Міністерство освіти і науки України
Харківський національний університет імені В. Н. Каразіна
Факультет комп'ютерних наук
Кафедра теоретичної та прикладної системотехніки

«Затверджую»
Зав. кафедри теоретичної та
прикладної системотехніки
_____ д.т.н., проф. С. І. Шматков
«___» _____ 2024 р

Пояснювальна записка

до кваліфікаційної роботи
бакалавра

на тему: «WEB-система онлайн-поліклініки»

Захищено на засіданні
Атестаційної комісії № 44
протокол № __ від __.06.2024 р.
Оцінка _____ / _____
Голова Атестаційної комісії
_____ Скоб Ю. О.
(підпис) (прізвище та ініціали)

Виконала:
студентка 4 курсу, групи КУ– 41
Галузь знань: 15 – Автоматизація та приладобудування
Спеціальність: 151 – «Автоматизація та комп'ютерно-інтегровані технології»
Лофіцька Анастасія Олексіївна

(підпис)

Керівник:
Старший викладач ЗВО кафедри ТПС
Павлов Анатолій Миколайович

Рецензент:
доцент ЗВО кафедри БІСТ, канд. техн. наук
Колованова Євгенія Павлівна

_____
(підпис)

Харків – 2024

АНОТАЦІЯ

Пояснювальна записка до кваліфікаційної роботи бакалавра складається зі вступу, трьох розділів, висновків, списку використаних джерел і чотирьох додатків. Загальний обсяг роботи складає 79 сторінок, із яких 52 сторінки основної частини з 31 рисунком, 6 таблицями, 16 найменуваннями списку використаних джерел та чотирма додатками.

Метою кваліфікаційної роботи є розробка клієнтської і серверної частин інформаційної системи реєстратури.

Об'єкт дослідження – процес електронного запису людей на прийом до лікаря та опрацювання заявок.

Предмет дослідження – інформаційні технології і програмні методи створення клієнтської і серверної частин WEB-системи, що дозволяє автоматизувати запис на прийом до лікаря.

Завдання, яке вирішується в кваліфікаційній роботі полягає в тому, щоб розробити онлайн-систему за допомогою технології ASP.NET MVC і мови програмування C#, та бази даних MySQL. Ця система спрямована на зменшення обсягу роботи для лікарів та пацієнтів, а також на автоматизацію та систематизацію роботи поліклініки.

Область застосування – розробка веб-додатків, спрямована на створення програмного продукту з широким спектром застосування як у повсякденному житті, так і в медичній сфері.

Ключові слова: WEB-система, клієнтська частина, серверна частина, технологія ASP.NET MVC, мова програмування C #, MySQL.

ABSTRACT

The explanatory note to the bachelor's thesis consists of an introduction, three chapters, conclusions, a list of references and four appendices. The total volume of the work is 79 pages, including 52 pages of the main part with 31 figures, 6 tables, 16 references and four appendices.

The purpose of the qualification work is to develop the client and server parts of the registry information system.

The object of research is the process of electronic registration of people for an appointment with a doctor and processing of applications.

The subject of research is information technologies and software methods for creating client and server parts of the WEB-system, which allows to automate the appointment with a doctor.

The task that is solved in the qualification work is to develop an online system using ASP.NET MVC technology and the C# programming language, and the MySQL database. This system is aimed at reducing the amount of work for doctors and patients, as well as automating and systematizing the work of the clinic.

The field of application is web application development aimed at creating a software product with a wide range of applications both in everyday life and in the medical field.

Keywords: WEB-system, client part, server part, ASP.NET MVC technology, C# programming language, MySQL.

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ І УМОВНИХ ПОЗНАЧЕНЬ

WBIS – (англ. Web-Based Information System) інформаційна система на основі веб-технологій;

SEO – (англ. Search Engine Optimization) оптимізація пошукових систем;

SSR – (англ. Server-Side Rendering) рендеринг на стороні сервера;

API – (англ. Application Programming Interface) інтерфейс прикладного програмування;

HTML – (англ. HyperText Markup Language) мова розмітки гіпертексту;

JS – JavaScript;

CSS – (англ. Cascading Style Sheets) каскадні таблиці стилів;

CDN – (англ. Content Delivery Network) мережа доставки контенту;

SSG – (англ. Static Site Generator) генератор статичних сайтів;

SPA – (англ. Single-Page Application) односторінковий додаток;

PWA – (англ. Progressive Web Application) прогресивний веб-додаток;

HTTPS – (англ. HyperText Transfer Protocol Secure) протокол безпечної передачі гіпертексту;

UI/UX – User Interface/User Experience;

PHP – (англ. Hypertext Preprocessor) препроцесор гіпертексту;

AWS – (англ. Amazon Web Services) веб-служби Amazon;

ASE – (англ. Advanced Encryption Standard) стандарт шифрування даних;

CMS – (англ. Content Management System) система керування контентом;

IDE – (англ. Integrated Development Environment) інтегроване середовище розробки;

SQL – (англ. Structured Query Language) мова структурованих запитів;

EF Core – Entity Framework Core;

UML – (англ. Unified Modeling Language) уніфікована мова моделювання.

ВСТУП

Наразі медичні установи намагаються покращити сервіс для своїх пацієнтів, приступаючи до впровадження нових інноваційних рішень.

Одним із ключових напрямків у цьому процесі є впровадження веб-систем онлайн-поліклініки, що дозволяють зручно та ефективно записуватися на прийом до лікарів через веб-додаток. Ця функція, інтегрована з інформаційною системою закладу, пропонує значні переваги як для пацієнтів, так і для медичного персоналу.

Це забезпечує зручність та доступність запису на прийом з будь-якого пристрою, що підключений до Інтернету, в той час як медичний заклад отримує можливість більш ефективно розподіляти навантаження та використовувати ресурси.

Підкреслюючи важливість цієї технології, кваліфікаційна робота спрямовується на ретельне вивчення процесу впровадження та використання веб-систем онлайн-поліклініки з метою ефективного впливу на якість медичного обслуговування та задоволення потреб пацієнтів.

Актуальність теми. Тема "WEB-система онлайн-поліклініки" є актуальною у зв'язку з постійним прагненням покращити доступність медичної допомоги та зробити процес отримання медичних послуг більш зручним для пацієнтів.

Її впровадження може вирішити низку проблем, зокрема, забезпечити можливість отримання медичних консультацій для тих, хто знаходиться віддалено від медичних закладів або має обмежені можливості особистого відвідування лікаря.

Крім того, система онлайн-поліклініки допоможе зменшити черги та час очікування на прийом, оскільки пацієнти зможуть записатися на прийом через Інтернет і навіть отримувати консультації дистанційно.

Автоматизація процесів, пов'язаних з записом на прийом та обліком медичних даних, покращить роботу медичного персоналу та забезпечить безпеку та конфіденційність медичної інформації.

Метою дослідження є розробка та реалізаія web-системи онлайн-поліклініки, яка надасть зручний та ефективний інструмент для запису на прийоми, збереження та керування медичною інформацією пацієнтів, а також спростить взаємодію між лікарями та пацієнтами.

Об'єкт дослідження – процес електронного запису людей на прийом до лікаря та опрацювання заявок.

Предмет дослідження – інформаційні технології і програмні методи створення клієнтської і серверної частин WEB-системи, що дозволяє автоматизувати запис на прийом до лікаря.

Методи дослідження: системний аналіз, проєктування та моделювання, статистичний аналіз, тестування та валідація.

Завдання дослідження

1. Створити макети інтерфейсів для основних користувачів системи: пацієнтів, лікарів та адміністраторів.
2. Реалізувати інтерфейси з урахуванням принципів зручності та доступності.
3. Провести тестування з участю реальних користувачів для оцінки зручності та функціональності системи.

РОЗДІЛ 1
АНАЛІЗ МЕТОДІВ ПОБУДОВИ WEB-СИСТЕМ ТА ТИПІВ ВЕБ-САЙТІВ

1.1 Форми веб-сайтів

Форми веб-сайтів можуть варіюватися в залежності від їхніх цілей, призначення та специфіки контенту. Розглянемо деякі загальні форми веб-сайтів:

1. Сайт-візитка. Основне завдання платформи в короткій формі ознайомити з компанією або людиною, що надає товари або послуги, а також подати дані про способи зв'язку (адреса, телефон). Сайт - візитка дає можливість при мінімальних фінансових і тимчасових витратах надати потенційним клієнтам всю необхідну інформацію в короткому або тезовому виді. Раніше такий сайт займав не більше однієї сторінки, де була викладена вся необхідна інформація, практично так, як на звичайній паперовій візитці. Але з часом такі міні-сайти стали розробляти в обсязі 1-5 сторінок, де кожна присвячується окремому питанню. Наприклад, на першій і головній сторінці вказується назва організації, логотип і, наприклад, види діяльності. На наступних сторінках описуються послуги і переваги, і окрема сторінка присвячується зв'язку. Тобто вказуються всі способи, якими можна зв'язатися з представниками компанії або перейти на офіційний сайт для більш детального ознайомлення.
2. Корпоративний сайт або презентаційний сайт - це офіційний представник компанії, який може виступати онлайн-офісом, магазином або просто офіційним сайтом, що надає всю необхідну інформацію про послуги. Зазвичай такі сайти розробляються в індивідуальному порядку і мають багато сторінок. Корпоративний ресурс включає в себе опис діяльності компанії, реклама наданих товарів або послуг, презентації або каталоги, контактні дані, форум або блог в форматі «питання - відповідь», а також відгуки покупців.

Корпоративний сайт дозволяє видавати онлайн розрахунки вартості, здійснювати продаж продукції, проводити маркетингові дослідження та рекламні акції, розсилати відвідувачам новини, збирати відгуки клієнтів, влаштовувати голосування і багато іншого;

3. Презентаційний сайт і сайт-візитка - найпростіші варіанти корпоративного представництва. Цілі - дати більш детальну інформацію про фірму і її найбільш цікаві пропозиції, а також відповідати на питання аудиторії і знімати зайве навантаження з традиційних каналів зв'язку;
4. Промо-сайт. Це інтернет-ресурс, спрямований на рекламу певного товару, послуги, бренду або події. Промо-сайт найчастіше запускається паралельно з рекламною кампанією, жорстко прив'язані до неї і є джерелом інформаційної підтримки рекламної кампанії. Промо-сайти крім текстового наповнення, містять велику кількість інтерактивних презентацій та демо-роликів, які демонструють продукт і його переваги;
5. Сайт-вітрина. Призначений для презентації товарів в режимі онлайн в зручному форматі. При наявності великого асортименту продукції електронний каталог дозволяє створити необхідні рубрики, швидко виконувати пошук необхідного товару за допомогою фільтрів і інших корисних функцій. За структурою сайт-вітрина нагадує інтернет-магазин, але в ньому відсутня опція покупки. Інтернет-вітрина - ефективний засіб реклами, збору заявок на продукцію і проведення маркетингових опитувань, підтримки зворотного зв'язку зі споживачами[1];
6. Сайт інтернет-магазину. Це сайт, призначений для продажу товарів, вибір яких здійснюється з каталогу. Інтернет-магазин може бути як спеціалізованим (наприклад, продавати тільки конкретний тип побутової техніки), так і універсальним. Найголовніше в онлайн - магазині - це простий доступний і легкий інтерфейс. Далі, важливо -

це фотографія та опис. Чим краще презентація та каталог, тим більше покупців. Доставка і оплата - це один з найважливіших аспектів інтернет - магазину. Передоплата зазвичай не радує покупця. Найкраще надати кілька способів оплати і доставки (кур'єром, службою перевезення, поштою, самовивезення).

Таким чином, в ході аналізу видів форм веб-сайту та їх відмінностей, можемо зробити висновок, що для створення моделі інформаційно сервісної служби необхідно розглянути методи розробки Web-сайтів.

1.2. Архітектура веб-додатків

Спочатку давайте визначимо, що таке веб-додаток. Це клієнт-серверний додаток, де є браузер (клієнт) і веб-сервер. Логіка веб-додатку розподілена між сервером і клієнтом, є канал для обміну інформацією та сховище даних, розташоване локально або в хмарі.

Веб-додатки з'явилися як етап еволюції веб-сайтів і, дійсно, мають багато спільного. Фактори, які відрізняють веб-сайт від веб-додатку, - це інтерактивність, інтеграція та автентифікація[2].

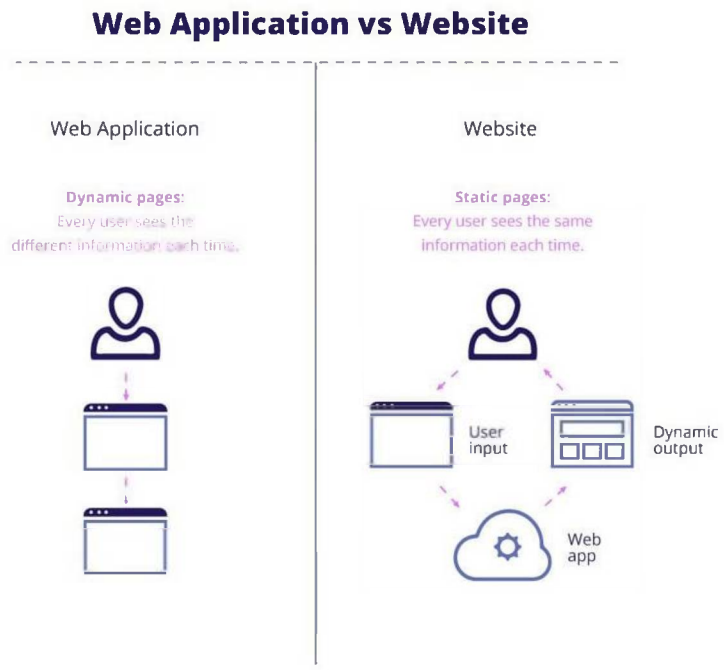


Рис. 1.3 - Веб-додаток проти веб-сайту

Іншими словами, чим більш кастомізованим, інтерактивним і функціональним є веб-сайт, тим ближче він до того, щоб називатися веб-додатком.

Сучасні веб-додатки все ще використовують концепцію 3-рівневої архітектури, яка розділяє додатки на рівень презентації, рівень додатку та рівень даних[2].

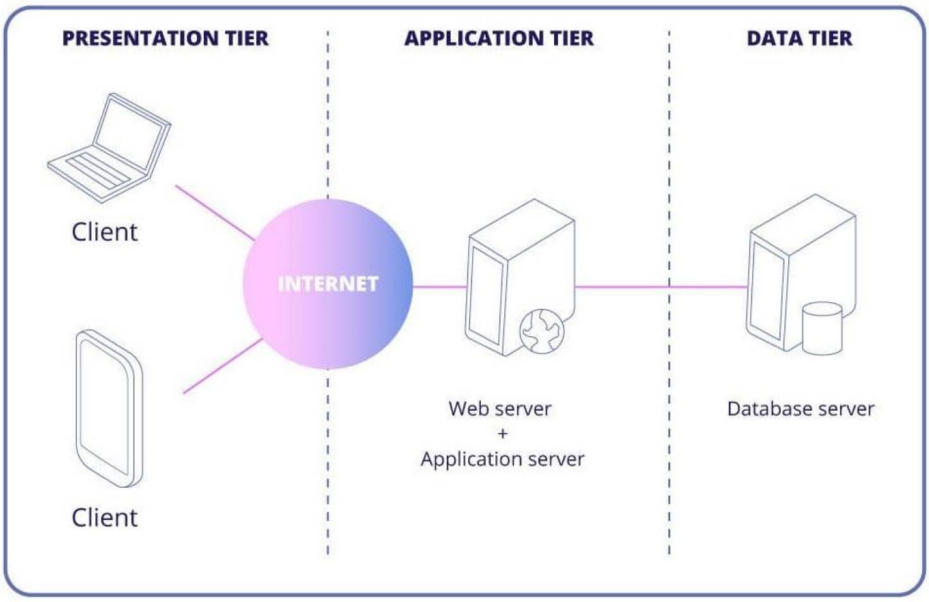


Рис. 1.4 – Три рівні архітектури

У 3-рівневій архітектурі веб-додатків кожен рівень працює на власній інфраструктурі і може розроблятися паралельно різними командами. Така структура дозволяє оновлювати та масштабувати кожен рівень за потреби, не впливаючи на інші рівні.

Потрібно розібратися детально, чи дійсно запропонований підхід до продукту відповідає потребам бізнесу. Оцінимо сучасні типи архітектури веб-додатків за критеріями, які вважаються найбільш важливими для бізнесу, а саме: продуктивність, користувацький інтерфейс, SEO, перелінковка та швидкість реалізації з боку розробників.

По-перше, розглянемо рендеринг на стороні сервера (SSR).

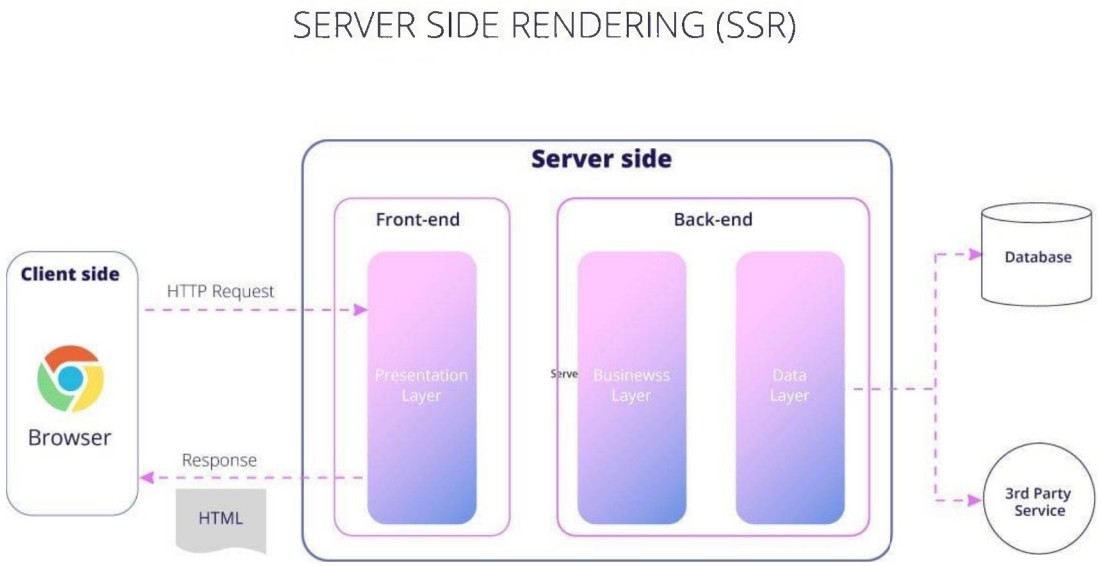


Рис. 1.5 - Рендеринг на стороні сервера (SSR).

