ 92 на Mobile, тоді як SSR ≈ 87 на Desktop та ≈ 89 на Mobile;

3) Best Practice – CSR ≈ 95 на Desktop та ≈ 95 на Mobile, тоді як SSR ≈ 95 на Desktop та ≈ 95 на Mobile;

4) SEO – CSR ≈ 100 на Desktop та ≈ 98 на Mobile, тоді як SSR ≈ 100 на Desktop та ≈ 98 на Mobile;

5) PWA – CSR ≈ 29 на Desktop та ≈ 38 на Mobile, тоді як SSR ≈ 29 на Desktop та ≈ 38 на Mobile.

Висновки по оцінкам продуктивності:

1) Performance – SSR має кращі оцінки продуктивності, ніж CSR;

2) Accessibility – обидва методи забезпечують високі показники доступності, але CSR трошки кращі;

3) Best Practice – оцінкі однакова, що показує відповідність стандартам коду;

4) SEO – оцінкі однакові для обох підходів, але взагалі SSR набагато краще підходить для SEO, оскільки контент заздалегідь рендериться і краще індексується пошуковими системами, що є критичним для контентних сайтів;

5) PWA – оцінкі однакові для обох підходів, але взагалі CSR трохи краще підтримує інтерактивні PWA-функції, що робить його кращим для додатків, які потребують роботи офлайн.

Переваги SSR для медичної клініки:

– SEO та видимість контенту: SSR забезпечує кращу індексацію контенту та миттєве відображення готових сторінок, що є важливим для інформаційних медичних додатків;

– швидше відображення контенту (FCP): Оскільки SSR генерує HTML на сервері, готовий контент надсилається безпосередньо користувачеві. Це зменшує час до першого відображення вмісту, а метрика FCP має нижче значення для SSR порівняно з CSR. Для медичних додатків це критично, адже важлива інформація, як-от розклад лікарів, відображається відразу;

– стабільність інтерфейсу (CLS): Користувачі зможуть швидко переглядати важливу інформацію, наприклад, розклад прийомів лікарів, без зміщень інтерфейсу, характерних для CSR, де контент може довантажуватися поступово;

– єдиний користувацький досвід на різних швидкостях мережі: SSR забезпечує однаковий досвід незалежно від якості мережі, оскільки контент з’являється в готовому стані без додаткових запитів клієнтської сторони, що може значно зменшити Speed Index для SSR.

Переваги вибору CSR:

– CSR часто показує швидше завершення подій DOMContentLoaded і Load, оскільки HTML завантажується з мінімальним контентом і багато запитів

надсилаються вже після основного завантаження. У той же час, в SSR Load може відбуватися довше, оскільки він включає більше завантажуваного HTML-контенту, але при цьому це забезпечує швидше візуальне заповнення контенту для користувача.

SSR є оптимальним для контенту, що вимагає миттєвого відображення та має потребу в гарній SEO-індексації. Для медичної клініки це особливо важливо, адже користувачі очікують швидкого доступу до інформації щодо лікарів та запису на прийом.

Якщо програма вимагає динамічного оновлення контенту або швидких переходів між сторінками, CSR може бути доцільнішим, наприклад, для панелей адміністрування або CRM-систем. CSR дозволяє зменшити час переходу між сторінками, що підвищує швидкість оновлення контенту у додатках із високою взаємодією.

Для веб-додатку, вбудованого в чат-бот для медичної клініки в Telegram, де ключовими вимогами є стабільність інтерфейсу, швидкий доступ до інформації та можливість індексації контенту, найбільш оптимальним варіантом є SSR.

5.4.3 Аналіз відповідності теоретичних та експериментальних досліджень

Аналіз відповідності теоретичних і експериментальних результатів дозволив оцінити, наскільки ефективність серверного (SSR) та клієнтського рендерингу (CSR) у веб-додатку відповідає теоретично очікуваним перевагам і недолікам цих підходів. Порівняння результатів показало наступні закономірності:

а) швидкість завантаження – теоретично очікувалось, що CSR буде більш швидким у відображенні контенту на швидких мережах, тоді як SSR повинен мати перевагу на повільних мережах, оскільки завантажує попередньо відрендерений HTML. Експериментально було підтверджено, що CSR забезпечує швидше завантаження сторінки у швидких мережах (без затримок та Fast 3G), тоді як SSR показав кращі

