Вінницький національний технічний університет
(повне найменування вищого навчального закладу)
Факультет інформаційних технологій та комп'ютерної інженерії
(повне найменування інституту)
Кафедра обчислювальної техніки
(повна назва кафедри)

МАГІСТЕРСЬКА КВАЛІФІКАЦІЙНА РОБОТА

на тему:

«Інформаційна система приймального відділення лікарні»

ПОЯСНЮВАЛЬНА ЗАПИСКА

08-23.МКР.003.00.000 ПЗ

Виконав: студент 2 курсу, групи 1КІ-21м
спеціальності 123 — «Комп'ютерна інженерія»
_____ Кацалап А. Ю.
Керівник: к.т.н., доц.каф. ОТ
_____ Тарновський М. Г.
«21» 12 2022 р.
Опонент: к.т.н., доц. каф. ПЗ
_____ Войтко В. В.
«22» 12 2022 р.

Допущено до захисту

Завідувач кафедри ОТ

_____ д.т.н., проф. Азаров О. Д.

«23» 12 2022 р

Вінниця ВНТУ - 2022 рік

...офесійні програми

Вінницький національний технічний університет
Факультет Інформаційних технологій та комп'ютерної інженерії
Кафедра Обчислювальної техніки
Рівень вищої освіти ІІ-ий (магістерський)
Галузь знань 12 – Інформаційні технології
Спеціальність 123 – «Комп'ютерна інженерія»
Освітньо-професійна програма «Комп'ютерна інженерія»

ЗАТВЕРДЖУЮ
Завідувач кафедри
обчислювальної техніки
д.т.н., проф. Азаров О. Д.
«15» 09 2022 року

З А В Д А Н Н Я
НА МАГІСТРСЬКУ КВАЛІФІКАЦІЙНУ РОБОТУ

студенту Кацалапу Андрію Юрійовичу

1 Тема роботи «Інформаційна система приймального відділення лікарні», керівник роботи Тарновський Микола Геннадійович, к.т.н., доцент затверджені наказом Вінницького національного технічного університету від "15" 09 2022 року №205-А

2 Терміу подання студентом роботи 23.12.2022 р.

3 Вихідні дані до роботи: база даних Postgre; сервер NodeJS; застосунок React PWA; тип застосунку кросплатформений.

4 Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити): аналіз способів побудови прогресивного додатку; розробка архітектури програмного забезпечення; вибір архітектури програмного комплексу; розробка програмного забезпечення інформаційної системи; тестування; економічна частина.

5 Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень): схема інформаційної системи , блок схема алгоритму входу в систему, блок схема алгоритму виходу з системи.

6. Консультанти розділів роботи приведені в таблиці 1

ність рекомендувати
я лабораторних робіт з
ві спеціальності 123
рофесійні програми

Таблиця 1 — Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
1,2,3	Тарновський М. Г. к.т.н., доцент		
4	Небава М. І. к.е.н., проф.	Небава	Небава

7.Дата видачі завдання 06.09.2022р.

8 Календарний план виконання МКР приведений в таблиці 2.

Таблиця 2 — Календарний план

№ з/п	Назва етапів МКР	Строк виконання	Примітка
1	Постановка задачі	06.09.22	вик
2	Огляд існуючих рішень	09.09.22	вик
3	Вибір архутектури програмного забезпечення інформаційної системи	15.09.22	вик
4	Розробка програмного забезпечення інформаційної системи	22.11.22	вик
5	Розрахунок економічної частини	29.11.22	вик
6	Оформлення пояснювальної записки та ілюстративного матеріалу	15.12.22	вик
7	Перевірка якості виконання магістерської кваліфікаційної роботи та усунення недоліків	18.12.22	вик
8	Підписи супроводжувальних документів у керівника, опонента, нормоконтролера	19.12.22	вик
9	Перевірка «антиплагіат»	15.12.22	вик
10	Попередній захист	19.11.22	вик.

Студент _____ Кацалап А. Ю.
(підпис) (прізвище та ініціали)

Керівник роботи _____ Тарновський М. Г.
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

Кацалап А. Ю. Інформаційна система приймального відділення лікарні. Магістерська кваліфікаційна робота зі спеціальності 123 — Комп'ютерна Інженерія, Вінниця: ВНТУ, 2022.

На укр. мові. Бібліогр.: 12 назв; рис.: 43; табл. 5.

Дана магістерська дипломна робота присвячена темі розробки медичної інформаційної системи з використанням сучасних технологій проектування.

В даній роботі зроблено аналіз відомих підходів по створенню веб-додатків. Також зроблено аналіз використання мов програмування, запропонований спосіб побудови клієнтської та серверної частини додатку, розроблена структура бази даних.

Ключові слова: React, PWA, NodeJS, Postgre, інформаційна система.

ANNOTATION

Katsalap A. Yu. Information system of the reception department of the hospital. Master's thesis on the specialty 123 — Computer Engineering, Vinnytsia: VNTU, 2022.

In Ukrainian speech Bibliography: 20 titles; Fig.: 43; table 5.

This master's thesis is devoted to the topic of developing a medical information system using modern design technologies.

This paper analyzes known approaches to creating web applications. An analysis of the use of programming languages was also made, a method of building the client and server parts of the application was proposed, and the database structure was developed.

Keywords: React, PWA, NodeJS, Postgre, інформаційна система.

ЗМІСТ

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 формаційниxx 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 функональною 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.

Інформаійні 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 лікуванні 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 магістерської роботи — процес обміну даними в інформаційних системах.

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

Метою дослідження магістерської роботи 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 процесс 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 магістерської роботи:

— здійснити аналіз способів побудови інформаційних систем;

— запропонувати ефективний підхід для реалізації простого і зрозумілого в користуванні кросплатформеного 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 магістерської роботи полягає у тому, що удосконалено метод кросплатформеного обміну даними.

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

Апробація Апробація результатів роботи здійснена в доповіді на Всеукраїнській науково-практичній Інтернет-конференції студентів, аспірантів та молодих науковців «Молодь в науці: дослідження, проблеми, перспективи (МН-2023)» (ВНТУ).

Матеріали роботи доповідались та опубліковувались:

Кацалап А. Ю. Особливості відправки push-сповіщень в прогресивних веб-додатках / А. Ю. Кацалап, М. Г. Тарновський // Всеукраїнська науково-практична Інтернет-конференція студентів, аспірантів та молодих науковців «Молодь в науці: дослідження, проблеми, перспективи (МН-2023)» (15 листопада 2022 р .- 12 травня 2023 р., Вінниця) : Режим доступу: https://conferences.vntu.edu.ua/index.php/mn/mn2023/paper/view/16652.

1 АНАЛІЗ СПОСОБІВ ПОБУДОВИ ПРОГРЕСИВНОГО ДОДАТКУ

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.

1.1 Поняття прогресивного веб-додатку та їх переваги

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 [3].

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.

Progresive Web Application — поєднує властивості нативної програми з функціональністю браузера і має свої переваги:

— 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 оноволення завантужуються 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.

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.

Предствлено 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. Адаптивний дизайн робитьваш 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 найваливіших 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 Бібліотека React

React.js був випущений інженером-програмістом, який працював на Facebook – Джорданом Волке, у 2011 році. React – це бібліотека JavaScript, зосереджена на створенні декларативних інтерфейсів користувача (UI) за допомогою концепції на основі компонентів. Він використовується для обробки шару перегляду та може використовуватися для веб-застосунків і мобільних програм. Головна мета React — бути обширним, швидким, декларативним, гнучким і простим.

React — це не фреймворк, це саме бібліотека. Пояснення цього полягає в тому, що React займається лише рендерингом інтерфейсів користувача та залишає багато речей на розсуд окремих проектів. Стандартний набір інструментів для створення програми за допомогою ReactJS часто називають стеком[6].

Об’єктна модель документа (DOM) — це API для дійсних документів HTML і добре сформованих XML.

Віртуальний DOM — це представлення справжнього DOM, створеного/обробленого браузерами. Розширені бібліотеки, такі як React, генерують дерево елементів у пам’яті, еквівалентне справжньому DOM, яке формує віртуальний DOM декларативним способом. Віртуальний DOM є однією з особливостей, які роблять фреймворк настільки швидким і надійним.

ReactJS — це бібліотека на основі компонентів, де компоненти роблять наш код придатним для повторного використання та розділяють наш інтерфейс користувача на різні частини. Компоненти поділяються на два типи: компоненти класу та компоненти функції. Усі компоненти React дотримуються принципу поділу проблем, що означає, що ми повинні розділити нашу програму на різні розділи для вирішення окремих проблем.

Отже, головне питання полягає в тому, чому ви повинні вибрати ReactJS як стек розробки інтерфейсу, тоді як є багато інших. Ось кілька причин:

— реактивінсть;
— підтримка компонентів;
— простота у використанні;
— пошукова оптимізація;
— одностороннє використання даних.

Реактивність. React дозволяє розробникам використовувати окремі частини своєї програми як на стороні клієнта, так і на сервері, і будь-які внесені ними зміни не вплинуть на логіку програми. Це робить процес розробки надзвичайно швидким.

Підтримка компонентів. Використання тегів HTML і кодів JS полегшує роботу з величезним набором даних, що містить DOM. React діє як посередник, який представляє DOM і допомагає вам вирішити, який компонент потребує змін, щоб отримати точні результати.

Простий у використанні та навчанні. ReactJS неймовірно зручний і робить будь-який інтерфейс користувача інтерактивним. Це також дозволяє швидко й ефективно створювати програми, що економить час як для клієнтів, так і для розробників.

Пошукова оптимізація. Поширена проблема, на яку скаржаться більшість веб-розробників, полягає в тому, що традиційні фреймворки JavaScript часто мають проблеми з SEO. ReactJS вирішує цю проблему, допомагаючи розробникам легко орієнтуватися в різних пошукових системах завдяки тому, що програма ReactJS може працювати на сервері, а віртуальний DOM візуалізує та повертає її у браузер як веб-сторінку.

Одностороннє прив'язування даних. Одностороння прив’язка даних передбачає, що будь-хто може відстежити всі зміни, внесені в сегмент даних. Це також одна з причин, чому React такий зручний під час розробки.

JSX — це синтаксис розширення JavaScript, який використовується в React для легкого спільного написання HTML і JavaScript. (рис. 1.1).

```jsx
function Farms() {

    return (
        <div className={classes.Farms}>
            <div className={classes.Header}>
                <PageTitle title={'Farms'}/>
                <div className={classes.FarmsFilter}>
                    <Button name={'Staked only'} class={'Primary'}/>
                    <Button name={'Active'} class={'Primary'}/>
                </div>
            </div>
            <div className={classes.FarmList}>
                <FarmCreate/>
                {
                    farms.map((farm, id) => {
                        return (
                            <Farm farm={farm} key={`Farm-${id}`}/>
                        )
                    })
                }
            </div>
        </div>
    );
}
```

Рисунок 1.1 — Приклад синтаксису JSX

Це простий код JSX у React. Але браузер не розуміє цей JSX, оскільки це недійсний код JavaScript. Це тому, що ми призначаємо HTML-тег змінній, яка не є рядком, а просто HTML-кодом.

Тому, щоб перетворити його на зрозумілий для браузера код JavaScript, у React використовується такий інструмент, як Babel, який є компілятором/транспілятором JavaScript.

Можна налаштувати власну конфігурацію babel за допомогою Webpack. Або можна використовувати create-react-app, який внутрішньо використовує Babel для перетворення JSX у JavaScript.

Кожен елемент JSX перетворюється на виклик функції React.createElement, який розуміє браузер.

React.createElement має такий синтаксис:

React.createElement(type, [props], [...children])

React використовує Компонент є одним із основних будівельних блоків React. Іншими словами, ми можемо сказати, що кожна програма, яку ви розроблятимете в React, складатиметься з частин, які називаються компонентами. Компоненти значно полегшують завдання створення інтерфейсів користувача. Ви можете бачити інтерфейс користувача, розбитий на кілька окремих частин, які називаються компонентами, і працювати над ними незалежно та об'єднувати їх усі в батьківський компонент, який стане вашим остаточним інтерфейсом користувача[7].

Загальний вигляд функціональної компоненти представлено на (рис.1.2).

```
function Hello(){
    return (
        <div>
            <p>Hellow World</p>
        </div>
    )
}
```

Рисунок 1.2 — Структура React-компоненти

Кожен компонент React має власний життєвий цикл, життєвий цикл компонента можна визначити як ряд методів, які викликаються на різних етапах існування компонента. Компонент React може пройти чотири етапи свого життя (рисунок 1.3).

Життєвий цикл компонента знаходиться в таких станах:

— монтування (додавання вузлів до DOM);

— оновлення (зміна існуючих вузлів у DOM);

— демонтування (видалення вузлів із DOM);

— обробка помилок (перевірка, що ваш код працює та не містить помилок)

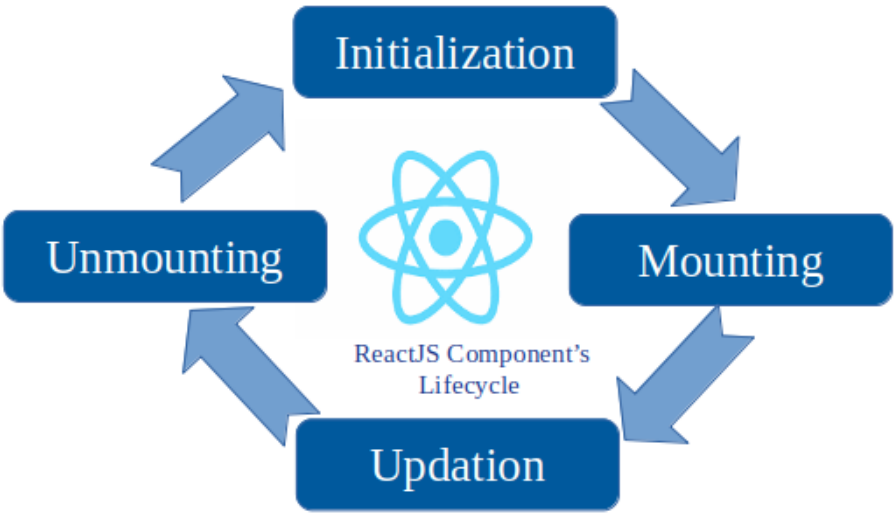


Рисунок 1.3 — Методи життєвого циклу компоненти

Перед візуалізацією компонент має пройти фазу монтування, оновлення та демонтування. Давайте розберемо це далі.

Встановлення компонента — це все одно, що народжена дитина. Це перший проблиск життя компонента. На цьому етапі компонент, який складається з вашого коду та внутрішніх елементів React, потім вставляється в DOM.

Після фази монтування компонент React «зростає» під час фази оновлення. Без оновлень компонент залишатиметься таким, яким він був під час початкового створення в DOM. Як ви можете собі уявити, багато компонентів, які ви пишете до, потребують оновлення, через зміну стану чи пропсів. Відповідно, вони також проходять етап оновлення.

Остання фаза називається фазою демонтажу. На цьому етапі компонент «гине». У жаргоні React його видалено з DOM.

Є ще одна фаза, яку може пройти компонент React: фаза обробки помилок. Це трапляється, коли ваш код не працює або десь є помилка. Подумайте про це як про щорічний медичний огляд.

Кожна фаза життєвого циклу React має ряд методів життєвого циклу, які ви можете змінити, щоб запускати код у визначений час під час процесу. Вони широко відомі як методи життєвого циклу компонентів.

На (рис. 1.4) показано методи життєвого циклу React, пов’язані з фазами життєвого циклу встановлення, оновлення, розмонтування та помилок:

Рисунок 1.4 — Блок схема методів життєвого циклу компоненти

Фаза монтування стосується фази, під час якої компонент створюється та вставляється в DOM.

Після виклику render компонент монтується до DOM і викликається метод componentDidMount. Ця функція викликається одразу після монтування компонента до DOM. Ви б використали метод життєвого циклу componentDidMount, щоб отримати вузол DOM із дерева компонентів відразу після його монтування.

Кожного разу, коли вноситься зміна в стан або властивості компонента React, компонент повторно відтворюється. Простіше кажучи, компонент оновлюється. Це етап оновлення життєвого циклу компонента React.

