

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
УМАНСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ САДІВНИЦТВА
Факультет економіки і підприємництва

Кафедра інформаційних технологій

Допущено до захисту

"_____ " _____ 2023 р.

Зав. кафедри, к.т.н., доцент

_____ Лішук Р.І.

КВАЛІФІКАЦІЙНА РОБОТА
МАГІСТРА

Інформаційна система для приймального відділення медичних
закладів

Виконав – студент 2 курсу, групи 21м-кн
освітньої програми "Комп'ютерні науки"
спеціальності 122 Комп'ютерні науки

Танасов Костянтин Віталійович

Науковий керівник – д.т.н., професор

Кулаков Павло Ігорович

Умань – 2023

Реферат
на кваліфікаційну роботу «Інформаційна система для приймального
відділення медичних закладів»

студента факультету економіки та підприємництва
Уманського національного університету садівництва

Танасова Костянтина Віталійовича

Структура та обсяг роботи - пояснювальна записка до кваліфікаційної роботи магістра складається з 3 розділів, містить 78 сторінок, 38 рисунків, 6 додатків, використано 50 літературних джерел.

Кваліфікаційна робота магістра присвячена розробці інформаційної системи для приймального відділення медичних закладів. В процесі виконання роботи проведено аналіз способів побудови інформаційних систем, в результаті якого було обрано архітектуру, сформульовано основні вимоги до інформаційної системи, її складові, та необхідний функціонал. Об'єктом дослідження є процес обміну даними в інформаційних системах, предметом дослідження роботи є методи та засоби обміну даними через різні платформи. Метою дослідження кваліфікаційної роботи магістра є вдосконалення методів комунікації в інформаційних системах за рахунок використання сучасних технологій проектування, таких як web-push-notification, firebase, PWA. Наукова новизна отриманих результатів полягає у тому, що удосконалено метод кросплатформеного обміну даними. Практичне значення одержаних результатів полягає в тому, що розроблений додаток дозволяє забезпечити оперативну співпрацю медичного персоналу.

У першому розділі кваліфікаційної роботи магістра проведено аналіз способів побудови прогресивного додатку, у другому розділі здійснено вибір архітектури програмного забезпечення інформаційної системи, у третьому розділі розроблено програмне забезпечення інформаційної системи.

Анотація

Кваліфікаційна робота магістра присвячена розробці інформаційної системи для приймального відділення медичних закладів. Проведено аналіз способів побудови інформаційних систем, в результаті якого було обрано архітектуру, сформульовано основні вимоги до інформаційної системи, її складові, та необхідний функціонал. Досліджено процес обміну даними в інформаційних системах, методи та засоби обміну даними через різні платформи, вдосконалено методи комунікації в інформаційних системах за рахунок використання сучасних технологій, таких як web-push-notification, firebase, PWA. Наукова новизна отриманих результатів кваліфікаційної роботи магістра полягає у тому, що удосконалено метод кросплатформеного обміну даними.

Ключові слова: інформаційна система, приймальне відділення, медичний заклад, інформаційне забезпечення.

Annotation

The master's qualification work is devoted to the development of an information system for the reception department of medical institutions. An analysis of the ways of building information systems was carried out, as a result of which the architecture was chosen, the main requirements for the information system, its components, and the necessary functionality were formulated. The process of data exchange in information systems, methods and means of data exchange through various platforms were studied, methods of communication in information systems were improved through the use of modern technologies, such as web-push-notification, firebase, PWA. The scientific novelty of the obtained results of the master's work is that the method of cross-platform data exchange has been improved.

Keywords: information system, reception department, medical institution, information support.

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
УМАНСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ САДІВНИЦТВА

Факультет економіки і підприємництва

Кафедра Інформаційних технологій

Освітня програма Комп'ютерні науки

Спеціальність 122 Комп'ютерні науки

(код, назва)

ЗАТВЕРДЖУЮ
Завідувач кафедри

Ліщук Р.І.

(підпис)

(ініціали, прізвище)

" " 2023 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

Студенту Танасову Костянтину Віталійовичу

(прізвище, ім'я, по батькові)

1. Тема проекту Інформаційна система для приймального відділення медичних закладів

керівник роботи Кулаков Павло Ігорович, д.т.н., професор кафедри інформаційних технологій

(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджені наказом по університету від "28" квітня 2023 року № 01-06/77

2. Строк подання студентом роботи 20.12.2023

3. Вихідні дані до роботи Розробити кросплатформений та кросбраузерний додаток інформаційної системи для забезпечення роботи приймального відділення медичного закладу з використанням технологій проектування web-push-notification, firebase, PWA. Для обміну інформацією використовувати технологію push-notification.

4. Перелік питань, які потрібно розробити Аналіз способів побудови прогресивного додатку, вибір архітектури програмного забезпечення інформаційної системи, розробка програмного забезпечення інформаційної системи, тестування.

5. Перелік графічного матеріалу 38 рисунків, 6 додатків, 50 джерел

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв

7. Дата видачі завдання _____

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів роботи	Строк виконання етапів роботи	Примітка
1.	Вибір теми кваліфікаційної роботи	05.09.2022	
2.	Одержання індивідуального завдання	07.09.2022	
3.	Складання календарного плану написання кваліфікаційної роботи	08.09.2022	
4.	Підготовка до виконання кваліфікаційної роботи - підбір та вивчення літератури - участь у виконанні науково-дослідних робіт - виконання курсових робіт - інші заходи	Впродовж I семестру	
5.	Уточнення теми кваліфікаційної роботи та календарного плану-графіка, виходячи зі специфіки базового підприємства, установи	До початку II семестру	
6.	Підготовка: - першого розділу - другого розділу - третього розділу - висновків	09.02.2023 23.03.2023 26.04.2023 10.05.2023	
7.	Доопрацювання кваліфікаційної роботи з урахуванням зауважень наукового керівника	18.05.2023	
8.	Написання та оформлення роботи в остаточному варіанті	25.05.2023	
9.	Перевірка роботи на академічний плагіат	12.06.2023	
10.	Одержання відгуку наукового керівника	13.06.2023	
11.	Попередній захист роботи на кафедрі	14.06.2023	
12.	Затвердження роботи завідувачем кафедри	15.06.2023	
13.	Одержання рецензії	16.06.2023	
14.	Захист кваліфікаційної роботи	19-21.06.2023	

Студент _____ **Танасов К.В.**
(підпис) (прізвище та ініціали)

Керівник роботи _____ **Кулаков П.І.**
(підпис) (прізвище та ініціали)

Зміст

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СКОРОЧЕНЬ І ТЕРМІНІВ.....	7
ВСТУП.....	8
РОЗДІЛ 1. АНАЛІТИЧНИЙ ОГЛЯД ПРЕДМЕТНОЇ ОБЛАСТІ.....	11
1.1. Поняття прогресивного веб-додатку та їх переваги.....	11
1.2. Відмінність прогресивного веб-додатку від нативних додатків.....	12
1.3. Постановка задачі.....	14
РОЗДІЛ 2. ВИБІР АРХІТЕКТУРИ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ІНФОРМАЦІЙНОЇ СИСТЕМИ.....	15
2.1. Визначення структурної організації інформаційної системи.....	15
2.2. Аналіз сучасних методів створення прогресивних веб-додатків.....	18
2.3. Вибір архітектури програмного комплексу.....	21
РОЗДІЛ 3. РОЗРОБКА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ІНФОРМАЦІЙНОЇ СИСТЕМИ.....	25
3.1. Розробка бази даних.....	25
3.2. Програмна реалізація клієнтської частини	33
3.3. Тестування інформаційної системи.....	59
ВИСНОВОК.....	65
СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ.....	67
ДОДАТОК А Схема інформаційної системи.....	71
ДОДАТОК Б Лістинг головної сторінки застосунку.	72
ДОДАТОК В Головна сторінка застосунку.	74
ДОДАТОК Г Блок схема входу в систему.	75
ДОДАТОК Д Блок схема виходу з системи.	76
ДОДАТОК Е Лістинг серверної частини.	77

Перелік умовних позначень, скорочень і термінів

ІТ	Інформаційні технології.
ІС	Інформаційна система.
САУ	Система автоматичного управління.
БД	База даних.
ПЗ	Програмне забезпечення.
ПД	Програмний додаток.
СУБД	Система управління базами даних.

Вступ

Інформаційні технології у світі застосовуються повсюдно. Охорона здоров'я не стала винятком. Сучасні ІТ розробки надають позитивний вплив в розвитку нових методів організації медичної допомоги населенню. Проведення телеконсультацій пацієнтів та персоналу, обмін інформацією про хворих між різними установами, дистанційне фіксування фізіологічних параметрів, контроль за проведенням операцій у реальному часі всі ці можливості дає впровадження інформаційних технологій у медицину.

Актуальність теми дослідження обумовлена тим, що впровадження інформаційних технологій у сферу охорони здоров'я дозволяє покращити якість обслуговування, помітно прискорити роботу персоналу та знизити витрати на обслуговування для пацієнтів. Завдяки впровадженню інформаційних технологій медицина набула сьогодні абсолютно нових рис. Цей процес супроводжується суттєвими змінами в медичній теорії та практиці, пов'язаними з внесенням коректив до підготовки медичних працівників. ІТ допомагають лікарю проводити об'єктивну діагностику захворювань, накопичувати й ефективно використовувати отриману інформацію на всіх стадіях лікувального процесу і, що найважливіше для медичної науки, є неоціненними у науковому пізнанні.

Сучасна лікарня є складною функціональною системою, у якій в реальному часі реалізується безліч різних процесів. Інформаційна система, являючи собою програмно-технічний комплекс, що готує і забезпечує процеси збирання, зберігання й обробки інформації, дозволяє не лише об'єднати в єдиний простір усі інформаційні потоки та ресурси медичного закладу, а й забезпечити новий рівень взаємодії як між персоналом, так і між персоналом та пацієнтами.

Інформаційні системи полегшують збирання клінічних записів пацієнтів, діагностичних сканів, записів медичних працівників, скорочують час та зусилля, необхідні для зберігання інформації та створення журналу даних, необхідного для медичних аудитів та контролю якості. Все це дозволяє знизити витрати часу

при обстеженні та лікуванні хворого, сприяє оптимізації клінічного робочого процесу.

Процес лікування пацієнта відбувається із залученням значної кількості співробітників: лікар, вузькі фахівці, координатори, молодший медичний персонал. Їхні дії повинні бути синхронізовані. Оперативне інформування про події, що відбуваються з пацієнтом, допомагає не лише прискорити лікування, а іноді і врятувати життя пацієнту. Особливо це стосується роботи приймального відділення, де комунікація між його співробітниками та лікарями, що перебувають у відділеннях, виходить на перший план.

Об'єкт дослідження кваліфікаційної роботи магістра — процес обміну даними в інформаційних системах.

Предмет дослідження кваліфікаційної роботи магістра — методи та засоби обміну даними через різні платформи.

Метою дослідження кваліфікаційної роботи магістра є вдосконалення методів комунікації в інформаційних системах за рахунок використання сучасних технологій проектування, таких як: web-push-notification, firebase, PWA. На відміну від існуючих аналогів розроблюваний застосунок є кросплатформним та кросбраузерним, що значно спрощує процес імплементації та вартість розробки, а для обміну інформацією використано технологію push-notification, яка працює як в браузерах так і на телефонах, в фоновому чи активному режимах.

Задачі дослідження:

- здійснити аналіз способів побудови інформаційних систем;
- запропонувати ефективний підхід для реалізації простого і зрозумілого в користуванні кросплатформного додатку;
- налаштування ефективної комунікації між всіма ланками медичного персоналу.

Методи дослідження - використовувались методи створення масштабованих web-додатків; методи розмітки гіпертекстових документів; методи побудови таблиць стилів CSS; методи створення прикладного

програмного інтерфейсу. Під час програмної реалізації web-додатку використано мови програмування та бібліотеки: React, NodeJs, Express, JSX, JavaScript, і віддалену базу даних Postgre

Наукова новизна отриманих результатів кваліфікаційної роботи магістра полягає у тому, що удосконалено метод кросплатформеного обміну даними.

Практичне значення одержаних результатів кваліфікаційної роботи магістра полягає в тому, що розроблений додаток дозволяє забезпечити оперативну співпрацю медичного персоналу.

РОЗДІЛ 1. АНАЛІТИЧНИЙ ОГЛЯД ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Поняття прогресивного веб-додатку та їх переваги

Прогресивна веб-програма (PWA) — найкращий спосіб для розробників прискорити завантаження та підвищити продуктивність своїх веб-додатків. PWA—це веб-сайт, який використовує сучасні веб-стандарти та дозволяє вам встановлювати його на комп'ютери або пристрої ваших користувачів. При використанні вони як додатки. Найвідоміший приклад - Twitter, який запустив сайт mobile.twitter.com як PWA React та Node.js.

PWA (Progressive Web App) — це веб-технологія, яка перетворює веб-сайти на програми. Його можна розмістити на головному екрані телефону прямо з браузера, і він надсилатиме push-повідомлення та отримуватиме доступ до апаратного забезпечення пристрою. Все це може використовуватись при нестабільному підключенні або офлайн [1-9].

Термін PWA або прогресивна веб-програма походить від технічного жаргону, але насправді є наступним кроком у створенні зручних програм. Вони поєднують у собі зручність і дружній інтерфейс програми, а розробляти їх так само легко, як і звичайний сайт. Прогресивні програми можна назвати адаптивними сайтами, оскільки вони підлаштовуються під можливості браузера користувача. Вони можуть автоматично покращувати вбудовані функції браузера, щоб зробити сайт більш схожим на рідну веб-програму[4]. Основні компоненти PWA:

- маніфест веб-програм, що надає власні функції, такі як посилання в вигляді іконки програми на робочому столі;
- скрипт Service Workers для виконання фонових завдань та роботи в автономному режимі;
- архітектура оболонки програми для швидкого завантаження через сервіс-воркер.

Progressive Web Application - поєднує властивості нативної програми з функціональністю браузера і має свої переваги: додоювання та моменту закінчення доїння і автоматичного або ручного зняття доїльних стаканів.

- PWA можуть запускатись у найпопулярніших операційних системах -Windows, iOS та Android-і можуть бути завантажені на комп'ютери, смартфони, планшети та пристрої у торгових залах;

- нові функції та оновлення випускаються розробниками віддалено, для того щоб користувачі могли користуватись змінами та покращеннями, користувачам ці оновлення завантажуються в фоновому режимі;

— прогресивні веб-програми індексуються Google та іншими пошуковими системами;

- робота в автономному режимі можлива завдяки сценаріям сервіс-воркерів та стратегіям кешування, які запускаються браузером у фоновому режимі;

- оскільки зовнішній інтерфейс відокремлений від внутрішнього, на розробку та реструктуризацію дизайну та логіки взаємодії PWA зклієнтами витрачається менше часу та ресурсів;

- PWA можна встановлювати без PlayMarket або App Store хоч встановлення програм з невідомих джерел заборонено, але антивірусні програми лояльні до PWA, а передача даних протоколу HTTPS є безпечною;

- з лютого 2019 року прогресивні веб додатки можуть бути завантаженими з App Store та Google Play, дозволяючи користувачам завантажувати програми зі своїх звичайних джерел [10-23].

1.2 Відмінність прогресивного веб-додатку від нативних додатків

Нативна програма — це програмне забезпечення, створене розробниками для використання на певній платформі чи пристрої. Оскільки розробники створюють нативну програму для використання на певному пристрої та його ОС, вона має можливість використовувати апаратне та програмне забезпечення для конкретного пристрою.

PWA це сайт, представлений у вигляді мобільного додатка. Він має всі функції встановленої програми, але працює безпосередньо у вашому браузері, що робить його схожим на Google Docs.

Представлено основні відмінності між PWA та нативними програмами:

- незалежність від інтернет-з'єднання;
- прогресивність;
- нативність.

Відмінною рисою PWA є те, що вони є прогресивними, оскільки це означає, що вони не обмежені традиційними додатками. Це означає, що PWA є максимально прогресивними, працюють у всіх наявних операційних системах, а також PWA можуть працювати в будь-якому браузері. PWA не можна було б назвати прогресивним, якби він не міг адаптуватися до середовища користувача. Отже, що відрізняє PWA від традиційних додатків, так це їхнє поступове поліпшення, яке гарантує, що вони будуть працювати у всіх існуючих сьогодні браузерах.

Сьогодні користувачі виходять в Інтернет зі смартфонів, планшетів, стаціонарних комп'ютерів, ноутбуків та інших пристроїв. При публікації PWA передбачалося, що всі користувачі робитимуть це знастільного ПК, але також було прийнято адаптивний дизайн, що перебудовує сайт за розмірами та параметрами сайту [23-25]. Адаптивний дизайн робить ваш PWA прогресивним і доступним на безлічі пристроїв.

