КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
ІМЕНІ ТАРАСА ШЕВЧЕНКА
Факультет комп'ютерних наук та кібернетики
Кафедра теоретичної кібернетики

Кваліфікаційна робота
на здобуття ступеня бакалавра

за спеціальністю 122 Комп'ютерні науки

на тему:

РОЗРОБКА ВЕБ-САЙТУ ТА ІНФОРМАЦІЙНОЇ СИСТЕМИ СТОМАТОЛОГІЧНОЇ ПОЛІКЛІНІКИ

Виконав студент 4-го курсу
Гліб СТРАТІЙ _____
(підпис)

Науковий керівник:
доцент, кандидат фіз.-мат. наук
Андрій СТАВРОВСЬКИЙ

_____
(підпис)

Засвідчую, що в цій роботі немає запозичень з праць інших авторів без відповідних посилань.
Студент _____
(підпис)

Роботу розглянуто й допущено до захисту на засіданні кафедри теоретичної кібернетики

«____»_____2023 р., протокол № ____

Завідувач кафедри

Юрій КРАК _____

(підпис)

Київ - 2023

РЕФЕРАТ

Обсяг роботи 65 сторінок, 29 ілюстрацій, 12 джерел посилань.

ПОЛІКЛІНІКА, ВЕБ ДОДАТОК, ВЕБ САЙТ, MERN STACK, JAVASCRIPT

Об’єктом роботи є створення веб-сайту. Предметом роботи є реалізація веб-додатка стоматологічної поліклініки на базі Javascript за допомогою MERN стеку.

Метою роботи є розроблення та програмна реалізація веб-сайту для надання можливості перегляду послуг, лікарів клініки, а також для можливості запису до лікаря.

Інструменти розроблення: інтегроване середовище розробки: VsCode, MongoDB Atlas, Create React App, Node.js, Express.

Результати роботи: виконано детальний огляд технологій стеку MERN, розроблено логіку реалізації додатка та виконано його базовий функціонал. Реалізовано client-server модель, а також порівняно її з іншими архітектурними рішеннями.

Розроблене програмне забезпечення може використовуватись медичними закладами для надання та просування своїх послуг. Може бути додатково змінено та використано в усіх галузях, де існує потреба запису до спеціаліста.

ЗМІСТ

СКОРОЧЕННЯ ТА УМОВНІ ПОЗНАЧЕННЯ

API (англійська абревіатура від "application programming interface") - це інтерфейс, який надає програмі можливість взаємодіяти з іншими програмами або компонентами.

CSRF (англ. Cross-Site Request Forgery) - це атака, при якій зловмисник намагається виконати підроблений запит від імені авторизованого користувача на іншому веб-сайті.

CORS (англ. Cross-Origin Resource Sharing) - це механізм, який дозволяє веб-браузерам обмінюватись ресурсами між різними джерелами (доменами) на веб-сторінках.

MERN стек (англ. MongoDB, Express.js, React, and Node.js) - набір технологій, який використовується для створення веб-додатків та веб-сайтів.

IDE (англ. Integrated Development Environment) - комплексне середовище розробки, яке надає інтерфейс та інструменти для зручного програмування та створення програмних продуктів.

HTML (англ. Hyper Text Markup Language) - мова, яка використовується для створення структури та розмітки гіпертекстових документів, що включають веб-сторінки.

HTTP (англ. Hyper Text Transfer Protocol) - стандартний протокол, що використовується для обміну даними між комп'ютерами через комп'ютерні мережі, зокрема веб-серверами та веб-браузерами.

HTTPS (англ. Hyper Text Transfer Protocol over Secure Socket Layer) - безпечна схема URI, що надає шифровану передачу даних між клієнтами та серверами в Інтернеті. Вона є розширенням звичайного протоколу HTTP і використовує шар захисту (Secure Socket Layer) або його сучасну версію - TLS (Transport Layer Security), що забезпечує конфіденційність та цілісність передачі інформації.

JSON (англ. JavaScript Object Notation) - легкий формат обміну даними, що використовується для зберігання та передачі структурованої інформації. Його основа базується на синтаксисі JavaScript і використовується для представлення об'єктів, масивів, рядків, чисел, булевих значень та null, що дозволяє легко обробляти та інтегрувати дані між різними системами та мовами програмування.

BSON (англ. Binary JSON) - бінарний формат обміну даними, заснований на JSON. Він розширює функціональність JSON, дозволяючи зберігати дані у бінарному вигляді, що забезпечує компактніше представлення, швидку серіалізацію та десеріалізацію, а також підтримку додаткових типів даних, таких як дата, бінарні дані та інші.

ES6 (англ. ECMAScript 6) - стандартна версія мови програмування ECMAScript, яка включає в себе розширення та покращення функціональності JavaScript.

CSS (англ. Cascading Style Sheets) - мова стилізації, що використовується для опису зовнішнього вигляду веб-сторінок. Вона дозволяє визначати розміщення, кольори, шрифти, фони, анімацію та інші візуальні аспекти елементів на веб-сторінці.

Client-server - модель розподіленої системи, в якій функціональність розподілена між двома типами компонентів: клієнтськими додатками, що надають інтерфейс користувача, та серверами, що забезпечують обробку та збереження даних, відповіді на запити та надання послуг.

MVC (англ. Model View Controller) - архітектурний шаблон програмування, який забезпечує розділення логіки програми на три основні компоненти: модель (Model), представлення (View) та контролер (Controller).

MVP (англ. Minimum Viable Product) - стратегічний підхід до розробки продукту, що передбачає створення мінімально функціональної версії продукту з набором ключових функцій і властивостей.

SPA (Single Page Application) - це веб-додаток, який працює на одній сторінці, без перезавантаження всього вмісту. Він завантажується один раз при запуску і динамічно оновлює лише частину сторінки при взаємодії користувача, забезпечуючи більш плавний та інтерактивний досвід.

Create React App (CRA) є інструментом, що допомагає швидко створювати нові проекти на React. Він забезпечує початкову налаштування та структуру проекту, що дозволяє розробникам швидко почати писати код без необхідності вручну налаштовувати інфраструктуру React додатків.

ВСТУП

З розвитком глобальної комп'ютерної мережі Internet, веб-розробка пройшла довгий шлях. Початково розробники створювали окремі програми для різних операційних систем, які потрібно було встановлювати локально на комп'ютері користувача. Такі програми називалися десктоп-додатками. Однак з появою веб-додатків, користувачам стало значно зручніше, оскільки вони могли отримати доступ до додатків безпосередньо через веб-браузер, незалежно від операційної системи.

Розвиток веб-додатків привів до появи сучасних, інтерактивних та інтелектуальних веб-додатків, в яких користувачі можуть взаємодіяти з сервером, слухати аудіо, переглядати відео та навіть малювати на екрані. Розробка веб-сайтів включає як фронтенд, так і бекенд розробку. Раніше для повноцінної веб-розробки використовувався стек LAMP (Linux, Apache, MySQL і PHP), де PHP використовувався для бекенду, а HTML, CSS і JavaScript - для фронтенду, а MySQL - для баз даних. Однак з появою Node.js, який дозволяє використовувати JavaScript як на фронтенді, так і на бекенді, розробники отримали змогу використовувати одну мову програмування для всього стеку. Node.js став найпоширенішою технологією для розробки бекенд-систем, а також існує багато фреймворків для фронтенду, таких як Angular, React, Vue.js.

Зараз веб-розробка використовує багато різних технологій та стеків для створення сучасних, масштабованих та інтерактивних додатків. Один з популярних стеків, що використовується, називається MERN, який використовується для повноцінної розробки веб-додатків.

MERN дозволяє розробникам використовувати JavaScript як на фронтенді, так і на бекенді, що спрощує розробку, оскільки вони можуть використовувати одну мову програмування для всього стеку. MongoDB забезпечує гнучку збереження даних, Express.js допомагає створювати бекенд-логіку, React дозволяє створювати інтерактивний фронтенд, а Node.js

забезпечує виконання серверного коду.

Мета цієї кваліфікаційної роботи полягала у вивченні різних аспектів веб-розробки з використанням Full Stack JavaScript, а також у розробці прототипу веб-додатка на основі MERN-стеку. Для досягнення цієї мети було вивчено використання Node.js, Express, MongoDB та React при розробці повноцінних веб-додатків Full Stack. Були детально вивчені різні версії JavaScript, зокрема ES6, а також особливості та реалізація платформ, заснованих на JavaScript, таких як Node.js, Express та MongoDB. Також було досліджено потенційні загрози безпеці для програми на Node.js та розглянуто різні способи їх пом'якшення. В результаті було розроблено мінімально життєздатний продукт - інтернет-ресурс стоматологічної клініки з використанням основних пакетів і модулів Node.js, проміжного програмного забезпечення Express та інших сторонніх компонентів.

РОЗДІЛ 1 РОЗВИТОК ВЕБ-САЙТІВ НАДАННЯ МЕДИЧНИХ ПОСЛУГ В СВІТІ ТА УКРАЇНІ

1.1 Розвиток медичних веб-ресурсів в світі

Останні роки свідчать про значний розвиток медичних веб-ресурсів по всьому світу. Завдяки швидкому розширенню Інтернету та зростанню числа користувачів, все більше людей шукають медичну інформацію та послуги онлайн, див. рис. 1.1 з джерела [1]:

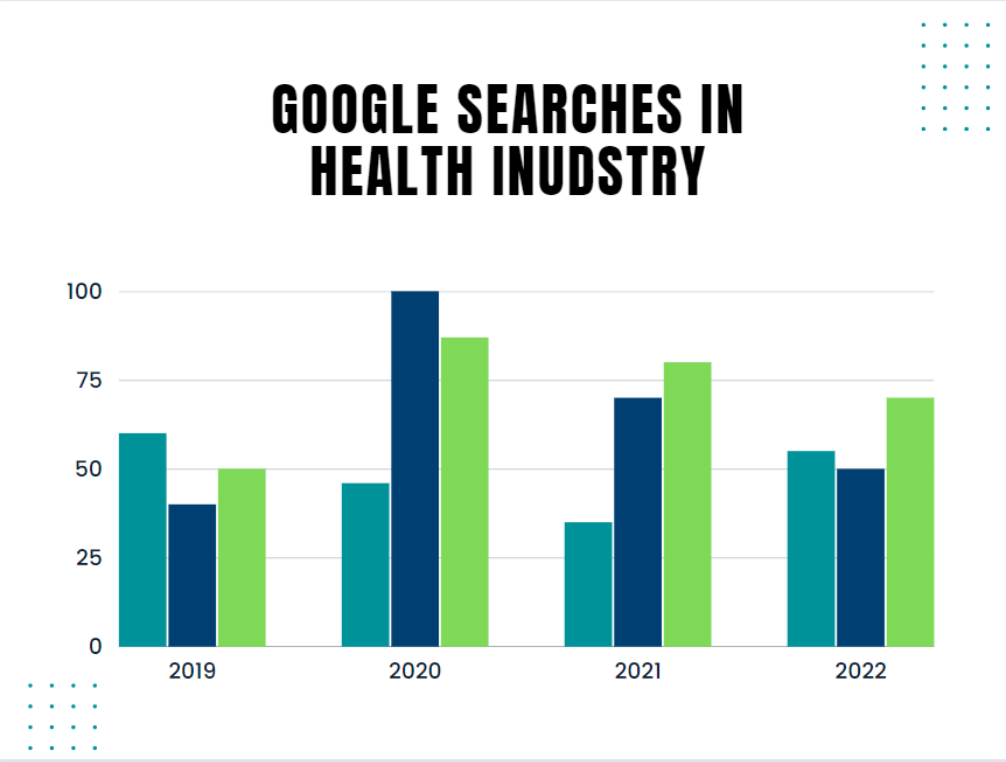

Рис. 1.1 Діаграма пошукових запитів на тему здоров’я

Медичні веб-ресурси відіграють важливу роль у покращенні доступності медичної інформації для громадськості. Вони надають можливість пацієнтам знайти відповіді на свої запитання, ознайомитися з симптомами та методами лікування різних захворювань. Крім того, ці ресурси також пропонують корисні поради з профілактики, здорового способу життя та догляду за собою.

Ще одним важливим аспектом розвитку медичних веб-ресурсів є зручність та доступність для користувачів. Вони надають можливість швидкого пошуку необхідної інформації, онлайн-консультацій з лікарями, запису на прийом та отримання результатів аналізів. Це дозволяє пацієнтам ефективно керувати своїм здоров'ям та економити час, уникаючи довгих черг у поліклініках.

Зростання медичних веб-ресурсів також стимулює розвиток електронної медицини та телемедицини. Завдяки цим технологіям пацієнти можуть отримувати консультації та лікування віддалено, не виходячи зі своїх домівок. Це особливо важливо для людей, які проживають в віддалених регіонах або мають обмежену можливість пересування.

Загалом, розвиток медичних веб-ресурсів відкриває нові перспективи для покращення медичної допомоги та забезпечення здоров'я населення. Вони забезпечують доступ до надійної медичної інформації, допомагають пацієнтам бути освіченими та самосвідомими щодо свого здоров'я. Такі ресурси сприяють зростанню якісної медичної допомоги та покращенню здоров'я суспільства в цілому.

1.2 Розвиток медичних веб-ресурсів в Україні

Останні роки в Україні спостерігається значний розвиток медичних веб-ресурсів, що відкриває нові можливості для покращення медичної сфери та надання здоров'язберігаючих послуг населенню. Цей тренд має велике значення, оскільки все більше українців шукають медичну інформацію та послуги в онлайн-середовищі.

Медичні веб-ресурси в Україні стають важливим інструментом для спілкування між пацієнтами та медичними закладами. Вони надають можливість пацієнтам швидко та зручно знайти необхідну

інформацію про медичні послуги, розклад роботи лікарів, можливості запису на прийом та отримання медичних консультацій. Це робить процес взаємодії між пацієнтами та медичними закладами більш ефективним та зручним, приклади можна побачити на рис. 1.2 та 1.3 з веб-сайту [2]:

Рис. 1.2 Надання списку послуг поліклініки онлайн

Одним із головних трендів в розвитку медичних веб-ресурсів в Україні є персоналізація. Медичні заклади активно працюють над створенням індивідуальних веб-сайтів, що відповідають потребам та очікуванням своїх пацієнтів. Це означає, що на веб-сайтах медичних закладів міститься інформація про конкретні послуги, спеціалізацію лікарів, цінову політику, а також відгуки та рекомендації задоволених пацієнтів. Такий індивідуальний підхід допомагає пацієнтам знайти необхідну інформацію швидко та зрозуміло.

