МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ІНЖЕНЕРНИЙ НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ

ім. Ю.М. Потебні

ЗАПОРІЗЬКОГО НАЦІОНАЛЬНОГО УНІВЕРСИТЕТУ

КАФЕДРА ЕЛЕКТРОНІКИ, ІНФОРМАЦІЙНИХ СИСТЕМ ТА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Кваліфікаційна робота

перший (бакалаврський)

(рівень вищої освіти)

на тему **Розроблення інтегрованої CRM-системи для автоматизації управління стоматологічною клінікою**

Виконав: студент 4 курсу, групи 6.1212-пзс
спеціальності 121 Інженерія програмного забезпечення
(код і назва спеціальності)
освітньої програми Програмне забезпечення систем
(код і назва освітньої програми)
А.В. Нахман
(ініціали та прізвище)
Керівник доцент А.І. Безверхий
(посада, вчене звання, науковий ступінь, підпис, ініціали та прізвище)
Рецензент директор ТОВ «Діскус»
Р.О. Лютий
(посада, вчене звання, науковий ступінь, підпис, ініціали та прізвище)

Запоріжжя

2026

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ІНЖЕНЕРНИЙ НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ ім. Ю.М. Потебні ЗАПОРІЗЬКОГО НАЦІОНАЛЬНОГО УНІВЕРСИТЕТУ

Кафедра електроніки, інформаційних систем та програмного забезпечення
Рівень вищої освіти перший (бакалаврський)
Спеціальність 121 Інженерія програмного забезпечення
(код та назва)
Освітня програма Програмне забезпечення систем
(код та назва)

ЗАТВЕРДЖУЮ

Завідувач кафедри Тетяна КРИТСЬКА

“ 01 ” березня 2026 року

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

Нахману Артему Володимировичу
(прізвище, ім’я, по батькові)

1. Тема роботи Розроблення інтегрованої CRM-системи для автоматизації управління стоматологічною клінікою
керівник роботи Безверхий Анатолій Ігорович, доцент,
(прізвище, ім’я, по батькові, науковий ступінь, вчене звання)
затверджена наказом ЗНУ від 23.12.2025 р. № 1830-с
2. Строк подання студентом кваліфікаційної роботи 02.06.2026
3. Вихідні дані бакалаврської роботи
   - комплект нормативних документів ;
   - технічне завдання до роботи.

4. Зміст розрахунково - пояснювальної записки (перелік питань, які потрібно розробити)
   - Огляд та збір літератури стосовно теми кваліфікаційної роботи;
   - Огляд та аналіз існуючих рішень та аналогів;
   - Аналіз сучасних технологій для створення Веб-застосунків;
   - Проєктування та розробка Веб-застосунку;

5. Перелік графічного матеріалу (з точним зазначенням обов’язкових креслень) слайдів презентації

6. Консультанти розділів бакалаврської роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата Завдання прийняв

7. Дата видачі завдання 01.03.2026

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів магістерської роботи	Строк виконання етапів магістерської роботи	Примітка
1	Аналіз предметної області	14.03.2026	виконано
2	Формулювання основної задачі дипломної роботи та узгодження її з науковим керівником	20.03.2026	виконано
3	Аналіз існуючих методів рішення	25.03.2026	виконано
4	Огляд та збір літератури стосовно теми кваліфікаційної роботи	05.04.2026	виконано
5	Огляд та аналіз існуючих рішень та аналогів	08.04.2026	виконано
6	Вивчення засобів для розробки Веб-застосунку	12.04.2026	виконано
7	Визначення функціоналу застосунку, що розробляється	20.04.2026	виконано
8	Узгодження подальших дій з науковим керівником	21.04.2026	виконано
9	Проєктування бази даних та створення проєкту застосунку	27.04.2026	виконано
10	Проєктування та конструювання застосунку	11.05.2026	виконано
11	Програмна реалізація застосунку та його апробація	25.05.2026	виконано
12	Оформлення звіту	05.06.2026	виконано
13	Оформлення презентації. Отримання рецензій від опонентів	23.06.2026	виконано

Студент _____ _____А.В. Нахман

Керівник роботи _____ _____А. І. Безверхий

Нормоконтроль пройдено

Нормоконтролер _____ _____І.А. Скрипник

АНОТАЦІЯ

Сторінок - 74

Рисунків - 30

Джерел - 19

Нахман А. В. Розроблення інтегрованої CRM-системи для автоматизації управління стоматологічною клінікою: кваліфікаційна робота бакалавра спеціальності 121 Інженерія програмного забезпечення / наук. керівник А. І. Безверхий. Запоріжжя : ЗНУ, 2026, 74 с.

У сучасному світі, заклади без цифрової системи втрачають як клієнтів, як репутацію, так і гроші на підтримку закладу. Медичні заклади теж не є виключенням. Задля гарної роботи стоматологічної клініки, потрібні різні систем та умови, одна із них, це прозорість клініки. Щоб пацієнт міг дізнатися, хто ним буде займатися, який у нього досвід, як до нього ставляться інші пацієнти. Також пацієнтам важливий контроль над історією, не кожна людина пам’ятає що було пів року тому, і щоб стоматолог виконав свою роботі вірно, він повинен знати певну інформацію про пацієнта.

Мета кваліфікаційної роботи – створення веб-застосунку, який дозволяє пацієнтам переглядати детальну інформацію про лікарів, історію операцій та іншу інформацію, а для працівників зручний інструмент для введення обліку стоматології.

У ході розробки програмного забезпечення, було розглянуто аналогічні застосунки, які активно використовуються. Було виявлено вимоги до нашого застосунку. Як результат розроблено застосунок, який має чітку модульну структуру, яка дозволяє поєднувати frontend та backend частини, інтеграція бази даних, для зберігання обліку стоматології та можливість подальшого розвитку застосунку задля поліпшення або розширення потреб користувачів.

Ключові слова: *стоматологічна клініка*, *веб-додаток*, *frontend*, *backend*, *база даних*, *Nest.js*, *Next.js*, *MySQL*.

ABSTRACT

Pages - 74

Drawings - 30

Sources -19

Nakhman A.V. Development of an integrated CRM-system for automating the management of a dental clinic: qualification work of a bachelor in the specialty 121 “Software Engineering” / scientific supervisor A. I. Bezverkhyi. Zaporizhzhia: ZNU, 74 p.

In the modern world, institutions without a digital system lose customers, reputation, and money to support the institution. Medical institutions are no exception. For a dental clinic to work well, various systems and conditions are needed, one of which is the transparency of the clinic. So that the patient can find out who will be taking care of him, what his experience is, how other patients need control over history, not everyone remembers what happened six months ago, and in order for the dentist to do his job correctly, he must know certain information about the patient.

The goal of this project is to create a web application that allows patients to view detailed information about doctors, surgery history and other information, and for employees a convenient tool for entering dental records. During the development of the application, similar applications that are actively used were considered. Requirements for our application were identified. As a result, an application was developed that has a clear modular structure that allows combining frontend and backend parts, database integration for storing dental records and the possibility of further development of the application to improve or expand user needs.

Keywords: *Dental clinic*, *web application*, *frontend*, *backend*, *database*, *Nest.js*, *Next.js*, *MySQL*

ЗМІСТ

ВСТУП 8
1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ 11
1.1 Сучасний стан стоматологічної галузі 11
1.2 Проблеми управління стоматологічними клініками 11
1.3 CRM-системи у медичній сфері 12
1.4 Технологічні аспекти інтеграції CRM 12
1.5 Аналіз існуючих рішень 12
1.6 Постановка задачі 15
1.7 Висновки з розділу 1 16
2 ОПИС ТА ПРОЄКТУВАННЯ ПРОГРАМНОГО ЗАСТОСУНКУ 18
2.1 Архітектура веб-застосунку 18
2.2 Дизайн інтерфейсу користувачів 19
2.3 Нефункціональні вимоги 21
2.4 Функціональність застосунку 22
2.4.1 Функціональність для пацієнтів 23
2.4.2 Функціональність для лікарів 24
2.4.3 Функціональність для персоналу рецепції 24
2.4.4 Функціональність для адміністратора 24
2.4.5 Додаткові функціональні можливості 25
2.5 База даних 26
2.5.1 MySql 27
2.6 Frontend 29
2.6.1 Стек технологій для Frontend 30
2.6.2 Опис Next.js 34
2.7 Backend 36
2.7.1 Стек технологій для Backend 38
2.7.2 Опис Nest.js 40
2.8 Висновки з розділу 2 42
3 СТРУКТУРА ПРОЄКТУ CPM СТОМАТОЛОГІЧНОЇ КЛІНІКИ 43

3.1 Структура бази даних проєкту 43
3.2 Клієнтська частина проєкту 48
3.3 Серверна частина проєкту 58
3.4 Використання застосунку 66
3.4.1 Функціонал адміністрації 66
3.4.2 Функціонал лікаря 67
3.4.3 Функціонал працівника реєстратури 70
3.5 Висновки з розділу 3 71
ВИСНОВКИ 72
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ 73

ВСТУП

Актуальність теми

Сучасна стоматологічна клініка – це не просто медичний заклад, а комплексний вузькоспеціалізований заклад, який потребує ефективного управління пацієнтами, лікарями, фінансовими потоками та внутрішніми процесами. Зростання конкуренції у сфері медичних послуг, підвищення вимог пацієнтів до якості обслуговування та необхідність оптимізації бізнес-процесів викликають потребу у впровадженні сучасних інформаційних технологій. Одним із ключових інструментів у цьому напрямку є CRM-системи (Customer Relationship Management), які дозволяють автоматизувати процеси взаємодії з пацієнтами, забезпечити прозорість роботи клініки та підвищити ефективність управління.

У стоматологічній практиці CRM-система виконує роль інтеграційної платформи, що об’єднує дані про пацієнтів, історію лікувань, фінансові операції та комунікацію між лікарем і пацієнтом. Це дає змогу не лише оптимізувати роботу персоналу, а й створити комфортні умови для пацієнтів, підвищити рівень довіри та лояльності. Таким чином, інтеграція CRM-системи у стоматологічну клініку є актуальним завданням, що виконує сучасні вимоги до медичної сфери.

Мета дослідження

Метою роботи є розробка проєкту, який дозволяє поліпшити лікування та оптимізувати управлінські процеси у стоматологічній практиці.

Завдання дослідження

Завдання дослідження полягають у тому, щоб проаналізувати сучасний стан стоматологічної галузі, визначити основні проблеми та потреби пацієнтів і лікарів, розробити концепцію стоматологічного проєкту та оцінити ефективність запропонованих рішень

Об’єкт і предмет дослідження

Об’єктом дослідження є стоматологічна клініка, як комплексна система надання медичних послуг.

Предметом дослідження виступають методи та інструменти оптимізації роботи стоматологічних клінік у рамках запропонованого проєкту

Практичне значення

Практичне значення дослідження полягає у тому, щоб вдосконалити управління стоматологічними клініками, зменшити грошові та репутаційні втрати.

Глосарій

CRM (Customer Relationship Management) – система керування взаємовідносинами з клієнтами, яка забезпечує збір, збереження та аналіз інформації про пацієнтів, їхні звернення, історія лікувань та комунікацію з клінікою.

API (Application Programming interface) – набір правил, інструментів та протоколів, що дозволяє різним продуктам взаємодіяти між собою.

Модуль – окрема функціональна складова CRM-системи, яка реалізує конкретне завдання (наприклад, запис пацієнта на прийом)

Автоматизація – застосування програмних рішень для виконання повторюваних операцій без постійного втручання людини, що знижує шанс появи помилок та підвищує ефективність роботи.

Frontend – клієнтська частина веб-додатку, яка відповідає за інтерфейс та взаємодію користувача з системою.

Backend – серверна частина веб-додатку, що забезпечує логіку роботи CRM-системи, обробку даних та інтеграцію з базою даних.

Клієнт (Пацієнт) – особа, яка звертається до стоматологічної клініки для отримання медичних послуг. У CRM-системі клієнт є ключовим об’єктом обліку, що має власні персональні дані, історію лікувань, фінансові записи та комунікацією з клінікою.

Працівник (Співробітник клініки) – особа, яка виконує професійні обов’язки у стоматологічній клініці (лікар, адміністратор, працівники приймальні). У CRM-системі працівники мають власні профілі та доступ до функціоналу відповідно до їхніх ролей.

Роль працівника – набір прав та обов’язків, визначених у CRM-системі для конкретної категорії співробітників (наприклад лікар має доступ до медичних записів).

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Сучасний стан стоматологічної галузі

Стоматологія є однією з галузей медицини, що найбільш динамічно розвиваються. Зростання кількості приватних стоматологічних клінік, підвищення конкуренції та вимог пацієнтів до якості обслуговування зумовлюють необхідність впровадження сучасних інформаційних технологій. Традиційні методи ведення документації та управління пацієнтами (паперові картки, ручне планування прийомів) поступово втрачають актуальність, оскільки вони є трудомісткими, схильними до помилок та не забезпечують належного рівня прозорості й контролю.

1.2 Проблеми управління стоматологічними клініками

Стоматологічні клініки, які не використовують CRM-системи, стикаються з низкою проблем. По-перше, неоптимізований облік пацієнтів призводить до того, що дані зберігаються у різних форматах, що ускладнює швидкий доступ до історії лікувань. По-друге, планування прийомів є складним процесом, оскільки відсутність автоматизованого календаря часто спричиняє накладки у графіку та втрату часу. Третім суттєвим недоліком є фінансові неточності: ручне ведення обліку платежів збільшує ризик помилок та ускладнює контроль доходів і витрат. Крім того, недостатня комунікація з пацієнтами, зокрема відсутність системи нагадувань та зворотного зв’язку, негативно впливає на рівень їхньої лояльності. Нарешті, клініки без CRM-системи не мають інструментів для аналітики, що унеможливлює ефективний аналіз роботи лікарів, завантаженості кабінетів та фінансових показників.

1.3 CRM-системи у медичній сфері

CRM-системи у медичній сфері виконують роль інтеграційних платформ, які дозволяють централізовано зберігати дані про пацієнтів, автоматизувати процес запису на прийом, вести фінансовий облік, забезпечувати комунікацію між лікарем і пацієнтом та формувати аналітичні звіти для керівництва клініки [10] [13]. У стоматології CRM-система має специфічні вимоги: вона повинна інтегруватися з електронними медичними картками, забезпечувати можливість зберігання рентгенівських знімків та результатів діагностики, а також підтримувати багатокористувацький доступ із різними рівнями прав.

1.4 Технологічні аспекти інтеграції CRM

Інтеграція CRM-системи у стоматологічну клініку передбачає використання сучасних веб-технологій [12]. На рівні frontend застосовуються React, Tailwind CSS та Formik, що дозволяють створити зручний та адаптивний інтерфейс користувача. На рівні backend використовуються Node.js та Express, які забезпечують реалізацію бізнес-логіки та взаємодію з базою даних. Для зберігання інформації про пацієнтів, лікарів та фінансові операції застосовуються бази даних PostgreSQL або MongoDB. Інтеграція з іншими системами, такими як бухгалтерські програми чи електронна пошта, здійснюється за допомогою REST API. Крім того, використання хмарних технологій забезпечує доступність CRM-системи з будь-якого пристрою [14] [15].

1.5 Аналіз існуючих рішень

У сучасній стоматологічній практиці ефективне управління пацієнтами, лікарями та фінансовими потоками є ключовим чинником

конкурентоспроможності клініки. Саме тому на ринку з’явилися спеціалізовані CRM-системи, які інтегрують адміністративні, медичні та маркетингові процеси в єдину інформаційну платформу. Серед поширених рішень варто розглянути **Curve Dental** [7] та **Dentrix** [6], які вже зарекомендували себе як провідні системи для стоматологічних клінік.

Curve Dental (див. рис. 1) є хмарною CRM-системою, що дозволяє клінікам працювати без власних серверів та складної IT-інфраструктури. Її функціонал охоплює онлайн-запис пацієнтів, автоматичні нагадування про прийом, цифрові форми для реєстрації. Окрему увагу приділено фінансовим процесам: система забезпечує автоматизоване виставлення рахунків, можливість оплати через мобільні додатки та цифрові гаманці. Завдяки цьому клініки досягають зменшення кількості пропусків прийомів та підвищення ефективності лікувальних планів. Перевагою Curve Dental є її мобільність та масштабованість, що робить систему зручною як для невеликих приватних практик, так і для великих клінік.



