

Факультет комп'ютерних наук і технологій

(повне найменування інституту, факультету)

Кафедра системного аналізу та обчислювальної математики

(повне найменування кафедри)

Пояснювальна записка

до дипломного проекту (роботи)

бакалавра

(ступінь вищої освіти)

на тему Розробка CRM системи для стоматологічної клініки

Виконав: студент(ка) 4 курсу, групи КНТ-827-СП

Спеціальності 124 Системний аналіз

(код і найменування спеціальності)

Освітня програма (спеціалізація)

Інтелектуальні технології та прийняття рішень в складних системах

Білий В.В.

(прізвище та ініціали)

Керівник

Денисенко О.І.

(прізвище та ініціали)

Рецензент

Денисенко О.І.

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»
(повне найменування закладу вищої освіти)

Інститут, факультет Комп'ютерних наук і технологій
Кафедра Системного аналізу та обчислювальної математики
Ступінь вищої освіти бакалавр
Спеціальність 124 Системний аналіз
(код і найменування)
Освітня програма (спеціалізація) Інтелектуальні технології та прийняття рішень в складних системах
(назва освітньої програми (спеціалізації))

ЗАТВЕРДЖУЮ

Завідувач кафедри проф. Г.В. Корніч

« 29 » травня 20 20 року

ЗАВДАННЯ
НА ДИПЛОМНИЙ ПРОЕКТ (РОБОТУ) СТУДЕНТА(КИ)

Білого Віталія Вікторовича

(прізвище, ім'я, по батькові)

1. Тема проекту (роботи) Розробка CRM системи для стоматологічної клініки

керівник проекту (роботи) Денисенко Олександр Іванович, к. т. н., доц.

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

затверджені наказом закладу вищої освіти від « 19 » травня 20 20 року № 136

2. Строк подання студентом проекту (роботи) _____

3. Вихідні дані до проекту (роботи) статті, що стосуються актуальності роботи і альтернативних підходів до її реалізації, перелік сучасних фреймворків для web-розробки

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити) аналіз предметної області та постановка задачі, вибір програмних засобів з урахуванням вимог до системи, проектування архітектури та розробка інтерфейсу користувача, програмна реалізація та тестування модулів системи

5. Перелік графічного матеріалу (зточним зазначенням обов'язкових креслень)

6. Консультанти розділів

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	прий викон завдан
1	Денисенко О.І.	2 ДБ	
2	Денисенко О.І.	ДБ	
3	Денисенко О.І.	ДБ	
4	Денисенко О.І.	ДБ	
5	Денисенко О.І.	ДБ	
Нормативна	Широкорад Д.В.		ДБ

7. Дата видачі завдання « 15 » листопада 2020 року.

КАЛЕНДАРНИЙ ПЛАН

[illegible]

Студент(ка)

Керівник проекту (роботи)

Білий В.
(прізвище та ініціали)

Денисенко
(прізвище та ініціали)

РЕФЕРАТ

ПЗ: 134 с., 38 рис., 20 джерел.

Об'єкт дослідження – система управління взаємодією з клієнтами (CRM-система) стоматологічної клініки.

Предмет дослідження – програмне забезпечення CRM-систем.

Мета роботи – створення спеціалізованої системи для підвищення ефективності роботи персоналу стоматології за рахунок оптимізації процесів взаємодії з клієнтами.

Під час виконання бакалаврської дипломної роботи було проаналізовано доцільність розробки, розглянуто існуючі аналоги та виявлено їх недоліки. Проведено варіантний аналіз засобів розробки, спроектовано структуру та моделі системи, розроблено програмні модулі системи.

Актуальність роботи обумовлена необхідністю автоматизації рутинних завдань з метою економії часу працівників, а також створенням єдиної інформаційної бази, доступ до якої можна отримати з будь-якого пристрою, підключеного до мережі Інтернет.

CRM, ВЕБ-ДОДАТОК, БАЗА ДАНИХ, SPA, СКРИПТ, HTML, CSS, MYSQL, LARAVEL

ЗМІСТ

Вступ.....	7
1 Аналіз завдання та визначення методу його вирішення.....	9
1.1 Мета, актуальність дипломної роботи	9
1.2 Огляд літератури	10
1.3 Аналіз існуючих програмних продуктів, що вирішують подібні завдання	11
1.4 Постановка задачі розробки	16
2 Вибір засобів розробки програмного продукту.....	18
2.1 Аналіз особливостей роботи веб-додатку.....	18
2.2 Огляд фреймворків для розробки системи	21
2.2.1 Вибір фреймворку для розробки серверної частини.....	22
2.2.2 Вибір фреймворку для розробки клієнтської частини	25
3 Проектування архітектури системи	29
3.1 Проектування логічної та фізичної структури сайту	29
3.2 Проектування діаграм на мові uml	32
3.2.1 Діаграма варіантів використання	32
3.2.2 Діаграми послідовності.....	37
3.3 Проектування API	39
3.4 Проектування структури інтерфейсу	47
3.5 Проектування бази даних	50
4 Програмна реалізація модулів системи	57
4.1 Налаштування підключення до БД	57
4.2 Налаштування міграцій бази даних.....	60

4.3 Розробка основних модулів системи.....	62
4.3.1 Розробка модуля авторизації	62
4.3.2 Розробка адміністративної частини	67
4.3.3 Модуль пацієнтів.....	70
4.3.4 Модуль реєстратури.....	75
4.3.5 Модуль аналітики.....	76
5 Тестування розробленої системи	78
Висновки.....	82
Перелік посилань	83
Додаток А Код серверної частини	85
Додаток Б Код клієнтської частини	105

ВСТУП

З розвитком мережі Інтернет широкого використання набувають онлайн сервіси, основну частину з яких складають системи управління для різного роду бізнесу та комерції. Вже стає складно уявити собі складський або бухгалтерський облік без застосування спеціалізованого програмного забезпечення, торгові представники використовують спеціальні програми для оформлення і відправлення замовлення в офіс прямо з планшета або мобільного телефону, достатньо велика частина замовлень приходить з сайту вже у вигляді готових до обробки документів. Але при цьому взаємини з клієнтами, принаймні, в середньому і малому бізнесі, чомусь дуже часто ведуться без впровадження автоматизації і достатньої уваги до обліку.

Основною стратегією успішного існування та подальшого розвитку сучасних медичних установ, нарівні з компаніями в інших сферах діяльності, поступово стає ефективне управління взаємодією з клієнтами. Орієнтація на удосконалення відносин з клієнтами обумовлена низкою тенденцій, зокрема посиленням конкуренції і відповідним підвищенням вимог до якості медичного обслуговування [1]. Знання своїх клієнтів, задоволення запитів і потреб кожного з них дозволить в значній мірі підвищити загальну ефективність діяльності стоматологічної клініки.

Одним з ефективних засобів вирішення описаних проблем є впровадження системи управління взаємодією з клієнтами, або як її часто називають CRM-системи. CRM - це прикладне програмне забезпечення, призначене для автоматизації стратегій взаємодії з замовниками (клієнтами), зокрема, для підвищення рівня продажів і якості обслуговування шляхом збереження інформації про клієнтів та історії взаємин з ними [2].

Навіть якщо компанія веде історію дзвінків і контактів на папері або в Excel - це можна вважати CRM-системою в тому випадку, якщо розроблена схема обліку працює і дозволяє контролювати всі варіанти взаємодії з клієнтами. Звичайно, такі методи відходять у минуле, адже в сучасному світі без ефективної автоматизації

складно уявити роботу будь-якого бізнесу. А тому, коли говорять про CRM-системи, то мають на увазі спеціальне програмне забезпечення [3].

CRM-системи, як правило, складаються з окремих модулів. Це дозволяє змінювати їх конфігурацію в залежності від особливостей організації, а також забезпечити необхідний функціонал із можливістю подальшого додавання/видалення модулів. Через це уявити склад CRM-системи доволі непросто, оскільки конкретні варіанти систем істотно відрізняються один від одного. Проте існують ключові функціональні області, які є базовими для систем, що використовуються у сфері медичного обслуговування:

- аналітичні й управлінські компоненти - модулі та засоби ведення управлінського обліку, інструменти аналізу якості та ефективності медичних послуг. Ці складові дозволяють проаналізувати стан медичної організації, виявити проблемні місця й оптимізувати бізнес-процеси. Результати аналізу можна вивести на екран у вигляді графіків або таблиць.

- медичні компоненти - усі модулі, пов'язані з реєстрацією пацієнтів, ведення реєстру електронних медичних карт, облік лікарняних листів, ведення протоколів лікування, медична статистика й аналітика, історія хвороби та багато іншого.

- фінансово-економічні компоненти - до них відносяться інструменти для проведення економічного аналізу діяльності організації та розрахунку собівартості лікування і тарифів на надання медичних послуг.

- загально-технічні компоненти - відповідають за контроль доступу користувачів і захист бази даних, а також підтримка можливостей інтеграції з іншими системами і програмами [4].

1 АНАЛІЗ ЗАВДАННЯ ТА ВИЗНАЧЕННЯ МЕТОДУ ЙОГО ВИРІШЕННЯ

1.1 Мета, актуальність дипломної роботи

Основною метою дипломної роботи є розробка системи управління взаємодією з клієнтами, що дозволить підвищити ефективність роботи стоматології за рахунок автоматизації процесів обліку пацієнтів. Передбачається, що впровадження цієї системи в структуру клініки надасть ряд переваг, серед яких можна виділити:

- зменшення витрат: автоматизація складних обчислень і інших операцій, зменшення зайвої комунікації між співробітниками;
- прискорення роботи: спрощення операцій обробки даних і відсутність потреби у виконанні рутинних завдань;
- підвищення прозорості: дані по роботі всіх співробітників заносяться в систему, є можливість робити на підставі цих даних звіти керівництву для розуміння ситуації на місцях;
- управління на основі KPI: для кожного процесу можна виділити показники ефективності (метрики) і почати їх відстежувати за допомогою CRM;
- стандартизація внутрішніх процесів: всі співробітники працюють за певною схемою, яка закладена в CRM-систему.

Для досягнення поставленої мети були визначені наступні задачі:

- провести аналіз предметної області;
- оцінити можливості існуючих рішень для управління стоматологічними клініками;
- проаналізувати сучасні методи веб-розробки;
- спроектувати логічну та фізичну структури системи;
- розробити структуру інтерфейсу;
- спроектувати базу даних;

- розробити програмні модулі, що реалізують усі необхідні в системі функції;
- провести тестування та виправлення помилок.

На підставі визначених задач було побудовано загальне дерево цілей (ARIS Objective diagram) для розробки CRM-системи (рис. 1.1).

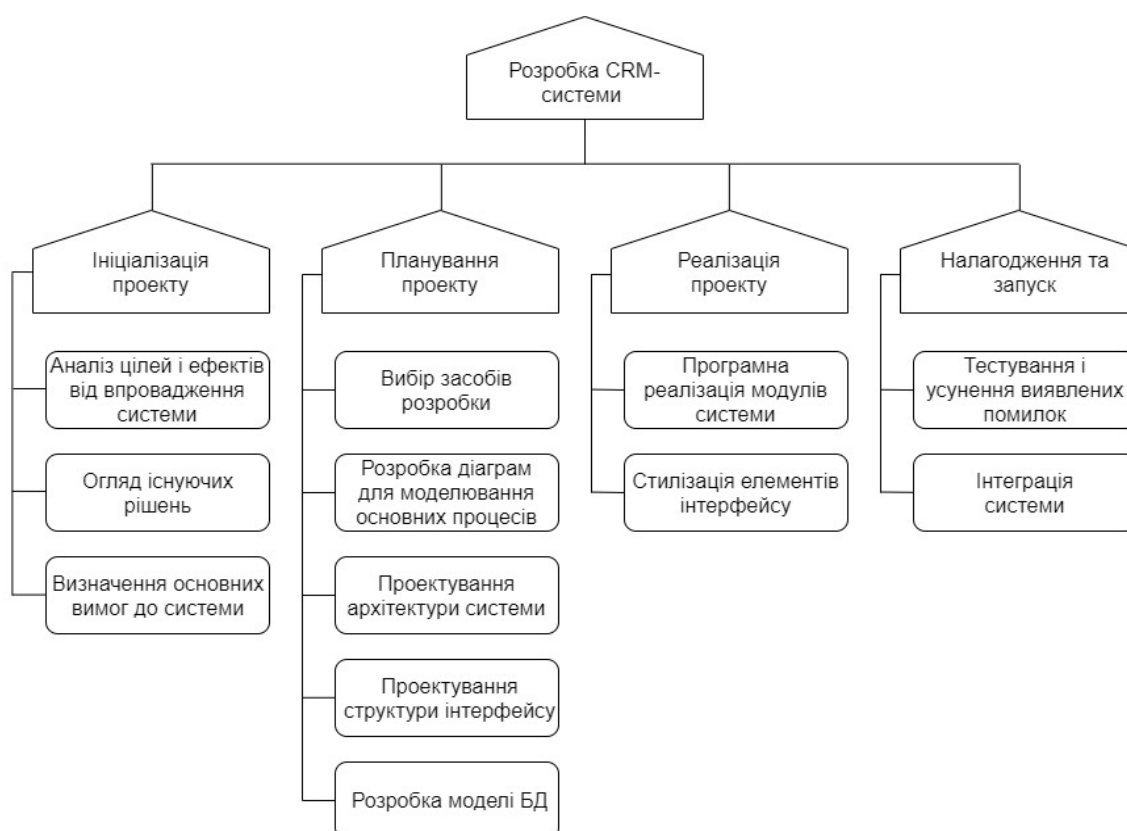


Рисунок 1.1 – Цілі дипломної роботи

1.2 Огляд літератури

Для проектування і реалізації цього проекту були розглянуті різні технології: SPA, фреймворки для написання серверної та клієнтської частин веб-додатку (Django, Laravel, Vue.js) тощо. Також були отримані деякі знання відносно підходів по розробці баз даних. Нижче наведені основні джерела інформації, що були використані в ході написання дипломної роботи:

- статті присвячені особливостям використання CRM-систем і перевагам від їх впровадження [1, 2].
- статті, що стосуються актуальності роботи і альтернативних підходів до її реалізації [3, 4, 5].
- статті, що описують принципи роботи і способи реалізації сучасних веб-додатків [6, 7, 8].
- офіційна документація та інші керівництва по використанню веб-фреймворков [9, 10, 11, 12, 13, 14].
- керівництво, що описує методи нормалізації і основні принципи проектування архітектури реляційних баз даних [17].

З розглянутого переліку літератури можна відзначити статті, в яких проаналізовані сучасні передумови для імплементації CRM та даються практичні рекомендації щодо побудови ефективної системи. Наведені в цих роботах наукові положення, результати досліджень та висновки підкреслюють доцільність застосування допоміжного ПЗ для управління взаємодією з клієнтами, а також актуальність роботи по створенню власної CRM-системи.

Матеріал, викладений в статтях про сучасні фреймворки дозволив реалістично оцінити і порівняти їх характеристики, що дало можливість обрати оптимальний інструмент для вирішення поставленого завдання.

Також варто згадати про керівництво по нормалізації БД. В ньому описуються так звані нормальні форми, що представляють собою список вимог, яким повинна відповідати база даних і показують як правильно проектувати таблиці для зберігання великої кількості інформації. Застосування цих підходів дозволило спростити ведення та підтримку бази даних і зробити її більш надійною.

1.3 Аналіз існуючих програмних продуктів, що вирішують подібні завдання

Для визначення вимог до розроблюваної системи був проведений огляд вже створених рішень, у ході якого дається короткий опис існуючих CRM-систем для

стоматологічних клінік, виконується аналіз методів реалізації подібного програмного забезпечення. Результати отримані шляхом пошуку аналогічних систем в мережі Інтернет та дослідження наданих ними можливостей будуть використані при розробці власного програмного продукту.

Розглянемо найбільш популярні CRM-системи:

iStom CRM – пропонує своїм користувачам електронний розклад прийому лікарів, автоматичний розрахунок стоматологічного плану лікування, інтеграцію з онлайн-касою і рентгенівськими системами, різні аналітичні інструменти. Даний продукт встановлюється локально на комп'ютери користувачів, що накладає певні обмеження при роботі з системою.

The screenshot displays the 'Карта пациента' (Patient Card) for Borisov Alexey Yuryevich. The interface includes fields for personal data, contact information, and medical history. Below the patient card, there is a section for 'Наряд' (Appointment) with a table showing the schedule for 17.02.2014. The table has columns for 'Вид работ' (Type of work), 'Дата' (Date), and 'Время' (Time). The schedule includes 'Начало работы' (Start of work) at 16:00, followed by 'Примерка №1' through 'Примерка №4' (Try-on #1 through #4), and 'Готовая работа' (Ready work). At the bottom, there is a legend for 'К' (Crown), 'Вкл' (Inlay), 'Внр' (Veneer), and 'О' (Absent).

Вид работ	Дата	Время
Начало работы	17.02.2014	16:00
Примерка №1		
Примерка №2		
Примерка №3		
Примерка №4		
Готовая работа		

Рисунок 1.2 – Интерфейс системы «iStom»

Denta Pro – це модульне CRM-рішення. Дозволяє розширити список базових функцій (онлайн-реєстратура, контроль фінансів, ведення клієнтської бази, облік матеріалів), додатковими модулями для роботи зі страховими компаніями,

лабораторією, тощо. До основних переваг системи можна віднести інформативну картку пацієнта і простий інтерфейс. Також є можливість використання спеціалізованого мобільного додатка для доступу до системи.

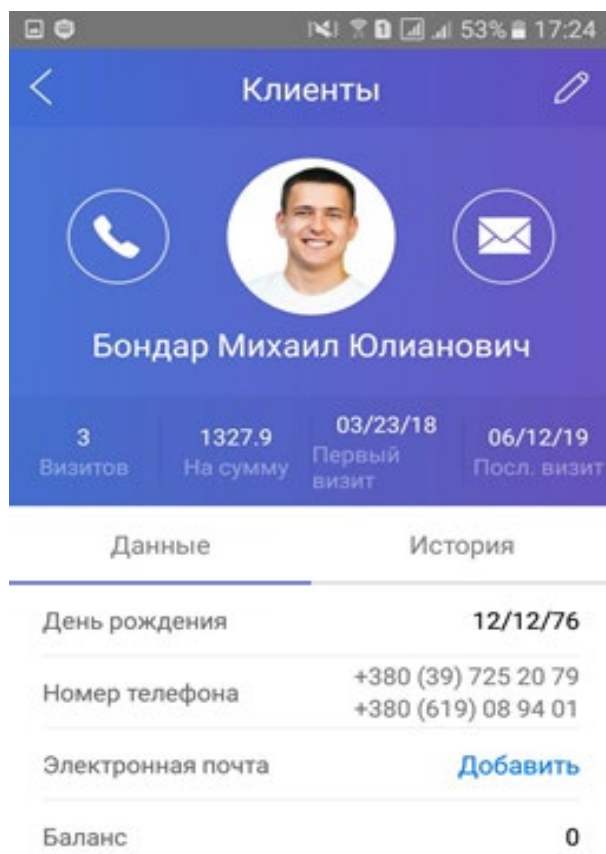


Рисунок 1.3 – Мобільний додаток системи «Denta Pro»

MedicalCRM – програмне забезпечення для автоматизації роботи медичних організацій. Добре підходить для застосування в стоматологічній галузі, так як дозволяє автоматизувати розрахунок вартості візиту і процеси формування рахунків, вести облік по філіям, вивантажувати з CRM в Excel інформацію для архіву чи аналізу, здійснювати облік проданих товарів і послуг, контролювати взаємодію з лабораторією та інше. Істотним недоліком є необхідність доплачувати за нових користувачів системи.

Ольга Сергеевна Костина [3]							
Информация о клиенте			История записей		Программы		
Всего визитов		Общая сумма заказов		Сумма услуг	Сумма товаров	Задолженность	Депозит
32		77552		77552	.00	0	5180
Дата	Прог.	A-з	Кол	Услуга/Товар	Сотрудник	Сумма	Акция
23/11/2018 15:00			1	Выезд на дом	Сидорова Г.П.	1300.00	
23/11/2018 11:00	■		1	Капельница	Федорова И.А.	.00	
23/11/2018 09:00			1	Подбор очков	Сидорова Г.П.	500.00	
21/11/2018 11:00	■		1	Массаж	Сидорова Г.П.	20.00	
14/08/2018 08:00	■		1	Первичный прием	Сидорова Г.П.	10.00	Первая акция
08/06/2018 10:30			1	Вторичный прием	Сидорова Г.П.	800.00	
05/06/2018 08:00			1	Вторичный прием	Сидорова Г.П.	800.00	
25/11/2017 13:00			1	Общий анализ крови	Толстов М.	500.00	
18/09/2017 10:00			1	Вторичный прием	Сидорова Г.П.	800.00	

Рисунок 1.4 – Интерфейс системы «MedicalCRM»

2V: Стоматологія – універсальна CRM-система, яка може використовуватися як в невеликій стоматологічній клініці, так і в мережевих компаніях. Серед особливостей даного програмного рішення можна виділити хмарне зберігання даних і практично необмежене масштабування. Система надає наступні можливості:

- робота з онлайн-касою і фіскальним реєстратором;
- гнучке налаштування прав доступу для співробітників;
- наявність функції запису на прийом через сайт;
- автоматичне заповнення і друк документів;
- облік витрати матеріалів;
- можливість інтеграції з програмою «1С: Бухгалтерія».