Говорячи про основні веб-принципи, ми зазвичай маємо на увазі архітектуру клієнт-сервер. Клієнт запитує контент з сервера, де знаходиться бізнес-логіка та база даних. Використовуючи простий JavaScript, статична веб-сторінка надсилає запит до сервісу (можливо, API). Сервіс повертає дані і відображає клієнту HTML-сторінку.

Якщо ваш додаток рендериться на стороні сервера, вміст завантажується з сервера і передається в браузер для відображення користувачеві. Якщо HTML-сторінка рендериться на стороні сервера, користувач повинен перейти на сторінку до того, як браузер отримає її з сервера. Це означає, що для відображення вмісту користувачеві потрібно більше часу. Для кешування вмісту сторінки ця схема часто постачається з Nginx, веб-сервером, який також може використовуватися як поштовий проксі-сервер і балансувальник навантаження.

Також говорячи про плюси і мінуси. Той факт, що HTML рендериться на сервері, надає ряд переваг, таких як SEO, перелінковка і миттєве завантаження першої сторінки. Рендеринг на стороні сервера працює, коли в браузері відключений JS. Оскільки код обробляється на сервері, до браузера не пред'являється ніяких специфічних вимог, - це дозволяє нам миттєво

виявляти помилки. Однак SSR не може обробляти важкі запити до сервера (повторювані HTML, CSS), що призводить до повільного рендерингу при завантаженні сервера або повного перезавантаження сторінки. Але справжньою ахіллесовою п'ятою цього базового типу архітектури веб-додатків є погана взаємодія з кінцевим користувачем і неможливість створення повноцінного інтерфейсу користувача. Іншими словами, SSR - це простий і економічно вигідний спосіб, якщо вам потрібно створити простий веб-сайт. Реалізація цього типу архітектури можлива з будь-якою мовою програмування та бекендом.

Процес створення статичного сайту передбачає використання генератора, який автоматизує кодування окремих HTML-сторінок, створюючи їх на основі шаблонів. Обираючи послугу Static Site Generation, ви отримуєте простий статичний веб-сайт, розміщений на CDN або будь-якому іншому сервері, який містить вже згенеровану HTML-сторінку, що надається користувачам за запитом. Таким чином, не потрібно генерувати її щоразу, коли хтось відвідує ваш сайт - сервер просто надсилає вже існуючі дані через API.

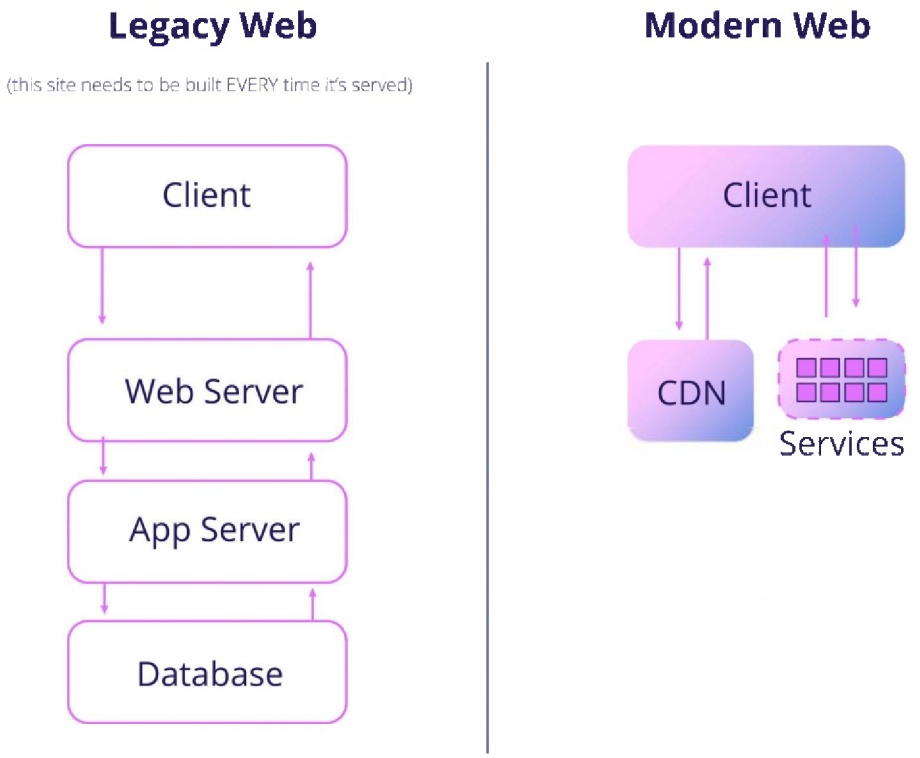


Рис. 1.6 - Застарілий веб проти сучасного

Перш за все, цей підхід підходить лише для веб-сайтів. При цьому вміст сторінок згенерованого веб-сайту не змінюється, якщо ви не додасте нові дані або компоненти. Це означає, що вам доведеться повністю перегенерувати веб-сайт, якщо ви захочете додати новий контент. Це один з основних недоліків, який серйозно обмежує бізнес-кейси, до яких він може бути застосований.

Серед переваг, однак, є висока швидкість статичного контенту, який доставляється через CDN. Крім того, в SSG всі серверні операції та робота з базою даних реалізуються через API, який не залежить від веб-сайту. Цей варіант є простим і тому виключно доступним для реалізації.

Прикладами простих генераторів статичних сайтів є Jekyll і Hugo, тоді як Gatsby і VuePress підходять для реалізації більш складних рішень.

Ще існують односторінкові додатки (SPA). SPA - це тип веб-додатків, які працюють у браузері. Він не вимагає перезавантаження сторінки, коли потрібно відобразити нові дані. Цей тип архітектури веб-додатків широко

використовується в нашому повсякденному житті: Facebook, Gmail, Google Maps, GitHub і Twitter - всі вони є односторінковими додатками.

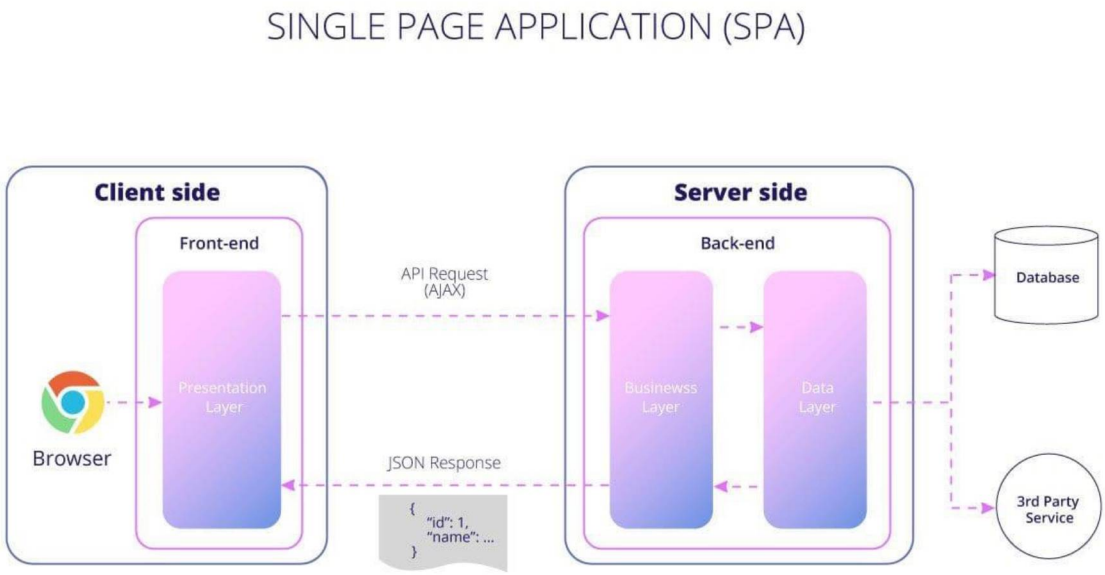


Рис. 1.7 – Односторінкові додатки

На відміну від SSR і SSG, SPA дозволяє створити інтерактивний веб-додаток. Він використовує API для зв'язку з сервером. Така архітектура добре підходить для легкого масштабування вашого продукту. Крім того, якщо вам потрібен мобільний додаток, не потрібно докладати додаткових зусиль для розробки API - мобільний додаток може використовувати той самий API, що і веб-додаток.

SPA забезпечує швидкий рендеринг після повного завантаження додатку в браузері і створює високочутливе програмне забезпечення для кінцевого користувача. Водночас це "вбиває" ваше SEO та обмежує кількість посилань, оскільки реалізація такого функціоналу потребуватиме додаткових зусиль. Серед інших недоліків - тривалий час, необхідний для першого завантаження, погана маршрутизація та обмежена підтримка застарілих браузерів. Будучи досить дорогим типом веб-архітектури, SPA підходить для створення адаптивного інтерфейсу для B2C-користувачів.

Останні кілька років всі говорять про розробку прогресивного веб-додатку (PWA).

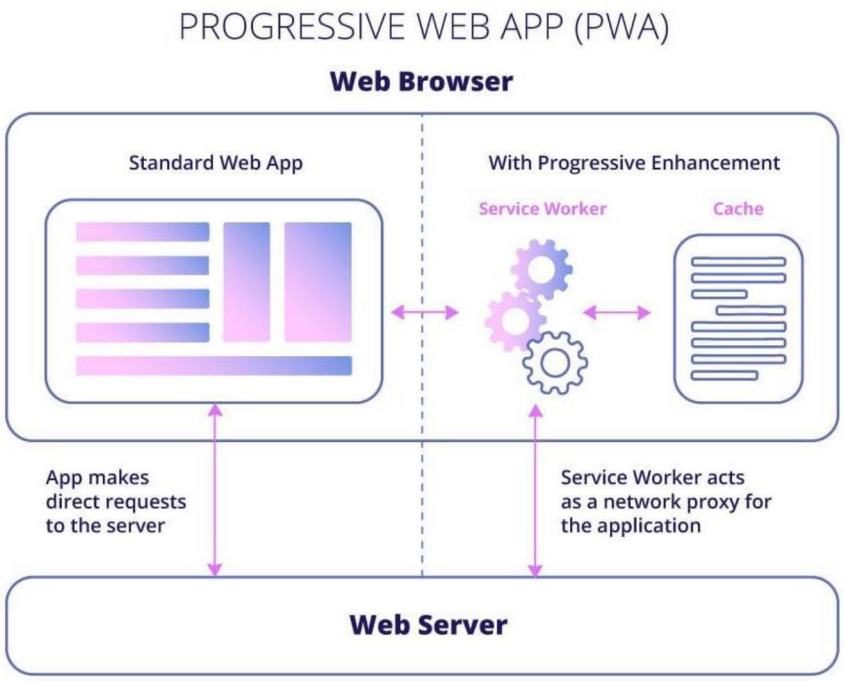


Рис. 1.8 – прогресивний веб-додаток (PWA)

Прогресивна архітектура веб-додатків використовує логіку односторінкового веб-додатку з деякими сервісами, що запускаються далі, у браузері. Це означає, що основний момент, який слід враховувати, полягає в тому, що і браузер, і операційна система повинні підтримувати цей набір стандартів.

Для кінцевого користувача прогресивний веб-додаток фізично означає спливаюче вікно з пропозицією додати додаток на стартовому екрані (не в браузері, а на екрані операційної системи), коли він відвідує веб-сайт. Якщо користувач погоджується, додаток автоматично додається на пристрій.

Реалізація PWA дозволяє вашому веб-додатку підтримувати роботу в автономному режимі, фонову синхронізацію та push-сповіщення. Це відкриває доступ до функціональності, яка раніше вимагала нативного додатку. Водночас, обираючи PWA-архітектуру для свого проекту, потрібно пам'ятати,

що більшість функцій недоступні на iOS. Варто проаналізувати кожен конкретний бізнес-кейс.

Прогресивна архітектура веб-додатків підтримується Windows, Android та iOS (для iOS, щоправда, відключений офлайн-режим). Розробники можуть віддалено додавати оновлення до веб-додатку. PWA є безпечним, оскільки використовує HTTPS. Водночас кінцеві користувачі можуть встановити PWA, навіть не відвідуючи Play Market або App Store. Серед недоліків цього типу архітектури - необхідність вибору браузера та операційної системи, які повністю її підтримують.

Варіанти на кшталт PWA не підходять для стартапів. Він досить ефективний, якщо бізнес стабільний, а власник продукту знає, хто його кінцеві користувачі і якого досвіду вони очікують (наприклад, більшість з них - користувачі Android). Підтримка прогресивних веб-додатків не така вже й широка. Нещодавно Firefox (який все ще займає 6,3% ринку США) припинив підтримку PWA", що доводить, що цей тип веб-архітектури все ще нестабільний [].

Ще одна сучасна архітектура веб-додатків називається ізоморфною. Це тип JavaScript-додатків, які можуть працювати як на стороні клієнта, так і на стороні сервера. Спочатку клієнт завантажує HTML-файл, в який у браузер завантажується JavaScript-додаток, після чого додаток починає працювати як SPA.

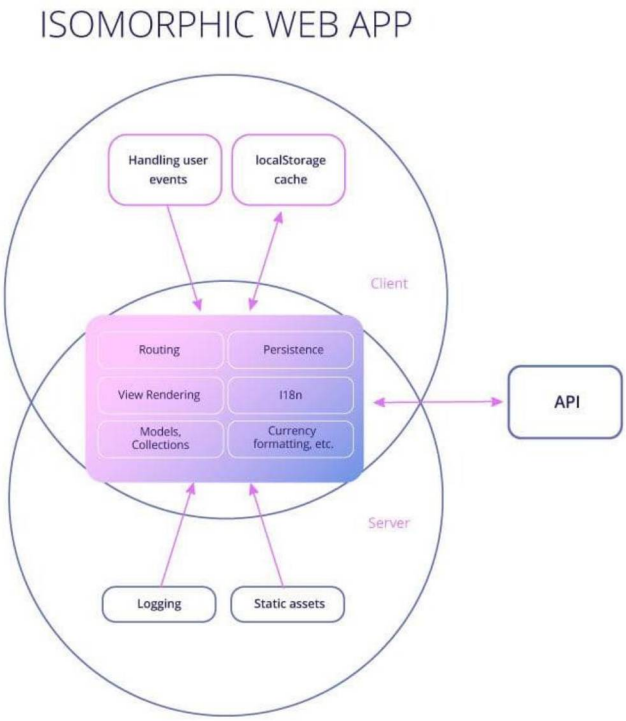


Рис. 1.10 – Ізоморфна архітектура

На відміну від SPA, тут перший рендеринг відбувається на сервері. Іншими словами, коли користувач вводить адресу веб-додатку, сервер спочатку повністю рендерить HTML (пакети JS). Це дозволяє пошуковим роботам Google вивчити веб-сторінки і виконати SEO вашого веб-додатку. В результаті, бізнес отримує вже відрендереною сторінку та JS-пакети веб-додатку SPA [].

На відміну від рендерингу на стороні сервера, ізоморфна веб-архітектура забезпечує швидке оновлення даних, чуйність і безліч варіантів UI/UX. Вона забезпечує швидший рендеринг, коли сервер завантажений, оскільки оброблений код передається клієнту. І на відміну від рендерингу на стороні клієнта, вона забезпечує миттєве відображення в браузері, зручну для користувача маршрутизацію, SEO та перелінковку. Єдиний недолік цього типу архітектури веб-додатків полягає в тому, що він повністю підтримується лише JavaScript. Найчастіше це означає, що набір технічних засобів обмежується фреймворками та інструментами на основі JS.

Серед інших принципів дизайну веб-додатків ми виділяємо мікрофронтенд - підхід, який будується на основі декомпозиції інтерфейсного додатку на окремі "мікро-додатки", що працюють разом. Для кінцевого користувача всі вони розташовуються на одній сторінці.