результати на повільних мережах (Slow 3G), що відповідає теоретичним припущенням;

b) метрики продуктивності:
    1) FCP (First Contentful Paint) – теоретично передбачалось, що SSR матиме нижчі значення FCP, оскільки початковий контент відображається раніше. Експериментально було підтверджено, що SSR дійсно забезпечує швидший FCP, особливо у повільних умовах мобільного інтернету;
    2) TTFB (Time to First Byte) – теоретично очікувалось, що значення TTFB будуть однаковими для обох підходів, що підтверджується експериментальними даними, де різниця у значеннях TTFB не була виявлена;
    3) SI (Speed Index) та LCP (Largest Contentful Paint) – теоретично передбачалось, що SSR матиме нижчий SI завдяки відображенню більшого обсягу контенту на початку, тоді як CSR зможе забезпечити кращий LCP через швидше завантаження окремих компонентів. Експерименти підтвердили ці припущення, оскільки SSR справді показав трохи нижчий SI, а CSR – кращий LCP, особливо на мобільних пристроях;
    4) CLS (Cumulative Layout Shift) – очікувалось, що SSR продемонструє нижчий CLS через стабільне початкове завантаження, і експеримент підтвердив, що SSR забезпечує значно нижчий показник CLS, що покращує стабільність відображення контенту.

Оцінки продуктивності:

1) Performance – теоретично SSR мав показувати вищу оцінку продуктивності завдяки попередньому рендерингу на сервері. Експеримент підтвердив цю гіпотезу, оскільки оцінки продуктивності для SSR були вищими, ніж для CSR;

2) Accessibility, Best Practice – передбачалось, що обидва підходи матимуть високі оцінки, але експеримент показав, що доступність для CSR трішки краща;

3) SEO – передбачалось, що SSR матиме вищі оцінки SEO порівняно з CSR, але експеримент показав, що результати ідентичні;

4) PWA – теоретично передбачалось результати будуть ідентичними для обох методів, що підтвердилось в результаті досліджень

Таким чином, експериментальні результати підтвердили основні теоретичні припущення щодо переваг і недоліків CSR та SSR для різних мережевих умов і платформ.

5.4.4 Характеристика новизни результатів

Дослідження рендерингу веб-додатків у середовищі Telegram є відносно новою темою, що зумовлено зростанням популярності Telegram як платформи для бізнес-інтеграцій, зокрема в галузі охорони здоров'я.

Нове в цьому дослідженні полягає в комплексному порівнянні SSR та CSR у специфічному середовищі Telegram, яке має свої обмеження та можливості, відмінні від звичайного веб-браузера. Врахування унікальних вимог платформи, таких як швидке завантаження на мобільних пристроях та стабільність інтерфейсу, дозволило отримати рекомендації щодо оптимального підходу для медичних додатків у Telegram.

Виявлено, що SSR забезпечує переваги у швидкості доступу та стабільності інтерфейсу, що є ключовим для додатків у сфері охорони здоров'я. Це дозволяє швидко та зручно надавати користувачам доступ до важливої інформації, такої як розклад лікарів та доступність запису на прийом. Отримані результати мають новизну завдяки адаптації стандартних методів рендерингу для додатків, що працюють у межах Telegram.

Завдяки дослідженню встановлено оптимальні сценарії використання SSR та CSR залежно від типу контенту. Зокрема, для додатків із великою кількістю

статичної інформації, як у випадку медичних веб-сервісів, рекомендовано використання SSR, що дозволяє досягти кращої індексації та швидкого першого відображення контенту. Водночас для додатків, де необхідні швидкі переходи між сторінками, як у CRM-системах, більше підходить CSR. Це дозволяє створити гнучкі рекомендації для вибору підходу в залежності від вимог до контенту і взаємодії.

Отримані результати є важливими для вдосконалення інтеграцій медичних сервісів у Telegram. Запропонована стратегія дозволяє створити зручний та швидкий користувацький досвід для пацієнтів, які використовують чат-бот клініки для запису на прийом. Це дослідження також створює основу для подальшого розвитку додатків на базі Telegram, що може знайти застосування не лише в медичній сфері, а й у інших галузях, де необхідна швидка й стабільна взаємодія із користувачем.

5.5 Висновки до розділу

У цьому розділі було проведено дослідження особливостей рендерингу веб-додатків, інтегрованих у Telegram, з використанням підходів SSR та CSR.

Було поставлено мету та завдання, методику, вимоги та сутність експерименту.

Розглянуто такі ключові показники продуктивності: швидкість завантаження DOMContentLoaded та Load, перше відтворення вмісту (FCP), час до першого байта (TTFB), індекс швидкості (SI), найбільше відтворення вмісту (LCP), кумулятивне зміщення макета (CLS). Також було розглянуто оцінки продуктивності: Performance – Продуктивність, Accessibility – Доступність, Best Practice – Відповідність найкращим практикам, SEO – Доступність, Progressive Web Application (PWA) – Відповідність прогресивним веб-додаткам.