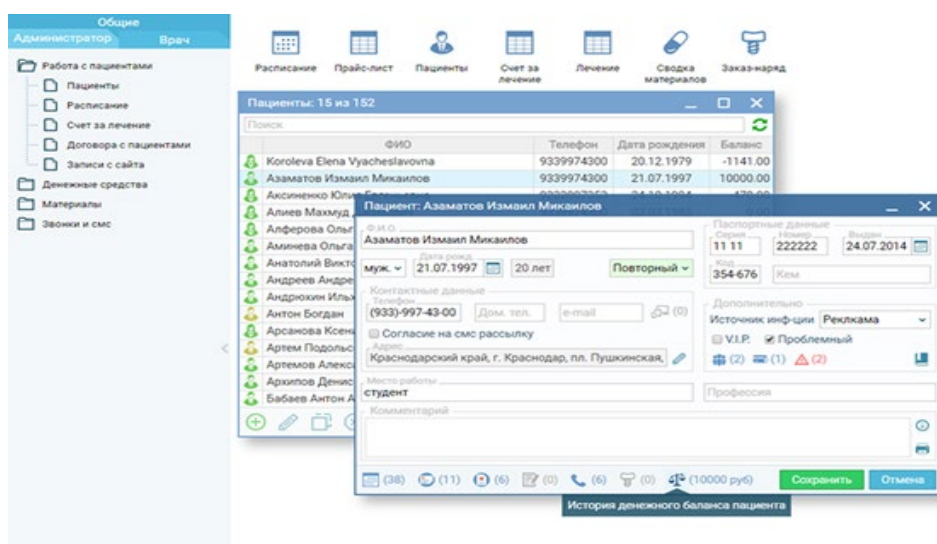


Рисунок 1.5 – Интерфейс системы «2V: Стоматология»

БІТ.Стоматологія – містить наступний набір інструментів для управління стоматологічною клінікою:

- блок реєстрації для рівномірного запису до фахівців;
- блок робочого місця лікаря для ведення стоматологічної карти пацієнта;
- шаблони документів і протоколів огляду;
- блок звітності для керівників.

Варто також згадати про можливість використання обмеженої версії цієї CRM безкоштовно [5].

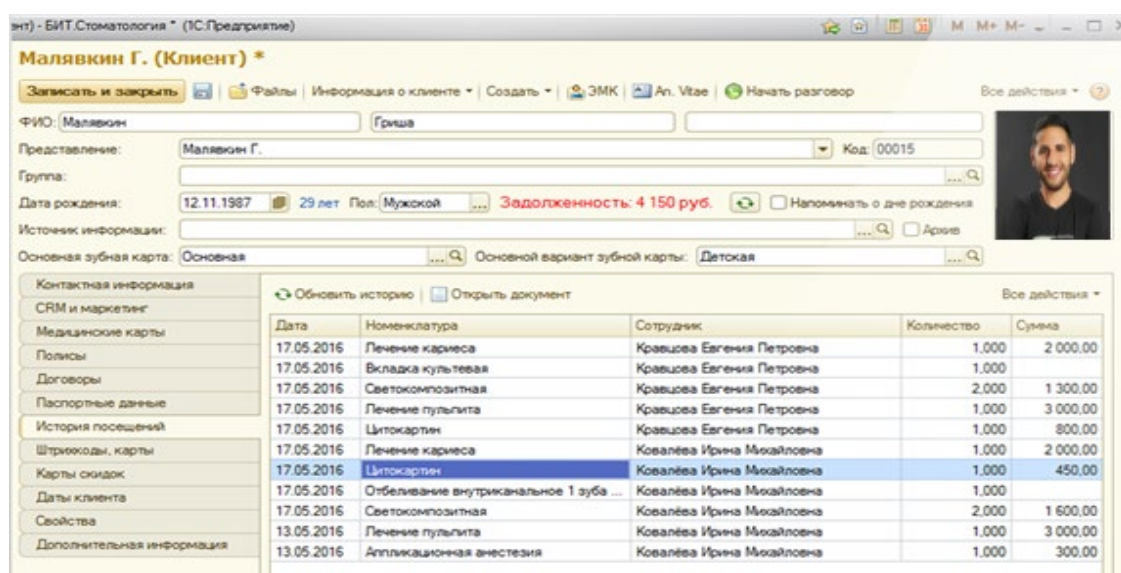


Рисунок 1.6 – Інтерфейс системи «БІТ.Стоматологія»

Розглянутий список систем є незначною частиною представлених на ринку продуктів, кожен з яких відрізняється набором доступних можливостей і способом реалізації. Деякі розробники представили свої системи у вигляді desktop-додатків, а інші — web-додатків. Відмінність першого варіанту полягає в тому, що всі обчислення виконуються на локальному комп'ютері, а в разі використання інтернет-додатку — на сервері. Локальні обчислення будуть перевагою, так як навантаження на сервер в цьому випадку помітно нижче, особливо при великій кількості користувачів. Але при бажанні зайти в систему не з робочого комп'ютера, а, наприклад, з дому, виникає необхідність інсталиувати програму в операційну систему. У той же час веб-додаток доступний з будь-якого пристрою, на якому

встановлений браузер (навіть з мобільного телефону). Також слід зазначити, що процес переходу на нові версії для інтернет-додатків значно легший (проста заміна файлів на сервері), а для настільних — доведеться вручну оновлювати програму на кожному комп'ютері.

Після проведеного аналізу можна зробити висновок, що більшість існуючих CRM-систем складні в використанні і мають зайві модулі та функції, застосування яких не передбачається в умовах клініки, а наявність цих модулів впливає на кінцеву вартість продукту. Слід додати, що таке програмне забезпечення потребує додаткових налаштувань та адаптації до специфіки стоматологічної клініки. Виходячи з цього, доцільно буде розробити власну систему враховуючи побажання та рекомендації практикуючих стоматологів. Подібна CRM-система в першу чергу повинна бути орієнтована на роботу з інформацією про пацієнтів, тому процес додавання нових записів треба зробити максимально простим, але при цьому інформативним в плані кількості полів для заповнення. Також варто подбати про зручність пошуку і перегляду інформації. Для забезпечення комфорту при роботі з системою варто враховувати наступне: записи і дані мають бути добре структуровані, система повинна володіти інтуїтивно зрозумілим інтерфейсом. Від цього безпосередньо залежить те, наскільки система проста або складна в використанні.

1.4 Постановка задачі розробки

На підставі проведеного огляду аналогічних програмних продуктів були сформовані основні вимоги до можливостей, що мають бути реалізовані в розробленій системі, а саме:

- наявність інтуїтивно зрозумілого інтерфейсу, що дозволить співробітникам стоматології швидко освоїти можливості системи;

- забезпечення доступу до ресурсу через Інтернет, для чого реалізувати систему у вигляді web-додатку;
- забезпечення надійного захисту особистих даних персоналу та пацієнтів клініки;
- розмежування прав для користувачів системи (доступ до повної клієнтської бази та фінансових результатів діяльності клініки обмежений для звичайних співробітників);
- наявність зручного календаря, який дозволить здійснити запис кожного пацієнта з урахуванням зайнятості лікаря;
- забезпечення швидкого доступу до розгорнутої інформації про пацієнтів: анамнез, історія лікування;
- забезпечення можливості лікарям робити записи стосовно прийому пацієнтів;
- видача інформації про отримані аванси та неоплачені заборгованості по кожному пацієнту;
- збір та наглядне зображення статистики щодо роботи клініки (кількість візитів, сума сплат та інше) за певний проміжок часу.

Перераховані можливості забезпечать комфортне використання системи і дозволять їй виконувати свою основну функцію — підвищення ефективності роботи клініки.

2 ВИБІР ЗАСОБІВ РОЗРОБКИ ПРОГРАМНОГО ПРОДУКТУ

2.1 Аналіз особливостей роботи веб-додатку

Так як розроблювана система управління взаємодією з клієнтами розгортається в мережі Інтернет, вона буде являти собою своєрідний веб-додаток. Програмне забезпечення такого роду функціонує за принципами «клієнт-серверної» архітектури, де в якості клієнта виступає браузер, а сервера – web-сервер. Особливість даного підходу полягає в тому, що сам додаток знаходиться і виконується на сервері – клієнт при цьому отримує тільки результати роботи. Основним завданням останнього є коректне відображення завантаженої з сервера інформації, а також відправка на сервер запитів і даних користувача [6]. Передача запитів і результатів їх обробки відбувається через Інтернет (рис. 2.1).



Рисунок 2.1 – Принцип функціонування веб-додатків [7]

За способами розробки веб-додатки розділяють на два основних типи: односторінкові (SPA) та багаторічкові (MPA). Розглянемо відмінності між ними, а також переваги і недоліки кожного типу.

Односторінкові додатки дозволяють імітувати роботу настільних програм, так як вони реагують на дії користувача без затримки. Концепція SPA полягає в

тому, що всі ресурси, необхідні для роботи (елементи CSS, скрипти та ін.), знаходяться на одній сторінці і завантажуються при ініціалізації. При такому підході немає необхідності в перезавантаженні або завантаженні додаткових сторінок – контент оновлюється динамічно за допомогою JavaScript (AJAX-запитів). На рисунку 2.2 наведено принцип роботи односторінкового додатку.

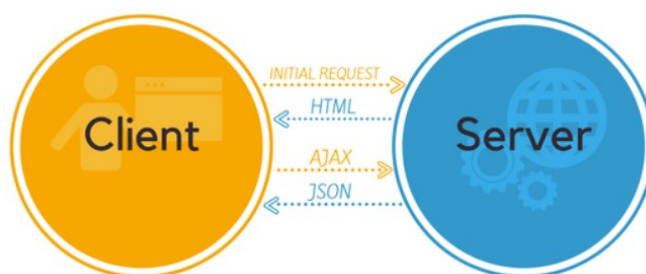


Рисунок 2.2 – Принцип роботи односторінкового додатку [8]

Переваги:

- при роботі завантажуються тільки необхідні дані (без повторюваних елементів і блоків), що дозволяє розвантажити ресурси сервера;
- відсутність повного перезавантаження сторінки підвищує швидкість роботи.
- код серверної частини (бекенду) можна повторно використати для створення власної мобільної програми.
- концепція SPA дозволяє ефективно кешувати будь-які дані. Програма відправляє всього один запит, збирає дані, а після цього може функціонувати в offline-режимі.

Недоліки:

- початкове завантаження триває досить довго, через великий розмір JavaScript коду і допоміжних бібліотек.
- для коректної роботи потрібен JavaScript. Якщо користувач вручну відключить JavaScript в браузері, він не зможе повноцінно використовувати весь функціонал програми.
- через особливості архітектури більшість сторінок просто недоступні для

сканування пошуковими роботами, що ускладнює SEO-оптимізацію.

Багатосторінкові додатки працюють за традиційною схемою. Це означає, що при незначній зміні даних чи переході на іншу сторінку браузер робить новий запит до сервера і знову завантажує всі ресурси, навіть ті компоненти, які повторюються на всіх сторінках (заголовок, нижній колонтитул, тощо). Таким чином, продуктивність витрачається на завантаження тих самих елементів. Принцип роботи даного підходу зображено на рисунку 2.3.

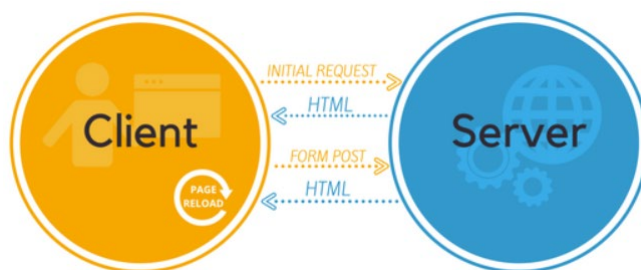


Рисунок 2.3 – Принцип роботи багатосторінкового додатку [8]

Переваги:

- МРА архітектура дозволяє легко оптимізувати web-додаток для пошукових систем. Є можливість додати мета-теги для кожної сторінки окремо;
- як правило, для розробки багатосторінкових додатків потрібен менший стек технологій, таким чином їх вартість виходить дешевшою.

Недоліки:

- розробка як десктопної, так і мобільної версії займає значно більше часу;
- тісний зв'язок між бекендом і фронтендом ускладнює процес додавання нових функціональних можливостей [9].

Підвівши підсумки, можна зробити висновок, що кожна архітектура має свої переваги для проектів певного типу. Так, варіант з використанням МРА добре підійде для тих випадків, коли потрібно відобразити велику кількість контенту, наприклад, при створенні інтернет-магазину, бізнес сайту, каталогу. Але, так як передбачається, що розроблювана система буде мати інтерфейс з досить високим ступенем інтерактивності, застосування підходу, при якому сторінка

перезавантажується після кожної дії користувача, не є доцільним. Виходячи з цих міркувань, для реалізації системи скористаємося концепцією односторінкового додатку.

2.2 Огляд фреймворків для розробки системи

Розробка веб-додатку з нуля вимагає багато роботи. При цьому в більшості випадків доводиться витратити час на відтворення функцій, що вже були виконані тисячі разів. Фреймворки допомагають обійти цю проблему, надаючи свого роду каркас майбутньої програми та набір додаткових інструментів, покликаних прискорити роботу зі створення продукту.

Фреймворк – це програмне забезпечення, яке спрощує створення і підтримку технічно складних або навантажених проектів. Як правило, вони містять тільки базові програмні модулі, а всі специфічні для проекту компоненти реалізуються розробником на їх основі. Тим самим досягається не тільки висока швидкість розробки, а й значна продуктивність і надійність рішень [10].

Загальні переваги від використання фреймворків:

- у методологіях фреймворків зазвичай закладено кращі практики програмної інженерії, тож просто дотримуючись цих правил можна уникнути багатьох проблем і помилок в проектуванні.

- екосистеми веб-фреймворків мають велику кількість готових рішень для нестандартних завдань. Також є можливість розширення функціоналу шляхом підключення сторонніх бібліотек або окремих класів;

- побудовані на базі фреймворків додатки мають уніфіковану структуру. Відповідно їх значно простіше супроводжувати і модернізувати, так як стандартизована організація компонентів зрозуміла всім розробникам на цій платформі.

– рішення на фреймворках, як правило, працюють значно швидше, витримують більші навантаження, а також мають вищий рівень безпеки у порівнянні з системами, написаними без їхнього використання [11].

Виходячи з того, що концепція SPA передбачає чітке розділення відповідальності між зовнішнім представленням (front-end) і внутрішньою реалізацією (back-end) варто окремо потурбуватися про вибір фреймворка як для серверної, так і для клієнтської частин. При цьому серверний фреймворк дозволить полегшити вирішення таких завдань, як взаємодія з базою даних, авторизація користувачів і забезпечення захисту від веб-загроз. Фреймворк клієнтської сторони, у свою чергу, допоможе в побудові багатofункціонального інтерфейсу користувача.

2.2.1 Вибір фреймворку для розробки серверної частини

Проведемо аналіз найпопулярніших серверних фреймворків:

Django – веб-фреймворк на мові Python. Його робота ґрунтується на принципах "DRY" (don't repeat yourself) – механізмі повторного використання, що дозволяє мінімізувати дублювання коду в ході розробки. Важливою архітектурною відмінністю цього фреймворка є спосіб організації обробників URL, які у випадку Django конфігуруються явно (за допомогою регулярних виразів), а не автоматично задаються із структури контролерів.

Особливості:

- ORM, API доступу до БД з підтримкою транзакцій;
- диспетчер URL на основі регулярних виразів;
- механізм інтернаціоналізації, що забезпечує можливість роботи додатка на декількох мовах;
- спеціалізована архітектура дозволяє встановлювати допоміжні модулі у будь-які Django-проекти;

- вбудований механізм авторизації та можливість підключення зовнішніх модулів аутентифікації: LDAP, OpenID;

- система фільтрів (middleware) для побудови додаткових обробників запитів, як наприклад включені в дистрибутив фільтри для кешування, нормалізації URL та підтримки анонімних сесій;

- бібліотека для роботи з формами (успадкування, побудова форм по існуючій моделі БД) [12];

Laravel – один з найпопулярніших PHP-фреймворків з відкритим вихідним кодом. Дозволяє значно спростити розробку, завдяки вбудованим засобам вирішення загальних завдань, серед яких аутентифікація, маршрутизація, сесії і кешування. Цей фреймворк є досить гнучким – можна створити власну структуру, яка відповідає вимогам конкретного проекту.

Особливості:

- пакети – дозволяють створювати і підключати модулі у форматі Composer. Багато додаткових можливостей вже доступні у вигляді таких модулів.

- Eloquent ORM – представляє таблиці бази даних у вигляді класів для полегшення доступу і маніпулювання даними.

- зворотня маршрутизація – пов'язує між собою згенеровані додатком посилання і маршрути. При створенні посилань за допомогою іменованих маршрутів Laravel автоматично генерує кінцеві URL.

- REST- контролери – додатковий шар для розділення логіки обробки GET і POST запитів HTTP.

- автозавантаження класів – механізм автоматичного завантаження класів PHP без необхідності підключати файли їх визначень через include.

- міграції – система управління версіями для баз даних. Дозволяє зв'язати зміни в коді додатка зі змінами, що треба внести в структуру БД. Це спрощує процес розгортання і оновлення додатка.

- модульне тестування – дозволяє перевірити окремі модулі вихідного коду програми на предмет регресій (помилки внаслідок оновлення коду або виправлення

інших помилок) [13].

Ruby on Rails – багаторівневий фреймворк для побудови веб-додатків, що використовують реляційні і NoSQL бази даних (наприклад, MySQL, MariaDB, PostgreSQL, MongoDB). Фреймворк написаний на мові програмування Ruby. Одним з ключових принципів розробки на Ruby on Rails є "convention over configuration" – при початковій ініціалізації фреймворк використовує угоди по конфігурації, типові для більшості проектів. Це значно спрощує процес створення додатків, оскільки явна специфікація конфігурації потрібна тільки в нестандартних випадках [14].

Особливості:

- бібліотека доступу до БД – реалізує шаблон проектування Active Record, який спрощує роботу з базами даних. Він дозволяє представити таблиці у вигляді класів, а рядки – об'єктів. За рахунок цього є можливість використовувати об'єктно-орієнтований підхід доступу до даних.

- бібліотеки для загальних завдань – фреймворк містить безліч допоміжних бібліотек, які створюють функціонал для роботи з файлами, відправки електронних листів, валідації форм, підтримки сесій, роботи із зображеннями тощо.

- конфігурація URL – Ruby on Rails має інструменти для гнучкого налаштування URL-адрес.

Встановимо ряд критеріїв для визначення оптимального серверного фреймворка:

- документація – наявність зрозумілої документації для швидкого старту;
- продуктивність – швидкість обробки вхідних запитів;
- безпека – ступінь захисту від веб-загроз;
- поріг освоєння – кількість часу, необхідного для освоєння фреймворка людиною з мінімальним досвідом у веб-розробці;
- гнучкість і масштабування – можливість впровадження допоміжних бібліотек для розширення базових можливостей.

У таблиці 2.1 зазначено відповідність розглянутих фреймворків встановленим критеріям.

Таблиця 2.1 – Аналіз серверних фреймворків

	Django	Laravel	Ruby on Rails
Документація	+	+	–
Продуктивність	+	+	+
Безпека	–	+	–
Поріг освоєння	±	+	–
Гнучкість і масштабування	+	+	+

За результатами проведеного аналізу можна зробити висновок, що всі фреймворки мають гарні показники відносно таких важливих характеристик, як продуктивність і гнучкість. Однак в плані безпеки лідирує Laravel, що пов'язано з наступними можливостями фреймворка:

- захист від SQL-ін'єкцій – вбудована ORM виключає можливість використання “сирих” SQL-запитів, а всі параметри при їх побудові нормалізуються.

- захист від XSS (кроссайтового скриптингу) – фреймворк забезпечує механізм екранування заборонених тегів (зокрема script, php та інших), при цьому код буде представлений у вигляді звичайного тексту без можливості його виконання на сервері.

До його переваг також можна віднести використання PHP – найбільш популярної мови у сфері веб-розробки, що забезпечує її підтримку будь-якими хостинг-провайдерами. Виходячи з цього для написання серверної частини CRM-системи було обрано Laravel.

2.2.2 Вибір фреймворку для розробки клієнтської частини

Проведемо аналіз найпопулярніших клієнтських фреймворків:

Vue.js – бібліотека для створення інтерактивних веб-інтерфейсів. Метою Vue.js є надання переваг реактивного зв'язування компонентів представлення з API.

Vue.js не являється повноцінним фреймворком, оскільки відповідає тільки за візуалізацію, тобто працює на рівні представлення. Проте у поєднанні з належними інструментами та допоміжними бібліотеками, Vue.js чудово підходить для створення веб-додатків будь-якого рівня складності [15].

Особливості:

- прив'язка даних – дозволяє встановити зв'язок між елементами веб-сторінки і даними об'єкту Vue.js. Таким чином можна керувати значеннями HTML-атрибутів, змінювати стилі, привласнювати CSS-класи.

- реактивні інтерфейси – при зміні даних автоматично оновлюється їх відображення на сторінці.

- компактність – ядро Vue.js містить лише функціонал для роботи з інтерфейсом. Тому воно компактне і легко інтегрується з іншими технологіями.

React – бібліотека для створення компонентно-орієнтованих інтерфейсів за допомогою JavaScript і (необов'язково) XML. Вона використовує концепцію розділення інтерфейсу користувача на компоненти, самодостатні частини, які просто розширювати і застосовувати повторно. Компоненти можна описати використовуючи JavaScript або XML з HTML-подібним синтаксисом (JSX).

Особливості:

- одностороння передача даних – компонент не може напряму змінювати властивості, що йому передані, а тільки за допомогою callback функції. Такий механізм називають «властивості донизу, події нагору». Він мінімізує ризик виникнення помилок та дозволяє легше відстежувати зміни.

- віртуальний DOM – React створює кеш структуру в пам'яті, що дозволяє визначити різницю між попереднім і поточним станами інтерфейсу для оптимального оновлення елементів сторінки.

- JavaScript XML (JSX) – це розширення синтаксису JavaScript, що дозволяє використовувати схожий на HTML синтаксис для опису структури інтерфейсу [16].

Angular – платформа побудована на TypeScript, що надає розробникам надійні інструменти для створення клієнтської частини веб-додатків. Архітектура фреймворка ґрунтується на трьох складових: модуль (надає контекст компіляції для

компонентів), компонент (визначає набір елементів інтерфейсу користувача, які Angular може змінювати відповідно до логіки програми і даних), маршрутизація (дозволяє зв'язати декілька компонентів один з одним) [17].

Особливості:

- модульність – розділення коду на окремі модулі, що полегшує процес тестування і дозволяє повторно використовувати різні частини програми.

- директиви – дозволяють розширювати базовий функціонал HTML і створювати нові конструкції.

- TypeScript – має ряд переваг у порівнянні зі звичайним JavaScript, серед яких: суворі типізація (дозволяє зробити поведінку коду більш передбачуваною і спростити налагодження програми), механізми об'єктно-орієнтованого програмування тощо.

Вибір фреймворка для клієнтської частини буде здійснюватись за наступними критеріями:

- простота – ступінь складності освоєння;
- документація – наявність детально описаного керівництва по використанню;
- швидкість рендерінгу – кількість часу, необхідного фреймворку для відображення на сторінці оновлених даних.

У таблиці 2.2 зазначено відповідність розглянутих фреймворків встановленим критеріям.