Значну увагу при розвитку медичних веб-ресурсів в Україні приділяють наданню корисної та достовірної медичної інформації. На

веб-сайтах можуть бути розміщені статті, пам'ятки, поради щодо здоров'я, профілактики хвороб та основних медичних процедур. Це допомагає пацієнтам бути освіченими щодо їх стану здоров'я, а також розуміти, як зберігати та покращувати своє здоров'я. Крім того, медичні веб-ресурси в Україні активно використовуються для просування медичних закладів та популяризації їх послуг. Вони надають можливість розміщення інформації про спеціалізацію медичних закладів, кваліфікацію лікарів, перелік наданих послуг, а також демонстрацію сучасного обладнання та інфраструктури. Це сприяє залученню нових пацієнтів та побудові довіри до медичних закладів.

Чудовим прикладом може бути український сервіс надання медичних послуг helsi.me. Helsi - це сервіс здоров'я та медичних послуг, який надає широкий спектр можливостей для зручного управління своїм здоров'ям онлайн. Сервіс дозволяє користувачам зареєструватися та створити персональний профіль, де вони можуть зберігати свої медичні дані, записувати візити до лікарів, контролювати графік прийому ліків та багато іншого.

За допомогою Helsi.me користувачі можуть швидко знайти лікаря, перевірити його кваліфікацію та розклад прийому, а також забронювати прийом онлайн або офлайн. Крім того, сервіс надає можливість отримувати консультації від лікарів через відеозв'язок, що робить доступ до медичних послуг ще зручнішим та ефективнішим.

Helsi.me також надає можливість замовляти медичні аналізи та дослідження, переглядати результати онлайн та отримувати рекомендації щодо діагностики та лікування. Користувачі можуть отримувати нагадування про прийом ліків, розклади щеплень та іншу корисну інформацію пов'язану зі своїм здоров'ям.

Загалом, Helsi.me допомагає людям зручно та ефективно керувати своїм здоров'ям, забезпечуючи доступ до медичних послуг та

інформації в одному місці через онлайн-платформу.

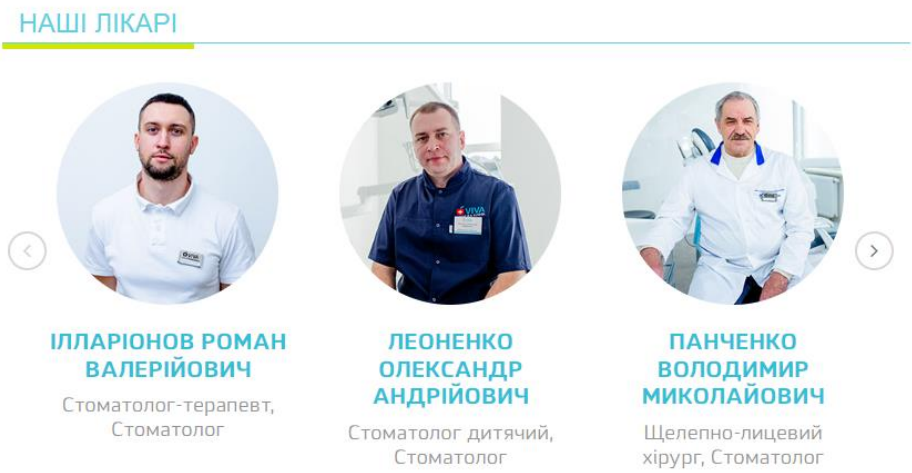

Рис. 1.3 Приклад представлення персоналу клініки

Висновки до розділу 1

Розвиток медичних веб-ресурсів є надзвичайно важливим явищем як у світі, так і в Україні. Ці ресурси відіграють значну роль у сприянні комунікації з пацієнтами, наданні достовірної медичної інформації та просуванні медичних закладів.

Популярність медичних веб-ресурсів зросла завдяки підходу до персоналізації, де веб-сайти пристосовані до потреб та очікувань пацієнтів. Вони надають зручний доступ до інформації про послуги, лікарів, розклад роботи та можливості запису на прийом онлайн. Крім того, ці ресурси акцентують увагу на наданні корисної та достовірної медичної інформації, що допомагає пацієнтам бути освіченими та здійснювати свідомий догляд за своїм здоров'ям.

Досягнення веб-ресурсів також полягають у популяризації медичних послуг та побудові довіри до медичних закладів. Інформація про спеціалізацію, кваліфікацію лікарів, відгуки пацієнтів та фотографії, що демонструють обладнання та інфраструктуру закладу, допомагають залучати нових клієнтів і визначати вибір медичного закладу.

Оптимізація медичних веб-ресурсів для різних пристроїв, зокрема

мобільних, сприяє зручному доступу до інформації, що важливо в сучасному світі, де все більше людей використовують мобільні телефони та планшети для пошуку інформації.

Розвиток медичних веб-ресурсів в Україні свідчить про стрімкий прогрес у сфері цифрової медицини та покращення доступу до якісних медичних послуг для населення.

РОЗДІЛ 2 ПЕРЕЛІК ТА ОПИС ТЕХНОЛОГІЙ РЕАЛІЗАЦІЇ

2.1 Етапи проектування MERN-додатків

Проектування MERN додатків включає кілька етапів, які допомагають створити функціональний та ефективний веб-додаток. Ось основні етапи проектування MERN додатків:

1. Аналіз та визначення вимог: У цьому етапі проводиться детальний аналіз вимог до додатка. Визначаються функціональні та нефункціональні вимоги, вимоги до інтерфейсу користувача, бази даних та інші технічні вимоги.
2. Проектування архітектури: На цьому етапі визначається загальна структура додатка та його архітектура. Обираються відповідні компоненти MERN стеку та встановлюються зв'язки між ними. Розробляється схема бази даних та визначаються API точки доступу.
3. Розробка серверної частини (Back-end): У цьому етапі розробляються серверні компоненти за допомогою Node.js та Express.js. Встановлюються маршрутизація, обробка запитів та з'єднання з базою даних MongoDB. Реалізується бізнес-логіка додатка та виконуються потрібні перевірки та аутентифікація.
4. Розробка клієнтської частини (Front-end): На цьому етапі розробляється клієнтська частина додатка з використанням React.js. Створюються компоненти інтерфейсу користувача, реалізується логіка взаємодії з сервером через API запити. Додається стилізація та реалізуються функціональні можливості додатка.
5. Інтеграція та тестування: На цьому етапі проводиться інтеграція серверної та клієнтської частин додатка.

Виконуються ретельні тести для перевірки працездатності, виявлення помилок та оптимізації. Тестування може включати автоматичні тести, тестування одиниць, функціональні та інші види тестів.

6. Розгортання та підтримка: Після успішного завершення тестування додаток готовий до розгортання на веб-сервері. Встановлюються потрібні середовища та конфігурації для роботи додатка у виробничому середовищі. Після розгортання забезпечується підтримка та виправлення помилок, а також можлива подальша розширення функціоналу.

Враховуючи ці етапи проектування, розробка MERN додатків стає структурованим та ефективним процесом, що дозволяє створити потужні та масштабовані веб-додатки для різних сфер, у тому числі й медичної галузі.

2.2 Мови розмітки та стилів веб-сторінки HTML та CSS

HTML (HyperText Markup Language) - це мова розмітки, яка використовується для створення структури та вмісту веб-сторінок. Вона визначає різні елементи та їхню взаємодію на веб-сторінці.

HTML використовується для створення структури веб-сторінок. Він складається з набору тегів, які описують різні елементи на сторінці. Основні теги HTML включають теги заголовків, параграфів, списків, посилань, зображень та таблиць. Крім того, HTML дозволяє вбудовувати мультимедійні елементи, форми введення даних, відео та аудіо.

CSS (Cascading Style Sheets) - це мова стилізації, яка використовується для визначення зовнішнього вигляду елементів

HTML. CSS використовується для стилізації веб-сторінок. Він визначає зовнішній вигляд елементів, таких як кольори, шрифти, розташування, розміри тощо. CSS працює з HTML, використовуючи селектори для вибору елементів та задання стилів для цих елементів. Він дозволяє створювати привабливий та однорідний дизайн для веб-сторінок, приклад з джерела [3] наведено нижче:

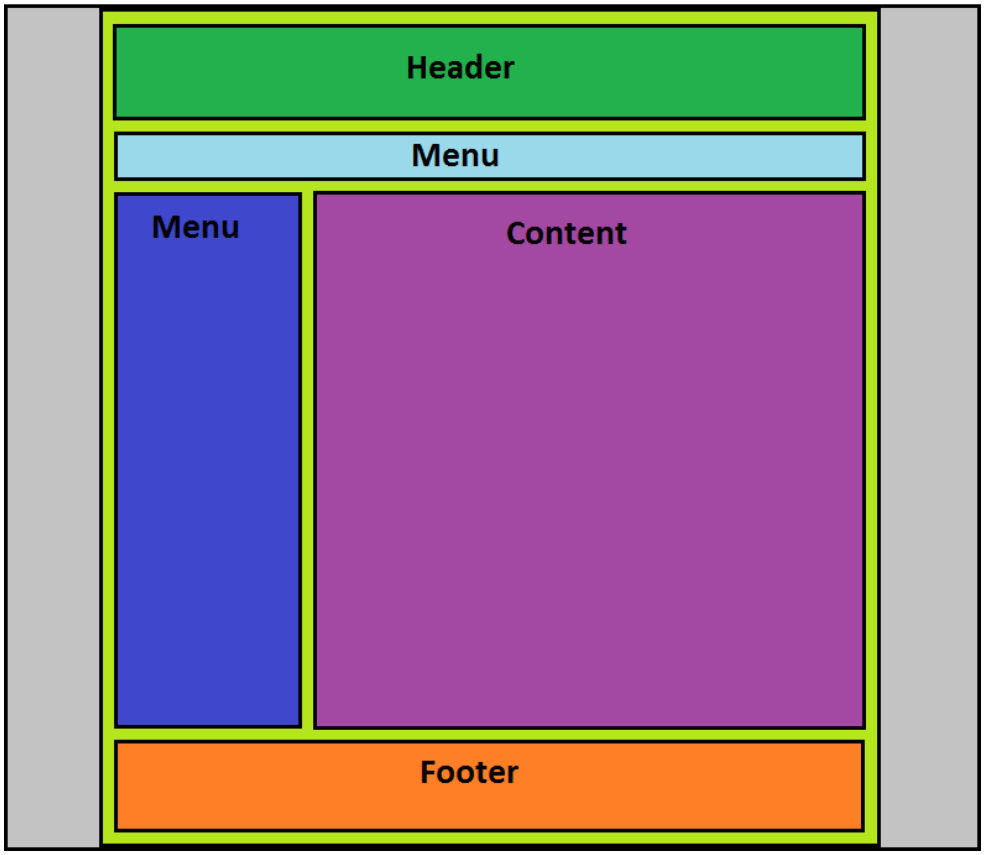

Рис. 2.1 Приклад розмітки веб-сторінки

2.3 МП Javascript та її особливості

JavaScript - це потужна мова програмування, яка використовується для розробки як клієнтської, так і серверної частини веб-додатків. Вона надає можливості для взаємодії з користувачем, обробки подій, маніпуляції веб-сторінками та взаємодії з сервером без перезавантаження сторінки. Ця мова програмування має динамічний характер, є Об’єктно-орієнтованою та базується на першокласних

функціях. Вона була заснована в 1995 році для того щоб взаємодіяти з Java в браузерах.

JavaScript є основною мовою для веб-розробки, оскільки вона дозволяє створювати динамічні та інтерактивні веб-сторінки. Завдяки своїй широкій підтримці в браузерах, JavaScript став стандартом для реалізації різних функцій, таких як перевірка валідності форм, анімація, валідація даних та багато іншого.

Однією з найбільших переваг JavaScript є його гнучкість і розширюваність. Багато бібліотек та фреймворків, таких як React, Angular та Vue.js, побудовані на основі JavaScript, що дозволяє розробникам швидко створювати складні та масштабовані веб-додатки.

JavaScript також має багато ресурсів та спільнот, які забезпечують великий об’єм документації, пакетів та розширень, що допомагають розробникам прискорити процес розробки та розв'язання різних завдань.

Однак, варто відзначити, що JavaScript має і свої обмеження:

- Вона є мовою зі слабою типізацією, що може викликати деякі проблеми при великих проектах з багатою логікою.
- Оскільки JavaScript виконується на клієнтській стороні, він піддається більшій вразливості і може бути підданий небажаним змінам або зламам.
- Обмежені можливості в обробці важких обчислень: JavaScript, зокрема з використанням середовища виконання Node.js, не є найкращим вибором для виконання важких обчислень, обробки великого обсягу даних, машинного навчання або складних алгоритмів.
- Обмежена підтримка старих браузерів: Деякі старі версії браузерів можуть не підтримувати останні функції та стандарти JavaScript.

Усупереч своїм недолікам, JavaScript продовжує займати провідні позиції в світі веб-розробки. Його постійний розвиток, широкі можливості та активна спільнота роблять його важливим інструментом для створення сучасних та інноваційних веб-додатків.