Рисунок 1 – *Головна сторінка Curve dental*

Dentrix (див. рис. 2), навпаки, пропонує як локальну, так і хмарну версію (Dentrix Ascend), що дозволяє адаптувати систему до різних умов роботи. Ця система дозволяє інтеграцію зі страховими компаніями, автоматизація перевірки страхових полісів та обробки заявок. Крім того, Dentrix активно використовує інструменти штучного інтелекту для підтримки діагностики та ведення клінічних нотаток. Система також включає маркетингові модулі: онлайн-бронювання, двосторонні повідомлення з пацієнтами, управління репутацією та SEO-оптимізацію. Dentrix існує на ринку вже понад 30 років і використовується у великій кількості клінік, що підтверджує її надійність та функціональну повноту.

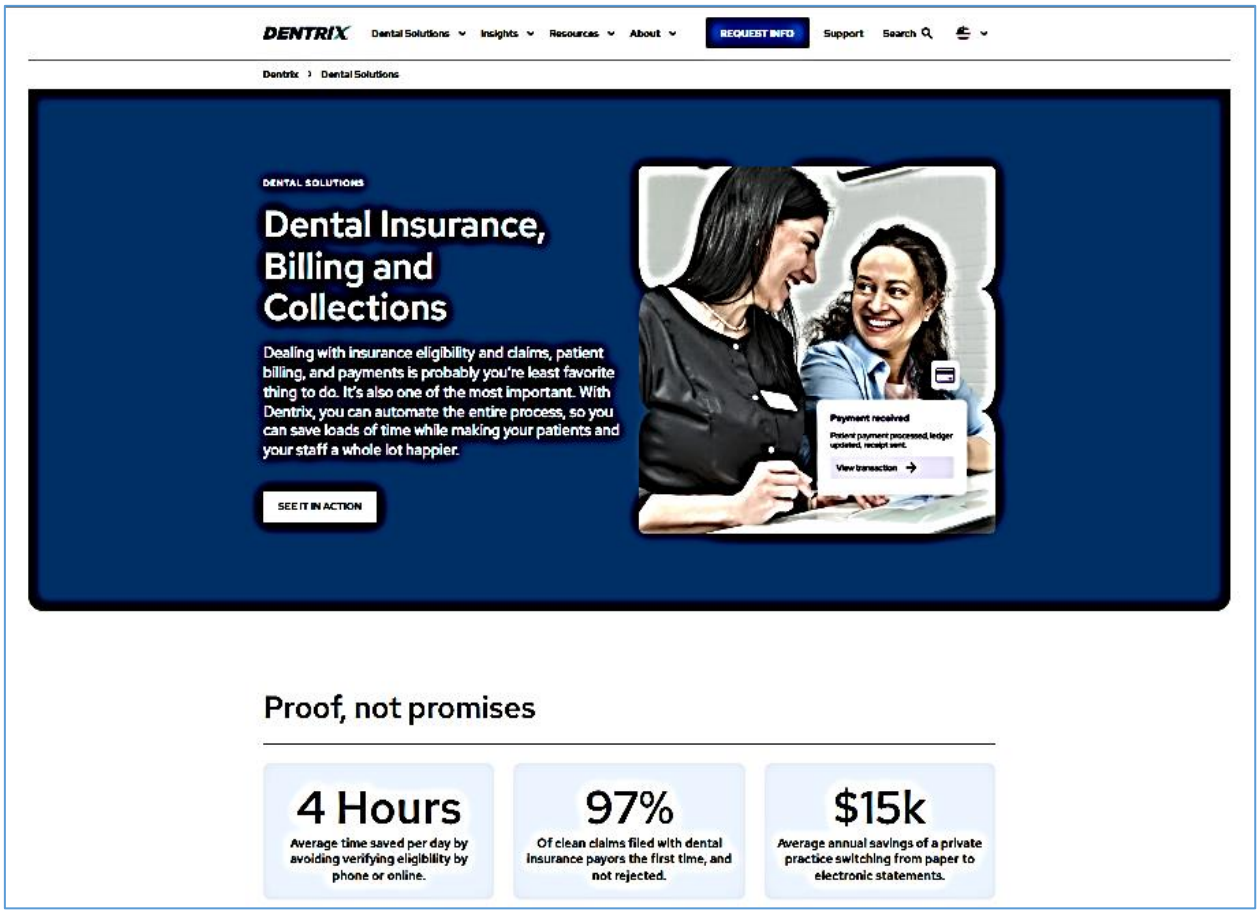


Рисунок 2 – *Головна сторінка Detrix*

Таким чином, аналіз існуючих рішень показує, що сучасні CRM-системи для стоматології, орієнтовані на комплексне управління клінікою. **Curve Dental** орієнтований для нових та середніх клінік, які прагнуть швидко впровадити хмарне рішення без значних інвестицій у технічну інфраструктуру. **Dentrix** переважно використовується у великих клінік та мереж, де важлива інтеграція зі страховими компаніями, багатолокаційна аналітика та розширені маркетингові можливості. Обидва приклади підтверджують актуальність розробки власного застосунку, який має враховувати потреби локального ринку та забезпечувати комплексне управління стоматологічною клінікою.

1.6 Постановка задачі

Метою цієї бакалаврської роботи є створення веб-застосунку для стоматологічної клініки, який допоможе пацієнтам ефективно та зручно планувати записи до лікаря. Основним завданням є створення функціонального та надійного веб-застосунку, що забезпечить користувачам високий рівень зручності, безпеки та довіри під час запису на прийом, отримання консультацій та управління лікувальними процесами [19].

Для реалізації поставленої мети необхідно виконати декілька ключових етапів. Одним із ключових етапів це проаналізувати існуючі рішення та аналоги. Під час огляду важливо відокремити аспекти реалізації, їхні переваги та недоліки, потрібно спиратися на ті аспекти які будуть корисні для нашого проєкту, під час розробки та під час підтримки у майбутньому. Під аспектами можуть виступати як функціональні так і нефункціональні вимоги. Особлива увага до цих вимог являється запорука успішності усього проєкту. Серед базових вимог являється безпека даних, зручність використання.

Після визначення вимог, наступає час проєктування архітектури проєкту. У цій частині дуже важливо передбачити як можна більше можливих сценаріїв та як результат отримати архітектуру яка буде відповідати поточним вимогам та можливим вимогам у майбутньому. Під проєктуванням архітектури мається

на увазі такі дії як: визначення структури проєкту, модулі які будуть використовуватися, розробка інтерфейсу користувача та структура бази даних.

Коли ми вже маємо архітектуру проєкту ми можемо переходити до самої реалізації проєкту. Реалізація проєкту це комплекс дій які потрібно виконати для отримання готового до використання проєкту. Ці дії включають написання коду який відповідає розробленій раніше архітектурі, інтеграція зовнішніх сервісів, таких як надсилання поштових повідомлень, реалізація бази даних відповідно до схеми та архітектурі.

Тестування проєкту нерідко не розглядають як окремий етап, і не завжди її реалізовують, але вона є невід'ємною частиною проєкту. Тестування дозволяє перевірити працездатність проєкту, очікуваний результат від методів та реалізації, виявити помилки ще на ранніх стадіях розробки, поки проєкт на ранніх стадіях. Як результат, тестування може призвести до покращення оптимізації, підвищення продуктивності та виявити проблеми у безпеці. Після усіх попередніх етапів здійснюється аналіз розробленого проєкту. Наскільки він відповідає створеній архітектурі, які у нього є недоліки та переваги, наскільки ці критерії відповідають поставленим задачам. У підсумку, формуються висновки щодо якості проєкту та перспективи подальшого використання.

Ці дії спрямовані на досягнення мети бакалаврської роботи та розробку веб-застосунку для стоматологічної клініки, який враховує сучасні вимоги пацієнтів та забезпечує інструменти для планування та організації стоматологічних послуг.

1.7 Висновки з розділу 1

Аналіз предметної області показав, що у сучасному світі цифровізація має великий вплив на різні сфери життя. Ця ситуація показує що впровадження CRM-системи у стоматологічну клініку є актуальним та необхідним. CRM-система дозволяє вирішити проблеми управління пацієнтами, фінансами та комунікацією, забезпечуючи прозорість роботи та підвищує конкурентоспроможність закладу. У той же час існуючі рішення підтверджують потребу у створенні власного застосунку, адаптованого під специфічні умови роботи клініки

2 ОПИС ТА ПРОЄКТУВАННЯ ПРОГРАМНОГО ЗАСТОСУНКУ

2.1 Архітектура веб-застосунку

При проєктуванні веб-застосунку для стоматологічної клініки важливо обрати архітектурний підхід, який забезпечить масштабованість, безпеку та простоту підтримки [1] [2] [8].

Монолітна архітектура є класичним рішенням, коли всі компоненти інтегровані в єдиний блок. Вона проста у реалізації та забезпечує оперативне розгортання системи, проте з часом виникають труднощі з масштабуванням.

Більш сучасним підходом являється модульна архітектура (layered Architecture), у якій система поділяється на декілька рівнів. Рівні залежать поставлених вимог, самої реалізації, технологій та інших причин, від презентаційного рівня до взаємодії із базою даних. Ця архітектура дозволяє розробнику чітко розуміти структуру та полегшує підтримку. Але при великому функціоналу можуть з’явитися проблеми у масштабуванні.

Варто розглянути архітектуру Domain-Driven Design (DDD) [9], його характерною рисою є формування модулів навколо бізнес-доменів. Такий підхід спрощує розширення функціоналу, але вимагає більш ретельного моделювання домену.

Для поточного проєкту використовується модульна архітектура з елементами DDD та багаторівневим розділенням відповідальності. Це рішення дозволяє:

- Розділення системи на незалежні модулі (Auth, Patients, Doctors, Appointments, Payments, Notifications, Reports), це рішення спрощує розробку та підтримку .
- Забезпечення масштабованості дозволяє додавати нові модулі без змін існуючих.
- Гарантування безпеки використовуючи JWT-аутентифікації з токенами та рольової моделі доступу (RBAC).

- Оптимізація роботи клініки за рахунок автоматичних сповіщень (email, SMS, push) та аналітики продуктивності лікарів і фінансових показників.
- Використання сучасного стеку технологій [16]:
    1. Frontend: Next.js + TypeScript + TailwindCSS;
    2. Backend: Nest.js + TypeOrm;
    3. Database: MySQL з нормалізацією даних (3NF).

На рис. 3 можна побачити схему взаємодії користувача із системою для додавання нового запису до лікаря.

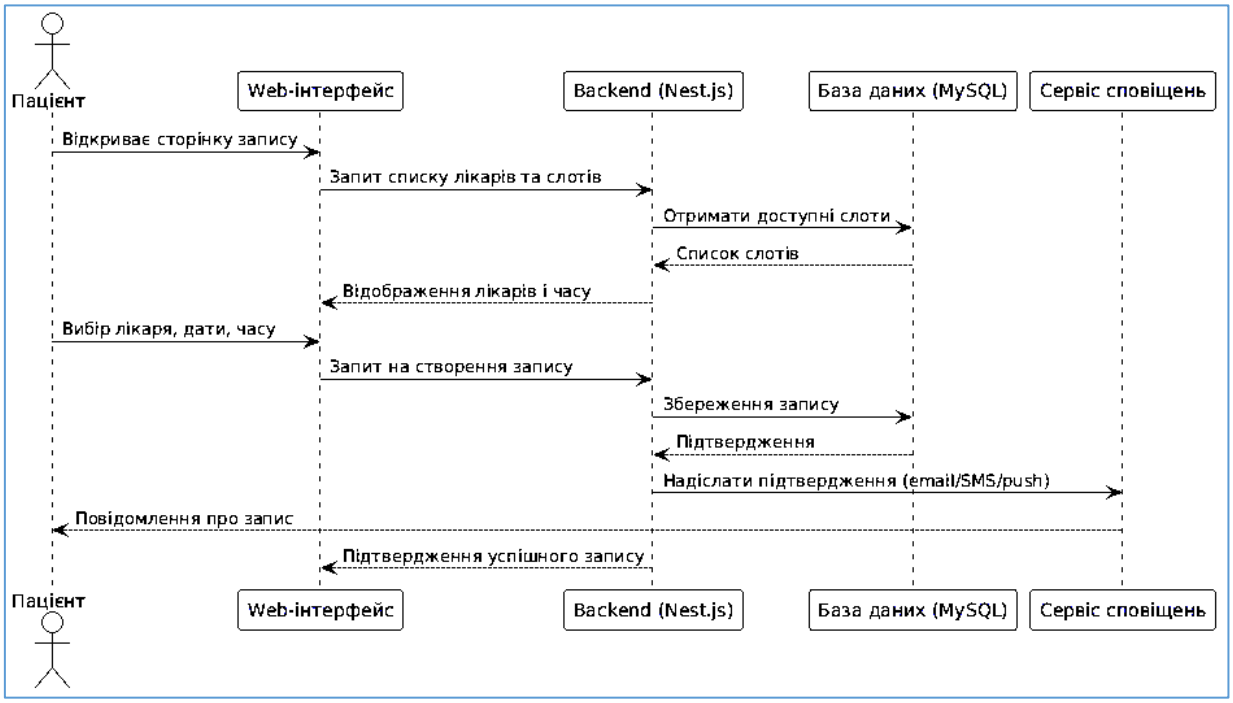


Рисунок 3 – *Запис клієнта до лікаря*

Як результат, архітектура поєднує модульність, гнучкість та усталені шаблони (MVC, SOLID), що забезпечує масштабованість, безпечність та готовність до інтеграції з зовнішніми сервісами, зокрема платіжні системи, відео-консультації, хмарні сервіси [17] [18].

2.2 Дизайн інтерфейсу користувачів

Дизайн інтерфейсу користувачів є важливим компонентом застосунку для стоматологічної клініки, оскільки він визначає спосіб взаємодії пацієнтів, лікарів та адміністративного персоналу із системою та безпосередньо впливає на їхній досвід використання. У цьому розділі розглядається структура та зовнішній вигляд застосунку, які спрямовані на оптимізацію користувацького досвіду та відповідність професійним стандартам.

Структура інтерфейсу користувачів повинна бути логічною та зрозумілою для різних категорій користувачів.

Основні компоненти:

- Головна сторінка (Dashboard): Пацієнт має зможу робити запис на прийом, переглядати історію лікування, інформацію про лікарів та послуги. Лікар може переглядати розклад прийомів та список пацієнтів. Адміністратор може переглядати статистику та керувати клінікою.
- Сторінка запису на прийом: Пацієнт може обрати лікаря, дату та час, переглянути доступні послуги та вартість. Передбачено інтеграцію з системою нагадувань такі як email, SMS та push.
- Сторінка управління пацієнтами: Для персоналу рецепції передбачено можливість додавання, редагування та видалення облікових записів пацієнтів, обробка платежів, формування рахунків.
- Особистий кабінет пацієнта: Пацієнт може переглядати історію лікування, майбутніх записів, управління профілем та контактними даними.
- Адміністративна панель: Управління персоналом, каталогом послуг, цінами, а також доступ до аналітики такі як завантаженість лікарів, фінансові показники, статистика відвідувань.

Інтерфейс має бути сучасним, професійним та відповідати медичній тематиці. Зовнішній вигляд має відповідати вимогам:

- Кольорова палітра: Використання спокійних та довірливих кольорів: білий (чистота), блакитний (медицина, довіра), зелений (здоров'я). Контрастність забезпечує читабельність тексту.
- Використання медичних іконок (зуби, інструменти, календар) для швидкої навігації. Фотографії лікарів та клініки створюють довіру та персоналізують досвід.
- Чіткі та легко читабельні шрифти. Важливі елементи виділяються жирним або кольором, щоб користувач швидко знаходив потрібну інформацію.
- Адаптивність: Інтерфейс оптимізований для різних пристроїв. Використання адаптивного дизайну (CSS Grid, Flexbox, TailwindCSS) забезпечує коректне відображення на будь-якому екрані.

Дизайн інтерфейсу користувача у стоматологічному веб-застосунку характеризується поєднанням професійності, доступності та зрозумілості. Це дозволяє пацієнтам швидко записуватися на прийом, лікарям ефективно управляти розкладом, а адміністраторам контролювати роботу клініки. Використання сучасних практик UI/UX-дизайну спрямоване на формування позитивного досвід та підвищує довіру до системи.