Таблиця 2.1 – Аналіз клієнтських фреймворків

	Vue.js	React	Angular
Простота	+	+	–
Документація	+	+	+
Швидкість рендерінгу	+	+	–

Виходячи з результатів аналізу React і Vue.js випереджають Angular за швидкістю роботи завдяки впровадженню концепції VirtualDOM. Цей підхід дозволяє оптимізувати використання ресурсів, що позначається на швидкості відображення змін. Ще одним недоліком Angular є складний синтаксис. Також для роботи з цим фреймворком треба додатково вчити TypeScript. Інші фреймворки чудово підходять для реалізації поставленої задачі. Але виходячи з особистого досвіду роботи, а також враховуючи той факт, що обраний раніше серверний фреймворк Laravel забезпечує офіційну підтримку Vue.js, при розробці клієнтської частини будемо використовувати саме його.

3 ПРОЕКТУВАННЯ АРХІТЕКТУРИ СИСТЕМИ

3.1 Проектування логічної та фізичної структури сайту

Для комфортного використання можливостей системи та полегшення пошуку потрібної інформації, вона повинна мати чітку і продуману структуру. З позиції розробника структуру умовно можна поділити на два рівні: логічний і фізичний. На логічному рівні вона представляє собою сукупність сторінок, які об'єднані між собою єдиним дизайном, стилем і посиланнями. На рисунку 3.6 наведено логічну структуру системи управління взаємодією з клієнтами стоматології.

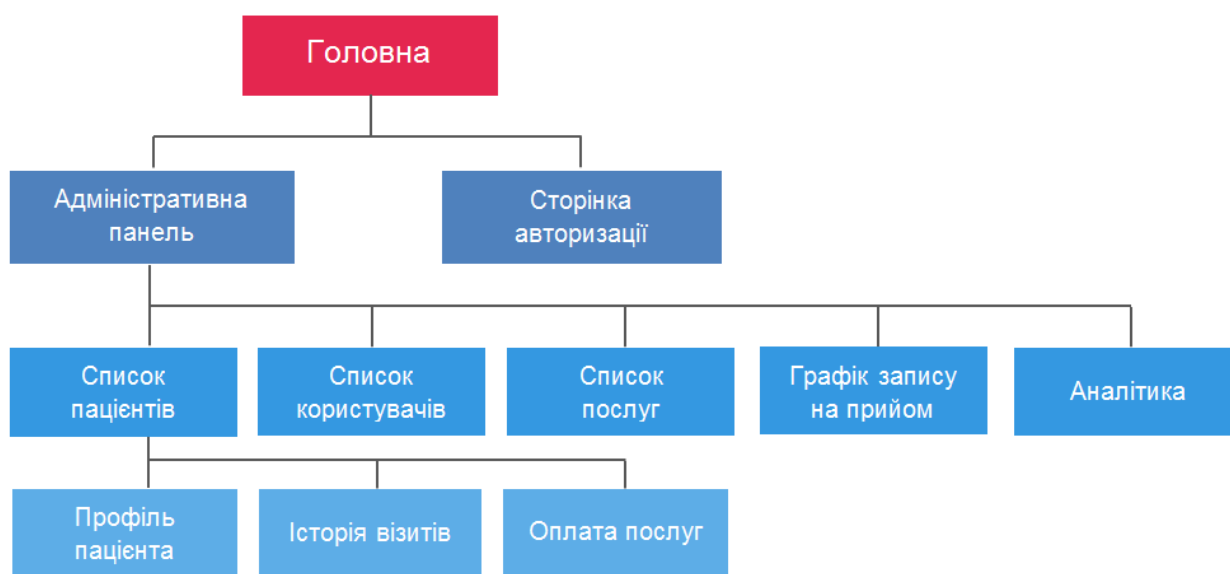


Рисунок 3.1 – Логічна структура

Як можна побачити із наведеної вище структури для реалізації функціоналу системи будуть розроблені такі модулі: модуль пацієнтів, модуль користувачів, модуль послуг, модуль реєстратури, модуль аналітики.

Модуль пацієнтів призначений для роботи з пацієнтами стоматологічної клініки. Включає в себе підмодулі для перегляду профілю пацієнта, його історії візитів та оплат. Забезпечує виконання наступних операцій:

- відображення списку пацієнтів;
- пошук пацієнта за ініціалами або номером телефону;

- внесення до бази інформації про нового пацієнта;
- редагування профілю пацієнта;
- видалення профілю пацієнта;
- відображення історії візитів пацієнта;
- внесення до бази інформації щодо проведеного прийому пацієнта;
- відображення заборгованості та історії оплат пацієнта;
- внесення до бази інформації щодо внесеної оплати.

Модуль користувачів призначений для роботи з користувачами системи. Доступ до нього є тільки у адміністратора. Забезпечує виконання наступних операцій:

- відображення списку користувачів;
- пошук користувача за ініціалами;
- реєстрація нового користувача;
- редагування профілю користувача;
- видалення профілю користувача.

Модуль послуг призначений для роботи з переліком послуг, що надаються стоматологічною клінікою. Забезпечує виконання наступних операцій:

- відображення списку послуг;
- пошук послуги за назвою;
- внесення до бази інформації про нову послугу;
- редагування даних окремої послуги;
- видалення інформації про послугу.

Модуль реєстратури призначений для перегляду для інформації стосовно запису пацієнтів. Забезпечує виконання наступних операцій:

- відображення графіку записів на прийом;
- відображення часу, на який був назначений прийом та даних пацієнта по кожному запису;
- внесення до бази інформації про запланований прийом;
- редагування інформації стосовно запису на прийом.

Модуль аналітики призначений для оцінки динаміки внутрішніх процесів клініки. Забезпечує виконання наступних операцій:

- відображення графічних звітів відносно кількості візитів, загальної вартості наданих послуг, суми платежів тощо;
- фільтрація інформації за періодом часу, лікарем або пацієнтом.

Фізичний рівень, у свою чергу, відповідає за файлову складову, а саме правильне розміщення елементів різного типу (HTML-сторінки, скрипти, зображення) у відповідних папках та файлах. В ході розробки системи була використана стандартна організація файлів фреймворку Laravel (рис. 3.7). В ній можна виділити наступні ключові складові:

- каталог `css` – містить файли, що підключаються до кореневої сторінки та будуть задавати стилі для всього додатку;
- каталог `js` – містить файли із описаними сценаріями javascript для різних інструментів та сценарії для підтримання динаміки сайту;
- каталог `storage` (`patient-avatars`, `user-avatars`) – містить файли, що завантажують користувачі системи (наприклад зображення профілю);
- каталог `images` – містить необхідні зображення для коректного відображення сторінок додатку.

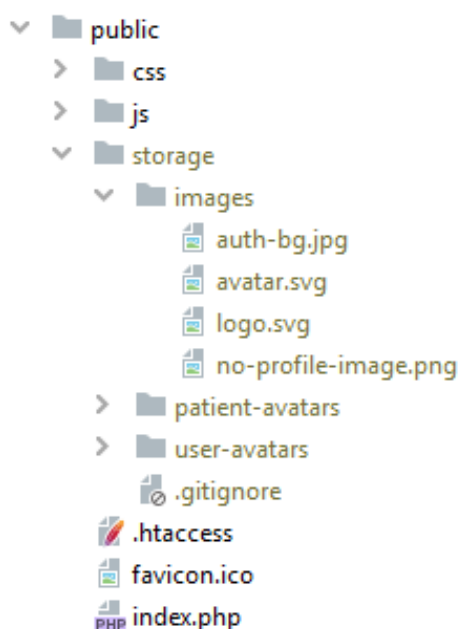


Рисунок 3.2 – Фізична структура

3.2 Проектування діаграм на мові uml

UML – об'єктно-орієнтована графічна мова для візуалізації та документування систем, де велика роль відводиться опису бізнес-процесів в інформаційних системах. UML є мовою широкого профілю, це відкритий стандарт, що використовує графічні позначення для створення абстрактної моделі системи, яка називається UML моделлю.

Переваги:

- об'єктно-орієнтований, внаслідок чого методи опису результатів аналізу і проектування семантично близькі до методів програмування на сучасних об'єктно-орієнтованих мовах;
- дозволяє описати систему практично з усіх можливих точок зору і різні аспекти поведінки системи;
- діаграми UML порівняно прості для читання після швидкого ознайомлення з синтаксисом [18];

3.2.1 Діаграма варіантів використання

Для детального опису функціональних вимог до програмного забезпечення скористаємося діаграмою варіантів використання (use cases). Вона складається з дійових осіб і прецедентів, а також специфікацій, що є текстовим описом конкретних послідовностей дій (потоків подій), які виконує користувач при роботі з системою.

В таблиці 3.1 наведено перелік дійових осіб та варіантів використання системи.

Таблиця 3.1 - Документування об'єктів діаграми варіантів використання.

Об'єкт	Документація
<i>Дійові особи</i>	
Адміністратор	Відповідає за додавання нових користувачів в систему. Має розширені права користувача. Ключовими особливостями є можливість редагування вартості послуг, перегляд інформації про усіх пацієнтів і зібраної статистики відносно роботи клініки.
Користувач	Відповідає за роботу з пацієнтами (додавання/видалення, редагування профілю тощо).
<i>Варіанти використання</i>	
Авторизація	Вхід в систему після успішної перевірки облікового запису (логіна і пароля). На цьому етапі визначається роль користувача.
Перегляд списку користувачів	Перегляд інформації про користувачів системи (ПІБ, контактні дані).
Додавання користувача	Реєстрація нового користувача і внесення інформації у БД.
Видалення користувача	Відправка запиту на видалення даних про користувача з БД.
Перегляд списку послуг	Перегляд інформації про послуги, що надає стоматологічна клініка.

Продовження таблиці 3.1

Об'єкт	Документація
Додавання послуги	Додавання нової послуги і внесення інформації у БД.
Редагування послуги	Редагування інформації про послугу і збереження змін у БД.
Видалення послуги	Відправка запиту на видалення даних про послугу з БД.
Перегляд списку пацієнтів	Перегляд основної інформації про пацієнтів (ПІБ, контактні дані, дата народження).
Додавання пацієнта	Реєстрація нового пацієнта і внесення інформації у БД.
Перегляд профілю пацієнта	Перегляд повної інформації про пацієнта (особисті дані і медична інформація).
Редагування профілю пацієнта	Редагування інформації про пацієнта і збереження змін у БД.
Перегляд історії візитів	Перегляд інформації про візити пацієнта (дата, список зроблених послуг, нарахована сума).
Додавання інформації щодо проведеного прийому	Додавання у БД інформації щодо проведеного прийому пацієнта.
Перегляд історії оплат	Перегляд інформації відносно оплати рахунків (дата, внесена сума).
Додавання інформації про внесену оплату	Додавання у БД інформації про внесену оплату.

Продовження таблиці 3.1

Об'єкт	Документація
Видалення пацієнта	Відправка запиту на видалення даних про пацієнта з БД.
Перегляд графіку запису пацієнтів на прийом	Перегляд інформації відносно запису пацієнтів (дата і час прийому, відомості про пацієнта).
Запис пацієнта на прийом	Додавання у БД інформації відносно запису пацієнта на прийом.
Перегляд статистики	Перегляд графічних звітів по динаміці наступних показників: кількість візитів, загальна вартість наданих послуг, сума платежів тощо.

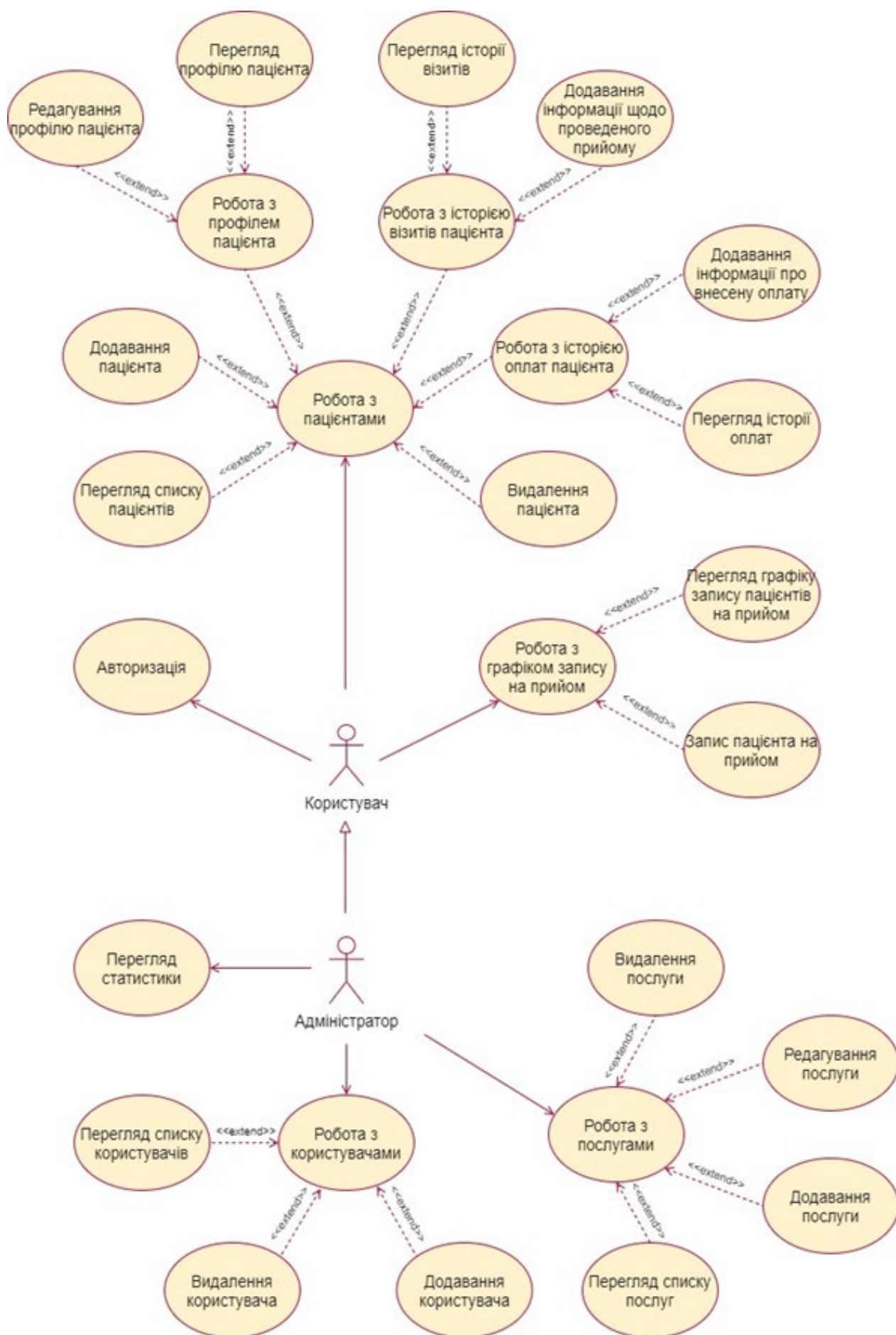


Рисунок 3.3 – Діаграма варіантів використання

3.2.2 Діаграми послідовності

Діаграма послідовності призначена для моделювання взаємодії об'єктів системи в часі, а також обміну повідомленнями між ними. Вона складається з таких елементів:

- позначення об'єктів (прямокутники з назвами об'єктів);
- вертикальні "лінії життя", що ілюструють плин часу;
- прямокутники, що відображають діяльність об'єкту або виконання їм певної функції (прямокутники на пунктирній "лінії життя");
- стрілки, що показують обмін сигналами або повідомленнями між об'єктами.

Нижче наведено ряд діаграм, що більш детально описують логіку основних сценаріїв використання.

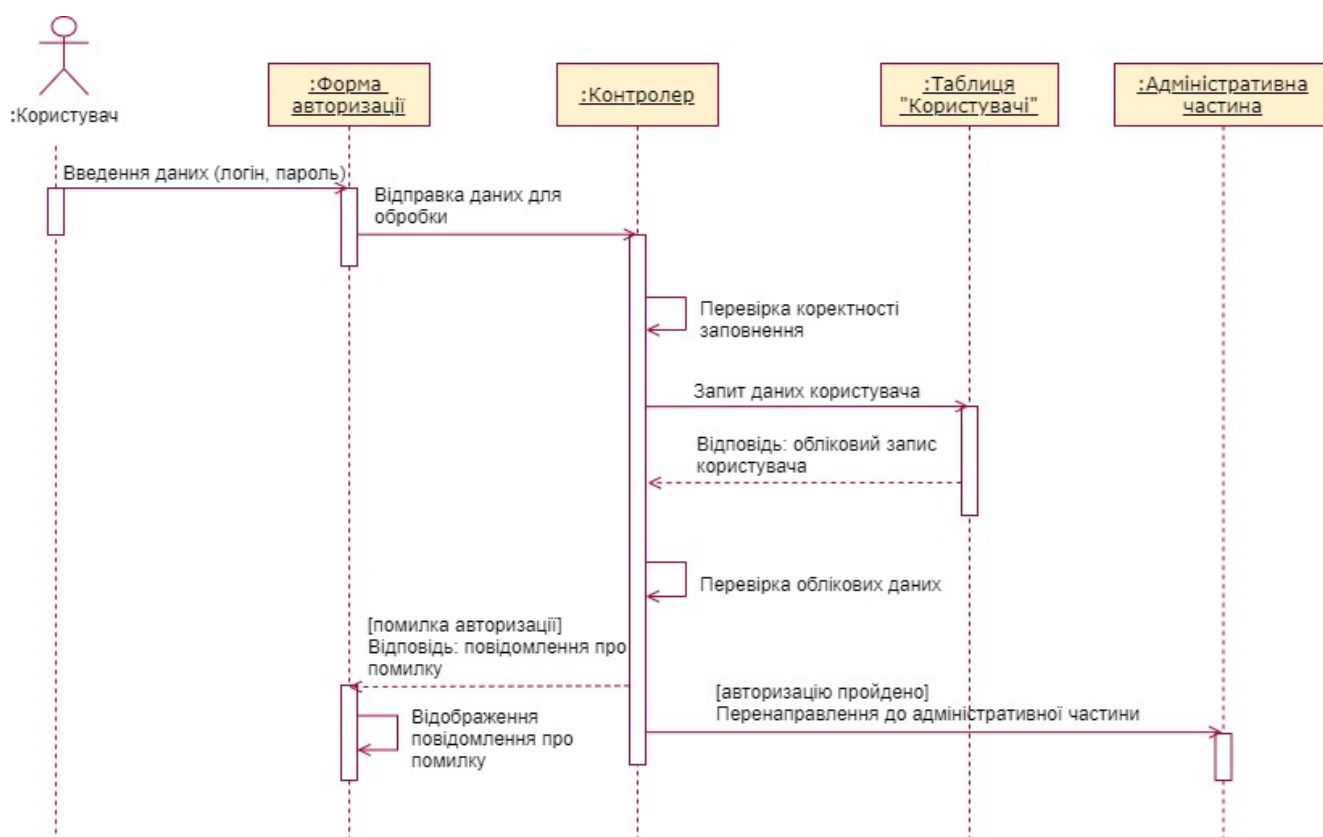


Рисунок 3.4 – Діаграма послідовності «Авторизація»

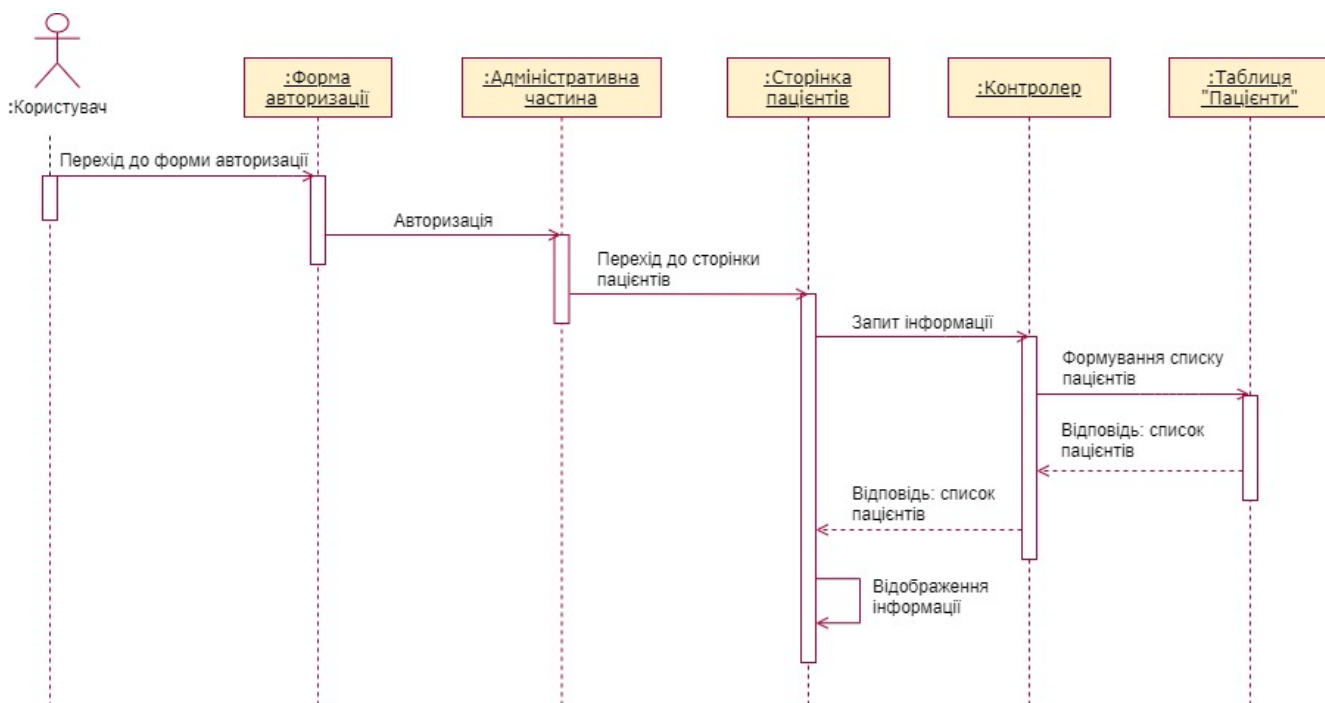


Рисунок 3.5 – Діаграма послідовності «Відображення списку пацієнтів»