Доступність додатку без інтернет-з'єднання є ключовим елементом PWA. Всі ми знаємо, що сайти не доступні при відключенні інтернету, але багато нативних програм, як і раніше, можуть працювати, незважаючи на низьку швидкість інтернету або його повну відсутність. PWA дозволяють користувачам взаємодіяти з програмами незалежно від підключення до інтернету. Це стало можливим завдяки попереднього кешування даних програми. Це кешування здійснюється за допомогою технології, яка називається сервісними працівниками. Надає програмний спосіб кешування ресурсів.

Хоча прогресивні веб-застосунки виходять за межі звичайних веб-програм, необхідно відмітити, що вони повинні підтримувати структуру, подібну до звичайних застосунків. Це одна з найважливіших відмінностей між PWA та веб-сайтами. Статичний контент, такий як контактна інформація, повідомлення в блогах та послуги. Щоб сайт розглядався як PWA, він може бути доданим на робочий стіл пристрою та містити інтерактивні функції, з якими може взаємодіяти користувач. Ви можете використовувати веб-застосунок як нативну програму.

1.3 Постановка задачі

Ціллю магістерської кваліфікаційної роботи є розробка інформаційної системи для приймального відділення медичних закладів. Для її реалізації необхідно здійснити аналіз способів побудови інформаційних систем, запропонувати ефективний підхід для реалізації простого і зрозумілого у користуванні кросплатформеного додатку, забезпечити налаштування ефективної комунікації між всіма ланками медичного персоналу за допомогою розробленого програмного забезпечення.

Завданням розробки є створення на інформаційної системи, яка призначена для використання у медичних закладах, та забезпечує вдосконалення методів комунікації за рахунок використання сучасних технологій, таких як web-push-notification, firebase, PWA. Розроблюваний застосунок повинен бути кросплатформеним та кросбраузерним, що значно спростить процес імплементації та вартість розробки, а для обміну інформацією використати технологію push-notification, яка працює як у браузерах так і в смартфонах, в фоновому чи активному режимах.

РОЗДІЛ 2. ВИБІР АРХІТЕКТУРИ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ІНФОРМАЦІЙНОЇ СИСТЕМИ

2.1 Визначення структурної організації інформаційної системи

В даному розділі описано розробку архітектури інформаційної системи. Запропонована реалізація архітектури з використанням сучасних технологій проектування. Представлена архітектура програмного комплексу, опис клієнтської та серверної частини прогресивного додатку, використані бібліотеки для зв'язку даних.

Інформаційна система - інтегрований набір компонентів для збору, зберігання й обробки даних, а також для надання інформації, знань і цифрових продуктів. Багато підприємств використовують інформаційні технології для завершення та керування своїми операціями, взаємодії зі своїми споживачами та випередження конкурентів. Деякі компанії сьогодні повністю побудовані на інформаційних технологіях, як-от eBay, Amazon, Alibaba та Google.

Інформаційні системи з роками набули величезної популярності в бізнесі. Майбутнє інформаційних систем та їх важливість залежить від автоматизації та впровадження технології ІІІ [26-30].

Розглянемо складові інформаційної системи:

- апаратне забезпечення;
- програмне забезпечення;
- дані;
- телекомунікації.

Апаратне забезпечення — це фізичний компонент технології. Сюди входять комп'ютери, жорсткі диски, клавіатури, iPad тощо. Вартість апаратного забезпечення швидко знизилася, а його швидкість і ємність пам'яті значно зросли. Проте сьогодні величезне занепокоєння викликає вплив використання обладнання на навколишнє середовище. Сьогодні послуги зберігання

пропонуються з хмари, до якої можна отримати доступ із телекомунікаційних мереж.

Програмне забезпечення може бути двох типів: системне та прикладне. Системне програмне забезпечення - це операційна система, яка керує обладнанням, програмними файлами та іншими ресурсами, пропонуючи користувачеві керувати ПК за допомогою графічного інтерфейсу користувача. Прикладне програмне забезпечення призначене для керування певними завданнями користувачів. Коротше кажучи, системне програмне забезпечення робить апаратне забезпечення придатним для використання, тоді як прикладне програмне забезпечення виконує конкретні завдання.

Прикладом системного програмного забезпечення є Microsoft Windows, а прикладного програмного забезпечення - Microsoft Excel.

Великі компанії можуть використовувати ліцензовані програми, які розроблені та керовані компаніями-розробниками програмного забезпечення для задоволення своїх конкретних потреб. Програмне забезпечення може бути пропрієтарним і відкритим, доступним у мережі для безкоштовного використання.

Дані являють собою набір фактів і марні самі по собі, але коли їх зібрати й упорядкувати разом, вони можуть бути дуже потужними для бізнес-операцій. Компанії збирають усі дані та використовують їх для прийняття рішень, які можна проаналізувати на предмет ефективності бізнес-операцій.

Телекомунікації використовуються для підключення до комп'ютерної системи або інших пристроїв для поширення інформації. Мережу можна встановити за допомогою дротового або бездротового режимів. Дротові технології включають оптоволокну та коаксіальний кабель, тоді як бездротові технології включають радіохвилі та мікрохвилі [31, 32].

Існують різні інформаційні системи, і тип інформаційної системи, яку використовує бізнес, залежить від його мети та завдання. Ось чотири основні типи інформаційних систем:

- системи підтримки операцій;
- інформаційні системи управління;
- системи підтримки прийняття рішень;
- системи виконавчої інформації.

Перший тип інформаційної системи — це система підтримки операцій. Такий тип інформаційної системи в основному підтримує певний вид діяльності в бізнесі. Прикладом є система обробки транзакцій, яка використовується в усіх банках світу. Цей тип інформаційної системи дозволяє постачальнику послуг оцінити конкретний бізнес-процес.

Друга категорія інформаційних систем це — інформаційні системи управління, що складається з інтеграції апаратного та програмного забезпечення, що дозволяє організації виконувати свої основні функції. Вони допомагають отримувати дані з різних онлайн-систем. Отримані таким чином дані не зберігаються системою; швидше, його аналізують продуктивно, щоб допомогти в управлінні організацією.

Система підтримки прийняття рішень може допомогти організації приймати обґрунтоване рішення щодо своєї діяльності. Вона аналізує інформацію, що швидко змінюється, яку неможливо визначити заздалегідь. Її можна використовувати в повністю автоматизованих системах і системах, керованих людиною. Однак для досягнення максимальної ефективності рекомендується поєднання систем, керованих людиною та комп'ютером.

EIS або система підтримки виконавчої влади є останньою категорією, яка слугує системами підтримки управління. Вони допомагають приймати рішення на вищому рівні для організації.

Для вирішення завдань поставлених перед розроблюваною інформаційною системою підходить тип - інформаційна система управління.

Розроблювана система повинна містити в собі такі ознаки:

- кросплатформеність;
- кросбраузерність;

- адаптивність.

Вони дозволяють використовувати її майже в будь-якому апаратному середовищі, що значно спрощує і здешевлює процес імплементації та реалізації. Основними функціями інформаційної системи є обмін інформацією між медичним персоналом (рис. 2.1), збереження та обробка даних, ведення та управління графіками чергувань.



Рисунок 2.1 - Схема обміну інформацією між персоналом

2.2 Аналіз сучасних методів створення прогресивних веб-додатків

Прогресивна веб-програма - це веб-програма, яка надає користувачам досвід, подібний до програми, використовуючи сучасні веб-можливості. Зрештою, це просто звичайний веб-сайт, який працює у браузері з деякими вдосконаленнями [33 - 40]. Це дає вам можливість:

- встановити його на головний екран мобільного телефону;
- отримати доступ до нього в автономному режимі;
- отримати доступ до камери;
- отримувати push-повідомлення;
- зробити фонову синхронізацію.

Однак, щоб мати можливість перетворити традиційну веб-програму на PWA, ми маємо її трохи налаштувати, додавши файл маніфесту веб-програми та сервіс-воркер.

Маніфест веб-програми - це простий файл JSON, який інформує браузер про вашу веб-програму. Він розповідає, як він повинен поводитися, коли встановлено на мобільний пристрій або робочий стіл користувача. А щоб показати підказку «Додати на головний екран», потрібен маніфест веб-програми (рис. 2.2).

Зрештою, це просто файл JSON з деякими обов'язковими та необов'язковими властивостями:

- name: коли браузер запускає екран-заставку, на екрані відображатиметься ім'я.
- short_name: це буде назва, яка відображатиметься під ярликом вашої програми на головному екрані;
- start_url: це буде сторінка, яка буде показана користувачеві, коли ваш додаток відкрито;
- дисплей: він повідомляє браузеру, як відображати програму. Є кілька режимів, як-от мінімальний інтерфейс, повноекранний режим, браузер тощо.

```
"lang": "en",
"dir": "ltr",
"name": "This is my awesome PWA",
"short_name": "myPWA",
"icons": [
  {
    "src": "/assets/images/touch/android-chrome-192x192.png",
    "sizes": "192x192",
    "type": "image/png"
  }
],
"theme_color": "#1a1a1a",
"background_color": "#1a1a1a",
"start_url": "/",
"display": "standalone",
"orientation": "natural"
```

Рисунок 2.2 - Структура файлу manifest.json

Тут ми використовуємо автономний режим, щоб приховати все, що стосується браузера.

- background_color: коли браузер запускає екран-заставку, він буде фоном екрана;
- theme_color: це буде колір тла рядка стану, коли ми відкриваємо програму;
- орієнтація: повідомляє браузеру орієнтацію, яку потрібно мати під час відображення програми;
- значки: коли браузер запускає екран-заставку, на екрані відображатиметься значок, тут можна використати усі розміри, щоб відповідати бажаній піктограмі будь-якого пристрою.

Service Worker - це сценарій, який ваш браузер виконує у фоновому режимі в окремому потоці. Це означає, що він працює в іншому місці та повністю відокремлений від вашої веб-сторінки. Ось чому він не може маніпулювати вашим елементом DOM (Рис. 2.3).



Рисунок 2.3 - Функція Service worker

Однак він дуже корисний. Сервіс-воркер може перехоплювати й обробляти мережеві запити, керувати кеш-пам'яттю, щоб увімкнути підтримку в автономному режимі, або надсилати push-повідомлення вашим користувачам.

Сервіс-воркери включені за замовчуванням у всіх сучасних браузерах.

Щоб запускати код за допомогою сервісних воркерів, вам потрібно буде обслуговувати свій код через HTTPS. Сервісні воркери можуть працювати лише через HTTPS з міркувань безпеки. Потрібен сервер, що підтримує HTTPS. Для розміщення експериментів ви можете використовувати такі служби, як GitHub, Netlify, Vercel тощо. Щоб полегшити локальну розробку, браузері також вважають що локальний хост безпечне джерело.

2.3 Вибір архітектури програмного комплексу

Для реалізації поставленої задачі було вирішено використовувати триланкову архітектуру, яка складається з таких компонентів: клієнт, сервер і база даних. Схема даної архітектури зображена на (рис. 2.4).

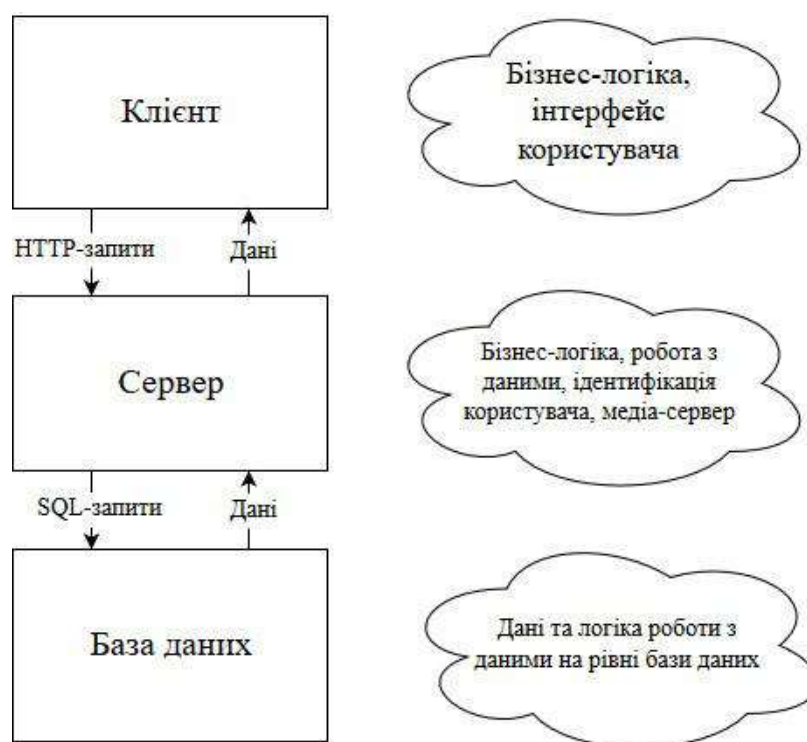


Рисунок 2.4 - Триланкова архітектура програмного комплексу

Основна перевага трьох рівнів архітектури клієнт-сервер полягає в тому, що ці рівні розробляються та підтримуються незалежно, і це не вплине на інші рівні у разі будь-якої модифікації. Це забезпечує кращу продуктивність і навіть

більшу масштабованість архітектури, оскільки зі збільшенням попиту можна додавати більше серверів.

Рівень презентації - це інтерфейс користувача та найвищий рівень в архітектурі. Його мета - прийняти запит від клієнта та відобразити інформацію клієнту. Він спілкується з іншими рівнями за допомогою веб-браузера, надаючи вихідні дані в браузері. Якщо говорити про веб-рівні, то вони розроблені з використанням таких мов, як HTML, CSS, JavaScript.

Рівень додатків - це середній рівень архітектури, також відомий як рівень логіки, оскільки тут детально обробляється інформація/запит, зібраний через рівень презентації. Він також взаємодіє з сервером, який зберігає дані. Він обробляє запит клієнта, форматує його та надсилає назад клієнту. Він розроблений з використанням таких мов, як Python, Java, PHP тощо.

Рівень даних - це останній рівень архітектури, також відомий як рівень бази даних. Він використовується для зберігання обробленої інформації, щоб її можна було отримати пізніше, коли це буде потрібно. Він складається з серверів баз даних, таких як Oracle, MySQL, DB2 тощо. Зв'язок між рівнем презентації та рівнем даних здійснюється за допомогою середнього рівня, тобто рівня додатків.

Розглянемо основні переваги та недоліки даного підходу.

Переваги:

- зберігається логічне розділення між рівнем презентації, рівнем додатків і рівнем бази даних;
- підвищення продуктивності, оскільки завдання розподілено на кілька машин у розподілених машинах і, крім того, кожен рівень не залежить від інших рівнів;
- зростаючий попит на додавання додаткових серверів також можна впоратися в архітектурі, оскільки рівні можна масштабувати незалежно;
- розробники не залежать від оновлення технології одного рівня, оскільки це не вплине на інші рівні;
- надійність покращується завдяки незалежності рівнів, оскільки проблеми одного рівня не впливатимуть на інші;

- програмісти можуть легко підтримувати базу даних, код презентації та бізнес/логіку програми окремо, якщо потрібні будь-які зміни в бізнес-логіці/логіці програми, це не впливає на код презентації та кодову базу;
- навантаження збалансовано, оскільки завдання рівня презентації відокремлено від сервера рівня даних;
- безпека покращена, оскільки клієнт не може безпосередньо спілкуватися з рівнем бази даних, крім того, дані перевіряються на рівні додатків перед передачею на рівень бази даних;
- цілісність даних зберігається;
- забезпечення розгортання різноманітних баз даних, а не обмеження однієї конкретної технології.

Недоліки:

- рівень презентації не може безпосередньо спілкуватися з рівнем бази даних;
- складність також зростає зі збільшенням рівнів в архітектурі;
- існує збільшення кількості ресурсів, оскільки кодову базу, код презентації та код програми потрібно підтримувати окремо.

NodeJS використовує архітектуру «Single Threaded Event Loop» для обробки кількох одночасних запитів клієнтів. Коли запит надходить на сервер Node, він працює як керована подіями модель обробки JavaScript із механізмом зворотного виклику. На сервері NodeJs є різні частини.

Запит надходить зі сторони клієнта, і користувач хоче отримати відповідь від сервера, щоб діяти. Ці дії можуть бути неблокуючими простими або вони можуть блокувати складні завдання.

NodeJS використовується на стороні сервера. Він приймає вхідний запит клієнта від користувачів, обробляє запити та відповідає цьому конкретному клієнту чи користувачеві.

У NodeJS черга подій зберігає запити та передає їх один за одним у цикл подій.

Цикл подій отримує запит із черги подій, обробляє запит і повертає

відповідь конкретному клієнту.