2.3 Нефункціональні вимоги

Нефункціональні вимоги визначають не конкретну реалізацію, а вимоги до якості веб-застосунку для стоматологічної клініки. Вони не описують конкретні функції системи, але встановлюють стандарти, яким проєкт повинен відповідати.

У сучасному світі, повільна робота застосунків негативно впливає на користувачів, що може викликати зниження ефективності роботи клініки, тож наш застосунок повинен забезпечувати належний рівень продуктивності і взаємодіяти зі значною кількістю даних використовуючи REST API. Так як цей

застосунок буде у відкритому доступі він повинен швидко обробляти велику кількість запитів, а це означає що база даних повинна бути оптимізованою, це питання вирішується за допомогою індексів та нормалізації бази даних. З часом з’являються нові системи, пристрої. У перспективі розвитку може виникнути потреба у розширенні функціоналу застосунку. А це означає, що проєкт повинен мати таку архітектуру, яка дозволить легко масштабувати та розширити, включаючи додавання нових повноцінних модулів, таких як хмарне середовище для зберігання фотографій. Важливою умовою сучасних застосунків є конфіденційність та безпека даних, ця тема становиться особливо важливою в останні роки, а це означає що проєкт повинен ще на стадії проєктування підготувати засоби безпеки проти витікання конфіденційних даних поза межі застосунку. Способів зробити це існує дуже багато, від використання хеш-ключів до використання JWT токенів. Існує багато приказок про те, що людина спочатку оцінює усе очима. Цей вислів може стосуватися будь якого проєкту, тому що стилізований інтерфейс може зацікавити пацієнта. Способів виконання такого інтерфейсу існує достатньо багато, від оформлення сторінки до функціоналу який доступний пацієнту (наприклад підтримка багатомовності).

Таким чином, нефункціональні вимоги гарантують, що веб-застосунок буде не лише функціональним, але й відповідатиме високим стандартам продуктивності, безпеки та зручності використання, що є критично важливим для медичних інформаційних систем

2.4 Функціональність застосунку

Функціональні можливості веб-додатку для стоматологічної клініки охоплюють набір інструментів, які забезпечують зручну взаємодію пацієнтів, лікарів, персоналу рецепції та адміністраторів та інших користувачів із системою. Головним завданням є створення інструментів, що автоматизує

ключові бізнес-процеси клініки, підвищить якість обслуговування та зменшить навантаження на персонал.

2.4.1 Функціональність для пацієнтів

На рисунку 3 представлено UserCase діаграму Клієнта, яка включає наступні дії:

- Реєстрація та авторизація через email із підтвердженням особи.
- Онлайн-запис на прийом із вибором лікаря, дати та часу.
- Перегляд історії лікування та призначених процедур.
- Отримання автоматичних нагадувань про прийоми (email, SMS, push).
- Перегляд вартості послуг та інформації про лікарів.
- Управління особистим профілем (контактні дані, медична інформація, алергії).

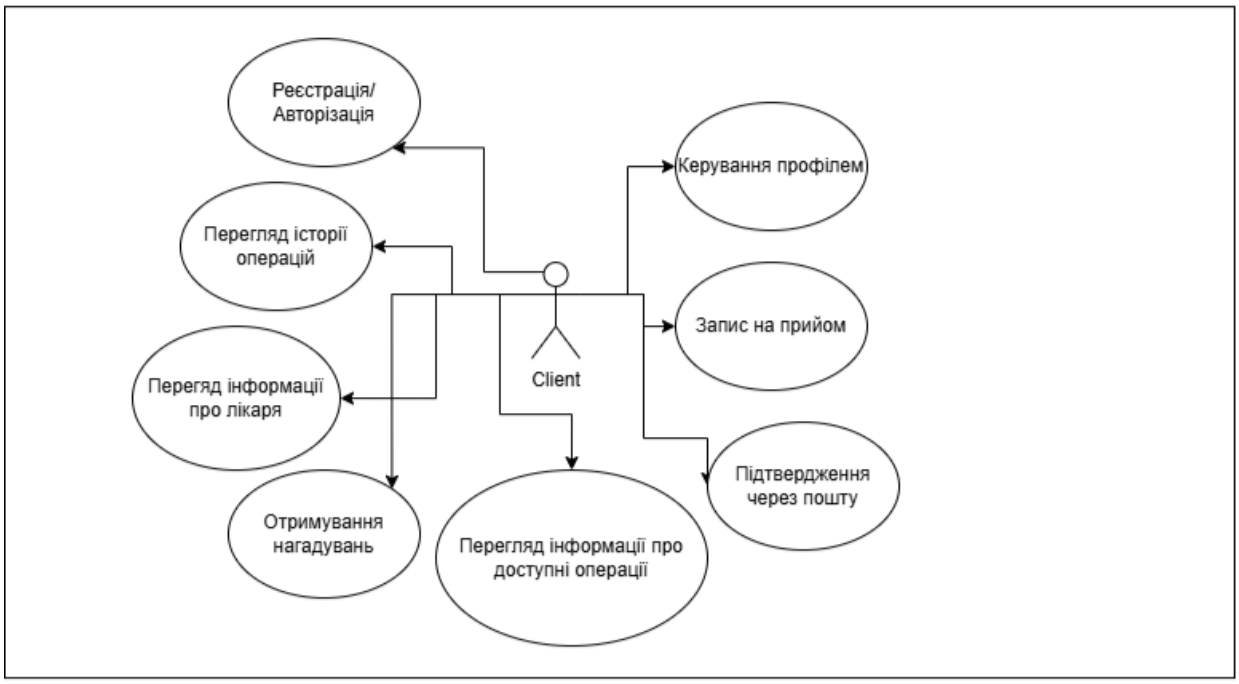


Рисунок 4 – *UserCase діаграма Клієнта*

2.4.2 Функціональність для лікарів

На рисунку 5 представлено UserCase діаграму працівників, яка включає наступні дії:

- Перегляд розкладу прийомів у календарному форматі.
- Отримання доступу до інформації пацієнтів (медична історія, алергії, попередні процедури).
- Додавання лікування до прийому (вибір послуг, примітки, рекомендації).
- Редагування та оновлення записів пацієнтів.
- Формування звітів про власну продуктивність та навантаження.

2.4.3 Функціональність для персоналу рецепції

Модуль забезпечує інструменти для ефективної роботи реєстратури та взаємодії з пацієнтами

- Управління базою пацієнтів (додавання, редагування, видалення).
- Запис пацієнтів на прийом від їхнього імені.
- Обробка платежів та позначення статусу оплати.
- Надсилання нагадувань про прийоми.
- Формування рахунків та квитанцій для пацієнтів.

2.4.4 Функціональність для адміністратора

Адміністратор отримує розширені можливості для управління клінікою та контролю її діяльності

- Управління персоналом (додавання та видалення лікарів та співробітників).
- Управління каталогом послуг та їхньою вартістю.
- Редагування інформації про клініку (контакти, графік роботи, акції).
- Отримання аналітичних звітів (статистика записів, дохід, продуктивність лікарів, завантаженість персоналу).
- Контроль безпеки та управління ролями користувачів (RBAC).

2.4.5 Додаткові функціональні можливості

Система також передбачає набір додаткових функцій для підвищення зручності та ефективності роботи

- Система сповіщень: автоматичні нагадування про прийоми (email, SMS, push), ручні дзвінки для важливих пацієнтів.
- Аналітика: Відображення статистики відвідувань, фінансові показники, продуктивність лікарів.
- Інтеграції: підтримка платіжних систем, SMS-шлюзів, email-сервісів.
- Багатомовність: підтримка української, англійської та російської мов.
- Адаптивний інтерфейс: коректне відображення на ПК, планшетах та смартфонах

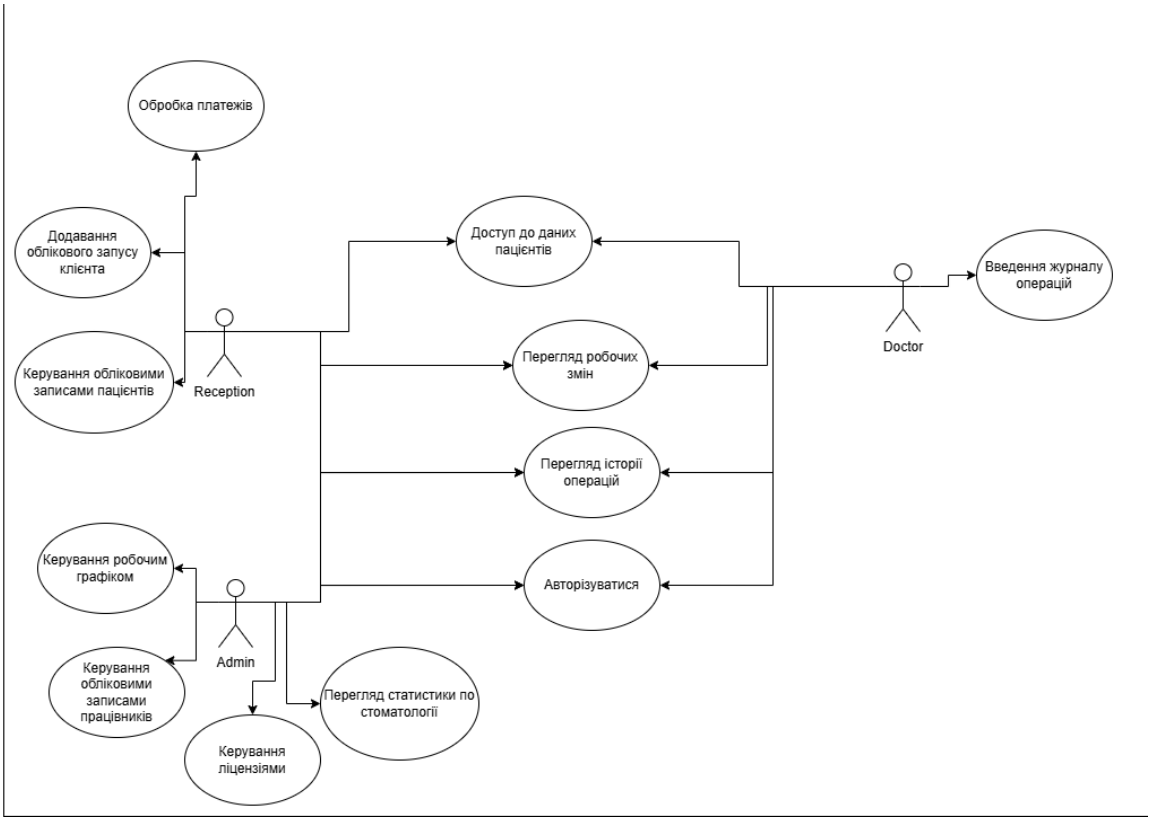


Рисунок 5 – *UserCase діаграма Працівники*

Таким чином, функціональність веб-застосунку охоплює всі ключові процеси стоматологічної клініки. Починаючи із запису пацієнтів до управління персоналом та фінансами. Це дозволяє створити комплексну систему, яка підвищує ефективність роботи клініки та рівень задоволеності пацієнтів.

2.5 База даних

База даних є ключовим елементом будь-якого сучасного програмного забезпечення, адже це забезпечує збереження, обробку та доступ до інформації. У випадку CRM-системи для стоматологічної клініки база даних виконує роль централізованого сховища, де зберігаються дані про пацієнтів, лікарів, прийоми, фінансові операції та інші бізнес-процеси

На рисунку 6 наведено узагальнену схему взаємодії користувачів із базою даних через центральну електронну обчислювальну машину (ЕОМ). Термінали формують запити, які надходять до центральної ЕОМ. Далі система управління базами даних (СУБД) аналізує ці запити, визначає потрібні таблиці та записи, звертається до бази даних і формує відповідь у вигляді результату, що повертається користувачам.

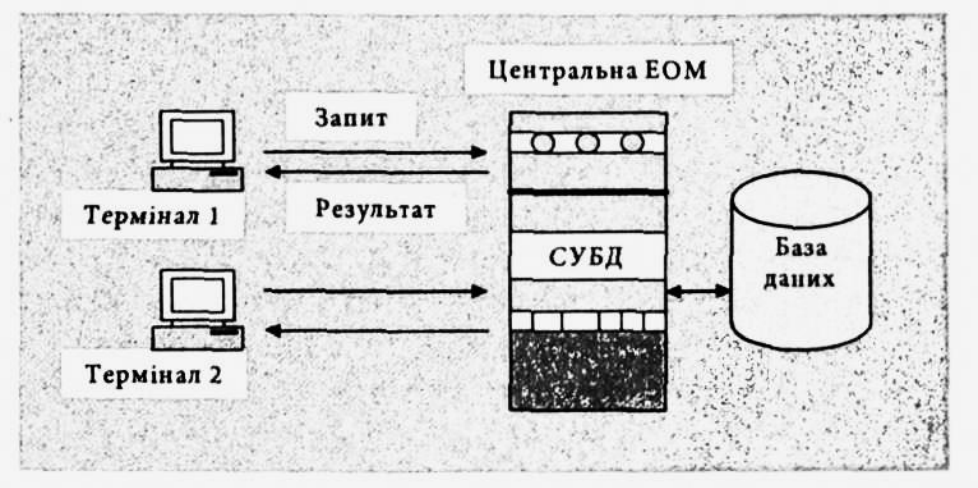


Рисунок 6 – *Схема взаємодії з базою даних*

Подібна структура забезпечує:

- Централізоване збереження інформації: Всі дані знаходяться у єдиному сховищі, що гарантує їх цілісність та узгодженість
- Контроль доступу та безпеку: Користувачі працюють із даними лише через СУБД, що дозволяє реалізувати механізми авторизації та захисту
- Ефективну обробку запитів: Система оптимізує пошук та вибірку даних, зменшуючи навантаження на користувачів
- Масштабованість та багатокористувацький режим: До центральної ЕОМ можуть підключатися численні термінали, що дозволяє одночасно працювати багатьом користувачам.

У контексті CRM-системи для стоматологічної клініки ця схема демонструє, як лікарі чи адміністратори отримують доступ до актуальних даних про пацієнтів, графік прийомів чи фінансові операції, не маючи прямого доступу до бази даних. Це гарантує безпеку, цілісність та контрольованість інформаційних потоків. Таким чином, схема на рисунку 3 ілюструє фундаментальний принцип роботи сучасних інформаційних систем, взаємодію користувачів із базою даних через СУБД, яка забезпечує узгодженість, захист та ефективність обробки даних.

2.5.1 MySql

MySQL (Рис. 7) є однією з найпоширеніших систем управління реляційними базами даних, яка широко використовується як у комерційних, так і в академічних проєктах завдяки своїй надійності, високій продуктивності та відкритому вихідному коду [3]. Це реляційна СУБД, що організовує дані у вигляді таблиць, які можуть бути взаємопов’язані між собою за допомогою ключів. Основною мовою взаємодії з системою є SQL (Structured Query Language) – це універсальна мова запитів, яка дозволяє створювати структури бази даних, модифікувати їх та виконувати різноманітні запити до даних.



Рисунок 7 – *Логотип MySQL*

MySQL застосовується для:

- збереження великих обсягів інформації у структурованому вигляді;
- виконання складних аналітичних запитів;
- підтримка транзакцій у фінансових операціях;
- інтеграції з веб-застосунками та CRM-системами;
- Підтримка роботи у багатокористувацькому середовищі, що потребують високої швидкості доступу та узгодженість даних.

Серед ключових переваг MySQL варто виділити:

- **Відкритий код і безкоштовна ліцензія**, завдяки чому система є доступною для широкого кола користувачів;
- **Високу продуктивність і масштабованість**, які дозволяють ефективно працювати як з невеликими проєктами, так і з великими корпоративними системами;
- **Підтримку транзакцій і реплікації**, що забезпечує цілісність даних та можливість створення резервних копій і балансування навантаження;
- **Велику спільноту користувачів і розробників**, яка активно підтримує розвиток системи, створює документацію, приклади та додаткові інструменти;
- **Гнучкість інтеграції**, завдяки якій MySQL підтримує інтеграцію з різними мовами програмування, фреймворками та платформами.

Таким чином, MySQL являється універсальним інструментом для роботи з даними, який поєднує простоту використання, відкритість та

потужний функціонал, що робить його одним із найпопулярніших рішень у світі для створення та підтримки сучасних інформаційних систем.