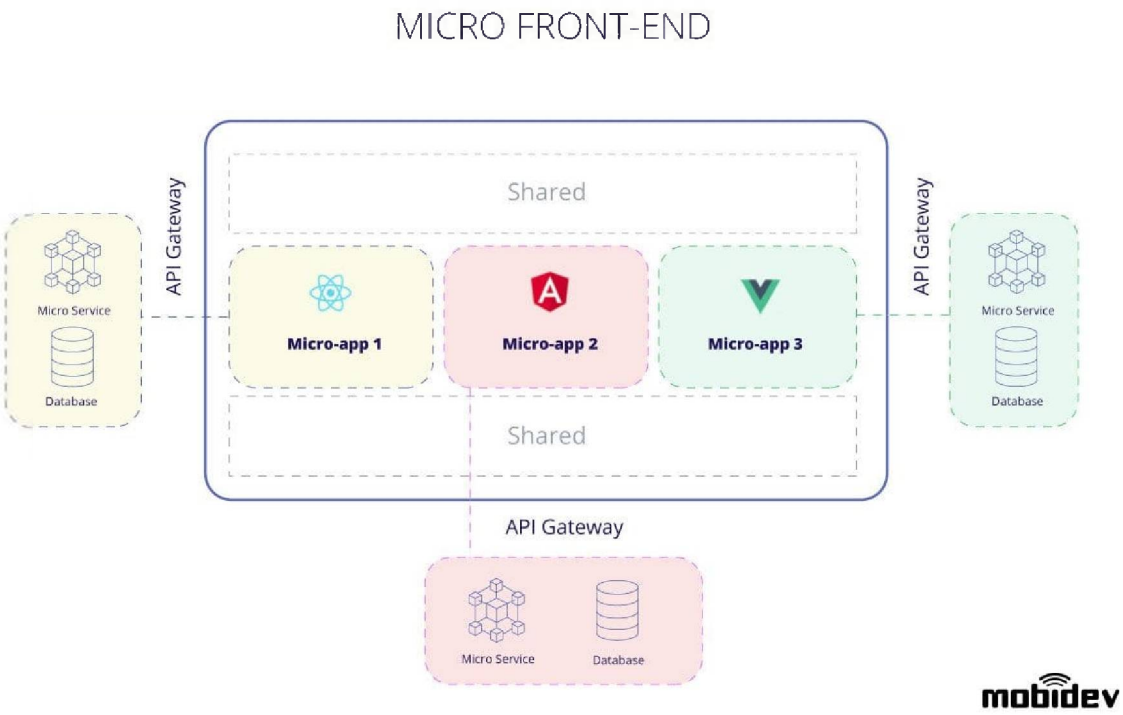


Рис. 1.11 – Мікрофронтенд-підхід

Цей тип архітектури веб-додатків є модульним, що означає, що сторінки та віджети є повністю незалежними додатками. При такому підході розробка та розгортання відбуваються паралельно. Але в той же час така структура робить ваш додаток складним і призводить до дублювання коду.

Незаперечною перевагою мікрофронтенд-архітектури є її масштабованість. Вона підходить для корпоративних клієнтів, оскільки часто розробка масивних частин програмного забезпечення розподіляється між кількома командами. Мікросервіси з'являються як на бек-, так і на фронтенді, і їх можна масштабувати відповідно до кількості ваших команд і навантаження, що працює в хмарі.

Концепцію Node.js та новий веб-фронтенд вперше сформулював у 2013 році Ніколас Закас, і зараз вона використовується для складних веб-рішень.

Відокремлення внутрішнього шару інтерфейсу користувача від внутрішньої бізнес-логіки має сенс лише у великих веб-архітектурах. Чому фронтенд-інженерів має хвилювати, яка мова на стороні сервера необхідна для виконання бізнес-критичних функцій? Чому це рішення повинно просочуватися в бекенд-рівень інтерфейсу? Потреби фронтенду кардинально відрізняються від потреб бекенду. Так Закас пояснює головний принцип цього типу розробки веб-додатків.

Цей тип архітектури веб-додатків складається з сервера, побудованого за допомогою Node.js, і шару інтерфейсу користувача. При цьому сервер бізнес-логіки може бути написаний будь-якою мовою (візьмемо для прикладу PHP) і використовувати API для зв'язку з сервером.

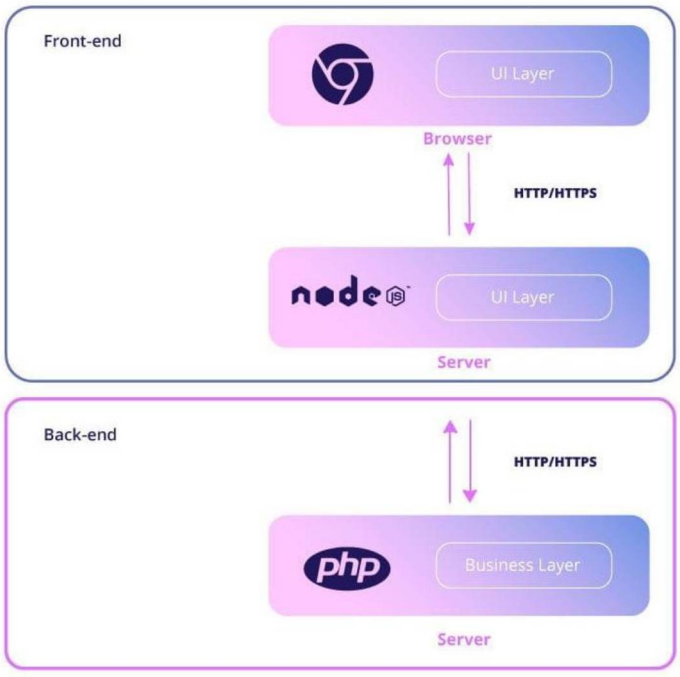


Рис. 1.12 – Концепція Node.js та новий веб-фронтенд

Фактично, новий підхід до веб-фронтенду дозволяє нам використовувати всі переваги ізоморфного типу архітектури, SSR або SPA, API для мобільних пристроїв та зв'язності. На відміну від ізоморфних веб-

додатків, тут немає жодних обмежень щодо мови чи платформи бізнес-логіки. Хоча слід зазначити, що розробка веб-додатку, який використовує нову логіку веб-фронтенду, займає більше часу, ніж той, що використовує тип SSG або SSR. Щоб заощадити час на розробку, ми пропонуємо використовувати фреймворки Next.js та Nuxt.js.

Концепція нового веб-фронтенду підходить, коли мова йде про розробку ізоморфного веб-додатку, де вже існує сервер API.

Говорячи про хмарну архітектуру для розробки веб-додатків, коротко розглянемо її серверну частину. Хмарна архітектура передбачає, що серверами керує хмарний провайдер, наприклад, Amazon AWS, Azure або Google Cloud. Окрім розміщення серверів на своїй стороні, ці провайдери пропонують комплекс послуг, які дозволяють створювати веб-додатки, розміщені та керовані в хмарі.

Дуже важливо вибрати правильну технологію для розробки хмарних веб-додатків. Популярним вибором є Python, який дозволяє автоматизувати завдання, керувати хмарними ресурсами та використовувати хмарні сервіси, що пропонуються різними провайдерами.

Безсерверні сервіси AWS - це одне з найпопулярніших хмарних рішень, яке використовується для реалізації таких популярних патернів, як мікросервіси, мобільні бекенди та односторінкові додатки. Наведена нижче схема дає розуміння того, як веб-сервіси AWS можуть бути використані для створення веб-додатків з використанням логіки 3-рівневої архітектури, яку ми пояснювали раніше

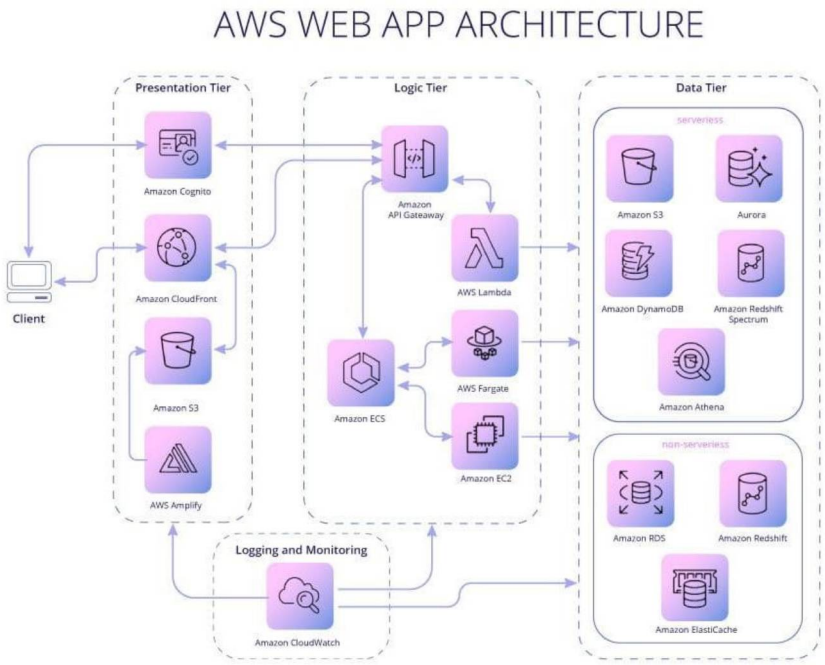


Рис. 1.13 – Безсерверна архітектура веб-додатків

Azure надає різноманітні сервіси хмарних обчислень, які дозволяють створювати базові та безпечні безсерверні веб-додатки.

Базовий веб-додаток створюється за допомогою Azure App Service і бази даних Azure SQL. Поєднання веб-додатків, Front Door, Function App, Azure DNS, Azure Cognitive Search і Azure DNS допомагає підвищити масштабованість і продуктивність веб-додатків Azure App Service.

Безсерверна веб-програма, розроблена за допомогою Azure, обслуговує статичний вміст зі сховища Azure Blob Storage і реалізує інтерфейс API за допомогою Azure Functions. API зчитує дані з Cosmos DB і повертає результати у веб-додаток.

Azure App Service Environment (ASE) використовується для розгортання веб-додатків, де потрібна додаткова безпека.

Обираючи найкращу архітектуру для веб та мобільного додатку, ви можете загубитися в різноманітті типів веб-архітектури, які доступні на ринку. Сучасна архітектура веб-додатків включає в себе ізоморфну, PWA, Micro front-end та інші.

1.3 Вивчення можливостей розробки веб-додатків

В даний час існує маса різних способів і засобів створити свій сайт, але будь-який з цих способів можна віднести до однієї з трьох категорій:

- за допомогою конструкторів;
- на базі CMS;
- самостійна розробка, в тому числі з використанням популярних інструментів і фреймворків.

По-перше, розглянемо метод розробки за допомогою конструкторів.

Без вміння працювати з HTML, створення веб-сайту може стати важкою задачею. Проте, існують конструктори веб-сайтів, які дозволяють швидко й ефективно створювати сайти навіть без глибоких знань програмування.

Конструктори веб-сайтів надають широкий вибір готових шаблонів, які можна налаштувати під власні потреби. Ці шаблони включають різноманітні дизайни та розміщення елементів, що робить процес створення сайту швидким та простим.

Однією з переваг конструкторів веб-сайтів є їх візуальні редактори. Ці редактори дозволяють змінювати розташування блоків, додавати графічні елементи та форматувати текст без необхідності вручного введення коду.

Крім того, деякі конструктори веб-сайтів включають в себе інші корисні функції, такі як можливість безкоштовного хостингу, що дозволяє публікувати ваш сайт в Інтернеті без додаткових витрат.

Використовуючи конструктор веб-сайтів, ви можете швидко створити свій особистий сайт з унікальним дизайном та контентом, не витрачаючи час на вивчення складних технологій програмування.

Конструктори веб-сайтів також дозволяють ефективно управляти вмістом вашого сайту, включаючи додавання, редагування та видалення контенту без необхідності втручання в код. Це робить процес оновлення сайту більш простим та зручним для користувачів без спеціалізованих навичок програмування або розробки веб-сайтів.

Таблиця 1.1

Переваги та недоліки використання метод розробки за допомогою конструкторів

Переваги	Недоліки
1. Низька ціна	1. Приховані витрати: конструктори можуть вимагати додаткові витрати за розміщення на хостингу, доменні імена та інші послуги.
2. Простота використання	2. Обмежені можливості доменного імені: безкоштовні версії конструкторів можуть обмежувати вибір доменного імені до доменів третього рівня або вимагати додаткової оплати за доменні імена другого рівня.
3. Вся рутина робиться конструктором	3. Ваговитість сайту: конструктори можуть генерувати зайвий програмний код, що призводить до повільного завантаження сторінок, особливо на повільних підключеннях до Інтернету.
	4. Відсутність SEO: навіть якщо конструктори мають інструменти SEO, їх можливості можуть бути обмеженими в порівнянні зі стандартними методами оптимізації для сайтів, створених на інших платформах.

Tilda Publishing - інтуїтивний конструктор сайтів, ідеально підходить для створення невеликих інформаційних, корпоративних ресурсів та інтернет-магазинів з обмеженою кількістю позицій. За останній рік Tilda значно розширила свої можливості, зокрема, додала повноцінний кошик з різними опціями доставки і оплати, блоки товарів з розширеним описом і збільшеними фотографіями, а також інтеграцію з платіжними системами. Функціонал Tilda зосереджений на стильному оформленні та включає багато блоків для різноманітного поєднання тексту, візуального та відео-контенту. Крім готових

шаблонів, користувачі можуть розробити свій власний дизайн з нуля, використовуючи конструктор.

Таблиця 1.2

Переваги та недоліки використання Tilda Publishing

Переваги	Недоліки
1. Можливість додати власні елементи за допомогою HTML, CSS і JS	1. Високі тарифи
2. Підключення стрічки постів Instagram з автоматичним оновленням	2. Потреба в певних знаннях веб-дизайну
3. Підключення HTTPS в інтерфейсі конструктора	
4. Функціонал для email-розсилок з подальшим відправленням через MailChimp, UniSender або інший сервіс за допомогою експорту коду	

WIX - потужний конструктор, придатний для створення різних типів веб-сайтів, таких як форуми, сайти-візитки, магазини, блоги тощо. Він пропонує простий інтуїтивний інтерфейс та широкий вибір стильних шаблонів (понад 500). Розробники регулярно оновлюють функціонал і додають нові можливості, зокрема, більше 250 віджетів у магазині додатків. Однак для новачків платформа може бути складною через перенасичену адміністративну панель, а для професіоналів і серйозного бізнесу може бути обмеженою, оскільки реалізація обмежена лише можливостями, передбаченими розробниками, навіть з можливістю додавання власного HTML-коду.

Таблиця 1.3

Переваги та недоліки використання WIX

Переваги	Недоліки
1. Мобільна і комп'ютерні версії редагуються окремо	1. Продуктивність сайту обмежена на тарифах, крім "Безлімітного"

Продовження таблиці 1.3

2. Багато додаткових віджетів і додатків, таких як конструктори форм, чати, кнопки соцмереж, сервіси аналітики, платіжні системи	2. Підключення власного домену доступне тільки на платних тарифах
3. Широкі можливості для створення стильних слайд-шоу, фото і відео-галерей	3. Техпідтримка по телефону надається тільки англійською, іспанською та португальською
4. Вбудований конструктор калькуляторів і форм	4. Додаткові витрати за підключення поштової скриньки Gmail з власною адресою
5. Можливість додати на сайт стрічку з Instagram і транслювати відео з Facebook	

По-друге, не менш важливим є аналіз методів розробки Web-сайтів на базі CMS. CMS, або система управління контентом, дозволяє створювати та керувати веб-сайтами без необхідності знання програмування. Вона поділяється на три типи: CMS з відкритим кодом, коробкові CMS і самописні CMS. CMS з відкритим кодом дають можливість модифікувати їх і володіють широким спектром розширень. Коробкові CMS мають обмежений функціонал, але є надійними та безпечними. Самописні CMS розробляються на замовлення та відповідають конкретним потребам, але можуть бути витратними та вимагати часу для підтримки.

Таблиця 1.4

Переваги та недоліки використання CMS

Перевага	Недоліки
1. Безкоштовний доступ: майже всі CMS безкоштовні, а також є безліч готових шаблонів для використання.	1. Уразливість сайту: навіть найпопулярніші CMS, такі як WordPress, часто стають об'єктом хакерських атак.

Продовження таблиці 1.4

2. Зручне управління контентом: легко керувати сайтом за допомогою панелі керування без потреби в глибоких знаннях.	2. Вимоги до знань: для розробки сайтів на CMS потрібні базові навички верстки та програмування.
3. Безліч готових рішень: у мережі доступна велика кількість модулів, плагінів та доповнень для різних потреб.	3. Великий сайт потребує великих витрат: розробка складних і великих проектів на CMS може бути витратною.

Розглянемо найпопулярніші CMS:

1. WordPress**.** Випущений вперше в 2003 році, CMS WordPress швидко завоював популярність як у просунутих розробників, так і простих користувачів. Завдяки простій налаштування, не найвищою вимогливістю до ресурсів хостингу і величезній кількості розширень ця CMS вже багато років займає перше місце. На сьогодні саме WordPress називають найкращою CMS для блогу. WordPress ідеально підходить для досить простих веб-сайтів, таких як щоденні блоги і новинні сайти, і для тих, хто шукає для себе просту CMS. Додатки дозволяють легко розширювати функціональність сайту. Наприклад, завдяки плагіну WooCommerce, з сайтів на движку WordPress виходить зручний для управління інтернет-магазин - один з найпоширеніших варіантів інтернет-магазинів в мережі.

Плюси WordPress:

- Безкоштовна CMS поширюється з відкритим вихідним кодом.
- Зручна панель адміністратора.
- Проста CMS для користувача. Відзначають простоту використання і легкість установки як движка, так і тим, і розширень.
- Досить висока продуктивність.

Мінуси WordPress:

- Відсутність технічної підтримки в НЕ SaaS варіантах.
- Багато плагіни написані неякісно, що створює проблеми в роботі і дірки в безпеці.
- Сайти на WordPress зламують найчастіше.

2. Joomla. Вперше побачила світ у 2005 році. Відображаючи філософію цього движка, його назвали словом, що звучить як «все разом». Фактично розробляється як CMS для порталів. Joomla дозволяє створювати сайти з більшою гнучкістю контенту і внутрішньої структури, ніж WordPress, але при цьому з досить простим і інтуїтивно зрозумілим інтерфейсом. Ця CMS підтримує електронну комерцію, соціальні мережі та багато іншого. Використовуючи цей движок, розробники створюють сайти-візитки, інтернет-магазини, фотогалереї, портали (включаючи новинні), блоги та інші сайти. Поруч користувачів, Joomla визнається найкращою CMS для сайту типу портал.