Пул потоків складається з доступних потоків на нашому сервері NodeJS, які виконуватимуть деякі завдання для виконання запиту клієнта.

Зовнішні ресурси — це ті, які повинні виконувати операції блокування в запитах клієнта. Ці ресурси можуть бути для обчислень, зберігання даних тощо.

Веб-сервер NodeJS отримуватиме запити від клієнтів, ці запити можуть містити блокуючі або неблокувальні операції. Сервер отримає ці запити та додасть їх до черги подій.

У черзі подій ці запити оброблятимуться один за одним і передаватимуться в цикл подій. Цикл подій перевірить, чи запит містить будь-які операції блокування, чи це простий запит.

Якщо запит блокує, його буде передано до пулу потоків, інакше він обробить простий запит, наприклад опитування вводу-виводу, і поверне відповідь клієнту.

Один потік із пулу потоків призначається одному складному запиту. Цей потік відповідатиме за виконання конкретної операції блокування за допомогою зовнішніх ресурсів, таких як обчислення, база даних, файлова система тощо. Після завершення завдання він передасть відповідь у цикл подій, а потім цикл подій надішле відповідь до клієнта.

РОЗДІЛ 3. РОЗРОБКА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ІНФОРМАЦІЙНОЇ СИСТЕМИ

3.1. Розробка бази даних

PostgreSQL - це провідна система баз даних з відкритим вихідним кодом, яка підтримує тисячі веб-сайтів, служб і програм. Це ACID-сумісна RDBMS. Іншими словами, він пропонує властивості атомарності, послідовності, ізоляції та довговічності. Розширені функції PostgreSQL включають збережені процедури, тригери, визначені користувачем функції, транзакції та реплікацію. PostgreSQL забезпечує продуктивність, доступність і функціональність масштабованості корпоративного рівня.

Таким чином, ви можете розгорнути PostgreSQL у розподіленій архітектурі, яка може обробляти великі обсяги даних [41].

Ще одна важлива особливість PostgreSQL полягає в тому, що він має відкритий вихідний код, що робить його доступним рішенням для бізнесу, щоб зберегти низькі витрати. Крім того, PostgreSQL має чудову історію надійності та безпеки даних. Він використовувався у виробництві клієнтами протягом десятиліть, і йому довіряють багато організацій і компаній у всьому світі. PostgreSQL має сильну сумісність із широким спектром інструментів і технологій [42 - 45].

Ви можете легко інтегрувати його з різними програмами. Він також підтримується сторонніми інструментами, розширеннями та іншими мовами програмування, такими як Python.

Графічний інтерфейс клієнта pgAdmin показано на (рис 3.1). Після успішного логіна створюється тільки одна база даних — postgres. Для створення нових баз потрібно написати команди за допомогою консольного клієнта psql або використовуючи графічний клієнт pgAdmin.

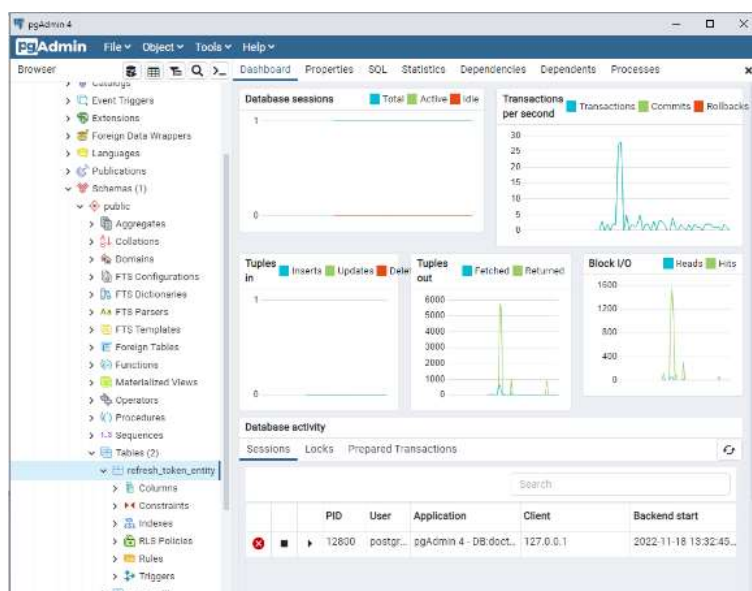


Рисунок 3.1 - Графічний інтерфейс клієнта pgAdmin

Для створення додаткової бази даних, необхідно вказати її назву та написнути кнопку зберегти (рис. 3.2).

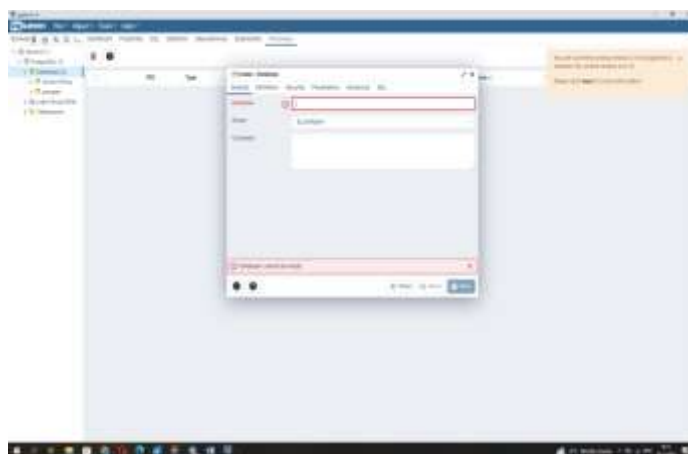


Рисунок 3.2 - Створення бази даних

Після створення бази даних, таблиці для збереження інформації будуть заповнюватись через сервер (Рис. 3.3).

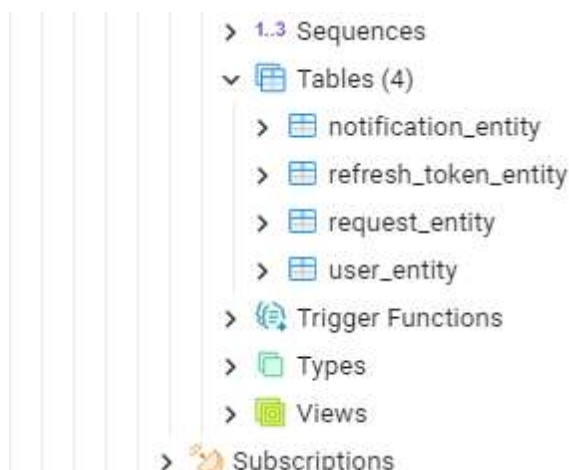


Рисунок 3.3 - Створені таблиці

TypeORM — це бібліотека TypeScript ORM (об'єктно-реляційний маппер), яка дозволяє легко зв'язати вашу програму TypeScript із базою даних реляційної бази даних. TypeORM підтримує MySQL, SQLite, Postgres, MS SQL Server та безліч інших традиційних варіантів. TypeORM може працювати на платформах NodeJS, Browser, Cordova, PhoneGap, Ionic, React Native, NativeScript, Expo та Electron, а також Використовується в TypeScript та JavaScript (ES5, ES6, ES7, ES8). Його мета — постійна підтримка новітніх функцій JavaScript та надання додаткових функцій, які допомагають розробляти програми, що використовують бази даних, від невеликих застосунків із кількома таблицями до великих корпоративних застосунків із кількома базами даних.

TypeORM підтримує як шаблони Active Record, так і Data Mapper, на відміну від усіх інших ORM JavaScript, які зараз існують, що означає, що ви можете писати високоякісні, слабо пов'язані, масштабовані та підтримувані програми найбільш продуктивним способом. Для встановлення typeorm необхідно виконати команду `yarn add typeorm` або `npm install typeorm — save`. Для підключення до бази даних необхідно викликати метод `createConnection` який перед цим імпортувати з бібліотеки typeorm — `import { createConnection } from 'typeorm';`

В метод `createConnection` необхідно передати об'єкт з налаштуваннями для

підключення до PostgreSQL:

```
export const AppDataSource: DataSourceOptions = {
  type: 'postgres',
  host: process.env.POSTGRES_HOST,
  port: Number(process.env.POSTGRES_PORT),
  username: process.env.POSTGRES_USERNAME,
  password: process.env.POSTGRES_PASSWORD,
  database: process.env.POSTGRES_DATABASE,
  synchronize: true,
  logging: false,
  entities: [env === 'production' ? 'build/entity/*.ts,.js' : 'src/entity/*.ts,.js'],
  migrations: [env === 'production' ? 'build/migration/*.ts,.js' :
    'src/migration/*.ts,.js'],
  subscribers: [env === 'production' ? 'build/subscriber/*.ts,.js' :
    'src/subscriber/*.ts,.js'],
};
```

Поле `type` визначає назву бази даних з якою буде створене з'єднання. Поле `port` визначає порт на якому запущена база даних. Властивості `username` і `password` визначають ім'я і пароль які було використано під час реєстрації в PostgreSQL. Властивість `database` визначає ім'я таблиці яку було створено в PostgreSQL. Поле `synchronize` дозволяє впевнитись в тому, що сутності синхронізуватимуться з базою даних щоразу, коли програму буде запущено. Властивість `logging` дозволяє ввімкнути логування всіх запитів і помилок, просто встановивши `logging: true` у параметрах джерела даних, також за допомогою додаткових параметрів, можна вказати на конкретні дані для логування.

```
{
  host: "localhost",
  logging: ["query", "error"]
}
```

Властивість `entities` вказує на сутності, які були створені в коді для їх подальшого відображення в базі даних.

Поле `migrations` - це лише один файл із запитом `sql` для оновлення схеми бази даних і застосування нових змін до існуючої бази даних.

Властивість `subscribers` встановлює методи зі спеціальною логікою, які прослуховують певні події сутності.

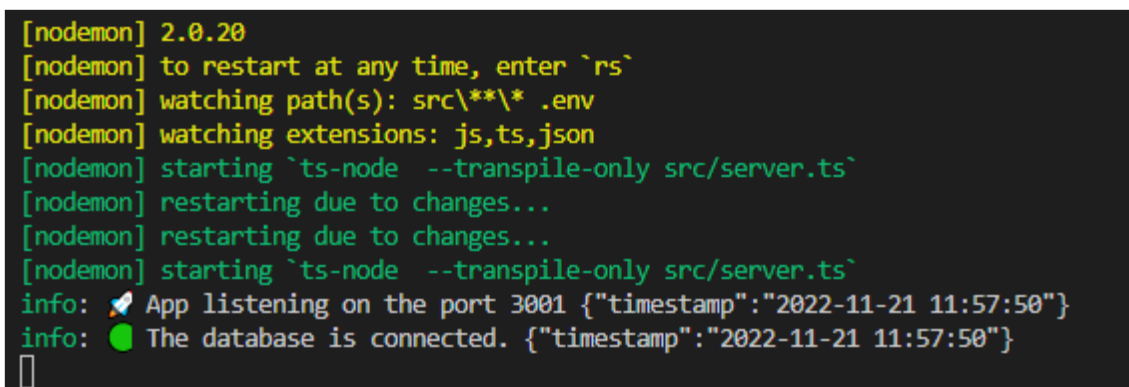
Таким чином, після виконання функції з вказаними параметрами:

```
private connectToDatabase() {
  createConnection(AppDataSource)
    .then(() => {
      logger.info('● The database is connected.');
```

```
    })
    .catch((error: Error) => {
      logger.error('● Unable to connect to the database: ${error}.');
```

```
    });
}
```

В консоль буде виведено результат .на (рис. 3.4).



```
[nodemon] 2.0.20
[nodemon] to restart at any time, enter `rs`
[nodemon] watching path(s): src\**\* .env
[nodemon] watching extensions: js,ts,json
[nodemon] starting `ts-node --transpile-only src/server.ts`
[nodemon] restarting due to changes...
[nodemon] restarting due to changes...
[nodemon] starting `ts-node --transpile-only src/server.ts`
info: 🚀 App listening on the port 3001 {"timestamp":"2022-11-21 11:57:50"}
info: ● The database is connected. {"timestamp":"2022-11-21 11:57:50"}
█
```

Рисунок 3.4 - Підключення до бази даних

Сутності - це клас, який відображається на таблицю бази даних (або колекцію при використанні MongoDB). Створити сутність можна визначивши

новий клас і позначивши його за допомогою @Entity():

```
@Entity()
export class RequestEntity {
  @PrimaryGeneratedColumn()
  id: number;

  @Column()
  @IsNotEmpty()
  requestData: string;

  @Column()
  @IsNotEmpty()
  petitionerId: string;

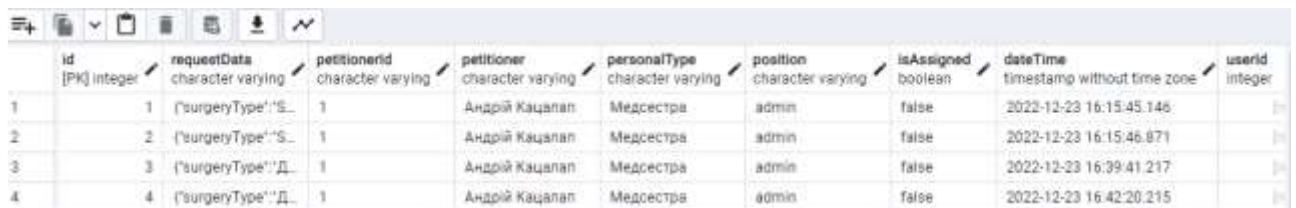
  @Column()
  @IsNotEmpty()
  petitioner: string;

  @Column()
  @IsNotEmpty()
  personalType: string;

  @Column()
  @IsNotEmpty()
  position: string;

  @Column()
  @IsNotEmpty()
  isAssigned: boolean;
}
```

Це створить наступну таблицю бази даних (Рис. 3.5).



	id [PK] integer	requestData character varying	petitionerId character varying	petitioner character varying	personalType character varying	position character varying	isAssigned boolean	dateTime timestamp without time zone	userId integer
1	1	('surgeryType':'S...	1	Андрій Кацалап	Медсестра	admin	false	2022-12-23 16:15:45.146	
2	2	('surgeryType':'S...	1	Андрій Кацалап	Медсестра	admin	false	2022-12-23 16:15:46.871	
3	3	('surgeryType':'Д...	1	Андрій Кацалап	Медсестра	admin	false	2022-12-23 16:39:41.217	
4	4	('surgeryType':'Д...	1	Андрій Кацалап	Медсестра	admin	false	2022-12-23 16:42:20.215	

Рисунок 3.5 - Створена таблиця в базі даних

Базові сутності складаються зі стовпців і відносин. Кожна сутність повинна мати primary column (або стовпець ObjectId, якщо використовується MongoDB).

Кожна сутність має бути зареєстрованою у параметрах джерела даних:
entities: ["entity/*.js"],

Для прикладу буде представлено сутність для користувача :

```
@Entity()
@Unique(['email'])
export class UserEntity implements User {
  @PrimaryGeneratedColumn()
  id: number;

  @Column()
  @IsNotEmpty()
  email: string;

  @Column()
  @IsNotEmpty()
  name: string;

  @Column()
  @IsNotEmpty()
  surname: string;

  @Column()
  @IsNotEmpty()
  position: string;

  @Column()
  @IsNotEmpty()
  phone: string;

  @Column()
  @IsNotEmpty()
  role: string;
```

```

@Column()
@IsNotEmpty()
isTemporary: boolean;
@Column()
@IsNotEmpty()
password: string;
@Column()
@CreateDateColumn()
createdAt: Date;
@Column()
@UpdateDateColumn()
updatedAt: Date;
@OneToOne(=> RefreshTokenEntity)
@JoinColumn()
refreshToken: RefreshTokenEntity
}

```

В даній сутності описана таблиця `user_entity` яка зберігається в базі даних, та зберігає в собі інформації стосовно користувача. Рисунок 3.6



	id	email	name	surname	position	phone	role	isTemporary	password	createdAt	updatedAt
1	5	tarjavelaia@pg...	Test	Katziarp	Chester	+38 (066) 764-62...	user	true	32b51059e9f4f...	2022-11-16 14:28:33.870746	2022-11-16 14:28:33.870746
2	1	andriakatsarap@pg...	Andree	Katziarp	test	+38 (036) 764-62...	admin	false	32b51059e9f4f...	2022-11-14 21:48:26.527064	2022-11-16 16:38:30.781664

Рисунок 3.6 - Підключення до бази даних

Спеціальний параметр `@Unique(['email'])` в сутності означає що поле `email` в базі, повинне бути унікальним значенням для таблиці `user_entity`. Поле `PrimaryGeneratedColumn` вказує на те що, поле `id` є автогенерованим значенням і буде використовуватись як `id`. Поле `@Column()` вказує на те що це значення буде займати в таблиці окрему колонку, і буде самостійним окремим значенням.