Вимірювання DOMContentLoaded та Load проводилось при різних мережевих умовах: затримок (No throttling), Fast 3G та Slow 3G. Вимірювання FCP,

TTFB, SI, LCP та CLS, а також оцінки продуктивності проводилось на десктопних та мобільних платформах.

На основі проведених тестів проаналізовано ефективність кожного з методів рендерингу в межах Telegram-бота для медичної клініки, визначено рекомендації щодо вибору оптимального підходу та наведена характеристика новизни результатів. Результати дослідження можуть бути використані для покращення користувацького досвіду у подібних додатках та створення ефективних інтеграцій у Telegram.

ВИСНОВКИ

Кваліфікаційна робота є завершеною науковою роботою, в якій вирішена науково-практична задача розробки чат-бота в Telegram для запису пацієнтів до медичної клініки “Amel Dental Clinic” з урахуванням методів клієнтського (CSR) і серверного (SSR) рендерингу для вбудованого у бот веб-додатка. Основні висновки і результати роботи полягають у наступному:

1. Проведено дослідження існуючих методів розробки чат-ботів та інструментів інтеграції веб-додатків з платформою Telegram, що дало змогу обрати найбільш ефективний підхід для створення системи, яка відповідає вимогам медичної сфери, забезпечує швидкий і надійний запис пацієнтів на прийом, та обробляє дані у реальному часі.

2. Створено та реалізовано архітектуру веб-додатка з клієнтським і серверним рендерингом (React для CSR та Next.js для SSR). Проведено порівняння ефективності роботи обох методів рендерингу, що дозволило оцінити їх вплив на швидкодію, навантаження на сервер, а також на якість обслуговування користувачів в умовах обмеженого ресурсу мобільних пристроїв.

3. Реалізовано API на основі Express.js, що забезпечує безпечну та швидку взаємодію з базою даних MongoDB для збереження й обробки записів пацієнтів та даних лікарів. Завдяки MongoDB досягнуто масштабованості системи та можливості зберігати різнорідні структури даних, необхідні для повноцінного функціонування медичного сервісу.

4. У процесі розробки Telegram-бота реалізовано функціонал перегляду інформації про лікарів, запису пацієнтів на прийом, скасування записів, відправки сповіщень лікарям та пацієнтам, а також підтримка зворотного зв'язку з користувачами. Цей функціонал забезпечує високий рівень автоматизації процесу запису, полегшуючи роботу медичного персоналу та підвищуючи зручність для пацієнтів.

5. Застосовано методи асинхронного обміну даними за допомогою Node.js та бібліотек для роботи з Telegram API.

6. Розроблено механізм валідації даних пацієнтів за допомогою бібліотеки Joi, що забезпечує цілісність та точність даних перед їх збереженням у базі даних. Це дозволяє мінімізувати помилки у процесі взаємодії пацієнтів з системою запису.

7. Для аналізу ефективності клієнтського і серверного рендерингу використано такі інструменти для дослідження, як Chrome DevTools і Lighthouse, які допомогли оцінити продуктивність веб-додатка (First Contentful Paint, Time to First Byte, Speed Index, Largest Contentful Paint, Cumulative Layout Shift), швидкість завантаження (DOMContentLoaded, Load).

8. Під час досліджень виконано порівняння SSR і CSR для різних сценаріїв використання (різні мережеві умови та пристрої). Зібрані дані демонструють, що для додатків, в яких потрібна швидка взаємодія з користувачем і мінімізація затримок при обробці запитів, серверний рендеринг є більш доцільним. Водночас CSR є більш придатним для веб-додатків, орієнтованих на індивідуальну взаємодію без інтенсивної серверної обробки.

Практична цінність отриманих результатів полягає в тому, що розроблений Telegram-бот із веб-додатком для запису пацієнтів може використовуватись у медичних закладах для спрощення процесу організації роботи. Використання SSR і CSR також є цінним інструментом для подальшого розвитку таких рішень, дозволяючи адаптувати систему під потреби конкретних закладів та користувачів.

ПЕРЕЛІК ПОСИЛАНЬ

1. Цвіркун Л.І. Практична підготовка. Методичні рекомендації до виконання виробничої та передатестаційної практик магістрами галузі знань 12 Інформаційні технології спеціальності 123 Комп'ютерна інженерія / Л.І. Цвіркун, Д.О. Бешта, С.М. Ткаченко, В.В. Гнатушенко ; М-во освіти і науки України, Нац. техн. ун-т «Дніпровська політехніка». – Дніпро: НТУ «ДП», 2022. – 21 с

2. Цвіркун Л.І. Використання мессенджерів як системи оповіщення користувачів локальних систем домашнього інтернету речей. / Л.І. Цвіркун, Л.В. Бешта, Ю.А. Миронов // Системні технології. Зб. наук. пр. НМЕтАУ. − 2021. − № 4. − с. 95−101.