Плюси Joomla:

- Безкоштовне розповсюдження з відкритим вихідним кодом за ліцензією GNU GPL v2, включно із оновленнями;
- Часте надання оновлень движка;
- Велике співтовариство користувачів і розробників;
- Велика кількість доступних платних і безкоштовних тем і плагінів;
- Щодо невисокий рівень вимог до розробника і користувачеві.

Мінуси Joomla:

- Відсутність технічної підтримки.
- Друга CMS по числу зломів.

3. Drupal. Вперше вийшла в 2000 році, CMS Drupal є потужним, зручним для розробників інструментом для створення складних сайтів. Як і більшість потужних інструментів, Drupal вимагає певних знань і досвіду для роботи. На основі Drupal часто створюють портали,

новинні сайти, форуми, інтернет-магазини - одні з найбільш просунутих сайтів. Проте Drupal є найскладнішим для користувача движків з трійки лідерів. Хоча його використання з кожним випуском і стає все простіше, якщо ви не готові зануритися у вивчення цього програмного забезпечення або не можете найняти когось, хто його знає, можливо, це не найкраща система управління контентом для вас.

Плюси Drupal:

- Стабільна робота ядра движка.
- Досить розвинене співтовариство розробників.
- Для вирішення типових задач є готові набори плагінів.
- Відомий своєю потужною таксономії і здатністю відзначати, класифікувати і організовувати складний контент.

Мінуси Drupal:

- Складність використання для новачків.
- Менша кількість доступних безкоштовних плагінів ніж у попередніх CMS.
- Відзначають велику вимогливість до хостингу за рахунок більш частих звернень движка до бази даних, ніж у інших двигунів.

4. Самостійна розробка. Найбільш творчий і вільний, а й самий трудомісткий процес. Написання сайту або веб-додатки вимагає серйозних знань не тільки по самим мовам програмування, але і розуміння архітектури, бізнес-процесів клієнта і багато чого іншого. При цьому, створюючи сайт з нуля, клієнт отримає унікальний і персоніфікований продукт, який буде вирішувати його завдання і не витрачати час на зайві процеси. Самостійна розробка дозволяє створювати проекти будь-якої складності і з будь-яких побажань клієнта.

Переваги такої розробки:

- Свобода вибору. Ви можете замовити або виконати самостійно все, що необхідно для ефективного вирішення бізнес-завдань. При цьому, весь функціонал буде написаний саме під ваші потреби, а не адаптований з будь-якого шаблону;
- Широкі можливості просування. На відміну від CMS і конструкторів, просунути в природному пошуку самостійно розроблений сайт набагато легше.
- Індивідуальний дизайн. Тільки чистий код дозволить вам створити той продукт, який Ви бачите і хочете отримати. Окремо можна відзначити, що грамотний підхід до UI / UX може бути тільки при використанні чистого коду, так як при використанні CMS, клієнт змушений звертатися до готових рішень, які можуть не відповідати всім його потребам.

Недоліки такої розробки:

- Ціна. Суперечливий, але все-таки недолік чистого коду. Хоча в разі великих проектів, розробка без конструкторів і CMS є найефективнішим і самим недорогим рішенням, так як в результаті не доведеться отримувати не те, що хотілося, а в подальшому не переробляти за будь-якої необхідності;
- Наявність знань. Самостійна розробка вимагає наявності великої кількості знань з мов програмування, побудови архітектури веб-додатків, алгоритмів і структур даних, бізнес-процесів і багато чому іншому, що тягне за собою необхідність звертатися до найманих фахівців або в веб-студії;
- Витрата часу. Як не крути, але використання CMS і конструкторів дозволяє зробити простий сайт швидше, ніж при розробці без них. У ситуаціях, коли час грає велику роль, звернення до чистого коду стає безглуздим.

Висновки за розділом 1

У двному розділі проведено аналіз різних методів створення веб-систем та класифікацію типів веб-сайтів. Він включає в себе розгляд форм веб-сайтів, їх структури та функціоналу. Також досліджено архітектуру веб-додатків, зокрема, способи організації програмного забезпечення для веб-серверів та клієнтів. Крім того, в розділі проведено огляд можливостей для розробки веб-додатків, їхніх особливостей та переваг.

РОЗДІЛ 2
ВИБІР ТЕХНОЛОГІЙ ПРОГРАМУВАННЯ ТА УПРАВЛІННЯ БАЗАМИ ДАНИХ

2.1. Обґрунтування використання технології ASP.NET MVC і мови C#

ASP.NET MVC - це фреймворк для розробки веб-рішень від корпорації Microsoft, випущений ще в 2009 році. Це альтернатива моделі кодування Web Forms для створення веб-додатків.

Фреймворк дозволяє швидко створювати надійні та масштабовані веб-додатки корпоративного класу, динамічні веб-сайти, інтерактивні сторінки та сервіси. Він підтримує HTML/JS, WML та XML, VB.NET, C#, J#, JScript.NET та Microsoft Visual Studio. Фреймворк успадкував найкращі риси ASP.NET. Варто згадати потужну серверну технологію, побудовану на основі Common Language Runtime (CLR) [3].

ASP.NET MVC також надає архітектуру Model View Controller, яка є передовим шаблоном проектування. Зазвичай він використовується для розробки інтерфейсу користувача. Контролер представлення моделі (MVC) розділяє веб-рішення на три окремі компоненти - модель (Model), представлення (View) і контролер (Controller). Після цього архітектурний патерн MVC визначає взаємодію між цими компонентами.

Раніше фреймворк був популярним серед розробників програмного забезпечення для програмування веб-додатків і сервісів. Такі додатки легко створювати, налагоджувати та розгортати. Деякі з найбільших міжнародних компаній мають свої веб-сайти з високим трафіком, написані на цьому фреймворку. Серед них Microsoft, Ikea, Volvo, Stack Overflow тощо.

Модель кодування MVC чудово підходить для створення веб-додатків та API. Вона також оптимізована для використання з ASP.NET Core. Фреймворки, що використовуються з моделлю програмування MVC, мають відкритий вихідний код, легкі та добре піддаються тестуванню. Вони

використовують найновіші веб-стандарти та підтримують TDD-дружню розробку.

Говорячи про суттєві відмінності між ASP.NET MVC та ASP.NET Core MVC, слід зосередитися на ASP.NET та ASP.NET Core як двох версіях фреймворку. Як ми вже згадували раніше, модель програмування MVC може бути легко використана з обома, тому це не є основним фактором, який слід враховувати [4].

Таблиця 2.1

Основні особливості версій фреймворку

Критерії порівняння	ASP.NET MVC	ASP.NET Core
Підтримка платформи	Windows	Windows, Linux та Mac
Архітектура	Базується на .Net Framework	Базується на .Net Core
Компоненти	Web Forms, MVC та Web API	MVC, Web API, Razor Pages та Blasor
Залежності	Менший контроль над залежностями	Суворий контроль над залежностями
Підтримка файлів	WCF, WF, WPF, VB, Web config та інші	Немає підтримки для Global.asax, Web config
Visual Studio	Підтримка всіх версій	Підтримка останніх версій з 2015 року

Він базується на паттерні проектування Model-View-Controller (MVC), який розділяє додаток на три основні компоненти:

1. Model (Модель): Представляє дані вашого застосунку та бізнес-логіку. Модель відповідає за зберігання, отримання, маніпулювання та оновлення даних.

2. View (Представлення): Відповідає за візуальне відображення даних користувачеві. Вона використовує дані з моделі для генерації HTML-сторінок, які бачить користувач.
3. Controller (Контролер): Є посередником між моделлю та представленням. Він обробляє запити користувача, взаємодіє з моделлю для отримання або оновлення даних, а потім вибирає відповідне представлення для відображення інформації.

Використання ASP.NET MVC:

ASP.NET MVC підходить для розробки різних типів веб-застосунків, зокрема:

- Веб-сайти з динамічним контентом;
- Інтернет-магазини;
- Веб-сервіси API;
- Односторінкові застосунки (SPA).

Переваги ASP.NET MVC:

- Чиста архітектура: розділення моделі, представлення та контролера забезпечує чисту та модульну архітектуру, що робить код більш організованим, легким для тестування та підтримки.
- Гнучкість: ASP.NET MVC дозволяє використовувати різні технології для представлення (наприклад, Razor Pages, шаблони MVC) та моделі (бази даних, файли тощо).
- SEO: оскільки представлення генеруються динамічно на основі даних, ASP.NET MVC дозволяє створювати веб-застосунки, які більш сприятливі для пошукових систем.
- Тестування: MVC архітектура сприяє написанню модульних та тестованих компонентів.

Недоліки ASP.NET MVC:

- Складність навчання: для початківців MVC може здатися складнішим за інші фреймворки, оскільки він вимагає розуміння всіх трьох компонентів (модель, представлення, контролер).
- Більше коду: через розділення на компоненти, може знадобитися писати більше коду, ніж у деяких інших фреймворках.

Мова програмування C# - це проста, сучасна, об'єктно-орієнтована та безпечна щодо типів мова програмування. C# походить із сімейства мов C і буде одразу знайома програмістам, які володіють мовами C, C++ та Java. C# стандартизована ECMA International як стандарт ECMA-334 та ISO/IEC як стандарт ISO/IEC 23270. Компілятор C# від Microsoft для .NET Framework є відповідною реалізацією обох цих стандартів. C# є об'єктно-орієнтованою мовою, але C# також включає підтримку компонентно-орієнтованого програмування. Сучасне проектування програмного забезпечення все більше покладається на програмні компоненти у вигляді самодостатніх і самоописуваних пакетів функціональності. Ключовим для таких компонентів є те, що вони представляють модель програмування з властивостями, методами та подіями; вони мають атрибути, які надають декларативну інформацію про компонент; і вони включають власну документацію. C# надає мовні конструкції для безпосередньої підтримки цих концепцій, що робить C# дуже природною мовою для створення та використання програмних компонентів. Деякі можливості C# допомагають створювати надійні та довговічні програми: Збір сміття автоматично звільняє пам'ять, зайняту невикористаними об'єктами; обробка винятків забезпечує структурований і розширюваний підхід до виявлення та відновлення помилок; а безпечний для типів дизайн мови унеможливлює читання з неініціалізованих змінних, індексування масивів за їх межами або виконання неперевірених приведень типів.

C# має уніфіковану систему типів. Всі типи C#, включаючи примітивні типи, такі як int та double, успадковуються від одного кореневого типу об'єкта.

Таким чином, всі типи поділяють набір спільних операцій, і значення будь-якого типу можна зберігати, транспортувати та оперувати з ними у послідовний спосіб. Крім того, C# підтримує як користувацькі типи посилань, так і типи значень, що дозволяє динамічно розподіляти об'єкти, а також зберігати в потоці легкі структури.

Щоб забезпечити сумісний розвиток програм і бібліотек C# з часом, при розробці C# значну увагу було приділено версіонуванню. У багатьох мовах програмування цьому питанню приділяється мало уваги. Як наслідок, програми, написані цими мовами, ламаються частіше, ніж потрібно, коли з'являються нові версії залежних бібліотек. Аспекти дизайну C#, на які безпосередньо вплинули міркування щодо версійності, включають окремі модифікатори virtual та override, правила дозволу перевантаження методів та підтримку явних оголошень членів інтерфейсу.

Розглянемо основні можливості мови C#:

- Об'єктно-орієнтований підхід: C# є повністю об'єктно-орієнтованою мовою програмування, що дозволяє створювати програми, орієнтовані на об'єкти. Це дозволяє структурувати програмний код у вигляді класів та об'єктів, що спрощує розробку та підтримку великих проектів.
- Сильна типізація: у C# всі змінні мають визначений тип даних, що дозволяє виявляти помилки на етапі компіляції. Це сприяє підвищенню надійності програм та полегшує роботу розробників.
- Інтеграція з .NET Framework та .NET Core: C# є частиною платформ .NET, які надають доступ до широкого спектру бібліотек та інструментів для розробки програм. Це дозволяє розробникам створювати різноманітні застосунки для різних платформ та пристроїв.
- Асинхронне програмування: мова має вбудовану підтримку асинхронних операцій, що дозволяє розробникам створювати ефективні програми з високою продуктивністю. Це особливо важливо для веб-

додатків та застосунків, які використовують велику кількість вхідних/вихідних операцій.

- Крос-платформовість: за допомогою .NET Core та .NET 5, C# став крос-платформовою мовою, що може працювати на різних операційних системах, таких як Windows, macOS та Linux. Це розширює коло застосування мови та дозволяє розробникам створювати програми для різних платформ.
- Стандартна бібліотека: C# має обширну стандартну бібліотеку класів та методів, які допомагають у розв'язанні різноманітних задач програмування, включаючи роботу зі стрічками, масивами, колекціями, файлами, мережевими операціями та багато іншого.
- Паралельне програмування: мова має інструменти для паралельного програмування, що дозволяє розробникам створювати програми, які ефективно використовують ресурси багатоядерних систем. Це дозволяє підвищити продуктивність та ефективність застосунків.
- Інтегровані середовища розробки (IDE): для розробки програм на C# існують різноманітні інтегровані середовища розробки, такі як Visual Studio та Visual Studio Code, які надають широкий спектр інструментів для підтримки розробки програм на мові C#. Ці середовища включають в себе редактори коду, засоби для налагодження програм, системи керування версіями, аналізатори коду та інші корисні функції.

Мова програмування C# має ще багато інших функцій та можливостей, які роблять її популярним вибором для розробки різних видів програмних застосунків.

Для розробки серверної частини була обрана технологія ASP.NET MVC і мову C #, що є найпопулярнішою мовою для написання бізнес логіки на платформі .NET, основні переваги технології ASP.NET MVC:

– використання вбудованого архітектурного шаблону MVC;

– масштабованість, так як платформа MVC побудована у вигляді серії незалежних компонентів, реалізуючи інтерфейс .NET або побудованих на

основі абстрактного базового класу, можна легко замінити систему маршрутизації, механізм візуалізації, фабрику контролерів або інші компоненти платформи іншими, з власної спеціальної реалізацією;

– жорсткий контроль над HTML і HTTP. У платформі ASP.NET MVC врахована важливість генерації ясного і відповідного стандартам коду розмітки. Її вбудовані допоміжні методи HTML створюють відповідний стандартам висновок, але в ній реалізовано і набагато більш значуще філософське зміна в порівнянні з Web Forms. Замість величезного обсягу важко піддається управлінню HTML-коду, платформа MVC стимулює створення простих і елегантних, стилізованих за допомогою CSS компонентів.

Як IDE для розробки серверної частини була обрана Visual Studio 2019 – це інтегроване середовище розробки Microsoft (IDE), що є самою функціональною IDE для розробки на платформі .NET.

Для реалізації клієнтської частини використовується технологія розробки SPA React. Вибір обґрунтований тим, що React є повністю незалежним від сервера, має декларативний підхід до написання коду. Також реактивний додаток не створюється з «коробки», що дає нам високу ступінь кастомізації архітектури додатку.

Головні переваги бібліотеки React:

– використовує віртуальний DOM, який є об'єктом JavaScript. Це покращує продуктивність додатків, оскільки віртуальний JavaScript DOM швидше, ніж звичайний DOM;

– може використовуватися як на стороні клієнта, так і на стороні сервера, а також на інших платформах;

– компоненти і моделі даних покращують читаність коду, що допомагає підтримувати більші додатки.

2.2. Вибір системи управління реляційною базою даних MySQL

Під час розробки системи на основі Entity Framework Core, важливо враховувати кращі практики і шаблони проектування. Наприклад, розподіл

коду на різні шари, такі як шар доступу до даних (Data Access Layer), бізнес-логіка (Business Logic Layer) та представлення (Presentation Layer), дозволяє підтримувати чистоту коду, полегшує тестування і підтримку [5].

Крім того, EF Core пропонує можливості для оптимізації продуктивності додатка. Наприклад, використання властивостей Lazy Loading і Eager Loading дозволяє ефективно завантажувати пов'язані об'єкти з бази даних, зменшуючи кількість запитів та час завантаження даних.

Також важливо розглядати можливості кешування даних на різних рівнях (наприклад, кешування рівня пам'яті, кешування рівня сервера, кешування CDN), щоб підвищити швидкодію додатка та зменшити навантаження на сервер баз даних.

Забезпечення безпеки даних є однією з ключових задач при розробці будь-якої системи. EF Core дозволяє використовувати параметризовані запити та інші методи, які запобігають SQL ін'єкціям та іншим видам атак на безпеку даних [6].

Необхідно також розглянути можливості міграції даних та автоматичного створення схеми бази даних. EF Core надає зручний механізм міграції, який дозволяє змінювати схему бази даних з практично нульовими витратами часу на ручне втручання [7].

Загалом, правильне використання Entity Framework Core в поєднанні з кращими практиками розробки програмного забезпечення дозволяє створювати надійні, ефективні та легко розширювані додатки з доступом до даних.

Шаблони проектування та архітектура:

- Шари: розбивка коду на шари (Data Access Layer (DAL), Business Logic Layer (BLL), Presentation Layer (PL)) сприяє чіткій структурі, кращій модульності, тестуванню та підтримці.
- Інверсія залежностей (DI): застосування DI-контейнерів (як Autofac, SimpleInjector) робить код більш гнучким, тестованим та розширюваним.

- Репозиторний шаблон: використання репозиторіїв для абстрагування доступу до даних дає кращий контроль над CRUD-операціями та спрощує тестування.