Властивість `@IsNotEmpty()` вказує на те, що задане поле, не може бути пустим, і не мати ініціалізованого значення, якщо спробувати зберегти сутність в базу, без колонки з цим параметром, з'явиться помилка, яка заборонить такі дії. `@CreateDateColumn` — це спеціальний стовпець, який автоматично встановлюється на дату введення сутності. Тому не потрібно встановлювати цей стовпець — він буде встановлений автоматично. `@UpdateDateColumn` — це спеціальний стовпець, який автоматично встановлюється на час оновлення сутності кожного разу, коли ви викликаєте збереження менеджера сутності або сховища. Вам не потрібно встановлювати цей стовпець — він буде встановлений автоматично.

3.2 Програмна реалізація клієнтської частини

Для ініціалізації React проекту необхідно виконати одну з команд (рис 3.7). Yarn — це новий менеджер пакетів, який замінює існуючий робочий процес для клієнта `npm` або інших менеджерів пакетів, залишаючись сумісним із реєстром `npm`. Він має той самий набір функцій, що й існуючі робочі процеси, але працює швидше, безпечніше та надійніше [46 - 50].

```
yarn create react-app client
# або
npm init react-app client
# або
npx create-react-app client
```

Рисунок 3.7 - Команди для ініціалізації проекту

Основною функцією будь-якого менеджера пакунків є встановлення деякого пакета — фрагмента коду, який служить певній меті — з глобального реєстру в локальне середовище розробника. Кожен пакет може залежати або не залежати від інших пакетів. Типовий проект може мати десятки, сотні або навіть тисячі пакетів у своєму дереві залежностей (рис. 3.8).

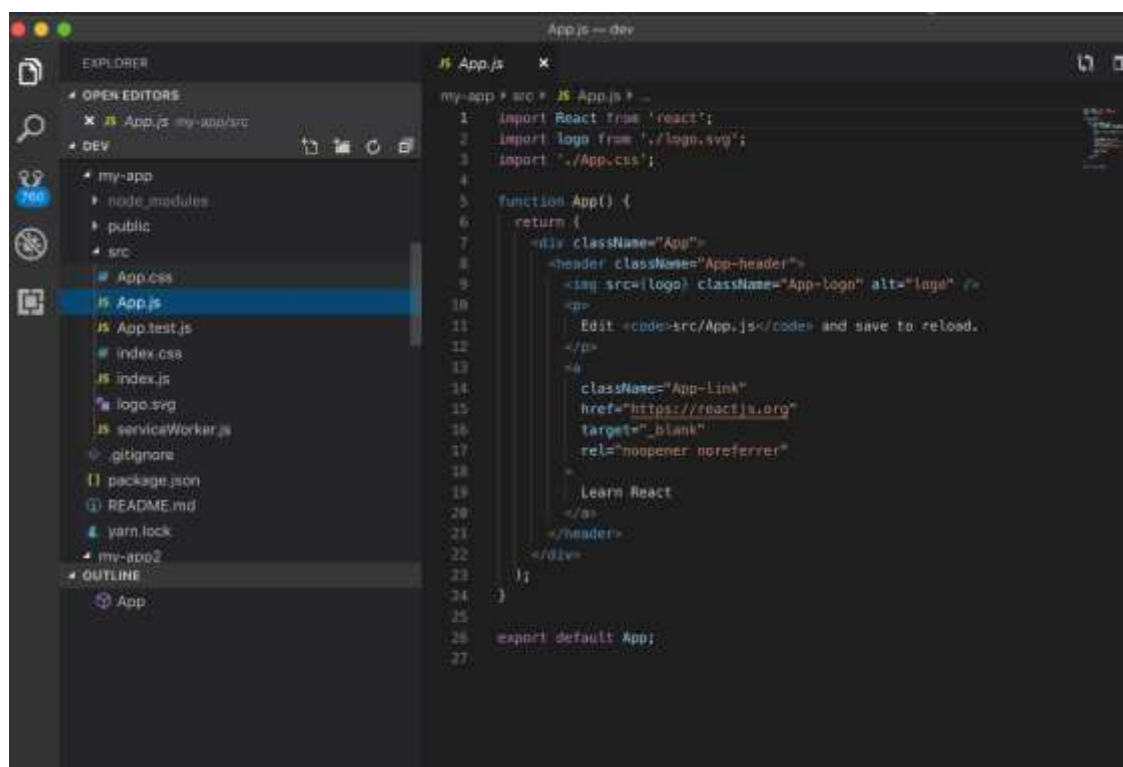


Рисунок 3.8 - Базова структура React проекту

Ці залежності керуються версіями та встановлюються на основі семантичного керування версіями (semver). Semver визначає схему управління версіями, яка відображає типи змін у кожній новій версії, незалежно від того, порушує зміна API, додає нову функцію чи виправляє помилку. Однак semver покладається на те, що розробники пакунків не допускать помилок — несправні зміни або нові помилки можуть потрапити у встановлені залежності, якщо залежності не заблоковано.

В екосистемі Node залежності розміщуються в каталозі `node_modules` у вашому проекті. Однак ця файлова структура може відрізнятися від фактичного дерева залежностей, оскільки повторювані залежності об'єднуються разом. Клієнт `npm` недетерміновано встановлює залежності в каталог `node_modules`. Це означає, що на основі порядку встановлення залежностей структура каталогу `node_modules` може відрізнятися від однієї особи до іншої. Ці відмінності можуть спричинити помилки «працює на моїй машині», пошук яких потребує часу.

Yarn вирішує ці проблеми щодо версії та недетермінізму за допомогою файлів блокування та алгоритму встановлення, який є детермінованим і надійним. Ці файли блокування блокують встановлені залежності до певної версії та гарантують, що кожне встановлення призводить до точно такої самої файлової структури в `node_modules` на всіх машинах. Написаний файл блокування використовує стислий формат із упорядкованими ключами, щоб забезпечити мінімальні зміни та простий перегляд.

Redux — це контейнер передбачуваного стану для програм JavaScript. Це допомагає писати програми, які ведуть себе узгоджено, запускаються в різних середовищах (клієнт, сервер) і які легко тестувати. Крім того, він забезпечує чудовий користувацький досвід для розробників, наприклад редагування коду в реальному часі в поєднанні з дебагером у часі.

Redux — це спосіб керування «станом», або ми можемо сказати, що це кеш або сховище, до якого можуть отримати доступ усі компоненти структурованим способом. До нього потрібно отримати доступ через «Reducer» і «Actions» (рис. 3.9).

Redux Toolkit — це офіційний рекомендований підхід для написання логіки Redux. Він охоплює ядро Redux і містить пакети та функції, які, на думку розробників, необхідні для створення програми Redux. Redux Toolkit використовує запропоновані нами передові практики, спрощує більшість завдань Redux, запобігає типовим помилкам і полегшує написання програм Redux.

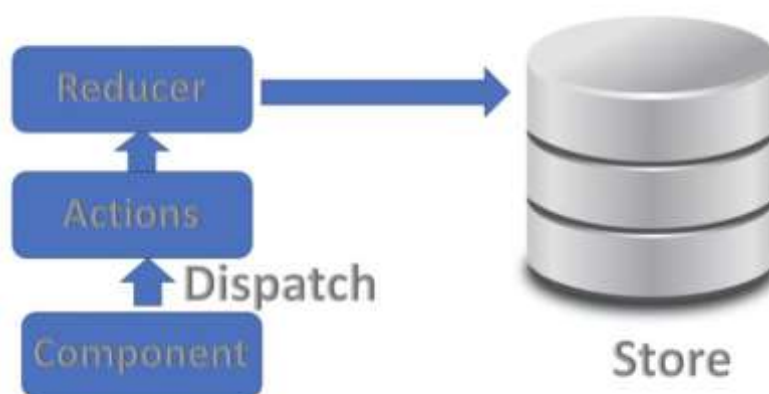


Рисунок 3.9 - Базова структура React проекту

RTK містить утиліти, які допомагають спростити багато поширених випадків використання, включаючи налаштування магазину, створення редукторів і написання незмінної логіки оновлення, і навіть створення цілих «зрізів» стану одночасно.

Незалежно від того, чи ви новий користувач Redux, який створює свій перший проект, чи досвідчений користувач, який хоче спростити існуючу програму, Redux Toolkit може допомогти вам покращити ваш код Redux.

Redux Toolkit доступний у вигляді пакета на NPM для використання з групувальником модулів або в програмі Node: `npm install @reduxjs/toolkit`

Основна бібліотека Redux доступна у вигляді пакета на NPM для використання з групувальником модулів або в програмі Node:

NPM `npm install redux` # Yarn `yarn add redux`

Для початку імплементації `redux` в застосунок, необхідно створити функцію `reducer` і передати її в метод `createStore()`. `createStore()` — створює сховище Redux, яке містить повне дерево стану програми.

У програмі має бути лише одне сховище даних.

Аргументи:

- редуктор (функція);
- `[preloadedState]` (будь-який): початковий стан;
- `[підсилювач]` (функція): підсилювач сховища даних.

Редуктор (функція): функція скорочення, яка повертає наступне дерево станів, враховуючи поточне дерево станів і дію для обробки.

`[preloadedState]` (будь-який): початковий стан. Можна додатково вказати його, щоб імплементувати стан із сервера в універсальних програмах або відновити попередньо серіалізований сеанс користувача. Якщо створено редуктор за допомогою `combineReducers`, це має бути звичайний об'єкт такої самої форми, що й передані йому ключі. В іншому випадку можна передавати все, що може зрозуміти редуктор.

[підсилювач] (функція): підсилювач сховища даних. Можна додатково вказати його, щоб розширити сховище сторонніми можливостями, такими як проміжне програмне забезпечення, стійкість тощо. Єдиним розширювачем сховища, який постачається з Redux, є `applyMiddleware()`.

Результатом виконання буде об'єкт, який містить повний стан програми. Єдиний спосіб змінити цей стан — це диспетчерські дії. Також можна підписатися на зміни в сховищі, щоб в залежності від них, оновити інтерфейс користувача.

В файлі `store.js` було створено сховище і експортоване для подальшого використання (рис. 3.10). Вбудований метод «`combineReducers`» дозволяє об'єднати всі редуктори, які необхідні для управління сховищем даних. Метод `configureStore` з переданими до нього редукторами, дозволяє ініціалізувати сховище, варто відмітити що в додатку може бути лише одне глобальне сховище даних.

```
import { combineReducers, configureStore } from '@reduxjs/toolkit';
import allUsersReducer from './reducers/allUsersReducer';
import calendarReducer from './reducers/calendarReducer';
import historyReducer from './reducers/historyReducer';
import receptionReducer from './reducers/receptionReducer';
import scheduleReducer from './reducers/scheduleReducer';
import userReducer from './reducers/userReducer';

const rootReducer = combineReducers({
  history: historyReducer,
  calendar: calendarReducer,
  reception: receptionReducer,
  schedule: scheduleReducer,
  user: userReducer,
  users: allUsersReducer
});

const store = configureStore({ reducer: rootReducer });

export default store;
```

Рисунок 3.10 - Ініціалізація сховища даних

В файлі `userReducer.js`, було створено редуктор, для керування сховищем за допомогою заздалегідь прописаних випадків (рис. 3.11).

```
import {LOGIN, SIGN_UP, LOG_OUT} from '../constants/index'

const initialState = {
  id: 0,
  email: '',
  name: '',
  surname: '',
  position: '',
  phone: '',
  isTemporary: true,
  isLoggedIn: false
}

export default function userReducer(state = initialState, action){
  switch (action.type){
    case SIGN_UP: {
      return {
        ...state,
        id: action.payload.id,
        email: action.payload.email,
        name: action.payload.name,
        surname: action.payload.surname,
        position: action.payload.position,
        phone: action.payload.phone,
        isTemporary: action.payload.isTemporary,
        role: action.payload.role,
        isLoggedIn: true,
      }
    }
  }
}
```

Рисунок 3.11 - Структура редуктора

Для редуктора першим параметром необхідно вказати початковий стан, який є схемою з початковими даними яка відображає вміст значень в сховищі, другим параметром редуктор приймає параметр «action» який є об'єктом що зазвичай містить в собі два поля: «type», «payload». Поле «type» вказує на тип події, яка розглядається в умові, і визначає який саме випадок буде виконаним, а поле «payload» містить в собі дані, якими зазвичай треба замінити поточні дані в сховищі.

Для виклику редуктора необхідно створити функцію - «actionFunction» яка буде викликатись через метод dispatch, це дозволить звертатись до редуктора і передавати в нього тип події та данні для внесення змін в сховище. Функція «action» - це чиста функція, яка зазвичай повертає об'єкт з двома полями: «action» та «payload» (рис. 3.12).

```
export function registrationAction(payload){
  return {
    type: SIGN_UP,
    payload
  }
}
```

Рисунок 3.12 - Типовий вигляд функції action

Метод dispatch отримується за допомогою хуку useDispatch який імпортується, з бібліотеки «react-redux», він дозволяє викликати редуктор і передати в нього всі необхідні дані (рис. 3.13).

```
import { useDispatch } from 'react-redux'

function Registration() {
  const dispatch = useDispatch()
  const [captcha, setCaptcha] = useState(null)

  const [captchaError, setCaptchaError] = useState(false)
  function registrationHandler(data){
    if(captcha){
      dispatch(thunkRegistration(data))
      setCaptchaError(false)
    }else{
      setCaptchaError(true)
    }
  }
}
```

Рисунок 3.13 - Виклик функції dispatch

Зареєстрований користувач - це користувач веб-сайту, програми чи інших систем, який раніше зареєструвався. Зареєстровані користувачі зазвичай надають системі певні облікові дані (наприклад, ім'я користувача або адресу електронної пошти та пароль), щоб підтвердити свою особу. Це не тільки дозволяє відстежувати, хто використовує веб-застосунок, але й дозволяє захистити певні сторінки від загального доступу та дозволяє користувачам вільно комунікувати, не турбуючись про анонімний спам.

В даному додатку було створено декілька типів реєстрації користувачів:

- адміністратор;
- користувач.

Таке розподілення користувачів, необхідне для того щоб розділити можливості різних груп користувачів і додати можливості адміністрування додатком для замовника. Для адміністратора в додатку будуть доступні методи які повинні передбачувати всі зміни що будуть стосуватись:

- управління профілями;
- створення профілів;
- зміни в графіках;
- доступу до загальної історії всіх викликів.

В додатку для адміністратора буде доступною спеціальна сторінка з посиланням на неї (рис. 3.14), в якій адміністратор зможе керувати користувачами.

Сторінка «управління акаунтами», необхідна для адміністрування всіх кейсів пов'язаних зі змінами в персоналі, чи їх персональних даних (рис. 3.15). Вона буде доступною тільки в тому випадку, якщо користувач зареєстрований як адміністратор, і всі наявні функції управління будуть виконуватись виключно в цьому ж випадку.

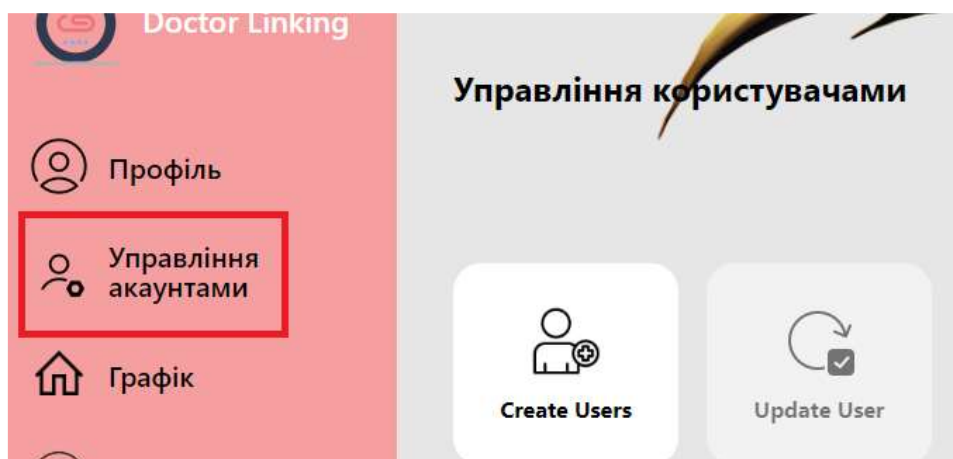


Рисунок 3.14 - Сторінка управління акаунтами

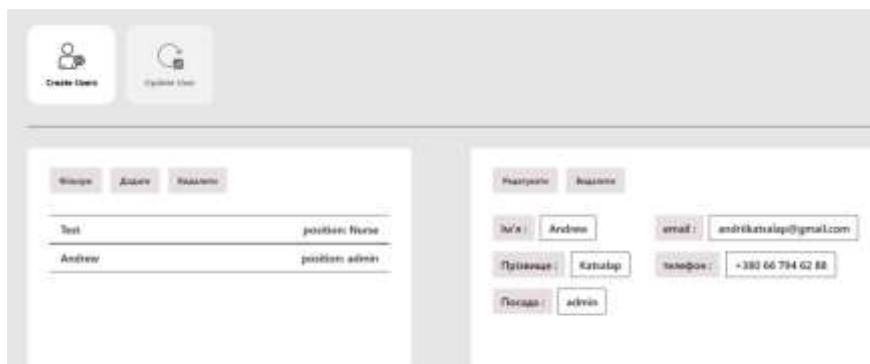


Рисунок 3.15 - Сторінка «управління акаунтами»

Для адміністратора в сфері впливу на користувачів представлені такі можливості:

- видалення/створення профілів користувачів;
- редагування профілів користувачів;
- доступ до історії викликів/запитів користувача.