3. Цвіркун Л.І. Розробка програмного забезпечення комп'ютерних систем. Програмування: навч. посіб. [Електронний ресурс] / Л.І. Цвіркун, А.А. Євстігнєєва, Я.В. Панферова ; під заг. ред. проф. Л.І. Цвіркуна ; М-во освіти і науки України, Нац. техн. ун-т «Дніпровська політехніка». – 1 електрон. опт. диск (CD-ROM) ; 12 см. – Систем. вимоги (мінімальні): Процесор 32-розрядний (x86) 233 МГц ; 512 МБ RAM ; 128 МБ Video ; від 4-х до 48-х CD-ROM ; Windows 7. – Назва з контейнера. – Дніпро: НТУ «ДП», 2019. – ISBN 978-966-350-638–8.

4. Додавання кнопки "Меню" з доступними командами в Telegram-бот – Education [Електронний ресурс] – Режим доступу до ресурсу: https://stackoverflow.com/questions/72594564/how-can-i-add-menu-button-in-telegram-bot

5. Створення телеграм-бота з Mini Apps – Education [Електронний ресурс] – Режим доступу до ресурсу: https://youtu.be/MzO-0IYkZMU?si=pRIYtdUo6D4jhSN0

6. Backend на Node.js та Express.js – Education [Електронний ресурс] – Режим доступу до ресурсу: https://youtu.be/tKM44vPHU0U?si=gnPTYVxu4AQWQN5t

7. Створення телеграм-бота на Node.js та Express.js, використовуючи MongoDB – Education [Електронний ресурс] – Режим доступу до ресурсу: https://youtu.be/mDgKjb5eWPk?si=qWaC68R0SPTZWngB

8. Next.js – Education [Електронний ресурс] – Режим доступу до ресурсу: https://youtu.be/Onn38VeEAC8?si=uXncTlBqadlqsfLj

9. Rest API – Education [Електронний ресурс] – Режим доступу до ресурсу: https://youtu.be/XaTwnKLQi4A?si=sEPsgQ_qhiPbuRAs

10. Документація React.js – Education [Електронний ресурс] – Режим доступу до ресурсу: https://react.dev/

11. Документація Telegram Bot API – Education [Електронний ресурс] – Режим доступу до ресурсу: https://core.telegram.org/bots/api#message

12. Рендеринг: різновиди, застосування, порівняльна характеристика – Education [Електронний ресурс] – Режим доступу до ресурсу: http://surl.li/wtbcqb

13. Lighthouse – Education [Електронний ресурс] – Режим доступу до ресурсу: https://developer.chrome.com/docs/lighthouse/overview?hl=ru

14. Різниця між DOMContentLoaded і Load [Електронний ресурс] – Режим доступу до ресурсу: https://www.geeksforgeeks.org/difference-between-domcontentloaded-and-load-events/

15. Chrome DevTools [Електронний ресурс] – Режим доступу до ресурсу: https://developer.chrome.com/docs/devtools/overview?hl=ru

16. Односторінкові (spa) і багатосторінкові (pwa) веб-додатки [Електронний ресурс] – Режим доступу до ресурсу: https://embo.com.ua/uk/blog/odnostorinkovi-spa-i-bagatostorinkovi-pwa/

17. Long Polling, Webhooks [Електронний ресурс] – Режим доступу до ресурсу: https://grammy.dev/guide/deployment-types

Додаток А

Текст клієнтської частини програми системи управління медичними записами для клініки «Amel Dental Clinic»

Міністерство освіти і науки України
НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ
“ДНІПРОВСЬКА ПОЛІТЕХНІКА”

ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ КЛІЄНТСЬКОЇ ЧАСТИНИ СИСТЕМИ УПРАВЛІННЯ МЕДИЧНИМИ ЗАПИСАМИ ДЛЯ КЛІНІКИ «AMEL DENTAL CLINIC»

Текст програми
804.02070743.24018-01 12 01
Листів 14

2024

АНОТАЦІЯ

Цей додаток містить програмний код, що реалізує клієнтську частини систему управління медичними записами для клініки «Amel Dental Clinic» через Telegram бот, реалізовану за допомогою React.js для клієнтського рендерингу (CSR) та Next.js для серверного рендерингу (SSR).