2.6 Frontend

Фронтенд є клієнтською частиною будь-якої сучасної інформаційної системи, яка забезпечує взаємодію користувача з програмним продуктом. Він відповідає за відображення даних, отриманих із серверної частини, та надання інструментів для введення, редагування й управління інформацією. Основна мета фронтенду полягає у створенні інтерфейсу, що характеризується доступністю, зрозумілістю та широкими функціональними можливостями, який дозволяє користувачам ефективно виконувати свої завдання.

Розробка фронтенду зазвичай здійснюється із використанням сучасних веб-технологій, таких як **HTML, CSS та JavaScript**, а також фреймворків і бібліотек, що забезпечують масштабованість та швидкість роботи. Серед найбільш поширених рішень можна виділити **React, Angular, Vue.js**, які дозволяють створювати компонентну архітектуру та забезпечують повторне використання елементів інтерфейсу.

Важливим аспектом фронтенду є адаптивність. Інтерфейс повинен зрозуміло відображатися на різних пристроях, включаючи комп’ютери, планшети та смартфони. Для цього застосовуються технології адаптивної верстки та бібліотеки стилів, такі як **Bootstrap** чи **TailwindCSS**.

У той же час фронтенд відповідає за інтеграцію з бекендом через **API-запити**. Це дозволяє отримувати дані з серверної частини, відображати їх у реальному часі та забезпечувати динамічну взаємодію. Для роботи з API часто використовуються інструменти по типу **Axios** або **Fetch API**, а для управління станом застосунку використовують такі як **Redux**, **MobX** чи інші менеджери стану.

Окрему увагу приділяють питанням безпеки та авторизації. Фронтенд реалізує механізми входу користувачів, перевірки прав доступу та захисту

персональних даних. Для цього застосовуються токени доступу (JWT), системи багатофакторної автентифікації та інтеграція з зовнішніми сервісами авторизації. Таким чином, фронтенд є ключовим елементом інформаційної системи, який поєднує технічну реалізацію з користувацьким досвідом. Він забезпечує зручність роботи, швидкий доступ до даних та інтеграцію з іншими компонентами системи, створюючи єдине середовище для ефективної взаємодії користувачів із програмним продуктом (див. рис. 8).

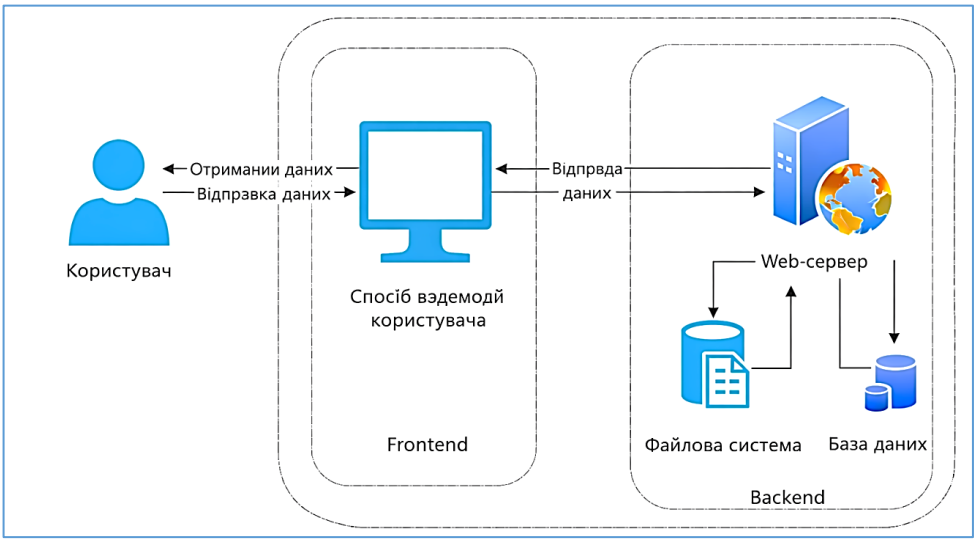


Рисунок 8 – *Схема взаємодії Frontend частини*

2.6.1 Стек технологій для Frontend

Для стоматологічної CRM-системи вчасності як і для інших проєктів, використання певних технологій може сприяти як до покращення так і до ускладненню реалізації проєкту. У проєкті використовується багато різних технологій, як за типом так і за призначенням. У таблиці 1 наведено технології, які були використані та їх призначення (таблиця 1).

Next.js. Основна технологія, на якій будується проєкт є Next.js . Використання Next.js дозволяє створити веб-додаток, що характеризується високою продуктивністю, масштабованістю та оптимізацію для пошукових систем, який відповідає потребам як пацієнтів, так і персоналу клініки.

Таблиця 1 – *Стек фронтенд-технологій*

Технологія	**Призначення**
Next.js	фреймворк на базі React
Awilix	контейнер інверсії залежностей для модульної архітектури.
Formik	бібліотека для роботи з формами та станом.
i18next	бібліотека для багатомовності та локалізації інтерфейсу.
normalizr	нормалізація вкладених API-даних для зручної роботи у Redux
react	бібліотека для створення компонентного інтерфейсу.
Redux-saga	middleware для управління асинхронними процесами та побічними ефектами.
yup	інструмент для схемної валідації даних у формах.
TailwindCSS	утилітарний CSS-фреймворк для швидкої та адаптивної верстки.

Awilix. Це сучасний контейнер для Dependency Injection (DI), який допомагає створювати масштабовані та добре структуровані застосунки з мінімізацією зовнішніх залежностей. Він дозволяє розділити логіку програми від механізмів створення об’єктів, роблячи код більш гнучким і зрозумілим.

Formik. Налаштування форм є трудомістким та потребує значних витрат часу, задля створення гарної структури форм. Formik дозволяє працювати із формами у React. Formik спрощує управління станом форм, обробку введених даних користувачем, валідацію та відправку даних (рис. 9).



Рисунок 9 – *Логотип Formik*

Yup. Для комфортної роботи із візуальною частиною застосунку, потрібні додаткові перевірки, які запобігають додаткового навантаження на сервер та швидкої перевірки введених даних.

I18Next. У сучасних умовах дуже важливо робити проєкти із добре реалізованими клієнтськими вимогами (рис. 10). Зручність використання, візуальний вигляд та інше. Серед найважливіших це розуміння контексту сторінки, тобто переклад проєкта багатьма мовами світу. I18Next - це бібліотека для інтернаціоналізації та локалізації. Вона дозволяє додавати багатомовну підтримку у застосунок, перемикаючи мови та працювати з перекладами у різних форматах.

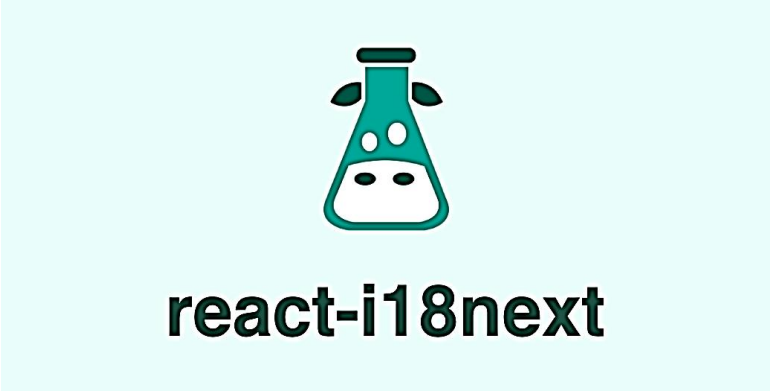


Рисунок 10 – *Логотип I18Next*

Normalizr. Під час розробки будь-якого проєкту виникають типові проблеми, такі як: переповнення пам’яті, швидкість роботи, коректність даних, способи та суцільність зберігання даних. Під час роботи із великою кількістю даних, виникає проблеми із нестачею виділеної пам’яті, дублювання даних. Способи вирішення цих проблем є багато, один із них це бібліотека Normalizr. Це бібліотека із інструментами для нормалізації даних. Normalizr перетворює

складні структури у прості та зручні для роботи вигляду (наприклад: використання цих даних у Redux)

Redux. Із розширенням проєкту постає питання щодо зберігання даних, а саме у якому вигляді їх зберігати, як обробляти, де зберігати та інші питання (рис. 11). Одним із підходів вирішення цієї проблеми це централізація даних у одному сховищі. Redux - це централізоване сховище стану для веб-додатків, яке дозволяє зберігати та керувати даними у єдиному місці. Redux забезпечує передбачуваний потік даних, спрощує відстеження змін та робить роботу з великими застосунками більш контрольованою й масштабованою.



Рисунок 11 – Логтипо Redux

Redux-saga. Це middleware для Redux, який дозволяє керувати асинхронними процесами (API-запити, побічні ефекти) через генератори (рис. 12). Це робить потік даних більш контрольованим та передбачуваним.



Рисунок 12 – *Логотип Redux-saga*

TailwindCSS. Візуальне оформлення є важливим компонентом сучасних проєктів (рис. 13). У деяких випадках відіграє більше значення ніж інші частини проєкту. Задля створення цікавого та привабливого дизайну потрібно багато часу, а TailwindCSS сприяє спрощенню та прискоренню процесу розробки. TailwindCSS - це утилітарний CSS-фреймворк, який дозволяє швидко створювати адаптивний дизайн без написання великої кількості кастомних стилів. Він підтримує мобільну адаптивність та значно прискорює розробку UI.



Рисунок 13 – *Логотип TailwindCSS*

2.6.2 Опис Next.js

Next.js (рис. 14) це сучасний фреймворк, який побудований на базі React і поєднує можливості клієнтського та серверного рендерингу [4].



Рисунок 14 – *Логотип Next*

Next.js функціонує як надбудова над React, а не як самостійна платформа, додаючи функціонал, необхідний для створення швидких, масштабованих і SEO-оптимізованих веб-додатків. Архітектура Next.js базується на сучасних веб-технологіях: React.js для компонентної моделі, Node.js як середовище виконання, інструменти як трансформацію та оптимізація коду, а також інтегровану підтримку TypeScript і ESLint для типізації та контролю якості.

Використання Next.js у проєкті стоматологічної клініки дозволяє вирішити ключові проблеми класичних SPA-додатків. Серверний рендеринг зменшує час відгуку інтерфейсу, що важливо для пацієнтів, які користуються мобільними пристроями. Пошукові системи краще індексують контент, оскільки він формується на сервері, що забезпечує SEO-оптимізацію та залучення нових клієнтів. Статична генерація сторінок використовується для інформаційних розділів, які не змінюються часто, наприклад опис послуг чи профілі лікарів, що зменшує навантаження на сервер і прискорює роботу сайту.

У реалізації проєкту Next.js забезпечує роботу особистого кабінету пацієнта, де відображається історія відвідувань, результати аналізів та електронна медична картка. Календар записів інтегрується з бекендом через API маршрутизацію Next.js, що дозволяє пацієнтам бачити актуальні часи для прийому в реальному часі. Панель адміністратора реалізована як захищений маршрут із допоміжним програмним забезпечення, який перевіряє права доступу, що гарантує безпеку даних. Інформаційні сторінки клініки генеруються статично, а багатомовність забезпечується інтеграцією з i18next, що дозволяє пацієнтам перемикати мову інтерфейсу.

Як результат, Next.js у проєкті стоматологічної клініки виступає як ключова технологія, що поєднує сучасну архітектуру, продуктивність та безпеку. Він дозволяє створити єдиний веб-додаток, який підтримує функціональність для різних категорій користувачів, забезпечує швидкий доступ до даних, підтримує адаптивний інтерфейс та інтеграцію з іншими компонентами системи.

2.7 Backend

Backend - це серверна частина програмного забезпечення, яка забезпечує реалізацію бізнес-логіки, обробку даних та взаємодію з базою даних у SaaS-системах та інших інформаційних системах. Backend реалізує управління даними, включаючи бізнес-логіку, обробку даних, та механізми взаємодії між користувачем та базою даних. У зв'язці із цілим проєктом frontend відповідає за користувацький інтерфейс системи, який доступний для різних категорій користувачів, зокрема лікарів, працівників реєстратури, адміністрації та пацієнтів, у цей же час backend здійснює обробку даних, які були введені користувачем. Backend виступає посередницьким рівнем між інтерфейсом користувача та базою даних та виконує повний контроль над тим що змінюється у базі даних.

Основна мета бекенду полягає у створенні надійної та масштабованої основи для роботи системи, яка гарантує стабільність, безпеку та ефективність усіх процесів

Основні функції Backend:

- Обробка запитів від користувачів. Frontend передає дані до Backend, який здійснює їх обробку. Обробка даних передбачає перевірку їх цілісності, коректності та відповідність встановленим критеріям, Після виконання цих дій, backend зберігає ці дані у базі даних
- Управління даними. Серверна частина відповідає за збереження та організацію даних отриманих від клієнтської . Backend гарантує, що дані завжди доступні, структуровані та захищені
- Безпека та авторизація. Забезпечує автентифікацію користувачів використовуючи логін, пароль та токени доступу, а також контроль ролей таких як: адміністратор, лікар, пацієнт. Також до безпеки відноситься і захист персональних даних.

- Інтеграція з іншими сервісами. Backend дозволяє системі взаємодіяти з платіжними системами, мобільними додатками та іншими сторонними сервісами використовуючи API.

Frontend та Backемними складовими будь-якого програмного забезпечення, кожен з яких виконує специфічні функції, необхідні для коректної роботи системи.

Backend у програмному забезпеченні реалізовує серверну логіку та забезпечує роботу всієї системи (див. рис. 15).

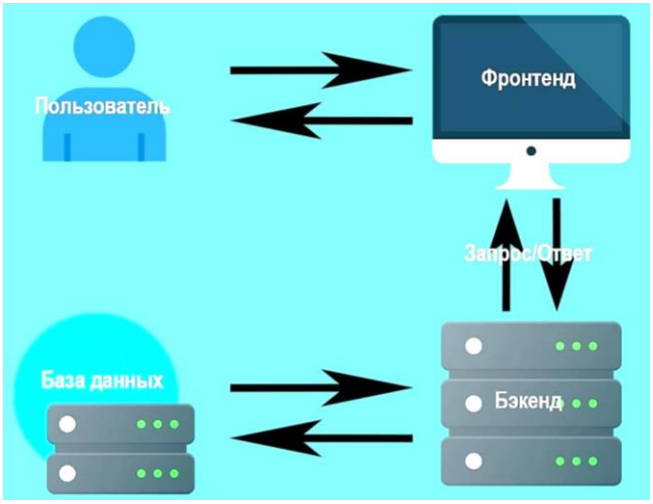


Рисунок 15 – *Схема роботи Backend*

Backend містить API-маршрути, завдяки яким frontend отримує доступ до даних і функціоналу. Це можуть бути Rest або GraphQL інтерфейси, що дозволяють ефективно організувати комунікацію між клієнтом і сервером. Однією із головних завдань backend це взаємодія із базою даних. Backend зберігає інформацію про сутності, а також забезпечує швидкий та безпечний доступ до даних. Backend повинен обробляти тисячі запитів одночасно, задля цього використовується технології для кешування даних, оптимізація навантаження та різні архітектури.

Таким чином, backend являється ядром системи, яка забезпечує стабільність, безпеку даних та продуктивність, створюючи основу для роботи frontend та взаємодії користувачів із застосунком.

2.7.1 Стек технологій для Backend

Для Backend-частини теж використовуються сторонні бібліотеки. Призначення бібліотек для backend та frontend відрізняється відповідно до їхніх функцій. У кожній частині проєкту, будь-то frontend чи backend, існують свої технології. Для серверної частини використання бібліотек є критичною частиною, оскільки бібліотеки забезпечують взаємодію із зовнішніми системами та сервісами, доступу до яких немає. Бібліотеки можуть реалізовувати різні функції, зокрема обробка даних, забезпечення безпеки та цілісності даних, інтеграція з базами даних та зовнішніми сервісами. Використання відповідних технологій дозволяє створити гнучку, масштабовану та надійну архітектуру, яка відповідає функціональним вимогам різних категорій користувачів (таблиця 2).

Таблиця 2 – *Стек технологій для Backend*

Технологія	**Призначення**
TypeORM	ORM для роботи з реляційними базами даних, зокрема MySQL, забезпечує зручну абстракцію над SQL-запитами.
mysql2	Драйвер для роботи з MySQL, оптимізований для продуктивності та асинхронних операцій.
faker	Генерація тестових даних для розробки та моделювання сценаріїв.