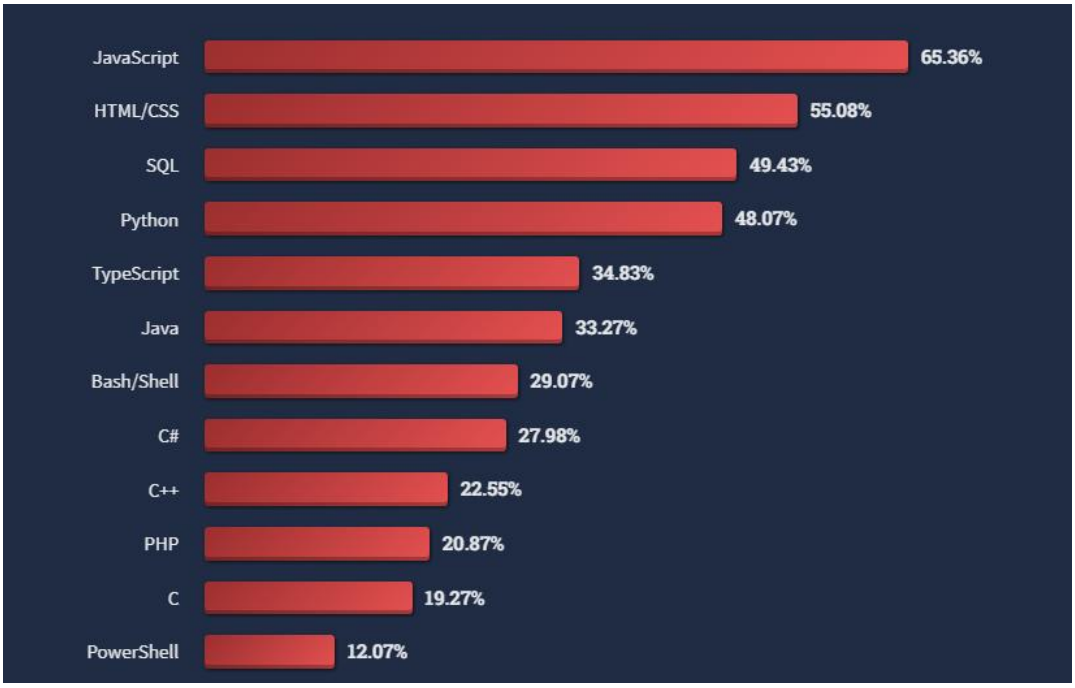

Рис. 2.2 2022 Developer Survey від Stack Overflow

Відповідно до результатів опитування, які можна побачити на рис. 2.2 з джерела [4], проведеного компанією Stack Overflow у 2022 році, серед понад 70000 професійних розробників програмного забезпечення та студентів, які бажають поділитися своїми улюбленими технологіями розробки, виявлено, що 65,36% учасників назвали JavaScript своєю найчастіше використовуваною мовою програмування. Це статистично підтверджує високу популярність JavaScript як однієї з найбільш поширених технологій.

JavaScript, раніше відомий як ECMAScript, має багату історію випуску нових версій, які вводять нові можливості та поліпшення до мови програмування. Він був стандартизований в 1996 році компанією

Netscape за допомогою ECMA (Ecma International - Європейська асоціація стандартизації інформаційно-комунікаційних систем). Починаючи з того часу, було випущено різні версії ECMAScript. Після випуску найбільш масштабної версії в 2015 році ECMA прагне видавати нові версії щороку. Тому ECMA змінила систему нумерації версій ECMAScript з номерів на роки випуску. Однак серед спільноти JavaScript більш популярною є система обчислення версій. Ось огляд основних версій ECMAScript:

- ECMAScript 5 (ES5): Версія від 2009 року, що додала практичної реалізації мові JavaScript і стала однією з найпідтримуваніших версій.
- ECMAScript 6 (ES6) або ECMAScript 2015 був революційним випуском JavaScript, який вийшов у 2015 році. Ця версія має величезне значення для мови програмування JavaScript, оскільки вона внесла значні покращення в синтаксис та введення нових можливостей для розробників.

ECMAScript 6 (ES6), також відомий як ECMAScript 2015, є однією з найсуттєвіших версій стандарту ECMAScript для мови програмування JavaScript. Випущений у 2015 році, ES6 вніс багато нових функцій та поліпшень, які значно покращили синтаксис та можливості JavaScript.

Починаючи з ES6, стандарт ECMAScript почав видаватися щорічно, а не залежно від номера версії. Це дозволяє швидше впровадження нових функцій та поліпшень в мові. Наразі продовжується розвиток ECMAScript, і нові версії, такі як ES10, ES11 і т. д., включають додаткові функції та розширення. Це дозволяє розробникам використовувати потужніші та сучасні можливості JavaScript для розробки веб-додатків.

Основні нововведення та поліпшення, які з'явилися у ECMAScript 6 (ES6) або ECMAScript 2015, виявилися революційними для мови JavaScript. ES6 вніс величезні зміни в синтаксис та впровадив нові можливості для розробників. Ось найважливіші зміни:

- Стрілкові функції: Було додано нові правила оголошення функцій з використанням стрілок (=>), що спрощує роботу з контекстом this та дозволяє створювати короткі та зручні функції.
- Шаблонні рядки: Додано можливість використовувати шаблонні рядки, де можна вставляти вирази за допомогою синтаксису ${вираз}.
- Імпорт та експорт модулів: Введено синтаксис модулів, що дозволяє розділити код на окремі модулі та імпортувати потрібні функції, класи або об'єкти з інших модулів.

2.4 Фронтенд фреймворк React

React - це потужний та інноваційний фреймворк для розробки веб-додатків. Його основною ідеєю є створення користувацького інтерфейсу з високою ефективністю та перевикористовуванням компонентів. За допомогою React, розробка веб-додатків стає захоплюючим та ефективним процесом. Він пропонує декларативний підхід до розробки, де ви можете описувати, як ваші компоненти повинні виглядати у різних станах.

Одним з найважливіших аспектів React є віртуальний DOM, який забезпечує швидкий та ефективний рендеринг компонентів. React вміє "розуміти" лише ті зміни, які потрібно внести до DOM, тим самим забезпечуючи оптимальну продуктивність.

Крім того, React має широку екосистему із багатьма додатковими бібліотеками та інструментами, які допомагають вам у розробці. Він також підтримує концепцію компонентного підходу, що сприяє

легкості розширення та підтримці вашого коду.

Завдяки своїм перевагам та популярності, React став одним з найулюбленіших виборів для розробки сучасних веб-додатків. Використовуючи React, ви можете створювати надзвичайно масштабовані та високоякісні додатки, які здатні задовольнити навіть найвибагливіші потреби користувачів.

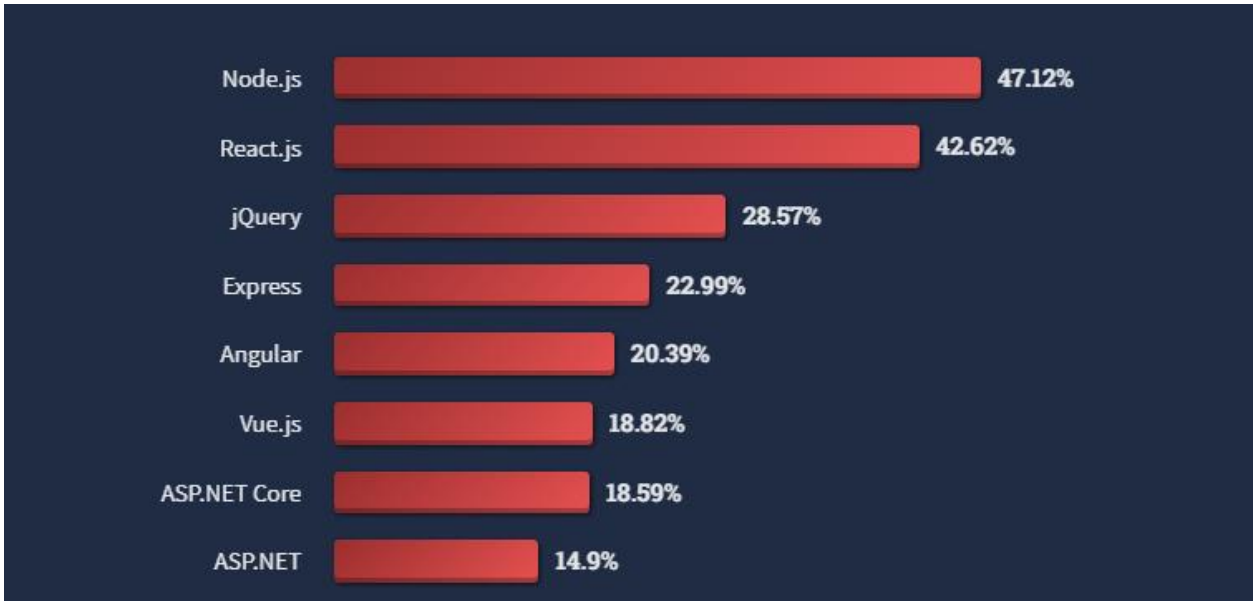

Рис. 2.3 Найвживаніші веб-фреймворки у 2022 році

Знову звертаючись до статистики яку щорічно підбиває компанія Stack Overflow, на рис. 2.3 з джерела [4], можна побачити що React.js являється одним з найпопулярніших фреймворків серед веб-розробників.

Головні принципи використання React включають:

- Компонентна архітектура: React базується на ідеї розбиття інтерфейсу на невеликі та перевикористовувані компоненти. Компоненти в React використовуються для побудови складних інтерфейсів шляхом комбінування простих компонентів.
- Віртуальний DOM: React використовує віртуальний DOM, що дає змогу ефективно оновлювати тільки необхідні елементи на сторінці, зменшуючи затрати на маніпулювання

реальним DOM.

- Одностороннє зв'язування даних: React пропонує одностороннє зв'язування даних, що означає, що дані відображаються від батьківського компонента до дочірніх компонентів, що полегшує керування станом додатка.
- Використання JSX: JSX є розширенням синтаксису JavaScript, яке дозволяє описувати структуру інтерфейсу у вигляді компонентів безпосередньо в коді JavaScript. Це полегшує розробку та читання коду.

Незважаючи на численні переваги, React також має кілька недоліків:

- Навчання та оволодіння React може вимагати певного часу та зусиль, особливо для розробників, які не мають досвіду з компонентною архітектурою.
- Висока складність додатків: Зі зростанням розміру та складності додатків, керування станом та взаємодія між компонентами може стати складнішою задачею.
- Екосистема та вибір інструментів: Вибір додаткових бібліотек, інструментів та підходів може бути викликом, оскільки велика кількість доступних варіантів може призвести до плутанини та необхідності додаткового дослідження.
- Першочерговість веб-розробки: React зосереджений переважно на розробці веб-інтерфейсів, тому для розробки інших типів додатків, таких як мобільні або настільні, можуть знадобитись додаткові інструменти та бібліотеки.

Незважаючи на ці недоліки, React продовжує бути популярним вибором для багатьох розробників завдяки своїм перевагам у швидкості, продуктивності та перевикористовуванні компонентів,

приклади яких можна побачити на рис. 2.4 та 2.5 з джерела [5].

Рис. 2.4 Приклад створення простого компоненту в React

Компоненти використовують метод render(), який обробляє вхідні дані та повертає вміст, що відображатиметься для користувача. У даному прикладі використовується JSX - синтаксис, схожий на XML. Звернення до переданих в компонент вхідних даних здійснюється за допомогою методу render() та this.props.

```
class Timer extends React.Component {
  constructor(props) {
    super(props);
    this.state = { seconds: 0 };
  }

  tick() {
    this.setState(state => ({
      seconds: state.seconds + 1
    }));
  }

  componentDidMount() {
    this.interval = setInterval(() => this.tick(), 1000);
  }
```

Рис. 2.5 Приклад React компоненту зі станом

Крім прийняття вхідних даних (доступних через this.props), компоненти також можуть зберігати внутрішні дані в стані (this.state), як зображено на рис. 2.4. Коли дані стану компонента змінюються, розмітка автоматично оновлюється шляхом виклику функції render().

2.5 Платформа Node.js

Node.js - це відкрите середовище виконання JavaScript, що

дозволяє виконувати JavaScript-код поза межами веб-браузера. Це потужний інструмент, заснований на двигуні V8, розробленому компанією Google для виконання JavaScript в браузері Chrome. Однак, Node.js відрізняється своїм підходом до виконання коду та забезпечує високу продуктивність.

Основою Node.js є однопотокова подієва модель, що означає, що він може обробляти події асинхронно, не блокуючи виконання коду, принцип роботи можна побачити на рис. 2.6 з джерела [6]. Це робить його ідеальним вибором для написання серверних додатків та іншого програмного забезпечення, що може вимагати великої навантаженості.

Node.js надає вбудовану бібліотеку вводу-виводу, яка дозволяє виконувати операції вводу-виводу асинхронно. Це включає роботу з мережевими запитами, файловою системою та іншими джерелами даних. Такий підхід дозволяє створювати ефективні та швидкі додатки.

Однією з найбільших переваг Node.js є його здатність обробляти багато запитів одночасно. Він використовує неблокуючий ввід-вивід та потоки, що забезпечує ефективне виконання операцій у багатопотоковому середовищі. Крім того, Node.js має широкий вибір сторонніх модулів і використовує пакетний менеджер npm, що дозволяє швидко розгортати проекти та використовувати готові компоненти. Node.js знайшло застосування в розробці серверних додатків, мережевих додатків, інструментів командного рядка, мікросервісів та іншого програмного забезпечення , де швидкість та ефективність є критичними факторами. Його популярність продовжує зростати завдяки його потужності та гнучкості.

Node Package Manager (NPM) - це пакетний менеджер для середовища Node.js. Він є стандартним інструментом, який допомагає в управлінні залежностями проекту та установці зовнішніх пакетів.

NPM дозволяє швидко знайти, завантажити, встановити та оновити пакети з репозиторію NPM, який є найбільшою колекцією відкритих пакетів для JavaScript. За допомогою NPM розробники можуть легко управляти залежностями свого проекту. Вони можуть визначити необхідні пакети та їх версії у файлі `package.json`. NPM автоматично завантажує ці пакети з репозиторію та встановлює їх у проект. Крім того, NPM дозволяє керувати версіями пакетів, оновлювати їх та вирішувати конфлікти між залежностями. Використовуючи NPM, розробники можуть спростити розгортання та розробку своїх проектів, використовуючи готові пакети і модулі, які відповідають їх вимогам. Крім того, NPM також дозволяє розробникам публікувати власні пакети, щоб інші розробники могли їх використовувати. В цілому, NPM є важливим інструментом в екосистемі Node.js, який допомагає розробникам ефективно управляти залежностями та підтримувати структуровані проекти.