Метод життєвого циклу componentDidUpdate викликається після getSnapshotBeforeUpdate. Як і метод getSnapshotBeforeUpdate, він отримує попередні властивості та стан як аргументи:

```
componentDidUpdate(prevProps, prevState) {
}
```

Будь-яке значення, яке повертає метод життєвого циклу getSnapshotBeforeUpdate, передається як третій аргумент методу componentDidUpdate.

Метод життєвого циклу componentWillUnmount викликається безпосередньо перед тим, як компонент демонтується та знищується. Це ідеальне місце для виконання будь-якого необхідного очищення, наприклад очищення таймерів, скасування мережевих запитів або очищення будь-яких підписок, створених у componentDidMount().

Коли у вашому коді все йде погано, виникають помилки. Наступні методи викликаються, коли помилка видається компонентом-нащадком (тобто компонентом нижче).

Метод componentDidCatch також викликається після виникнення помилки в компоненті-нащадку. Крім викинутої помилки, йому передається ще один аргумент, який представляє більше інформації про помилку.

```
componentDidCatch(error, info) {
  logToExternalService(error, info)
}
```

1.4 Платформа Node.js

Node.js дозволяє розробникам створювати як зовнішні, так і внутрішні програми за допомогою JavaScript. Його випустив у 2009 році Райан Дал. (рис. 1.5). Node.js є відкритим кодом: це означає, що вихідний код для Node.js є загальнодоступним. І його підтримують учасники з усього світу. Посібник із внесення до Node.js показує, як зробити внесок.

Node.js є кросплатформним: Node.js не залежить від програмного забезпечення операційної системи. Він може працювати на Linux, macOS або Windows.

Node.js — це середовище виконання JavaScript: коли ви пишете код JavaScript у своєму текстовому редакторі, цей код не може виконувати жодне

завдання, доки ви його не виконаєте (або не запустите). А щоб запустити ваш код, вам потрібне середовище виконання[8].

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

const hostname = '127.0.0.1';
const port = 3000;

const server = http.createServer((req, res) => {
  res.statusCode = 200;
  res.setHeader('Content-Type', 'text/plain');
  res.end('Hello World');
});

server.listen(port, hostname, () => {
  console.log(`Server running at http://${hostname}:${port}/`);
});
```

Рисунок 1.5 — Проста програма Node.js

Такі веб-переглядачі, як Chrome і Firefox, мають середовища виконання. Ось чому вони можуть запускати код JavaScript. До створення Node.js JavaScript міг працювати лише в браузері. І він використовувався для створення лише зовнішніх додатків.

Node.js забезпечує середовище виконання поза браузером. Він також побудований на движку JavaScript Chrome V8. Це дає змогу створювати внутрішні додатки, використовуючи ту саму мову програмування JavaScript.

І браузер, і Node.js здатні виконувати програми JavaScript. Але є деякі ключові відмінності, які вам потрібно знати. Вони включають наступне.

Доступ до DOM API. За допомогою середовища виконання браузера ви можете отримати доступ до об’єктної моделі документа (DOM). І ви можете виконувати всі операції DOM. Але Node.js не має доступу до DOM. Node.js надає вашим програмам майже всі системні ресурси. Це означає, що ви можете взаємодіяти з операційною системою, отримувати доступ до файлових систем,

читати та записувати файли. Але ви не маєте доступу до операційних систем і файлових систем із браузера.

За допомогою Node.js ви можете вибрати, на якій версії запускати програму на стороні сервера. У результаті ви можете використовувати сучасні функції JavaScript, не турбуючись про будь-які невідповідності версій.

Порівняйте це з середовищем виконання браузера. Як розробник, ви не можете контролювати версії браузерів, які ваші клієнти використовують для доступу до вашої програми.

Node.js пропонує готову підтримку для модулів CommonJS і ES. Ви можете завантажити модулі за допомогою ключового слова require (синтаксис CommonJS) і ключового слова import (синтаксис ES).

Деякі сучасні браузери підтримують модулі ES. Це означає, що ви можете використовувати імпортовані модулі ES. Але вам все одно потрібно буде створювати пакети, щоб обслуговувати старіші браузери, які не підтримують модулі ES.

Однією з переваг Node.js є те, що він дозволяє вам працювати як на інтерфейсі, так і на сервері програми. І для цього ви використовуєте одну мову програмування — JavaScript.

Це хороша новина для інтерфейсних розробників, які працюють з JavaScript. Якщо ви хочете почати працювати на стороні сервера, це легше, ніж вивчати нову серверну мову з нуля.

1.5 Мова програмування TypeScript

TypeScript є надмножиною JavaScript, тобто він виконує все, що й JavaScript, але з деякими додатковими функціями. Основною причиною використання TypeScript є додавання статичної типізації до JavaScript. Статична типізація означає, що тип змінної не можна змінити в будь-якому місці програми. Це може запобігти великій кількості помилок. TypeScript не сприймається браузерами, тому він має бути скомпільований у JavaScript за допомогою компілятора TypeScript (TSC).

Переваги використання typescript:

— дослідження показали, що TypeScript може виявити 15% поширених помилок;

— читабельність — легше побачити, що код повинен робити, а коли працюєш у команді, легше побачити, що задумали інші розробники;

— це популярно — знання TypeScript дозволить вам шукати більше хороших вакансій;

— вивчення TypeScript дасть вам краще розуміння та новий погляд на JavaScript.

Недоліки typescript:

— написання TypeScript займає більше часу, ніж JavaScript, оскільки вам потрібно вказати типи, тому для невеликих індивідуальних проектів його використання може бути не варто.

— TypeScript потрібно скомпілювати, що може зайняти час, особливо у великих проектах.

Але додатковий час, який вам доведеться витратити на написання більш точного коду та компіляцію, буде більш ніж заощаджено тим, скільки менше помилок ви матимете у своєму коді.

Для багатьох проектів, особливо середніх і великих проектів, TypeScript заощадить вам багато часу та головного болю.

І якщо ви вже знаєте JavaScript, TypeScript не буде надто складним для вивчення. Це чудовий інструмент, який варто мати у своєму арсеналі[9].

У JavaScript примітивне значення — це дані, які не є об’єктом і не мають методів. Існує 7 примітивних типів даних:

— string;

— number;

— bigint;

— boolean;

— undefined;

— null;

— symbol.

Примітиви незмінні: їх не можна змінити. Важливо не плутати сам примітив зі змінною, якій присвоєно примітивне значення. Змінній можна перепризначити нове значення, але існуюче значення не можна змінити таким чином, як можна змінити об’єкти, масиви та функції.

Повертаючись до TypeScript, ми можемо встановити тип, який ми хочемо додати до змінної: тип (так званий «анотація типу» або «сигнатура типу») після оголошення змінної. приклади:

```
let id: number = 5;
let firstname: string = 'danny';
let hasDog: boolean = true;
let unit: number;
unit = 5;
```

У JavaScript майже «все» є об’єктом. Насправді (і незрозуміло), рядки, числа та логічні значення можуть бути об’єктами, якщо їх визначено за допомогою нового ключового слова:

Отже, JavaScript має динамічний тип, тобто змінну, оголошену як число, можна перетворити на рядок, оскільки TypeScript має статичну типізацію, тобто ви заздалегідь оголошуєте, який тип значення буде містити змінна, і воно не змінюється.

По суті, TypeScript намагається допомогти JavaScript досягти нових висот і стати дуже масштабованим. Його можна виділити за такими особливостями:

— безкоштовна мова програмування з відкритим кодом, розроблена та підтримувана Microsoft;

— суворийий синтаксичний супернабір JavaScript, який компілюється у звичайний JavaScript;

— полегшує розробку великих програм, написаних на JavaScript;

— розширює JavaScript, додаючи статичні типи, класи, модулі, інтерфейси та генерики.

1.6 Система керування базами даних MongoDB

MongoDB — це документоорієнтована база даних з відкритим вихідним кодом, яка призначена для зберігання великого обсягу даних, а також дозволяє працювати з цими даними дуже ефективно. Її класифікують як базу даних NoSQL (не тільки SQL), оскільки зберігання та отримання даних у MongoDB не здійснюється у формі таблиць.

База даних MongoDB розроблена та керована MongoDB.Inc згідно SSPL (Server Side Public License) і спочатку випущена в лютому 2009 року. Вона також забезпечує офіційну підтримку драйверів для всіх популярних мов, таких як C, C++, C# та .Net, Go, Java, Node.js, Perl, PHP, Python, Motor, Ruby, Scala, Swift, Mongoid. Таким чином, ви можете створити програму, використовуючи будь-яку з цих мов. Сьогодні існує так багато компаній, які використовують MongoDB, наприклад Facebook, Nokia, eBay, Adobe, Google тощо для зберігання великої кількості даних.

Тепер ми побачимо, як все відбувається за кадром. Як ми знаємо, MongoDB — це сервер баз даних, і дані зберігаються в цих базах даних. Іншими словами, середовище MongoDB дає вам сервер, який ви можете запустити, а потім створити на ньому кілька баз даних за допомогою MongoDB.

Завдяки базі даних NoSQL дані зберігаються в колекціях і документах. Отже, база даних, колекція та документи пов’язані між собою, як показано (рис1.6).

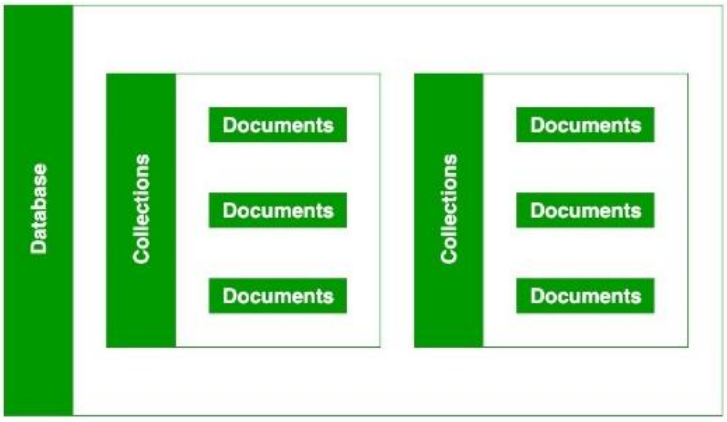


Рисунок 1.6 — Зв'язок бази з колекціями та документами

База даних MongoDB містить колекції так само, як база даних MYSQL містить таблиці. Вам дозволено створювати кілька баз даних і колекцій. Зараз у колекції ми маємо документи. Ці документи містять дані, які ми хочемо зберігати в базі даних MongoDB, і одна колекція може містити кілька документів, і ви не маєте схем, що означає, що один документ не обов'язково схожий на інший.

Документи створюються за допомогою полів. Поля — це пари ключ-значення в документах, це так само, як стовпці в базі даних відношень. Значення полів можуть мати будь-який тип даних BSON, як-от double, string, boolean тощо.

Дані, що зберігаються в MongoDB, мають формат документів BSON. Тут BSON означає двійкове представлення документів JSON. Іншими словами, сервер MongoDB перетворює дані JSON у двійкову форму, відому як BSON, і цей BSON зберігається та виконується більш ефективно.

У документах MongoDB ви можете зберігати вкладені дані. Таке вкладення даних дозволяє створювати складні зв'язки між даними та зберігати їх в одному документі, що робить обробку та отримання даних надзвичайно ефективними порівняно з SQL. У SQL вам потрібно написати складні об'єднання, щоб отримати дані з таблиць 1 і 2. Максимальний розмір документа BSON становить 16 МБ.

2 ВИБІР АРХІТЕКТУРИ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ 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.

2.1 Визначення структурної організації інформаційної системи

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 ШІ.

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 оптоволокно та коаксіальний кабель, тоді як бездротові технології включають радіохвилі та мікрохвилі.

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 дозволять 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.

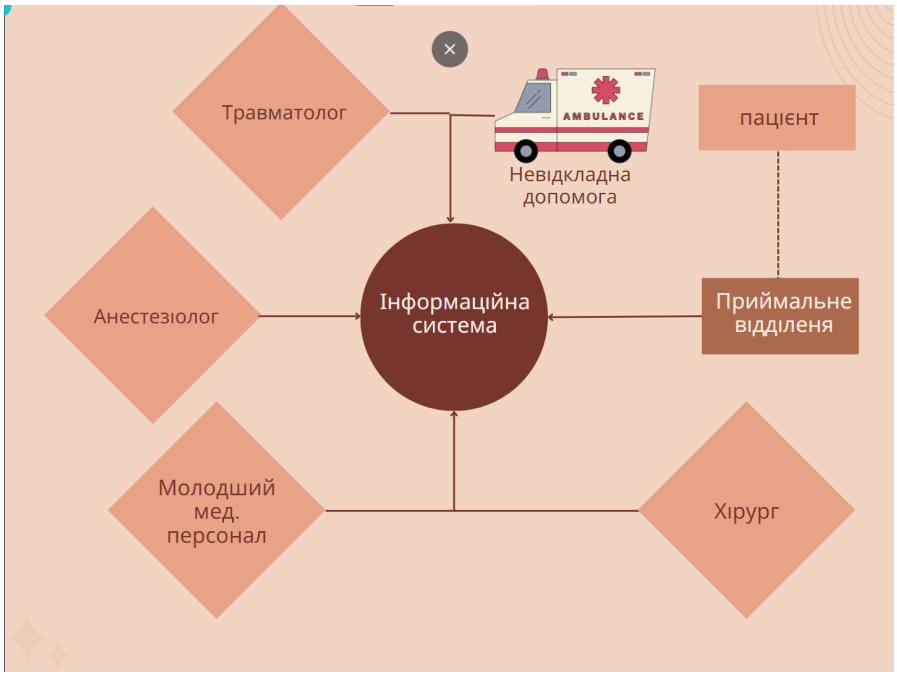


Рисунок 2.1 — Схема обміну інформацією між персоналом

2.2 Поняття односторінкового застосунку

Багатосторінкові сайти містять безліч сторінок, і коли користувачі запитують ресурси або взаємодіють з ним, програма має перейти на сервер, завантажити ресурс і показати його користувачеві. Процес досить повільний і болісний, коли мова йде про продуктивність чи про зручність користування, тому було вирішено будувати застосунок саме як Односторінковий додаток.

Односторінкові програми є найпопулярнішим способом створення веб-сайтів, які завантажуються швидше, не зачіпаючи сервер щоразу, коли користувач взаємодіє з програмою.

Односторінкова програма (SPA) — це платформа JavaScript для розподілу функціональних можливостей програми через Інтернет. Три найпопулярніші СПА засоби:

— Angular: розроблений Google і має свою популярність майже десять років;

— React: створений Facebook кілька років потому;

— Vue: розроблено колишнім співробітником Google і яке користується сплеском популярності за останні кілька років.

До появи односторінкових додатків додатки створювалися та відтворювалися на стороні сервера та доставлялися – повністю сформовані – на клієнтський пристрій: коли користувач переходив у додатку, кожна сторінка створювалася з нуля, тому її потрібно було постійно переглядати. оновлюється, що призводить до мерехтіння екрана. Веб-програми, що інтенсивно використовують дані, які отримували дані з кількох різних джерел, також можуть завантажуватися повільно.

У односторінкових програмах логіка презентації відображається на стороні клієнта, а візуальна структура веб-програми залишається незмінною протягом сеансу, а нещодавно запитувані дані оновлюються у фоновому режимі. Це забезпечує набагато плавніший досвід для користувача (рис. 2.2).

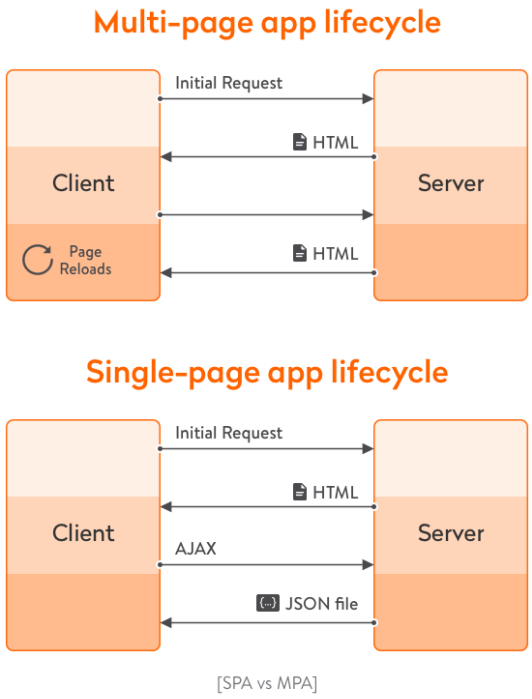


Рисунок 2.2 — Схема порівняння роботи підходів

SPA може запропонувати найкраще з обох світів — миттєву реакцію персонального докладання поряд з перенесенням і доступністю сайту. JavaScript SPA доступне більш ніж мільярду пристроїв, які підтримують сучасні браузери і

Переваги односторінкових додатків.

Ось деякі з переваг односторінкових програм, які роблять цей тип програм таким популярним і прийнятим деякими гігантами галузі, такими як Facebook, Gmail, Google Maps, LinkedIn і Netflix:

— швидкість завантаження;

— проста розробка;

— легко налагоджувати;

— можна розповсюджувати як прогресивну веб-програму;

— не потребують сторонніх модулів.

Найбільшою перевагою SPA є швидкість. Численні дослідження від Google та інших показали, що повільне завантаження негативно впливає на задоволеність користувачів, змушуючи покупців залишати візки та зменшуючи готовність відвідувачів повертатися на ваш сайт. Після завантаження односторінкової програми обсяг даних, що передається між клієнтським пристроєм і сервером, дуже малий, тому час завантаження мінімізується.

Проста розробка. Під час розробки односторінкової програми код на стороні сервера використовується повторно та ефективно відокремлюється від зовнішнього інтерфейсу користувача. Це означає, що команди бек-енду та фронт-енду можуть зосередитися на виконанні своєї роботи, не турбуючись про те, що розробляється в інших місцях.

Легко налагоджувати: дуже легко тестувати та налагоджувати односторінкові програми за допомогою інструментів розробника, наданих Google у веб-переглядачі Chrome.

Можна розповсюджувати як прогресивну веб-програму. Після створення SPA можна легко розповсюджувати як прогресивну веб-програму (PWA), щоб надати досвід, дуже схожий на мобільний додаток, але зі значно зменшеним слідом.

Не потребують сторонніх модулів. Приклавши трохи зусиль, можна зробити так, що воно буде працювати на настільних ПК, планшетах і смартфонах

з різними операційними системами. SPA легко оновлювати і поширювати, зазвичай це не вимагає ніяких дій з боку користувача [12].

Саме перераховані плюси односторінкового додатку перед звичайним сайтом роблять його оптимальним вибором для реалізації даного проекту.

Традиційний підхід багатосторінкової програми (MPA) не потребує досвіду роботи з JavaScript у ваших командах розробників (хоча поєднання інтерфейсу та серверної частини означає, що створення сайтів займає більше часу).

Ви можете масштабувати вміст як завгодно, додавши іншу сторінку, і оскільки вміст на кожній сторінці є статичним, пошукова оптимізація (SEO) загалом проста. З іншого боку, MPA працюють набагато повільніше, оскільки кожна нова сторінка завантажується з нуля. Однак якщо ваш веб-вміст (здебільшого) доступний лише для читання, MPA може надати вам усе необхідне.

Що стосується односторінкових додатків, їх основною перевагою є швидкість. Крім того, SPA набагато краще підходять для розкриття багатих функцій, ніж MPA, і вони також кешують інформацію, щоб програму можна було використовувати в автономному режимі.

Найбільшим недоліком SPA є те, що динамічний характер його вмісту ускладнює пошукову оптимізацію та видимість. Однак, оскільки все більше компаній застосовують SPA, сканери та пошукові системи розвиваються, щоб краще працювати з цим типом додатків.

Проте односторінкові програми не є кращими за багатосторінкові або навпаки. Обидва підходи мають свої переваги. Однак важливо зазначити, що коли проблеми веб-сканера та індексування, традиційно пов’язані з односторінковими програмами, будуть вирішені, переваги MPA над SPA почнуть зникати, і останні справді стануть стандартом де-факто для сучасних веб-програм.

2.3 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. 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. Рисунок 2.3.

Рисунок 2.3 — Структура файлу manifest.json

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. 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. Рисунок 2.4.

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, браузери також вважають що

локальний хост безпечне джерело.