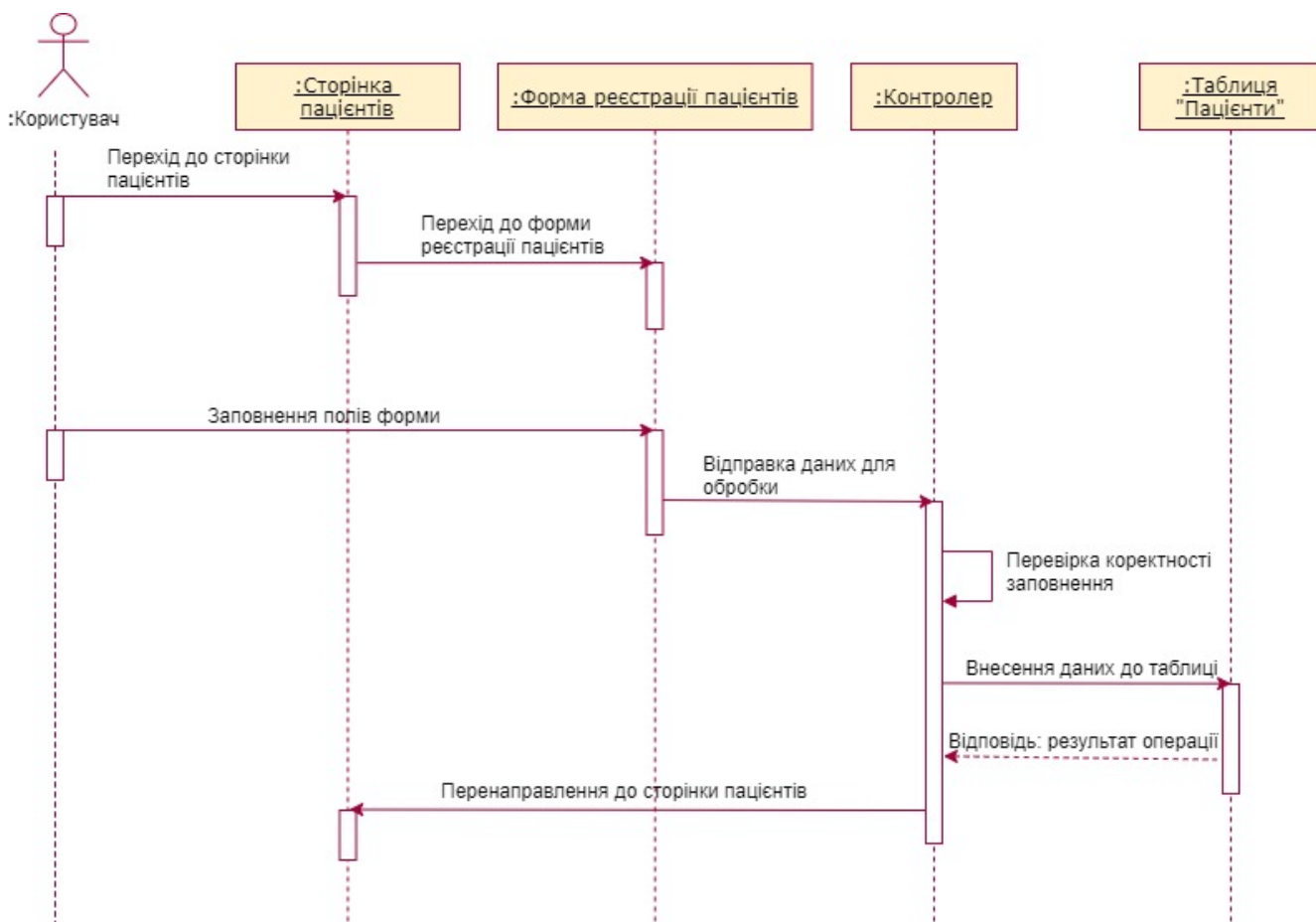


Рисунок 3.6 – Діаграма послідовності «Реєстрація нового пацієнта»

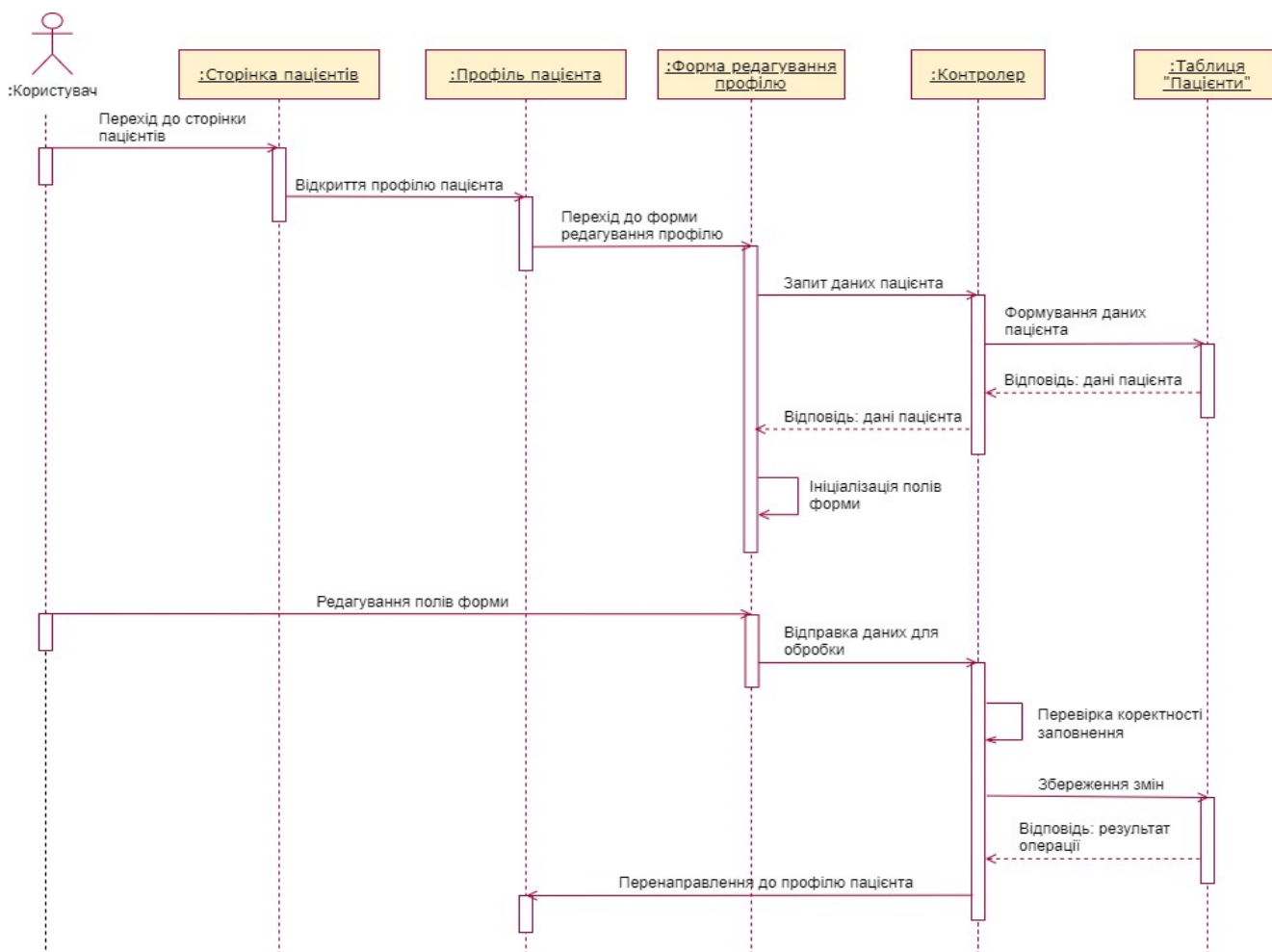


Рисунок 3.7 – Діаграма послідовності «Редагування профілю пацієнта»

3.3 Проектування API

У веб-додатку, побудованому за архітектурою SPA серверна сторона обмежена бекендом через деякі кінцеві точки входу API (Application Programming Interface). Клієнт відправляє запити на ці кінцеві точки, а сервер повертає відповідь. Проте сервер не піклується про те, як клієнт буде відображати отриману інформацію, що ідеально відповідає принципу розділення відповідальності. Ця архітектура дозволяє розробникам створювати надійні веб-додатки, а також додатки для різних пристроїв [19].

Нижче наведено декілька IDEF0 діаграм, що відображають принцип обробки вхідних запитів.



Рисунок 3.8 – Загальна IDEF0 діаграма

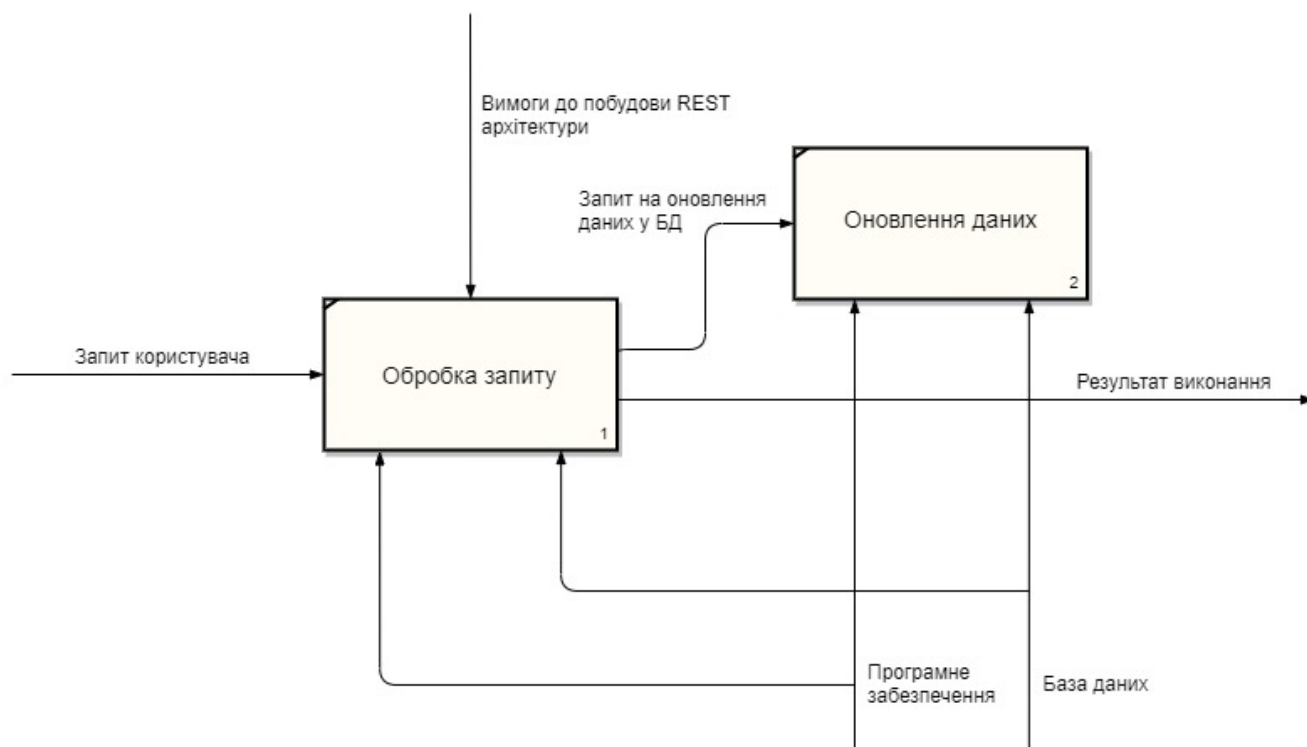


Рисунок 3.9 – Декомпозиція блоку "Сервер"

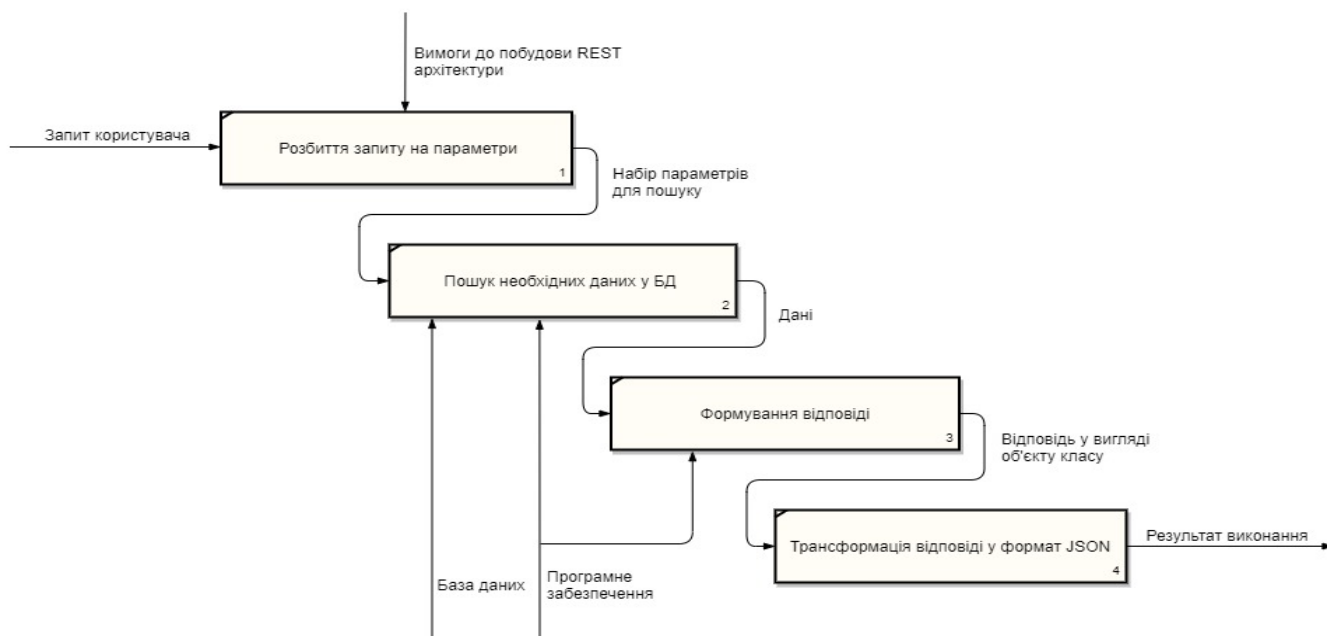


Рисунок 3.10 – Декомпозиція блоку "Обробка запиту"

У цій роботі для створення RESTful API буде використовуватися Laravel версії 6.2. Термін "RESTful" вказує на те, що цей інтерфейс буде працювати за принципами REST. У загальному розумінні REST - це архітектурний стиль для проектування розподілених систем. Він не є стандартом, але визначає обмеження, такі як відсутність станів, клієнт-серверний взаємозв'язок і уніфікований інтерфейс. При такому підході тіло запиту і відповіді від сервера представляє собою JSON об'єкт.

JSON (JavaScript Object Notation) – стандартний текстовий формат для представлення структурованих даних на основі синтаксису об'єкту JavaScript. В якості прикладу розглянемо як виглядають дані пацієнта у форматі JSON:

```

"patient": {
  "id": 1,
  "full_name": "Rebeka Zemlak",
  "phone": "+38 (074) 689-84-56",
  "email": "balistreri.kayden@example.com",
  "gender": 1,
  "birth_date": "1992-07-13",

```

```

"medical_info":" Lorem ipsum dolor sit amet, consectetur adipiscing elit.",
"total_accrued":"235.00",
"total_paid":"0.00",
"anamnesis":{
    "id":5,
    "name":"Lorem ipsum",
    "created_at":"2020-03-16 16:47:29",
    "updated_at":"2020-03-16 16:47:29"
},
"image_path":"http://127.0.0.1:8000/storage/patient-
avatars/zuOm2ZFZz5qR96HiPznhB4d2I3yw2AXHg82c8ltS.png"
"created_at":"2020-03-16 16:47:29",
"updated_at":"2020-04-08 20:44:59",
}

```

Відповідно до архітектурних принципів REST, існує щонайменше 4 HTTP-методи, що відповідають основним операціям над сутностями: завантаження даних (GET), збереження (POST), редагування (PUT/PATCH) і видалення (DELETE). Цей перелік також супроводжується такими операціями, як обробка помилок в запиті, розмежування доступу і валідація вхідних даних. Структура HTTP-методів, що були використані в ході розробки представлена в таблиці 3.2.

Таблиця 3.2 – HTTP-методи, що використовувалися при розробці

Метод	Операція	Тіло запиту	Тіло відповіді
GET	Отримати інформацію	NA	JSON string
POST	Створити новий запис	JSON string	JSON string
PUT/PATCH	Оновити запис	JSON string	JSON string
DELETE	Видалити запис	NA	JSON string

Універсальні відповіді від сервера представлені в таблиці 3.3.

Таблиця 3.3 – Універсальні відповіді від сервера

Помилка	Код відповіді	Опис
Bad request	400	Некоректний запит (універсальна помилка)
Not found	404	Ресурс не знайдено
Unprocessable Entity	422	Неможливо виконати операцію над ресурсом
Not authorized	403	Необхідна авторизація
Server error	500	Помилка сервера

У разі помилки API повертає код статусу HTTP і відомості про її джерело. Крім того, у відповіді міститься інформація відносно того, що призвело до цієї помилки. Об'єкт помилки представлено в таблиці 3.4.

Таблиця 3.4 – JSON-об'єкт помилки

Атрибут	Тип	Призначення
http_code	number	Ідентифікатор помилки
message	string	Повідомлення про помилку
data	object	Інформація, що асоціюється з помилкою

Всі інші JSON-об'єкти, якими обмінюються між собою front-end і back-end, відображають свої сутності. Для прикладу розглянемо об'єкт типу «Користувач» (табл. 3.5).

Таблиця 3.5 – JSON-об'єкт користувача

Атрибут	Тип	Призначення
id	number	Ідентифікатор
full_name	string	ПІБ

Продовження таблиці 3.5

Атрибут	Тип	Призначення
phone	string	Номер телефону
email	string	Адреса електронної пошти
image_path	string	Місце розміщення зображення профілю на сервері
created_at	string	Дата реєстрації аккаунта
updated_at	string	Дата оновлення аккаунта

Після проходження етапу проектування JSON-об'єктів, необхідно створити URL-адресу для кожної сутності, що дозволить унікально ідентифікувати REST запит. Зазвичай вона складається з пар виду "назва сутності/ідентифікатор", об'єднаних між собою URL роздільником "/". Ідентифікатор виступає в ролі первинного ключа для визначення об'єкту сутності та є обов'язковим параметром.

Представимо структуру кінцевих точок, що використовуються клієнтською стороною для взаємодії з сервером, у вигляді діаграм класів UML.