Видалення користувачів важлива деталь в повноцінному функціонуванні додатку яка є необхідної для реалізації.

Для видалення користувача було створено маршрут на сервері, в який необхідно передати id користувача, якого необхідно видалити. Після відповіді з серверу, про успішне видалення, буде оновлено сховище даних на клієнтській частині і в базі даних.

Для створення акаунту необхідно натиснути кнопку «Додати», після чого з'явиться форма, в якій буде необхідно заповнити всі вказані поля (рис. 3.16).

Рисунок 3.16 - Форма для створення користувача

Після заповнення інформації на клієнтській частині, на сервер буде відправлено запит з заповненими даними, після чого на сервері буде збережено новоствореного користувача (рис. 3.17).

id	email	name	surname	position	phone	role	password	created_at	updated_at	refresh_token	doctor_signup_status
1	doctor@doctor.com	Doctor	Doctor	Doctor	+380 99 794 62 88	admin	1234567890	2022-12-09 16:40:02.594704	2022-12-09 17:14:02.819988		1
2	patient@patient.com	Test	Patient	Patient	+380 (099) 794 62 88	user	1234567890	2022-12-09 16:40:45.777084	2022-12-09 16:40:02.272005		2

Рисунок 3.17 - Створений користувач в базі даних

Для відправки запиту на сервер, було викликано функцію «thunkDoctorSignUp» з її допомогою буде відправлено запит на сервер, та відображено відповідні зміни на клієнтській частині.

```
export function doctorSignUpAction(payload){
  return {
    type: DOCTOR_SIGN_UP,
    payload
  }
}

function fetchDoctorSignUp(payload){
  return $api.post('/doctor-signup', payload )
}

export function thunkDoctorSignUp(payload){
  return async (dispatch) => {
    try{
      let response = await fetchDoctorSignUp(payload)
      console.log(response)
    }catch(e){
      console.log(e)
    }
  }
}
```

Функція «thinkDoctorSignUp» використовується в даному випадку для виконання асинхронного запиту на сервер, та для того щоб отримати доступ до функції «dispatch» через яку будуть внесені зміни для сховища даних на клієнтській частині додатку.

Веб-повідомлення - це невеликі сповіщення про повідомлення, які відображаються на робочому столі, планшеті чи мобільному пристрої відвідувача, коли він відкрив веб-переглядач, або в фоновому режимі.

Якщо донедавна push-сповіщення були привілеєм для нативних програм, тепер це змінилося. Push-повідомлення тепер можна надсилати безпосередньо до PWA: як і у випадку з нативною програмою, браузер не обов'язково повинен бути відкритий, навіть у фоновому режимі.

Web Push Notification — це протокол, у якому беруть участь 4 учасники (рис. 3.18).

- користувач - це той, хто хоче отримувати сповіщення.
- програма або PWA — запускається в браузері, які використовують Push API, містять службу push, яка відповідає за маршрутизацію push-повідомлень від сервера до користувача;
- service worker — діє як проксі-сервер між програмою, браузером і мережею;
- push-сервер або сервер - він надсилає push-повідомлення service worker через службу push.

Потік починається, коли програма запитує у користувача дозвіл на отримання push-повідомлень. Після того, як користувач надав дозвіл, програма реєструє Service Worker. Коли PWA отримує реєстрацію від Service Worker, PWA може використовувати її для створення push-підписки. Служба push також потрібна для створення підписки. Push-підписка також містить кінцеву точку служби push для надсилання push-повідомлень. У цей момент програма надсилає створену push-підписку на сервер. Серверу потрібна кінцева точка push-підписки, щоб надсилати push-повідомлення. Коли сервер отримує push-підписку, вона зберігається в базі даних під назвою subscriptionsDB. Сервер

надсилає push-повідомлення до кінцевої точки з push-підписки. Потім push-сповіщення направляється через службу push до Service Worker, який прослуховує push-подію.

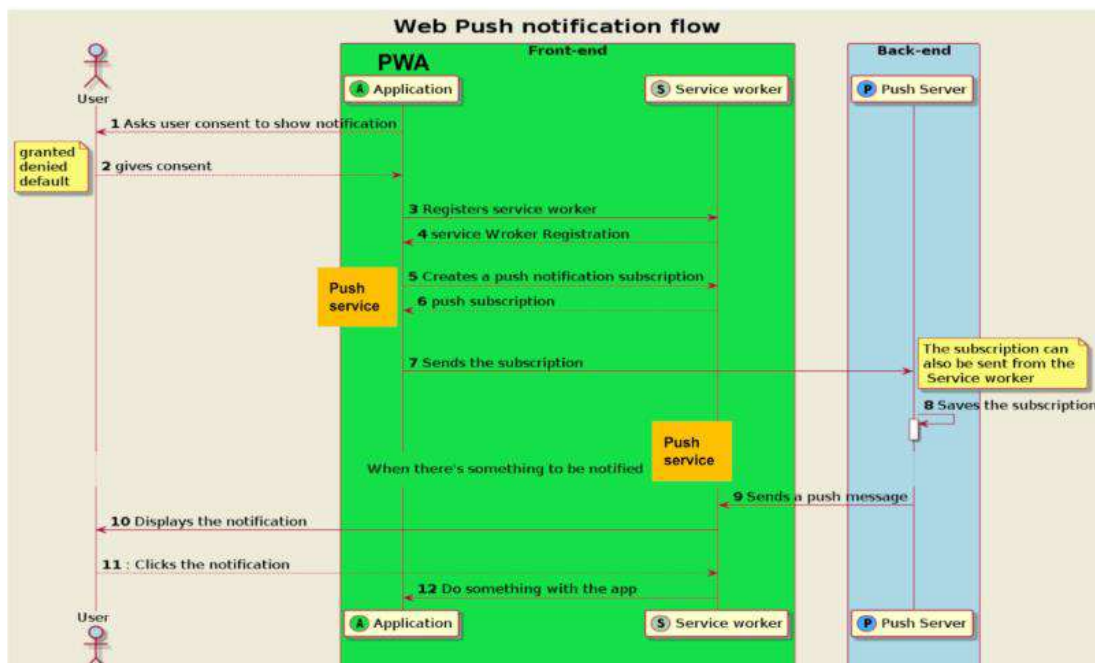


Рисунок 3.18 - Обмін даних для відправки push-повідомлень

Потім Service Worker пересилає push-сповіщення користувачеві. Коли користувач натискає push-сповіщення, Service Worker отримує його через подію натискання сповіщення. Service Worker тепер може робити майже все, що забажає, за допомогою push-повідомлень.

В ролі push-серверу буде виступати Firebase. Firebase — це компанія, яку Google придбала в 2014 році. З тих пір Google зробив кілька вдосконалень платформи до такої міри, що тепер вона пропонує Firebase як своє єдине рішення для мобільних додатків, що працює як серверна частина.

Серед кількох рішень Google, таких як централізована автентифікація, бази даних у реальному часі та хмарні функції (еквівалент Google AWS Lambda), які є частиною парасольки Firebase, є одне, яке виділяється як найкраще рішення для керування сповіщеннями програм: Firebase Cloud Обмін повідомленнями (FCM),

раніше Google Cloud Messaging (GCM).

Для того щоб з'єднати застосунок з firebase необхідно увійти або зареєструйтеся на консолі Firebase і натиснути «Додати проект». Щоб додати проект, необхідно виконати такі дії:

- дати проекту відповідну назву;
- увімкнути або вимкнути аналітику;
- дочекатись завершення налаштування.

Тепер потрібно створити зв'язок між проектом Firebase і додатком React (рис.3.19).

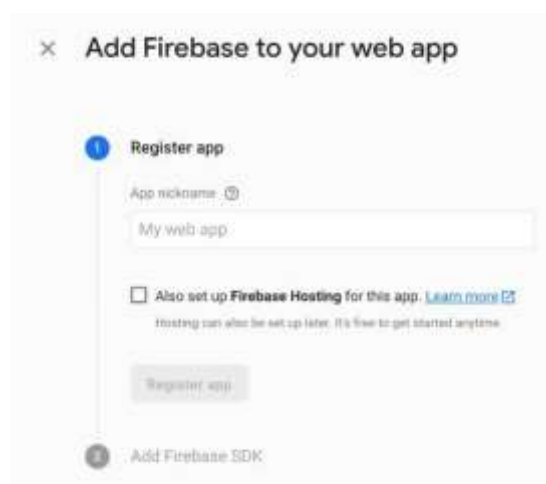


Рисунок 3.19 - Реєстрація додатку в firebase

Після цього, на екрані згенерується конфігурація, яку буде необхідно інтегрувати в застосунок React. Так має виглядати кінцева згенерована конфігурація (рис.3.20).

Об'єкт `firebaseConfig` буде інтегровано в застосунок React, який зв'яже його з цим проектом Firebase. Далі в проекті необхідно створити файл `firebase.js` та передати в нього дані з конфігурації створеної в додатку firebase. Для того щоб інтегрувати можливість хмарних повідомлень потрібно створити ключ веб-сертифіката `push`, для цього потрібно перейти на вкладку хмарного обміну повідомленнями для створеного проекту та перейти до розділу веб-конфігурації.

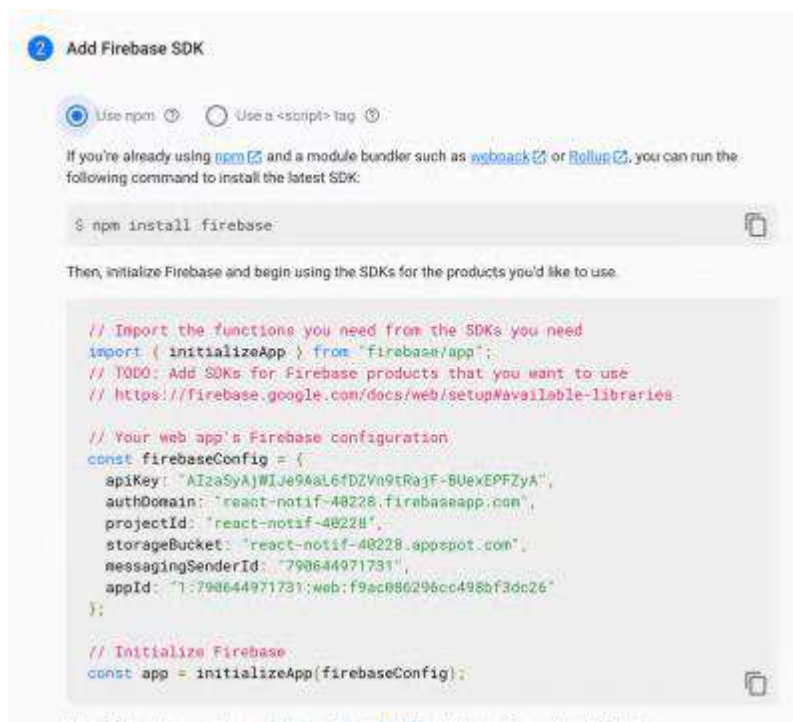


Рисунок 3.20 - Конфігурація створеного додатку в firebase

У розділі Сертифікати Web Push потрібно натиснути «Створити пару ключів» (рис.3.21).

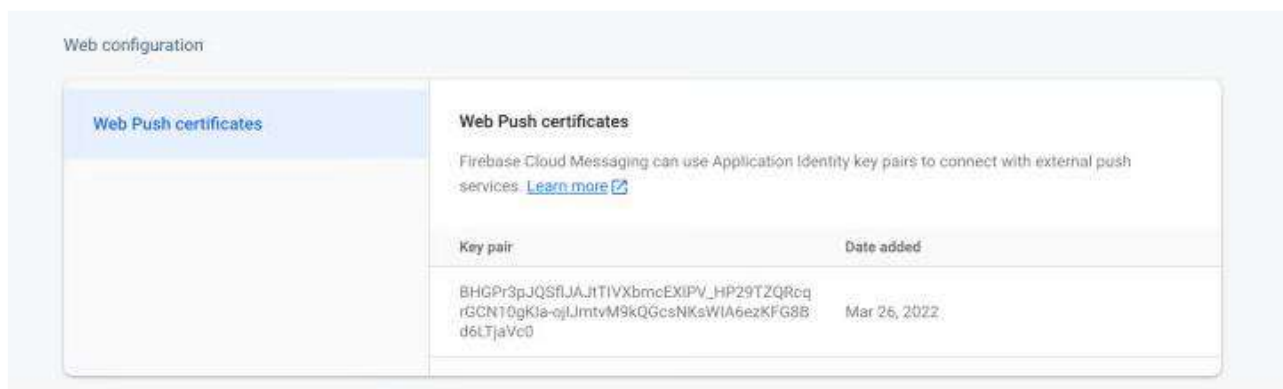


Рисунок 3.21 - Створений ключ, для обміну повідомленнями

Щоб надіслати push-сповіщення в браузер, спочатку потрібно отримати дозвіл від користувача. Коли даний функціонал буде зроблено, відкриється вікно «Увімкнути сповіщення?» спливаюче вікно, яке можна було бачити на інших веб-сайтах. Спосіб ініціювати цей запит — викликати метод `getToken`, наданий

Firebase.

```
export const getToken = (setTokenFound) => {
  return getToken(messaging, {vapidKey:
    'GENERATED_MESSAGING_KEY'}).then((currentToken) => {
    if (currentToken) {
      setTokenFound(true);
    } else {
      setTokenFound(false);
    }
  }).catch((err) => {
    console.log('An error occurred while retrieving token. ', err);
  });
}
```

Функція setTokenFound передається у функцію getToken. Це робиться для того, щоб була змога відстежувати, чи був отриманий маркер клієнта (тобто чи надано дозвіл на сповіщення). Тепер після збереження цієї функції та виконання коду, ми отримуємо спливаюче вікно із запитом сповіщень (рис. 3.22).

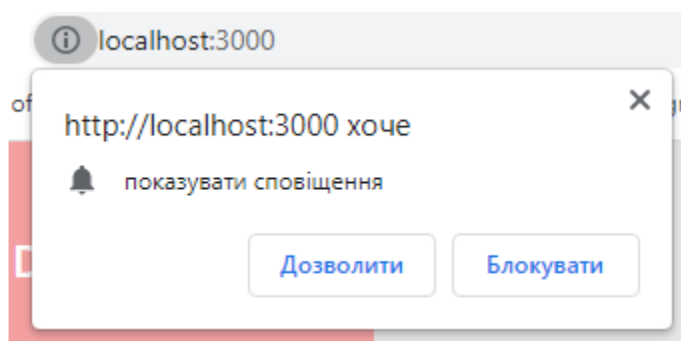


Рисунок 3.22 - Запит на відправку сповіщень

Метод getToken, якщо він постачається з об'єктом обміну повідомленнями, відображає інтерфейс користувача для сповіщень, очікує на введення користувача та, дозволивши, призначає унікальний маркер клієнту, який можна

побачити на консолі. Тепер, коли надані дозволи на сповіщення та посилення на маркер клієнта на стороні браузера, наступним кроком є додавання слухача до вхідного push-сповіщення, яке спрямовується на клієнта реалізується це, додаванням файлу service worker firebase-messaging-sw.js у загальну папку в додатку React, а потім додаємо такий код:

```
importScripts('https://www.gstatic.com/firebasejs/9.0.0/firebase-app-compat.js');
importScripts('https://www.gstatic.com/firebasejs/9.0.0/firebase-messaging-
compat.js');
var firebaseConfig = {
  apiKey: "YOUR_API_KEY",
  authDomain: "YOUR_AUTH_DOMAIN",
  projectId: "YOUR_PROJECT_ID",
  storageBucket: "YOUR_STORAGE_BUCKET",
  messagingSenderId: "YOUR_MESSAGING_SENDER_ID",
  appId: "YOUR_APP_ID"
};
firebase.initializeApp(firebaseConfig);
const messaging = firebase.messaging();
messaging.onBackgroundMessage(function(payload) {
  const notificationTitle = payload.notification.title;
  const notificationOptions = {
    body: payload.notification.body,
  };
  self.registration.showNotification(notificationTitle,
    notificationOptions);
});
```

В даному випадку використовуються сумісні версії сценаріїв із importScripts. Це тому, що отримуються сценарії Service Worker, сумісні з Firebase v8. Якщо отримати сценарії v9, знадобиться додатковий етап компіляції,

оскільки вони не працюватимуть з модулями ES. Це та сама причина, чому зберігається код всередині Service Worker у стилі Firebase v8 і не змінюю його на v9.