Продовження таблиці 2

Технологія	**Призначення**
swagger	Автоматична документація REST API, що спрощує інтеграцію та тестування.

react-email	Інструмент для створення та рендерингу шаблонів електронних листів.
class-transformer	Перетворення plain-об’єктів у класи та навпаки, зручне для роботи з DTO.
argon2	Алгоритм хешування паролів з високим рівнем безпеки.
bcrypt	Популярний алгоритм хешування паролів, використовується як альтернатива або доповнення до Argon2.
class-validator	Валідація даних на рівні класів, інтегрується з class-transformer.
connect-redis	Middleware для зберігання сесій у Redis, що підвищує масштабованість.
ioredis	Клієнт для роботи з Redis, підтримує кластеризацію та високу продуктивність.

TypeOrm. TypeOrm - це сучасний ORM для роботи з реляційними базами даних, зокрема MySQL [11]. TypeOrm рабезпечує опис сутностей у вигляді класів та працює з ними без написання складних SQL-запитів. Використання ORM підвищує зрозумілість, гнучкість та підтримування програмного коду, а також спрощує міграції та тестування

Mysql2. Це високопродуктивний драйвер для роботи з MySQL у Node.js. Він оптимізований для асинхронних операцій, підтримує проміси та забезпечує швидку взаємодію з базою даних. Застосування mysql2 сприяє стабільності та ефетивності системи.

Faker. У процесі розробки програмного забезпечення виникає необхідність у використанні тестових даних. Faker - це бібліотека, що використовується для генерації випадкових і реалістичних даних, зокрема імен, адрес, телефонів тощо. Бібліотека забезпечує швидке моделювання сценаріїв та тестування роботи системи без використання реальних даних.

Swagger. Swagger - це інструмент для автоматичної генерації та документування Rest API. Swagger забезпечує автоматизоване формування інтерактивної документації, яка підтримує процеси тестування та інтеграцію сервісів. Використання Swagger забезпечується зрозуміле представлення структури та функціональних можливостей API.

React-email. Це бібліотека для створення шаблонів електронних листів на базі React. Ця бібліотека дозволяє будувати адаптивні та зрозумілі Email-повідомлення, які можуть бути інтегровані у серверну частини системи для відправки повідомлень користувачам.

Argon2. Це сучасний алгоритм хешування паролів, який широко використовується у веб-додатках. Алгоритм Argon2 реалізує механізм додавання солі, що підвищує стійкість до атак.

Class-transformer. Class-transformer - це бібліотека, що забезпечує перетворення простих об’єктів у класи та навпаки. Бібліотека оптимізована для роботи з DTO (Data Transfer Objects) та допомагає підтримувати чисту архітектуру у проєкті.

Class-validator. Class-validator - це інструмент для перевірки даних на рівні класів. Він інтегрується з class-transformer та реалізує механізми перевірки правильності введених даних до їхнього збереження у базі. Це зменшує кількість помилок та підвищує надійність системи.

Connect-redis. Connect-redis - це middleware для зберігання сесій у Redis. Connect-redis забезпечує масштабованість системи, швидкий доступ до даних та ефективне управління з користувацькими сесіями.

Ioredis. Це клієнт для роботи з Redis, що підтримує кластеризацію та високу продуктивність. Він застосовується для кешування, управління сесіями та швидкої обробки даних у реальному часі.

2.7.2 Опис Nest.js

Nest.js (рис. 16) - це фреймворк для створення серверних застосунків на Node.js, що базується на TypeScript та реалізує модульну архітектуру [5]. Він не є окремою мовою чи платформо, а функціонує як надбудова над Express, забезпечуючи структурованість, інверсію контролю та підтримку декораторів.

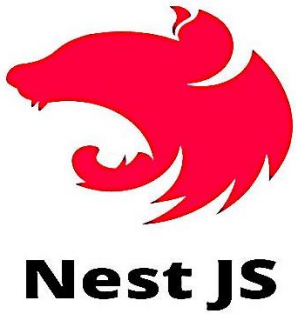


Рисунок 16 – *Логотип Nest.js*

Таким чином Nest.js забезпечує створення масштабованих, безпечних та добре організованих серверних системи, які легко інтегруються з клієнтськими застосунками.

У бекенд частині проєкту Nest.js вирішує ключові завдання, пов’язані з безпекою, масштабованістю та інтеграцією з іншими системами:

- Особистий кабінет пацієнта реалізований у Nest.js забезпечує API для доступу до історії відвідувань, результатів аналізів та електронної медичної картки. Механізм Guards забезпечує контроль доступу, що обмежує перегляд даних лише необхідним категоріям користувачів.
- Календар записів реалізований як окремий модуль, що взаємодіє з базою даних через ORM. API-маршрути забезпечують бронювання часових проміжків пацієнтами та управління графіком лікарів у реальному час.
- Адміністративна панель реалізована через захищені маршрути з використанням JWT-автентифікації та ролей (admin, doctor, patient).

Middleware здійснює перевірку прав доступу, забезпечуючи безпеку даних.

- Nest.функціонує як серверний рівень, що обслуговує запити від клієнтської частини. Такий підхід забезпечує розподіл відповідальності, де Next.js відповідає за рендеринг та UI, а Nest.js забезпечує бізнес-логіку та роботу з даними.
- Мультимовність та інтернаціоналізація Nest.js підтримує інтеграцію з i18next або іншими бібліотеками для локалізації, щоб забезпечити узгодженість між багатомовним інтерфейсом Next.js.

Таким чином, Nest.js у проєкті стоматологічної клініки використовується як основний серверним фреймворк. У поєднанні з Next.js він формує повноцінну архітектуру сучасного веб-додатку, завдяки чому забезпечується узгодженість під час взаємодії між різними категоріями користувачів.

2.8 Висновки з розділу 2

У результаті проведеного дослідження було установлено, що CRM-система базується на сучасних веб-технологія та принципах модульної архітектури. Було установлено функціональні та не функціональні вимоги до CRM-системи. Обраний технологічний стек показав, що цей стек забезпечує усі необхідні поточні вимоги до CRM-системи із можливістю подальшого розширення

3 СТРУКТУРА ПРОЄКТУ CPM СТОМАТОЛОГІЧНОЇ КЛІНІКИ

3.1 Структура бази даних проєкту

Структура бази даних проєкту побудована як реляційна модель, що відображає основні процеси роботи стоматологічної клініки. Вона реалізована у вигляді взаємопов’язаних таблиць, кожна з яких відповідає за окремий аспект діяльності системи. Така організація дозволяє забезпечити узгодженість даних, централізоване зберігання інформації та можливість комплексної аналітики. У центрі моделі знаходиться блок прийомів пацієнтів, який пов’язує клієнтів і лікарів, містить дані про дату, час, статус та примітки, що забезпечує можливість відстежування історії лікування та планування робочого графіку персоналу.

Додатково передбачено таблиці для опису дій та операцій, що виконуються під час прийому, а також модуль персоналу клініки, де зберігаються дані про лікарів, їхні спеціалізації, ліцензії та робочі зміни. Така організація даних забезпечує підтвердження кваліфікації співробітників і забезпечує оптимальний розподіл навантаження. Окремо виділено таблиці для клієнтів, які містять персональну інформацію та медичні особливості, що забезпечує персоналізований підхід до лікування. Фінансовий блок представлений таблицею платежів, яка містить дані про суми, методи оплати та статуси транзакцій, що забезпечує формування звітності і контроль розрахунків. Таблиць сповіщень реалізує механізми комунікації, зокрема нагадування та інформаційні повідомлення (див. рисунок 17).

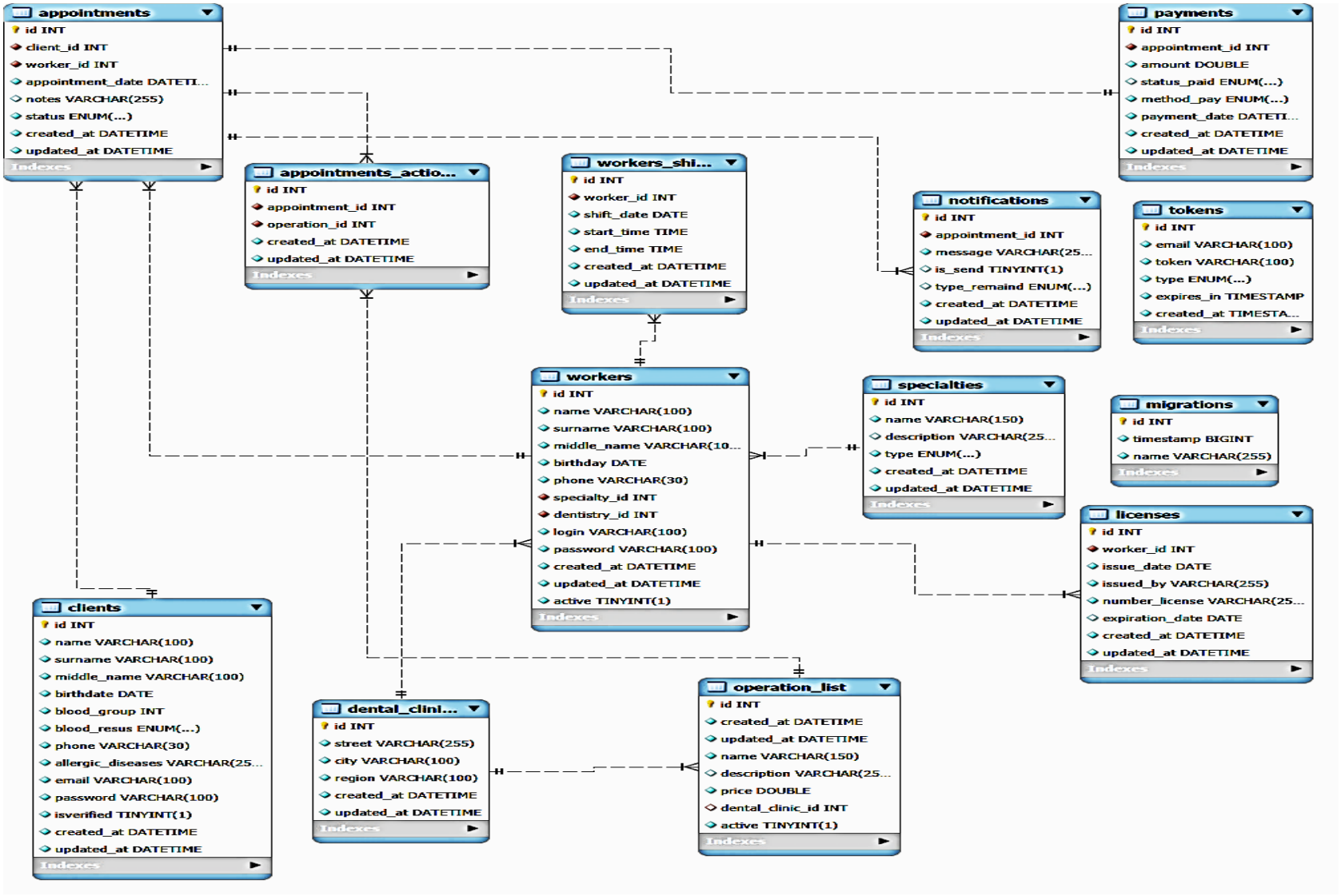


Рисунок 17 – *Схема бази даних*

Таким чином, структура бази даних забезпечує узгодженість даних, підтримку транзакцій, аналітичну обробку та інтеграцію з іншими модулями CRM-системи. База даних виступає основою для реалізації функціоналу проєкту та гарантує стабільність роботи всієї інформаційної системи. Для підтвердження практичної реалізації наведемо приклади створення окремих таблиць, які відіграють ключову роль у роботі системи.

Таблиця appointments призначена для збереження інформації про прийоми пацієнтів. Таблиця містить дані про клієнта, лікаря, дату та час прийому, статус і додаткові примітки. У структурі передбачено зовнішні ключі, які забезпечують зв'язок із таблицями клієнтів та лікарів. Завдяки цьому забезпечується цілісність даних, а також автоматичне оновлення або видалення записів при зміні пов'язаних сутностей. У результаті система Забезпечує відстеження історії лікування, контроль робочого навантаження лікарів та формування аналітики щодо проведених процедур (див. лістинг1).

Лістинг 1 – *Реалізація таблиці Appointment*

```
CREATE TABLE IF NOT EXISTS appointments (
    id INT PRIMARY KEY AUTO_INCREMENT,
    client_id INT NOT NULL,
    worker_id INT NOT NULL,
    appointment_date DATETIME NOT NULL,
    notes VARCHAR(255),
    status
ENUM('schedule','completed','wait_paid','cancelled')     NOT
NULL DEFAULT 'schedule',
    created_at DATETIME NOT NULL DEFAULT CURRENT_TIMESTAMP,
    updated_at DATETIME NOT NULL DEFAULT CURRENT_TIMESTAMP ON
UPDATE CURRENT_TIMESTAMP,
    CONSTRAINT fk_appointment_client FOREIGN KEY (client_id)
REFERENCES clients(id) ON DELETE CASCADE ON UPDATE CASCADE,
```

```
    CONSTRAINT fk_appointment_worker FOREIGN KEY (worker_id)
REFERENCES workers(id) ON DELETE CASCADE ON UPDATE CASCADE
) ENGINE=InnoDB;
```

Таблиця tokens призначена для зберігання токенів авторизації та безпеки, що застосовуються у механізмах автентифікації та безпеки. Кожен токен характеризується типом, датою завершення дії та автоматично генерується у процесі виконання відповідної операції. Така реалізація забезпечує захист персональних даних користувачів і підвищує рівень безпеки системи (див. лістинг 2).

Лістинг 2 – *Реалізація таблиці tokens*

```
CREATE TABLE IF NOT EXISTS tokens (
    id INT NOT NULL AUTO_INCREMENT,
    email VARCHAR(100) NOT NULL,
    token VARCHAR(100) NOT NULL UNIQUE,
    type ENUM('VERIFICATION', 'TWO_FACTOR', 'PASSWORD_RESET')
NOT NULL,
    expires_in TIMESTAMP NOT NULL,
    created_at TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP,
    PRIMARY KEY (id)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4;
```

Таким чином, приклади створення таблиць демонструють практичну реалізацію ключових механізмів бази даних. Кожна таблиця характеризується певним призначенням та виконує визначену функцію у CRM-системи стоматологічної клініки. З метою узагальнення подано короткий опис основних таблиць, з яких складається структура бази даних.(див. таблиця 3)

Таблиця 3 – *Призначення кожної таблиці*

Назва таблиці	**Призначення**
Appointment	Зберігає інформацію про прийоми пацієнтів, включаючи клієнта, лікаря, дату, час, статус та примітки.
Appointment_action	Деталізує дії та процедури, що виконуються під час прийому.
Clients	Зберігає персональну інформацію про пацієнтів, їхні контактні дані та медичні особливості.
Dental_clinics	Зберігає інформацію про клініку: назву, адресу, контактні дані
Worker_shift	Відображає графік роботи лікарів та персоналу клініки.
Workers	Містить дані про лікарів та персонал, включаючи спеціалізацію, ліцензії та графік роботи.
Operation_list	Містить перелік стоматологічних операцій та їхню вартість.
Notifications	Використовується для надсилання сповіщень клієнтам та персоналу.
Specialties	Містить інформацію про спеціалізації лікарів, їхні типи та опис.
Licenses	Використовується для зберігання даних про ліцензії лікарів, їхні терміни дії та органи, що їх видали.
Payments	Відповідає за облік фінансових операцій, зберігає інформацію про суми, методи оплати та статуси транзакцій.
Tokens	Використовується для зберігання токенів авторизації, підтвердження електронної пошти, двофакторної автентифікації та відновлення паролю.

Таким чином, узагальнена характеристика таблиць ілюструє логічну цілісність структури бази даних та розуміння, як кожна сутність інтегрується у загальну систему управління стоматологічною клінікою.