```
if ('serviceWorker' in navigator) {
  window.addEventListener('load', () => {
    navigator.serviceWorker.register('/sw.js').then( registration => {
      // Registration was successful
      console.log(`ServiceWorker registred ${registration.scope}`);
    }, err => {
      // Registration failed
      console.log(`ServiceWorker registration failed, ${err}`);
    });
  });
}
```

Рисунок 2.4 — Функція Service worker

2.4 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).

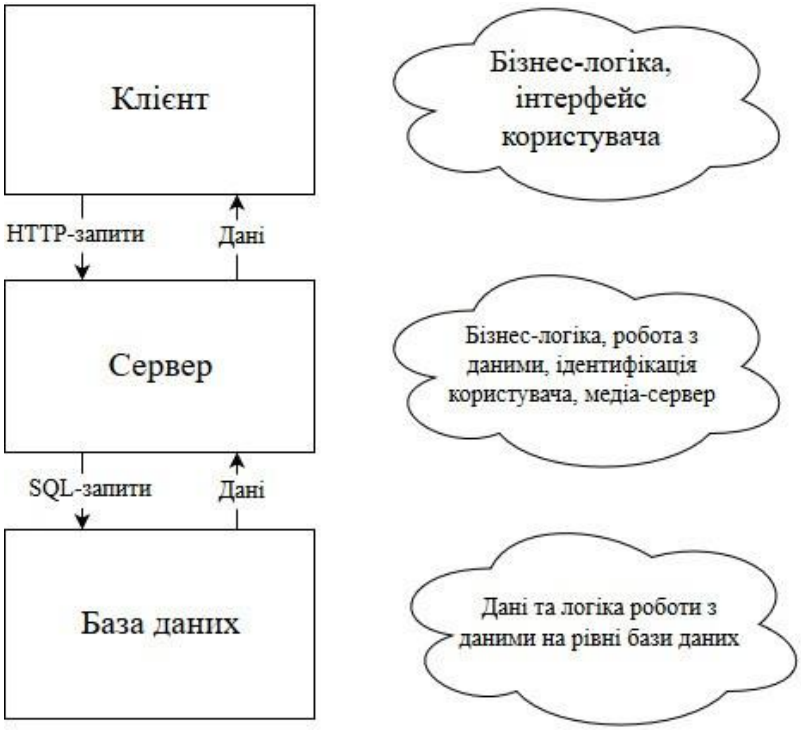


Рисунок 2.4 — Триланкова архітектура програмного комплексу

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.

2.5 Опис архітектури серверу

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 клієнт.

2.6 Пуш сповіщення в PWA

Сповіщення — короткі повідомлення, які з’являються на пристрої користувача. Сповіщення можуть надсилатися локально з відкритих програм,

яким це дозволено, або надсилатися із сервера користувачеві, коли програма не запущена. Вони можуть працювати без поштовху, але в поєднанні з ним вони дозволяють максимально використати цю функцію. Таким чином, push-повідомлення доставляють новий вміст до програми без будь-яких дій з боку клієнта. Вони є потужними маркетинговими інструментами, які можуть допомогти ефективно повторно залучити аудиторію, тримати її в курсі нових оновлень і, у певний момент, спонукати їх до певної дії. Багато компаній поєднують це зі збором і обробкою даних користувачів, щоб запускати push-повідомлення з дуже персоналізованим вмістом. Щойно користувачі погодяться отримати їх, вони отримають спеціальні повідомлення, створені на основі їх активності на веб-сайті. Це має великий потенціал, щоб зробити ваш бренд більш помітним і утримати клієнтів.

Push-сповіщення є однією з найважливіших функцій і переваг PWA. Вони дають користувачам відчуття рідності, навіть якщо вони використовують веб-застосунок, яким і є PWA. Автономний доступ, швидке завантаження та невелика вага створюють міцну основу для push-сповіщень, щоб розширити можливості прогресивних веб-програм. Push-сповіщення дозволяють PWA вийти далеко за межі браузера та дозволити користувачам взаємодіяти з вашою програмою, не повертаючись на ваш веб-сайт. Таким чином, PWA не тільки підтримує push-повідомлення, але й робить їх привабливими та популярними. Відображаючи значок вашої компанії та кілька рядків тексту, вони залучають споживачів і нагадують їм про вас. Вони також є для них цінним джерелом інформації, яку їм не потрібно шукати самостійно у ваших оновленнях, чи листах.

Щоб увімкнути push-сповіщення та офлайн-доступ загалом, PWA використовує Service Worker. Це файл JavaScript, який працює окремо від основного потоку браузера та дозволяє програмі контролювати мережеві запити, кешувати їх і отримувати доступ до отриманого вмісту. Для функції push-сповіщень Service Workers надають початкову точку для Notifications API і Push API. Завдяки першому API програма може відображати сповіщення для користувача. Другий API дозволяє додатку підписуватися на службу push і

отримувати повідомлення. Service Workers необхідно зареєструвати, встановити та активувати. З маніфестом веб-програми та деякими вимогами безпеки Service Workers є компонентами, які визначають веб-сайт як PWA.

Таким чином, після створення PWA та завершення налаштування Service Workers вам потрібно налаштувати перевірку дозволів, тобто чи дозволив користувач надсилання сповіщень. Крім того, запит користувача на дозволи має бути правильно закодований. Нарешті, вам також потрібно додати спосіб, за допомогою якого користувач відмовиться від push-повідомлень. Існують функції для всіх цих завдань. Звичайно, це дуже загальний опис налаштування push-повідомлень. Ймовірно, ваш розробник зробить цю роботу за вас, але варто знати, як виглядає процес.

Більшість провідних браузерів сьогодні дозволяють надсилати push-повідомлення. Тому веб-сайт із налаштованими веб-повідомленнями може надсилати їх на будь-який пристрій, який підтримує ці браузери, але користувачеві не потрібно нічого встановлювати. Крім того, користувачі можуть отримувати push-повідомлення під час перегляду іншого веб-сайту, і вам не потрібно мати програму для надсилання веб-повідомлень push.

Якщо бути точним, push-повідомлення програми та веб-повідомлення — це не одне й те саме. Раніше цей термін стосувався лише програм, але тепер він охоплює обидва типи. Отже, щоб нарешті відповісти, чи можуть веб-додатки мати push-повідомлення — так, вони можуть. Однак дуже важливо правильно це реалізувати, пам’ятаючи про Service Workers і HTTPS, оскільки веб-повідомлення push-повідомлень можна застосовувати лише в доменах із підтримкою HTTPS, щоб запобігти потенційним зловмисним атакам. Веб-сайти HTTP також можуть виконувати функції веб-натискання, але спочатку потрібно виконати певну роботу.

На практиці офіційної підтримки push-повідомлень від Apple немає. Проте є деякі способи, які розробники знаходять, щоб трохи обійти систему. Більшість із них вимагає реєстрації ідентифікатора програми на порталі розробників Apple із схваленням відповідної служби. Існує також спосіб використання Apple Wallet

для цієї мети. На жаль, це непрості способи, і вони справді належать до функціональності PWA. Можливо, у майбутньому Apple дозволить реалізувати push-повідомлення простим офіційним способом. Адже сам Стів Джобс говорив про концепцію PWA та розсилку сповіщень через них понад десять років тому.

Push-сповіщення є надзвичайною перевагою PWA. За правильного використання вони є потужним маркетинговим інструментом, який може приносити значний дохід. Якщо їх використовувати з розумом, передаючи відповідну інформацію споживачам, вони познайомлять користувачів із вашим брендом, заохочуть повернутися до вашої програми чи магазину та стануть лояльними постійними клієнтами. У поєднанні з аналітичними інструментами, які відстежують трафік користувачів у мережі та збирають дані про їх активність, push-сповіщення можуть стати потужним інструментом для розповсюдження інформації.

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.

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. 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).

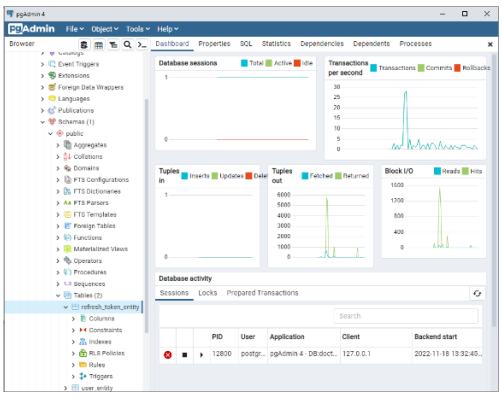


Рисунок 3.1 — Графічний інтерфейс клієнта pgAdmin

Після успішного логіна створюється тільки одна база данних — postgres. Для створення нових баз потрібно написати команди за допомогою консольного клієнта psql або використовуючи графічний клієнт pgAdmin.

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

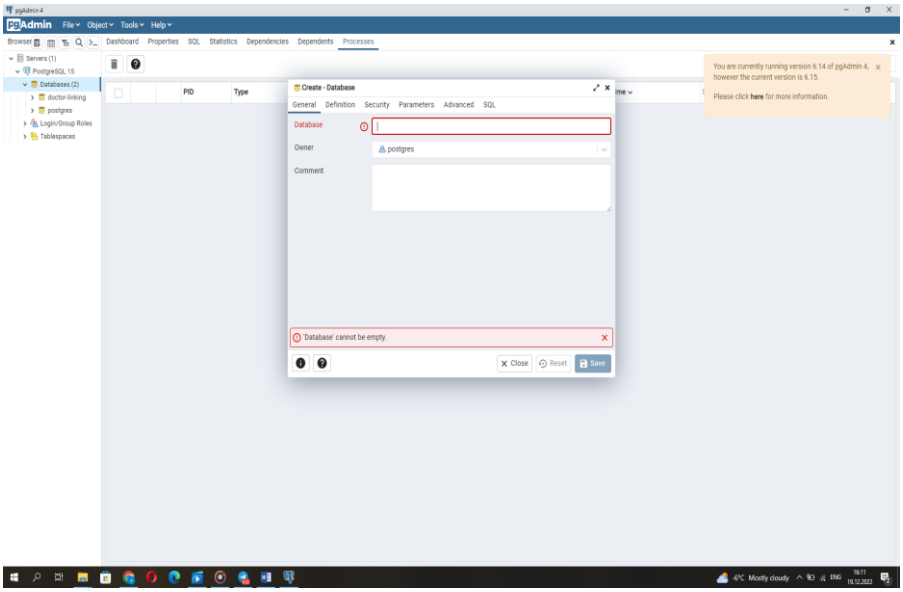


Рисунок 3.2 — Створення бази даних

Після створення бази даних, таблиці для збереження інформації будуть заповнюватись через сервер. Рисунок.3.3.

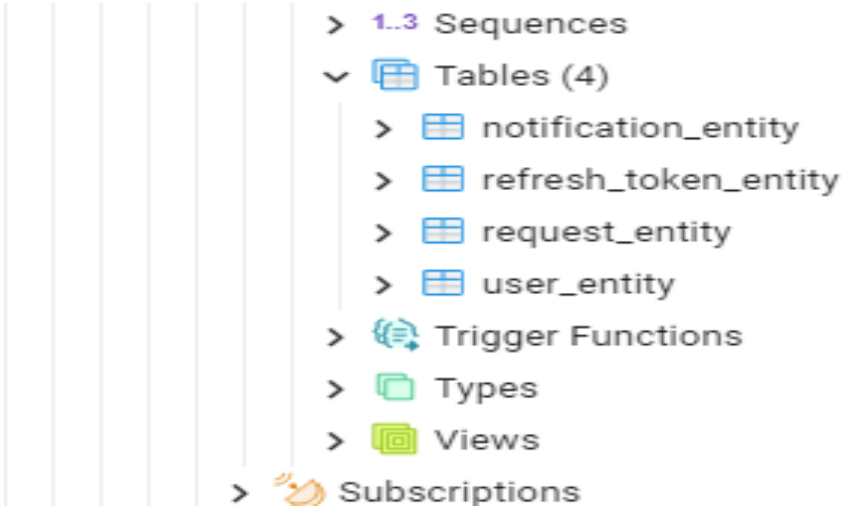


Рисунок 3.3 — Створені таблиці

3.2 Підключення до бази даних за допомогою бібліотеки typeorm

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 ВикористовуєтьсявTypeScripttaJavaScript (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, слабко 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 обєкт з налаштуваннями для підключення до PostgreSQL:

```
export const AppDataSource: DataSourceOptions = {
  type: 'postgres',
  host: process.env.POSTGRESQL_HOST,
  port: Number(process.env.POSTGRESQL_PORT),
  username: process.env.POSTGRESQL_USERNAME,
  password: process.env.POSTGRESQL_PASSWORD,
  database: process.env.POSTGRESQL_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 зєднання. Поле port визначає порт на якому запущена база даних. Властивісті username і password визначають імя і пароль які було використано під час реєстрації в PostgreSQL. Властивість database визначає імя таблиці яку було створено в PostgreSQL. Поле synchronize дозволяє впевнетись в тому, що сутності синхронізуватимуться з базою даних щоразу, коли програму буде запущено. Властивіть logging дозволяє ввімкнути логування всіх запитів і помилок, просто встановивши logging: true у параметрах джерела даних, також за допомогою додаткових парметрів, 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 запитами 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).