Тестування відправки нагадувань, реалізується за допомогою програми Postman (рис. 3.23). Postman - програма, яка використовується для тестування API. Це HTTP-клієнт, який тестує HTTP-запити, використовуючи графічний інтерфейс користувача, за допомогою якого ми отримуємо різні типи відповідей, які згодом потрібно перевірити.

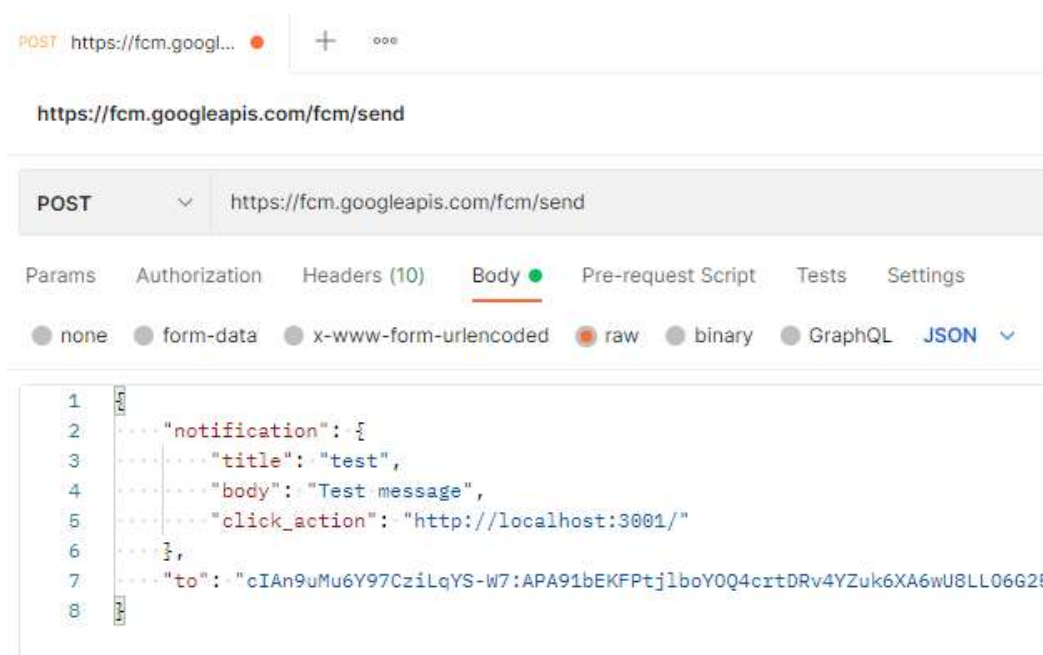


Рисунок 3.23 - Тестування відправки нагадування

Для відправки нагадування запит необхідно надсилати за адресою: `https://fcm.googleapis.com/fcm/send`. Використовуючи Firebase Admin SDK або протоколи сервера додатків FCM, ви можете створювати запити на повідомлення.

Далі необхідно сформулювати тіло запиту в яке треба передати дві властивості об'єкт нагадування з полями:

- title: заголовок нагадування;
- body: тіло нагадування;

- `click_action`: дія, пов'язана з користувачем, клацнувши сповіщення.

Також необхідно вказати ключ який отримувач нагадування отримує коли дозволяє сповіщення від браузера. В результаті виконання запиту в браузер буде надіслано сповіщення (рис. 3.24).

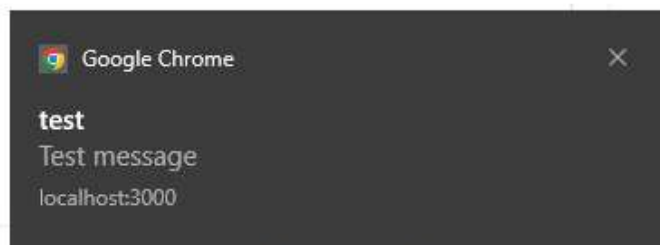


Рисунок 3.24 - Результат тестування надсилання сповіщень

В результаті проведення тестування, було отримано сповіщення.

Для ініціалізації серверу було використано модуль «typescript-express-starter» (рис. 3.25).



Рисунок 3.25 - Модуль «typescript-express-starter»

Бібліотека Express написана на JavaScript, що робить її вразливою до визначень типів. Даний пакет налаштовано на використання TypeScript замість JavaScript. В проекті створеному з використанням цього модуля, відразу будуть наявними початкові налаштування, та багато інших корисних методів:

- ініціалізація серверу;
- підключення до бази даних;
- налаштовані маршрути;

- контролери;
- сервіси;
- проміжне програмне забезпечення;
- тести;
- об'єкти передачі даних;
- сутності.

Наявність такого функціоналу, значно спрощує, і пришвидшує процес розробки серверу, крім цього стиль написання коду який представляють розробники використовує найкращі практики, і всі сучасні технології проектування. Для того щоб використовувати цей пакет, спочатку його треба встановити глобально, використовуючи пакетний менеджер (рис. 3.26).

```
$ npm install -g typescript-express-starter
```

Рисунок 3.26 - Встановлення модулю

Після цього, використовуючи команду — `npx typescript-express-starter "project name"`, можна завантажити шаблон для додатку. Npx розшифровується як Node Package Execute і постачається разом із npm, коли ви встановлюєте npm вище 5.2.0 версії, npx встановлюється автоматично. Це програма запуску пакетів npm, яка може виконувати будь-який пакет із реєстру npm, навіть не інсталиючи цей пакет.

Якщо не ввести назву проекту, за замовчуванням буде `typescript-express-starter`.

Під час встановлення бібліотека запитає який варіант шаблону ви хочете обрати:

- default: звичайний застосунок з Express;
- routing Controllers: створює структуровані, декларативні та красиво організовані контролери на основі класів із інтенсивним використанням декораторів;
- sequelize: простий у використанні ORM на кількох діалектах SQL для Node.js;

- Mongoose: об'єктне моделювання MongoDB (ODM), призначене для роботи в асинхронному середовищі;
- TypeORM: ORM, який може працювати в Node.js та інших середовищах;
- Prisma: сучасний доступ до бази даних для TypeScript і Node.js;
- Кnex: конструктор запитів SQL для Postgres, MySQL, MariaDB, SQLite3 і Oracle;
- Mikro ORM: TypeScript ORM для Node.js на основі шаблонів Data Mapper, Unit of Work та Identity Map. Підтримує бази даних MongoDB, MySQL, MariaDB, PostgreSQL і SQLite;

Після встановлення шаблону для серверу будуть доступні наступні команди:

- запуск серверу в режимі продакшн: `npm run start` or `Start typescript-express-starter` в VS Code;
- запуск серверу в режимі розробника: `npm run dev` or `Dev typescript-express-starter` в VS Code;
- запуск всіх юніт тестів: `npm test` or `Test typescript-express-starter` в VS Code
- перевірка всіх помилок літера: `npm run lint` or `Lint typescript-express-starter` в VS Code;
- виправлення помилок літера: `npm run lint:fix` or `Lint:Fix typescript-express-starter` в VS Code.

Розглянемо структуру створеного проекту, наведену на рис. 3.27. Основними елементами API є `services`, `routes`, `controllers`, `database`, `tests`, `middlewares`. Розглянемо їх:

- `services` містять основну логіку виконання маршрутів;
- `routes` зберігає всі маршрути які присутні в API;
- `controllers` містить методи для попередньої обробки запиту, прокидання помилок та відправки відповіді клієнту;
- `database` містить файл підключення до бази даних;
- `tests` відповідає за всі юніт тести для API;
- `middlewares` зберігає все проміжне програмне забезпечення.

Додаткові модулі встановлені для розширення функціоналу містяться в папці `node_modules`, а переглянути їх можна в файлі `package.json` (рис. 3.287).

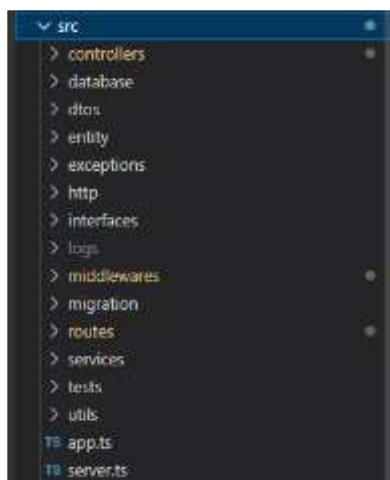


Рисунок 3.27 - Структура проекту

```

"dependencies": {
  "axios": "^1.1.3",
  "bcrypt": "^5.1.0",
  "class-transformer": "^0.5.1",
  "class-validator": "^0.13.2",
  "compression": "^1.7.4",
  "cookie-parser": "^1.4.6",
  "cors": "^2.8.5",
  "cross-env": "^7.0.3",
  "dotenv": "^16.0.3",
  "envalid": "^7.3.1",
  "express": "^4.18.2",
  "generate-password": "^1.7.0",
  "helmet": "^5.0.0",
  "hpp": "^0.2.3",
  "jest": "^29.2.1",
  "jsonwebtoken": "^8.5.1",
  "mongodb": "^3.7.3",
  "morgan": "^1.10.0",
  "nodemailer": "^6.8.0",
  "pg": "^8.8.0",
  "reflect-metadata": "^0.1.13",
  "swagger-jsdoc": "^6.0.0",
  "swagger-ui-express": "^4.5.0",
  "ts-jest": "^29.0.3",
  "ts-node": "^10.9.1",
  "typeorm": "^0.3.10",
  "typescript": "^4.8.4",
  "winston": "^3.8.2",
  "winston-daily-rotate-file": "^4.7.1"
}

```

Рисунок 3.28 - Додаткові модулі проекту

Розглянемо лише основні модулі.

Axios - це HTTP-клієнт на основі промісів для `node.js` і браузера. Він ізоморфний (він може працювати в браузері та `nodejs` з однаковою кодовою базою). На стороні сервера він використовує власний `http`-модуль `node.js`, тоді як на стороні клієнта (браузер) він використовує `XMLHttpRequests`.

Bcrypt - це функція хешування паролів, розроблена Нільсом Провосом і Девідом Мазьєром, заснована на шифрі Blowfish і представлена на USENIX у 1999 році. Окрім використання солі для захисту від атак веселкової таблиці, bcrypt є адаптивною функцією: з часом кількість ітерацій можна збільшити, щоб зробити її повільнішою, тож вона залишається стійкою до атак грубої сили навіть із збільшенням обчислювальної потужності.

CORS - це пакет node.js для надання проміжного програмного забезпечення Connect/Express, яке можна використовувати для ввімкнення CORS із різними параметрами.

Jest - це платформа для тестування JavaScript, зосереджена на простоті. Він працює з проектами, які використовують: Babel, TypeScript, Node, React, Angular, Vue тощо.

Pg - Неблокуючий клієнт PostgreSQL для Node.js.

REST — це абревіатура від REpresentational State Transfer і архітектурний стиль для розподілених гіпермедійних систем. Рой Філдінг вперше представив його у 2000 році у своїй знаменитій дисертації. Як і інші архітектурні стилі, REST має свої керівні принципи та обмеження. Ці принципи мають бути виконані, якщо інтерфейс служби потрібно називати RESTful.

Шаблон проектування «клієнт-сервер» забезпечує поділ завдань, що допомагає компонентам клієнта та сервера розвиватися незалежно. Відокремлюючи інтерфейс користувача (клієнт) від проблем зберігання даних (сервер), ми покращуємо переносимість інтерфейсу користувача між кількома платформами та покращуємо масштабованість за рахунок спрощення компонентів сервера. Поки клієнт і сервер розвиваються, ми маємо переконатися, що інтерфейс/контракт між клієнтом і сервером не порушується.

Серверні маршрути в додатку буде розглянуто на прикладі маршруту «RequestRoute» його код:

```
class RequestRoute implements Route {
  public router = Router();
```

```

public requestController = new RequestController();
constructor() {
  this.initializeRoutes();
}
private initializeRoutes() {
  this.router.post('/set-request', authMiddleware, this.requestController.setRequest);
}
}
export default RequestRoute;

```

В додатку всі маршрути розділені між собою за специфікою їх використання, в класі RequestRoute будуть наявні всі маршрути що будуть стосуватись запитів які будуть надсилатись лікарям.

В даному методі створюється публічна змінна router в яку передається результат виконання функції Router імпортованої з бібліотеки express. Об'єкт Router— це ізольований екземпляр проміжного програмного забезпечення та маршрутів. По суті це «міні-програма», здатна лише виконувати функції проміжного програмного забезпечення та маршрутизації. Кожна програма Express має вбудований маршрутизатор програми.

Маршрутизатор поводитьсь як проміжне програмне забезпечення, тому його можна використовувати як аргумент для app.use() або як аргумент для методу use() іншого маршрутизатора.

В даному класі також реалізований приватний метод «initializeRoutes» який ініціалізує маршрути. У змінній router викликається метод по типу запиту, в даному випадку це post-запит, і в нього передається 3 параметри:

- адреса маршруту;
- проміжне програмне забезпечення;
- функція контроллер;

Функції проміжного програмного забезпечення є невід'ємною частиною програми, створених за допомогою інфраструктури Express. Вони отримують

доступ до об'єктів HTTP-запиту та відповіді та можуть завершити HTTP-запит або переслати його для подальшої обробки іншій функції проміжного програмного забезпечення.

Функції проміжного програмного забезпечення приєднані до одного або кількох обробників маршрутів у програмі Express і виконуються послідовно з моменту отримання HTTP-запиту програмою до надсилання HTTP-відповіді абоненту. Ця можливість виконання функцій проміжного програмного забезпечення Express у ланцюжку дозволяє нам створювати менші потенційно багаторазові компоненти на основі принципу єдиної відповідальності (SRP).

«authMiddleware» це функція проміжного програмного забезпечення яка перевіряє чи автентифікований користувач надсилає запит до серверу.

```
const authMiddleware = async (req: RequestWithUser, res: Response, next:
NextFunction) => {
  try {
    const authService = new AuthService();
    const authorizationHeader = req.headers.authorization;
    if(!authorizationHeader){
      next(new HttpException(404, 'Authentication token missing'));
    }
    const accessToken = authorizationHeader.split(' ')
    if(!accessToken){
      next(new HttpException(404, 'Authentication token missing'));
    }
    const userData = await authService.validateAccessToken(accessToken)
    if(!userData){
      next(new HttpException(404, 'Authentication token missing'));
    }
    req.user = userData;
    next()
```

```

    } catch (error) {
    next(new HttpException(401, 'Wrong authentication token'));
    }
  };
  export default authMiddleware;

```

В даному методі перевіряється чи наявний заголовок автентифікації під час запиту, якщо він відсутній то запит завершується з помилкою 404, якщо ж ні то токен валідується і в результаті повертається інформація про користувача, в подальшому ці дані присвоюються об'єкту `req.user` і можуть бути доступними в контролері.

Публічна змінна `requestContorller` буде зберігати в собі екземпляр класу «RequestController», це дозволить викликати необхідні методи, для подальшої обробки запитів від клієнта.

```

class RequestController {
  public requestService = new RequestService();
  public setRequest = async (req: Request, res: Response, next: NextFunction):
    Promise<void> => {
    try {
      const requestData = req.body;
      const result = await this.requestService.setRequest(requestData);
      res.status(201).json({ message: 'request sended' });
    } catch (error) {
      next(error);
    }
  }
}

```

В класі `RequestController` реалізовано метод `setRequest` який буде виконуватись під час запиту до серверу по маршруту «`http://localhost:3001/set-request`». В цьому методі реалізована обробка помилок, відповідь клієнту, та

виклик сервісу, в якому реалізується логіка виконання маршруту.

QueryBuilder - одна з найпотужніших функцій TypeORM — вона дозволяє створювати SQL-запити з використанням елегантного та зручного синтаксису, виконувати їх і отримувати автоматично трансформовані сутності.