ЗМІСТ

1. index.jsx

```
import React, { useEffect, useState } from 'react';
import ReactDOM from 'react-dom/client';
import App from './App';
import reportWebVitals from './reportWebVitals';
import { UserIdContext } from './context/context';

const root = ReactDOM.createRoot(document.getElementById('root'));
root.render(
  <React.StrictMode>
    <Main />
  </React.StrictMode>
);

function Main() {
  const [userId, setUserId] = useState(null);

  useEffect(() => {
    const telegramParams = sessionStorage.getItem('__telegram__initParams');

    if (telegramParams) {
      const params = JSON.parse(telegramParams);
      const id = params.user_id;

      if (id) {
        localStorage.setItem('userId', id);
        setUserId(id);
      }
    } else {
      const storedId = localStorage.getItem('userId');
```

```
      if (storedId) {
        setUserId(storedId);
      }
    }
  }, []);

  return (
    <UserIdContext.Provider value={userId}>
      <App />
    </UserIdContext.Provider>
  );
}

reportWebVitals();
```

2. App.jsx

```
// Імпорт необхідних модулів і бібліотек
import { HashRouter, Navigate, Route, Routes } from "react-router-dom";
import Header from './components/Header/Header';
import './App.scss';
import s from './App.module.scss';
import { lazy, Suspense, useState, useEffect, useContext } from "react";
import Preloader from "./components/Preloader/Preloader";
import Footer from "./components/Footer/Footer";
import { UserIdContext } from "./context/context";

// Лазливо імпортуються компоненти для оптимізації завантаження
const Doctors = lazy(() => import('./components/Doctors/Doctors/Doctors'));
const DoctorDetails = lazy(() =>
import('./components/Doctors/DoctorDetails/DoctorDetails'));
```

```
const Registration = lazy(() => import('./components/Registration/Registration'));
const Records = lazy(() => import('./components/Records/Records'));
const Error = lazy(() => import('./components/Error/Error'));

const App = () => {
  // Стан для відкриття/закриття меню
  const [isMenuOpen, setIsMenuOpen] = useState(false);

  // Отримується userId з контексту
  const userId = useContext(UserIdContext);

  // Використовується для блокування прокрутки при відкритті меню
  useEffect(() => {
    if (isMenuOpen) {
      // Додається клас блокування прокрутки
      document.body.classList.add('lock');
    } else {
      // Видаляється клас при закритті меню
      document.body.classList.remove('lock');
    }
  }, [isMenuOpen]);

  // Використовується для зміни класу body в залежності від наявності userId
  useEffect(() => {
    if (!userId) {
      // Додається клас для браузера, якщо userId немає
      document.body.classList.add('browser');
    } else {
      // Видаляється клас, якщо userId є
      document.body.classList.remove('browser');
```

```
    }
  }, [userId]);

  return (
    <HashRouter>
      <div className={s.wrapper}>
        <Header isMenuOpen={isMenuOpen}
setIsMenuOpen={setIsMenuOpen}/>
        <main className={s.main}>
          <div className={s.container}>
            {/* Відображається прелоадер під час завантаження компонентів */}
            <Suspense fallback={<Preloader />}>
              {/* Визначаються маршрути для навігації */}
              <Routes>
                <Route path="/" element={<Navigate to={'/doctors'} />} />
                <Route path="/doctors" element={<Doctors/>} />
                <Route path="/doctors/:doctorId" element={<DoctorDetails />} />
                <Route path="/registration" element={<Registration/>} />
                <Route path="/records" element={<Records/>} />
                <Route path="*" element={<Error />} />
              </Routes>
            </Suspense>
          </div>
        </main>
        <Footer/>
      </div>
    </HashRouter>
  )
}
```

export default App; // Експортується компонент App

3. doctors-api.js

import { instance } from "./api"; // Імпортується налаштований екземпляр axios для API запитів

```
// Об'єкт doctorsAPI, що містить методи для роботи з лікарями
export const doctorsAPI = {
  // Метод для отримання списку всіх лікарів
  getAll() {
    return instance
      .get(`doctors`)
      .then(response => response.data);
  },
  // Метод для отримання інформації про одного лікаря за Telegram ID
  getOne(telegramId) {
    return instance
      .get(`doctor/${telegramId}`)
      .then(response => response.data);
  }
}
```

4. Preloader.jsx

```
// Імпорт необхідних модулів і бібліотек
import { createPortal } from 'react-dom';
import preloader from './../../images/preloader.gif';
import s from './Preloader.module.scss';

// Компонент Preloader, що показує індикатор завантаження
const Preloader = () => {
```

```
  return createPortal(
    // Створюється портал для індикатора завантаження
    <div className={s.preloader}>
      {/* Зображення індикатора завантаження */}
      <img src={preloader} alt="Loading..." />
    </div>,
    document.body,
  );
};

export default Preloader;
```