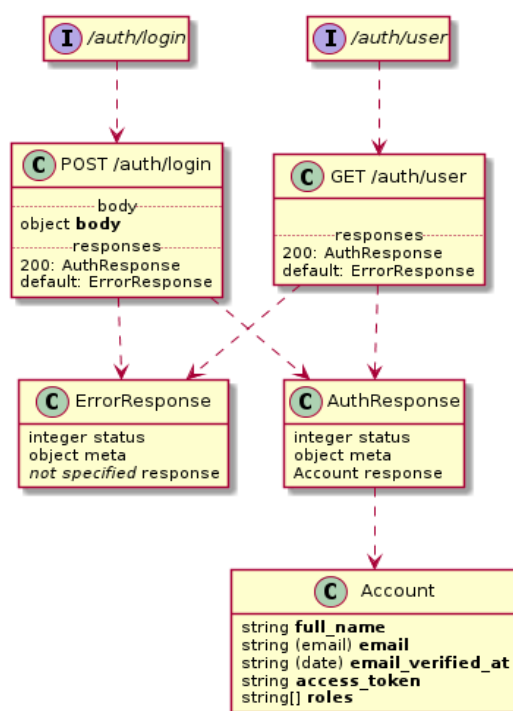


Рисунок 3.11 – Кінцеві точки для авторизації в системі

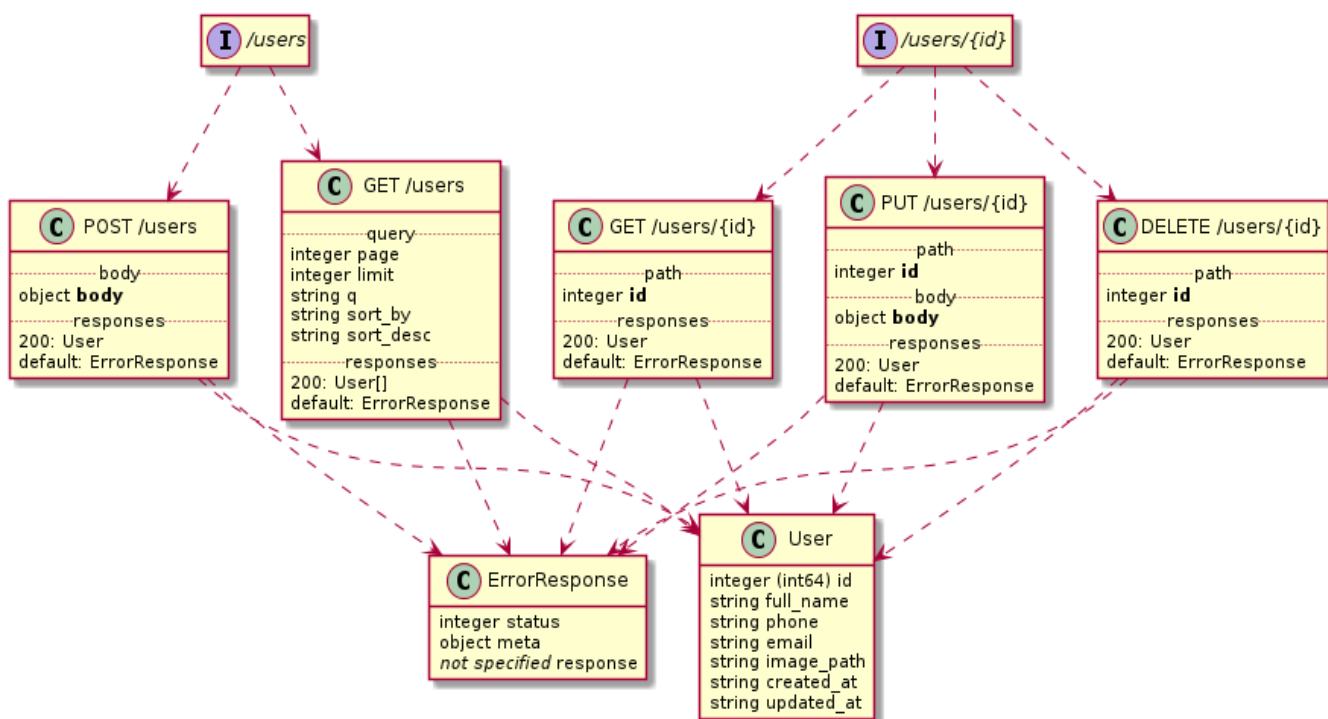


Рисунок 3.12 – Кінцеві точки для роботи з користувачами

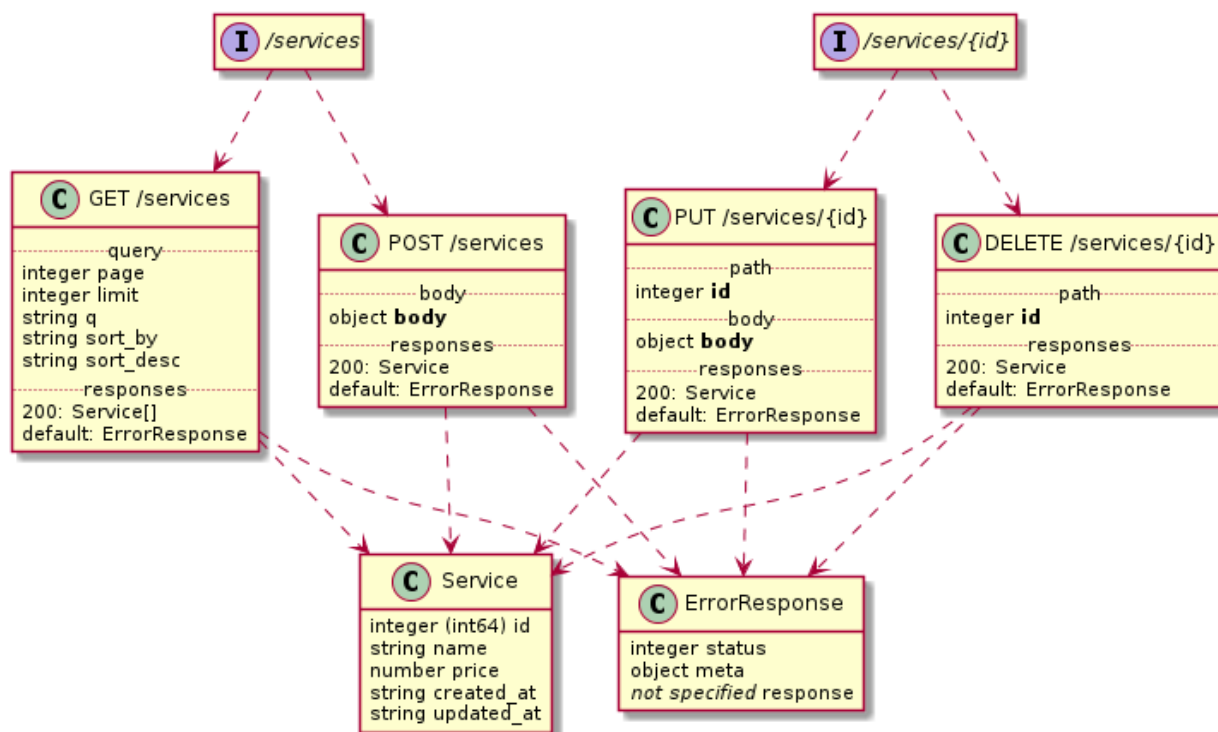


Рисунок 3.13 – Кінцеві точки для роботи з послугами

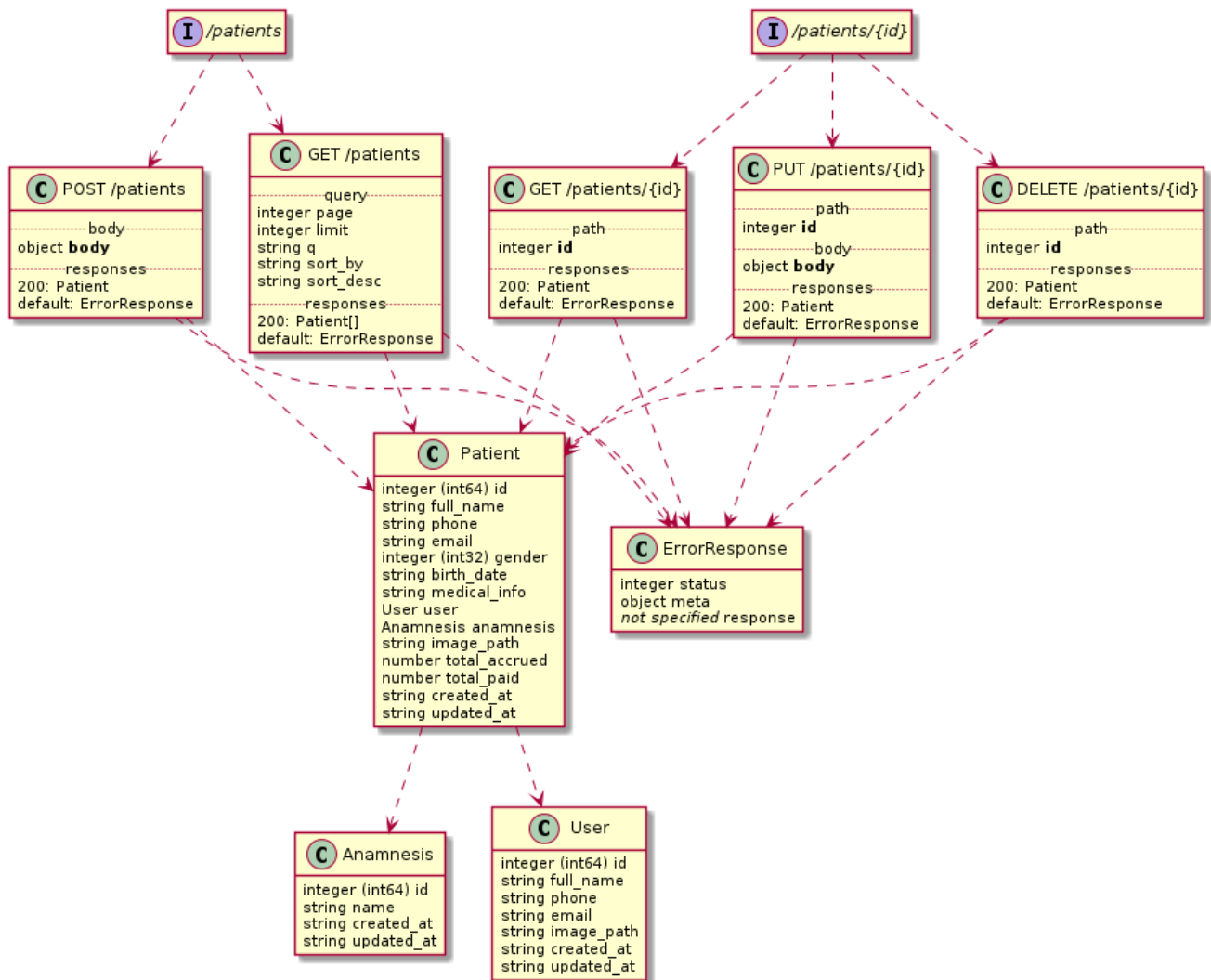


Рисунок 3.14 – Кінцеві точки для роботи з пацієнтами

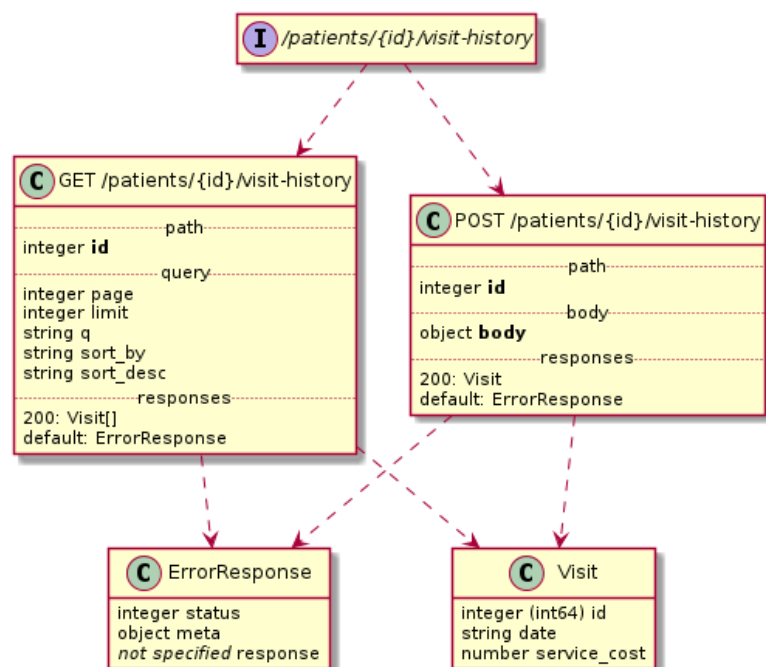


Рисунок 3.15 – Кінцеві точки для роботи з історією візитів

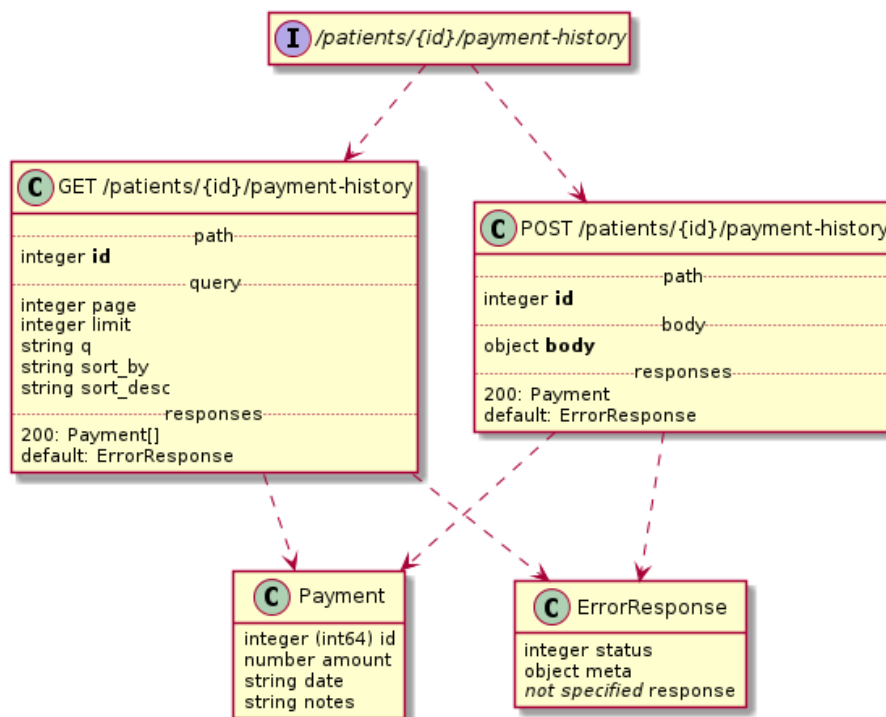


Рисунок 3.16 – Кінцеві точки для роботи з історією оплат

3.4 Проектування структури інтерфейсу

Структура інтерфейсу використовується при розробці дизайну веб-додатку та показує найбільш важливі елементи, їх положення та взаємозв'язок між сторінками. При розробці структури інтерфейсу зазвичай будують певний каркас, на якому схематично зображають найбільш важливі елементи, такі як верхній і нижній колонтитули (header та footer), панель навігації тощо.

Основні переваги розробки структури інтерфейсу перед створенням веб-додатку:

- раціональний розподіл часу і зосередження саме на тому, для чого призначена кожна сторінка;
- видалення зайвих елементів, що можуть виявитися непотрібними при роботі з системою;

- отримання чіткого уявлення про те, що потрібно реалізувати;
- зменшення обсягу роботи при розробці дизайну;

Нижче наведено структуру інтерфейсу основних сторінок системи.

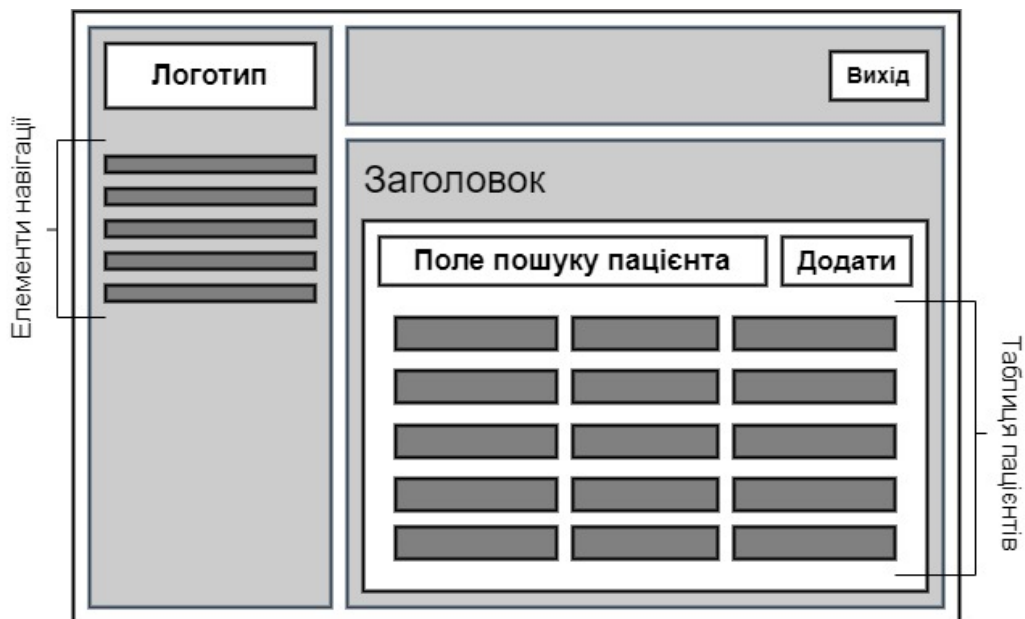


Рисунок 3.17 – Структура сторінки пацієнтів



Рисунок 3.18 – Структура сторінки з аналітикою

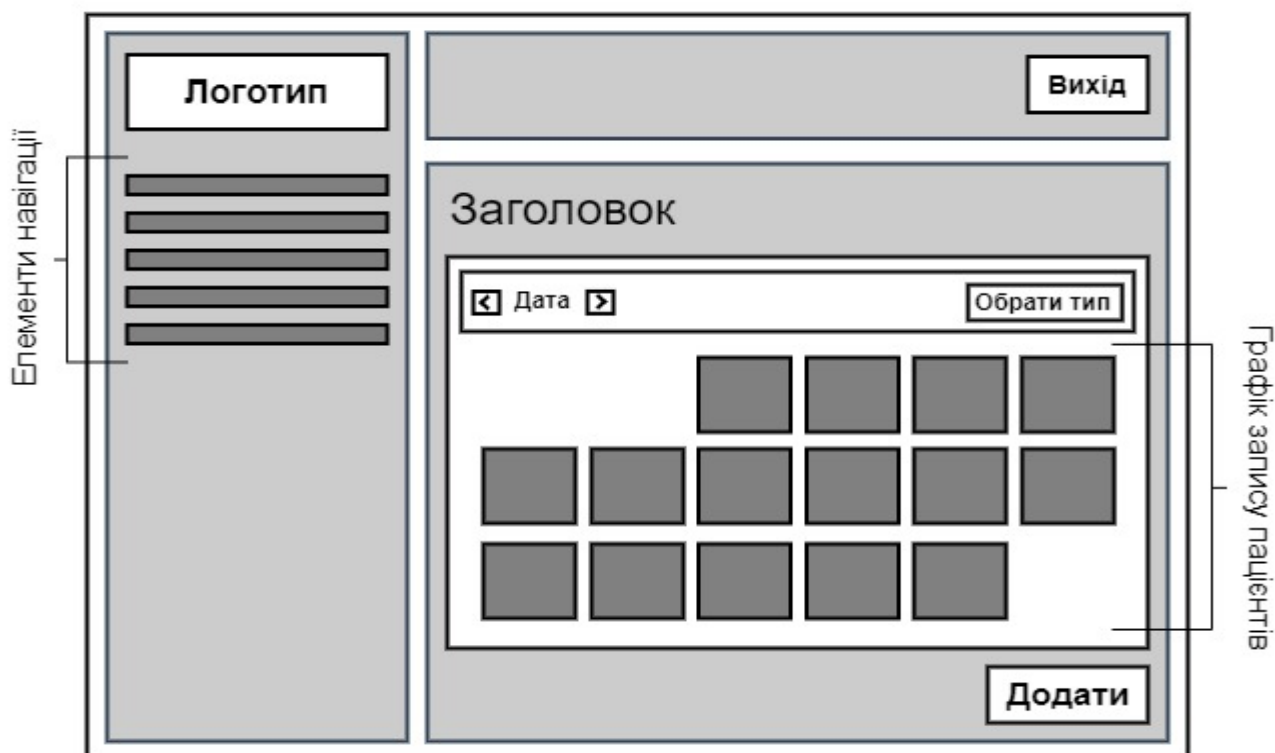


Рисунок 3.19 – Структура сторінки з графіком запису пацієнтів



Рисунок 3.20 – Структура сторінки з профілем пацієнта

При розробці інтерфейсу був зроблений акцент на зручність роботи. Таким чином, доступ до основних функцій користувач може отримати з головного меню на будь-якій сторінці системи.

3.5 Проектування бази даних

Ключове місце в системі управління взаємодією з клієнтами займає база даних, на проектування якої слід звернути особливу увагу, оскільки існує потреба в роботі з великими обсягами інформації. Виходячи з цього під час розробки структури БД були використані механізми нормалізації.

Нормалізація полягає в приведенні структури зберігання даних до нормальних форм. Нормальна форма – це ряд вимог, що пред'являється до структури таблиць для забезпечення цілісності бази даних і усунення надмірних функціональних залежностей. В цій системі база даних буде приведена до перших трьох нормальних форм:

- перша нормальна форма: кожна таблиця повинна мати первинний ключ; кожен атрибут має складатись тільки з елементарних (неподільних) значень і не містити групи, що повторюються (атомарність);
- друга нормальна форма: дані, що повторно з'являються в декількох рядках, виносяться в окремі таблиці;
- третя нормальна форма: дані не повинні зберігатися в таблиці, якщо їх можна отримати з не ключових полів [20].

Для формування бази даних були визначені наступні сутності: «Користувач», «Пацієнт», «Роль», «Повноваження», «Анамнез», «Послуга», «Візит», «Оплата», «Запис на прийом». Враховуючи всі особливості для кожної сутності була створена таблиця з відповідними атрибутами. Зв'язки між таблицями бази даних визначаються за допомогою ключів двох типів:

- первинний ключ – значення, що унікально ідентифікує кожний запис;
- зовнішній ключ – значення, що відповідає первинному ключу з іншої таблиці.

Розглянемо існуючі таблиці бази даних системи, а також їх призначення.

Users – зберігає інформацію про користувачів системи. Під користувачем мається на увазі співробітник стоматологічної клініки. Структуру полів наведено в таблиці 3.6.

Таблиця 3.6 – Структура полів таблиці Users

Назва	Тип	Інформаційний зміст
id	bigint	Унікальний ідентифікатор запису
full_name	varchar	Прізвище, ім'я, по-батькові користувача
phone	varchar	Номер телефону
email	varchar	Адреса електронної пошти (використовується в якості логіну при вході в систему)
email_verified_at	timestamp	Дата підтвердження електронної пошти
password	varchar	Пароль
hash_file_name	varchar	Назва файлу (зображення профілю)
created_at	timestamp	Дата реєстрації профілю
updated_at	timestamp	Дата оновлення профілю

Patients – зберігає інформацію про пацієнтів. Структуру полів наведено в таблиці 3.7.

Таблиця 3.7 – Структура полів таблиці Patients

Назва	Тип	Інформаційний зміст
id	bigint	Унікальний ідентифікатор запису
full_name	varchar	Прізвище, ім'я, по-батькові пацієнта
phone	varchar	Номер телефону
email	varchar	Адреса електронної пошти
gender	tinyint	Стать
birth_date	date	Дата народження

Продовження таблиці 3.7

Назва	Тип	Інформаційний зміст
medical_info	text	Додаткова медична інформація
user_id	bigint	Код користувача
anamnesis_id	bigint	Код анамнезу
total_accrued	decimal	Нарахована сума за весь час
total_paid	decimal	Сплачена сума за весь час
hash_file_name	varchar	Назва файлу (зображення профілю)
created_at	timestamp	Дата реєстрації профілю
updated_at	timestamp	Дата оновлення профілю

Roles – містить інформацію про ролі користувачів. Ці дані будуть використані для реалізації механізму розмежування прав доступу. Структуру полів наведено в таблиці 3.8.

Таблиця 3.8 – Структура полів таблиці Roles

Назва	Тип	Інформаційний зміст
id	bigint	Унікальний ідентифікатор запису
name	varchar	Назва ролі

Permissions – містить інформацію про повноваження. Повноваження визначають, чи є у ролі необхідні права на виконання певної операції. Структуру полів наведено в таблиці 3.9.

Таблиця 3.9 – Структура полів таблиці Permissions

Назва	Тип	Інформаційний зміст
id	bigint	Унікальний ідентифікатор запису
name	varchar	Назва операції

Permission_role – використовується для організації зв'язку "багато-до-багатьох" між таблицями ролей і повноважень. Структуру полів наведено в таблиці 3.10.

Таблиця 3.10 – Структура полів таблиці Permission_role

Назва	Тип	Інформаційний зміст
permission_id	bigint	Код повноваження
role_id	bigint	Код ролі

Role_user – використовується для організації зв'язку "багато-до-багатьох" між таблицями користувачів і ролей. Структуру полів наведено в таблиці 3.11.

Таблиця 3.11 – Структура полів таблиці Role_user

Назва	Тип	Інформаційний зміст
role_id	bigint	Код ролі
user_id	bigint	Код користувача

Anamnesis – зберігає список поширених захворювань, які можуть вплинути на перебіг лікування пацієнта. Структуру полів наведено в таблиці 3.12.

Таблиця 3.12 – Структура полів таблиці Anamnesis

Назва	Тип	Інформаційний зміст
id	bigint	Унікальний ідентифікатор запису
name	varchar	Назва захворювання

Services – зберігає перелік послуг, що надаються стоматологічною клінікою. Структуру полів наведено в таблиці 3.13.