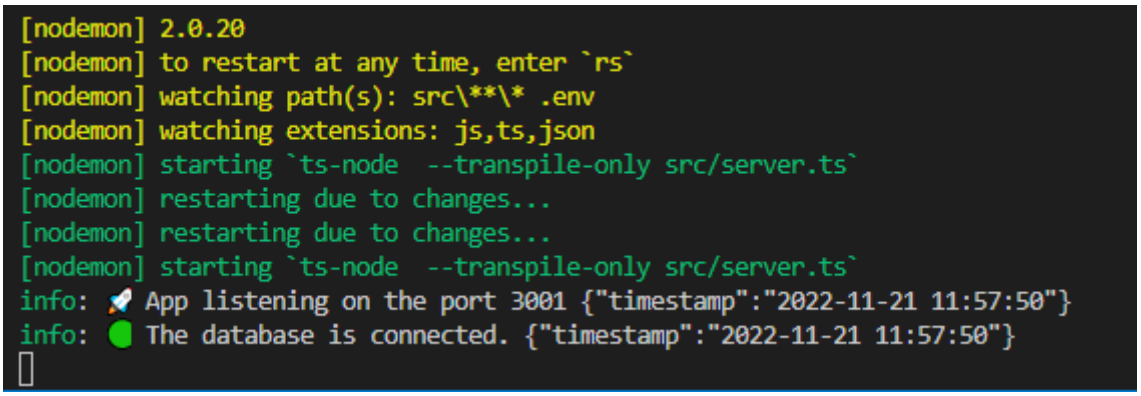


Рисунок 3.4 — Підключення до бази даних

3.3 Створення сутностей

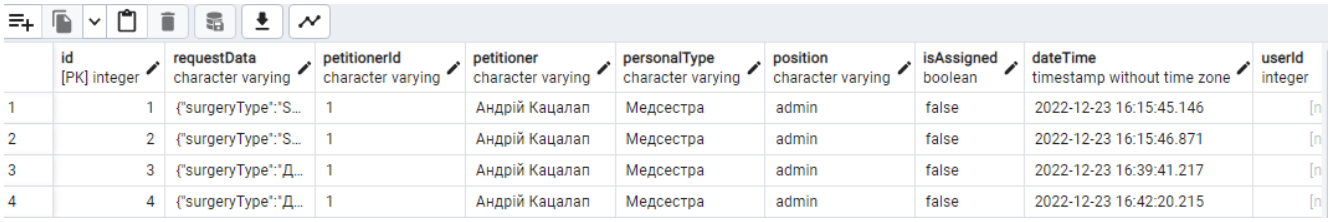
Сутності — це клас, який відображається на таблицю бази даних (або колекцію при використанні MongoDB). Створити сутність можна визначивши новий клас і позначивши його за допомогою @Entity():

```
@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. Рисунок 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	[n
2	2	{"surgeryType":"S...	1	Андрій Кацалап	Медсестра	admin	false	2022-12-23 16:15:46.871	[n
3	3	{"surgeryType":"Д...	1	Андрій Кацалап	Медсестра	admin	false	2022-12-23 16:39:41.217	[n
4	4	{"surgeryType":"Д...	1	Андрій Кацалап	Медсестра	admin	false	2022-12-23 16:42:20.215	[n

Рисунок 3.5 — Створена таблиця в дазі 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"],
```

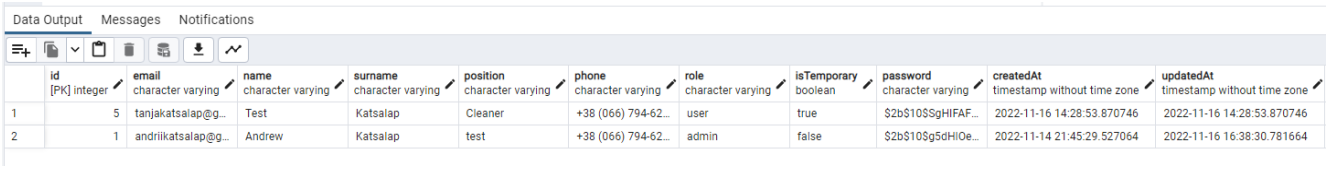
Для прикладу буде представлено сутність для користувача :

```
@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



Data Output | Messages | Notifications

	id [PK] integer	email character varying	name character varying	surname character varying	position character varying	phone character varying	role character varying	isTemporary boolean	password character varying	createdAt timestamp without time zone	updatedAt timestamp without time zone
1	5	tanjakatsalap@g...	Test	Katsalap	Cleaner	+38 (066) 794-62...	user	true	$2b$10$SgHIFAF...	2022-11-16 14:28:53.870746	2022-11-16 14:28:53.870746
2	1	andriikatsalap@g...	Andrew	Katsalap	test	+38 (066) 794-62...	admin	false	$2b$10$g5dHlOe...	2022-11-14 21:45:29.527064	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.

3.4 Створення зв'язків між сутностями

Зв’язки допомагають легко працювати з пов’язаними об’єктами. Існує кілька видів відносин:

— один до одного за допомогою @OneToOne;

— багато-до-одного за допомогою @ManyToOne;

— один до багатьох за допомогою @OneToMany;

— багато-до-багатьох за допомогою @ManyToMany,

Є кілька параметрів, які можна вказати для налаштування відносин:

— eager: boolean;

— cascade: boolean | ("insert" | "update")[] ;

— nullable: boolean;

— orphanedRowAction: "nullify" | "delete" | "soft-delete" | disable.

Ager — якщо встановлено значення true, відношення завжди завантажуватиметься з основною сутністю під час використання методів find* або QueryBuilder для цієї сутності.

Cascade: boolean | ("insert" | "update")[] — якщо встановлено значення true, відповідний об'єкт буде вставлено та оновлено в базі даних. Ви також можете вказати масив параметрів каскаду.

Nullable: boolean — вказує, чи є стовпець цього відношення nullable чи ні. За замовчуванням він нульовий.

OrphanedRowAction: "nullify" | "delete" | "soft-delete" | disable — коли батьківський елемент зберігається (увімкнено каскад) без дочірнього/дітей, які все ще існують у базі даних, це контролюватиме, що з ними станеться. delete видалить цих дітей із бази даних. soft-delete позначає дітей як soft-deleted. nullify видалить ключ відношення. disable збереже відношення незмінним. Щоб видалити, потрібно використовувати власне сховище.

@JoinColumn не лише визначає, яка сторона зв’язку містить об’єднаний стовпець із зовнішнім ключем, але також дозволяє налаштувати ім’я об’єднаного стовпця та назву стовпця, на який посилається.

Коли ми встановлюємо @JoinColumn, він автоматично створює стовпець у базі даних під назвою propertyName + referencedColumnName. Наприклад:

```
@ManyToOne(type => Category)
@JoinColumn({ name: "cat_id" })
category: Category;
```

Зв’язки між сутностями будуть продемонстровані на прикладі таблиць user_entity та refresh_token_entity. В сутності user_entity є колонка яка називається @OneToOne вона ініціалізує зв'язок один до одного який буде зв’язувати її та таблицю refresh_token_entity.

```
@OneToOne(()=> RefreshTokenEntity)
@JoinColumn()
 refreshToken: RefreshTokenEntity
```

Відповідно в таблиці refresh_token_entity встановлені зв'язки для user_entity.

```
@OneToOne( () => UserEntity )
  @JoinColumn()
  user: UserEntity;
```

Для того щоб встановити зв'язки також необхідно під час збереження сутності user_entity передати в його параметр refreshToken раніше створену та збережену сутність refresh_token_entity.

```
const tokens = this.createToken({ email, id });
    const savedToken = await this.saveToken(user.id, tokens.refreshToken);
    await userRepository.update({
      id: id
    }, {
      refreshToken: savedToken
    })
```

Після цього, після збереження user_entity в нього з'явиться поле (рис. 3.7).

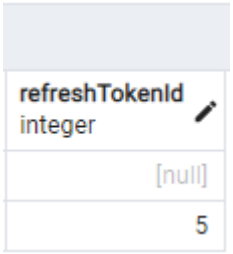


Рисунок 3.7 — Команди для ініціалізації проекту

3.5 Програмна реалізація клієнтської частини

Для ініціалізації React проекту необхідно виконати одну з команд (рис 3.8)

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

Рисунок 3.8 — Команди для ініціалізації проекту

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.

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.

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 допускатимуть 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 багато 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.

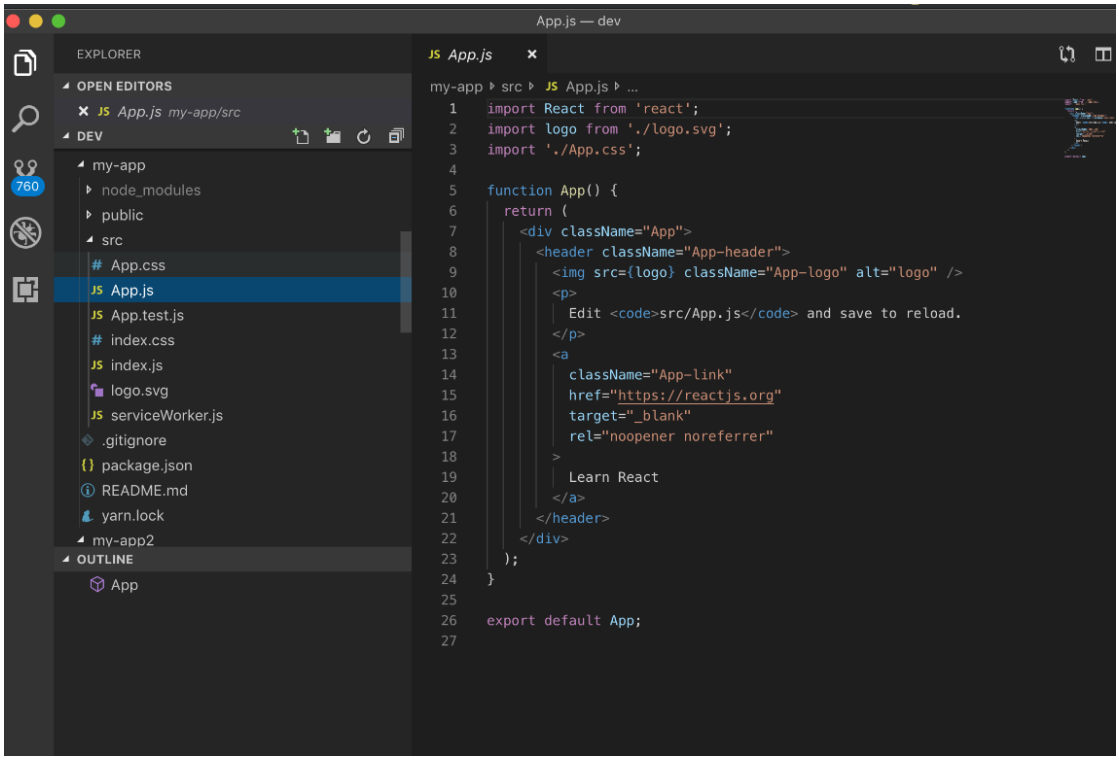


Рисунок 3.9 — Базова структура React проекту

3.5 Створення глобального стейту, підключення бібліотеки redux

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. 3.10).

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.

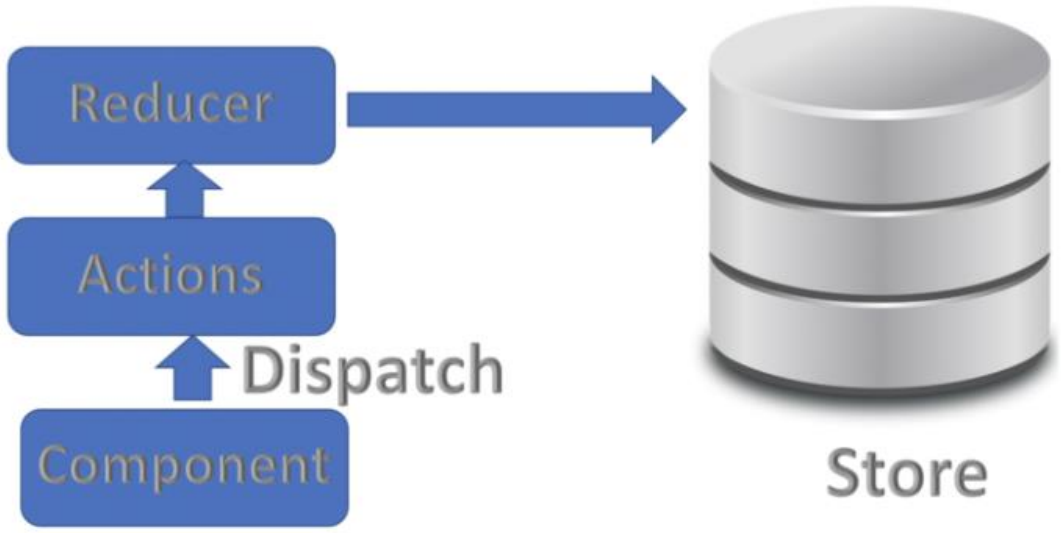


Рисунок 3.10 — Базова структура React проекту

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. 3.11). 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 відмити що в додатку може буде 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 3.11 — ініціалізація сховища даних

В файлі userReducer.js, було створено редуктор, для керування сховищем за допомогою заздалегідь прописаних випадків (рис. 3.12).

Рисунок 3.12 — 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. 3.13).

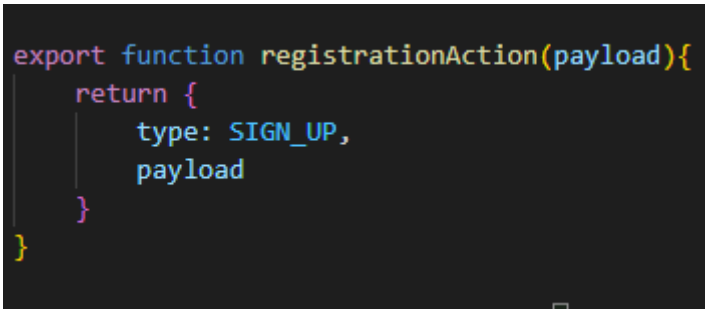


Рисунок 3.13 — Типовий вигляд функції action

Метод dispatch отримується за допомогою хуку useDispatch який імортується, з бібілотеки «react-redux», він дозволяє викликати редуктор і передати в нього всі необхідні дані (рис. 3.14).

```
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 3.14 — Виклик функції dispatch

3.6 Реалізація реєстрації користувача на клієнстькій частині

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. 3.13), в якій адміністратор зможе керувати користувачами.