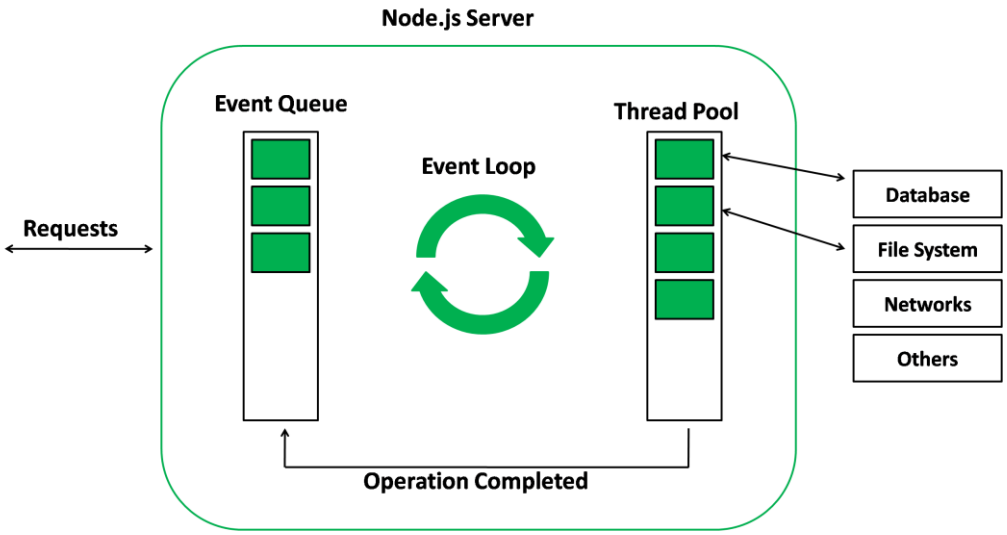

Рис. 2.6 Принцип роботи Event loop в середовищі Node.js

2.6 Бекенд фреймворк Express

Express - це веб-фреймворк для розробки веб-додатків на основі Node.js. Він надає простий та інтуїтивний спосіб побудови веб-

серверів і API, дозволяючи розробникам швидко створювати потужні та масштабовані додатки.

Одна з головних переваг Express - це його мінімалістичний підхід. Він надає лише основні функції і можливості, не нав'язуючи жорстких правил і обмежень. Це дає розробникам велику свободу в побудові архітектури свого додатка і виборі необхідних компонентів.

Express пропонує простий і зрозумілий API для обробки маршрутів (routes) і обробки запитів (request) та відповідей (response). Розробники можуть легко визначати маршрути, обробники запитів та виконувати різні дії, такі як отримання даних з бази даних, відправлення відповідей клієнту, рендерінг сторінок та багато іншого.

Express також підтримує велику кількість сторонніх модулів (middleware), які допомагають розширити його функціональність. Middleware - це функції, які обробляють запити перед тим, як вони досягнуть основного обробника запитів. Це дозволяє виконувати різні завдання, такі як аутентифікація, логування, обробка помилок і багато іншого.

Завдяки своїй популярності та активній спільноті, Express має велику кількість ресурсів, документацію, приклади та плагіни, що дозволяє розробникам ефективно працювати з цим фреймворком. Він використовується для розробки різноманітних веб-додатків, в ключаючи веб-сайти, API, мікросервіси та інші типи додатків.

Express використовує методи HTTP (GET, POST, PUT, DELETE) та шляхи до ресурсів для реалізації маршрутизації. Кожен маршрут може мати власне проміжне програмне забезпечення, яке обробляє пов'язані з ним запити. Проміжне програмне забезпечення виконує специфічні дії, такі як перевірка аутентифікації, обробка даних або бізнес-логіка. Їх можна додавати до маршрутів за допомогою методів .use(), .get() або .post(), залежно від потреб рівня проміжного програмного забезпечення, на якому вони застосовуються.

```
// GET method route
app.get('/', function (req, res) {
  res.send('GET request to the homepage');
});

// POST method route
app.post('/', function (req, res) {
  res.send('POST request to the homepage');
});
```

Рис. 2.7 Приклад обробки маршрутів в Express

В Express також існує можливість обробки помилок за допомогою спеціального проміжного програмного забезпечення, приклад можна побачити на рис. 2.7. Це проміжне програмне забезпечення можна додати до додатка та воно буде викликане у випадку виникнення помилки. Воно отримує аргументи, такі як помилка, об'єкт запиту та об'єкт відповіді, і здатне виконувати різні дії, такі як відправлення користувачеві повідомлень про помилку або їх журналювання.

```
function errorHandler(err, req, res, next) {
  if (res.headersSent) {
    return next(err);
  }
  res.status(500);
  res.render('error', { error: err });
}
```

Рис. 2.8 Приклад обробки помилок в Express

2.7 База даних MongoDB та Mongoose

MongoDB - це документоорієнтована, NoSQL база даних, яка зберігає дані у вигляді документів у форматі BSON (Binary JSON). Вона використовує модель даних, яка дозволяє гнучко зберігати та

організовувати дані без заданої схеми, що дає можливість швидко вносити зміни у структуру даних.

MongoDB часто використовується в парі з Mongoose. Mongoose - це об'єктно-документний (ODM) інструмент для Node.js, який надає зручний спосіб моделювання даних MongoDB у вигляді схем та взаємодії з базою даних. Він надає високорівневий API, який дозволяє розробникам зручно виконувати операції збереження, оновлення, видалення та запитів до даних MongoDB.

Ключові переваги використання MongoDB в парі з Mongoose:

- Зручне моделювання даних: Mongoose дозволяє визначати схеми для даних MongoDB, що спрощує моделювання та валідацію даних.
- Зручний API для взаємодії з базою даних: Mongoose надає високорівневий API, який спрощує виконання операцій збереження, оновлення, видалення та запитів до бази даних MongoDB.
- Middleware: Mongoose надає можливість використовувати middleware, яке дозволяє вам виконувати дії перед або після виконання операцій з даними.
- Керування відносинами між даними: MongoDB має можливість вкладеності та посилання між документами, а Mongoose надає зручний спосіб використовувати ці можливості.
- Підтримка запитів та агрегацій: MongoDB надає потужні можливості для виконання запитів та агрегаційних операцій.

Висновки до розділу 2

Сфера розробки веб-додатків дуже швидко та постійно еволюціонує. Останнім часом з’являється багато нових технологій та фреймворків і розробники маються швидко адаптовуватися під такі

правила. Та навпаки, раніше вибір мови програмування був дуже обмеженим і не існувало такого варіанту який би можна було використовувати як на клієнті(фронтенд), так і на сервері(бекенд).

Проте, з появою Node.js як серверного середовища JavaScript, та багатьох фронтенд фреймворків, наприклад React, стало можливим розробляти повноцінне програмне забезпечення, використовуючи лише одну мову програмування. Тому для цього конкретного проекту вибір було зроблено на користь JavaScript та MERN-стеку, що включає в себе вищезазначені MongoDB, Express, React та Nodejs і дозволяє ефективно та зручно розробляти веб-додатки.

Основні переваги розробки використовуючи MERN-стек:

- Усі компоненти MERN-стеку використовують JavaScript, що дозволяє розробникам працювати з однією мовою на всіх рівнях стеку.
- MERN-стек підтримує прогресивний підхід до веб-розробки, особливо завдяки використанню React(SPA).
- MERN-стек дозволяє побудовувати масштабовані веб-додатки завдяки використанню MongoDB як бази даних, яка працює на основі документів.
- MERN-стек дозволяє використовувати багато готових модулів, пакетів та бібліотек для розширення функціональності веб-додатків.

Загалом, використання MERN-стеку надає розробникам потужні та гнучкі інструменти для швидкого розгортання веб-додатків.

РОЗДІЛ 3 ПРОЦЕС РОЗРОБКИ ВЕБ-САЙТУ ПРИ ВИКОРИСТАННІ СТЕКУ MERN

3.1 Введення в проект

Веб-сайт "Oxydent", який розроблено як прототип, ілюструє використання Fullstack Javascrtipt розробки, а саме MERN стеку. Цей сайт є мінімально життєздатним продуктом (MVP), створеним для поліклініки Oxydent з метою надання онлайн-послуг в галузі стоматології своїм клієнтам.

Додаток включає в себе основні функціональні можливості, які можуть потребувати клієнти поліклініки Oxydent. У майбутньому MVP може бути розширений, щоб охопити інші аспекти надання послуг в Інтернеті, пов'язані зі стоматологічною поліклінікою Oxydent.

Після детального аналізу основних функцій та послуг які надаються веб-сайтами у сфері медицини, а також після ретельного вивчення потреб замовника та клієнтів саме поліклініки Oxydent, було визначено необхідний мінімальний функціонал, що включає в себе:

1. Реєстрація користувача
2. Аутентифікація користувача
3. Вихід з особистого кабінету
4. При умові авторизації, користувач отримує змогу запису до лікаря
5. Користувач має змогу вибору лікаря та послуги
6. Користувач має змогу переглядати свій обліковий запис(свої записи до лікаря)
7. Користувач має змогу видаляти записи
8. Доступ до сторінок «Про нас», «Ціни», «Контакти»

Для наочного прикладу можливостей користувача було розроблено use-case діаграму, яку можна побачити на рис. 3.1

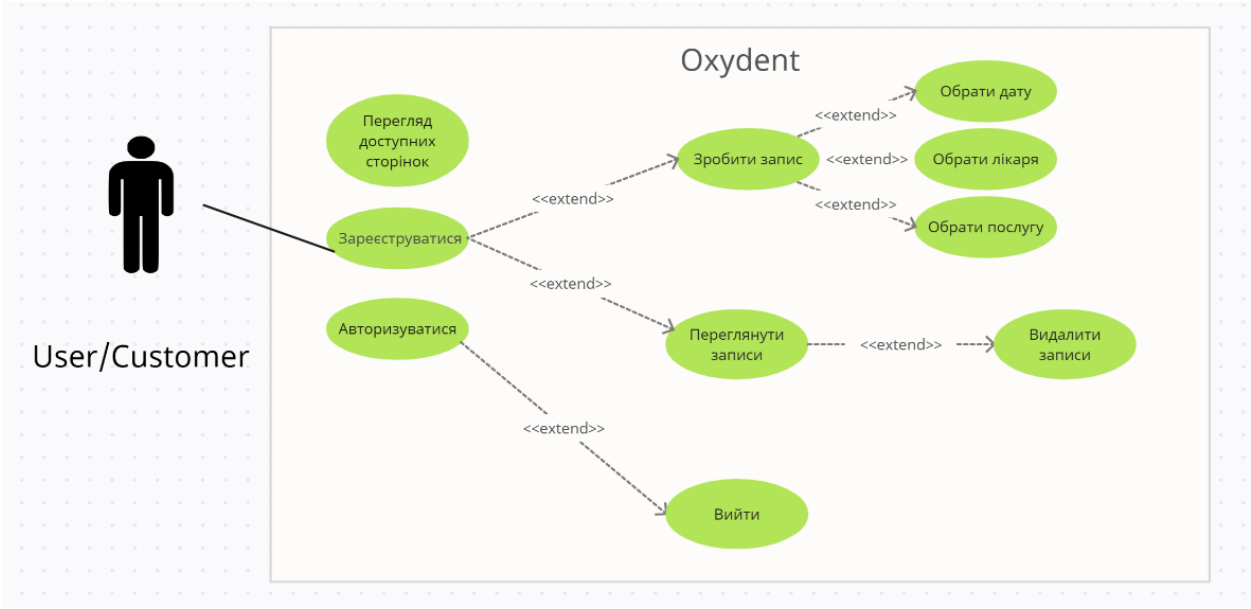

Рис. 3.1 Oxydent use-case діаграма

Для розробки веб-сайту було використано архітектурний патерн Клієнт-сервер. Модель клієнт-сервер є архітектурним підходом, що використовується в розподілених системах для обміну інформацією між клієнтами та сервером. У цій моделі клієнт і сервер взаємодіють за допомогою мережі, де клієнт звертається до сервера для отримання ресурсів, послуг або обробки даних.

Основна ідея полягає в тому, що клієнти (наприклад, веб-браузери) і сервери (наприклад, веб-сервери) виконують різні ролі в цій взаємодії. Клієнтська сторона, яка зазвичай перебуває на кінцевому пристрої користувача, ініціює запити до сервера, передаючи йому дані або запитуючи інформацію. Серверна сторона, з свого боку, обробляє ці запити, зберігає та надає доступ до ресурсів або послуг, які вимагаються клієнтами.

Модель клієнт-сервер забезпечує децентралізовану архітектуру, де клієнти можуть бути підключені до різних серверів і обмінюватись інформацією. Це дозволяє розділити завдання обробки даних між клієнтами та серверами, сприяючи більш ефективному та масштабованому розподілу ресурсів.

Модель клієнт-сервер є широко використовуваним підходом у веб-розробці, де клієнти взаємодіють з сервером за допомогою протоколів

передачі даних, таких як HTTP. Вона дозволяє розробникам ефективно розділяти логіку клієнтського і серверного додатків, забезпечуючи більшу гнучкість та масштабованість у розробці та підтримці систем.

Детальний огляд моделі можна побачити на рис. 3.2 з джерела [7]:

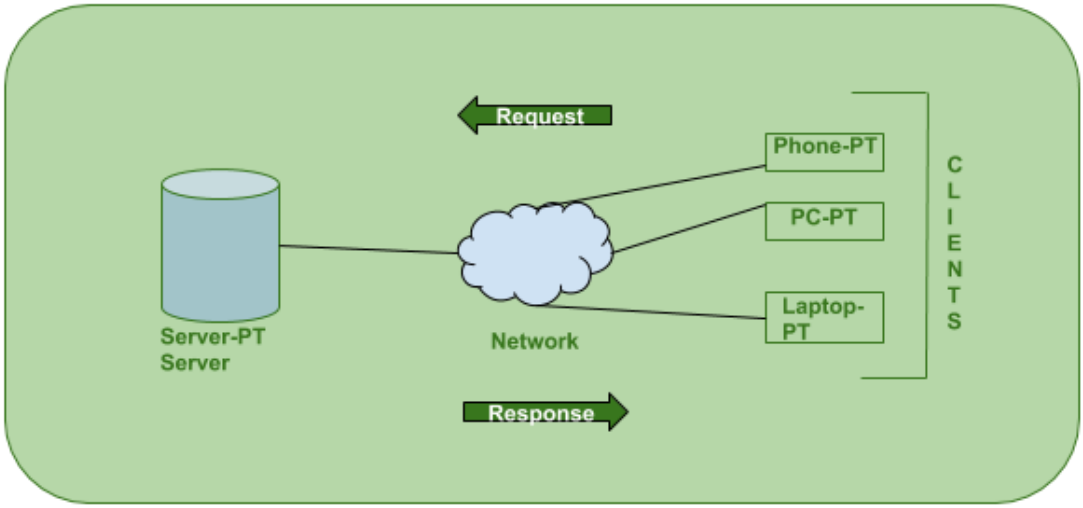

Рис. 3.2 Архітектура client-server схематично

Основні переваги використання клієнт-серверної моделі:

- Розподілена обробка: Модель клієнт-сервер дозволяє розподілити обчислювальні завдання між клієнтськими і серверними компонентами системи.
- Централізований доступ до даних: За моделлю клієнт-сервер, сервер може бути відповідальним за зберігання та керування даними.
- Забезпечення безпеки: Використання моделі клієнт-сервер дозволяє впроваджувати механізми безпеки на рівні сервера.
- Масштабованість: При збільшенні обсягу роботи або кількості клієнтів можна розширити обчислювальні ресурси сервера або додати додаткові сервери для обробки навантаження.