Для оптимізація продуктивності можна використовувати наступне:

- Lazy Loading: застосування Lazy Loading для завантаження даних за потребою, а не всіх пов'язаних об'єктів одразу, економить ресурси та час завантаження.
- Eager Loading: Eager Loading використовується, коли потрібні всі пов'язані дані одразу, що може покращити продуктивність при меншій кількості запитів.
- Кешування: використання кешу на різних рівнях (пам'ять, сервер, CDN) зменшує навантаження на БД та покращує час відповіді.
- Індексування: створення індексів для часто використовуваних полів БД може значно покращити продуктивність запитів.
- Оптимізація запитів: написання ефективних запитів SQL з мінімальною кількістю JOIN та WHERE може суттєво вплинути на продуктивність.

Для безпеки даних слід використовувати наступне:

- Параметризовані запити: застосування параметризованих запитів замість вбудованих рядків SQL запобігає SQL-ін'єкціям та іншим атакам.
- Шифрування: шифрування даних у стані спокою та під час передачі захищає їх від несанкціонованого доступу.
- Аутентифікація та авторизація: застосування механізмів аутентифікації та авторизації для контролю доступу до даних та ресурсів системи.
- Рольовий доступ: Надання користувачам різних рівнів доступу до даних та функцій залежно від їх ролей.

Entity Framework Core - це потужний фреймворк для доступу до даних, який пропонує широкий спектр можливостей для розробки надійних, ефективних та масштабованих додатків. Використання кращих практик,

шаблонів проектування та можливостей оптимізації EF Core може значно покращити вашу продуктивність та безпеку [8].

2.3. Аналіз переваг інтеграції обраних технологій для web-системи

Інтеграція обраних технологій - ASP.NET MVC, мови програмування C# та бази даних MySQL - має значні переваги для реалізації онлайн клініки. Ось деякі з них:

1. ASP.NET MVC та C#: ASP.NET MVC - це відмінний веб-фреймворк, який забезпечує швидку розробку та підтримку веб-додатків. Його модель розробки, розділяючи код на модель, представлення та контролери, сприяє створенню добре організованих та легко розширюваних додатків. Мова програмування C# має потужний функціонал та велику спільноту розробників, що робить її ідеальним вибором для реалізації бізнес-логіки онлайн клініки [9].
2. Ефективна розробка: інтеграція ASP.NET MVC та C# дозволяє розробникам швидко розробляти функціонал онлайн клініки, використовуючи шаблони проектування та готові компоненти фреймворку ASP.NET [10].
3. Безпека: ASP.NET забезпечує широкі можливості для захисту від різних типів атак, таких як SQL-ін'єкції, XSS і CSRF. Це дозволяє створювати безпечні додатки для збереження конфіденційної медичної інформації пацієнтів.
4. Широкі можливості розширення: ASP.NET MVC дозволяє легко розширювати функціонал додатків за допомогою сторонніх бібліотек та плагінів. Це дозволить легко додавати нові функції та інтегрувати сторонні сервіси для поліпшення функціоналу онлайн клініки.
5. MySQL як база даних: MySQL - це потужна та надійна реляційна база даних, яка відома своєю швидкодією та ефективністю. Вона має широку підтримку серед розробників та великий вибір інструментів для адміністрування та розробки [11].

6. Масштабованість: якщо потрібно масштабувати онлайн клініку в майбутньому, обрані технології дозволять легко реалізувати горизонтальне та вертикальне масштабування, забезпечуючи високу доступність та продуктивність системи.

Таблиця 2.2

Порівняння ASP.NET MVC та C# з базою даних MySQL

Критерії порівняння	ASP.NET MVC	MySQL
Швидкодія розробки	Висока	-
Масштабованість	Так	Так
Ефективність	Висока	Висока
Безпека	Висока	Висока
Можливість розширення	Велика	-
Спільнота та підтримка	Велика	Велика
Швидкодія та ефективність БД	-	Висока
Функціональність	Розширена	Розширена
Масштабування БД	-	Так
Інструменти адміністрування	-	Широкий вибір
Вартість та ліцензування	Залежить від конкретної реалізації	Відкрита (безкоштовна)

Інтеграція ASP.NET MVC, мови програмування C# та бази даних MySQL надасть потужні засоби для реалізації онлайн клініки з ефективним функціоналом, високою безпекою та можливістю масштабування.

Висновки за розділом 2

Інтеграція ASP.NET MVC, мови програмування C# та бази даних MySQL надає високі потужності та забезпечує оптимальні умови для успішної реалізації онлайн клініки. З використанням цих технологій можна створити додаток з ефективним функціоналом, забезпечити високий рівень безпеки для конфіденційної медичної інформації пацієнтів та забезпечити можливість масштабування системи для відповіді на зростаючі потреби користувачів. Такий підхід дозволяє створити надійну та функціональну онлайн клініку, яка відповідає сучасним вимогам та потребам користувачів.

РОЗДІЛ 3
РОЗРОБКА ТА ТЕСТУВАННЯ WEB-СИСТЕМИ ОНЛАЙН-ПОЛІКЛІНІКИ

3.1. Проєктування та розробка web-системи

Діаграма прецедентів або «Use Case Diagram» використовується для відображення відносини між акторами і прецедентами. Актори це хтось або щось, що має взаємодіяти з системою. Прецеденти використовуються для побудови діалога між актором і системою, вони визначають можливості, що Діаграма прецедентів або «Use Case Diagram» використовується для відображення відносини між акторами і прецедентами. Актори це хтось або щось, що має взаємодіяти з системою. Прецеденти використовуються для побудови діалога між актором і системою, вони визначають можливості, що забезпечуються системою для актора.

У системі існують актори, наприклад: «неавторизований користувач», «авторизований користувач» і «адміністратор».

Неавторизований користувач – це користувач, який не пройшов аутентифікацію. Якщо йому потрібно створити запис, то у нього є функції реєстрації або авторизації.

Авторизований користувач – це користувач, який пройшов аутентифікацію і його функціонал залежить від ролі. Він може бути авторизованим користувачем або адміністратором. Авторизований користувач має перегляду профілю, вихід з облікового запису та може створити запис.

Адміністратор – це користувач, який пройшов аутентифікацію має роль адміністратора. Також має функції адміністратора це – перегляд усіх записів і зміна замовлення статусу на підтверджений або відхилений, який спочатку мав статус в очікуванні.

На рисунку 3.1 зображена діаграма прецедентів (Use Case Diagram).

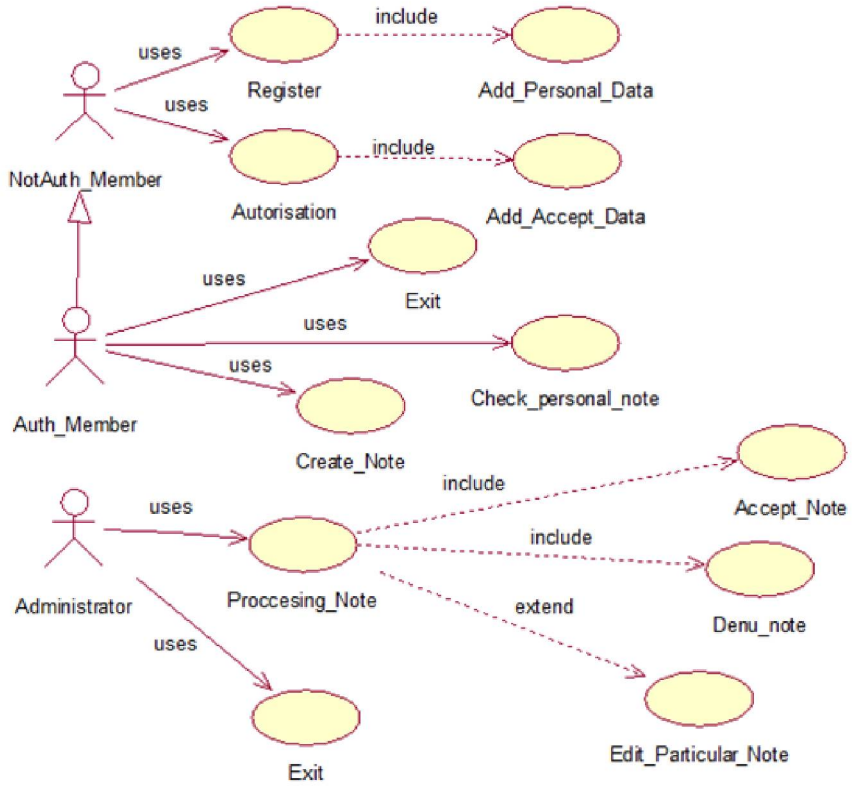


Рисунок 3.1 – Діаграма прецедентів (Use Case Diagram)

Зображено діаграма випадків використання UML (Unified Modeling Language), яка показує взаємодію між користувачами та системою, розподілену за різними типами користувачів: “NotAuth Member”, “Auth Member” та “Administrator”. Кожен тип користувача має пов’язані дії, представлені овалами, такими як “Register”, “Add_Personal_Data” та “Create_Note”. Діаграма використовує лінії для з’єднання користувачів з їхніми діями, що вказує на відносини, такі як ‘uses’, ‘include’ та ‘extend’. Такий тип діаграми є актуальним для розробки програмного забезпечення та проектування систем, оскільки допомагає зрозуміти, як різні користувачі будуть взаємодіяти з системою.

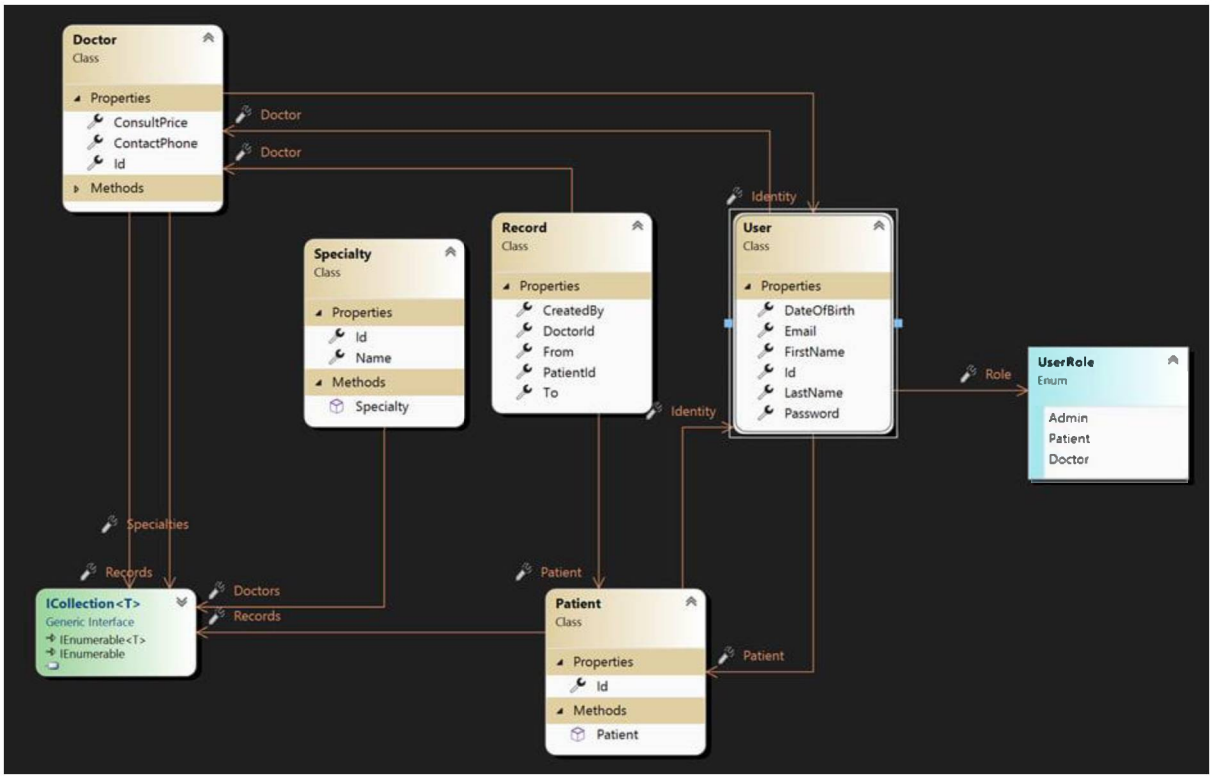


Рис. 3.2 – Структура БД

Зображено структуру БД UML (Unified Modeling Language), яка використовується в програмній інженерії для візуального представлення структури системи. На ній показані класи, такі як “Лікар”, “Користувач”, “Пацієнт” та “Запис”, їх атрибути, операції (або методи) та відносини між об’єктами.

Ось основні елементи структури:

- Класи: кожен клас має свій набір властивостей. Наприклад, клас “Лікар” має властивості “Спеціальність” та “Ім’я”.
- Відносини: лінії, що з’єднують класи, представляють асоціації, такі як наслідування (показане порожніми стрілками) та композиція (зображена заповненими ромбами).

Ця структура UML корисна, оскільки надає чітке та організоване розуміння того, як різні компоненти системи взаємодіють один з одним, що є важливим для розробки програмного забезпечення та аналізу.

Якщо ж дивитися на розробку вимог до функцій серверної частини, то можна виділити наступне. Під час розробки вимог до серверної частини необхідно враховувати різноманітні аспекти, щоб забезпечити оптимальну продуктивність та надійність системи. Серверна частина повинна бути здатна обробляти запити всіх користувачів одночасно, забезпечуючи швидку обробку всіх замовлень та їх відображення у базі даних. Важливо також врахувати потреби користувачів, які реєструються в системі, та забезпечити зручний та безперебійний процес реєстрації та авторизації [12].

Щодо серверного оточення, важливо забезпечити максимальну продуктивність, масштабованість та надійність. Одне серверне середовище, що включає веб-сервер, сервер додатків та сервер баз даних, повинно бути оптимізоване для роботи з великою кількістю одночасних запитів, забезпечуючи швидку відповідь та надійну обробку даних. Також важливо враховувати вартість та простоту управління серверним оточенням, забезпечуючи ефективне використання ресурсів та легке адміністрування.

Не менш важливою є розробка вимог до функцій клієнтської частини. Клієнтська частина системи повинна забезпечувати зручний та доступний інтерфейс для користувачів, що дозволяє легко отримувати доступ до всіх функцій системи. Наприклад, користувачам повинна бути доступна інформація про лікаря, можливість записатися на прийом та переглядати історію своїх візитів. Важливо також забезпечити швидку завантаження системи та доступність з різних пристроїв, а також забезпечити зрозумілий та простий інтерфейс, який відповідає потребам користувачів.

Розробка вимог до функцій серверної та клієнтської частини системи є ключовим етапом у процесі створення ефективної та функціональної онлайн клініки. Врахування різноманітних аспектів, таких як продуктивність, масштабованість, доступність та зручність інтерфейсу, допоможе створити систему, що відповідає всім потребам користувачів.

3.2 Приклади SQL запросів

Було протестовані всі функції: реєстрація, авторизація, запис даних до сесії, обробка записів, відображення записів.

Також було протестовані всі функції валідації (при введенні даних під час авторизації, під час реєстрації та створення запису). Проведені тести на функцію доступу до сторінки адміністратора після введення посилання.

Запити сформульовані за допомогою синтаксису LINQ (Language Integrated Query) і включають команди SELECT для отримання даних з таблиць ‘Users’ та ‘Doctors’. На скріншоті показано виконання цих запитів з параметрами, такими як ‘Email’, ‘Password’, ‘FirstName’, ‘LastName’, ‘Id’ та ‘Role’.

```
      Content root path: C:\Users\Vlad\Source\Repos\DoctorsApp\Artem.Doctors.Api
info: Microsoft.EntityFrameworkCore.Database.Command[20101]
      Executed DbCommand (43ms) [Parameters=[@__model_Email_0='?' (Size = 4000)], CommandType='Text', CommandTimeout='30']
      SELECT TOP(1) [u].[Id], [u].[DateOfBirth], [u].[Email], [u].[FirstName], [u].[LastName], [u].[Password], [u].[Role]
      FROM [Users] AS [u]
      WHERE [u].[Email] = @__model_Email_0
info: Microsoft.EntityFrameworkCore.Database.Command[20101]
      Executed DbCommand (5ms) [Parameters=[@__get_Item_0='?' (DbType = Guid)], CommandType='Text', CommandTimeout='30']
      SELECT TOP(1) [u].[Id], [u].[DateOfBirth], [u].[Email], [u].[FirstName], [u].[LastName], [u].[Password], [u].[Role]
      FROM [Users] AS [u]
      WHERE [u].[Id] = @__get_Item_0
info: Microsoft.EntityFrameworkCore.Database.Command[20101]
      Executed DbCommand (0ms) [Parameters=[@__get_Item_0='?' (DbType = Guid)], CommandType='Text', CommandTimeout='30']
      SELECT TOP(1) [u].[Id], [u].[DateOfBirth], [u].[Email], [u].[FirstName], [u].[LastName], [u].[Password], [u].[Role]
      FROM [Users] AS [u]
      WHERE [u].[Id] = @__get_Item_0
```

Рис. 3.3 - Авторизація адміна в системі

У коді присутні кілька операторів SELECT, які використовуються для отримання даних з різних таблиць. Використовуються JOIN операції для об’єднання рядків з двох або більше таблиць за допомогою спільних стовпців.

Цей код не містить математичних завдань, він виключно складається з SQL коду.

```
info: Microsoft.EntityFrameworkCore.Database.Command[20101]
      Executed DbCommand (10ms) [Parameters=[], CommandType='Text', CommandTimeout='30']
      SELECT [d].[Id], [u].[FirstName], [u].[LastName], [u].[DateOfBirth], [d].[ContactPhone], [d].[ConsultPrice], [u].[Email], [u].[Id], [t].[Id], [t].[Name], [t
      FROM [Doctors] AS [d]
      INNER JOIN [Users] AS [u] ON [d].[Id] = [u].[Id]
      LEFT JOIN (
          SELECT [s].[Id], [s].[Name], [d0].[DoctorsId], [d0].[SpecialtiesId]
          FROM [DoctorSpecialty] AS [d0]
          INNER JOIN [Specialties] AS [s] ON [d0].[SpecialtiesId] = [s].[Id]
      ) AS [t] ON [d].[Id] = [t].[DoctorsId]
      ORDER BY [d].[Id], [u].[Id], [t].[DoctorsId], [t].[SpecialtiesId], [t].[Id]
```

Рисунок 3.4 – Адмін здійснює пошук лікарів

Код включає команди SQL, такі як SELECT CASE, FROM, WHERE, INSERT INTO та VALUES, які використовуються для запитів та маніпуляцій з даними в базі даних. Показано використання умовних операторів для визначення результату та вставки даних у конкретні таблиці бази даних.