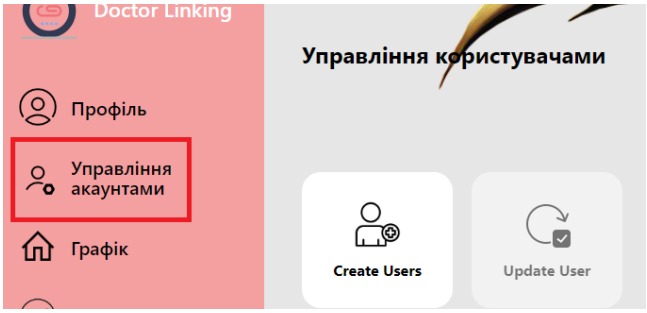


Рисунок 3.13 — Сторінка управління акаунтами

Сторінка «управління акаунтами», необхідна для адміністрування всіх кейсів пов’язаних зі змінами в персоналі, чи їх персональних даних (рис. 3.14). Вона буде доступною тільки в тому випадку, якщо користувач зареєстрований як адміністратор, і всі наявні функції управління будуть виконуватись виключно в цьому ж випадку.

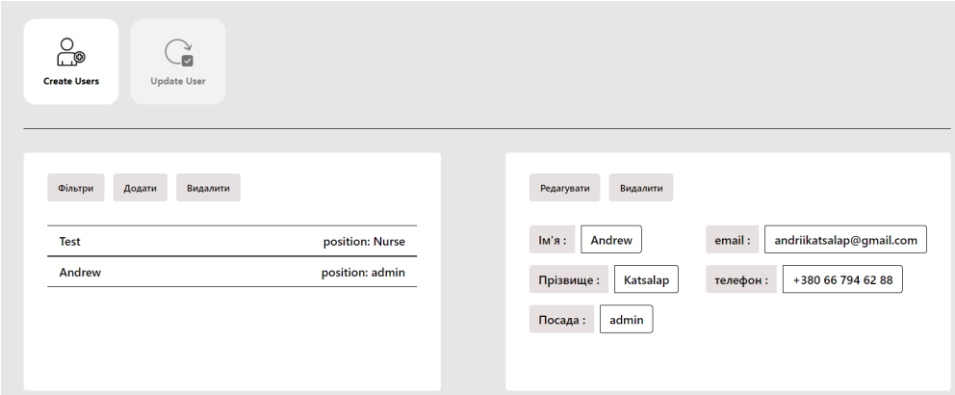


Рисунок 3.14 — Сторінка «управління акаунтами»

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. 3.15).

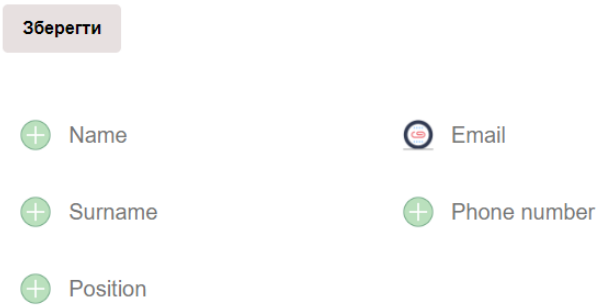


Рисунок 3.15 — Форма для створення користувача

Після заповнення інформації на клієнтській частині, на сервер буде відправлено запит з заповненими даними, після чого на сервері буде збережено новоствореного користувача (рис. 3.16).

	id [PK] integer	email character varying	name character varying	surname character varying	position character varying	phone character varying	role character varying	isTemporary boolean	password character varying	createdAt timestamp without time zone	updatedAt timestamp without time zone	refreshTokenId integer	doctorRequestId integer
1	1	andriikatsalap@g...	Andrew	Katsalap	admin	+380 66 794 62 88	admin	false	$2b$10$e5ZzM...	2022-12-06 16:43:32.534754	2022-12-06 17:14:52.810955	1	[null]
2	2	tanjakatsalap@g...	Test	Katsalap	Nurse	+38 (066) 794-62...	user	false	$2b$10$MlyCUn...	2022-12-06 16:44:45.177384	2022-12-06 16:48:02.27305	2	[null]

Рисунок 3.16 — 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 «thunkDoctorSignUp» 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.

3.7 Налаштування push повідомлень.

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. 3.17).

— 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.

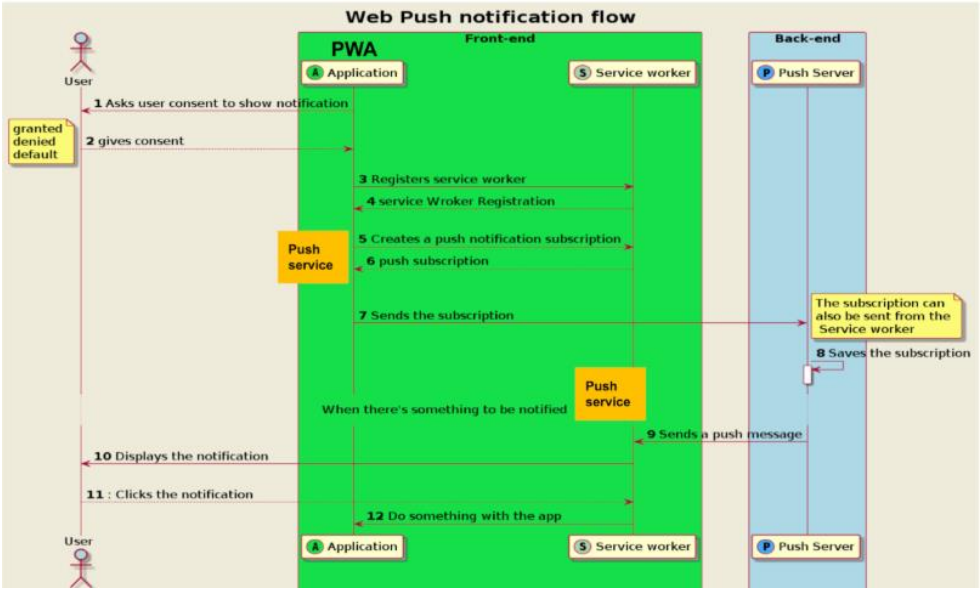


Рисунок 3.17 — Обмін даних для відправки push-повідомлень

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.

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 виконайти 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.3.18).

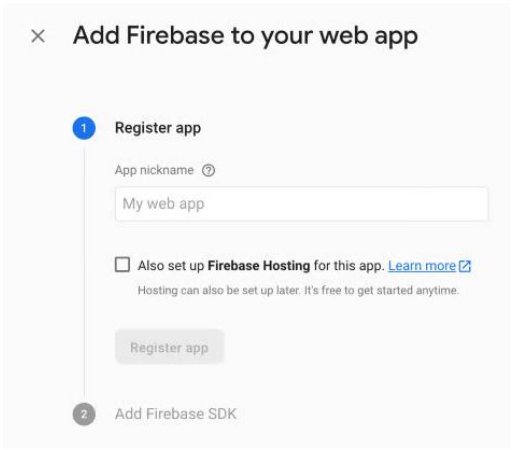


Рисунок 3.18 — Реєстрація додатку в firebase

Після цього, на екрані згенерується конфігурація, яку буде необхідно інтегрувати в застосунок React. Так має виглядати кінцева згенерована конфігурація, (рис.3.19.).

Об’єкт firebaseConfig буде інтегровано в застосунок React, який зв’яже його з цим проектом Firebase.

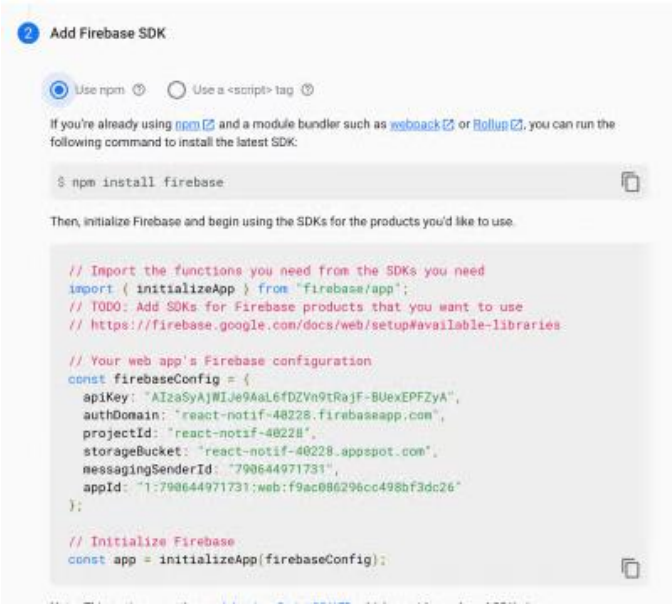


Рисунок 3.19 — Конфігурація створеного додатку в firebase

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. У розділі Сертифікати Web Push потрібно натиснути «Створити пару ключів» (рис.3.20).

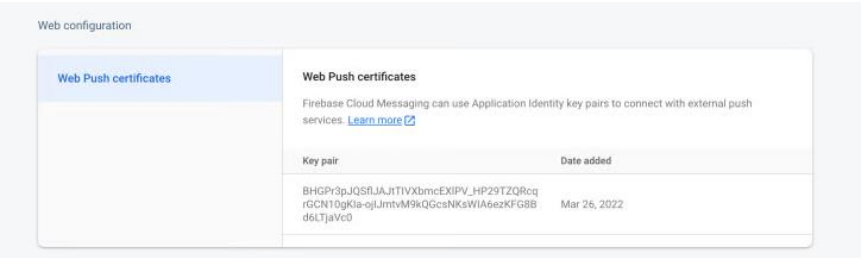


Рисунок 3.20 — Створений ключ, для обміну повідомленнями

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. 3.21).

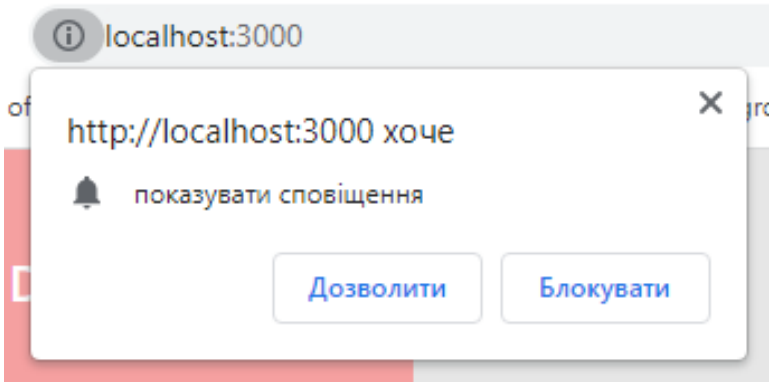


Рисунок 3.21 — Запит на відправку сповіщень

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 посилання 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 3.22). 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.

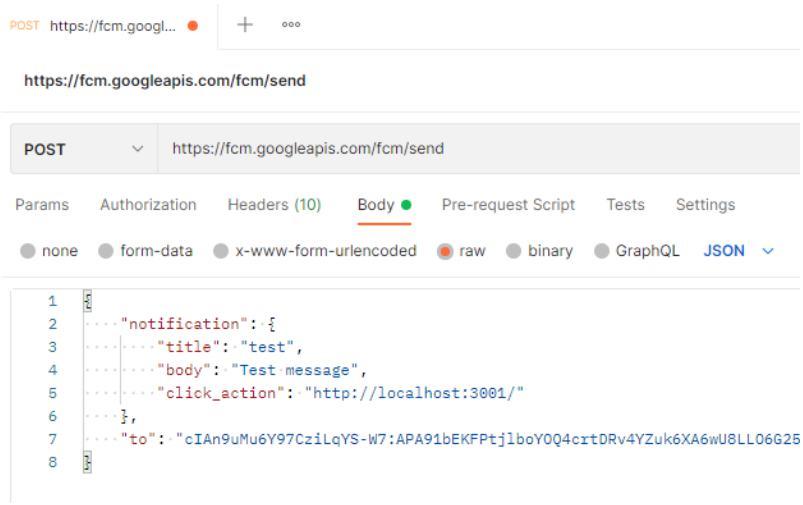


Рисунок 3.22 – Тестування відправки нагадування

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 сформувати 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. 3.23).

Рисунок 3.23 — Результат тестування надсилання сповіщень

В результаті проведення тестування, було отримано сповіщення.

3.6 Програмна реалізація серверної частини додатку

Для ініціалізації серверу було використано модуль «typescript-express-starter» (рис. 3.24).

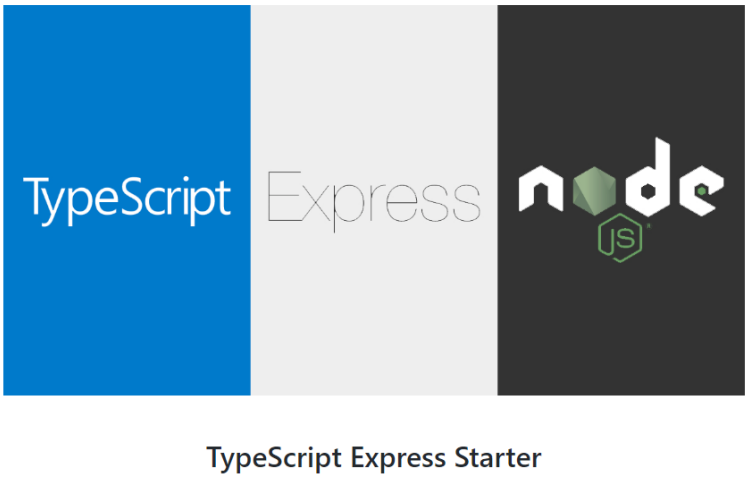


Рисунок 3.24 — Модуль «typescript-express-starter»

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;
— контроллери;
— 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. 3.25).

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

Рисунок 3.25 — 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 інсталюючи цей пакет.

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

Під час встановлення бібліотека запитиає 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;

— Knex: 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 лінтера: npm run lint or Lint typescript-express-starter в VS Code;

— виправлення помилок лінтеру: npm run lint:fix or Lint:Fix typescript-express-starter в VS Code.

Розглянемо структуру створеного проекту, на ведену на рис. 3.26. Основними елементами апі є services, routes, controllers, database, tests, middlewares. Розглянемо їх:

— services містять основну логіку виконання маршрутів;

— routes зберігає всі маршрути які присутні в апі;

— 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 апі;

— 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. 3.27).