Таблиця 3.13 – Структура полів таблиці Services

Назва	Тип	Інформаційний зміст
id	bigint	Унікальний ідентифікатор запису
name	varchar	Назва послуги
price	decimal	Вартість послуги

Payments – зберігає дані про оплату послуг пацієнтами. Структуру полів наведено в таблиці 3.14.

Таблиця 3.14 – Структура полів таблиці Payments

Назва	Тип	Інформаційний зміст
id	bigint	Унікальний ідентифікатор запису
patient_id	bigint	Код пацієнта
amount	decimal	Внесена сума
date	date	Дата оплати
notes	text	Примітка

Visits – зберігає інформацію про візити до лікарів. Структуру полів наведено в таблиці 3.15.

Таблиця 3.15 – Структура полів таблиці Visits

Назва	Тип	Інформаційний зміст
id	bigint	Унікальний ідентифікатор запису
patient_id	bigint	Код пацієнта
date	date	Дата візиту

Service_visit – використовується для організації зв'язку "багато-до-багатьох" між таблицями візитів і послуг. Структуру полів наведено в таблиці 3.16.

Таблиця 3.16 – Структура полів таблиці Service_visit

Назва	Тип	Інформаційний зміст
id	bigint	Унікальний ідентифікатор запису
visit_id	bigint	Код візиту
service_id	bigint	Код послуги
service_count	tinyint	Кількість послуг за певний візит
total_cost	decimal	Загальна вартість наданих послуг

Events – зберігає інформацію про записи пацієнтів на прийом. Структуру полів наведено в таблиці 3.17.

Таблиця 3.17 – Структура полів таблиці Events

Назва	Тип	Інформаційний зміст
id	bigint	Унікальний ідентифікатор запису
patient_id	bigint	Код пацієнта
user_id	bigint	Код користувача
start	datetime	Початок прийому
end	datetime	Кінець прийому
status	enum	Статус
details	text	Деталі

На рисунку 3.21 зображено модель бази даних системи.

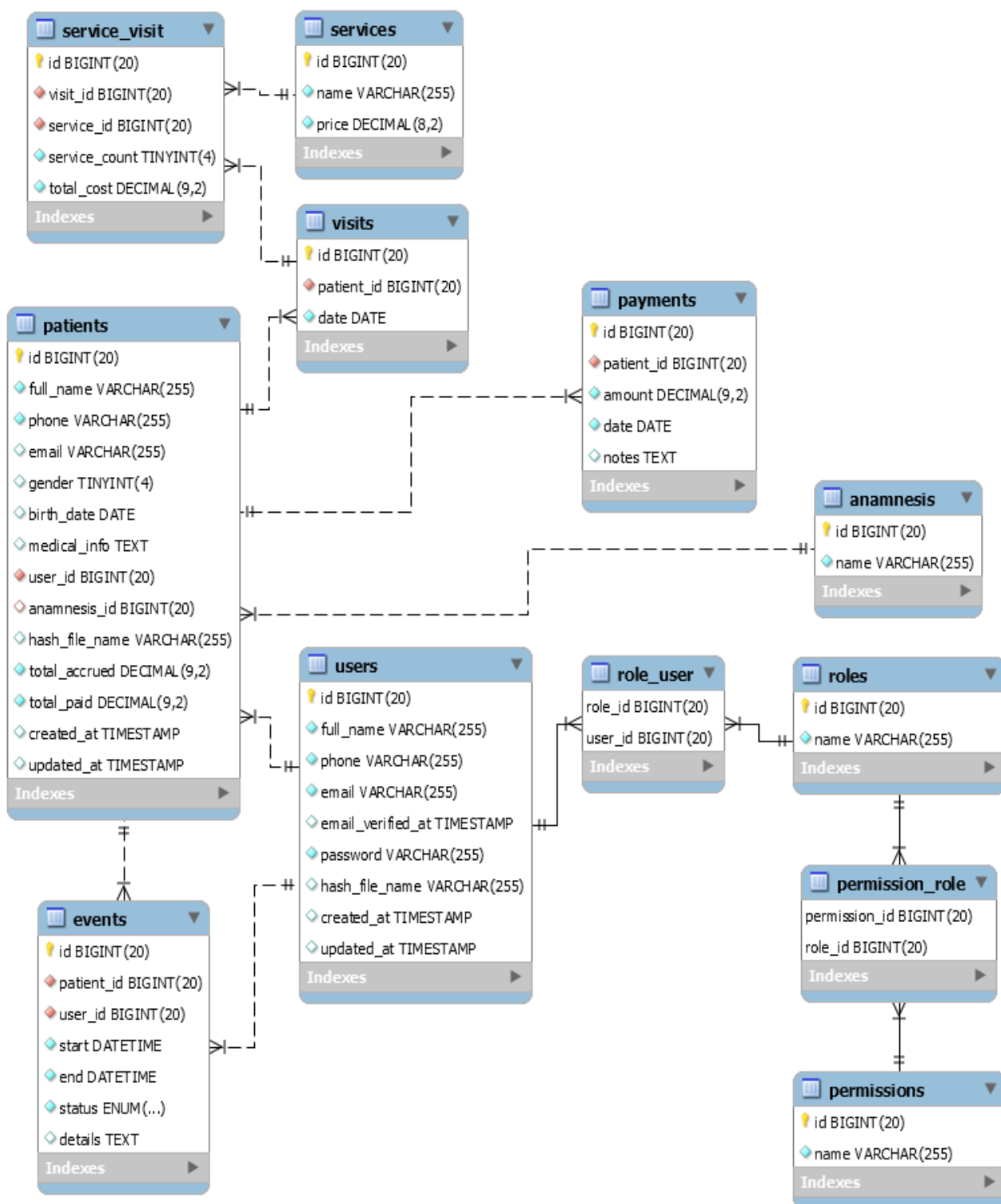


Рисунок 3.21 – Модель базы данных

4 ПРОГРАМНА РЕАЛІЗАЦІЯ МОДУЛІВ СИСТЕМИ

4.1 Налаштування підключення до БД

Як було сказано в попередніх розділах за зберігання, організацію і управління доступом до даних відповідає серверна сторона, що в цьому проєкті була реалізована за допомогою фреймворку Laravel. Однією з його переваг є наявність спеціалізованого модуля для роботи з базами даних – Eloquent ORM. За допомогою цього інструменту описуються сутності і їх зв'язки, визначається як та чи інша сутність відображається у базі. Він бере на себе значну частину роботи по генерації SQL-запитів, завантаженню даних та кастингу (перетворення типів бази даних в типи цільової мови). У результаті виходить, що Eloquent ORM ховає основну роботу з базою даних (вимагаючи тільки правильної конфігурації) і самостійно виконує усі необхідні запити. Однак, у складних випадках, можливостей цього модулю може не вистачити. Для вирішення подібних проблем Laravel містить Query Builder – зручний та виразний інтерфейс, який спрощує генерацію власного SQL-коду.

Laravel значно спрощує взаємодію з БД завдяки тому, що він працює трьох рівнях: чистий SQL, конструктор запитів (Query Builder), об'єктні моделі Eloquent ORM.

На даний момент фреймворк підтримує роботу з наступними СКБД: MySQL, PostgreSQL, SQLite і SQL Server. Однак, при розробці системи було використано саме MySQL, у зв'язку з її високою швидкістю роботи, надійністю і гнучкістю.

Важливим етапом перед початком роботи з базою даних є налаштування конфігурації. Файл з основними параметрами налаштування БД знаходиться в `config/database.php`. У ньому можна задати різні варіанти підключення до бази даних, обрати необхідну СКБД, а також визначити підключення за замовчуванням. Ключові параметри конфігурації наведено нижче:

```
'mysql' => [  
    'dsn' => 'mysql',
```

```

'host' => '127.0.0.1',
'port' => '3306',
'database' => 'dental_crm',
'username' => 'root',
'password' => '',
'charset' => 'utf8mb4',
]

```

В даному випадку, dsn містить назву драйвера mysql; host – доменне ім'я або IP-адресу сервера БД; database – назву бази даних, username і password – ім'я користувача з правами доступу до БД та його пароль; charset – кодування (використовується для коректного сортування і індексації даних).

Іноді виникає необхідність використати одно з'єднання для виконання запиту SELECT, а інше для запитів INSERT, UPDATE і DELETE. Laravel максимально спрощує цей процес, причому не важливо, які при цьому використовуються засоби – сирі запити, Query Builder чи Eloquent ORM. Розглянемо приклад подібної конфігурації:

```

'mysql' => [
    'read' => [ 'host' => '192.168.1.1' ],
    'write' => [ 'host' => '196.168.1.2' ],
    'dsn'   => 'mysql',
    'database' => 'database',
    'username' => 'root',
    'password' => "",
    'charset' => 'utf8',
]

```

З наведеного прикладу можна побачити, що в масиві налаштувань з'явилося два нових ключа: read і write, що дозволяють визначити адресу сервера (host) для

виконання відповідних операцій. Таким чином, 192.168.1.1 буде використовуватися для завантаження інформації, а 192.168.1.2 - для запису.

Для адміністрування сервера MySQL було використано допоміжне програмне забезпечення phpMyAdmin. За допомогою зручного веб-інтерфейсу phpMyAdmin дозволяє через браузер здійснювати управління базами даних MySQL, запускати запити SQL, переглядати та редагувати вміст таблиць. Оскільки цей інструмент в більшості випадків забезпечує можливість обійтися без самотійного введення команд SQL, то робота з базами даних є простою та швидкою.

Розглянемо процес створення нової бази даних за допомогою інтерфейсу phpMyAdmin. Для цього у рядку «Створити БД» необхідно ввести назву бази даних і обрати кодування (рис. 3.1). Для розробленої бази даних було обрано кодування utf8mb4_unicode_ci, що дозволить зберігати в таблицях символи кирилиці.

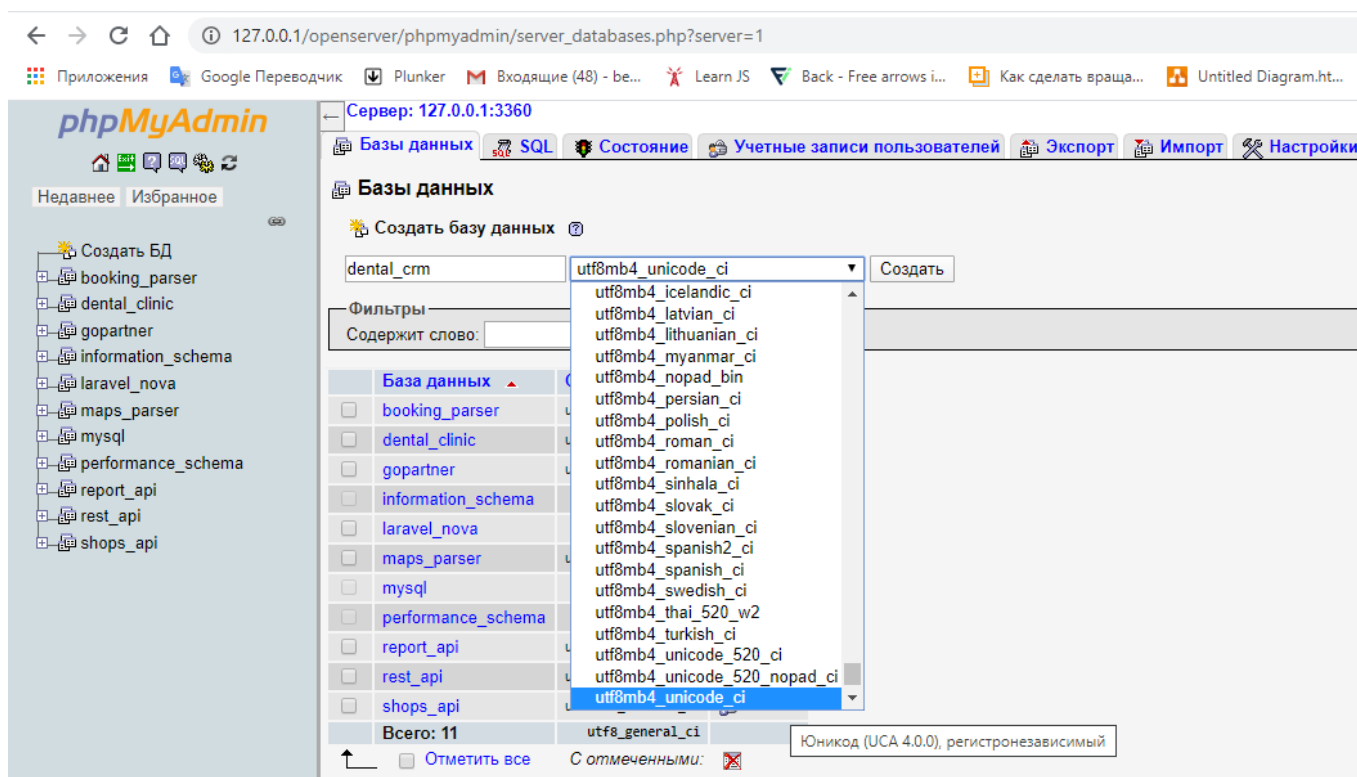


Рисунок 4.1 – Створення БД «dental_crm»

4.2 Налаштування міграцій бази даних

Після створення нової бази даних необхідно наповнити її відповідними таблицями для успішного функціонування системи. З цією метою було використано механізм міграцій Laravel.

В ході розробки і ведення баз даних структура таблиць розвивається, відповідно до вихідного коду веб-додатку. Наприклад, при розробці додатка може бути виявлено, що для підвищення продуктивності запитів має бути створений певний індекс. Міграції схеми бази даних, у свою чергу, дозволяють виправляти помилки та адаптувати дані до змін. Вони виконуються коли необхідно створити, оновити, або повернути схему бази даних до старішої версії.

Для створення міграції в Laravel використовується консольна команда `php artisan make:migration <назва таблиці>`. Нова міграція буде розташована в директорії `database/migrations`. Також, в її назві буде міститись мітка часу, яка дозволить фреймворку визначати порядок застосування міграцій. В наведеній команді можна також використати параметри `--table` і `--create` для вказівки імені таблиці і того факту, що міграція має створити нову таблицю, а не змінювати існуючу.

В структурі класу створеної міграції слід відзначити два основних методи: `up()` і `down()`. Метод `up()` використовується для додавання нових таблиць, стовпців або індексів в БД, а метод `down()` просто скасовує операції, виконані методом `up()`. В якості прикладу розглянемо клас для створення таблиці користувачів:

```
class CreateUsersTable extends Migration
{
    public function up()
    {
        Schema::create('users', function (Blueprint $table) {
            $table->bigIncrements('id');
            $table->string('full_name');
            $table->string('phone');
```

```

        $table->string('email')->unique();
        $table->timestamp('email_verified_at')->nullable();
        $table->string('password');
        $table->string('hash_file_name')->nullable();
        $table->timestamps();
    });
}

public function down()
{
    Schema::dropIfExists('users');
}
}

```

Для створення нової таблиці БД тут використовується метод `create` фасаду `Schema`. Він приймає два аргументи: ім'я таблиці, та функцію `Closure`, в якій можна визначити структуру нової таблиці. При необхідності редагування існуючої таблиці можна використати метод `table` фасаду `Schema`. Як і метод `create`, він приймає два аргументи: ім'я таблиці і замикання, що отримує екземпляр `Blueprint`, який можна використати для додавання нових стовпців. Для видалення таблиці в `down()` застосовується метод `dropIfExists`.

В розглянутому прикладі для визначення структури таблиці «users» використовуються наступні типи стовпців:

- `$table->bigIncrements('id')` - інкрементний ID (первинний ключ), що використовує еквівалент "UNSIGNED BIG INTEGER";
- `$table->string()` - еквівалент `VARCHAR`;
- `$table->timestamp()` - еквівалент `TIMESTAMP`;

Для деяких полів використовується модифікатор `->nullable()`, який вказує на те, що поле може бути пустим. Метод `$table->timestamps()` використовується для створення двох полів «created_at» та «updated_at» типу `TIMESTAMP`, що будуть

використані для відображення часу і дати створення запису в таблиці, а також її оновлення.

Запуск міграції виконується за допомогою команди `php artisan migrate`. Результат її виконання можна побачити використавши phpMyAdmin (рис. 3.2).

Сервер: 127.0.0.1:3360 » База данных: dental_crm » Таблица: users

Обзор Структура SQL Поиск Вставить Экспорт Импорт Привилегии Операции Триггеры

Структура таблицы Связи

#	Имя	Тип	Сравнение	Атрибуты	Null	По умолчанию	Комментарии	Дополнительно	Действие
☐	1 id	bigint(20)		UNSIGNED	Нет	Нет		AUTO_INCREMENT	
☐	2 full_name	varchar(255)	utf8mb4_unicode_ci		Нет	Нет			
☐	3 phone	varchar(255)	utf8mb4_unicode_ci		Нет	Нет			
☐	4 email	varchar(255)	utf8mb4_unicode_ci		Нет	Нет			
☐	5 email_verified_at	timestamp			Да	NULL			
☐	6 password	varchar(255)	utf8mb4_unicode_ci		Нет	Нет			
☐	7 hash_file_name	varchar(255)	utf8mb4_unicode_ci		Да	NULL			
☐	8 created_at	timestamp			Да	NULL			
☐	9 updated_at	timestamp			Да	NULL			

↑ ☐ Отметить все С отмеченными:

Рисунок 4.2 – Структура таблиці «users»

4.3 Розробка основних модулів системи

Основні модулі, необхідні для реалізації усіх функціональних можливостей системи було визначено на етапі проектування архітектури. До них можна віднести: модуль авторизації, модуль пацієнтів, модуль користувачів, модуль послуг, модуль реєстратури, модуль аналітики. Послідовно розглянемо розробку деяких з них.

4.3.1 Розробка модуля авторизації

Головним завданням модулю авторизації є керування рівнями доступу до ресурсів системи, тобто надання користувачам прав на виконання певних дій в

залежності від їх ролі. Передбачається, що в розробленій системі буде наступне розподілення ролей: клієнти, які мають доступ лише до можливості перегляду інформації та запису на прийом, адміністратори, які можуть спостерігати за всім процесом роботи системи та працівники.

Ключовим елементом даного модулю є форма авторизації (вікно для введення логіну та паролю). Код, що відповідає за її відображення наведено нижче:

```
<h1 class="form-title">Авторизація</h1>
<v-card class="form-card elevation-2">
  <v-card-text class="card-content">
    <v-form>
      <v-text-field
        label="Email"
        name="email"
        prepend-icon="email"
        type="email"
        v-model="email"
        maxlength="255"
        :error-messages="emailErrors"
        @blur="$v.email.$touch()"
        @keyup.native.enter="submit"
      ></v-text-field>
      <v-text-field
        label="Password"
        name="password"
        prepend-icon="lock"
        :append-icon="showPassword ? 'mdi-eye' : 'mdi-eye-off'"
        @click:append="showPassword = !showPassword"
        :type="showPassword ? 'text' : 'password'"
        v-model="password"
        maxlength="20"
        :error-messages="passwordErrors"
```

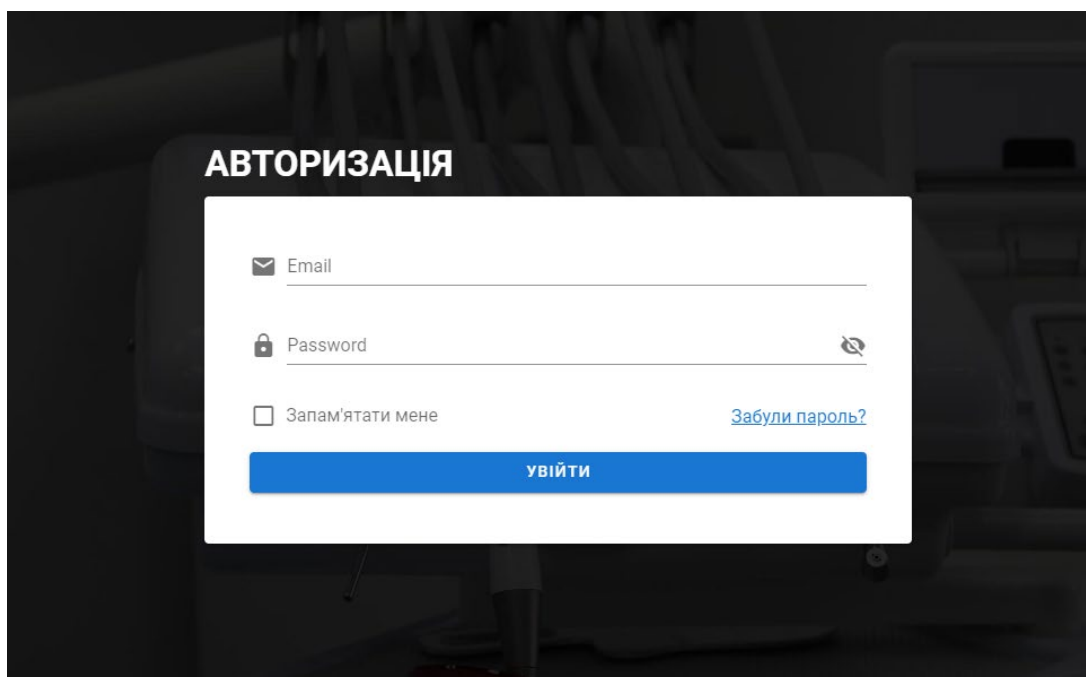
```

    @blur="$v.password.$touch()"
    @keyup.native.enter="submit"
  ></v-text-field>
  <div class="remember">
    <v-checkbox
      label="Запам'ятати мене"
      color="primary"
      hide-details
      v-model="remember"
      class="mt-2"
    ></v-checkbox>
    <router-link :to="/forgot-password">Забули пароль?</router-link>
  </div>
</v-form>
</v-card-text>
<v-card-actions class="card-actions">
  <v-btn class="form-submit" @click.prevent="submit">Login</v-btn>
</v-card-actions>
</v-card>

```

В розглянутій частині коду можна побачити HTML елементи, що не відповідають жодному зі стандартних тегів, як наприклад `<v-card></v-card>`. В такий спосіб Vue.js дозволяє впровадити в структуру сторінки власні компоненти – невеликі частини інтерфейсу користувача з ізольованими стилями і скриптами. Кожен з них реалізує певний функціонал, а при необхідності їх можна з легкістю використати знову в іншому місці. Компонент, який в DOM виглядає як тег з нестандартною назвою, може включати не лише HTML-розмітку, але і логіку своєї роботи і використання інших компонентів. Використання компонентного підходу дозволяє розділяти складні сторінки на менші частини, які легше підтримувати.