5. TimeSlotList.jsx

```
// Імпорт необхідних модулів і бібліотек
import { useContext, useState } from 'react';
import s from './TimeSlotList.module.scss';
import cn from 'classnames';
import { UserIdContext } from './../../../context/context';

// Компонент TimeSlotList для вибору доступних часових слотів
function TimeSlotList({ availableSlots, handleSelectSlot, selectedDate }) {
  const [activeSlot, setActiveSlot] = useState(null); // Стан для збереження активного слоту
  const userId = useContext(UserIdContext); // Отримання userId з контексту

  // Функція для обробки вибору часового слоту
  const onClickSlot = (slot) => {
    setActiveSlot(slot); // Оновлення активного слоту
    handleSelectSlot(slot); // Виклик функції для обробки вибору
  };
```

```
        // Класи для елемента списку часових слотів
        const itemClass = cn(s.time__item, {
          [s.browser]: !userId, // Додаток класу, якщо userId відсутній
        });

        return (
          <div className={s.time}>
            <h2 tabIndex={0} className={s.time__title}>Доступні часи прийому {selectedDate}</h2>
            <ul className={s.time__list}>
              {availableSlots.map(slot => (
                <li
                  key={slot.time}
                  onClick={() => onClickSlot(slot.time)} // Обробка кліку
                  onKeyDown={(e) => {
                    if (e.key === 'Enter' || e.key === ' ') {
                      e.preventDefault(); // Запобігання стандартної поведінки
                      onClickSlot(slot.time); // Обробка клавіатурного вводу
                    }
                  }}
                  tabIndex={0}
                  role="button"
                  className={cn(itemClass, {
                    [s.time__item_active]: slot.time === activeSlot, // Додаток класу для активного слоту
                    [s.browser_active]: slot.time === activeSlot,
                  })}
                  aria-pressed={slot.time === activeSlot} // Додаткова інформація про активність
```

```
            aria-label={`Оберіть час ${slot.time}`} // Опис для асистивних
технологій
          >
            {slot.time}
          </li>
        ))}
      </ul>
    </div>
  );
}
export default TimeSlotList;
```

6. useTelegram.jsx

```
import { useCallback } from 'react';

// Отримується об'єкт Telegram WebApp
const tg = window.Telegram.WebApp;

export function useTelegram() {
  // Викликається, коли WebApp готовий до використання
  const onReady = useCallback(() => {
    tg.ready();
  }, []);

  // Функція для відображення основної кнопки
  const showMainButton = useCallback(() => {
    tg.MainButton.show();
  }, []);

  // Функція для приховання основної кнопки
```

```
const hideMainButton = useCallback(() => {
  tg.MainButton.hide();
}, []);
// Функція для встановлення тексту основної кнопки
const setButtonText = useCallback((buttonText) => {
  tg.MainButton.setParams({ text: buttonText });
}, []);

// Функція для додавання обробника події натискання на основну кнопку
const setEventMainButtonClicked = useCallback((callback) => {
  tg.onEvent('mainButtonClicked', callback);
}, []);

// Функція для видалення обробника події натискання на основну кнопку
const removeEventMainButtonClicked = useCallback((callback) => {
  tg.offEvent('mainButtonClicked', callback);
}, []);

// Функція для відправки даних до Telegram
const sendDataToTelegram = useCallback((data) => {
  tg.sendData(JSON.stringify(data));
}, []);

// Перевіряється, чи активна темна тема
const isDarkTheme = tg.colorScheme === 'dark';

// Повертається об'єкт з функціями та інформацією про тему
return {
  tg,
  onReady,
```

```
    showMainButton,
    hideMainButton,
    setButtonText,
    setEventMainButtonClicked,
    removeEventMainButtonClicked,
    sendDataToTelegram,
    isDarkTheme
  };
}
```

7. index.html

```
<!DOCTYPE html>
<html lang="en">

<head>
  <meta charset="utf-8" />
  <link rel="icon" href="%PUBLIC_URL%/favicon.ico" />
  <meta name="viewport" content="width=device-width, initial-scale=1" />
  <meta name="theme-color" content="#000000" />

  <!-- SEO мета теги -->
  <meta name="description"
    content="Amel Dental Clinic – зручний онлайн-запис і керування прийомом до стоматолога через Telegram WebApp" />
  <meta name="keywords"
    content="Amel Dental Clinic, стоматологічні послуги, онлайн-бронювання, Telegram WebApp, запис на прийом до стоматолога, охорона здоров'я" />

  <meta name="Author" content="Amel Dental Clinic">
  </meta>
```

```
    <meta name="Copyright" content="Amel Dental Clinic">
    </meta>
    <meta name="Address" content="м. Дніпро, бул. Слави, 2-Б, ж/м Перемога-5">
    </meta>

    <meta property="og:image" content="../src/images/doctors.jpg" />
    <meta property="og:type" content="website" />

    <meta name="format-detection" content="telephone=no">

    <!-- Telegram WebApp мета теги -->
    <meta name="telegram:webapp" content="Amel Dental Clinic WebApp для легкого керування записами." />
    <script src="https://telegram.org/js/telegram-web-app.js"></script>

    <meta name="robots" content="index, follow">
    </meta>

    <title>Amel Dental Clinic</title>
  </head>

  <body>
    <div id="root"></div>
  </body>

  </html>
```