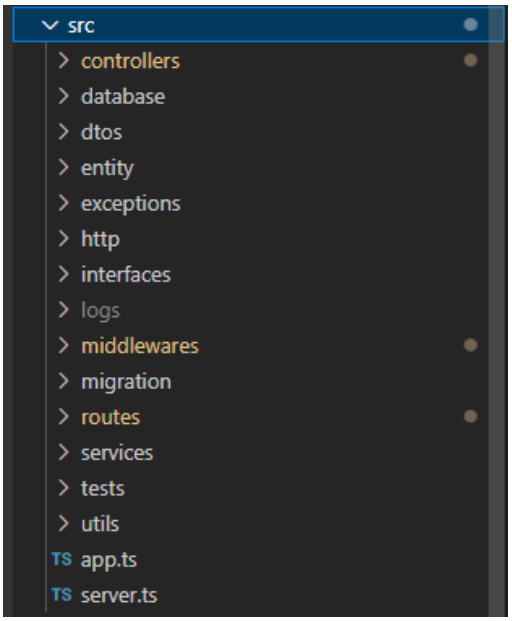


Рисунок 3.26 — Структура проекту

```
"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": "^6.0.0",
  "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 3.27 — 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.

3.8 Розробка серверних маршрутів та логіка роботи додатку

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 поводиться 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(' ')[12]
    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 обєкту 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.

3.9 Створення запитів до бази даних

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 вибіркових запитів: 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" }.

4 ТЕСТУВАННЯ

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.

С початку розглянемо процес реєстрації користувача. Для того щоб зареєструватись користувач повинен заповнити всі поля, та пройти капчу (рис. 4.1).

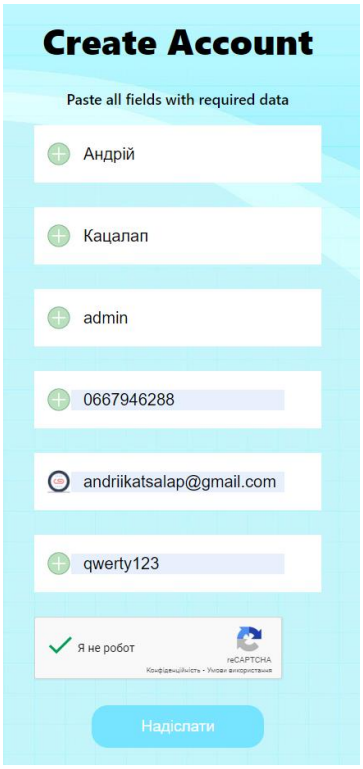
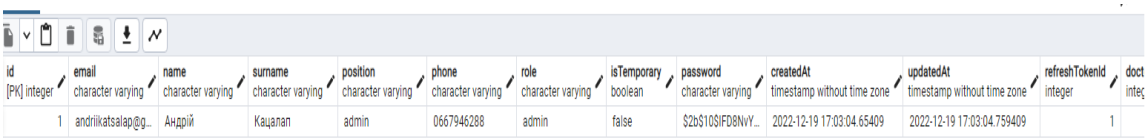


Рисунок 4.1 — Форма реєстрації

Після підтвердження реєстрації користувача перенаправить на головну сторінку додатку, а в базі даних створиться відповідна таблиця, з інформацією які ввів користувач (рис.4.2).



id [PK] integer	email character varying	name character varying	surname character varying	position character varying	phone character varying	role character varying	isTemporary boolean	password character varying	createdAt timestamp without time zone	updatedAt timestamp without time zone	refreshTokenId integer	doct integ
1	andriikatsalap@g...	Андрій	Кацалап	admin	0667946288	admin	false	$2b$10$IFD8NvY...	2022-12-19 17:03:04.65409	2022-12-19 17:03:04.759409	1	

Рисунок 4.2 — 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. 4.3).

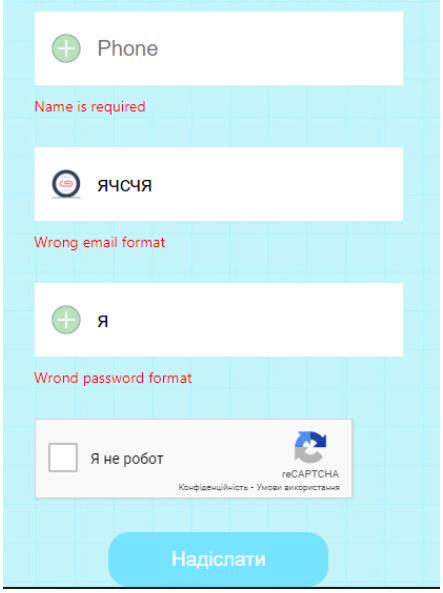


Рисунок 4.3 — Спроба введення не валідних даних

Під час автентифікації користувача необхідно вказати логін та пароль та пройти капчу, в результаті вірно ведених даних користувача буде перенаправлено на головну сторінку (рис. 4.4).

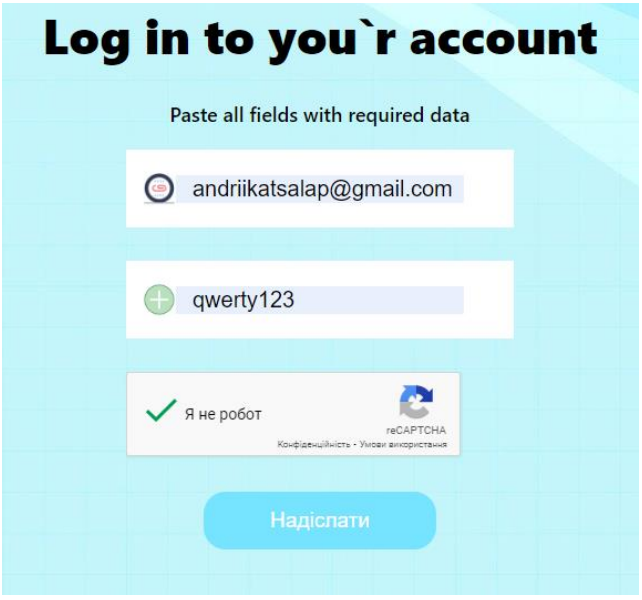


Рисунок 4.4 — Авторизація користувача

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. 4.5), з тимчасовим паролем, його потрібно буде ввести під час автентифікації, але даний пароль є тимчасовим, і його необхідно буде замінити.

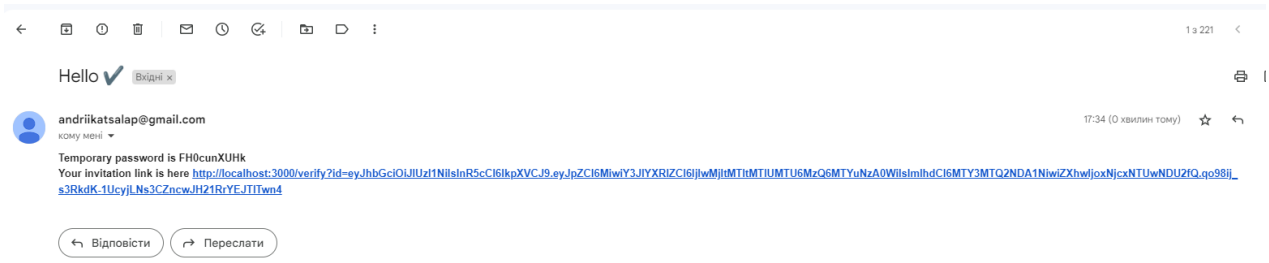


Рисунок 4.5 — Лист запрошення з тимчасовим паролем

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.

Примітка. 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 знущань, бібліотеці тверджень і запуску тестів, Jest діє як повноцінна платформа для тестування, яка дозволяє вам писати

тести для бібліотечних проектів 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 використовується 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.4.6.

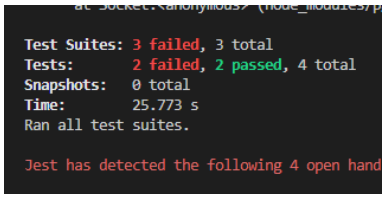


Рисунок 4.5 — Результат виконання тестування

5 ЕКОНОМІЧНА ЧАСТИНА

5.1 Комерційний та технологічний аудит науково-технічної розробки

Метою даного розділу є проведення технологічного аудиту, в даному випадку нового програмного продукту медичної інформаційної система з використанням сучасних технологій проектування. Дана система підвищить ефективність взаємодії медичного персоналу на всіх ланках медичної допомоги. Особливістю програми є те, що при розробці було використано сучасний підхід проектування PWA (progressive web application) що дозволило одночасно використовувати застосунок як: веб-сайт, нативний застосунок, та мобільний застосунок (IOS/Android).

Приблизним аналогом може бути інформаційна система Clinica Web, проте в цілому вона не вирішує ті задачі, які вирішує запропонована розробка. Ціни коливаються від 1200 грн/міс для обслуговування 10 осіб, т.б. 2760 грн/рік на обслуговування 1 особи, до 2300 грн/міс для обслуговування 5 осіб, т.б. 2880 грн/рік на обслуговування 1 особи.

Для проведення комерційного та технологічного аудиту залучають не менше 3-х незалежних експертів. Оцінювання науково-технічного рівня розробки та її комерційного потенціалу рекомендується здійснювати із застосуванням п’ятибальної системи оцінювання за дванадцятьма критеріями, у відповідності із табл. 5.1.

Таблиця 5.1 – Рекомендовані критерії оцінювання комерційного потенціалу розробки та їх можлива бальна оцінка

Бали (за 5-ти бальною шкалою)					
Крит.	0	1	2	3	4
Технічна здійсненність концепції					
1	Достовірність концепції не підтверджена	Концепція підтверджена експертними висновками	Концепція підтверджена розрахунками	Концепція перевірена на практиці	Перевірено роботоздат-ність продукту в реальних умовах

Продовження табл. 5.1

Ринкові переваги					
2	Багато аналогів на малому ринку	Ринкові п Мало аналогів на малому ринку	Кілька аналогів на великому ринку	Один аналог на великому ринку	Продукт не має аналогів на великому ринку
3	Ціна продукту значно вища за ціни аналогів	Ціна продукту дещо вища за ціни аналогів	Ціна продукту приблизно дорівнює цінам аналогів	Ціна продукту дещо нижче за ціни аналогів	Ціна продукту значно нижче за ціни аналогів
4	Технічні та споживчі властивості продукту значно гірші, ніж в аналогів	Технічні та споживчі властивості продукту трохи гірші, ніж в аналогів	Технічні та споживчі властивості продукту на рівні аналогів	Технічні та споживчі властивості продукту трохи кращі, ніж в аналогів	Технічні та споживчі властивості продукту значно кращі, ніж в аналогів
5	Експлуатаційні витрати значно вищі, ніж в аналогів	Експлуатаційні витрати дещо вищі, ніж в аналогів	Експлуатаційні витрати на рівні експлуатаційних витрат аналогів	Експлуатаційні витрати трохи нижчі, ніж в аналогів	Експлуатаційні витрати значно нижчі, ніж в аналогів
Ринкові перспективи					
6	Ринок малий і не має позитивної динаміки	Ринок малий, але має позитивну динаміку	Середній ринок з позитивною динамікою	Великий стабільний ринок	Великий ринок з позитивною динамікою
7	Активна конкуренція великих компаній на ринку	Активна конкуренція	Помірна конкуренція	Незначна конкуренція	Конкурентів немає
Практик на здійсненність					
8	Відсутні фахівці як з технічної, так і з комерційної реалізації ідеї	Необхідно наймати фахівців або витрачати значні кошти та час на навчання наявних фахівців	Необхідне незначне навчання фахівців та збільшення їх штату	Необхідне незначне навчання фахівців	Є фахівці з питань як з технічної, так із комерційної реалізації ідеї
9	Потрібні значні фінансові ресурси, які відсутні. Джерела фінансування ідеї відсутні	Потрібні незначні фінансові ресурси. Джерела фінансування відсутні	Потрібні значні фінансові ресурси. Джерела фінансування є	Потрібні незначні фінансові ресурси. Джерела фінансування є	Не потребує додаткового фінансування
10	Необхідна розробка нових матеріалів	Потрібні матеріали, що використовуються у військово-промисловому комплексі	Потрібні дорогі матеріали	Потрібні досяжні та дешеві матеріали	Всі матеріали для реалізації ідеї відомі та давно використовуються у виробництві

Продовження табл. 5.1

11	Термін реалізації ідеї більший за 10 років	Термін реалізації ідеї більший за 5 років. Термін окупності інвестицій більше 10-ти років	Термін реалізації ідеї від 3-х до 5-ти років. Термін окупності інвестицій більше 5-ти років	Термін реалізації ідеї менше 3-х років. Термін окупності інвестицій від 3-х до 5-ти років	Термін реалізації ідеї менше 3-х років. Термін окупності інвестицій менше 3-х років
12	Необхідна розробка регламентних документів та отримання великої кількості дозвільних документів на виробництво та реалізацію продукту	Необхідно отримання великої кількості дозвільних документів на виробництво та реалізацію продукту, що вимагає значних коштів та часу	Процедура отримання дозвільних документів для виробництва та реалізації продукту вимагає незначних коштів та часу	Необхідно тільки повідомлення відповідним органам про виробництво та реалізацію продукту	Відсутні будь-які регламентні обмеження на виробництво та реалізацію продукту

Усі дані по кожному параметру занесено в таблиці 5.2

Таблиця 5.2 – Результати оцінювання комерційного потенціалу розробки

Критерії оцінювання	ПІБ експертів		
	Експерт 1	Експерт 2	Експерт 3
	Бали		
Технічна здійсненність концепції	3	3	4
Наявність аналогів на ринку	3	3	4
Цінова політика	4	4	3
Технічні та споживчі властивості виробу	4	3	4
Експлуатаційні витрати	3	4	3
Ринок збуту	4	3	4
Конкурентоспроможність	3	4	3
Фахівці з технічної і комерційної реалізації	4	3	4
Фінансування	4	4	3
Матеріально-технічна база	4	4	4
Термін реалізації ідеї	4	4	4
Супровідна документація	3	3	4
Сума	43	42	44
Середньоарифметична сума балів	(43+42+44) / 3 = 43		

За даними таблиці 5.2 можна зробити висновок щодо рівня комерційного потенціалу даної розробки. Для цього доцільно скористатись рекомендаціями, наведеними в таблиці 5.3.

Таблиця 5.3 - Рівні комерційного потенціалу розробки

Середньоарифметична сума балів СБ , розрахована на основі висновків експертів	Рівень комерційного потенціалу розробки
0-10	Низький
11-20	Нижче середнього
21-30	Середній
31-40	Вище середнього
41-48	Високий

Як видно з таблиці, рівень комерційного потенціалу розроблюваного нового програмного продукту є високим, що досягається за рахунок того, що програмний продукт відрізняється від існуючих тим, що дана технологія є медичною інформаційною системою з використанням сучасних технологій проектування. Дана система підвищить ефективність взаємодії медичного персоналу на всіх ланках медичної допомоги. Особливістю програми є те, що при розробці було використано сучасний підхід проектування PWA (progressive web application) що дозволило одночасно використовувати застосунок як: веб-сайт, нативний застосунок, та мобільний застосунок (IOS/Android).

5.2 Прогнозування витрат на виконання науково-дослідної (дослідно-конструкторської) роботи

5.2.1 Основна заробітна плата розробників, яка розраховується за формулою:

$$З_о = \frac{M}{T_р} \cdot t, \quad (5.1)$$

де M – місячний посадовий оклад конкретного розробника (дослідника), грн.;

$T_р$ – число робочих днів в місяці, 22 днів;

t – число днів роботи розробника (дослідника).

Результати розрахунків зведемо до таблиці 5.4.

Таблиця 5.4 – Основна заробітна плата розробників

Найменування посади	Місячний посадовий оклад, грн.	Оплата за робочий день, грн.	Число днів роботи	Витрати на заробітну плату, грн.
Керівник проекту	38000	1727,27	44	76000,000
Програміст	35000	1590,91	44	70000,000
Всього				146000,00

Так як в даному випадку розробляється програмний продукт, то розробник виступає одночасно і основним робітником, і тестувальником розроблюваного програмного продукту.

5.2.2 Додаткова заробітна плата розробників, які приймали участь в розробці обладнання.

Додаткова заробітна плата прийнято розраховувати як 15 % від основної заробітної плати розробників та робітників:

$$З_д = З_о \cdot 15\ \% / 100\ \%\ , \qquad (5.2)$$

$$З_д = (146000{,}00 \cdot 15\ \% / 100\ \%\ ) = 21900{,}00\ \text{(грн.)}$$

4.2.3 Нарахування на заробітну плату розробників.

Згідно діючого законодавства нарахування на заробітну плату складають 22 % від суми основної та додаткової заробітної плати.

$$Н_з = (З_о + З_д) \cdot 22\ \% / 100\%\ , \qquad (5.3)$$

$$Н_з = (146000{,}00 + 21900{,}00) \cdot 22\ \% / 100\ \% = 36938{,}00\ \text{(грн.)}$$

5.2.4. Оскільки для розроблювального пристрою не потрібно витрачати матеріали та комплектуючи, то витрати на матеріали і комплектуючі дорівнюють нулю.

5.2.5 Амортизація обладнання, яке використовувалось для проведення розробки.

Амортизація обладнання, що використовувалось для розробки в спрощеному вигляді амортизація обладнання, що використовувалась для розробки розраховується за формулою:

$$A = \frac{Ц}{Тв} \cdot \frac{t_{вик}}{12} \text{ [грн.].} \quad (5.4)$$

де Ц — балансова вартість обладнання, грн.;

Т — термін корисного використання обладнання згідно податкового законодавства, років

$t_{вик}$ – термін використання під час розробки, місяців

Розрахуємо, для прикладу, амортизаційні витрати на комп'ютер балансова вартість якого становить 32000 грн., термін його корисного використання згідно податкового законодавства — 2 роки, а термін його фактичного використання – 2,00 міс.

$$A_{обл} = \frac{32000}{2} \times \frac{2}{12} = 2666{,}67 \ грн.$$

Аналогічно визначаємо амортизаційні витрати на інше обладнання та приміщення. Розрахунки заносимо до таблиці 4.2. Так як вартість ліцензійної ОС та спеціалізованих ліцензійних нематеріальних ресурсів менше 20000 грн,. (Windows 10, Firebase) то даний нематеріальний актив не амортизується, а його вартість включається у вартість розробки повністю, $В_{нем.ак.}$ = 1183 грн.

5.2.6 Тарифи на електроенергію для непобутових споживачів (промислових підприємств) відрізняються від тарифів на електроенергію для населення. При цьому тарифи на розподіл електроенергії у різних постачальників (енергорозподільних компаній), будуть різними. Крім того, розмір тарифу залежить від класу напруги (1-й або 2-й клас). Тарифи на розподіл електроенергії для всіх енергорозподільних компаній встановлює Національна комісія з

регулювання енергетики і комунальних послуг (НКРЕКП). Витрати на силову електроенергію розраховуються за формулою:

Таблиця 5.5 — Амортизаційні відрахування матеріальних і нематеріальних ресурсів для розробників.

Найменування обладнання	Балансова вартість, грн.	Строк корисного використання, років	Термін використання обладнання, місяців	Амортиза-ційні відрахуван-ня, грн.
Комп'ютер та комп'ютерна периферія (HP ZBook)	32000	2	2,00	2666,667
Офісне обладнення (меблі)	22000	4	2,00	916,667
Приміщення	850000	20	2,00	7083,333
Всього				10666,67

$$В_е = В \cdot П \cdot Ф \cdot К_п, \quad (5.6)$$

де В – вартість 1 кВт-години електроенергії для 1 класу підприємства, В = 6,2 грн./кВт;

П – встановлена потужність обладнання, кВт. П = 0,35 кВт;

Ф – фактична кількість годин роботи обладнання, годин.

$К_п$ – коефіцієнт використання потужності, $К_п$ = 0,9.

$$В_е = 0,9 \cdot 0,35 \cdot 8 \cdot 44 \cdot 6,2 = 687,456 \text{ (грн.)}$$

5.2.7 Інші витрати та загальновиробничі витрати.

До статті «Інші витрати» належать витрати, які не знайшли відображення у зазначених статтях витрат і можуть бути віднесені безпосередньо на собівартість досліджень за прямими ознаками. Витрати за статтею «Інші витрати» розраховуються як 50…100% від суми основної заробітної плати дослідників:

$$I_в = (З_о + З_р) \cdot \frac{Н_{ів}}{100\%}, \quad (5.7)$$

де $Н_{ів}$ – норма нарахування за статтею «Інші витрати».

$$І_в = 146000,00 * 80\% / 100\% = 116800 \text{ (грн.)}$$

До статті «Накладні (загальновиробничі) витрати» належать: витрати, пов'язані з управлінням організацією; витрати на винахідництво та раціоналізацію; витрати на підготовку (перепідготовку) та навчання кадрів; витрати, пов'язані з набором робочої сили; витрати на оплату послуг банків; витрати, пов'язані з освоєнням виробництва продукції; витрати на науково-технічну інформацію та рекламу та ін. Витрати за статтею «Накладні (загальновиробничі) витрати» розраховуються як 100…150% від суми основної заробітної плати дослідників:

$$Н_{нзв} = (З_о + З_р) \cdot \frac{Н_{нзв}}{100\%}, \quad (5.8)$$

де $Н_{нзв}$ – норма нарахування за статтею «Накладні (загальновиробничі) витрати».

$$Н_{нзв} = 146000,00 * 123 \% / 100 \% = 179580 \text{ (грн.)}$$

5.2.9 Витрати на проведення науково-дослідної роботи.

Сума всіх попередніх статей витрат дає загальні витрати на проведення науково-дослідної роботи:

$$В_{заг} = 146000,00+21900,00+36938,00+10666,67+1183+687,46+116800+$$
$$+179580 = 513755,12 \text{ грн.}$$

5.2.11 Розрахунок загальних витрат на науково-дослідну (науково-технічну) роботу та оформлення її результатів.