Загалом, модель клієнт-сервер має численні переваги, що робить її популярним вибором для багатьох комп'ютерних систем.

3.2 Розробницьке середовище

Правильно обрані інструменти і середовища розробки є ключовим для досягнення ефективності і працездатності продукту. Використання вірно підібраних технологій і мов програмування спрощує процес розробки ПЗ не лише в сфері веб-програмування.

При розробці Oxydent.com було використано ноутбук MSI GS65 з операційною системою Windows 11. Середовищем розробки було обрано VScode - кроссплатформений, легкодоступний але досить функціональний редактор коду. VScode надає розробникам зручні інструменти для підтримки кодування, навігації по коду, рефакторингу та тестування. Аналогами VScode можуть виступати наступні текстові редактори та IDE: WebStorm, Sublime text, Atom, Eclipse та інші.

Однією з головних складових частин процесу розробки ПЗ є керування версіями проекту. Для цього було обрано Git, який є розподіленою системою керування версіями. Git надає швидкість, цілісність даних та підтримку розподілених нелінійних робочих процесів. Код програми та його історія зберігалися на платформі GitHub, яка є популярним сервісом для хостингу та керування версіями з великою кількістю зареєстрованих користувачів та репозиторіїв. Використання цих інструментів та платформ спрощує розробку, забезпечує надійність і підвищує продуктивність роботи.

В якості аналогів можна використовувати наступні системи керування версіями та платформи для хостингу: Mercurial, SVN, GitLab, Bitbucket. Ці аналоги Git та GitHub надають схожі можливості з керування версіями, хостингу репозиторіїв та співпраці в команді. Вони допомагають зберігати, відстежувати та керувати кодовою базою проекту.

3.3 Розробка серверної частини

3.3.1 Встановлення технологій для розробки серверу

Першим і основним компонентом розробки серверної частини є **Nodejs**. Для встановлення Nodejs з офіційного сайту nodejs.org потрібно виконати такі

кроки як: перейти на офіційний сайт, обрати версію, нажати «Встановити» та обрати директорію в яку потрібно встановити Nodejs. Було встановлено останню, на момент розробки, версію 20.0.0. Для того щоб перевірити і переконатися що все встановлено коректно можна скористатися відповідною командою "*node -v*" у терміналі.

Разом з Nodejs, на комп'ютер встановлюється і менеджер пакетів NPM. **NPM** (Node Package Manager) - це відкритий інструмент для керування пакетами, який широко використовується для завантаження та встановлення JavaScript-пакетів, доступних в мережі Інтернет. Крім цього, він забезпечує можливість спільного використання коду, керування залежностями додатків та обміну відгуками між розробниками. NPM є незамінним інструментом для розробки додатків на основі Node.js.

Наступним кроком після установки має бути ініціалізація node проекту. Цього можна досягти перейшовши в необхідну директорію та використавши наступну команду *"npm init"*, під час виконання якої ми вказуємо всю необхідну інформацію для проекту. Після ініціалізації проекту скрипт автоматично створює файл *package.json* в якому буде зберігатися вся інформація про проект та його залежності. Для того щоб перевірити, або змінити, поточні версії встановлених пакетів та залежностей, існує наступна команда *"npm version"*.

Після того як ми переконалися що на нашому комп'ютері попередньо встановлено Nodejs та NPM, а також після того як проект було ініціалізовано в системі, можна рухатися далі. Наступною технологією, яка дуже сильно спрощує розробку серверної частини додатка, є бекенд фреймворк **Express**. Для того щоб ініціалізувати Express у проекті нам знадобиться наступна команда *"npm install express –save -g"*. Ця команда одразу встановлює Express глобально, а також зберігає пакет в залежностях проекту. Флаг *–save* указує на те що npm має зберегти Express як залежність проекту у файлі *package.json*. А флаг *-g* або *--global* вказує на те що пакет необхідно встановити глобально, тобто бути готовим для використання в будь-якому проекті системи. Треба

додати, що глобальне встановлення пакетів може використовуватись для інструментів розробки або утиліт командного рядка, які будуть доступними в будь-якому проекті. Для залежностей проекту, які використовуються лише у конкретному проекті, бажано інсталювати їх локально.

Після встановлення базових залежностей і пакетів у проекті, необхідно перейти до наступного етапу, а саме підключення бази даних до проекту. Існує велика кількість різноманітного ПЗ для використання баз даних у проектах, але при MERN розробці використовується БД **MongoDB**.

MongoDB може використовуватись як локальна база даних на комп'ютері, за допомогою MongoDB Compass, або як хмарне сховище використовуючи MongoDB Atlas. MongoDB Atlas - це хмарна платформа, яка дозволяє зберігати дані у хмарі, надаючи легке масштабування та керування даними. За допомогою MongoDB Atlas можна створювати, керувати та масштабувати кластери MongoDB у хмарі, забезпечуючи безпеку та доступність даних.

При розробці веб-сайту Oxydent був використаний саме хмарний варіант MongoDB Atlas, і були виконані наступні кроки для ініціалізації та підключення до проекту:

1. Необхідно зареєструватися на сайті mongodb.com.
2. У своєму обліковому запису створити відповідний проект.
3. Після створення нового проекту треба обрати безкоштовний кластер MongoDB(детальніше див. рис. 3.4).
4. Далі необхідно створити користувача з дозволом на читання та запис до бази даних, яка використовується у нашому додатка для зберігання та отримання даних.
5. Потім необхідно додати IP-адреса машини до списку дозволених IP-адрес, щоб додаток міг взаємодіяти з сервером.
6. Останнім кроком необхідно встановити пакет MongoDB у нашому проекті за допомогою команди *"npm install mongodb --save"*.

Таким чином, використання MongoDB Atlas у розробці веб-сайту

Oxydent дозволило забезпечити зручне зберігання, керування та масштабування бази даних у хмарі, забезпечуючи безпеку та доступність даних.

В акаунті MongoDB Atlas можна знайти створений кластер. В цьому кластері відображається різноманітна інформація, така як географічне розташування сервера, статистика операцій читання та запису, кількість активних з'єднань на сервері та розмір вашої бази даних.

Для спрощення взаємодії між сервером і базою даних було використано **Mongoose**. Mongoose є об'єктно-документним моделювальником (ODM), який дозволяє визначати об'єкти зі строго типізованою схемою, що відповідає структурі документів MongoDB. Її встановлено за допомогою команди *"npm install mongoose --save"*. При створенні та налаштуванні кластера MongoDB Atlas був згенерований спеціальний URL, за допомогою якого встановлюється з'єднання між сервером і хмарною базою даних за допомогою Mongoose. Нижче, на рис. 3.3, наведений приклад команди для встановлення з'єднання з використанням Mongoose.

```
mongoose
  .connect(
    "mongodb+srv://nevermore7331:5pkx8fts@cluster0.dpdoy1a.mongodb.net/",
    {
      useNewUrlParser: true,
      useUnifiedTopology: true,
    }
  )
  .then(() => {
    console.log("З'єднано з базою даних");
  })
  .catch((error) => {
    console.error("Помилка при підключенні до БД", error);
  });
```

Рис.3.3 Приклад підключення до MongoDB за допомогою Mongoose

Також, окрім основних технологій, до серверної частини було встановлено наступні пакети:

- Пакет для авторизації та аутентифікації користувачів **jsonwebtoken.** Оскільки в нашому додатка присутня авторизація

та аутентифікація користувачів, нам потрібен механізм для контролю доступу до певних ресурсів. jsonwebtoken є бібліотекою, яка дозволяє генерувати та перевіряти JSON Web Tokens (JWT). Ця бібліотека дозволяє нам створювати безпечні токени для аутентифікації та авторизації користувачів у нашому додатка. Крім того, на офіційному веб-сайті Jwt.io ми можемо перевірити закодовану нами інформацію за допомогою нашого "секретного слова".

bcryptjs - це додатковий пакет для Node.js, призначений для хешування паролів з використанням алгоритму bcrypt. Bcrypt - це алгоритм хешування паролів, розроблений Нільсом Провосом і Девідом Мазьєром на основі шифру Blowfish. Назва "bcrypt" складається з двох частин: b і crypt, де b означає Blowfish, а crypt - це назва функції хешування, яка використовується в системі паролів Unix. Bcrypt був створений в результаті того, що Crypt не зміг адаптуватися до розвитку технологій та апаратного забезпечення. Bcrypt розроблений як повільний алгоритм, що добре, коли мова йде про хешування паролів. Тому bcrypt ідеально підходить для хешування паролів, оскільки він зменшує кількість атак грубої сили.

- **Dotenv** - це модуль з нульовою залежністю, який завантажує змінні оточення з файлу .env в process.env. Зберігання конфігурації в оточенні окремо від коду базується на методології The Twelve-Factor App.
- Пакет **nodemon** - це інструмент, який допомагає розробникам автоматично перезавантажувати сервер після зміни коду в проекті. Це досягається за рахунок зменшує час, витрачений на ручне перезавантаження сервера, що роблячи процес розробки більш ефективним і продуктивним. Nodemon є пакетом для Node.js і встановлюється через npm.
- **cors** - це пакет node.js для забезпечення проміжного програмного забезпечення Connect/Express, яке можна використовувати для

включення CORS з різними опціями.

Підсумовуючи, ми отримуємо такий загальний список пакетів та залежностей необхідних для реалізації серверної частини:

```
"dependencies": {
  "bcrypt": "^5.1.0",
  "body-parser": "^1.20.2",
  "cors": "^2.8.5",
  "dotenv": "^16.0.3",
  "express": "^4.18.2",
  "jsonwebtoken": "^9.0.0",
  "mongoose": "^7.1.1",
  "nodemon": "^2.0.22"
}
```

Рис. 3.4 Список пакетів та залежностей серверної частини

3.3.2 Процес розробки серверної частини

Після того як всі необхідні технології були встановлені та підготовлені до розробки, можна починати писати код. Загалом, після процесу підготовки, у backend директорії сформується файлова структура, наведена на рис. 3.5.

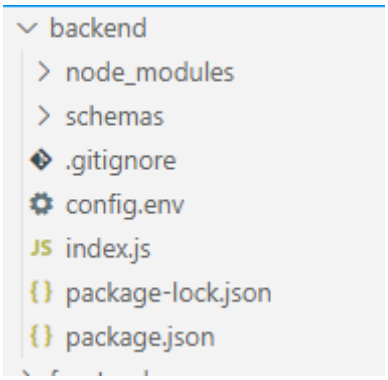

Рис. 3.5 Файлова структура backend директорії

Відповідно, папка node_modules відповідає за встановлені нами раніше залежності серверної частини додатка, вони зберігаються саме в цій папці. У файлі package.json та package-lock.json відповідно описується та вказується така інформація, як версії та назви пакетів та залежностей.

Файл config.env містить в собі чутливу інформацію, яка є необхідною для запуску та налаштування серверу, наприклад ПОРТ та пароль для

підключення до БД.

Файл .gitignore відповідно надає можливість вилучення необхідних файлів, таких як config.env та node_modules, при зберіганні коду у систему контролю версій, адже вони там непотрібні.

У створеній папці schemas знаходиться стандартна структура для зберігання документів, які використовуються в нашому додатка. Ця структура охоплює кожну з основних моделей, а саме: "Користувач", "Лікар" та "Послуга". Кожна з цих моделей містить колекцію документів відповідного типу. Наприклад, модель "Користувач" містить документи, пов'язані з користувачами, такі як профілі користувачів, особисті дані тощо. Модель "Лікар" містить документи, пов'язані з докторами, наприклад, їх кваліфікацію, розклад роботи тощо. А модель "Послуга" містить документи, які описують різні послуги, які надаються в рамках нашого додатка.

На рис. 3.6 та 3.7 показано детальний вигляд моделі "Лікар". В цій моделі містяться всі необхідні документи та інформація, пов'язана з докторами, що включає їх профілі, спеціалізації, контактні дані та інше.

```
const mongoose = require('mongoose');

const doctorSchema = new mongoose.Schema({
  name: {
    type: String,
    required: [true, 'Name is required']
  },
  surname: {
    type: String,
    required: [true, 'Surname is required']
  },
  age: {
    type: Number,
    required: [true, 'Age is required']
  },
  specialization: {
    type: String,
    required: [true, 'Specialization is required']
  },
  tel: {
    type: String,
    required: [true, 'Telephone number is required']
  },
  description: String,
  image: {
    type: String,
    required: [true, 'Image URL is required']
  }
});

const Doctor = mongoose.model('Doctor', doctorSchema);

module.exports = Doctor;
```

Рис. 3.6 Опис моделі «Лікар» в коді

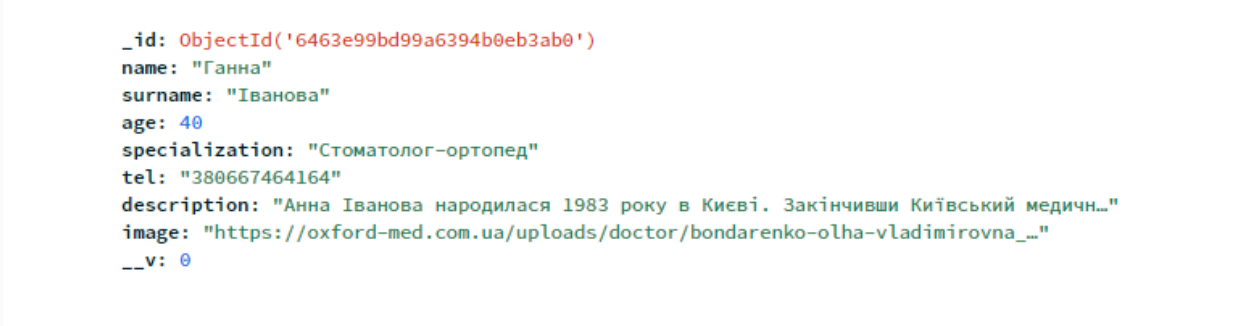

Рис. 3.7 Вигляд моделі «Лікар» у MongoDB Atlas

Така структура дозволяє організувати і зберігати дані в нашому додатка в зручний та логічний спосіб, що сприяє ефективній роботі з ними та

забезпечує належну організацію інформації.