Додаток Б

Текст серверної частини програми системи управління медичними записами для клініки «Amel Dental Clinic»

Міністерство освіти і науки України
НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ
“ДНІПРОВСЬКА ПОЛІТЕХНІКА”

ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ СЕРВЕРНОЇ ЧАСТИНИ СИСТЕМИ УПРАВЛІННЯ МЕДИЧНИМИ ЗАПИСАМИ ДЛЯ КЛІНІКИ «AMEL DENTAL CLINIC»

Текст програми
804.02070743.24018-01 12 01
Листів 11

2024

АНОТАЦІЯ

Цей додаток містить програмний код, що реалізує серверну частини систему управління медичними записами для клініки «Amel Dental Clinic» через Telegram бот. Чат-бот реалізовано за допомогою платформи Node.js, а API за допомогою фреймворку Express.js

ЗМІСТ

1. Чат-бот

1.1 index.js

```
require('dotenv').config();

const bot = require('./bot/bot');

const PORT = process.env.PORT || 3000;

const express = require('express');

const app = express();

// Обробляється помилка при опитуванні бота
bot.on('polling_error', console.log);

app.listen(PORT, () => {
  console.log(`Бот запущений на порту ${PORT}`);
});
```

1.2 api.js

```
// Імпорт необхідних модулів і бібліотек
const axios = require('axios');
const { API_URL } = require('../config/env');

// Функція отримує доступні дати для лікаря за userId
const fetchAvailableDates = (userId) =>
axios.get(`${API_URL}doctor/${userId}/available-dates`);

// Функція отримує записи за обрану дату для лікаря за userId
```

```
const fetchRecordsByDate = (userId, date) =>
axios.get(`${API_URL}doctor/${userId}/records/${date}`);

// Експортуються функції для використання в інших модулях
module.exports = { fetchAvailableDates, fetchRecordsByDate };
```

1.3 telegram.js

```
// Імпорт необхідних модулів і бібліотек
const { WEB_APP_URL } = require('../config/env');

// Функція отримує URL веб-додатку з параметром user_id
const getWebAppUrlWithUserId = (userId) =>
`${WEB_APP_URL}?user_id=${userId}`;

// Експортується функція для використання в інших модулях
module.exports = { getWebAppUrlWithUserId };
```

1.4 callbackQueryHandler.js

```
// Імпорт необхідних модулів і бібліотек
const { fetchRecordsByDate } = require('../../utils/api');
const { ERROR_MESSAGE,
DOCTOR_DATE_SCHEDULE_MESSAGE_BODY,
DOCTOR_DATE_SCHEDULE_MESSAGE_TITLE } =
require('../../config/constants');

// Обробка запитів з клавіатури
const handleCallbackQuery = async (bot, callbackQuery) => {
  const chatId = callbackQuery.message.chat.id; // Отримання ID чату
  const selectedDate = callbackQuery.data; // Отримання вибраної дати
  const userId = callbackQuery.from.id; // Отримання ID користувача
```

```
    try {
      // Отримання записів за вибраною датою
      const recordsResponse = await fetchRecordsByDate(userId, selectedDate);
      const records = recordsResponse.data; // Збереження отриманих записів

      // Формування повідомлення з заголовком
      let message =
DOCTOR_DATE_SCHEDULE_MESSAGE_TITLE(selectedDate);

      // Додавання записів до повідомлення
      records.forEach(record => {
        message += DOCTOR_DATE_SCHEDULE_MESSAGE_BODY(record);
      });

      // Відправлення повідомлення з розкладом
      await bot.sendMessage(chatId, message);
    } catch (error) {
      // Відправлення повідомлення про помилку
      await bot.sendMessage(chatId, ERROR_MESSAGE);
    }
  };

  // Експорт функції для використання в інших модулях
  module.exports = { handleCallbackQuery };
```

1.5 bot.js

```
// Імпорт необхідних модулів і бібліотек
const TelegramBot = require('node-telegram-bot-api');
const { TELEGRAM_BOT_TOKEN } = require('../config/env');
```

```
const { handleMessage } = require('./handlers/messageHandler');
const { handleCallbackQuery } = require('./handlers/callbackQueryHandler');

// Ініціалізація бота з токеном і режимом опитування
const bot = new TelegramBot(TELEGRAM_BOT_TOKEN, { polling: true });

// Обробка вхідних повідомлень
bot.on('message', (msg) => handleMessage(bot, msg));
// Обробка запитів з клавіатури
bot.on('callback_query', (callbackQuery) => handleCallbackQuery(bot, callbackQuery));

// Експорт об'єкту бота для використання в інших модулях
module.exports = bot;
```