Загальні витрати на завершення науково-дослідної (науково-технічної) роботи та оформлення її результатів розраховуються *ЗВ*, визначається за формулою:

$$ЗВ = \frac{B_{заг}}{\eta} \text{ (грн)}, \quad (5.9)$$

де η – коефіцієнт, який характеризує етап (стадію) виконання науково-дослідної роботи.

Так, якщо науково-технічна розробка знаходиться на стадії: науково-дослідних робіт, то η=0,1; технічного проектування, то η=0,2; розробки конструкторської документації, то η=0,3; розробки технологій, то η=0,4; розробки дослідного зразка, то η=0,5; розробки промислового зразка, то η=0,7; впровадження, то η=0,9. Оберемо η = 0,5, так як розробка, на даний момент, знаходиться на стадії дослідного зразка:

$$ЗВ = 513755{,}12 / 0{,}5 = 1027510 \text{ грн.}$$

5.3 Розрахунок економічної ефективності науково-технічної розробки за її можливої комерціалізації потенційним інвестором

В ринкових умовах узагальнювальним позитивним результатом, що його може отримати потенційний інвестор від можливого впровадження результатів тієї чи іншої науково-технічної розробки, є збільшення у потенційного інвестора величини чистого прибутку. Саме зростання чистого прибутку забезпечить потенційному інвестору надходження додаткових коштів, дозволить покращити фінансові результати його діяльності, підвищить конкурентоспроможність та може позитивно вплинути на ухвалення рішення щодо комерціалізації цієї розробки.

Для того, щоб розрахувати можливе зростання чистого прибутку у потенційного інвестора від можливого впровадження науково-технічної розробки необхідно:

— вказати, з якого часу можуть бути впроваджені результати науково-технічної розробки;

— зазначити, протягом скількох років після впровадження цієї науково-технічної розробки очікуються основні позитивні результати для потенційного інвестора (наприклад, протягом 3-х років після її впровадження);

— кількісно оцінити величину існуючого та майбутнього попиту на цю або аналогічні чи подібні науково-технічні розробки та назвати основних суб'єктів (зацікавлених осіб) цього попиту;

— визначити ціну реалізації на ринку науково-технічних розробок з аналогічними чи подібними функціями.

При розрахунку економічної ефективності потрібно обов'язково враховувати зміну вартості грошей у часі, оскільки від вкладення інвестицій до отримання прибутку минає чимало часу. При оцінюванні ефективності інноваційних проектів передбачається розрахунок таких важливих показників:

— абсолютного економічного ефекту (чистого дисконтованого доходу);

— внутрішньої економічної дохідності (внутрішньої норми дохідності);

— терміну окупності (дисконтованого терміну окупності).

Аналізуючи напрямки проведення науково-технічних розробок, розрахунок економічної ефективності науково-технічної розробки за її можливої комерціалізації потенційним інвестором можна об'єднати, враховуючи визначені ситуації з відповідними умовами.

5.3.1 Розробка чи суттєве вдосконалення програмного засобу (програмного забезпечення, програмного продукту) для використання масовим споживачем.

В цьому випадку майбутній економічний ефект буде формуватися на основі таких даних:

$$\Delta П_i = (\pm \Delta Ц_0 \cdot N + Ц_0 \cdot \Delta N)_i \cdot \lambda \cdot \rho \cdot (1 - \frac{\vartheta}{100}), \quad (4.10)$$

де $\pm \Delta Ц_o$ — зміна вартості програмного продукту (зростання чи зниження) від впровадження результатів науково-технічної розробки в аналізовані періоди часу;

N — кількість споживачів які використовували аналогічний продукт у році до впровадження результатів нової науково-технічної розробки;

$Ц_o$ — основний оціночний показник, який визначає діяльність підприємства у даному році після впровадження результатів наукової розробки, $Ц_o = Ц_б \pm \Delta Ц_o$;

$Цб$ — вартість програмного продукту у році до впровадження результатів розробки;

ΔN — збільшення кількості споживачів продукту, в аналізовані періоди часу, від покращення його певних характеристик;

λ — коефіцієнт, який враховує сплату податку на додану вартість. Ставка податку на додану вартість дорівнює 20%, а коефіцієнт $\lambda = 0{,}8333$.

p — коефіцієнт, який враховує рентабельність продукту;

ϑ — ставка податку на прибуток, у 2022 році ϑ =18%.

Припустимо, що при прогнозованій ціні 1100 грн. за одиницю виробу, термін збільшення прибутку складе 3 роки. Після завершення розробки і її вдосконалення, можна буде підняти її ціну на *100* грн. Кількість одиниць реалізованої продукції також збільшиться: протягом першого року – на 20000 шт., протягом другого року — на 25000 шт., протягом третього року на 30000 шт. Всього в Україні ЕСОЗ зареєстровано 272 264 лікарів та медичного персоналу [1]. До моменту впровадження результатів наукової розробки реалізації продукту не було.

$\Delta П_1$ = (0*100 + (1100 + 100)*20000)* 0,8333* 0,3) * (1 - 0,18) = 4509999,8 грн;

$\Delta П_2$ = (0*100 + (1100 + 100)*(20000+25000)* 0,8333* 0,3) * (1 - 0,18) = 11069999,6 грн;

$\Delta П_3$=(0*100+(1100+100)*(20000+25000+30000)* 0,8333* 0,3) * (1 - 0,18) = 18449999,3 грн.

Отже, комерційний ефект від реалізації результатів розробки за три роки складе 34029998,64 грн.

5.3.2 Розрахунок ефективності вкладених інвестицій та періоду їх окупності.

Розраховуємо приведену вартість збільшення всіх чистих прибутків *ПП*, що їх може отримати потенційний інвестор від можливого впровадження та комерціалізації науково-технічної розробки:

$$ПП = \sum_{1}^{T} \frac{\Delta П_i}{(1+\tau)^t}, \quad (5.11)$$

де $\Delta П_i$ – збільшення чистого прибутку у кожному із років, протягом яких виявляються результати виконаної та впровадженої науково-дослідної (науково-технічної) роботи, грн;

Т — період часу, протягом якою виявляються результати впровадженої науково-дослідної (науково-технічної) роботи, роки;

τ – ставка дисконтування, за яку можна взяти щорічний прогнозований рівень інфляції в країні, $\tau = 0,05...0,15$;

t — період часу (в роках).

Збільшення прибутку ми отримаємо починаючи з першого року:

ПП= (4509999,820/(1+0,1)[1])+(11069999,557/(1+0,1)[2])+(18449999,262/(1+0,1)[3]) = 4099999,84 + 9148759,965 + 13861757,52 = 27110517,32 грн.

Далі розраховують величину початкових інвестицій *PV*, які потенційний інвестор має вкласти для впровадження і комерціалізації науково-технічної розробки. Для цього можна використати формулу:

$$PV = k_{інв} * ЗВ, \quad (5.12)$$

де $k_{інв}$ — коефіцієнт, що враховує витрати інвестора на впровадження науково-технічної розробки та її комерціалізацію. Це можуть бути витрати на підготовку приміщень, розробку технологій, навчання персоналу, маркетингові заходи тощо; зазвичай $k_{інв}=2...5$, але може бути і більшим;

ЗВ — загальні витрати на проведення науково-технічної розробки та оформлення її результатів, грн.

$$PV = 2 * 1027510 = 2055020{,}49 \text{ грн.}$$

Тоді абсолютний економічний ефект $E_{абс}$ або чистий приведений дохід (NPV, Net Present Value) для потенційного інвестора від можливого впровадження та комерціалізації науково-технічної розробки становитиме:

$$E_{абс} = ПП - PV, \tag{5.13}$$

$$E_{абс} = 27110517{,}32 - 2055020{,}49 = 25055496{,}83 \text{ грн.}$$

Оскільки $E_{абс} > 0$ то вкладання коштів на виконання та впровадження результатів даної науково-дослідної (науково-технічної) роботи може бути доцільним.

Для остаточного прийняття рішення з цього питання необхідно розрахувати внутрішню економічну дохідність або показник внутрішньої но-рми дохідності (IRR, Internal Rate of Return) вкладених інвестицій та порівняти її з так званою бар'єрною ставкою дисконтування, яка визначає ту мінімальну внутрішню економічну дохідність, нижче якої інвестиції в будь-яку науково-технічну розробку вкладати буде економічно недоцільно.

Розрахуємо відносну (щорічну) ефективність вкладених в наукову розробку інвестицій $E_в$. Для цього використаємо формулу:

$$E_в = \sqrt[T_ж]{1 + \frac{E_{абс}}{PV}} - 1, \quad (5.14)$$

$T_ж$ – життєвий цикл наукової розробки, роки.

$E_в = \sqrt[3]{(1 + 25055496,83/2055020,49} - 1 = 1,363$

Визначимо мінімальну ставку дисконтування, яка у загальному вигляді визначається за формулою:

$$\tau = d + f, \quad (5.15)$$

де d — середньозважена ставка за депозитними операціями в комерційних банках; в 2022 році в Україні d = (0,09…0,14);

f – показник, що характеризує ризикованість вкладень; зазвичай, величина f = (0,05...0,5).

$$\tau_{\min} = 0,14 + 0,05 = 0,19.$$

Так як $E_в > \tau_{min}$, то інвестор може бути зацікавлений у фінансуванні даної наукової розробки.

Розрахуємо термін окупності вкладених у реалізацію наукового проекту інвестицій за формулою:

$$T_{ок} = \frac{1}{E_в}, \quad (5.16)$$

$T_{ок}$ = 1 / 1,363 = 0,73 р.

Оскільки $T_{ок}$ < 3-х років, а саме термін окупності рівний 0,73 роки, то фінансування даної наукової розробки є доцільним.

Висновки до розділу: економічна частина даної роботи містить розрахунок витрат на розробку нового програмного продукту, сума яких складає 1027510

гривень. Було спрогнозовано орієнтовану величину витрат по кожній з статей витрат. Також розраховано чистий прибуток, який може отримати виробник від реалізації нового технічного рішення, розраховано період окупності витрат для інвестора та економічний ефект при використанні даної розробки. В результаті аналізу розрахунків можна зробити висновок, що розроблений програмний продукт за ціною дешевший за аналог і є висококонкурентоспроможним. Період окупності складе близько 0,73 роки.

ВИСНОВКИ

В результаті виконання дипломної роботи було створено медичну інформаційну систему для медичного персоналу.

Було 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.

Налаштовано ефективну комунікацію між всіма ланками медичного персоналу.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

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-prilozhenija_ preimuxhestva_i_nedostatki .html

3. Розробка Web-додатків [Електронний ресурс] — Режим доступу до ресурсу: http://clashdash.ru/directions/web-applications

4. Що таке Progressive Web App та в чому їх переваги [Електронний ресурс] — Режим доступу до ресурсу: https://timeweb.com/ru/community/articles/chto-takoe-progressive-web-apps

5. Все що необхідно знати про Progressive Web App [Електронний ресурс] — Режим доступу до ресурсу: https://habr.com/ru/company/wrike/blog/481240/

6. Методи життєвого циклу [Електронний ресурс] — Режим доступу до ресурсу: https://dev-gang.ru/article/kak-ponjat-metody-zhiznennogo-cikla-komponenta-v-reactjs-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 программ. —: Фінанси та статистика, 1982. — 114 с.

11. Статистика росту працівників [Електроний ресурс] — Режим доступу до ресурсу: https://www.kmu.gov.ua/news/u-zakladah-shcho-uklali-dogovori-z-nacionalnoyu-sluzhboyu-zdorovya-zrosla-kilkist-medichnih-pracivnikiv-esoz

12. Using Service Workers [Електроний ресурс] — Режим доступу до ресурсу: https://developer.mozilla.org/en-US/docs/Web/API/Service_Worker_API/Using_Service_Workers

13. PostgreSQL [Електроний ресурс] — Режим доступу до ресурсу: https://builtin.com/learn/tech-dictionary/postgresql

14. Single-page application vs. multiple-page application [Електроний ресурс] — Режим доступу до ресурсу: https://medium.com/@NeotericEU/single-page-application-vs-multiple-page-application-2591588efe58

15. ReactJS Lifecycle of Components [Електроний ресурс] — Режим доступу до ресурсу: https://www.geeksforgeeks.org/reactjs-lifecycle-components/

16. What is React [Електроний ресурс] — Режим доступу до ресурсу: https://flatlogic.com/blog/what-is-react/

17. Lifecycle Methods [Електроний ресурс] — Режим доступу до ресурсу: https://www.codecademy.com/learn/react-101/modules/react-102-lifecycle-methods-u/cheatsheet

ДОДАТОК А

Технічне завдання

Міністерство освіти та науки України

Вінницький національний технічний університет

Факультет інформаційних технологій та комп'ютерної інженерії

Кафедра обчислювальної техніки

ЗАТВЕРДЖУЮ

Завідувач кафедри__ОТ__

проф., д.т.н.. Азаров О.Д.

_____

" ____ " _____ 2022 р.

ТЕХНІЧНЕ ЗАВДАННЯ

на виконання магістерської кваліфікаційної роботи

«Інформаційна система приймального відділення лікарні»

08-23.МКР.003.00.000 ТЗ

Науковий керівник: доцент к.т.н.

_____ Тарновський М. Г.

Виконав: студент групи 1КІ-21м

_____ Кацалап А. Ю.

ВНТУ 2022

1 Підстава для виконання магістерської кваліфікаційної роботи (МКР)

1.1 Необхідність забезпечити оперативну комунікацію між міпвробітниками приймального відділення та лікарями.

1.2 Наказ про затвердження теми МКР.

2 Мета МКР і призначення розробки

2.1 Метою магістерської роботи 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 процесс 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;

2.1 Призначення розробки — створення інформаційної системи приймального відділення лікарні.

3 Вихідні дані для виконання МКР

3.1 Створити базу даних Postgre;

3.2 Реалізувати сервер на NodeJS;

3.3 Створити застосунок React PWA;

3.4 Обраний тип застосунку кросплатформений.

4 Вимоги до виконання МКР

4.1 Провести аналіз способів побудови прогресивного додатку;

4.2 Вибрати архітектуру програмного забезпечення інформаційної системи;

4.3 Розробити програмне забезпечення інформаційної системи;

4.4 Оцінити комерційний потенціал розробки.

5 Етапи МКР та очікувані результати

Етапи роботи та очікувані результати приведено в Таблиці А.1.

Таблиця А.1 — Етапи МКР

№ ета-пу	Назва етапу	Термін виконання		Очікувані результати
		початок	кінець	
1	Аналіз способів побудови прогресивного додатку			Вступ Розділ 1
2	Вибір архітектури програмного забезпечення інформаційної системи			Розділ 2 Блок схема алгоритмів
3	Розробка програмного забезпечення інформаційної системи			Розділ 3, Блок схема алгоритмів
4	Оцінка комерційного потенціалу розробки			Розділ 5
5	Оформлення пояснювальної записки, графічного матеріалу і презентації			Поясню-вальна записка, графічний матеріал, презентація

6 Матеріали, що подаються до захисту МКР

До захисту подаються: пояснювальна записка МКР, графічні і ілюстративні матеріали, протокол попереднього захисту МКР на кафедрі, відгук наукового керівника, відгук опонента, протоколи складання державних екзаменів, анотації до МКР українською та іноземною мовами, довідка про відповідність оформлення МКР діючим вимогам.

7 Порядок контролю виконання та захисту МКР

Виконання етапів графічної та розрахункової документації МКР контролюється науковим керівником згідно зі встановленими термінами. Захист МКР відбувається на засіданні Екзаменаційної комісії, затвердженої наказом ректора.

8 Вимоги до оформлювання та порядок виконання МКР

8.1 При оформлюванні МКР використовуються:

— ДСТУ 3008 : 2015 «Звіти в сфері науки і техніки. Структура та правила оформлювання»;

— ДСТУ 8302 : 2015 «Бібліографічні посилання. Загальні положення та правила складання»;

— ГОСТ 2.104-2006 «Единa система конструкторської документації. Основні написи»;

— методичні вказівки. Кафедра обчислювальної техніки 2022;

— документами на які посилаються у вище вказаних.

8.2 Порядок виконання МКР викладено в «Положення про кваліфікаційні роботи на другому (магістерському) рівні вищої освіти СУЯ ВНТУ-03.02.02-П.001.01:21».

ДОДАТОК Б

Схема інформаційної системи

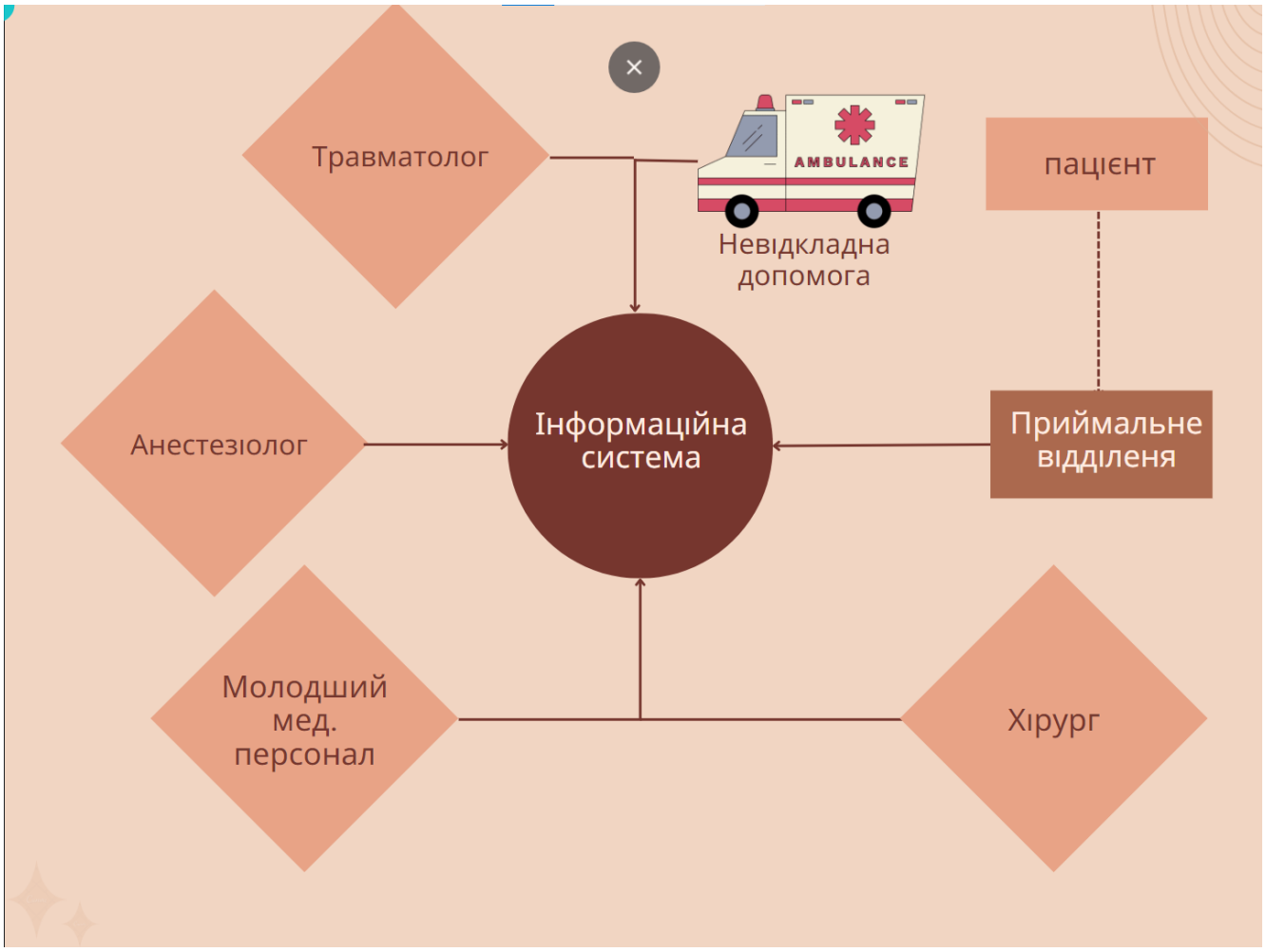


Рисунок Б.1 – Схема інформаційної системи

ДОДАТОК В

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 UsersManagmant 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('/')
      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 Г

Головна сторінка застосунку

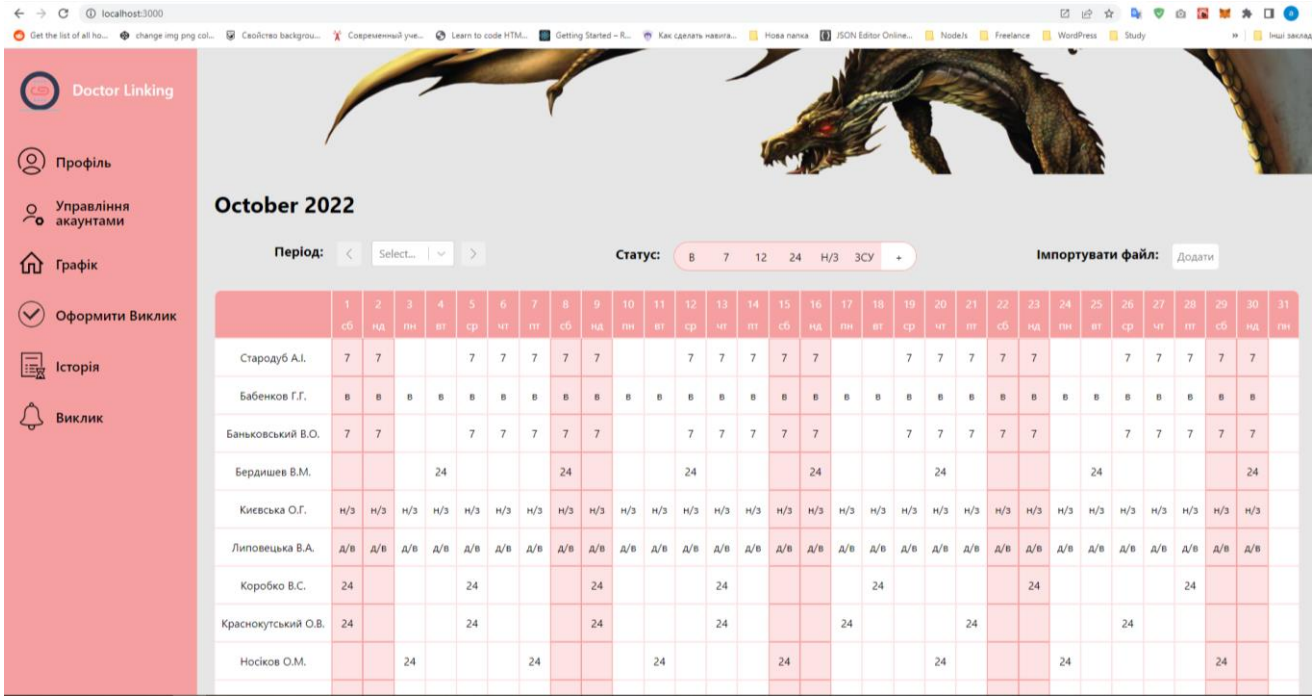


Рисунок Г.1 – Головна сторінка додатку

ДОДАТОК Д

Блок схема входу в систему

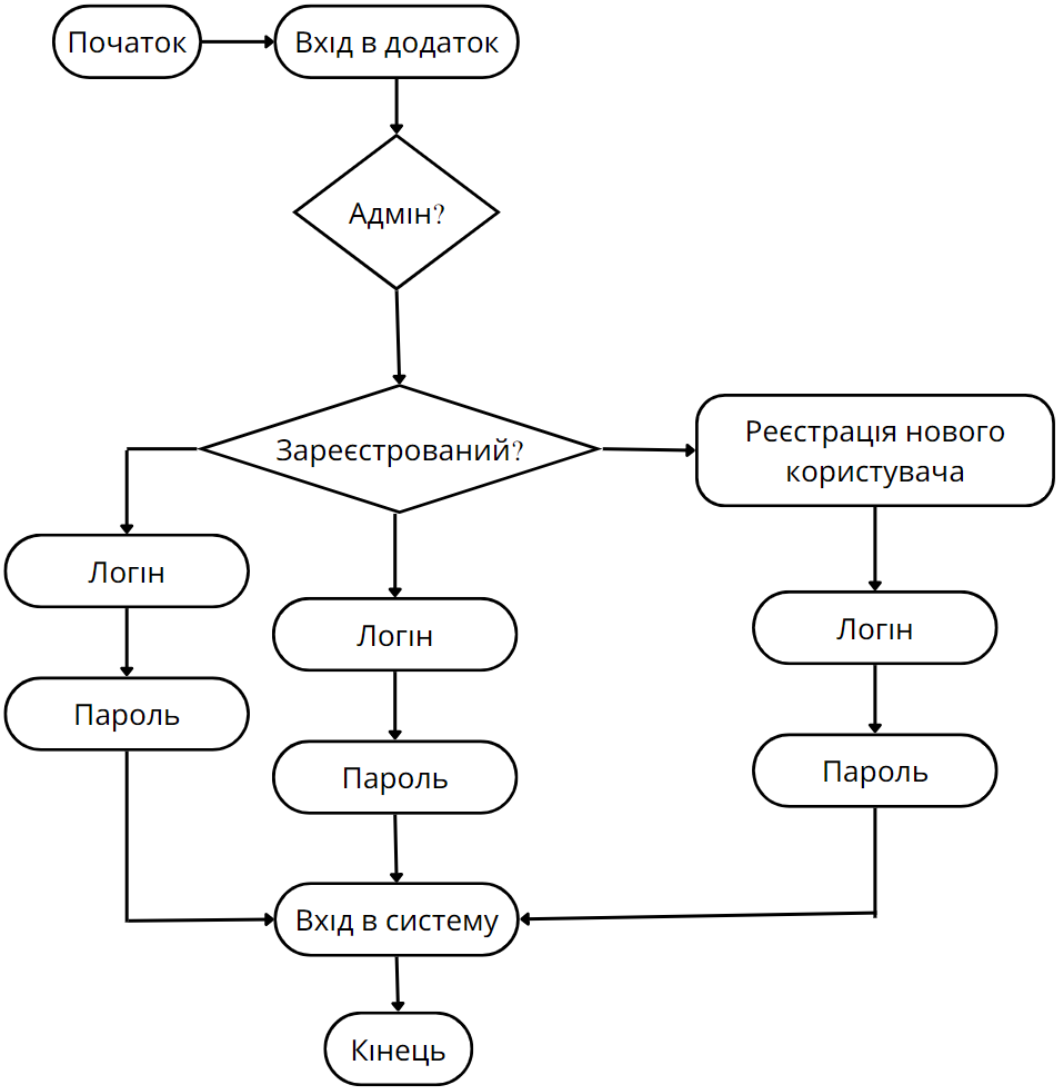


Рисунок Д.1 – Блок схема входу в систему

ДОДАТОК Ж

Блок схема виходу з системи

Рисунок Ж.1 – 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;
```