Лікарська модель використовує визначені у схемі властивості докторів, які представлені на рис. 3.6. Згідно з цією схемою, наявні наступні обов'язкові поля: Ім'я (string), Прізвище (string), Вік (number), Спеціалізація (string), Телефон (string), Опис (string) та Фото (string). Всі ці властивості, крім Опису, є обов'язковими для заповнення. Після опису схеми в Mongoose, модель експортується для використання в інших частинах програми за допомогою наступних рядків коду:

```
const mongoose = require('mongoose');
const Doctor = mongoose.model('Doctor', doctorSchema);
module.exports = Doctor;
```

Також, нижче наведено відповідний приклад для моделі «Користувач»:

```
const mongoose = require('mongoose')

const userSchema = new mongoose.Schema({
  username: {
    type: String,
    required: true
  },
  password: {
    type: String,
    required: true
  },
  email: {
    type: String,
    required: true
  },
  appointments: [{
    doctor: String,
    date: String,
    service: String
  }]
})

const User = mongoose.model('User', userSchema)

module.exports = User;
```

Рис. 3.8 Опис моделі «Користувач»

Наступним, і основним, етапом розробки серверної частини веб-додатка є описання логіки ендпоінтів у головному файлі index.js нашого сервера.

Ендпоінт - це певна URL-адреса (шлях) на сервері, за якою доступний певний функціонал або ресурс веб-додатка або API. В іншому розумінні, ендпоінт представляє собою точку доступу до веб-сервера, де можна взаємодіяти з програмним забезпеченням, отримувати або надсилати дані. Наприклад, для отримання списку користувачів з додатка було використано ендпоінт зі шляхом "/users".

Розглянемо наш еднпоінт “/users” детальніше(див. рис. 3.9):

```
app.get("/users", async (req, res) => {
  User.find()
    .then((users) => {
      res.json(users);
    })
    .catch((error) => {
      res.status(500).json({ error: "Failed to retrieve users" });
    });
});
```

Рис. 3.9 Ендпоінт для отримання списку користувачів

Загалом, при GET запиті на цей ендпоінт відбудеться наступне:

1. app.get("/users", async (req, res) => { - Цей рядок встановлює обробник запиту HTTP GET для шляху "/users". Це означає, що коли клієнт зробить GET-запит на "/users", буде виконана наступна функція-обробник.
2. User.find() - Цей код викликає метод find() на моделі User. User є моделлю, яка представляє колекцію користувачів у базі даних. Метод find() шукає всіх користувачів у цій колекції.
3. .then((users) => { - Цей блок виконується, коли метод find() успішно завершується і повертає знайдених користувачів у змінну users.
4. res.json(users); - За допомогою методу json() відповіді res ми відправляємо клієнту знайдених користувачів у форматі JSON.
5. res.status(500).json({ error: "Failed to retrieve users" }); - У випадку помилки, ми встановлюємо статус відповіді HTTP на 500 (Внутрішня помилка сервера) і відправляємо клієнту об'єкт JSON

з повідомленням про помилку "Failed to retrieve users".

Отже, коли клієнт робить GET-запит на шлях "/users", сервер відповідає знайденими користувачами у форматі JSON. У разі помилки, сервер повертає відповідь з повідомленням про помилку.

Але також, можливі інші методи запитів з клієнта на шлях “/users”, наприклад розглянемо запит з методом POST:

```
app.post("/users", async (req, res) => {
  try {
    const { username, password, email } = req.body;
    const candidate = await User.findOne({ username });
    if (candidate) {
      return res
        .status(400)
        .send("Користувач з таким ім'ям вже існує!");
    }
    const hashPassword = bcrypt.hashSync(password, 6);
    const fetchuser = new User({ username, password: hashPassword, email });
    await fetchuser.save();
    const token = jwt.sign({ userId: fetchuser._id }, "secret-key", {
      expiresIn: "24h",
    });

    res.status(200).json({ token });
  } catch (err) {
    console.error(err);
    console.log(req.body);
    res.status(500).send("Server Error");
  }
});
```

Рис. 3.10 Ендпоінт для реєстрації користувача

Таким чином буде оброблятися POST запит для шляху “/users”:

1. app.post("/users", async (req, res) => { - Цей рядок встановлює обробник POST-запиту для шляху "/users". Це означає, що коли клієнт надсилає POST-запит на "/users", буде виконана наступна функція-обробник.
2. const { username, password, email } = req.body; - Звернення до req.body дозволяє отримати дані, які клієнт відправляє в тілі POST-запиту. Тут ми деструктуризуємо req.body і отримуємо значення username, password та email.

3. const candidate = await User.findOne({ username }); - Здійснюється пошук користувача в базі даних за його username за допомогою методу findOne() моделі User.
4. if (candidate) { return res.status(400).send("Користувач з таким ім'ям вже існує!"); } - Якщо знайдений користувач з таким же username, відправляється відповідь зі статусом 400 (Помилковий запит) і повідомленням "Користувач з таким ім'ям вже існує!".
5. const hashPassword = bcrypt.hashSync(password, 6); - Застосовується хешування паролю з використанням бібліотеки bcrypt. Оригінальний пароль password хешується і отримується хешована версія паролю hashPassword. Аргумент 6 визначає раунди хешування.
6. const fetchuser = new User({ username, password: hashPassword, email }); - Створюється новий об'єкт User, в якому властивості username, password і email мають значення з вхідних даних.
7. await fetchuser.save(); - Зберігається новий користувач в базі даних.
8. const token = jwt.sign({ userId: fetchuser._id }, "secret-key", { expiresIn: "24h" }); - Створюється JWT-токен за допомогою бібліотеки jsonwebtoken (jwt). У токені вказується ідентифікатор користувача (userId) та секретний ключ ("secret-key"). Токен також має термін дії 24 години (expiresIn: "24h").
9. res.status(200).json({ token }); - Відправляється відповідь зі статусом 200 (OK) та об'єктом JSON, який містить створений JWT-токен.
10. catch (err) { console.error(err); console.log(req.body); res.status(500).send("Server Error"); }Якщо виникає помилка під час виконання будь-якого кроку в блоку try, виконується блок catch.

Отже, коли клієнт надсилає POST-запит на шлях "/users" з відповідними даними (username, password, email), сервер перевіряє, чи існує користувач з таким самим username. Якщо такий користувач існує, повертається помилка.

Якщо користувач не існує, пароль хешується, новий користувач створюється та зберігається в базі даних, а також генерується та відправляється JWT-токен у відповідь клієнту. У випадку помилки, сервер повертає відповідь з повідомленням про помилку.

3.4 Розробка клієнтської частини

3.4.1 Встановлення технологій для клієнтської частини

У клієнтській частині додатка було використано наступні технології: React, Bootstrap, axios та бібліотеку MaterialUI. Правильно комбінуючи між собою всі ці компоненти, можна отримати повноцінний фронтенд додаток з гарними UI(Користувацький Інтерфейс) та UX(Користувацький досвід) параметрами.

Почати варто з ініціалізації основного компоненту, а саме фреймворку **React**. В межах проекту Oxydent, React було встановлено за допомогою утиліти Create React App. CRA є інструментом, що допомагає швидко створювати нові проекти на React. Він забезпечує початкову налаштування та структуру проекту, що дозволяє розробникам швидко почати писати код без необхідності вручну налаштовувати інфраструктуру React додатків.

Ініціалізація за допомогою CRA має наступні переваги:

- Швидке створення проекту: Create React App автоматично налаштовує початкову структуру проекту з усіма необхідними налаштуваннями та залежностями.
- Налаштування та конфігурація: CRA надає конфігурацію за замовчуванням, яка підходить для більшості випадків використання, але також дозволяє змінювати налаштування згідно з потребами проекту.
- Розробка з підтримкою живого перезавантаження: При розробці CRA автоматично перезавантажує додаток після збереження змін, що дозволяє швидко бачити результати без необхідності вручну оновлювати сторінку.

- Оптимізація для продакшну: При готовності до випуску, CRA дозволяє збирати проект з оптимізованим і мініфікованим кодом, що допомагає покращити продуктивність та завантаження додатків.
- Підтримка сервіс-воркерів та інші функції: CRA надає можливість легко налаштовувати сервіс-воркери, що дозволяють розробникам створювати офлайн-додатки. Він також підтримує останні версії JavaScript та має вбудовану підтримку тестування.

Таким чином, CRA являє собою потужний інструмент для спрощення створення фронтенд додатків та надає змогу одразу приступати до написання логіки додатка.

Процес ініціалізації проекту за допомогою CRA не є складним, і займає лише кілька кроків:

1. Перейти в директорію проекту.
2. Написати наступну команду в терміналі "*npx create-react-app my-app*", де замість *my-app* можна вказати свою назву проекту.
3. Перейти до папки створеного проекту за допомогою *"cd my-app"*.
4. Запустити проект за допомогою *"npm start"*.

Така послідовність дій запустить розробчий сервер та відкриє додаток в браузері. Можна буде бачити зміни, які відбуваються в реальному часі завдяки функціональності живого перезавантаження.

Наступним кроком є додавання до проекту фреймворку **Bootstrap**. Bootstrap - це найпопулярніший інструмент для розробки веб-інтерфейсів. Він надає набір готових стилів, компонентів і JavaScript-інструментів, які допомагають швидко створювати сучасні та респонсивні веб-додатки.

Використання Bootstrap надає наступні переваги:

- Готові компоненти: Bootstrap надає широкий набір готових компонентів, таких як кнопки, форми, навігаційні панелі, каруселі,

модальні вікна та інші.

- Сітка (Grid system): Bootstrap має потужну систему сітки, яка дозволяє легко організовувати елементи на сторінці за допомогою рядків і колонок.
- Респонсивний дизайн: Bootstrap підтримує респонсивний дизайн, що означає, що веб-додаток адаптується до різних пристроїв і розмірів екранів.
- Налаштування та теми: Bootstrap надає можливість налаштовувати та кастомізувати свої стилі за допомогою Sass-змінних або CSS-перезаписів.
- JavaScript-компоненти: Bootstrap містить JavaScript-компоненти, такі як випадаючі меню, модальні вікна, каруселі, переключання вкладок та інші.

Для того щоб додати Bootstrap у створений проект необхідно виконати наступні кроки:

1. В директорії проекту, в термінали встановити Bootstrap за допомогою наступної команди *“npm install bootstrap”*
2. Імпортувати компоненти в необхідних React компонентах за допомогою *“import 'bootstrap/dist/css/bootstrap.css'”*
3. Використання компонентів Bootstrap в проекті

Наступним кроком є встановлення бібліотеки компонентів **MaterialUI**. Material-UI - це популярна бібліотека компонентів для розробки веб-інтерфейсів на React. Вона заснована на дизайні Material Design від Google і надає готові компоненти, стилі і інструменти, що допомагають розробникам створювати сучасні та привабливі користувацькі інтерфейси.

Принцип встановлення дуже схожий на Bootstrap та включає в себе наступні кроки:

1. Введення команди *“npm install @mui/material”*
2. Імпорт компонентів бібліотеки *“import { Button, Typography } from '@mui/material';”*

3. Використання компонентів

Загалом MaterialUI є потужним інструментом для розробки естетичних та функціональних веб-інтерфейсів на основі дизайну Material Design.

Останнім необхідним для розробки компонентом був **axios.** Axios - це бібліотека JavaScript, яка дозволяє здійснювати HTTP-запити з браузера або з середовища Node.js. Вона надає простий та зручний інтерфейс для взаємодії з API і отримання даних з сервера.

Процес встановлення є ідентичним до попередніх компонентів, а основними перевагами є наступні речі:

- Простий у використанні: Axios надає простий і зрозумілий інтерфейс, який дозволяє легко виконувати HTTP-запити.
- Підтримка промісів: Axios використовує проміси для обробки результатів запиту.
- Перехоплення помилок: Axios має вбудовану підтримку перехоплення та обробки помилок під час виконання запиту.
- Передача даних та заголовків: Axios дозволяє легко передавати дані у тілі запиту або у параметрах URL.

3.4.2 Процес розробки клієнтської частини

Коли всі необхідні технології встановлені і готові до використання, можна перейти до написання коду. Після підготовки проекту, структура файлів у директорії frontend буде такою:

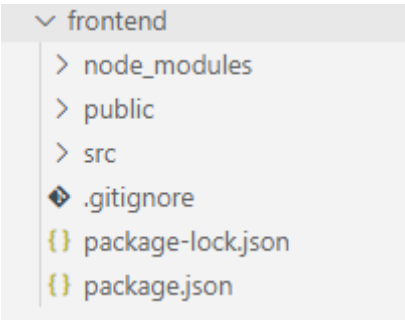

Рис. 3.11 Файлова структура frontend директорії

Призначення файлів та директорій таких як: node_modules, .gitignore та

package.json вже обговорювалося під час описання розробки серверної частини додатка. Зараз варто зазначити для чого потрібні інші папки public та src.

Папка "public" у React-проекті має важливе значення. Вона містить статичні ресурси, які будуть доступні безпосередньо через браузер. Основна причина існування папки "public" полягає в тому, щоб забезпечити доступ до файлів, які не потребують обробки React. Наприклад, це можуть бути зображення, шрифти, стилі CSS, JavaScript-файли тощо. Ці ресурси можуть бути використані безпосередньо в HTML-файлах або імпортовані в React-компоненти. Папка "public" також містить файл "index.html", який є основним файлом шаблону HTML для вашого React-проекту. Цей файл містить кореневий елемент (зазвичай * *), до якого буде рендеритись React-додаток.