3.2 Клієнтська частина проєкту

Стоматологічна CRM-система реалізована як web-додаток, який функціонує у браузері та реалізує взаємодію користувачів із системою. Клієнтська частина побудована на Next.js, який поєднує можливості React та серверного рендерингу, що забезпечує формування масштабованого та SEO-оптимізованого інтерфейсу. Клієнтська частина також забезпечує різний функціонал задля забезпечення користувацького досвіду, зокрема адаптований дизайн та багатомовність системи. Адаптований дизайн забезпечує коректність подання інформації користувачу при різних форматах відображення, а багатомовність забезпечує зрозумілість даних для різних категорій людей. Для кожної категорії користувачів передбачено окремі сторінки з відповідними функціональними особливостями. Наприклад для адміністраторів реалізовано функцію додавання графіку роботи працівників та керувати обліковими записами працівників. Для працівників реєстратури передбачено функціонал для додавання облікових записів клієнтів та додавання записів до лікаря (див. рис. 18).

Frontend реалізований на основі модульній архітектури, як забезпечує навігацію сторінками та підтримує масштабованість системи (див. рис. 19).

Next.js забезпечує підтримку всіх категорій користувачів, а саме лікарів, адміністрації, персонал рецепції та клієнтів. Для кожного з них реалізовано окремі сторінки з відповідним функціоналом. У проєкті використано взаємопов’язані технології, які працюють у зв’язці одна з одною.

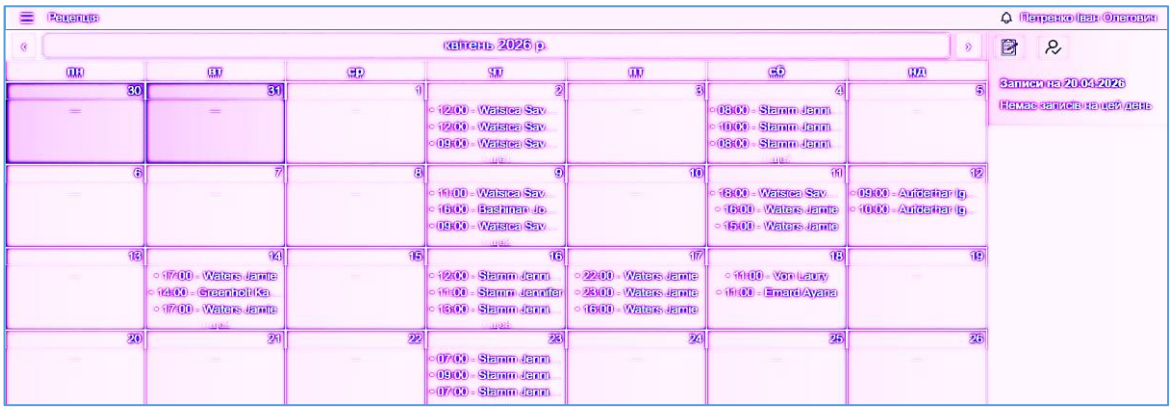


Рисунок 18 – *Приклад реалізації сторінки для працівників приймальні*



Рисунок 19 – *Структура проекту*

1. Formik + Yup. Ці технології використовуються для ефективної роботи з формами у веб-додатках. Formik реалізує управління станом форми, обробку введення користувача, відправку даних та зручну інтеграцію з React-

компонентами. Використання Formik усуває потребу у ручному відстеженні стану кожного поля форми. Yup забезпечує опис правил перевірки у вигляді декларативної схеми. Formik інтегрується з Yup і автоматично перевіряє дані відповідно до заданих правил та автоматично формує повідомлення про помилку без необхідності додаткового коду (див. лістинг 3 та лістинг 4).

Лістинг 3 – *Приклад використання Yup*

```
export const CreateAppointmentSchema = Yup.object().shape({
  clientId: Yup.number()
    .required("Client ID є обов’язковим")
    .positive("Client ID має бути додатнім")
    .integer("Client ID має бути цілим числом"),
  dentistId: Yup.number()
    .required("Dentist ID є обов’язковим")
    .positive("Dentist ID має бути додатнім")
    .integer("Dentist ID має бути цілим числом"),
  appointment_date: Yup.date()
    .required("Дата прийому є обов’язковою")
    .min(new Date(), "Дата прийому не може бути в минулому"),
  notes: Yup.string()
    .max(500, "Нотатки не можуть перевищувати 500 символів")
    .optional(),
});
```

Лістинг 4 – *Приклад використання Formik*

```
 <Formik
          initialValues={initialValues}
          validationSchema={CreateAppointmentSchema}
          onSubmit={onSubmit}
        >
          <Form>
```

```
            {/* Client full name */}
            <ClientField />
            {/* Worker full name */}
            <WorkerField />
            {/* appointment date */}
            <AppointmentDateField />
            {/* notes */}
            <NotesField />
            <SubmitAddingAppointment />
          </Form>
        </Formik>
```

Таким чином, Formik реалізує механізм управління формами, а Yup забезпечує механізм валідації введених даних, що характеризуються гнучкістю та надійністю. У поєднані Formik та Yup формують комплексний інструмент для роботи з формами у проєкті (див. рис. 20).

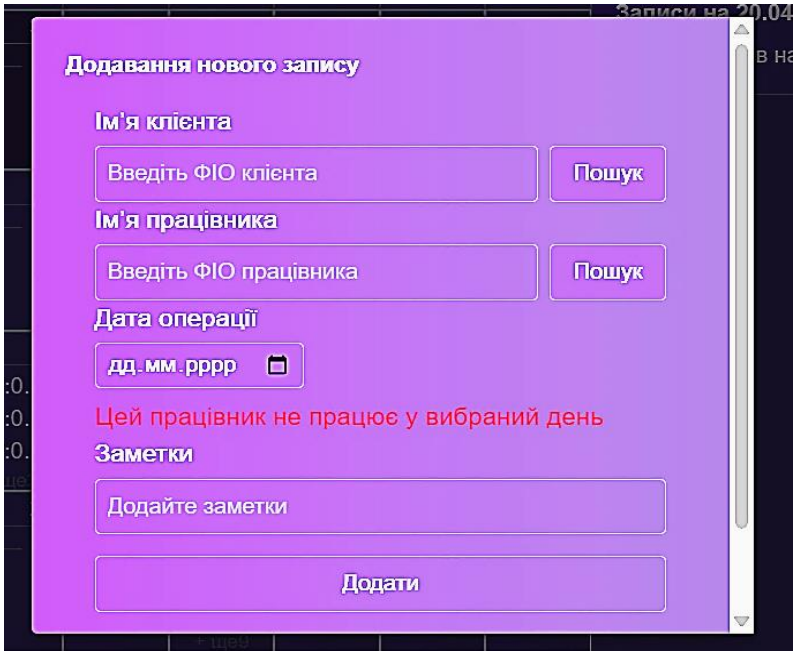


Рисунок 20 – *Реалізація Formik та Yup*

2. Redux + Redax-saga. Комбінація Redux та Redux-saga застосовується для централізованого управління станом застосунку та асинхронними процесами. Redux функціонує як єдине сховище даних, де зберігається інформація про клієнтів, записи, розклади тощо. Він забезпечує передбачуваний потік даних, а саме виконувати зміни використовуючи дії та редюсери, завдяки чому забезпечується контрольованість та прозорість.
Redux-saga у цій зв'язці виконує роль middleware, який керує асинхронними операціями, наприклад викликами API. Замість того щоб писати асинхронний код безпосередньо у компонентах, Redux-saga використовує генератори (function*) для опису побічних ефектів. Це дозволяє відокремити бізнес-логіку від інтерфейсу та зробити її більш передбачуваною (див. лістинг 5).

Лістинг 5 – *Приклад реалізації Redux та Redux-saga*

```
export enum ClientActionSaga {
  AddClient = "client/addClientSaga",
  GetClientsByFullName = "client/getAllByFullName",
}

@EntityReducer(EntitiesRedux.Clients)
export class ClientEntity extends BaseEntity {
  constructor(ctx: any) {
    super(ctx, EntitiesRedux.Clients, {});
  }
  *getClientSaga(action: GetClientAction) {
    yield  call(this.xRead.bind(this),  `/clients/test/all`,
ActionReducer.Get);
  }
  *addClientSaga(action: AddClientAction) {
    yield call(
      this.xSave.bind(this),
      `/auth/registrationClient`,
```

```
      action.payload,
      ActionReducer.Post,
    );
  }
  *watch() {
    yield              takeLatest(ClientActionSaga.GetClient,
this.getClientSaga.bind(this));
    yield              takeLatest(ClientActionSaga.AddClient,
this.addClientSaga.bind(this));
      yield takeLatest(ClientActionSaga.GetClientsByFullName,
this.GetClientsByFullNameSaga.bind(this));

  }
}
```

Такий підхід забезпечує чітке розділення відповідальності: компоненти інтерфейсу лише відправляють дії, а Redux-saga виконує запит до API, а Redux зберігає результати у централізованому сховищі. У результаті система характеризується масштабованістю, передбачуваністю та спрощеним процесом підтримки. Чітке розділення обов'язків у системі забезпечує розробнику швидке виявлення причин та локалізації проблеми, пов'язаних із використанням та зберіганням даних (див. рис. 21).

3. Awilix + Redux. Комбінація Awilix та Redux застосовується для підвищення модульності та керованості архітектури. Redux функціонує як централізоване сховище стану, де зберігаються усі дані застосунку, такі як клієнти, записи, розклад. Redux забезпечує передбачуваний потік даних: будь-які зміни відбуваються через екшени та редюсери, що забезпечує контрольованість та прозорість.

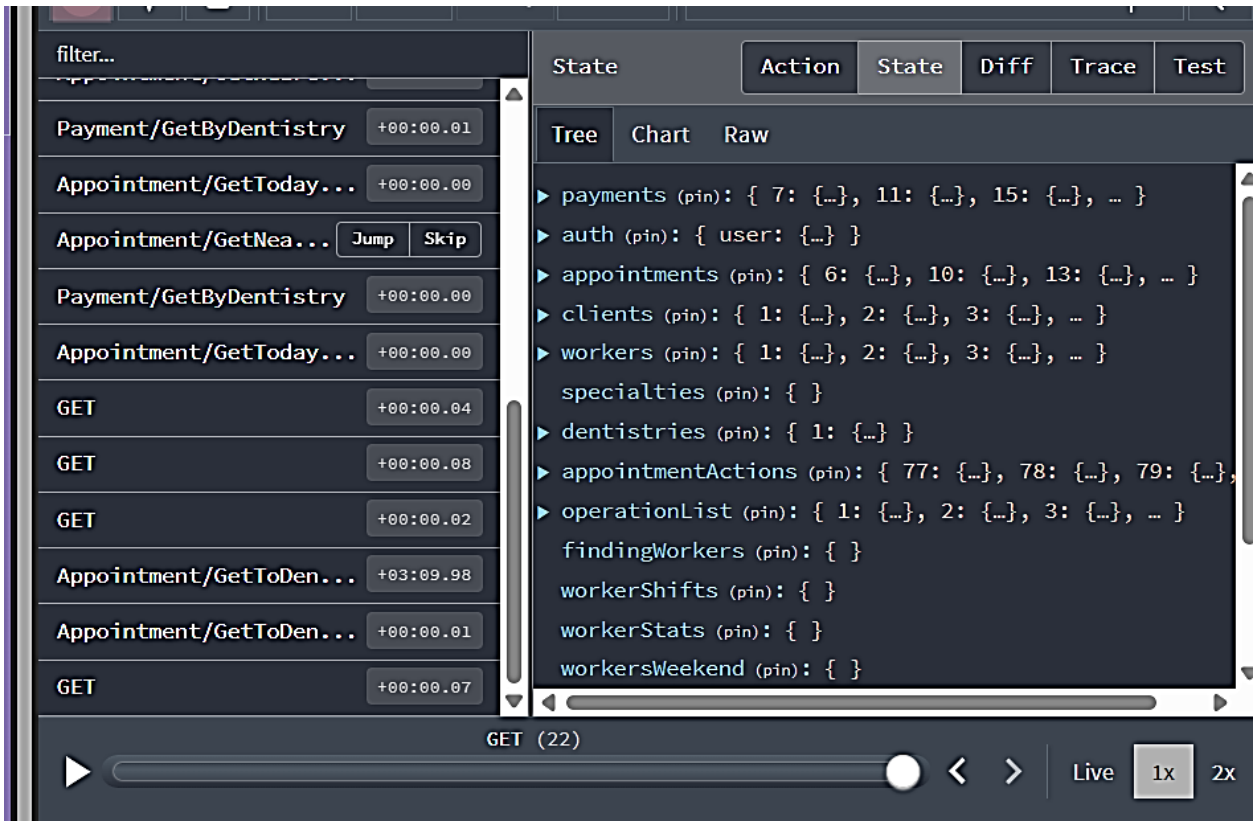


Рисунок 21 – *Реалізація Redux у браузері*

Awilix у цій зв’язці додає шар інверсії залежностей (Dependency Injection). Такий підхід передбачає відмову від жорсткого зв’язкування компонентів і сервісів. Компоненти та сервіси реєструються у контейнері Awilix і можуть динамічно інтегруватися у логіку Redux. Такий підхід забезпечує можливість заміни або тестування окремих частин системи, не змінюючи весь код. Наприклад, у проєкті передбачена можливість інтеграції різних сервісів для роботи з API або логікою клієнтів через Awilix, а Redux буде використовувати їх для оновлення стану (див. лістинг 6).

Лістинг 6 – *Реалізація DI із класами*

```
const clientContainer = createContainer({
  injectionMode: "PROXY",
});
const expansion = {
```

```
  ClientEntity: asClass(ClientEntity).singleton(),
  DentistryEntity: asClass(DentistryEntity).singleton(),
  WorkerEntity: asClass(WorkerEntity).singleton(),
  SpecialtyEntity: asClass(SpecialtyEntity).singleton(),
};

clientContainer.register({ ...expansion });

export default clientContainer;
```

Таким чином, Redux реалізує механізми управління станом, а Awilix забезпечує організацію залежностей між сервісами та бізнес-логікою. Разом вони створюють гнучку архітектуру. Redux зберігає дані у централізованому сховищі, а Awilix забезпечує інтеграцію потрібних сервісів для роботи з даними. У проєкті стоматологічної клініки це означає, що логіка роботи з клієнтами, записами чи лікарями може бути розширена або модифікована без порушення загальної структури системи.

4. Normalizr + Redux. Комбінація Normalizr та Redux застосовуються для підвищення передбачуваності та ефективності роботи з даними у централізованому сховищі . Redux функціонує як централізоване сховище стану застосунку у вигляді об’єктів, однак отримані з серверної частини дані можуть мати складну вкладену структуру, що ускладнює їх обробку.

Normalizr реалізує перетворення вкладених API-відповідей у нормалізовану форму, розділяючи складні об’єкти на прості сутності (наприклад, окремо клієнти, окремо записи, окремо лікарі) та зберігає їх у вигляді словників із ключами-ідентифікаторами (див. рис. 22).

Це дозволяє Redux працювати з даними як із плоскими структурами, завдяки чому спрощується оновлення стану та уникнення дублювання (див. лістинг 7).