Зовнішній вигляд форми, що є результатом виконання фрагменту коду, наведеного вище, можна побачити на рисунку 3.3.



The image shows a web form for authorization. At the top, the title 'АВТОРИЗАЦІЯ' is displayed in white bold text on a dark background. Below the title is a white rectangular form containing the following elements: an 'Email' input field with an envelope icon, a 'Password' input field with a lock icon and a toggle eye icon, a checkbox labeled 'Запам'ятати мене', a blue link 'Забули пароль?', and a blue button labeled 'УВІЙТИ'.

Рисунок 4.3 – Форма авторизації

Одним з головних принципів архітектури REST, яка застосовувалася в ході реалізації методів API (інтерфейсу для взаємодії клієнта з сервером), є *stateless* – незалежність від стану. Це означає, що сервер не може зберігати інформацію про користувача між запитами. Виходячи з цього, кожен запит з клієнтської сторони повинен містити усі необхідні параметри для його обробки. Такий підхід потребує відправки на сервер секретного токена доступу для ідентифікації користувача і отримання прав на виконання певних операцій.

Розглянемо особливості процесу авторизації з використанням токенів. Після успішного входу в систему на сервері буде згенерована унікальна послідовність символів фіксованої довжини – токен, що ідентифікує поточну сесію. В якості відповіді при виконанні процедури авторизації токен буде переданий на сторону клієнта для подальшого збереження. Надалі його можна буде використовувати в ході відправки запитів для підтвердження особи користувача. Зазвичай, для цього використовується спеціальний `http` -заголовок `Bearer: <token>`. Після завершення сеансу роботи (процедура `logout`), токен стає не дійсним і видаляється на стороні клієнта. При спробі отримати доступ до захищених ресурсів системи без

відповідних параметрів авторизації сервер поверне відповідну помилку – 401 Unauthorized.

Код контролера Laravel, який відповідає за логіку авторизації, наведено нижче:

```
class AuthController extends Controller {
    public function login(AuthLogin $request) {
        $credentials = $request->only('email', 'password');
        if (!$auth()->attempt($credentials)) {
            return response()->api(null, 422);
        }
        $user = User::where('email', $request->email)->first();
        return $this->respondWithToken($user);
    }

    public function user() {
        $user = auth()->user();
        return $this->respondWithToken($user);
    }

    private function respondWithToken(Authenticatable $user) {
        return response()->api([
            'full_name' => $user->full_name,
            'email' => $user->email,
            'email_verified_at' => $user->email_verified_at,
            'access_token' => $user->api_token,
            'roles' => $user->roles->pluck('name')
        ]);
    }
}
```

У класі AuthController можна виділити такі методи:

– login() – відповідає за обробку облікових даних, відправлених з форми авторизації. У цьому методі виконується пошук користувача по логіну і пароллю, за що відповідає auth() ->attempt (\$credentials). Якщо за цими даними у БД немає збігів – з сервера прийде повідомлення про помилку (response () ->api (null, 403)). У разі успіху операції метод поверне токен доступу разом з допоміжною інформацією (список ролей, дата підтвердження електронної адреси).

– user() – завантажує інформацію про користувача на основі переданого токена авторизації (auth () ->user()) і повертає її у вигляді JSON об'єкту.

4.3.2 Розробка адміністративної частини

Адміністративна частина, що призначена тільки для співробітників стоматологічної клініки має досить широкий спектр можливостей, серед яких:

- перегляд контактних даних пацієнтів та інформації стосовно їх візитів;
- контроль фінансів;
- ведення розкладу прийомів;
- перегляд звітності для контролю ключових показників роботи клініки.

Адміністративна частина є основною складовою CRM-системи, доступ до якої здійснюється за допомогою розпізнавання ролі користувача. Використання механізмів авторизації та розмежування прав доступу дозволяє убезпечити систему від несанкціонованого доступу. При проектуванні адміністративної частини була мета забезпечити повний контроль над роботою стоматології, а завдяки простому та інтуїтивно зрозумілому інтерфейсу освоєння її можливостей не відніме багато часу.

Для навігації по сторінкам адміністративної частини, використовується меню, зображене на рисунку 3.4.

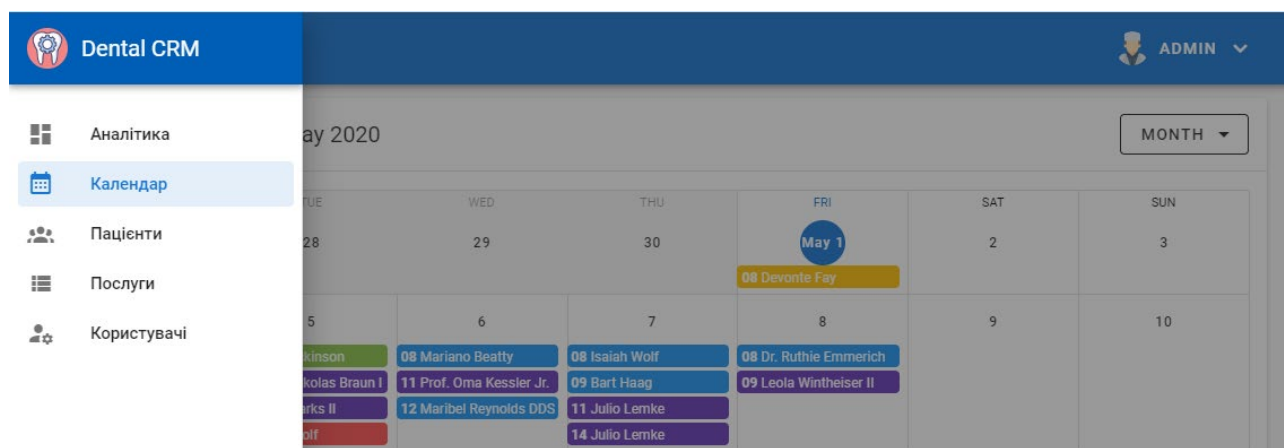


Рисунок 4.4 – Головне меню

Відображення пунктів меню та права доступу до відповідних сторінок залежать від ролі авторизованого користувача. Так, адміністратор системи, на відміну від звичайного користувача, має змогу використовувати можливості модулів аналітики, послуг та користувачів.

Код Vue.js компоненту, що використовується для відображення меню можна побачити нижче:

```
<template>
```

```
  <v-navigation-drawer class="app__drawer" :mini-variant.sync="mini" app v-model="drawer"
:width="drawWidth">
```

```
    <v-toolbar color="primary darken-1" dark>
```

```
      
```

```
      <v-toolbar-title class="ml-0 pl-3">
```

```
        <span class="hidden-sm-and-down">Dental clinic</span>
```

```
      </v-toolbar-title>
```

```
    </v-toolbar>
```

```
    <v-list dense nav expand class="py-6">
```

```
      <v-list-item-group color="primary">
```

```
        <v-list-item
```

```
          v-for="item in menu"
```

```
          ripple="ripple"
```

```
          :to="item.name ? { name: item.name } : null"
```

```

      :key="item.name"
    >
      <v-list-item-icon v-if="item.icon">
        <v-icon>{{ item.icon }}</v-icon>
      </v-list-item-icon>
      <v-list-item-content>
        <v-list-item-title>{{item.title}}</v-list-item-title>
      </v-list-item-content>
    </v-list-item>
  </v-list-item-group>
</v-list>
</v-navigation-drawer>
</template>

```

```

<script>
import menu from "@/js/config/menu";
export default {
  props: {
    drawWidth: {
      type: [Number, String],
      default: "260"
    },
  },
  data() {
    return {
      mini: false,
      scrollSettings: {
        maxScrollbarLength: 160
      }
    }
  },
  computed: {
    logo() {

```

```

        return "/storage/images/logo.svg";
    },
    isAdmin() {
        return this.$store.getters['auth/isAdmin'];
    },
    menu() {
        return menu.filter(item => item.admin ? this.isAdmin : true);
    },
    drawer: {
        get: function () {
            return this.$store.state.app.drawer;
        },
        set: function (newValue) {
            this.$store.dispatch('app/setDrawerState', newValue);
        }
    },
},
}
</script>

```

Фрагмент, що відповідає за фільтрацію елементів навігації в залежності від ролі знаходиться в методі menu().

4.3.3 Модуль пацієнтів

Модуль пацієнтів являє собою автоматизоване робоче місце лікаря, мета якого спростити виконання рутинних завдань. Головною перевагою цього модуля є зручна робота з клієнтською базою.

При переході на головну сторінку даного модуля першим, що побачить користувач системи буде таблиця, яка містить основну інформацію про пацієнтів: ПІБ, контактні дані тощо (рис. 3.5).

Пацієнти

+ ДОДАТИ

ID	ПІБ	Телефон	Email	Стать	Дата народження	
1	Maribel Reynolds DDS	+38 (052) 795-88-34	orion67@example.org	female	19-07-1974	⋮
2	Nyah Dickinson	+38 (074) 481-49-84	willard90@example.org	female	18-03-2020	⋮
3	Ryleigh Effertz	+38 (082) 448-95-54	kerluke.columbus@example.net	male	09-09-1980	⋮
4	Elena Marks II	+38 (054) 900-72-08	gay.stiedemann@example.net	male	14-10-1995	⋮
5	Mariano Beatty	+38 (064) 816-26-06	keyon.farrell@example.net	female	20-03-2018	⋮
6	Coty Maggio	+38 (077) 460-60-25	zkihn@example.org	male	26-11-2003	⋮
7	Amari Prohaska	+38 (027) 150-98-51	vwindler@example.com	male	06-11-2013	⋮
8	Dawson Block	+38 (019) 519-51-72	george07@example.org	female	21-07-2009	⋮
9	Leola Wintheiser II	+38 (067) 602-70-56	lindgren.sven@example.org	female	20-06-2009	⋮
10	Bart Haag	+38 (072) 672-73-94	amara28@example.com	female	02-04-1988	⋮

Rows per page: 15
1-15 of 20
< >

Рисунок 4.5 – Таблиця пацієнтів

Відображенням списку пацієнтів займається компонент `Patients.vue`. Серед інших методів цього компоненту можна виділити `loadPatients()`, завданням якого є асинхронне завантаження даних при ініціалізації:

```
loadPatients() {
  let params = {
    q: this.search,
    page: this.options.page || 1,
    limit: this.options.itemsPerPage || 15,
    sort_by: this.options.sortBy || [],
    sort_desc: this.options.sortDesc || [],
  };
}
```

```

    this.isLoading = true;
    this.$store.dispatch('patients/loadPatients', { params })
      .finally(() => {
        this.isLoading = false;
      });
  }

```

В наведеній частині коду можна побачити js об'єкт `params`, що буде відправлений на сервер у вигляді query параметрів для реалізації пошуку (`q: this.search`), пагінації (`page: this.options.page`, `limit: this.options.itemsPerPage`) та сортування (`sort_by: this.options.sortBy`, `sort_desc: this.options.sortDesc`). З урахуванням переданих параметрів кінцева url для завантаження списку пацієнтів буде виглядати наступним чином: `<base_url>/api/patients?limit=15&page=2&q=test&sort_by=full_name&sort_desc=false`.

Зі сторони сервера, у свою чергу, було реалізовано контролер для обробки запиту по завантаженню даних пацієнтів. Розглянемо метод, що відповідає за виконання цієї операції:

```

public function index(PatientsIndex $request)
{
    $limit = $request->get('limit') ?? 10;
    $patients = Patient::query()
        ->when($request->get('q'), function(Builder $query) use (&$request) {
            $query->where('id', 'LIKE', '%'. $request->get('q') .'%')
                ->orWhere('full_name', 'LIKE', '%'. $request->get('q') .'%')
                ->orWhere('phone', 'LIKE', '%'. $request->get('q') .'%');
        })
        ->when($request->get('sort_by'), function(Builder $query) use (&$request) {
            $direction = $request->get('sort_desc');
            $query->orderBy($request->get('sort_by'), (isset($direction)
                && json_decode($direction)) ? 'DESC' : 'ASC');

```

```

    })
    ->orderBy('created_at', 'DESC')
    ->paginate($limit);
    return response()->api($patients);
}

```

В даному методі використовується засоби Eloquent ORM для завантаження необхідної інформації з БД. При цьому, за допомогою об'єкту request в автоматично згенерований SQL-запит будуть передані вилучені query параметри. Так, 'sort_by' визначає стовпець, за яким буде проведено сортування результатів вибірки:

```

->when($request->get('sort_by'), function(Builder $query) use (&$request) {
    $direction = $request->get('sort_desc');
    $query->orderBy($request->get('sort_by'), (isset($direction)
    && json_decode($direction)) ? 'DESC' : 'ASC');
})

```

Згенерований SQL-запит:

```

SELECT *
FROM `patients`
WHERE `id` LIKE <query>
    OR `full_name` LIKE <query>
    OR `phone` LIKE <query>
ORDER BY `full_name` ASC, `created_at` DESC
LIMIT 15 offset 0

```

Результатом виконання методу index() стане JSON об'єкт, сформований на основі інформації, отриманої за допомогою наведеного SQL-запиту:

```

{
    "status":1,

```

```

"meta":{
  "http_code":200,
  "pagination":{
    "limit":15,
    "current_page":1,
    "total":1,
    "last_page":1
  }
},
"response":[
  {
    "id":1,
    "full_name":"Maribel Reynolds DDS",
    "phone":"+38 (052) 795-88-34",
    "email":"orion67@example.org",
    "gender":1,
    "birth_date":"1974-07-19",
    "medical_info":"Inventore molestias iusto in nihil nam sunt expedita.",
    "user_id":7,
    "anamnesis_id":5,
    "original_file_name":null,
    "hash_file_name":null,
    "total_accrued":"26035.00",
    "total_paid":"31767.00",
    "created_at":"2020-05-01 13:00:50",
    "updated_at":"2020-05-01 13:00:50"
  },
]
}

```

4.3.4 Модуль реєстрації

Іншою, не менш важливою складовою системи є модуль реєстрації. За допомогою цього модуля адміністратор клініки може переглядати час прийому відразу декількох фахівців одночасно, легко і швидко управляти статусами візитів. При цьому у лікарів є можливість контролювати свій розклад з особистого кабінету – відмічати виконання послуг, бачити скасовані прийоми і нещодавно записаних пацієнтів, а також переносити записи на інший час і дату.

Для наочного відображення інформації відносно запису пацієнтів використовується календар подій (рис. 3.6). Щоб дати лікарям можливість швидко орієнтуватися в розкладі прийомів і статусах візитів, додатково було реалізовано кодування записів за кольорами. Також можна обрати декілька варіантів відображення: місяць, тиждень або день.

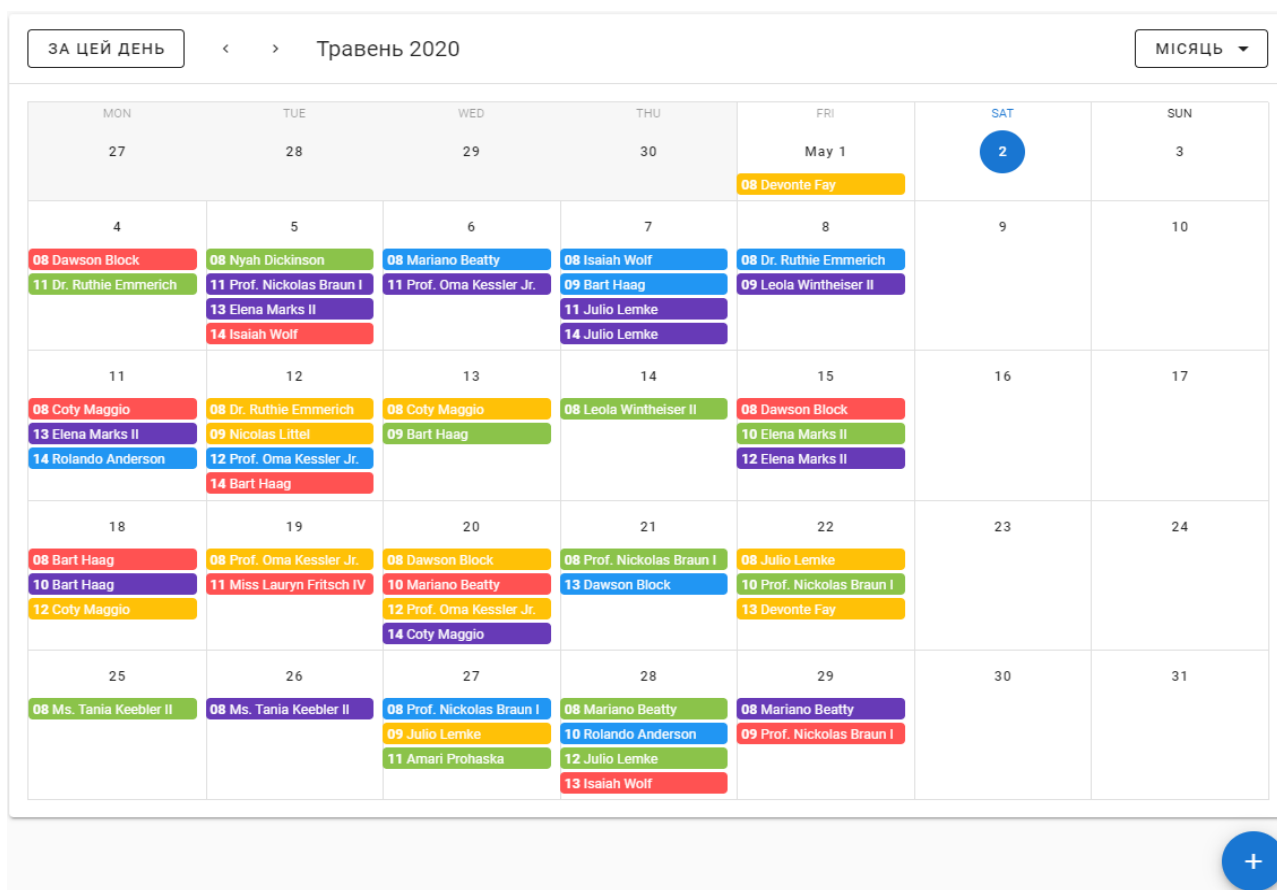


Рисунок 4.6 – Календар подій

Розглянемо код, що відповідає за внесення до бази даних інформації відносно запису пацієнта на прийом:

```
public function store(EventsStore $request)
{
    $event = Event::create([
        'patient_id' => $request->patient_id,
        'user_id' => auth()->user()->id,
        'start' => $request->start,
        'end' => $request->end,
        'status' => $request->status,
        'details' => $request->details ?? null,
    ]);
    return response()->api($event);
}
```

В наведеному фрагменті частина `Event::create()` дозволяє додати новий рядок в таблицю `Events`. Для цього використовуються значення, що були надіслані з відповідної форми на сторінці модуля реєстрації (`$request->patient_id`, `$request->start`, `$request->end` і тд).

4.3.5 Модуль аналітики

Метою модуля аналітики є збір та представлення статистичної інформації щодо основних показників діяльності медичної установи для оцінки ефективності роботи клініки. Інформація для аналізу відображається за допомогою графіків та діаграм (рис. 3.7), що дозволяє полегшити сприйняття статистичного матеріалу, зробити його наочним і доступним. Основною перевагою розробленого модуля є можливість фільтрації результатів за періодом часу, співробітником чи пацієнтом.

Модуль аналітики включає в себе графічні звіти за такими показниками:

- кількість візитів;
- вартість наданих послуг;
- сума платежів;
- розподіл пацієнтів за віком та статтю.

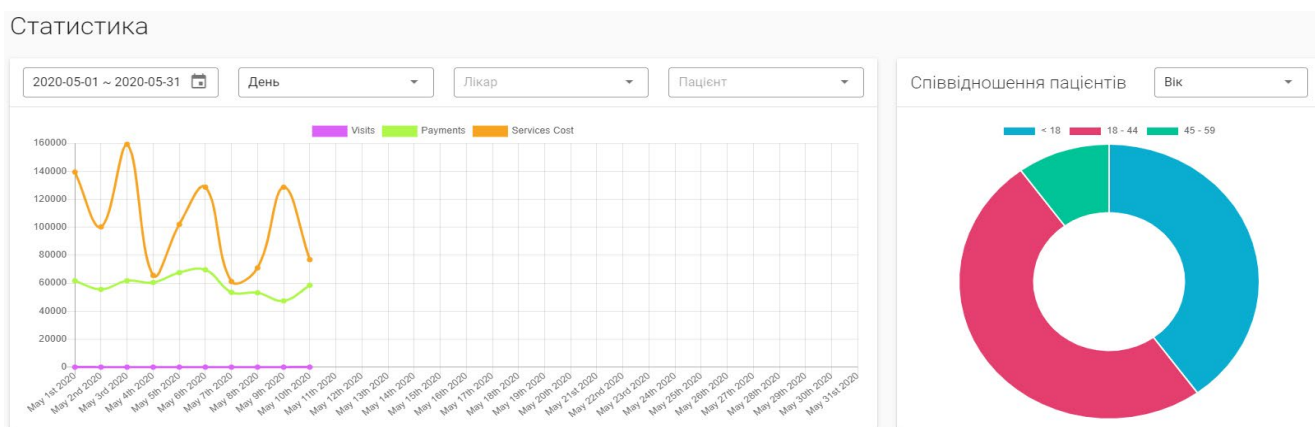


Рисунок 4.7 – Відображення статистики

Для відображення звітів за показниками роботи стоматології необхідно виконати попередній збір інформації з БД. В якості прикладу розглянемо SQL-запит для завантаження статистики відносно вартості наданих послуг:

```
SELECT Sum(`service_visit`.`total_cost`) AS chart_value, Max(`visits`.`date`) AS max_date
FROM `visits`
INNER JOIN `service_visit` ON `visits`.`id` = `service_visit`.`visit_id`
WHERE `patient_id` = '13'
    AND Date(`date`) >= '2020-05-01'
    AND Date(`date`) <= '2020-05-31'
GROUP BY Date(date)
ORDER BY `max_date` ASC
```

Наведений запит дозволяє отримати нараховану суму для візитів пацієнта (WHERE `patient\_id` = '13'), що потрапили в заданий проміжок часу (WHERE Date(`date`) >= '2020-05-01' AND Date(`date`) <= '2020-05-31').