Простий приклад QueryBuilder:

```
const firstUser = await dataSource
  .getRepository(User)
  .createQueryBuilder("user")
  .where("user.id = :id", { id: 1 })
  .getOne()
```

Він створює такий SQL-запит:

```
SELECT
user.id as userId,
user.firstName as userFirstName,
user.lastName as userLastName
FROM users user
WHERE user.id = 1
```

Є два типи результатів, які можна отримати за допомогою конструктора вибіркового запиту: сутності або необроблені результати. У більшості випадків вам потрібно вибрати реальні сутності з вашої бази даних, наприклад, користувачів. Для цього ви використовуєте `getOne` і `getMany`. Але іноді потрібно вибрати певні дані, скажімо, суму всіх фотографій користувача. Ці дані не є сутністю, вони називаються необробленими даними. Щоб отримати необроблені дані, ви використовуєте `getRawOne` і `getRawMany`.

Можна було б написати: `where("user.name = " + name + ")`, однак це небезпечно, оскільки відкриває код для SQL-ін'єкцій. Безпечним способом є використання цього спеціального синтаксису: `where("user .name = :name", { name: "Timber" })`, де `:name` — це ім'я параметра, а значення вказано в об'єкті: `{ name: "Timber" }`.

3.3 Тестування інформаційної системи

Веб-тестування - це тестування програмного забезпечення, яке зосереджується на веб-додатках. Повне тестування веб-системи перед запуском може допомогти вирішити проблеми до того, як система стане доступною для громадськості. Проблеми можуть включати безпеку веб-додатку, базові функціональні можливості сайту, його доступність для користувачів з обмеженими можливостями та повними можливостями, його здатність адаптуватися до безлічі робочих столів, пристроїв і операційних систем, а також готовність до очікуваного трафіку та кількість користувачів і здатність вижити після значного сплеску трафіку користувачів, обидва з яких пов'язані з тестуванням навантаження.

Тестування програмного забезпечення - це перевірка артефактів і поведінки тестованого програмного забезпечення шляхом валідації та перевірки. Тестування програмного забезпечення також може надати об'єктивний, незалежний погляд на програмне забезпечення, щоб дозволити компанії оцінити та зрозуміти ризики впровадження програмного забезпечення.

Методи тестування включають, але не обов'язково обмежуються:

- аналіз вимог щодо повноти та правильності продукту в різних контекстах, таких як галузеві перспективи, бізнес-перспективи, доцільність і життєздатність впровадження, зручність використання, продуктивність, безпека, інфраструктурні міркування тощо.
- перегляд архітектури продукту та загального дизайну продукту.
- робота з розробниками продуктів щодо вдосконалення методів кодування, шаблонів проектування, тестів, які можна писати як частину коду на основі різних методів, таких як граничні умови тощо.
- виконання програми або програми з метою вивчення поведінки.
- перегляд інфраструктури розгортання та пов'язаних сценаріїв і автоматизації.
- брати участь у виробничій діяльності, використовуючи методи моніторингу та спостереження.

Тестування програмного забезпечення може надати користувачам або спонсорам об'єктивну незалежну інформацію про якість програмного забезпечення та ризики його невдачі.

Розглянемо процес реєстрації користувача. Для того щоб зареєструватись користувач повинен заповнити всі поля, та пройти капчу (рис. 3.29).

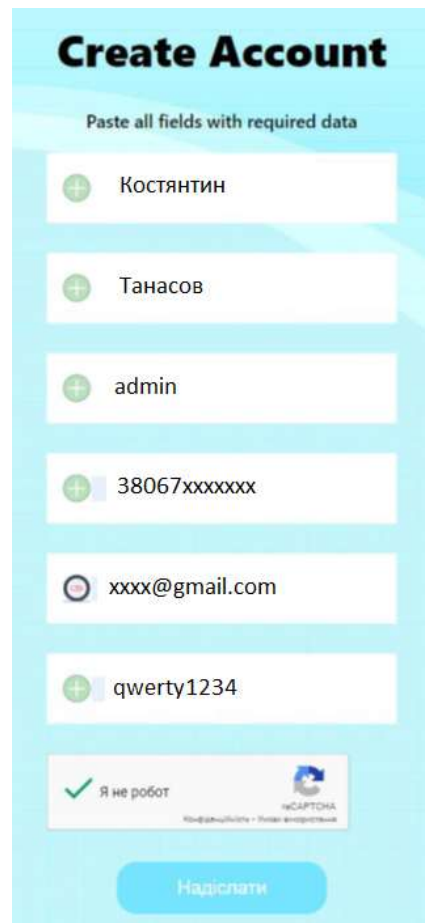


Рисунок 3.29 - Форма реєстрації

Після підтвердження реєстрації користувача перенаправить на головну сторінку додатку, а в базі даних створиться відповідна таблиця, з інформацією які ввів користувач (рис. 3.30).



id	email	name	surname	phone	role	password	created_at	updated_at	deleted_at
1	admin@tanaseg.com	Kostiantin	Tanasov	38067040000	admin	380710970000	2023-02-04 17:05:04.6545	2023-02-04 17:05:04.75945	

Рисунок 3.30 - Створений користувач в базі даних

Під час реєстрації при спробі користувача відправити не валідні дані, форма не відправиться, а йому буде повідомлено про не відповідність даних (рис. 3.31).

A registration form on a light blue background. It contains four input fields. The first field is labeled 'Phone' with a green plus icon and has a red error message 'format is required' below it. The second field contains 'ЯЧСЯ' with a red error message 'Wrong email format' below it. The third field contains 'Я' with a green plus icon and a red error message 'Wrong password format' below it. The fourth field is a checkbox labeled 'Я не робот' with a CAPTCHA image to its right. At the bottom is a blue button labeled 'Надіслати'.

Рисунок 3.31 - Спроба введення не валідних даних

Під час автентифікації користувача необхідно вказати логін та пароль та пройти капчу, в результаті вірно ведених даних користувача буде перенаправлено на головну сторінку (рис. 3.32).

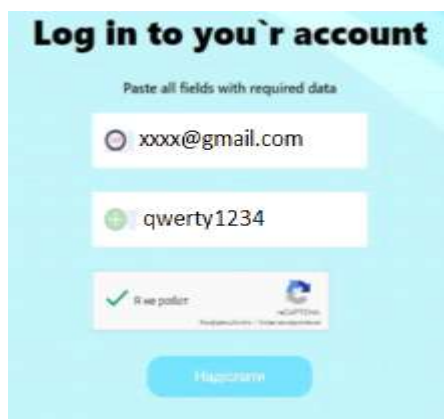
A login form titled 'Log in to you`r account' on a light blue background. It has a subtitle 'Paste all fields with required data'. There are three input fields: the first contains 'xxxx@gmail.com', the second contains 'qwerty1234', and the third is a checkbox labeled 'Я не робот' with a green checkmark icon to its left and a CAPTCHA image to its right. At the bottom is a blue button labeled 'Надіслати'.

Рисунок 3.32 - Авторизація користувача

Створення і видалення акаунтів для персоналу буде доступним лише для адміністратора, таку особливість було введено для того щоб контролювати

кількість користувачів, та обмежити можливість їх реєстрації. Для створення користувача адміністратору необхідно перейти на сторінку «Управління акаунтами» та натиснути кнопку «Додати». Після заповнення форми, на вказану адресу буде надіслано лист запрошення (рис. 3.33), з тимчасовим паролем, його потрібно буде ввести під час автентифікації, але даний пароль є тимчасовим, і його необхідно буде замінити.



Рисунок 3.33 - Лист запрошення з тимчасовим паролем

Після оновлення паролю в базі даних зміниться поле «isTemporary» що буде означати що користувач ввів власний постійний пароль, поки цього не відбудеться доступ користувача до додатку буде обмеженим.

Для тестування серверних маршрутів було використано бібліотеку Jest. У багатьох випадках модульні тести не дають точних результатів під час виконання на інтерфейсі будь-якого програмного забезпечення. Jest зменшує цю проблему, дозволяючи писати швидші й ефективніші зовнішні тести.

Ви також можете запускати тестові сценарії Jest у хмарі LambdaTest. Крім того, Jest можна використовувати для перевірки практично всього, що стосується JavaScript, особливо візуалізації веб-додатків у браузері. Завдяки своєму інтуїтивно зрозумілому API та легкому налаштуванню й інсталяції Jest став одним із найпопулярніших фреймворків тестування JavaScript, доступних сьогодні. Завдяки вбудованій бібліотеці знуцань, бібліотеці тверджень і запуску тестів, Jest діє як повноцінна платформа для тестування, яка дозволяє писати тести для бібліотечних проєктів JavaScript, таких як AngularJS, Vue JS, Node JS, Babel і TypeScript.

В ході розробки бібліотека Jest використовувалась для тестування маршрутів, і їх реакції на введені дані.

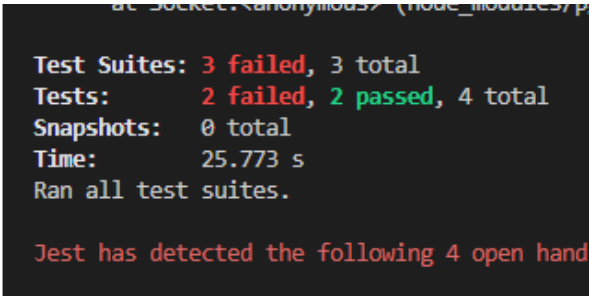
```
afterAll(async () => {
  await new Promise<void>(resolve => setTimeout(() => resolve(), 500));
});
describe('Testing Auth', () => {
  describe('[POST] /signup', () => {
    it('response should have the Create userData', () => {
      const userData: CreateUserDto = {
        email: 'test@email.com',
        password: 'q1w2e3r4!',
      };
      const authRoute = new AuthRoute();
      const app = new App([authRoute]);
      return request(app.getServer()).post('/signup').send(userData);
    });
  });
  describe('[POST] /login', () => {
    it('response should have the Set-Cookie header with the Authorization token', async ()
    => {
      const userData: CreateUserDto = {
        email: 'test@email.com',
        password: 'q1w2e3r4!',
      };
      process.env.JWT_SECRET = 'jwt_secret';
      const authRoute = new AuthRoute();
      const app = new App([authRoute]);
      return request(app.getServer())
        .post('/login')
```

```

.send(userData)
.expect('Set-Cookie', /^Authorization=.+/);
});
});
describe('[POST] /logout', () => {
it('logout Set-Cookie Authorization=; Max-age=0', () => {
const authRoute = new AuthRoute();
const app = new App([authRoute]);
return request(app.getServer())
.post('/logout')
.expect('Set-Cookie', /^Authorization=\;/);
});
});
});

```

Розглянемо ключові компоненти побудови тестів. Хук `afterAll` використовуватися для виконання налаштування лише один раз, на початку файлу. Це може бути особливо набридливим, коли налаштування є асинхронним, тому ви не можете зробити це вбудовано. Jest надає хуки `beforeAll` і `afterAll` для вирішення цієї ситуації. `describe(name, fn)` створює блок, який об'єднує кілька пов'язаних тестів. `it(string, fn)` - використовується для поділу тесту на блоки. В результаті виконання тестів буде відображено відповідну статистику (рис. 3.34).



```

Test Suites: 3 failed, 3 total
Tests:       2 failed, 2 passed, 4 total
Snapshots:   0 total
Time:        25.773 s
Ran all test suites.

Jest has detected the following 4 open hand

```

Рисунок 3.34 - Результат виконання тестування

ВИСНОВОК

У кваліфікаційної роботи магістра розроблено інформаційну систему для приймального відділення медичних закладів. В процесі виконання роботи проведено аналіз способів побудови інформаційних систем, в результаті якого було обрано архітектуру, сформульовано основні вимоги до інформаційної системи, її складові, та необхідний функціонал. Запропоновано ефективний підхід для реалізації простого і зрозумілого в користуванні кросплатформеного додатку, за рахунок використання сучасних і ефективних технологій проектування. За допомогою розробленої системи забезпечено ефективну комунікацію між всіма ланками медичного персоналу.

Актуальність теми проведеного дослідження зумовлена тим, що впровадження інформаційних технологій у сферу охорони здоров'я дозволяє покращити якість обслуговування, помітно прискорити роботу персоналу та знизити витрати на обслуговування для пацієнтів. Об'єктом дослідження є процес обміну даними в інформаційних системах. Предметом дослідження є методи та засоби обміну даними через різні платформи. Метою дослідження є вдосконалення методів комунікації в інформаційних системах за рахунок використання сучасних технологій проектування, таких як web-push-notification, firebase, PWA. На відміну від існуючих аналогів розроблюваний застосунок є кросплатформеним та кросбраузерним, що значно спрощує процес імплементації та вартість розробки, а для обміну інформацією використано технологію push-notification, яка працює як в браузерах так і на телефонах, в фоновому чи активному режимах.

Завдяки виконанню завдань дослідження здійснено аналіз способів побудови інформаційних систем, запропоновано ефективний підхід для реалізації простого і зрозумілого в користуванні кросплатформеного додатку, забезпечена ефективна комунікації між всіма ланками медичного персоналу.

Наукова новизна отриманих результатів полягає у тому, що удосконалено метод кросплатформеного обміну даними. Практичне значення одержаних

результатів полягає в тому, що розроблений додаток дозволяє забезпечити оперативну співпрацю медичного персоналу.

У першому розділі кваліфікаційної роботи магістра проведено аналіз способів побудови прогресивного додатку, у другому розділі здійснено вибір архітектури програмного забезпечення інформаційної системи, у третьому розділі розроблено програмне забезпечення інформаційної системи.

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. Що таке прогресивні веб-додатки [Електронний ресурс] - Режим доступу до ресурсу: <https://smile-ukraine.com/ua/pwa/introduction>
2. Web-додатки – переваги і недоліки [Електронний ресурс] - Режим доступу до ресурсу: https://mydiv.net/arts/view-web-dodatki_perevagy_i_nedliki.html
3. Розробка Web-додатків [Електронний ресурс] - Режим доступу до ресурсу: <http://clashdash/directions/web-applications>
4. Що таке Progressive Web App та в чому їх переваги [Електронний ресурс] - Режим доступу до ресурсу: <https://timeweb.com/community/articles/chtotakoeprogressive-web-apps>
5. Все що необхідно знати про Progressive Web App [Електронний ресурс] - Режим доступу до ресурсу: <https://habr.com/company/wrike/blog/481240/>
6. Методи життєвого циклу [Електронний ресурс] - Режим доступу до ресурсу: <https://dev-gang/article/metody-cikla-komponenta-vreactjs-m3v6725v7q/>
7. Node.js Documentation [Електронний ресурс] - Режим доступу до ресурсу: <https://nodejs.org/dist/latest-v14.x/docs/api/>
8. TypeScript [Електронний ресурс] - Режим доступу до ресурсу: <https://www.typescriptlang.org/>
9. Тестування ПЗ [Електронний ресурс] - Режим доступу: https://uk.wikipedia.org/wiki/Тестування_програмного_забезпечення.
10. Майерс Гленфорд Дж. Мистецтво тестування програм: Фінанси та статистика, 1982. — 114 с.
11. Статистика росту працівників [Електронний ресурс] - Режим доступу до ресурсу: <https://www.kmu.gov.ua/news/u-zakladah-shcho-uklali-dogovorz-nacionalnoyu-sluzhboyu-zdorovya-zroslo-kilkist-medichnih-pracivnikiv-esoz>
12. Using Service Workers [Електронний ресурс] - Режим доступу до ресурсу: https://developer.mozilla.org/en-US/docs/Web/API/Service_Worker_API/Using_Service_Workers

13. PostgreSQL [Электронный ресурс] - Режим доступа до ресурсу: <https://builtin.com/learn/tech-dictionary/postgresql>
14. Single-page application vs. multiple-page application [Электронный ресурс] - Режим доступа до ресурсу: <https://medium.com/@NeotericEU/single-pageapplication-vs-multiple-page-application-2591588efe58>
15. ReactJS Lifecycle of Components [Электронный ресурс] - Режим доступа до ресурсу: <https://www.geeksforgeeks.org/reactjs-lifecycle-components/>
16. What is React [Электронный ресурс] - Режим доступа до ресурсу: <https://flatlogic.com/blog/what-is-react/>
17. Lifecycle Methods [Электронный ресурс] - Режим доступа до ресурсу: <https://www.codecademy.com/learn/react-101/modules/react-102-lifecycle-methodsu/cheatsheet>
18. Harris Drucker, Yann Le Cun. Improving Generalization Performance Using Back-propagation / Harris Drucker, Yann Le Cun // IEEE Transactions on Neural Networks. - 1992. - Vol.3, №5 - p. 991-997.
19. Build on BNB Chain [Электронный ресурс] - Режим доступа: <https://docs.bnbchain.org/docs/bnbIntro>
20. What is Solana and how it work [Электронный ресурс] - Режим доступа: <https://minfin.com.ua/ua/invest/articles/shcho-take-solana/>
21. Blockclique: Scaling Blockchains through Transaction Sharding in a Multithreaded Block Graph [Электронный ресурс] - Режим доступа: <https://arxiv.org/pdf/1803.09029.pdf>
22. NgRx is a framework for building reactive applications in Angular [Электронный ресурс]. – Режим доступа до ресурсу: <https://ngrx.io/docs>
23. IPFS is a distributed system [Электронный ресурс]. - Режим доступа до ресурсу: <https://docs.ipfs.tech/concepts/what-is-ipfs/>
24. Getting started with WebStorm [Электронный ресурс] : echo / Режим доступа: <https://www.jetbrains.com/help/webstorm/getting-started-withwebstorm.html>