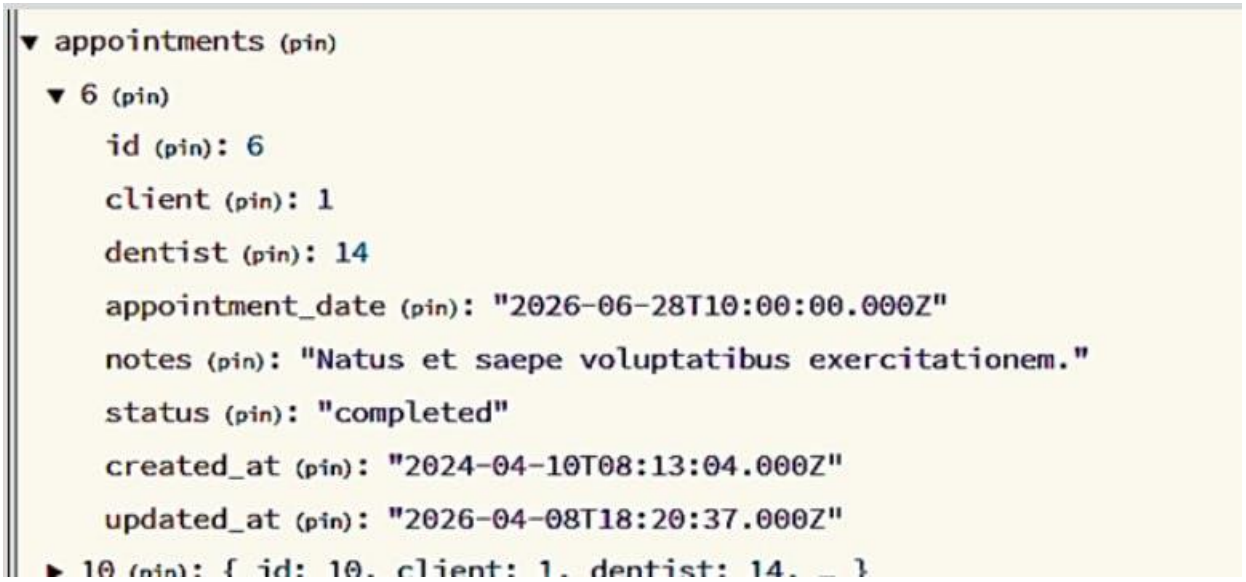


Рисунок 22 – *Приклад нормалізації даних*

Лістинг 7 – *Схема нормалізації сутності*

```
@EntityReducer(EntitiesRedux.AppointmentActions)
export class AppointmentActionsEntity extends BaseEntity {
  constructor(ctx: any) {
    super(ctx, EntitiesRedux.AppointmentActions, {
      appointment:                                    new
schema.Entity(EntitiesRedux.Appointments, {
        client: new schema.Entity(EntitiesRedux.Clients),
        dentist: new schema.Entity(EntitiesRedux.Workers, {
          specialty:                                  new
schema.Entity(EntitiesRedux.Specialties),
          dentistry:                                  new
schema.Entity(EntitiesRedux.Dentistries),
        }),
        appointment_actions: [
          new schema.Entity(EntitiesRedux.AppointmentActions,
{
            operation:                                new
schema.Entity(EntitiesRedux.OperationList, {
```

```
                dental_clinics:                                    new
schema.Entity(EntitiesRedux.Dentistries),
              }),
            }),
          ],
        }),
      });
  }
```

Таким чином, Redux реалізує централізоване управління станом, а Normalizr забезпечує нормалізовану структуру даних. Разом вони забезпечують масштабованість, структуровану архітектуру та ефективну роботу з великими обсягами інформації у клієнтській частині проєкту.

Отже, Клієнтська частина стоматологічної CRM-системи побудована на сучасному технологічному стеку, який забезпечує продуктивність, масштабованість та забезпечує користувацьку ефективність. Основою є Next.js, що поєднує React із серверним рендерингом та статичною генерацією сторінок, що забезпечує високу продуктивність інтерфейсу та його оптимізацію для пошукових систем. Для роботи з формами використовується комбінація Formik та Yup реалізує управління станом та забезпечує гнучку перевірку даних. Централізоване управління станом реалізовано через Redux, а асинхронна логіка та API-запити обробляються за допомогою Redux-saga, забезпечуючи чітке розділеня бізнес-логіки та інтерфейсу. Для організації залежностей між сервісами використовується Awilix, який забезпечує інверсію залежностей і гнучку архітектуру. Інтерфейс системи стилізується за допомогою TailwindCSS, який надає механізми для швидкого створення адаптивного дизайну, а багатомовність реалізована через i18next, що робить застосунок доступним для різних категорій користувачів.

Таким чином, клієнтська частина проєкту поєднує набір сучасних технологій, які функціонують у взаємопов’язаних інтеграції: від управління станом і асинхронними процесами до валідації форм, нормалізації даних та

стилізації інтерфейсу. Це забезпечує стабільну роботу системи, зручність для пацієнтів, лікарів та адміністративного персоналу, а також можливість для подальшого масштабування й розвитку проєкту.

3.3 Серверна частина проекту

Стоматологічна CRM-система реалізована як web-застосунок, що функціонує у браузері. У цій реалізації Next.js реалізує інтерфейс та забезпечує генерацію сторінок для користувача, а Nest.js виступає як серверний фреймворк, який реалізує бізнес-логіку, механізми безпеки та інтеграцію з базами даних. Backend характеризується власною структурою з визначеними особливостями (див. рис. 23).



Рисунок 23 – *Структура Backend-частини*

Одним із ключових елементів є каталог mirations. У цьому каталозі розміщаються файли, призначенні для міграції бази даних. До таких операцій належать створення, видалення та оновлення колонок, таблиць або структури бази даних. У наведеному коді (див. лістинг 8) продемонстровано структуру міграції для бази даних, зокрема на прикладі створення таблиці Workers.

Лістинг 8 – *Приклад файлу міграції*

```
  public async up(queryRunner: QueryRunner): Promise<void> {
    await queryRunner.query(`CREATE TABLE IF NOT EXISTS
workers (
    id INT PRIMARY KEY AUTO_INCREMENT,
    name VARCHAR(100) NOT NULL,
    surname VARCHAR(100) NOT NULL,
    middle_name VARCHAR(100) NOT NULL,
    birthday DATE NOT NULL,
    phone VARCHAR(30) NOT NULL,
    specialty_id INT NOT NULL,
    dentistry_id INT NOT NULL,
    login VARCHAR(100) NOT NULL,
    password VARCHAR(100) NOT NULL,
    created_at DATETIME NOT NULL DEFAULT CURRENT_TIMESTAMP,
    updated_at DATETIME NOT NULL DEFAULT CURRENT_TIMESTAMP ON
UPDATE CURRENT_TIMESTAMP,
    CONSTRAINT fk_worker_specialty FOREIGN KEY (specialty_id)
REFERENCES specialties(id) ON DELETE CASCADE ON UPDATE
CASCADE,
    CONSTRAINT fk_worker_clinic FOREIGN KEY (dentistry_id)
REFERENCES dental_clinics(id) ON DELETE CASCADE ON UPDATE
CASCADE
)
```

Ще одним важливим каталогом є seeds. Seeds – це спеціальні файли, призначені для заповнення бази даних конкретними даними. Це може бути як

тестові дані із даними, що не мають значення для контексту, так і зі змістом який повторює реальний стан бази даних. Також він може використовуватися для першої ініціалізації бази даних, наприклад, створення базового облікового запису для ініціалізації роботи стоматологічної клініки. У проєкті є дві категорії сидів, а саме тестові сіди та виробничі. Тестові сиди потрібні для перевірки усіх зв'язків та налаштувань бази даних, так як при певній кількості даних база даних або backend може функціонувати по різному, тестові сіди забезпечують перевірку коректності усіх зав'язків. А виробничі сиди використовуються для симуляції роботи стоматологічної клініки (див. лістинг 9).

Лістинг 9 – *Сід для генерації клієнтів*

```
export async function seedDentistries(dataSource: DataSource,numbersPoint=20) {
  const repo = dataSource.getRepository(Dentistry);


  const clinics: Dentistry[] = [];

  for (let i = 0; i <numbersPoint; i++) {
    const clinic = repo.create({
      street: faker.location.streetAddress(),
      city: faker.location.city(),
      region: faker.location.state(),
    });
    clinics.push(clinic);
  }

  await repo.save(clinics);
```

Наступним вже йде основні модулі backend частини (див. рис. 24). Тут зосереджено усі основі модулі проєкту, від ініціалізації сутності до зовнішніх запитів (наприклад надсилання повідомлень).



Рисунок 24 – *Модулі проєкту*

Кожен каталог сутності характеризується визначеною структурою, окрім сторонніх методів. Для кожної сутності передбачено такі файли як controller, service, module, entity (див. рисунок 25).

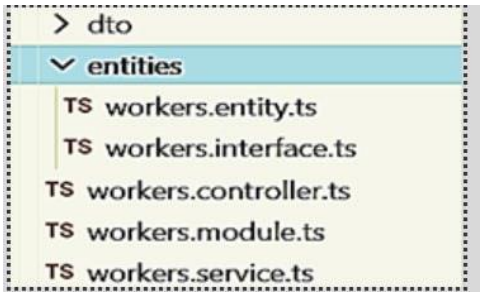


Рисунок 25 – *Структура сутності*

Кожен файл відповідає за певну частину виконання. Controller приймає http запити від користувачів, обробляє їх та передає їх у сервіс (див Лістинг 10).

Лістинг 10 – *Приклад методу контролера*

```
@Get('all/doctors')
  async GetDoctorsByDentistry(
    @Query('dentistry') dentistryId: number,
  ): Promise<IWorker[]> {
     return
this.workersService.GetDoctorsDentistry(dentistryId);
```

Сервіс відповідає за бізнес-логіку застосунку. Він отримує дані від контролера, виконує необхідні обчислення, перевірки або взаємодію з іншими компонентами системи, а потім повертає результат. Таким чином, сервіс ізолює логіку від контролера та забезпечує повторне використання коду (див. лістинг 11).

Лістинг 11 – *Приклад реалізації сервісу*

```
 public async findByLogin(login: string): Promise<Worker> {
    const worker = await this.workerRepo.findOne({
      where: { login },
      relations: [
        'specialty',
        'dentistry',
        'licenses',
        'appointments',
        'shifts',
      ],
    });
    return worker;
```

```
  }
```

Module функціонує як контейнер, що об’єднує всі складові певної сутності (controller, service, entity та інші залежності). Module визначає складові системи, що імпортуються та експортуються. Завдяки цьому забезпечується структурованість і модульність проєкту. Модуль функціонує як точка входу для інтеграції сутності в загальну архітектуру (див. лістинг 12).

Лістинг 12 – *Реалізація модуля для workers*

```
@Module({
  imports:   [TypeOrmModule.forFeature([Worker,   Specialty,
Dentistry])],
  controllers: [WorkersController],
  providers: [WorkersService],
  exports: [WorkersService],
})
export class WorkersModule {}
```

Entity визначає модель даних, яка відображає структуру таблиці в базі даних. Entity містить визначення полів, їхні типів та зв’язки з іншими сутностями. Entity застосовується у сервісах та контролерах для роботи з даними, що забезпечує узгодженість між програмною логікою та базою даних (див. лістинг 13).

Лістинг 13 – *Реалізація сутності Worker*

```
@Entity({ name: 'worke' })
export class Worker implements IWorker {
  @PrimaryGeneratedColumn()
  id: number;
  @Column({ type: 'varchar', length: 100, nullable: false })
```

```
  name: string;
  @Column({ type: 'varchar', length: 100, nullable: false })
  surname: string;
  @Column({ type: 'varchar', length: 100, nullable: false })
  middle_name: string;
  @Column({ type: 'date', nullable: false })
  birthday: Date;
  @Column({ type: 'varchar', length: 30, nullable: false })
  phone: string;
  @ManyToOne(() => Specialty, (specialty) =>
specialty.workers)
  @JoinColumn({ name: 'specialty_id' })
  specialty: Specialty;
  @ManyToOne(() => Dentistry, (dentistry) =>
dentistry.workers)
  @JoinColumn({ name: 'dentistry_id' })
  dentistry: Dentistry;
  @Column({ type: 'varchar', length: 100, nullable: false,
unique: true })
  login: string;
  @Column({ type: 'varchar', length: 100, nullable: false })
  password: string;
  @CreateDateColumn({ name: 'created_at' })
  created_at: Date;
  @UpdateDateColumn({ name: 'updated_at' })
  updated_at: Date;
  @Column({ type: 'boolean', default: true })
  active: boolean;
  @OneToMany(() => License, (license) => license.worker)
  licenses: License[];
  @OneToMany(() => WorkerShifts, (shift) => shift.worker)
  shifts: WorkerShifts[];
```

```
  @OneToMany(() => Appointment, (appointment) =>
appointment.dentist)
  appointments: Appointment[];
}
```

У масштабних проєктах з часом постає проблема перегляду існуючих API маршрутів. Wrapper використовується для вирішення цієї проблеми, забезпечуючи представлення API ендпоїнтів у структурованому та зрозумілому для користувача вигляді. Wrapper функціонує як обгортка для backend частини який забезпечує уніфікований інтерфейс із необхідною інформацією для виконання ендпойнтів (див. рис. 26).

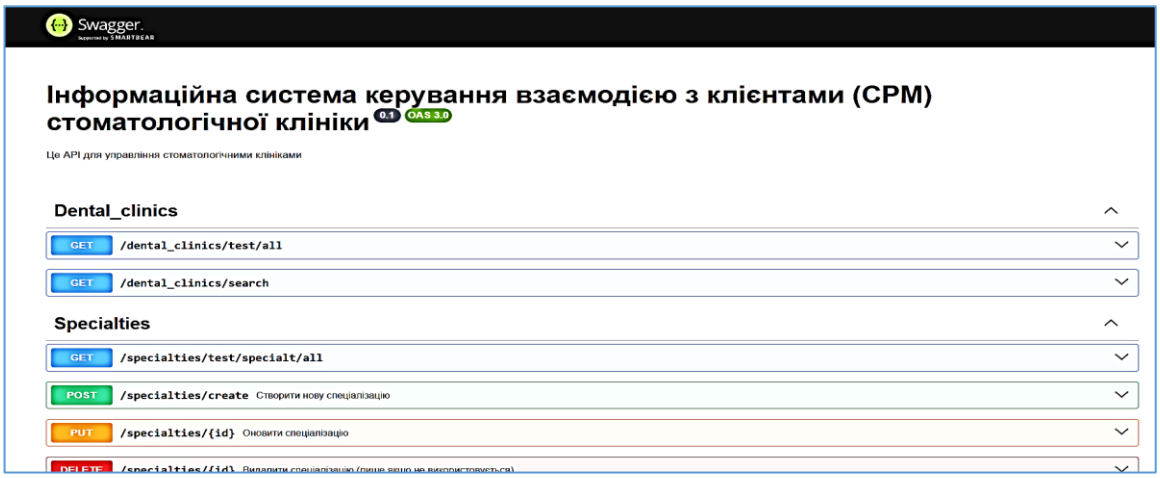


Рисунок 26 – *Wrapper обгортка для API endpoints*

Завдяки обгортці архітектура набуває модульності, а код характеризується спрощеною підтримкою та можливістю розширення.

Отже, Backend у стоматологічні CRM-системі функціонує як центральний компонент архітектури, який забезпечує стабільність, безпеку та продуктивність роботи застосунку. Backend виконує операції, що залишаються невидимими для користувача та є невід'ємною складовою проєкту. Backend забезпечує узгоджену роботу між всіма компонентами

системи, створюючи основу для взаємодії користувачів із застосунком та гарантує надійність і безпеку даних.

3.4 Використання за стосунку

Як зазначалося раніше, передбачено кілька ролей, а саме пацієнт, лікар, адміністрація та працівники реєстратури. Кожна з цих ролей має функціонал який співпадає із іншими ролями, так і такий який є унікальним для ролі.

3.4.1 Функціонал адміністрації

Для функціоналу адміністрації характерні такі дані як: звіти, статистика та менеджмент стоматологією. Для адміністрації було реалізовано 7 сторінок, зокрема такі сторінки як головна, календар, історія операцій, ліцензії, оплата, працівники, графік робочих змін (див. Рис. 27).

Головна сторінка. Головна сторінка демонструє загальну статистику по стоматології, зокрема майбутні операції, список неоплачуваних операцій, кількість вихідних для поточного місяця, загальна кількість операцій для лікаря та список лікарів які зараз проводять операцію.

Календар. На сторінці календар, демонструються записи для стоматології, а саме повна інформація про запис який включає опис попередніх дій, дата та час проведення, лікаря та пацієнта.

Сторінка оплати. На сторінці оплати показуються детальна інформація про оплату усіх операцій для певної стоматології. Сторінка оплати містить фільтр, за яким можна шукати певні операції та їхню оплати задля отримання інформації. Для кожної операції можна подивитися з чого складається ціна, а саме список виконаних дій та їхня вартість.

Працівники. На сторінці працівників, адміністрація має функціонал для додавання працівника, змінення існуючих облікових записів працівників та фільтрування списку працівників по декільком критеріїв.