2. API

2.1 index.js

```
// Підключаємо змінні середовища з файлу .env
require('dotenv').config();

// Імпортуємо необхідні модулі
const express = require("express");
const cors = require("cors");
const bodyParser = require("body-parser");
const database = require('./config/database');
const doctorRoutes = require('./routes/doctorRoutes');
const doctorsRoutes = require('./routes/doctorsRoutes');
const recordRoutes = require('./routes/recordRoutes');

// Ініціалізуємо Express-додаток
```

```
const app = express();
// Встановлюємо порт для сервера
const PORT = process.env.PORT || 3000;

// Визначаємо дозволені домени для CORS
const allowedOrigins = ['https://dmytrosokolovsky.github.io', 'https://next-diplom.vercel.app'];

// Налаштовуємо CORS
app.use(cors({
  origin: function (origin, callback) {
    if (!origin || allowedOrigins.indexOf(origin) !== -1) {
      callback(null, true);
    } else {
      callback(new Error('Не дозволено CORS'));
    }
  }
}));

// Парсимо JSON у запитах
app.use(bodyParser.json());

// Налаштовуємо маршрути
app.use('/api/doctor', doctorRoutes);
app.use('/api/doctors', doctorsRoutes);
app.use('/api/records', recordRoutes);

// Запускаємо сервер і підключаємося до бази даних
app.listen(PORT, () => {
  database.connectToDatabase()
```

```
    .then(() => console.log(`Server running on port ${PORT}`))
    .catch(err => console.error('Помилка підключення до бази даних:', err));
});
```

2.2 doctorRoutes.js

```
// Імпортуємо модуль Express та функції контролера лікаря
const express = require('express');
const { getDoctorByTelegramId, getDoctorSchedule, getDoctorAvailableDates, getDoctorRecordsDate } = require('../controllers/doctorController');

// Створюємо новий роутер
const router = express.Router();

// Створюємо маршрути
router.get('/:telegram_id', getDoctorByTelegramId);
router.get('/:id/schedule', getDoctorSchedule);
router.get('/:id/available-dates', getDoctorAvailableDates);
router.get('/:id/records/:date', getDoctorRecordsDate);

// Експортуємо роутер
module.exports = router;
```

2.3 recordModel.js

```
// Імпортуємо бібліотеку Joi для валідації даних
const Joi = require('joi');

// Оголошуємо схему валідації для запису на прийом
const recordSchema = Joi.object({
  doctor: Joi.string().required(),
  doctor_id: Joi.number().integer().required(),
```

```
  specialization: Joi.string().required(),
  date: Joi.string().required(),
  time: Joi.string().required(),
  user_id: Joi.alternatives().try(Joi.number().integer(), Joi.allow(null)).required(),
  patient_name: Joi.string().required(),
  patient_phone_number: Joi.number().integer().required(),
});

// Експортуємо схему валідації
module.exports = { recordSchema };
```

2.4 doctorsController.js

```
// Імпортуємо модуль підключення до бази даних
const database = require('../config/database');

// Отримуємо список всіх лікарів
const getDoctors = async (req, res) => {
  try {
    const doctors = await
database.getDatabase().collection('doctors').find({}).toArray();
    res.status(200).json(doctors);
  } catch (error) {
    console.error('Помилка при отриманні лікарів:', error);
    res.status(500).json({ message: 'Помилка сервера' });
  }
};

// Отримуємо лікарів за спеціалізацією
const getDoctorsBySpecialization = async (req, res) => {
  try {
```

```
        const specialization = req.params.specialization;
        const doctors = await database.getDatabase().collection('doctors').find({ specialization }).toArray();
        res.json(doctors);
      } catch (error) {
        console.error("Помилка при отриманні лікарів:", error);
        res.status(500).json({ message: 'Помилка сервера при отриманні лікарів' });
      }
    };

    // Отримуємо список унікальних спеціалізацій лікарів
    const getSpecializations = async (req, res) => {
      try {
        const specializations = await database.getDatabase().collection('doctors').distinct("specialization");
        res.json(specializations);
      } catch (error) {
        console.error("Помилка при отриманні спеціалізацій:", error);
        res.status(500).json({ message: 'Помилка сервера під час отримання спеціалізацій' });
      }
    };

    // Експортуємо функції
    module.exports = { getDoctors, getDoctorsBySpecialization, getSpecializations };
```