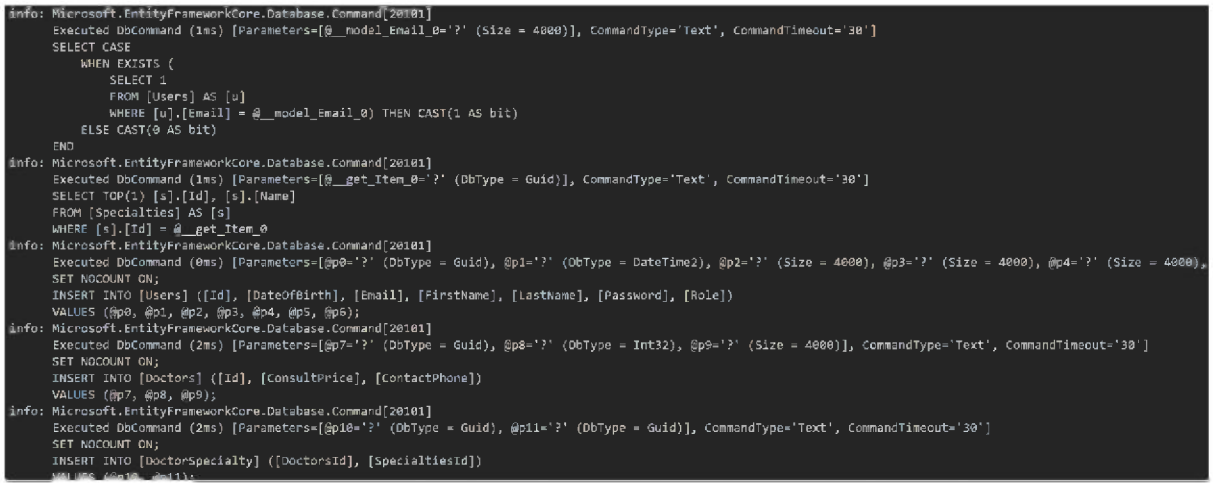


Рис. 3.5 – Адмін створив акаунт для нового окуліста

На рис. 3.6 зображено код. Код містить команди для виконання запитів SELECT до таблиці ‘Users’, де вибираються поля Id, FirstName, LastName, Email, PasswordHash, PasswordSalt та Role. Також присутні посилання на параметри “@GetOne” та “@model.Email”, що свідчить про те, що код є частиною більшого додатку, пов’язаного з аутентифікацією або управлінням користувачами.

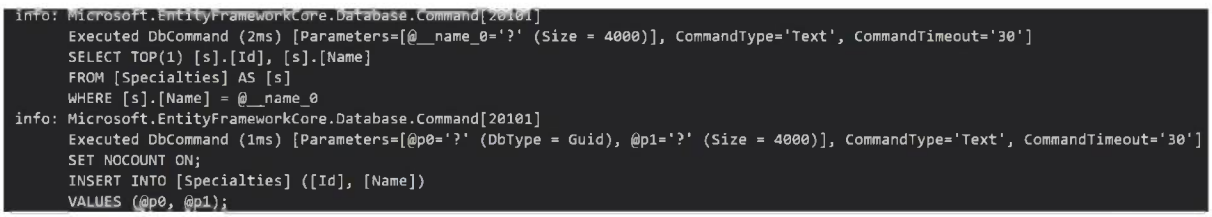


Рисунок 3.6 – Адмін додав нову спеціальність therapist

Цей запит вибирає поля UpdatedOn, CreatedBy, From та To з таблиці Records, де значення колонки PatientId відповідає заданому параметру. Це

типовий запит для отримання даних про пацієнтів з бази даних у додатках, що використовують .NET та Entity Framework.

```
info: Microsoft.EntityFrameworkCore.Database.Command[20101]
      Executed DbCommand (2ms) [Parameters=[@__patientId_0='?' (DbType = Guid)], CommandType='Text', CommandTimeout='30']
      SELECT [r].[DoctorId], [r].[PatientId], [r].[From], [r].[To]
      FROM [Records] AS [r]
      WHERE [r].[PatientId] = @__patientId_0
```

Рисунок 3.7 – Пацієнт відкрив список записів на прийом

Код запиту містить декілька операторів SELECT, які використовуються для запиту даних з бази даних. Зокрема, в коді йдеться про запити даних з таблиць ‘Documents’ та ‘Patients’, можливо, їх об’єднання за певними умовами та вибір конкретних полів, таких як ‘doc_id’ та ‘pat_id’. Також присутні умови WHERE для фільтрації результатів за певними критеріями, такими як ‘doc_type_id’, ‘model_to_use’ та інші умови. Цей скріншот може бути корисним для тих, хто вивчає SQL або налагоджує запити до бази даних.

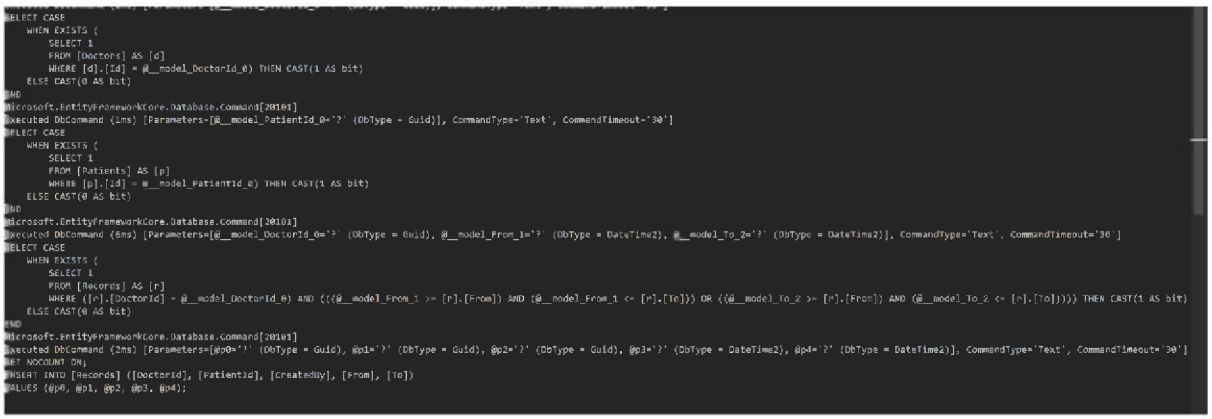


Рис. 3.8 – Пацієнт відкрив список докторів та записався на прийом

Код включає оператори SELECT для отримання даних з однієї або декількох таблиць бази даних, а також логічні умовні оператори, такі як IF…ELSE. Також присутні коментарі, позначені подвійними тире (–), які пояснюють або анотують частини коду для кращого розуміння. Цей скріншот

цікавий тим, що демонструє структуру та логіку виконання запитів до баз даних.

Рис. 3.9 – Створення нового акаунту

3.3 Графічний інтерфейс користувача та тестування

На рис. 3.10 зображено веб-сторінку, яка призначена для запису на прийом до лікаря онлайн. Ось основні елементи:

- Заголовок: у верхньому лівому куті є слово “Лікарі” з іконкою, що нагадує стетоскоп.
- Розділи: є дві вкладки “Для лікарів” та “Адміністрація пацієнтів”, які, ймовірно, ведуть до розділів для лікарів та адміністрування пацієнтів.
- Пошук: головна частина містить пошуковий рядок з текстом-заповнювачем “ Записуйтесь на прийом не виходячи з дому ” та синю кнопку “Пошук”.
- Фон: фон скріншоту - це градієнт від фіолетового у верхній частині до світло-блакитного у нижній.

Це інтерфейс для зручного пошуку та запису на медичні консультації онлайн.

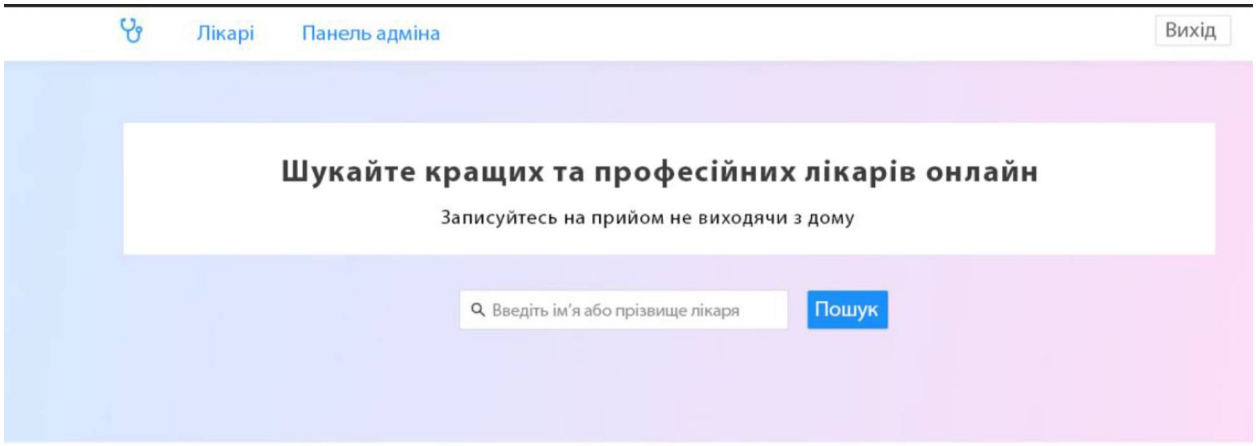


Рис 3.10 – Home page

На рис. 3.11 зображено сторінку web-системи запису на прийом до лікаря. Ось ключові деталі:

- Назва сторінки: “Список лікарів”.
- Навігаційна панель: зверху є опції, такі як ‘Лікарі’ та ‘Адміністративна панель’.
- Кнопка додавання: є кнопка ‘Додати лікаря’.
- Таблиця: нижче є таблиця з колонками для ‘Прізвище’, ‘Ім’я’, ‘Телефон’, ‘E-mail’ та ‘Спеціалізація/категорія’, а також ‘Видалення’.
- Записи: у таблиці є три записи з відповідними деталями та опцією видалення кожного запису.

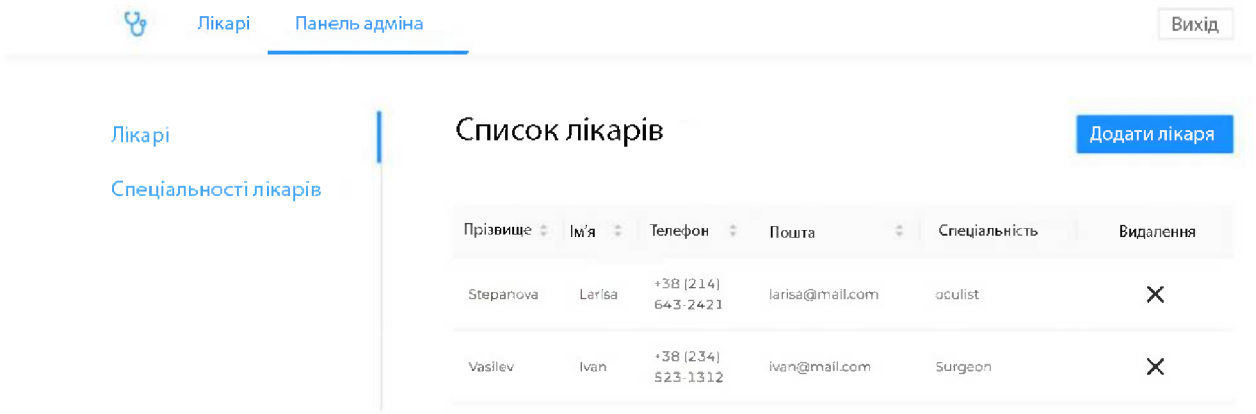


Рис 3.11 – Панель адміна

На рис. 3.12 зображено веб-сторінку з переліком медичних фахівців-хірургів, їхніми іменами та вартістю консультації. Ось ключові деталі:

- Профілі: є три профілі хірургів, кожен з фотографією та вказаною вартістю консультації.
- Фільтри: на сторінці встановлені фільтри для вибору вартості від 500 до 5000 гривень та спеціалізації “Хірург”.
- Пошук: у верхньому правому куті є пошукова строка для здійснення пошуку на сайті.

Ця інформація може бути корисною для осіб, які шукають інформацію про медичних фахівців, зокрема хірургів, їх імена та вартість консультацій.

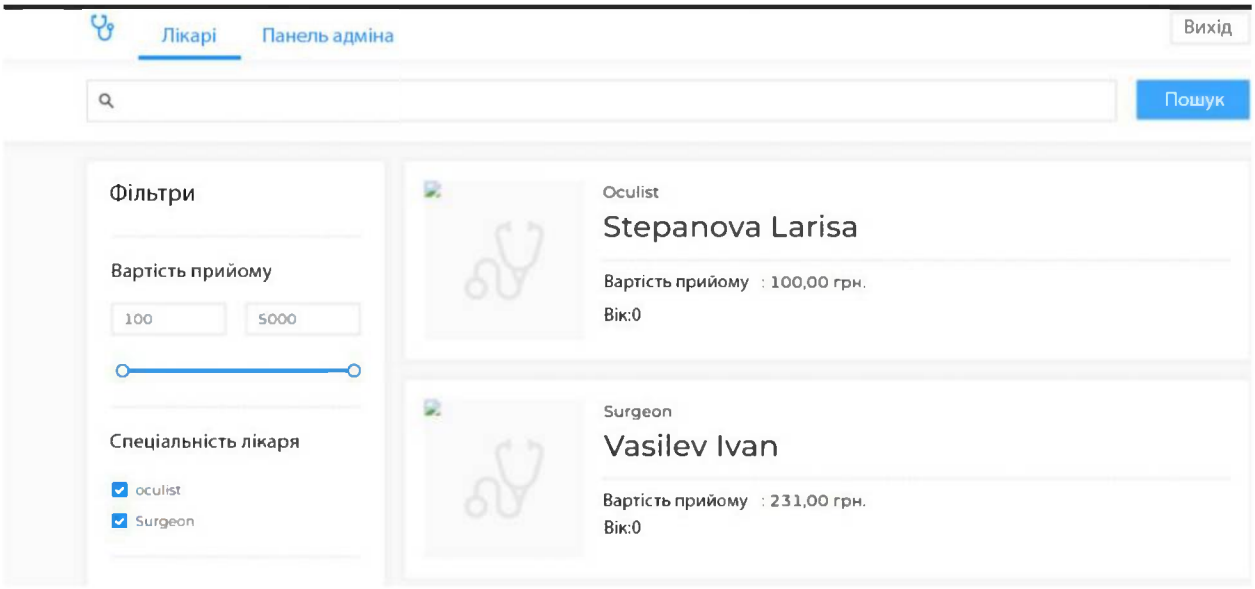


Рис 3.12 – Список лікарів

На рис. 3.13 зображена веб-сторінка, яка відображає систему бронювання медичних прийомів. Є два варіанти запису на прийом до хірургів: “Степанова Лариса” та “Василєв Іван”. Кожен варіант включає вартість прийому та синю кнопку “Записатися”. Вартість прийому з Василєвим Іваном - 231,00 грн. Інтерфейс має фільтри вгорі для вибору різних критеріїв, таких як діапазон цін та спеціальність. На зображенні не присутні математичні завдання чи домашні завдання.

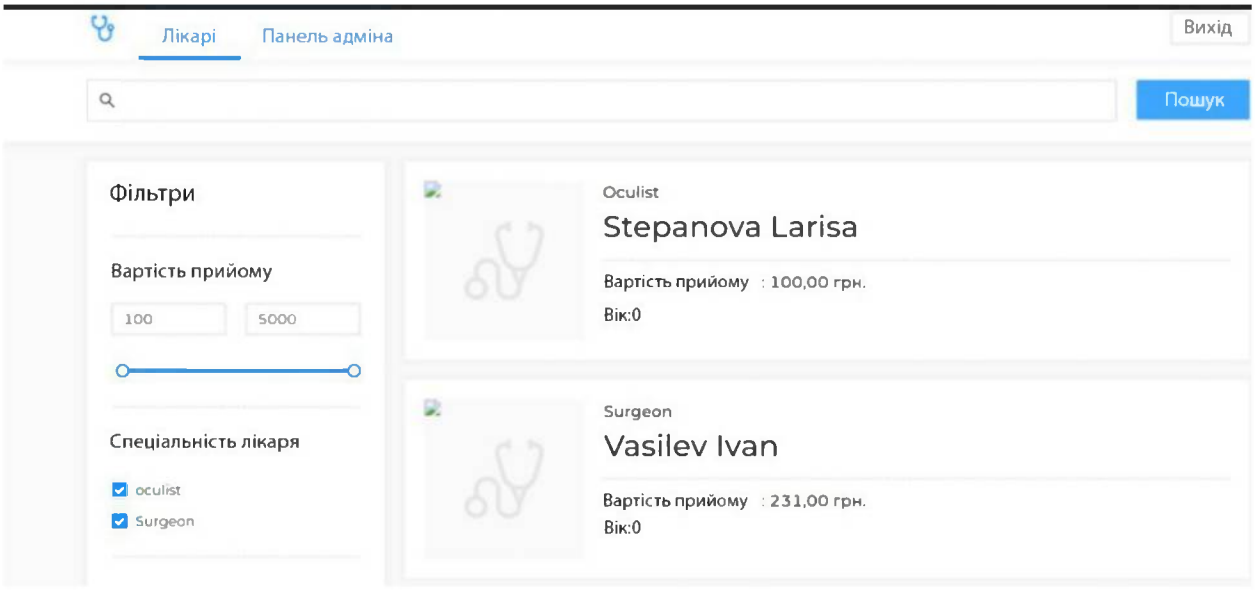


Рис 3.13 – Інтерфейс користувача

На рис. 3.14 зображено веб-інтерфейс для системи бронювання прийомів. Він показує форму з заповненими полями для запису на прийом. Дата встановлена на “13 червня 2024 р.”, час з “06:00 до 06:00”, а ім’я лікаря - “Степанова Лариса”. На дні є кнопка, яка говорить “Скасувати запис”.