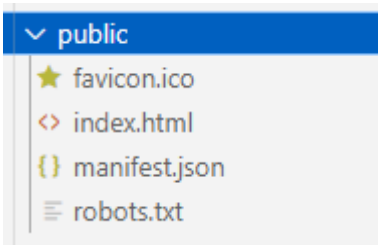

Рис. 3.12 Загальний вигляд директорії public

Папка "src" (або "source") є основною папкою в React-проекті. Вона містить вихідний код додатка React, включаючи компоненти, стилі, зображення, файли даних та інші ресурси. Основна причина існування папки "src" полягає в тому, щоб групувати всі файли, пов'язані з розробкою додатка, в одному місці. У цій папці створюється та організовуються React-компоненти, стилізація, логіка, маршрутизація та інші елементи додатка. Наприклад, можна мати підпапки в "src" для компонентів (src/components), стилів (src/styles), зображень (src/images), файлів даних (src/data) та іншого. Це допомагає зберігати код проекту організованим та легко знаходимим. Папка "src" є місцем, де буде активно вестися розробка React-додатка. Всі зміни, які будуть відбуватися, зазвичай відбуваються саме в цій папці.

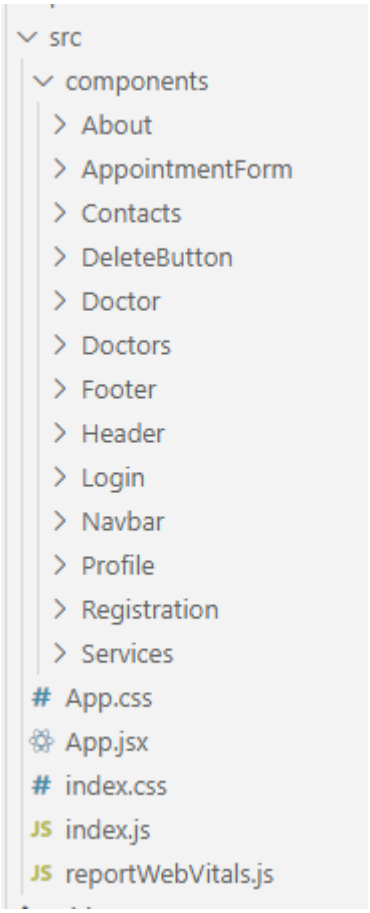

Рис. 3.13 Детальний огляд папки src та її підпапки components

Загалом, більшу частину проекту, як можна побачити на рисунку 3.13 займають саме React компоненти, на яких все і будується. Компонент в React - це незалежна і відокремлена частина користувацького інтерфейсу, яка може бути повторно використана та містить в собі логіку, розмітку та стилізацію. Основна ідея компонентного підходу в React полягає в тому, щоб розділити користувацький інтерфейс на окремі компоненти, які можуть бути розроблені та управляються окремо. Кожен компонент може мати свою внутрішню логіку, стан, властивості (props) та методи, які виконують певні функції.

Нижче наведено приклад, та пояснено компонент Profile.jsx, який відповідає за відображення Особистого кабінету користувача:

```
import Appointments from './Appointments/Appointments';
import classes from './Profile.module.css';
import React, { useEffect, useState } from "react";
import axios from "axios";
```

```
const Profile = () => {
  const [user, setUser] = useState({})
  const token = localStorage.getItem("token");

  if (!token) {
    window.location.href = '/login';
  }

  useEffect(() => {
    const fetchUserInfo = async () => {
      const userInfo = await axios.get("http://localhost:3001/user", {
        headers: {
          Authorization: token,
        },
      });

      if (userInfo) {
        setUser(userInfo.data.user)
      }
    };

    fetchUserInfo();
  }, []);

  return (
    <div className={classes.content}>
      <div className={classes.background}>
        <img
          src="https://helpx.adobe.com/content/dam/help/en/photoshop/using/convert-color-image-black-white/jcr_content/main-pars/before_and_after/image-before/Landscape-Color.jpg"
          className={classes.background_image}
        />
      </div>
      <div className={classes.ava}>
        <img
          src="https://www.pngitem.com/pimgs/m/22-223968_default-profile-picture-circle-hd-png-download.png"
          alt="ava"
        />
      </div>
      <div className={classes.bio}>
        <h3><span className={classes.bio_title}>Ім'я користувача: {user.username}</span></h3>
        <h3><span className={classes.bio_title}>Email: {user.email}</span></h3>
      </div>
      <Appointments />
    </div>
  );
};
```

```
export default Profile;
```

Код використовує деякі імпорти. Зокрема, компонент "Appointments" імпортується з файлу './Appointments/Appointments'. Також, використовуються стилі CSS з файлу './Profile.module.css'. Крім цього, імпортуються пакети React, useEffect, useState та axios.

Сам компонент "Profile" - це функціональний компонент, який використовує хук useState для зберігання стану користувача (об'єкт user). Він також використовує хук useEffect для виконання побічного ефекту - отримання інформації про користувача з сервера після завантаження компонента.

В тілі компонента є умовна перевірка наявності токену, для перевірки користувача на авторизацію. Якщо токен відсутній, відбувається перенаправлення на сторінку '/login'. Далі, у блоку useEffect відбувається виконання функції fetchUserInfo, яка здійснює запит до сервера за допомогою axios для отримання інформації про користувача. Отримані дані про користувача зберігаються у стані компонента за допомогою функції setUser.

У рендерингу компонента повертається JSX-розмітка, яка включає різні елементи і дані профілю користувача, такі як фонове зображення, аватар, ім'я користувача та електронна пошта. Крім того, включений компонент "Appointments", який припускається, що він відображає розклад зустрічей користувача. Загалом, компонент "Profile" відповідає за відображення профілю користувача та взаємодію з сервером для отримання даних.

Також розберемо компонент Login, який призначено для авторизації користувача:

```
import s from "./Login.module.css";
import "bootstrap/dist/css/bootstrap.min.css";
import Button from "@mui/material/Button";
import React, { useState } from "react";
import axios from "axios";

const Login = (props) => {
  const [username, setUsername] = useState("");
```

```
const [password, setPassword] = useState("");
const [error, setError] = useState("");

const handleSubmit = async (e) => {
  e.preventDefault();
  try {
    const response = await axios.post("http://localhost:3001/login", {
      username,
      password,
    });

    const { token } = response.data;

    localStorage.setItem("token", token);
    window.location.href = '/';
  } catch (error) {
    setError("Невірний логін або пароль");
  }
};

return (
  <div className={s.content}>
    <form onSubmit={handleSubmit}>
      <div className="mb-3">
        <label htmlFor="username" className="form-label">
          Логін
        </label>
        <br />
        <input
          className="form-control"
          type="text"
          placeholder="Name"
          value={username}
          onChange={(event) => setUsername(event.target.value)}
        />
      </div>
      <div className="mb-3">
        <label htmlFor="password" className="form-label">
          Пароль
        </label>
        <br />
        <input
          className="form-control"
          type="password"
          placeholder="Password"
          value={password}
          onChange={(event) => setPassword(event.target.value)}
        />
      </div>
      {error && <p className={s.error}>{error}</p>}
```

```
        <Button variant="contained" color="primary" type="submit">
          Увійти
        </Button>
      </form>
    </div>
  );
};

export default Login;
```

У цьому компоненті з’являються деякі нові імпорти. Зокрема, імпортується компонент "Button" з пакету "@mui/material/Button".

Сам компонент "Login" - це функціональний компонент, який використовує хук useState для зберігання стану полів "username" та "password", а також стану "error", який відповідає за відображення помилки.

У компоненті є функція handleSubmit, яка обробляє подання форми. При виклику цієї функції відбувається асинхронний POST-запит до сервера за допомогою axios для автентифікації користувача. В разі успішної автентифікації, отриманий токен зберігається у локальному сховищі, а сторінка перенаправляється на головну сторінку.

3.5 Безпека веб-додатків

Забезпечення безпеки відіграє надзвичайно важливу роль у всіх веб-додатках та веб-сайтах, оскільки загрози хакерських нападів і порушень даних зростають щодня. Кібер-злодії використовують різні методи, такі як фішинг та злам, щоб отримати доступ, який має обмежуватися належним чином, до облікових даних та кредитних карток. Вразливі маршрути можуть стати джерелом атак для зловмисників. Особливо важливо забезпечити надійну безпеку веб-сайтів, які здійснюють фінансові операції або надають свої послуги в Інтернеті. Безпека повинна бути високим пріоритетом, оскільки навіть маленькі порушення можуть призвести до великих втрат для замовника та його клієнтів.

Процес передачі даних від користувачів до сервера завжди потрібно пильно прослідковувати та зробити його максимально безпечним. Не є

рідкими випадки, коли користувачі додатків виявляються зловмисниками, які намагаються завдати шкоди серверу або викрасти конфіденційну інформацію з БД. Тому дані, що надсилаються на сервер, обов'язково повинні пройти перевірку на наявність можливих загроз безпеці. Цю перевірку можна здійснити як на стороні клієнта, так і на стороні сервера, але перевірка на стороні клієнта не є достатньою для захисту від шкідливих атак. Тому для всіх програм, що містять дані користувача, обов'язковою є перевірка на стороні сервера, як показано в попередніх прикладах.

CSRF-атаки здійснюються шляхом викрадення сеансів зареєстрованих користувачів і відправки підроблених запитів на реальний сервер. Ці атаки не дозволяють крадіжки даних, але можуть змінювати дані користу вача, такі як електронна пошта, пароль або навіть здійснювати фінансові операції. Для захисту від таких атак можна використовувати пакет вузлів під назвою "csurf". Цей пакет генерує маркер CSRF, який вбудовується в усі форми, що змінюють дані користувача на сервері. Потім цей маркер надсилається на сервер у кожному запиті і перевіряється за допомогою "csurf". Якщо маркер є дійсним, запит обробляється, а якщо ні, то він відхиляється. Це дозволяє запобігти виконанню запитів з фальшивих сайтів з викраденими сеансами, де маркер CSRF відсутній.

Щоб захистити файли cookie від несанкціонованого доступу, рекомендується застосовувати підписані файли з важкодоступним секретом та встановлювати різні параметри при ініціалізації сеансу. Наприклад, параметри "secure" та "httpOnly" дозволяють передавати файли cookie лише через захищене з'єднання HTTPS і забороняють доступ до них через скрипти JavaScript. Крім того, параметри домену та шляху використовуються для порівняння домену файлу cookie з доменом сервера і надсилають файли cookie лише на відповідні запити. З метою зменшення ризику неправильного використання, термін дії файлів cookie може бути обмежений певним проміжком часу.

MongoDB Atlas - це безпечний хмарний сервіс для зберігання даних,

який використовує різноманітні заходи безпеки для запобігання атакам на базу даних. При підключенні бази даних до сервера для кожного кластера встановлюються користувачі з різними ролями, які необхідно налаштувати. Тільки авторизований користувач з відповідними дозволами може виконувати операції читання, запису, оновлення та видалення даних у базі даних. Крім того, MongoDB Atlas дозволяє доступ до бази даних лише з IP-адрес, які перелічені у білому списку проекту. Для успішного підключення до бази даних необхідно вказати IP-адресу машини, на якій розгорнутий проект.

3.6 Демонстрація веб-сайту

Нижче наведені приклади сторінок веб-сайту:

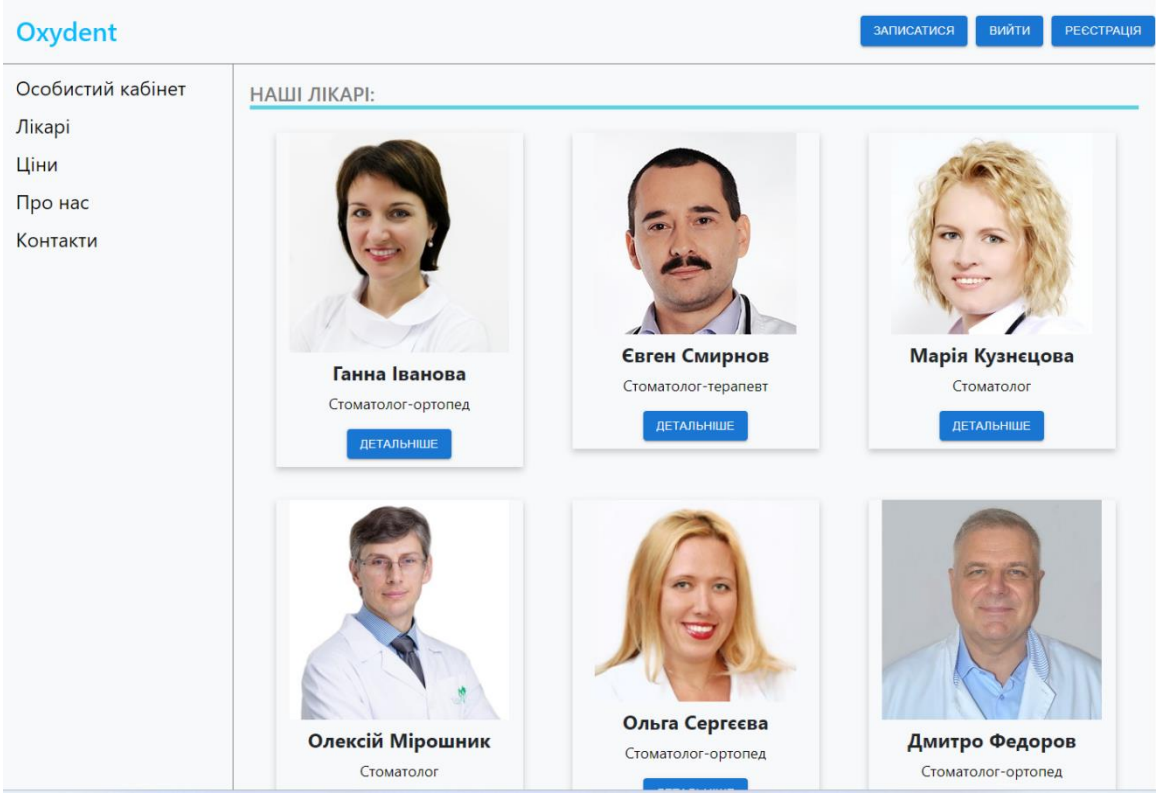

Рис. 3.14 Демонстрація сторінки /doctors

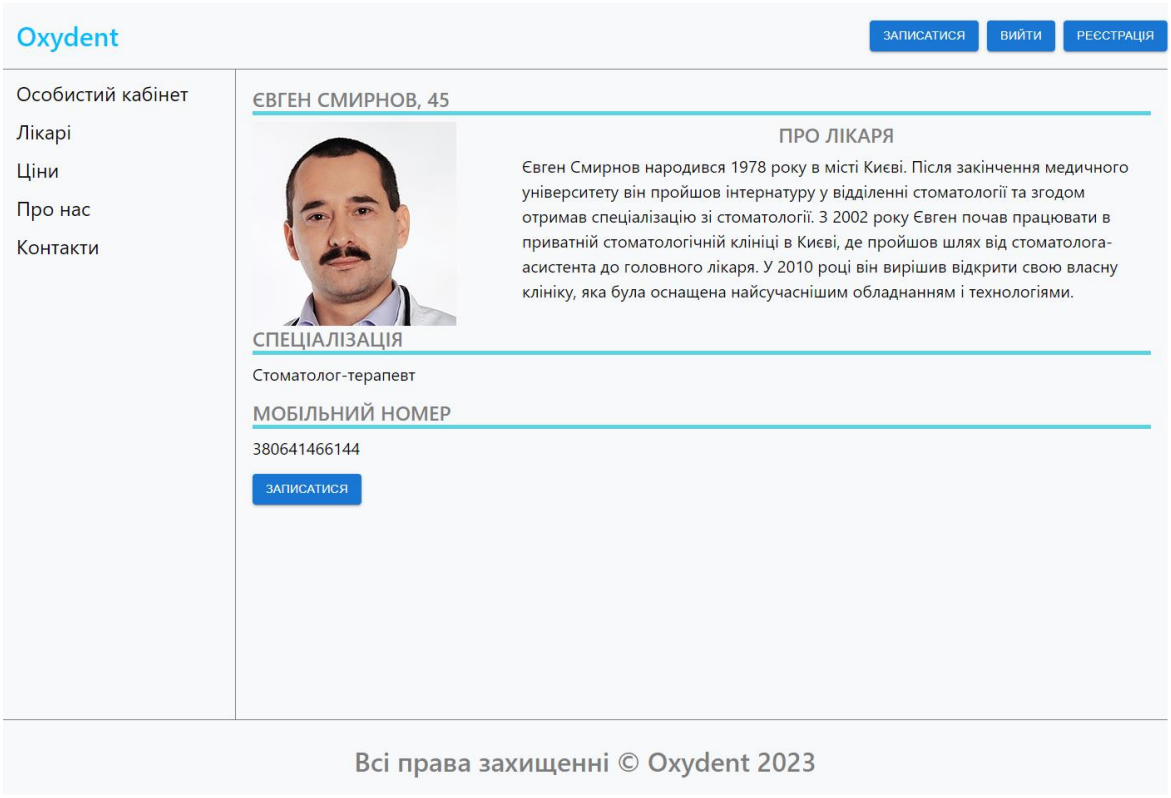

Рис. 3.15 Демонстрація унікальної сторінки лікаря /doctors/id

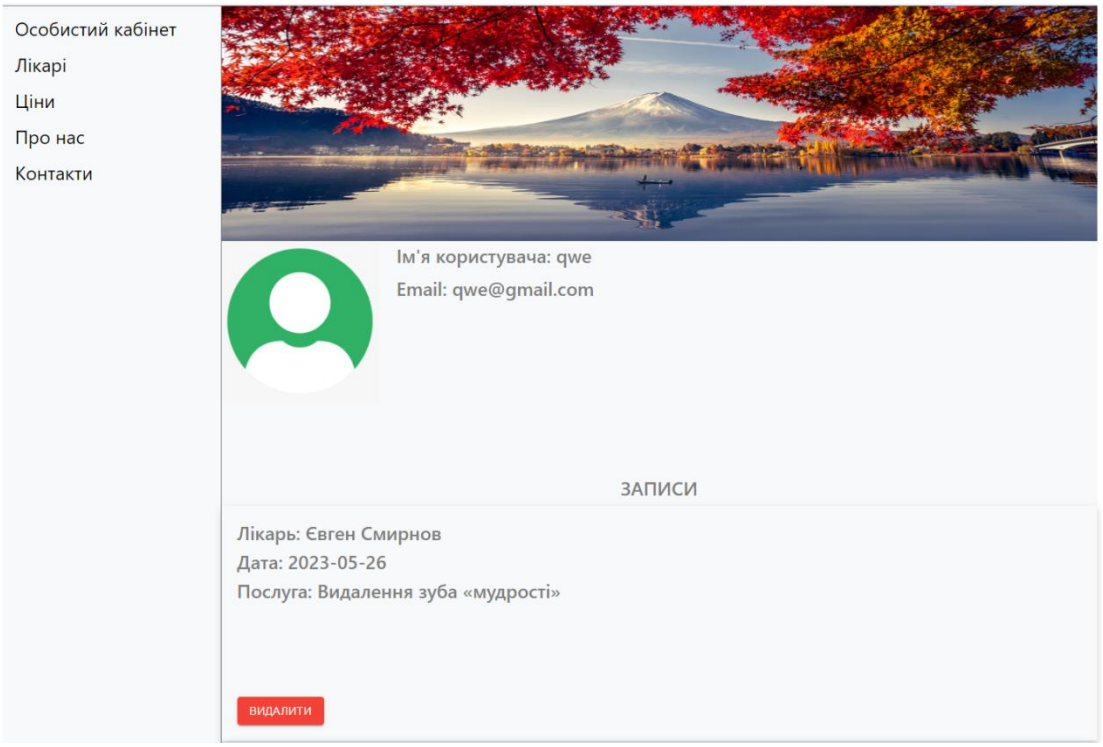

Рис. 3.16 Сторінка особистого кабінету користувача /account

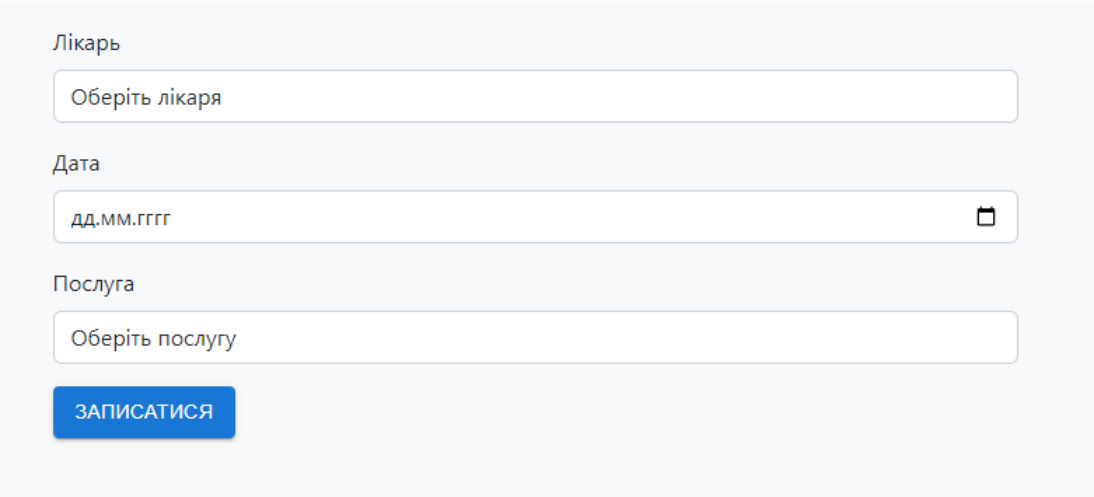

Рис. 3.17 Форма запису до лікаря на сторінці /appointment

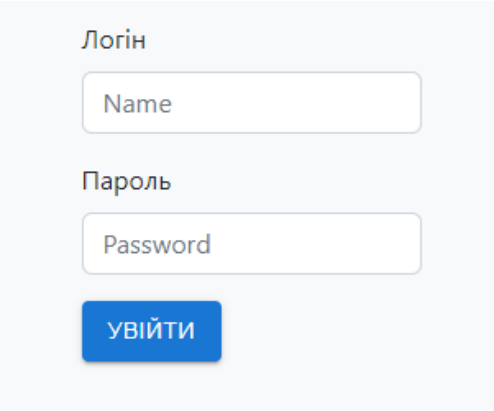

Рис. 3.18 Форма авторизації /login

Висновки до розділу 3

У цьому розділі було описано процес розробки веб-сайту стоматологічної поліклініки Oxydent, а також його програмну логіку. Для організації логіки було використано шаблон програмування client-server, що дозволяє обмін даними між клієнтською та серверною частинами додатка.

Для забезпечення безпеки та ідентифікації користувачів була реалізована система аутентифікації на сервері за допомогою JWT та bcrypt. Також були встановлені і інші необхідні пакети та бібліотеки, підключена база даних MongoDB та використаний фреймворк Express для розробки серверної сторони. Для забезпечення безпечної роботи додатка були встановлені всі необхідні пакети та реалізовані відповідні заходи безпеки. Головною метою даного розділу було розроблення та демонстрація прототипу веб-сайту

стоматологічної поліклініки.

На клієнтській частині додатка, комбінуючи такі технології як: React, Bootstrap та MaterialUI було досягнуто наступних речей:

- Зручна та швидка розробка інтерфейсу: Обидва фреймворки (Bootstrap та Material UI) надають готові компоненти, які значно спрощують створення користувацького інтерфейсу.
- Консистентний та привабливий дизайн: Обидва фреймворки надають готові стилізовані компоненти, які допомагають забезпечити консистентний та привабливий вигляд додатка.
- Сумісність з React: Обидва фреймворки побудовані на основі React, тому вони добре інтегруються з існуючими React-проектами.

ВИСНОВКИ

Метою кваліфікаційної роботи було реалізувати проект веб-сайту стоматологічної поліклініки використовуючи MERN стек підхід у розробці. Головним завданням було вивчення кожного окремого компоненту MERN стеку, а також порівняння з аналогічними технологіями.

Автор кваліфікаційної роботи вже володів певними знаннями та досвідом використання МП JavaScript та React, але є новачком у використанні бекенд-фреймворків. Тому велику частину часу було витрачено на вивчення таких технологій та фреймворків як: Node.js, Express та MongoDB. В ході дослідження були оцінені переваги та недоліки використання цих технологій.

Під час виконання кваліфікаційної роботи, було виявлено, що Javascript Fullstack підхід, а саме MERN стек, є одним з найкращих, на даний момент, підходів до розробки різного роду веб-сайтів та веб-додатків.

У стеку MERN (MongoDB, Express, React, Node.js) кожен компонент виконує свою роль у розробці веб-додатків. Ось підсумування ролей кожного компоненту:

- MongoDB: MongoDB є документ-орієнтованою базою даних, яка зберігає дані у вигляді JSON-подібних документів. Вона використовується для збереження, організації та керування даними вашого додатка.
- Express: Express є фреймворком для побудови веб-додатків на основі Node.js. Він надає простий та ефективний спосіб роботи з маршрутизацією, обробкою запитів та відправленням відповідей на серверній стороні.
- React: React - це бібліотека JavaScript для створення користувацьких інтерфейсів. Вона дозволяє розробникам створювати компоненти, які представляють окремі частини інтерфейсу та забезпечують швидку та ефективну рендерізацію змін.

- Node.js: Node.js - це середовище виконання JavaScript на сервері. Воно дозволяє виконувати JavaScript-код на серверній стороні, що дозволяє побудову повноцінних веб-додатків. Node.js надає широкі можливості для роботи з мережевими запитами, файловою системою та іншими операціями на сервері.

Під час використання стеку MERN, також необхідно враховувати певні недоліки:

- Складність налаштування: Одним з недоліків стеку MERN є його складність налаштування для початківців. Кожен компонент стеку має свою власну конфігурацію, і правильне налаштування та забезпечення взаємодії між ними може вимагати певного рівня технічного досвіду.
- Потреба в знаннях з багатьох технологій: Використання стеку MERN передбачає роботу з різними технологіями, такими як JavaScript, MongoDB, Express і React. Це означає, що розробник повинен мати глибокі знання в кожній з цих областей, що може бути вимогливим для новачків.
- Безпека: Використання стеку MERN також вимагає належної уваги до питань безпеки. Недостатньо правильно налаштувати безпеку бази даних, сервера або додатка. Незахищені точки взаємодії можуть стати потенційними місцями для атак зловмисників.
- Потреба в масштабуванні: Іноді масштабування додатка, побудованого на стеці MERN, може бути складним завданням. Залежно від розміру проекту і обсягу даних, необхідно приділити достатньо уваги оптимізації його продуктивності та масштабованості.

СПИСОК БІБЛІОГРАФІЧНИХ ПОСИЛАНЬ ТА ВИКОРИСТАНИХ ДЖЕРЕЛ

1. “Google Search Statistics” [Електронний ресурс]. Режим доступу: https://99firms.com/blog/google-search-statistics/ (дата звернення 05.05.2023) - Назва з екрану.
2. “Веб-сайт поліклініки Viva” [Електронний ресурс]. Режим доступу: https://viva.clinic/ua (дата звернення 07.05.2023) - Назва з екрану.
3. “HTML Layout: Creating a Simple Page Layout Design” [Електронний ресурс]. Режим доступу: https://www.bitdegree.org/learn/html-layout (дата звернення 08.05.2023) - Назва з екрану.
4. “2022 Developer Survey” [Електронний ресурс]. Режим доступу: https://survey.stackoverflow.co/2022 (дата звернення 08.05.2023) - Назва з екрану.
5. “Quick Start” [Електронний ресурс]. Режим доступу: https://react.dev/learn (дата звернення 08.05.2023) - Назва з екрану.
6. “Nodejs Event-loop” [Електронний ресурс]. Режим доступу: https://www.geeksforgeeks.org/node-js-event-loop (дата звернення 09.05.2023) - Назва з екрану.
7. “Client-server Model” [Електронний ресурс]. Режим доступу: https://www.geeksforgeeks.org/client-server-model (дата звернення 09.05.2023) - Назва з екрану.
8. “The Definitive Guide, 6th Edition”, Девід Фленаган: навч. посіб., 2011. – 1100 ст., ISBN 978-0-596-80552-4.
9. “Beginning Node.js”, Басарат Саїд: навч. посіб., 2014. – 326 ст., ISBN 9781484201886
10. “Інструменти Reactjs на всі випадки життя” [Електронний ресурс]. Режим доступу: https://codeguida.com/post/3084 (дата звернення

10.05.2023) - Назва з екрану.

11. “MongoDB in Action”, Кайл Бенкер: навч. посіб., 2012. – 287 ст., ISBN 9785457572584.
12. “ПОЧАТОК РОБОТИ З NODE.JS І EXPRESS” [Електронний ресурс]. Режим доступу: https://www.8host.com/blog/nachalo-raboty-s-node-js-i-express (дата звернення 15.05.2023) - Назва з екрану.