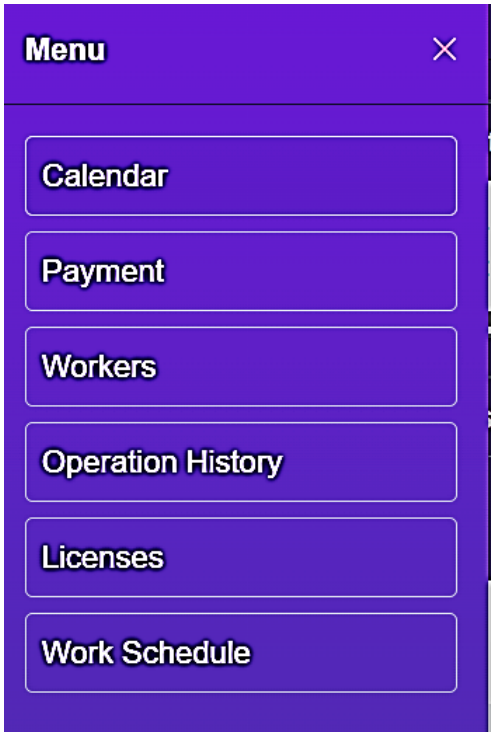


Рисунок 27 – *Список сторінок для адміністрації*

Історія операцій. Сторінка історії операції демонструються усі операції, які пов’язані із стоматологією. На сторінці історії операцій демонструються операції усіх станів, а саме ті які заплановані та виконані із можливістю фільтрування списку по декільком параметрів, зокрема пошук по ФІО клієнта чи працівника, спеціальності працівника, статус оплати та дати операції.

Ліцензії. Сторінка ліцензій демонструє ліцензії усіх працівників стоматології. Адміністратор має функціонал для перегляду інформації про працівника, зокрема перегляд, додавання та видалення ліцензії.

Графік робочих змін. Сторінка графіку робочих змін дозволяє адміністрації керувати змінами працівників. На сторінці демонструються основні дані працівника, а адміністратор вже має функціонал для додавання та видалення змін працівника.

3.4.2 Функціонал лікаря

Характерна риса функціоналу лікаря це ведення журналу операції. Для ролі лікаря, окрім його особливого функціоналу, доступний загальний функціонал, який базується на функціоналу адміністрації але із обмеженим доступом. Якщо адміністрація бачить статистику для усіх працівників стоматології, то лікар бачить лише ту статистику яка стосується його, або його пацієнта. Для ролі лікаря було реалізовано 6 сторінок, а саме головна, календар, оплата, історія операцій, ліцензії, графік робочих змін, операція (рис. 28).

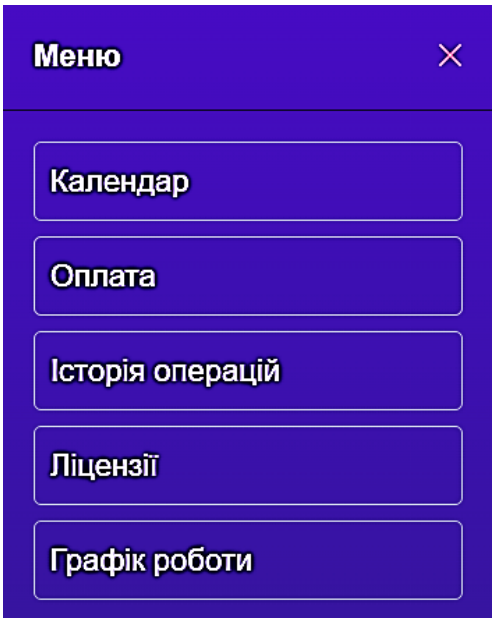


Рисунок 28 – *Список сторінок для лікаря*

Головна. Головна сторінка показує статистику але тільки для авторизованого лікаря На головній сторінці для лікаря немає списку лікарів які зараз проводять операцію, порівняно зі головною сторінкою для адміністрації, а інші списки які є на головній сторінці адміністрації стосуються лише авторизованого лікаря.

Календар. На сторінці календаря, лікар має функціонал для перегляду операцій із детальним описом, які стосуються авторизованого лікаря. Саме на сторінці календаря, лікар починає операцію і переходить до сторінки операції.

Сторінка операції. Сторінка операції має функціонал для додавання та видалення дій які були застосовані під час операції, переглядати історію операцій пацієнта та переглядати інформацію про пацієнта та обирати попередній спосіб оплати (див. рис. 29).

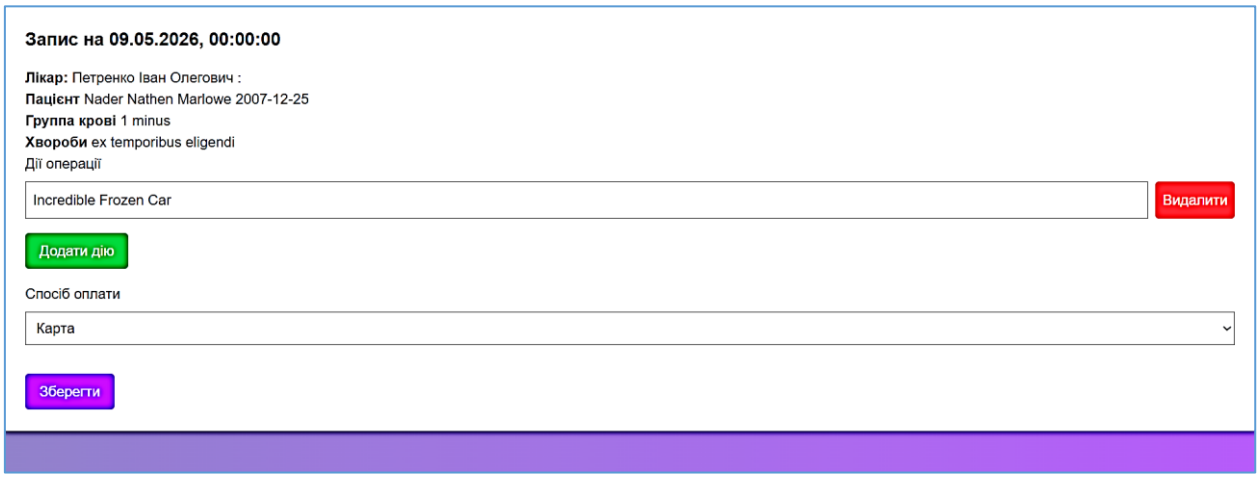


Рисунок 29 – *Сторінка операції лікаря*

Сторінка оплати. На сторінці оплати показуються детальна інформація про оплату усіх операцій для певної стоматології. Сторінка оплати містить фільтр, за яким можна шукати певні операції та їхню оплати задля отримання інформації. Для кожної операції можна подивитися з чого складається ціна, а саме список виконаних дій та їхня вартість. Характерною рисою цієї сторінки є вибірка даних, а саме демонстрація даних які стосуються лише авторизованого лікаря

Історія операцій. Сторінка історії операції демонструються усі операції, які пов’язані із лікарем. На сторінці історії операцій демонструються операції усіх станів, а саме ті які заплановані та виконані із можливістю фільтрування списку по декільком параметрів, зокрема пошук по ФІО клієнта чи працівника, спеціальності працівника, статус оплати та дати операції.

Ліцензії. Сторінка ліцензій демонструє ліцензії для лікаря стоматології. Лікар має функціонал для перегляду своєї інформації, зокрема перегляду ліцензії.

Графік робочих змін. Сторінка графіку робочих змін дозволяє лікарю переглядати список змін. На сторінці демонструються основні дані працівника та демонструє список змін, який можна фільтрувати по року та місяці.

3.4.3 Функціонал працівника реєстратури

Функціонал для реєстратури дещо схожий між функціоналом адміністрації та лікаря. Для реєстратури було реалізовано 5 сторінок, а саме головна, календар, оплата, працівники, історія операцій (див. Рис. 30).

Головна сторінка. Головна сторінка демонструє три таблиці, а саме лікарі які зараз оперують, пацієнти які не оплатили операцію та найближчі операції, ці таблиці показують дані які стосуються стоматології.

Календар. Сторінка календар має такий же функціонал що і адміністратор, але із розширеним функціоналом, а саме реєстратура має функціонал для додавання нового запису до лікаря та створювати обліковий запис пацієнта.

Оплата. Сторінка оплата демонструє операції які були проведені у стоматології. На сторінці є механізм фільтрування даних за декількома категоріями, а саме за даними працівника, сумою, статусом оплати, методом оплати, статусом запису та діапазоном дат. Реєстратура має функціонал для перегляду інформації про запис та виконаних дій.

Працівники. Сторінка працівників демонструє інформацію про працівника та має функціонал для фільтрування даних за такими параметрами як: повне ім’я працівника, спеціальності, датою народження.ч

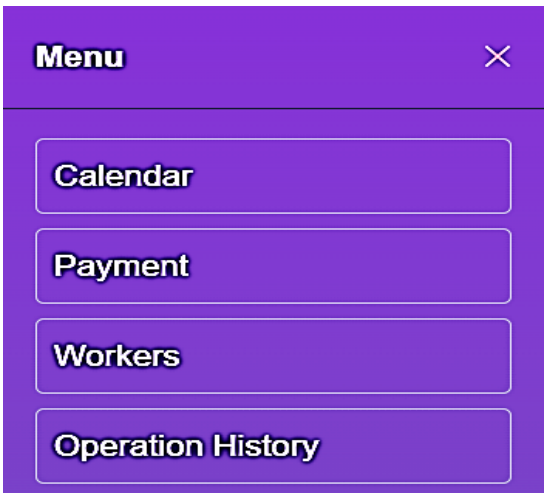


Рисунок 30 – *Список сторінок для реєстратури*

Історія операцій. Сторінка історії операцій розділений на 2 частини, а саме частина із фільтром для даних та таблицею для демонстрації даних. Фільтрація даних можлива за такими параметрами як: повне ім’я клієнта та працівника, спеціальності, статусу оплати та дати запису. У таблиці показується інформація про запис, також можна передивитися список виконаних дій для закінчених операцій, де описується стоїмось кожної дії

3.5 Висновки з розділу 3

У розділі описано програмну реалізацію та способи використання веб-застосунку. Зокрема, описано реалізовану структуру проєкту, особливості реалізації та функціонал для різних категорій користувачів. Описано особливості реалізації інтерфейсу та функціонал для таких ролей як адміністрація, лікарі, пацієнти та реєстратура. У розділі наведено демонстрацію коду, описано призначення коду та взаємодію різних частин реалізації між собою.

ВИСНОВКИ

1. Під час аналізу стоматологічної сфери було проаналізовано системи аналоги та на основі аналізу було сформовано вимоги до CRM-систем у медичній сфері.
2. Було сформовано функціональні, нефункціональні та технічні вимоги для стоматологічної CRM-системи, який призначений покращити взаємодію між пацієнтами та працівниками стоматологічної клініки.
3. Спроєктовано архітектуру CRM-системи, яка визначає основні компоненти та їхню взаємодію між собою. Розроблено архітектуру для різних частин застосунку, зокрема серверну, клієнтську та базу даних. Спроєктована архітектура визначає структуру яка забезпечує виконання поточних вимог із можливість подальшого розширення функціоналу системи.
4. Було реалізовано серверну частину застосунку, який забезпечує бізнес-логіку та обробку даних із можливістю подальшого розширення, зокрема додавання нових інструментів та додавання нового функціоналу.
5. Реалізовано клієнтську частину застосунку, який забезпечує інтуїтивно зрозумілий інтерфейс, орієнтований на різних типів користувачів із різним списком задач. Розроблено структуру, яка забезпечує технічні вимоги до клієнтської частини та забезпечує подальше розширення системи. Клієнтська частина забезпечує різним користувачам зручний досвід використання застосунку, зокрема багатомовність системи та адаптивний дизайн

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Fowler M. Patterns of Enterprise Application Architecture. – Boston: Addison-Wesley, 2002. 533 p.
2. Krasner G. E., Pope S. T. A Description of the Model-View-Controller User Interface Paradigm in the Smalltalk-80 System // Journal of Object-Oriented Programming. – 1988. – Vol. 1, No. 3. P. 26–49.
3. MySQL Documentation [Електронний ресурс]. – (c) Oracle Corporation. – Режим доступу: https://dev.mysql.com/doc/ (дата звернення: 04.2026).
4. Next.js Documentation [Електронний ресурс]. (c) Vercel Inc. – Режим доступу: https://nextjs.org/docs/app – (дата звернення: 04.2026).
5. NestJS Documentation [Електронний ресурс]. (c) Kamil Myśliwiec та NestJS Contributors. Режим доступу: https://docs.nestjs.com/controllers (дата звернення: 04.2026).
6. Dentrix Ascend. Henry Schein One. Режим доступу: https://www.dentrixascend.com/ (дата звернення 10.03.2026)
7. Curve Dental. Curve Dental, Inc Режим доступу: https://www.curvedental.com/ (дата звернення 10.03.2026)
8. Безвехий А.І., Нахман А.В. Розроблення інтегрованої CRM-системидля автоматизації управління стоматологічною клінікою. Зб. праць Міжнародної мультидисциплінарної наукової Інтернет-конференції «Світ наукових досліджень», секція 2 Інформаційні системи і технології. Вип. 50, 2026. С. URL: https://www.economy-confer.com.ua/full-article/6771/ (дата звернення 08.05.2026)
9. Wojciechowski, M. Clean Architecture and Domain-Driven Design [Електронний ресурс]. - 2025. - Режим доступу: https://wojciechowski.app/en/articles/clean-architecture-domain-driven-design-2025 (дата звернення: 09.05.2026).
10. Fuzen. Common Dental CRM Mistakes in Patient Follow-Ups and How Clinics Fix Them [Електронний ресурс]. - 2026. - Режим доступу:

https://www.fuzen.io/posts/common-dental-crm-mistakes-in-patient-follow-ups-and-how-clinics-fix-them (дата звернення: 09.05.2026).

11. TypeORM. Official Documentation [Електронний ресурс]. - Режим доступу: https://typeorm.io (дата звернення: 09.05.2026).
12. Fuzen. Best CRM for Dentists 2026: Compare Dental CRM Software [Електронний ресурс]. - 2026. - Режим доступу: https://www.fuzen.io/posts/best-crm-for-dentists-2026-compare-dental-crm-software (дата звернення: 09.05.2026)
13. Mohit Mittal. The Transformative Role of CRM Systems in Modern Healthcare [Електронний ресурс]. – PhilArchive, 2023. – Режим доступу: https://philarchive.org/archive/MOHTTR (дата звернення: 09.05.2026).
14. Parekh, Limesh. Cracking the CRM Code: How to Prevent Failures in Buying, Implementing, and Using CRM. – India: Orange Publishers, 2021. – 210 с.
15. Fatouretchi, Max. The Art of CRM: Proven Strategies for Modern Customer Relationship Management. – London: Routledge, 2020. – 280 с.
16. VGD Technologies. NestJS with Next.js: Building a Powerful Full-Stack App [Електронний ресурс]. – LinkedIn, 2025. – Режим доступу: https://www.linkedin.com/pulse/nestjs-nextjs-building-powerful-full-stack-app-2025-byhvf (дата звернення: 09.05.2026).
17. GeeksforGeeks. How Web Works – Web Application Architecture for Beginners [Електронний ресурс]. – 2025. – Режим доступу: https://www.geeksforgeeks.org/blogs/how-web-works-web-application-architecture-for-beginners/ (дата звернення: 09.05.2026).
18. Springer. Guide to Web Application and Platform Architectures. – Berlin: Springer, 2025. – 412 с.
19. Нахман А. Розроблення інтегрованої CRM-системи для автоматизації управління стоматологічною клінікою. *Збірник наукових праць студентів, здобувачів PhD, докторантів і молодих вчених, «МОЛОДА НАУКА-2026» Том V «Секції Інженерного навчально-наукового інституту ім. Ю.М. Потебні ЗНУ*. Запоріжжя, 2026. С. 108–109.

Декларація
академічної доброчесності
здобувача ступеня вищої освіти ЗНУ

Я, Нахман Артем Володимирович, студент 4 курсу, форми навчання денної, Інженерного навчально-наукового інституту, спеціальності 121 Інженерія програмного забезпечення, адреса електронної пошти se1212-15@stu.zsea.edu.ua, – підтверджую, що написана мною кваліфікаційна робота на тему «Розроблення інтегрованої CRM-системи для автоматизації управління стоматологічною клінікою» відповідає вимогам академічної доброчесності та не містить порушень, що визначені у ст.42 Закону України «Про освіту», зі змістом яких ознайомлений.

- Заявляю, що надана мною для перевірки електронна версія роботи є ідентичною її друкованій версії

- Згоден на перевірку моєї роботи на відповідність критеріям академічної доброчесності у будь-який спосіб, у тому числі а допомогою інтернет-системи, а також на архівування моєї роботи в базі даних цієї системи.

Дата 07.06.2026 _____ Нахман А.В.
(підпис) (студент)

Дата 07.06.2026 _____ Безверхий А.І.
(підпис) (керівник)