25. Visual Studio Code Tips and Tricks [Электронный ресурс]. – Режим доступа: до ресурсу: <https://code.visualstudio.com/docs/getstarted/tips-and-tricks>
26. How to Use Sublime Text: A Quick Overview [Электронный ресурс]. – Режим доступа до ресурсу: <https://kinsta.com/blog/how-to-use-sublime-text/>
27. Java is a powerful general-purpose programming language. [Электронный ресурс]. – Режим доступа: <https://www.programiz.com/javaprogramming>
28. <https://acloudguru.com/blog/engineering/what-is-the-javascriptprogramming-language>
29. «CPU vs. GPU Rendering: Which Should You Choose and Why?». IncrediBuild. <https://www.incredibuild.com/blog/cpu-vs-gpu-rendering-whichshould-you-choose-and-why> (accessed Nov. 21, 2022).
30. «CPU vs. GPU Renderer: Which is Better?» AcademyOfAnimatedArt. <https://academyofanimatedart.com/cpu-vs-gpu-renderer-which-isbetter/#:~:text=Modern%20GPUs%20offer%20superior%20processing,times%20faster%20than%20CPU%20rendering> (accessed Nov. 21, 2022).
31. «What is a Render Farm? Everything you need to know about Render Farms». RebusFarm. <https://us.rebusfarm.net/en/blog/2990-what-is-a-render-farmeverything-you-need-to-know-about-render-farms> (accessed Nov. 21, 2022).
32. «About». OneShader. <https://oneshader.net/about> (accessed Nov. 21, 2022).
33. «Modern Physically-Based Rendering». RenderMan. <https://renderman.pixar.com/tech-specs> (accessed Nov. 21, 2022).
34. «Download & Installation». RenderMan. https://renderman.pixar.com/intro_logged_in#:~:text=RenderMan%20is%20free%20for%20all,%2C%20research%2C%20and%20personal%20projects (accessed Nov. 21, 2022).
35. «Renderman Review». Slant. <https://www.slant.co/options/24860/~render-man-review> (accessed Nov. 21, 2022).
36. «Rendering». Blender. <https://www.blender.org/features/rendering/> (accessed Nov. 21, 2022).

37. «Winning the render wars with Chad Ashley». MotionGrapher. <https://motionographer.com/2017/08/14/winning-the-render-wars-with-chad-ashley/> (accessed Nov. 21, 2022).
38. A. Glawion. «What's the Best 3D Render Engine (GPU & CPU) for your Needs?». CGDirector. <https://www.cgdirector.com/best-3d-render-engines/> (accessed Nov. 21, 2022).
39. React vs Vue vs Angular [Електронний ресурс]. – Режим доступу: <https://habrreact.com/company/auriga/blog/703836/>
40. Aptos Token [Електронний ресурс]. – Режим доступу: <https://aptos.dev/concepts/coin-and-token/aptos-token/>
41. iOS [Електронний ресурс] // Режим доступу: <https://en.wikipedia.org/wiki/IOS>
42. Xcode [Електронний ресурс] // Режим доступу: <https://en.wikipedia.org/wiki/Xcode>
43. Kotlin [Електронний ресурс] // Режим доступу: <https://uk.wikipedia.org/wiki/Kotlin>
44. Android studio [Електронний ресурс] // Режим доступу: https://uk.wikipedia.org/wiki/Android_Studio
45. Андерс Йоранссон Ефективне використання потоків в операційній системі Android
46. Ияну Аделекан Kotlin. Програмування на прикладах
47. Федотенко М. Розробка мобільних додатків. Перші кроки.
48. Герберт Шилдт. Java 8. Посібник для початківців 2016. – 669 с.
49. Fundamentals of Testing [Електронний ресурс]. – 2021. – Режим доступу до ресурсу: <https://developer.android.com/training/testing/fundamentals>
50. Software R. Understanding The Difference Between MVC, MVP and MVVM Design Patterns [Електронний ресурс] / Rishabh Software. – 2021. – Режим доступу до ресурсу: <https://www.linkedin.com/pulse/understanding-difference-between-mvc-mvpmvvm-design-rishabh-software/>.

ДОДАТОК А

Схема інформаційної системи



ДОДАТОК Б

Лістинг головної сторінки застосунку

```
import React, { useEffect, useState } from 'react';
import './App.css';
import Panel from './Components/Panel/Panel';
import CalendarItem from './View/CalendarItem/CalendarItem';
import { Route, Routes, Navigate, useNavigate } from "react-router-dom";
import HistoryItem from './View/HistoryItem/HistoryItem';
import DoctorRequest from './View/DoctorRequest/DoctorRequest';
import Notifications from './View/Notification/Notification';
import RegisterPage from './View/RegisterPage/RegisterPage';
import { checkAuth, loginAction } from './root/actions/userActions';
import { thunkGetAllUsers } from './root/actions/allUsersAction';
import { useDispatch, useSelector } from 'react-redux';
import UserPage from './View/UserPage/UserPage';
import UsersManagment from './View/UsersManagment/UsersManagement';
import { getVerifyToken, onMessageListener } from './firebase';
function App() {
  const [show, setShow] = useState(false)
  const [notification, setNotification] = useState({title: "", body: ""});
  const [isTokenFound, setTokenFound] = useState(false);
  let data = useSelector(state => state.user)
  let redirect = useNavigate()
  const dispatch = useDispatch()
  useEffect(() => {
    if (data.isLoggedIn) {
      return redirect('/')
    } else {
      return redirect('/login')
    }
  }, [data.isLoggedIn])
  onMessageListener().then(payload => {
    setShow(true);
    setNotification({title: payload.notification.title, body: payload.notification.body})
    console.log(payload);
  }).catch(err => console.log('failed: ', err));
  useEffect(() => {
    const fetchData = async () => {
      if (localStorage.getItem('token')) {
        const response = await checkAuth()
        localStorage.setItem('token', response.accessToken)
        dispatch(loginAction(response.user))
        redirect('/')
      }
    }
    fetchData()
  })
}
```

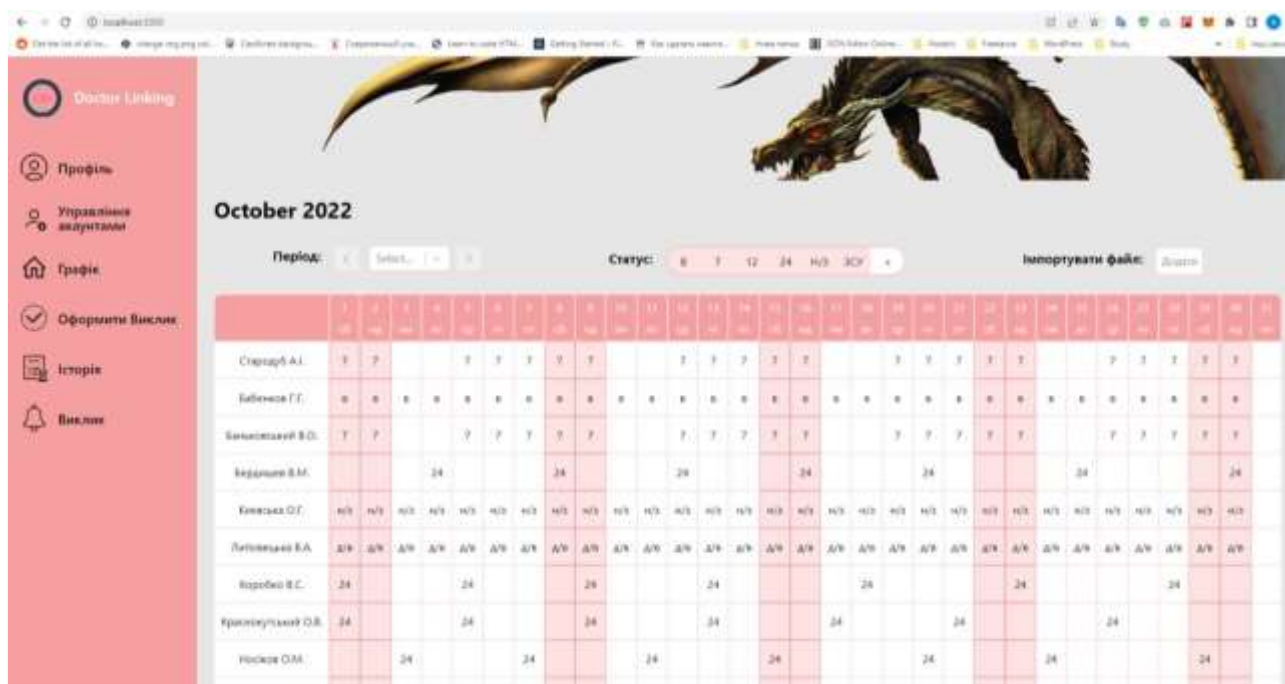
```

await dispatch(thunkGetAllUsers())
}
}
getVerifyToken(setTokenFound);
fetchData()
}, [])
return (
<div className="App">
{
data.isLoggedIn ? <Panel /> : null
}
{
show ? <p>{notification.body} {notification.title}</p> : null
}
<Routes>
<Route path="/doctorRequest" element={<DoctorRequest />} />
<Route path="/user" element={<UserPage />} />
<Route path="/user-management" element={<UsersManagmant />} />
<Route path="/notification" element={<Notifications />} />
<Route path="/login" element={<RegisterPage />} />
<Route path="/" element={<RequireAuth
redirectTo={'/login'}
><CalendarItem /> </RequireAuth>} />
<Route path="/history" route={'/history'} element={
<RequireAuth redirectTo={'/login'}>
<HistoryItem />
</RequireAuth>
} />
<Route path="/protected"
element={
<RequireAuth redirectTo="/login">
<RegisterPage />
</RequireAuth>} />
</Routes>
</div>
);
}
export default App;
function RequireAuth({ children, redirectTo }) {
let data = useSelector(state => state.user)
return (
data.isLoggedIn ? children : <Navigate to={redirectTo} />
)
}

```

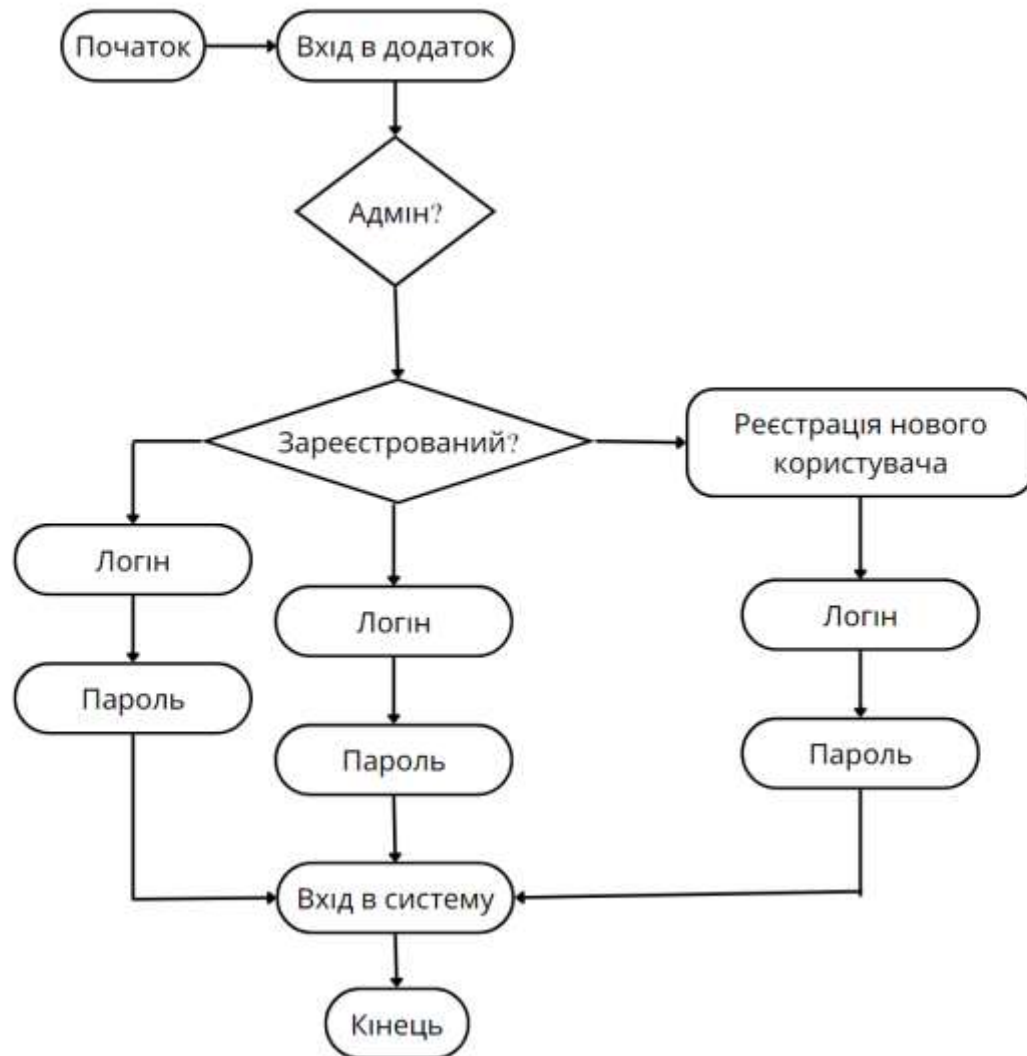
ДОДАТОК В

Головна сторінка застосунку



ДОДАТОК Г

Блок схема входу в систему



ДОДАТОК Д

Блок схема виходу з системи



ДОДАТОК Е

Лістинг серверної частини

```

import 'reflect-metadata';
import cookieParser from 'cookie-parser';
import cors from 'cors';
import express from 'express';
import helmet from 'helmet';
import hpp from 'hpp';
import morgan from 'morgan';
import compression from 'compression';
import swaggerUi from 'swagger-ui-express';
import swaggerJSDoc from 'swagger-jsdoc';
import { createConnection } from 'typeorm';
import { AppDataSource } from './database';
import Routes from './interfaces/routes.interface';
import errorMiddleware from './middlewares/error.middleware';
import { logger, stream } from './utils/logger';
class App {
  public app: express.Application;
  public port: string | number;
  public env: string;
  constructor(routes: Routes[]) {
    this.app = express();
    this.port = process.env.PORT || 3000;
    this.env = process.env.NODE_ENV || 'development';
    this.connectToDatabase();
    this.initializeMiddlewares();
    this.initializeRoutes(routes);
    this.initializeSwagger();
    this.initializeErrorHandling();
  }
  public listen() {
    this.app.listen(this.port, () => {
      logger.info(`🚀 App listening on the port ${this.port}`);
    });
  }
  public getServer() {
    return this.app;
  }
  private connectToDatabase() {
    createConnection(AppDataSource)
      .then(() => {
        logger.info('🟢 The database is connected.');
```

```

    })
    .catch((error: Error) => {
    logger.error('● Unable to connect to the database: ${error}.');
    });
  }
  private initializeMiddlewares() {
    if (this.env === 'production') {
      this.app.use(morgan('combined', { stream }));
      this.app.use(cors({ origin: 'your.domain.com', credentials: true }));
    } else if (this.env === 'development') {
      this.app.use(morgan('dev', { stream }));
      this.app.use(cors({ origin: 'http://localhost:3000', credentials: true }));
    }
    this.app.use(hpp());
    this.app.use(helmet());
    this.app.use(compression());
    this.app.use(express.json());
    this.app.use(express.urlencoded({ extended: true }));
    this.app.use(cookieParser());
  }
  private initializeRoutes(routes: Routes[]) {
    routes.forEach(route => {
      this.app.use('/', route.router);
    });
  }
  private initializeSwagger() {
    const options = {
      swaggerDefinition: {
        info: {
          title: 'REST API',
          version: '1.0.0',
          description: 'Example docs',
        },
      },
      apis: ['swagger.yaml'],
    };
    const specs = swaggerJSDoc(options);
    this.app.use('/api-docs', swaggerUi.serve, swaggerUi.setup(specs));
  }
  private initializeErrorHandling() {
    this.app.use(errorMiddleware);
  }
}
export default App;

```