ДОДАТОК К

Протокол перевірки кваліфікаційної роботи

Назва роботи: Інформаційна система приймального відділення лікарні

Тип роботи: магістерська кваліфікаційна робота
(БДР, МКР)

Підрозділ кафедра обчислювальної техніки
(кафедра, факультет)

Показники звіту подібності Unicheck

Оригінальність 95.6% Схожість 4.4%

Аналіз звіту подібності (відмітити потрібне):

- ✓ Запозичення, виявлення у роботі, оформлені коректно і не містять ознак плагіату.
- □ Виявлені у роботі запозичення не мають ознак плагіату, але їх надмірна кількість викликає сумніви щодо цінності роботи і відсутності самостійності її виконання автором. Роботу направити на розгляд експертної комісії кафедри.
- □ Виявлені у роботі запозичення є недобросовісними і мають ознаки плагіату та/або в ній містяться навмисні спотворення тексту, що вказують на спроби приховування недобросовісних запозичень.

Особа, відповідальна за перевірку _____ Захарченко С.М.
(підпис) (прізвище, ініціали)

Ознайомлені з повним звітом подібності, який був згенерований системою Unicheck щодо роботи.

Автор роботи _____ Кацалап А. Ю.
(підпис) (прізвище, ініціали)

Керівник роботи _____ Тарновський М. Г.
(підпис) (прізвище, ініціали)