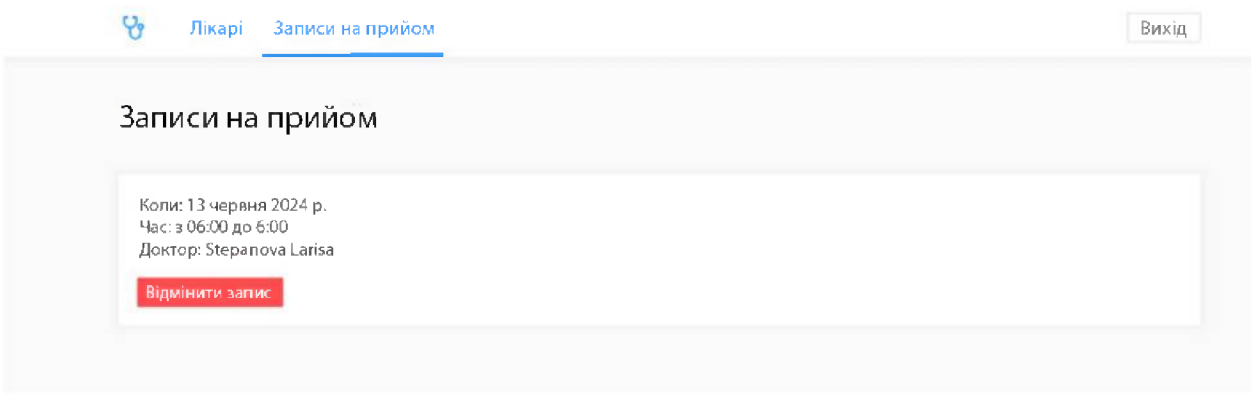


Рис 3.14 – приклад запису на прийом

На рис. 3.15 зображено інтерфейс входу для облікового запису на вебсайті. Є такі опції, як “Вхід”, “Авторизація” та “Реєстрація” у верхньому правому куті. Основна частина екрану показує форму з двома полями: одне з позначкою “Пошта *” з прикладом електронної адреси “admin@mail.com” та інше з позначкою “Пароль *” з точками, що позначають прихований пароль.

Під полем для пароля є опція “Запам’ятайте мене” з прапорцем, а нижче - синя кнопка з написом “Увійти зараз”.

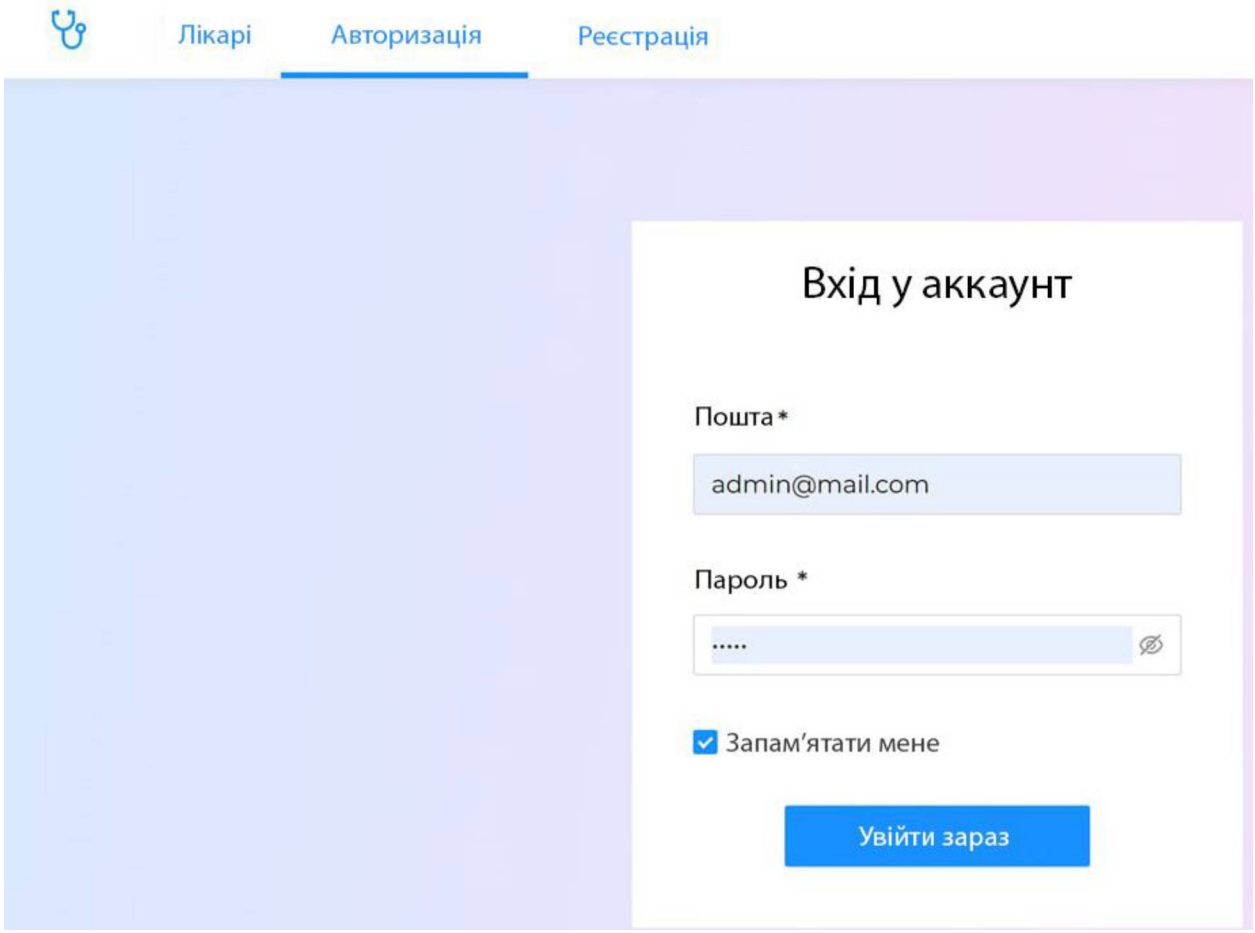


Рис 3.15 – Авторизація адміна в системі

На скріншоті зображено веб-інтерфейс медичної бази даних або системи планування прийомів, який використовується для управління інформацією про лікарів. Ось основні елементи:

- Пошукова панель: у верхній частині є пошукова панель з написом “Лікарі” та вкладка адміністративної панелі.
- Таблиця лікарів: нижче розташована таблиця з назвою “Список лікарів”, де вказані спеціалізація, ім’я, телефон, електронна пошта, категорія спеціалізації та можливість видалення.
- Інформація про лікарів: у таблиці є записи чотирьох лікарів з їх контактною інформацією та спеціалізаціями, такими як хірург або

окуліст.

- Кнопка додавання: праворуч є кнопка “Додати лікаря”.

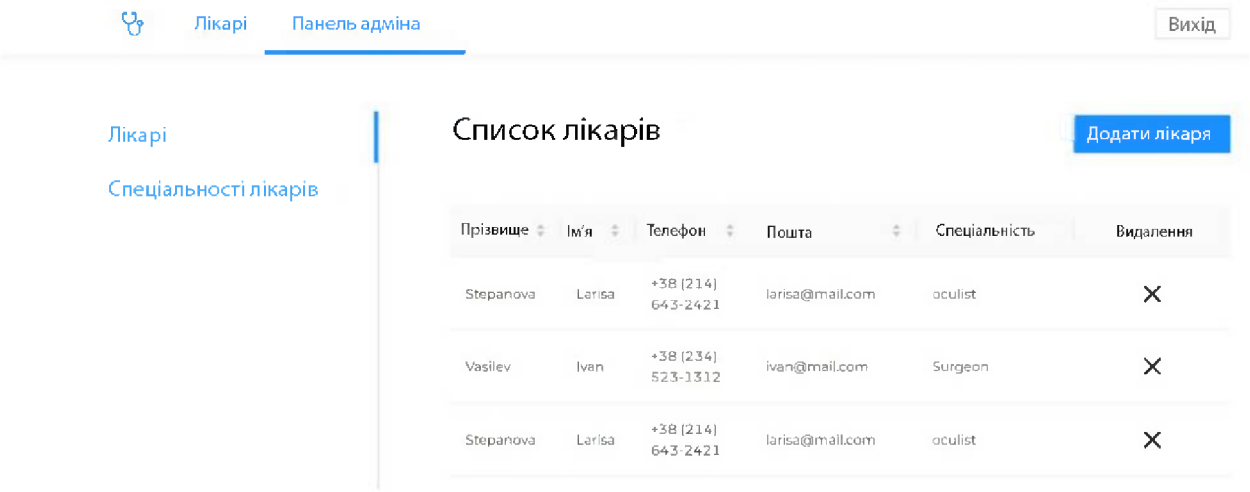


Рис 3.16 – Адмін здійснює пошук врачів

На рис. 3.17 зображено цифрову візитку або картку контактної інформації. Вона містить ім’я “Kilabotsu Alexander”, номер телефону “+38 (213) 312-3123” та електронну адресу “alexander@gmail.com”. Також є слово “окуліст”, яке може вказувати на професію особи. З правого боку є символ ‘X’, який зазвичай представляє кнопку для закриття або виходу.

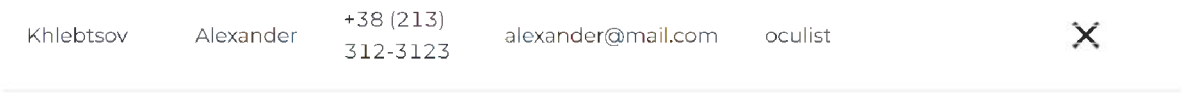


Рис 3.17 – Створений акаунт окуліста

На рис. 3.18 зображено інтерфейс користувача, який містить розділ “Спеціальності лікарів” з переліком трьох спеціальностей: терапевт, окуліст і хірург. Кожна спеціальність має опцію для видалення, позначену хрестиком ‘X’.

Рис 3.18 – Адмін додав нову спеціальність therapist

Висновки за розділом 3

У цьому розділі було проведено розробку та тестування веб-системи онлайн-поліклініки. Було описано процес проєктування та розробки системи, наведено приклади SQL-запитів для взаємодії з базою даних, а також описано тестування графічного інтерфейсу користувача та функціональності системи. Результати показали готовність системи до використання з можливими рекомендаціями щодо подальшого вдосконалення.

ВИСНОВКИ

У ході виконання кваліфікаційної роботи була розроблена та впроваджена інформаційна система для онлайн-поліклініки, що значно полегшило процес надання медичних послуг та взаємодії з пацієнтами.

Початковим етапом проекту був аналіз предметної області реєстратури медичних закладів, що дозволив чітко визначити основні бізнес-процеси та бізнес-функції, які необхідно було врахувати при проектуванні системи. На основі цього аналізу було створено базу даних, що забезпечує зберігання та обробку всієї необхідної інформації.

Для розробки серверної та клієнтської частин інформаційної системи використовувалися сучасні технології та інструменти програмування. Серверна частина була написана мовою програмування C#, а для зберігання та обробки даних у базі даних використовувався інструмент Entity Framework. Клієнтська частина системи розроблялася з використанням мови програмування Typescript і представляє собою зручний графічний інтерфейс у вигляді single page application, що надає користувачам можливість використовувати усі функції системи зручним та інтуїтивно зрозумілим способом.

Завдяки розробленій інформаційній системі пацієнти можуть легко та зручно записуватись на прийом до лікарів через веб-додаток, користуючись будь-яким пристроєм з доступом до Інтернету. Крім того, система дозволяє рівномірно розподіляти навантаження між фахівцями, що підвищує ефективність роботи медичного закладу в цілому.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Даманаков, М. І. (2018). "Проектування та розробка веб-сайтів на платформі WordPress". Київ: Видавництво "Альтернативні шляхи".
2. McFarland, D. (2017). "HTML and CSS: Design and Build Websites". John Wiley & Sons.
3. Duckett, J. (2014). "JavaScript and JQuery: Interactive Front-End Web Development". John Wiley & Sons.
4. Krug, S. (2014). "Don't Make Me Think, Revisited: A Common Sense Approach to Web Usability". New Riders.
5. Sebesta, R. W. (2014). "Programming the World Wide Web". Pearson.
6. Simpson, K. (2014). "Responsive Web Design with HTML5 and CSS3". Packt Publishing.
7. Zeldman, J., & Marcotte, E. (2010). "Designing with Web Standards". New Riders.
8. Robbins, J. N. (2016). "Learning Web Design: A Beginner's Guide to HTML, CSS, JavaScript, and Web Graphics". O'Reilly Media.
9. Freeman, A., & Sanderson, R. (2016). "Pro ASP.NET MVC 5". Apress.
10. Galloway, J., & Wilson, B. (2017). "Professional C# 7 and .NET Core 2.0". Wrox.
11. Dubois, P. (2013). "MySQL Cookbook: Solutions for Database Developers and Administrators". O'Reilly Media.
12. Albahari, J., Albahari, B., & Torgersen, M. (2017). "C# 7.0 in a Nutshell: The Definitive Reference". O'Reilly Media.
13. Kauffman, D., & Kabaker, A. (2017). "ASP.NET Core Application Development: Building an application in four sprints". Manning Publications.
14. McLaughlin, M. (2019). "ASP.NET Core 3 and Angular 9: Full-Stack Web Development with .NET Core 3.0 and Angular". Packt Publishing.
15. Sellar, M. (2016). "Learning MySQL and MariaDB: Heading in the Right Direction with MySQL and MariaDB". O'Reilly Media.

ДОДАТКИ

Додаток А

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Харківський національний університет імені В. Н. Каразіна

Факультет комп'ютерних наук
Кафедра теоретичної та прикладної системотехніки
Рівень вищої освіти (освітньо-кваліфікаційний рівень) **бакалавр**
галузь знань:**15 – Автоматизація та приладобудування**
спеціальність:**151 – Автоматизація та комп'ютерно-інтегровані технології**

ЗАТВЕРДЖУЮ
Завідувач кафедри теоретичної
та прикладної системотехніки
_____д.т.н., проф. Шматков С. І.
«21» грудня 2023 року

З А В Д А Н Н Я
НА КВАЛІФІКАЦІЙНУ РОБОТУ

Лофіцької Анастасії Олексіївна

(прізвище, ім'я, по батькові студента)

1. Тема роботи «WEB-система онлайн-поліклініки»

керівник роботи Павлов Анатолій Миколайович, старший викладач
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджені наказом по університету від «03» травня 2024року № 4101-5/909

2.Строк подання студентом роботи 31 травня 2024року

3. Перелік питань, які потрібно розробити

1) Аналіз способів побудови WEB-систем.
2) Аналіз різновидів типів сайтів для онлайн-поліклініки.
3) Обґрунтування використання технології ASP.NET MVC і мови програмування C # та реляційної системи управління базами даних MySQL.
4) Розробка та тестування WEB системи онлайн-поліклініки.

4. План роботи

№ з/п	Назви етапів роботи	Термін виконання етапів роботи
1	Підбір та аналіз наукової літератури	21.12.2023 - 25.01.2024
2	Аналіз способів побудови WEB-систем та типів сайтів для онлайн-поліклініки	19.12.2023 - 2.01.2024
3	Вибір мови програмування та баз даних	2.01.2024 - 2.02.2024
4	Розробка сайту онлайн-поліклініки	2.01.2024 - 2.02.2024
5	Тестування комп'ютерної моделі	3.02.2024 - 30.03.2024
6	Підготовка тез доповіді на семінар	3.03.2024 - 30.04.2024
7	Розробка пояснювальної записки.	31.03.2024 - 27.05.2024
8	Оформлення звіту за результатами переддипломної практики	15.05.2024 – 31.05.2024
9	Представлення кваліфікаційної роботи керівнику та рецензенту	31.05.2024
10	Оформлення пояснювальної записки та підготовка презентації	31.05.2024

5. Дата видачі завдання 21.12.2023

Студент Лофіцька А. О. _____
ініціали, прізвище підпис

Керівник роботи Павлов А. М. _____
ініціали, прізвище підпис

Додаток Б

Затверджую

_____

«___» _____ 2023 р.

Технічне завдання
на розробку програмного виробу «WEB-система онлайн-поліклініки»

1.	Введення	1.1. Назва: WEB-система онлайн-поліклініки 1.2. Галузь застосування: Інформаційні технології
2.	Підстава для розробки	2.1. Навчальний план за спеціальністю 151 – Автоматизація та комп'ютерно-інтегровані технології 2.2. Завдання на кваліфікаційну роботу бакалавра №4101-5/895 від «23» травня 2024 (представити як Додаток А до пояснювальної записки до кваліфікаційної роботи).
3.	Призначення розробки	3.1. Мета розробки: розробка клієнтської і серверної частин інформаційної системи реєстратури. 3.2. Призначення розробки: створення платформи, яка забезпечує зручний та ефективний доступ до медичних послуг через інтернет. Також запис на прийом до лікаря, консультації онлайн, моніторинг стану здоров'я, управління медичним персоналом 3.3. Вихідні дані розробки: веб-сайт онлайн-поліклініки; вхідні дані розробки: персональн дані пацієнта.
4.	Технічні вимоги до програмного виробу	4.1. Вимоги до функціональних характеристик: реєстрація користувачів, управління медичними записами, онлайн-запис на прийом, нагадування про прийоми, безпека даних. Також приємний інтерфейс для взаємодії користувача із сайтом. 4.2. Вимоги до надійності: забезпечення безперебійної роботи програмного виробу при будь-яких вимогах користувача в рамках призначення виробу . 4.3.Вимоги до умов експлуатації: немає 4.4. Вимоги до складу і параметрів технічних засобів: для виконання програми повинен підходити ПК із

<table>
<tr><td></td><td></td><td colspan="2">будь-якою операційною системою сімейства Windows, Linux/Unix, Mac OS.
4.5. Вимоги до інформаційної та програмної сумісності: підтримка ОС Linux або Windows 11, підтримка мови програмування, підтримка різних платформ.
4.6. Вимоги до маркування та упаковки: вимоги до маркування та упакування не представляються.
4.7. Вимоги до транспортування і зберігання: вимоги до транспортування та зберігання не представляються.
4.8. Спеціальні вимоги: спеціальні вимоги до програмного виробу не пред'являються.</td></tr>
<tr><td>5.</td><td>Вимоги до програмної документації</td><td colspan="2">Програмною документацією до виробу «Метод аналізу інформативності змінних стану при діагностиці систем з використанням інформаційних критеріїв» вважати:
1) Справжнє Технічне завдання на розробку виробу (представити у вигляді Додатку Б до пояснювальної записки до кваліфікаційної роботи).
2) Методику розрахунку інформативності змінних стану (у вигляді глав 3.2 та 3.3 пояснювальної записки до кваліфікаційної роботи).
3) Опис виробу (представити в розділі 3 пояснювальної записки до кваліфікаційної роботи)</td></tr>
<tr><td>6.</td><td>Вимоги до техніко-економічних показників</td><td colspan="2">Програмною документацією до виробу «WEB-
система онлайн-поліклініки» вважати:
1) Справжнє Технічне завдання на розробку виробу (представити у вигляді Додатку Б до пояснювальної записки до кваліфікаційної роботи).
2) Опис програмного виробу (представити в Розділі 3
пояснювальної записки до кваліфікаційної роботи).
3) Джерела базової інформації.</td></tr>
<tr><td rowspan="2">7.</td><td rowspan="2">Стадії і етапи розробки</td><td>Дата</td><td>Назва етапу</td></tr>
<tr><td>від 21 грудня 2023
до 25 січня 2024

від 19 грудня 2023
до 2 січня 2024

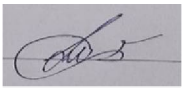
від 2 січня 2024
до 2 лютого 2023</td><td>Підбір та аналіз наукової літератури

Аналіз способів побудови WEB-систем та типів сайтів для онлайн-поліклініки</td></tr>
</table>

		від 2 січня 2024 до 2 лютого 2024 від 3 лютого 2024 до 30 березня 2024 від 3 березня 2024 до 30 квітня 2024 від 31 березня 2024 до 27 травня 2024 від 15 травня 2024 до 31 травня 2024 31 травня 2024 31 травня 2024	Вибір мови програмування та баз даних Розробка сайту онлайн-поліклініки Тестування комп'ютерної моделі Підготовка тез доповіді на семінар Розробка пояснювальної записки. Оформлення звіту за результатами переддипломної практики. Представлення кваліфікаційної роботи керівнику та рецензенту Оформлення пояснювальної записки та підготовка презентації
8.	Порядок контролю і приймання програмного продукту (моделі)	1. Перевірку ходу розробки програми виконувати раз в 3 тижні. 2. Захист розробленої моделі провести на засіданні Атестаційної комісії. 3. Пояснювальну записку подати на паперових носіях в 1 примірнику.	

Виконавець
студентка групи КУ- 41
Лофіцька А. О.

Замовник
старший викладач
Павлов А. М.



Додаток В

Програма і методика випробувань програмного виробу

«WEB-система онлайн-поліклініки»

1 Об'єкт випробувань

1. Назва програмного виробу : «WEB-система онлайн-поліклініки»
2. Галузь застосування : Інформаційні технології
3. Перераховані відомості запозичуються з відповідних розділів Технічного завдання.

2. Мета випробувань

Перевірка відповідності функціональності програмної реалізації системи заявленим функціональним можливостям в технічному завданні (Додаток Б до пояснювальної записки до кваліфікаційної роботи).

3. Загальні положення

1. Підстави для проведення випробувань

Підставою для проведення випробувань є наказ про призначення атестаційної комісії.

2. Місце і тривалість випробувань

Приймальні (приймально-здавальні) випробування проводяться на базі комп'ютерного класу кафедри в період роботи атестаційної комісії.

3. Обсяг випробувань

Приймальні випробування програмного вироби проводяться в обсязі відповідному цієї програми і методики випробувань.

4. Організації, які беруть участь у випробуваннях

Приймальні випробування проводяться атестаційною комісією напередодні засідання (або в процесі засідання) за участю Замовника, Виконавця та інших осіб, присутніх на засіданні.

4. Вимоги до програми або програмного виробу

Модель повинна задовольняти наступним вимогам:

4.1. Вимоги до функціональних характеристик: реєстрація користувачів, управління медичними записами, онлайн-запис на прийом, нагадування про прийоми, безпека даних. Також приємний інтерфейс для взаємодії

користувача із сайтом.
4.2. Вимоги до надійності: забезпечення безперебійної роботи програмного виробу при будь-яких вимогах користувача в рамках призначення виробу .
4.3.Вимоги до умов експлуатації: немає
4.4. Вимоги до складу і параметрів технічних засобів: для виконання програми повинен підходити ПК із будь-якою операційною системою сімейства Windows, Linux/Unix, Mac OS.
4.5. Вимоги до інформаційної та програмної сумісності: підтримка ОС Linux або Windows 11, підтримка мови програмування, підтримка різних платформ.
4.6. Вимоги до маркування та упаковки: вимоги до маркування та упакування не представляються.
4.7. Вимоги до транспортування і зберігання: вимоги до транспортування та зберігання не представляються.
4.8. Спеціальні вимоги: спеціальні вимоги до програмного виробу не пред'являються.

5. Вимоги до програмної документації

Документацією до виробу «WEB-система онлайн-поліклініки» вважати:

1) Документація по мові програмування та додаткові мануали.

2) Програму і методику випробувань розробленої програми (представити як Додаток В до пояснювальної записки до кваліфікаційної роботи).

3) Опис програмного виробу (представити в Розділі 3 пояснювальної записки до кваліфікаційної роботи).

4) Джерела базової інформації.

6. Засоби і порядок випробувань

6.1 Засоби випробувань

Засоби випробувань представлено на ПК на яких встановлено наступні програмні засоби: інтерпретатор мови програмування.

6.2 Порядок проведення випробувань

Як правило, випробування проводяться в два етапи:

- ознайомчий (1-й етап);
- власне випробування програмного виробу (2-й етап).

Перелік перевірок, що проводяться на 1 етапі випробувань, включає в себе:

1) Перевірку комплектності складу програмної документації здійснюється за критерієм наявності зазначеної в ТЗ документації.

2) Перевірку якості програмної документації. Перевірку здійснювати за критерієм відповідності вимогам ГОСТ 19.301-79 ЕСПД. «Програма і методика випробувань».

Перелік перевірок, що проводяться на 2 етапі випробувань, включає в себе:

1) Перевірку відповідності технічних характеристик програми вимогам технічного завдання.

2) Перевірку ступеня виконання функціональних вимог до програми.

3) Методику проведення перевірок:

a) Запустити програмне забезпечення.

b) Порядок проведення випробувань:

– Зробити налаштування.

– Перевірити чи працює програма.

– Перевірити чи формується звіт.

4) Якщо перевірки на першому та другому етапах виконано успішно, то виріб вважається таким, що пройшов випробування.

Для проведення випробувань пропонується тест 1, тест 2 та тест 3.

Тест 1

1. Перевірка виконання програми;
2. Адміністратор здійснює пошук лікарів;
3. Отримання відповіді серверу.

Рис. В.1 Тест 1

Тест 2

1. Перевірка виконання програми
2. Головна сторінка;
3. Отримання результатів.

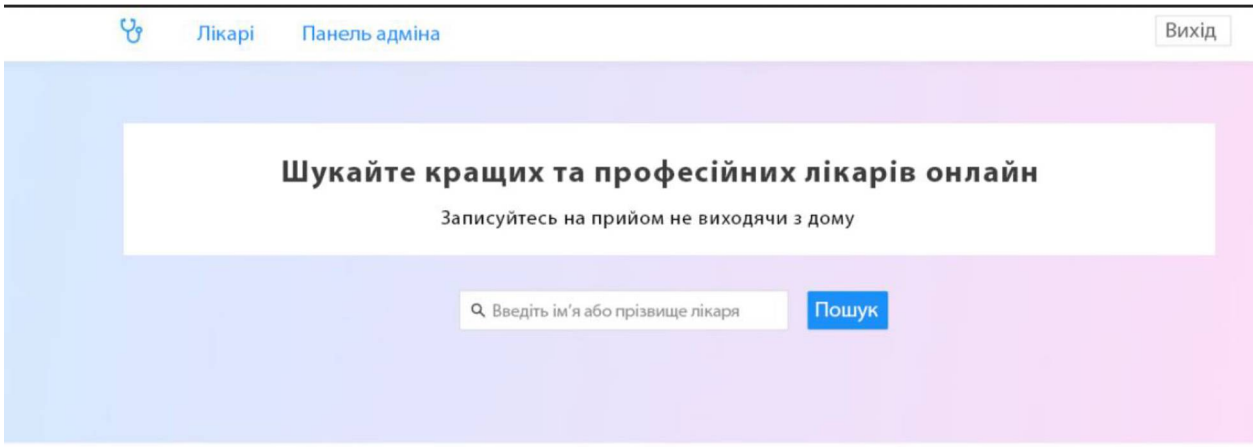


Рис. В.2 Тест 2

Тест 3

4. Перевірка виконання програми
5. Запис пацієнта на прийом до лікаря;
6. Отримання результату.

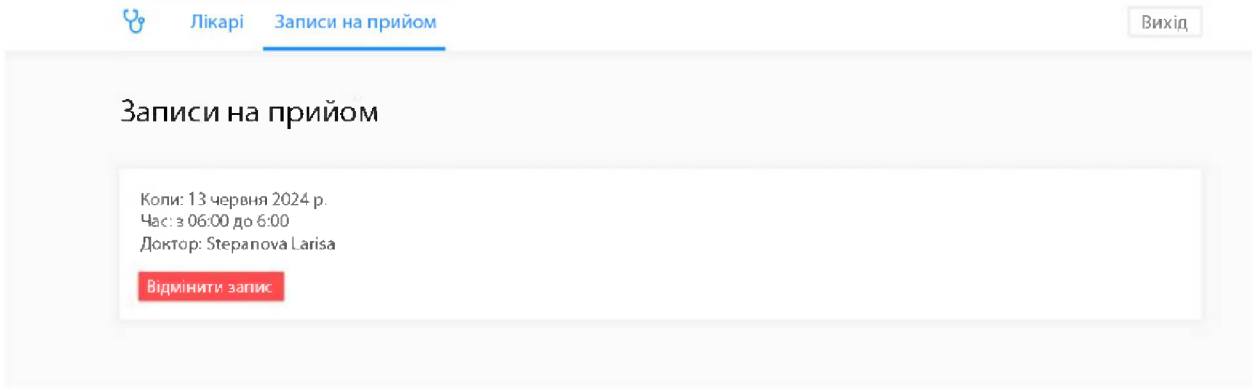


Рис. В.3 Тест 3

Тест вважається пройденим, якщо відбуваються вказані операції і їх відображення у програмному продукті.

Висновки: тест 1 успішно пройшов випробування, тест 2 успішно пройшов випробування і тест 3 успішно пройшов випробування. Випробування пройшло успішно.

Виконавець: студентка групи КУ-41, Лофіцька А. О.

Додаток Г

Лістинг коду

```
using Anastasia.Doctors.Api.ConfigurationModels;
using Anastasia.Doctors.Data.Models;
using Microsoft.IdentityModel.Tokens;
using System;
using System.Collections.Generic;
using System.IdentityModel.Tokens.Jwt;
using System.Security.Claims;
using System.Text;

namespace Anastasia.Doctors.Api.Authorization
{
   public static class JWTHelper
   {
      public static SecurityKey CreateTokenSignInKey(string secretString)
      {
         var securityKeyBytes = Encoding.UTF8.GetBytes(secretString);
         return new SymmetricSecurityKey(securityKeyBytes);
      }

      public static SigningCredentials CreateTokenSignInCredential(string
secretString)
      {
         return new SigningCredentials(CreateTokenSignInKey(secretString),
SecurityAlgorithms.HmacSha256);
      }
```

```
        public static string CreateTokenJson(User user, JWTConfig jwtConfig)
        {
            var claims = new List<Claim>
            {
                new Claim(JwtRegisteredClaimNames.Sub, user.Id.ToString()),
                new Claim("role", user.Role.ToString())
            };

            var token = new JwtSecurityToken(jwtConfig.Issuer, jwtConfig.Audience,
claims, DateTime.Now, DateTime.Now.AddDays(1),
CreateTokenSignInCredential(jwtConfig.TokenSecurityKey));
            var tokenJson = new JwtSecurityTokenHandler().WriteToken(token);

            return tokenJson;
        }
    }
}

using Anastasia.Doctors.Data.Models;
using Microsoft.EntityFrameworkCore;
using System;

namespace Anastasia.Doctors.Data
{
    public class DoctorsDbContext : DbContext
    {
        public DoctorsDbContext(DbContextOptions<DoctorsDbContext> options) :
base(options) { }
```

```
public DbSet<Doctor> Doctors { get; set; }
public DbSet<Patient> Patients { get; set; }
public DbSet<Record> Records { get; set; }
public DbSet<User> Users { get; set; }
public DbSet<Specialty> Specialties { get; set; }

protected override void OnModelCreating(ModelBuilder modelBuilder)
{
    base.OnModelCreating(modelBuilder);

    ConfigureRelations(modelBuilder);
    ConfigureData(modelBuilder);
    modelBuilder.Entity<User>()
            .Property(u => u.Role)
            .HasConversion<string>();
}

private void ConfigureRelations(ModelBuilder modelBuilder)
{
    modelBuilder.Entity<Doctor>()
            .HasMany(d => d.Specialties)
            .WithMany(s => s.Doctors);

    modelBuilder.Entity<Doctor>()
            .HasOne(d => d.Identity)
            .WithOne(i => i.Doctor)
```

```
            .HasForeignKey<Doctor>(d => d.Id);

    modelBuilder.Entity<Patient>()
            .HasOne(p => p.Identity)
            .WithOne(i => i.Patient)
            .HasForeignKey<Patient>(p => p.Id);

    modelBuilder.Entity<Record>()
            .HasKey(r => new { r.DoctorId, r.PatientId });

    modelBuilder.Entity<Record>()
            .HasOne(r => r.Doctor)
            .WithMany(d => d.Records)
            .HasForeignKey(r => r.DoctorId)
            .OnDelete(DeleteBehavior.NoAction);

    modelBuilder.Entity<Record>()
            .HasOne(r => r.Patient)
            .WithMany(p => p.Records)
            .HasForeignKey(r => r.PatientId)
            .OnDelete(DeleteBehavior.NoAction);
}

private void ConfigureData(ModelBuilder modelBuilder)
{
    modelBuilder.Entity<User>().HasData(new User[]
    {
```

```
            new User
            {
                Id = Guid.NewGuid(),
                Email = "admin@mail.com",
                Password = "admin",
                FirstName = "Admin",
                LastName = "Admin",
                DateOfBirth = DateTime.Now,
                Role = UserRole.Admin
            }
        });

        modelBuilder.Entity<Specialty>().HasData(new Specialty[]
        {
            new Specialty
            {
                Id = Guid.NewGuid(),
                Name = "Surgeon"
            }
        });
    }
  }
}

ul,
ol {
  padding: 0;
  margin: 0;
```

```
}

body,
h1,
h2,
h3,
h4,
p,
li,
figure,
figcaption,
blockquote,
dl,
dd {
  margin: 0;
}

[tabindex="-1"]:focus:not(:focus-visible) {
  outline: 0 !important;
}

pre,
code,
kbd,
samp {
  font-family: SFMono-Regular, Menlo, Monaco, Consolas, "Liberation Mono",
"Courier New", monospace;
  font-size: 1em;
```

```
}

body {
    min-height: 100vh;
    scroll-behavior: smooth;
    text-rendering: optimizeSpeed;
    line-height: 1.5;
}

ul,
ol {
    list-style: none;
}

a:not([class]) {
    text-decoration-skip-ink: auto;
}

img {
    max-width: 100%;
    display: block;
    border-style: none;
}

article>*+* {
    margin-top: 1em;
}
```

```
/* Задаём кнопкам нужный тип курсора */
button:not(:disabled),
[type="button"]:not(:disabled),
[type="reset"]:not(:disabled),
[type="submit"]:not(:disabled) {
   cursor: pointer;
}

textarea {
   overflow: auto;
   resize: vertical;
}

input,
button,
textarea,
select {
   font: inherit;
}

 [hidden] {
   display: none !important;
}

@media (prefers-reduced-motion: reduce) {
   * {
```

```
        animation-duration: 0.01ms !important;
        animation-iteration-count: 1 !important;
        transition-duration: 0.01ms !important;
        scroll-behavior: auto !important;
    }
}
```