5 ТЕСТУВАННЯ РОЗРОБЛЕНОЇ СИСТЕМИ

Тестування, як завершальний етап розробки системи, відіграє життєво важливу роль у процесі створення якісного програмного забезпечення. У кінці проекту, коли всі сутності системи синхронізуються, можуть виникнути помилки. Таким чином, після реалізації основного функціоналу системи, слід перевірити, що всі елементи працюють так, як було заплановано.

Під час тестування слід звернути особливу увагу на працездатність серверного коду, оскільки саме він відповідає за критично важливі частини, без яких нормальне функціонування системи неможливе. Варто переконатись, що інформація з бази даних дійсно досягає клієнтської сторони в правильному форматі. REST використовує стандартні HTTP коди стану для підтвердження успішності запиту, тож необхідно перевірити, що API повертає очікувані коди, особливо в сценаріях помилок.

Існує ряд зовнішніх інструментів, які можна застосувати для тестування API, однак використання вбудованих засобів Laravel є набагато кращою альтернативою. Тестування усередині Laravel дозволяє здійснювати перевірку структури і результатів API, зберігаючи при цьому повний контроль над базою даних.

В якості прикладу розглянемо як відбувається перевірка кінцевих точок, що використовуються для завантаження інформації про послуги стоматологічної клініки. Для створення тесту скористаємося консольною командою `php artisan make:test ServicesTest`. Вона розмістить новий клас `ServicesTest` в директорії `tests`. Далі слід визначити методи, що дозволять перевірити виконання таких операцій: завантаження повного переліку послуг, додавання послуги, оновлення послуги, видалення послуги.

Розроблені методи для тестування контролера `ServicesController`, який відповідає за виконання операцій над сутністю «Послуга», наведено нижче:

```

class ServicesTest extends TestCase {

    use RefreshDatabase;

    public function testServicesAreListedCorrectly() {
        factory(Service::class)->create([
            'name' => 'First Service',
            'price' => '100',
        ]);
        factory(Service::class)->create([
            'name' => 'Second Service',
            'price' => '200',
        ]);
        $user = factory(User::class)->create();
        $token = $user->api_token;
        $headers = ['Authorization' => "Bearer $token"];
        $this->json('GET', '/api/services', [], $headers)
            ->assertStatus(200)
            ->assertJsonStructure(['response' =>
                [
                    '*' => ['id', 'name', 'price', 'created_at', 'updated_at']
                ]
            ])
            ->assertJson(['response' =>
                [
                    ['name' => 'First Service', 'price' => '100'],
                    ['name' => 'Second Service', 'price' => '200']
                ]
            ]);
    }

    public function testsServicesAreCreatedCorrectly() {
        $user = factory(User::class)->create();
        $token = $user->api_token;
    }
}

```

```

$headers = ['Authorization' => "Bearer $token"];
$payload = [
    'name' => 'Test Service',
    'price' => '100',
];
$this->json('POST', '/api/services', $payload, $headers)
    ->assertStatus(200)
    ->assertJsonStructure([
        'response' => ['id', 'name', 'price', 'created_at', 'updated_at']
    ])
    ->assertJson(['response' =>
        [ 'name' => $payload['name'], 'price' => $payload['price'] ]
    ]);
// Assert that service was added to database
$this->assertEquals(1, Service::count());
}

public function testsServicesAreUpdatedCorrectly() {
    $user = factory(User::class)->create();
    $token = $user->api_token;
    $headers = ['Authorization' => "Bearer $token"];
    $service = factory(Service::class)->create([
        'name' => 'Test Service',
        'price' => '100',
    ]);
    // Assert that service was added to database
    $this->assertEquals(1, Service::count());
    $payload = [
        'name' => 'Changed Service',
        'price' => '200',
    ];
    $this->json('PUT', '/api/services/' . $service->id, $payload, $headers)
        ->assertStatus(200)

```

```

->assertJsonStructure([
    'response' => ['id', 'name', 'price', 'created_at', 'updated_at']
])
->assertJson(['response' =>
    [ 'id' => $service->id, 'name' => $payload['name'], 'price' => $payload['price'] ]
]);
}

public function testsServicesAreDeletedCorrectly() {
    $user = factory(User::class)->create();
    $token = $user->api_token;
    $headers = ['Authorization' => "Bearer $token"];
    $service = factory(Service::class)->create([
        'name' => 'Test Service',
        'price' => '100',
    ]);
    $this->assertEquals(1, Service::count());
    $this->json('DELETE', '/api/services/' . $service->id, [], $headers)
        ->assertStatus(200);
    // Assert that service was deleted from database
    $this->assertEquals(0, Service::count());
}
}

```

Для запуску тестів потрібно виконати команду `\vendor\bin\phpunit` в терміналі. Результат можна побачити на рисунку 4.1.



```

PS D:\Projects\Laravel\dental-site\src> vendor\bin\phpunit
PHPUnit 8.5.2 by Sebastian Bergmann and contributors.

....                                                    4 / 4 (100%)

Time: 12.9 seconds, Memory: 26.00 MB

OK (4 tests, 35 assertions)

```

Рисунок 5.1 – Результат виконання тестів

ВИСНОВКИ

У даній роботі була спроектована система управління взаємодією з клієнтами стоматологічної клініки. Особливий акцент робився на забезпеченні доступу до ресурсу як зі стаціонарних комп'ютерів, так і з мобільних пристроїв, що дало б змогу лікарям отримувати та вносити інформацію про пацієнтів безпосередньо на робочих місцях.

Після визначення основної задачі було проведено пошук і порівняльний аналіз готових рішень. Враховуючи всі недоліки та переваги розглянутих аналогів були сформовані вимоги до функціональних можливостей системи.

Наступним етапом був варіантний аналіз існуючих технологій та методів створення веб-додатків і обґрунтування їх вибору для процесу розробки. Так, було прийнято рішення реалізувати систему на основі архітектури SPA, що дозволить побудувати зручний інтерфейс з високим ступенем інтерактивності. Крім цього були обрані фреймворки і бібліотеки, які б допомогли у вирішенні поставленого завдання.

По завершенні попереднього аналізу було проведено проектування архітектури та програмна реалізація модулів системи. Після чого, з метою оцінки продуктивності роботи, функціональних можливостей та якості готового програмного забезпечення виконано тестування системи. Проведене тестування підтвердило, що розроблений програмний продукт працює стабільно та повністю придатний для використання.

ПЕРЕЛІК ПОСИЛАНЬ

1. **Затулов С. Н.** Що таке CRM і як його впровадити на підприємстві [Текст] / С.Н. Затулов // *Економічний вісник фармації*, 2003. – № 9. – С. 31-41.
2. **Рожков І. В.** Інформаційні системи і технології в маркетингу: монографія [Текст] / І. В. Рожков // 2014. – 199 с.
3. **Птащенко О. В.** Побудова CRM-системи як основи формування комунікаційної політики між організацією та кінцевим споживачем [Текст] / О. В. Птащенко // *Вісник Східноукраїнського національного університету ім. Володимира Даля*, 2016. – № 6. – С. 108-115.
4. **Тарасенко Е. А.** Управління клієнтською політикою на основі технологій CRM (на прикладі приватної лікувально-профілактичної установи) [Текст] / Е. А. Тарасенко , Т. Б. Рижкова // *Вісник Російського державного гуманітарного університету*, 2015. – № 3(146). – С. 89-99.
5. Огляд CRM-систем для стоматології [Електронний ресурс]. – Режим доступу: https://crm-systems.info/dlja-stomatologii/#_CRM (дата звернення: 22.03.2020).
6. Принципи роботи і структура web-додатків на основі ASP.NET [Електронний ресурс]. – Режим доступу: <https://www.sibsau.ru/sveden/edufiles/145507/> (дата звернення: 25.03.2020).
7. Методика проектування систем Web 2.0 [Електронний ресурс]. – Режим доступу: https://crm-systems.info/dlja-stomatologii/#_CRM (дата звернення: 25.03.2020).
8. Багатосторінкові та односторінкові додатки, їх переваги та недоліки [Електронний ресурс]. – Режим доступу: <https://dou.ua/forums/topic/25444/> (дата звернення: 29.03.2020).
9. Single-page application vs multiple-page application [Електронний ресурс]. – Режим доступу: <https://neoteric.eu/blog/single-page-application-vs-multiple-page-application/>

?utm_source=medium.com&utm_medium=social&utm_content=neo&utm_campaign=blog (дата звернення: 29.03.2020).

10. **Горбунов-Посадов М. М.** Розширювані програми [Текст] / М. М. Горбунов-Посадов // Москва : Полиптих, 1999. – 336 с.

11. **Проценко М. М.** Аналіз фреймворків як засобів розробки Web-додатків [Текст] / М. М. Проценко // Міжнародний науковий журнал, 2016. – № 6(1). – С. 86-89.

12. Django [Електронний ресурс]. – Режим доступу: <https://ru.wikipedia.org/wiki/Django> (дата звернення: 29.03.2020).

13. Laravel [Електронний ресурс]. – Режим доступу: <https://ru.wikipedia.org/wiki/Laravel> (дата звернення: 29.03.2020).

14. Advantages of Web Development with Ruby on Rails [Електронний ресурс]. – Режим доступу: <https://onix-systems.com/blog/advantages-of-web-development-with-Ruby-on-Rails> (дата звернення: 29.03.2020).

15. Vue.js – Overview [Електронний ресурс]. – Режим доступу: https://www.tutorialspoint.com/vuejs/vuejs_overview.htm (дата звернення: 30.03.2020).

16. React [Електронний ресурс]. – Режим доступу: <https://uk.wikipedia.org/wiki/React> (дата звернення: 30.03.2020).

17. Introduction to Angular concepts [Електронний ресурс]. – Режим доступу: <https://angular.io/guide/architecture> (дата звернення: 30.03.2020).

18. **Буч Г.** Мова UML. Інструкція користувача [Текст] / Г. Буч, Д. Рамбо, А. Джекобсон // Вид. 2. – СПб. : ДМК Пресс, 2004. – 432 с.

19. **Fielding R.** Architectural Styles and the Design of Network-based Software Architectures [Електронний ресурс] / R. Fielding. – Режим доступу: <https://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm>. (дата звернення: 07.04.2020).

20. Нормалізація реляційних баз даних [Електронний ресурс]. – Режим доступу: <https://club.shelek.ru/viewart.php?id=177> (дата звернення: 04.04.2020).

ДОДАТОК А

Код серверної частини

```
<?php
namespace App\Http\Controllers\Api\Auth;
use App\Http\Controllers\Controller;
use App\Http\Requests\Auth\Register as AuthRegister;
use App\Http\Requests\Auth>Login as AuthLogin;
use App\Models\User;
use Illuminate\Contracts\Auth\Authenticatable;
use Illuminate\Auth\Events\Registered;
use Illuminate\Support\Facades\Hash;
use Illuminate\Support\Str;

class AuthController extends Controller
{
    public function login(AuthLogin $request)
    {
        $credentials = $request->only('email', 'password');
        if (!auth()->attempt($credentials)) {
            return response()->api(null, 422);
        }
        $user = User::where('email', $request->email)->first();
        return $this->respondWithToken($user);
    }

    public function user()
    {
        $user = auth()->user();
        return $this->respondWithToken($user);
    }
}
```

```

private function respondWithToken(Authenticatable $user)
{
    return response()->api([
        'user' => [
            'id' => $user->id,
            'full_name' => $user->full_name,
            'email' => $user->email,
            'email_verified_at' => $user->email_verified_at,
            'access_token' => $user->api_token,
            'roles' => $user->roles->pluck('name')
        ]
    ]);
}
}

```

```

<?php
namespace App\Http\Controllers\Api\Auth;
use App\Http\Controllers\Controller;
use Illuminate\Foundation\Auth\SendsPasswordResetEmails;
use Illuminate\Http\Request;

class ForgotPasswordController extends Controller
{
    use SendsPasswordResetEmails;

    protected function sendResetLinkResponse(Request $request, $response)
    {
        return response()->api(trans($response));
    }

    protected function sendResetLinkFailedResponse(Request $request, $response)
    {
        return response()->api(trans($response), 422);
    }
}

```

```

    }
}

```

```

<?php
namespace App\Http\Controllers\Api\Auth;
use App\Http\Controllers\Controller;
use Illuminate\Foundation\Auth\ResetsPasswords;
use Illuminate\Http\Request;

class ResetPasswordController extends Controller
{
    use ResetsPasswords;

    protected function sendResetResponse(Request $request, $response)
    {
        return response()->api(trans($response));
    }

    protected function sendResetFailedResponse(Request $request, $response)
    {
        return response()->api(trans($response), 422);
    }
}

```

```

<?php
namespace App\Http\Controllers\API\Auth;
use App\Http\Controllers\Controller;
use Illuminate\Foundation\Auth\VerifiesEmails;
use Illuminate\Http\Request;
use Illuminate\Auth\Events\Verified;

class VerificationController extends Controller
{

```

```
use VerifiesEmails;
```

```
public function __construct()
```

```
{
```

```
    $this->middleware('auth');
```

```
    $this->middleware('signed')->only('verify');
```

```
    $this->middleware('throttle:6,1')->only('verify', 'resend');
```

```
}
```

```
public function verify(Request $request)
```

```
{
```

```
    $user = $request->user();
```

```
    if ($request->route('id') == $user->getKey() &&
```

```
        $user->markEmailAsVerified()) {
```

```
        event(new Verified($user));
```

```
    }
```

```
    return response()->api(null, 200);
```

```
}
```

```
public function resend(Request $request)
```

```
{
```

```
    if ($request->user()->hasVerifiedEmail()) {
```

```
        return response()->api(null, 422);
```

```
    }
```

```
    $request->user()->sendEmailVerificationNotification();
```

```
    return response()->api(null, 200);
```

```
}
```

```
}
```

```
<?php
```

```
namespace App\Http\Controllers\Api;
```

```
use App\Http\Controllers\Controller;
```

```
use App\Http\Requests\Anamnesis\Index as AnamnesisIndex;
```

```
use App\Models\Anamnesis;
```

```
class AnamnesisController extends Controller
{
    public function index(AnamnesisIndex $request)
    {
        $anamnesis = Anamnesis::all();
        return response()->api($anamnesis);
    }
}
```

```
<?php
namespace App\Http\Controllers\Api;
use App\Http\Controllers\Controller;
use App\Components\FilterBag;
use App\Components\ChartDatasets\VisitsChartDataset;
use App\Components\ChartDatasets\PaymentsChartDataset;
use App\Components\ChartDatasets\ServicesCostChartDataset;
use App\Components\ChartDatasets\RatioChartDataset;
use App\Components\Filters\PeriodFilter;
use App\Components\Filters\PeriodTypeFilter;
use App\Components\Filters\UserFilter;
use App\Components\Filters\PatientFilter;
use App\Http\Requests\Charts\ShowVisits as ChartsShowVisits;
use App\Http\Requests\Charts\ShowPayments as ChartsShowPayments;
use App\Http\Requests\Charts\ShowServicesCost as ChartsShowServicesCost;
use App\Http\Requests\Charts\ShowRatio as ChartsShowRatio;
```

```
class ChartsController extends Controller
{
    private function getFilterBag(array $params): FilterBag
    {
        $filters = [];
    }
}
```

```

if (isset($params['patient_id'])) {
    $filters[] = new PatientFilter($params['patient_id'], 'patient_id');
} elseif (isset($params['user_id'])) {
    $filters[] = new UserFilter($params['user_id'], 'patient_id');
}
$filters[] = new PeriodFilter($params['start_date'], $params['end_date'], 'date');
$filters[] = new PeriodTypeFilter($params['period'], 'date');
return new FilterBag($filters);
}

```

```

public function showVisits(ChartsShowVisits $request)
{
    $filterBag = $this->getFilterBag($request->all());
    $dataset = new VisitsChartDataset($filterBag);
    return response()->api([
        'data' => $dataset->get(),
    ]);
}

```

```

public function showPayments(ChartsShowPayments $request)
{
    $filterBag = $this->getFilterBag($request->all());
    $dataset = new PaymentsChartDataset($filterBag);
    return response()->api([
        'data' => $dataset->get(),
    ]);
}

```

```

public function showServicesCost(ChartsShowServicesCost $request)
{
    $filterBag = $this->getFilterBag($request->all());
    $dataset = new ServicesCostChartDataset($filterBag);
    return response()->api([

```

```

        'data' => $dataset->get(),
    ]);
}

```

```

public function showRatio(ChartsShowRatio $request)
{
    $dataset = new RatioChartDataset($request->get('type'));
    return response()->api([
        'data' => $dataset->get(),
    ]);
}
}

```

```

<?php
namespace App\Http\Controllers\Api;
use App\Http\Controllers\Controller;
use App\Http\Requests\Events\Index as EventsIndex;
use App\Http\Requests\Events\Store as EventsStore;
use App\Http\Requests\Events\Update as EventsUpdate;
use App\Http\Requests\Events\Destroy as EventsDestroy;
use App\Models\Event;
use App\Models\Patient;
use App\Models\User;
use Illuminate\Database\Eloquent\Builder;
use Illuminate\Support\Facades\Storage;

class EventsController extends Controller
{
    public function index(EventsIndex $request)
    {
        $events = Event::query()
            ->when(!auth()->user()->hasRole('admin'), function(Builder $query) {
                $query->where('user_id', '=', auth()->user()->id);
            });
    }
}

```

```

    })
    ->get();
$events->transform(function ($item, $key) {
    if (isset($item->patient_id)) {
        $patient = Patient::findOrFail($item->patient_id);
        if (isset($patient->hash_file_name)) {
            $patient->image_path = Storage::disk('patient_avatars')->url($patient->hash_file_name);
        }
        $item->patient = $patient;
    }
    if (isset($item->user_id)) {
        $user = User::findOrFail($item->user_id);
        $item->user = $user;
    }
    return $item;
});
return response()->api($events);
}

```

```

public function store(EventsStore $request)
{
    $event = Event::create([
        'patient_id' => $request->patient_id,
        'user_id' => auth()->user()->id,
        'start' => $request->start,
        'end' => $request->end,
        'status' => $request->status,
        'details' => $request->details ?? null,
    ]);
    return response()->api($event);
}

```

```

public function update(EventsUpdate $request, Event $event)

```

```
{
    $event->update([
        'patient_id' => $request->patient_id ?? $event->patient_id,
        'start' => $request->start ?? $event->start,
        'end' => $request->end ?? $event->end,
        'status' => $request->status ?? $event->status,
        'details' => $request->details ?? $event->details,
    ]);
    return response()->api($event);
}
```

```
public function destroy(EventsDestroy $request, Event $event)
{
    $event->delete();
    return response()->api($event);
}
}
```

```
<?php
namespace App\Http\Controllers\Api;
use App\Http\Controllers\Controller;
use App\Http\Requests\Patients\Index as PatientsIndex;
use App\Http\Requests\Patients\Store as PatientsStore;
use App\Http\Requests\Patients\Show as PatientsShow;
use App\Http\Requests\Patients\Update as PatientsUpdate;
use App\Http\Requests\Patients\Destroy as PatientsDestroy;
use App\Http\Requests\Patients\ShowVisitHistory as PatientsShowVisitHistory;
use App\Http\Requests\Patients\CreateVisitHistoryRecord as PatientsCreateVisitHistoryRecord;
use App\Http\Requests\Patients\ShowPaymentHistory as PatientsShowPaymentHistory;
use App\Http\Requests\Patients\CreatePaymentHistoryRecord as PatientsCreatePaymentHistoryRecord
;
use App\Models\Anamnesis;
use App\Models\Patient;
```

```

use Illuminate\Database\Eloquent\Builder;
use Illuminate\Support\Facades\DB;
use Illuminate\Support\Facades\Storage;

class PatientsController extends Controller
{
    public function index(PatientsIndex $request)
    {
        $limit = $request->get('limit') ?? 10;
        $patients = Patient::query()
            ->when(!auth()->user()->hasRole('admin'), function(Builder $query) {
                $query->where('user_id', '=', auth()->user()->id);
            })
            ->when($request->get('q'), function(Builder $query) use (&$request) {
                $query->where('id', 'LIKE', '%'. $request->get('q') .'%')
                    ->orWhere('full_name', 'LIKE', '%'. $request->get('q') .'%')
                    ->orWhere('phone', 'LIKE', '%'. $request->get('q') .'%');
            })
            ->when($request->get('sort_by'), function(Builder $query) use (&$request) {
                $direction = $request->get('sort_desc');
                $query->orderBy($request-
>get('sort_by'), (isset($direction) && json_decode($direction)) ? 'DESC' : 'ASC');
            })
            ->orderBy('created_at', 'DESC')
            ->paginate($limit);
        $patients->getCollection()->transform(function ($item, $key) {
            if (isset($item->hash_file_name)) {
                $item->image_path = Storage::disk('patient_avatars')->url($item->hash_file_name);
            }
            return $item;
        });

        return response()->api($patients);
    }
}

```

```
}
```

```
public function store(PatientsStore $request) {
    $photo = $request->file('photo');
    $originalFileName = null;
    $hashFileName = null;
    if (isset($photo)) {
        $imagePath = $photo->store('/', 'patient_avatars');

        if ($imagePath) {
            $originalFileName = $photo->getClientOriginalName();
            $hashFileName = basename($imagePath);
        }
    }
    $patient = Patient::create([
        'full_name' => $request->full_name,
        'phone' => $request->phone,
        'email' => $request->email ?? null,
        'gender' => $request->gender ?? null,
        'birth_date' => $request->birth_date ?? null,
        'medical_info' => $request->medical_info ?? null,
        'user_id' => auth()->user()->id,
        'anamnesis_id' => $request->anamnesis_id ?? null,
        'original_file_name' => $originalFileName,
        'hash_file_name' => $hashFileName
    ]);
    return response()->api($patient);
}
```

```
public function show(PatientsShow $request, Patient $patient) {
    if (isset($patient->hash_file_name)) {
        $patient->image_path = Storage::disk('patient_avatars')->url($patient->hash_file_name);
    }
}
```

```

if (isset($patient->anamnesis_id)) {
    $patient->anamnesis = Anamnesis::find($patient->anamnesis_id);
}
return response()->api($patient);
}

public function update(PatientsUpdate $request, Patient $patient) {
    $photo = $request->file('photo');
    $originalFileName = $patient->original_file_name;
    $hashFileName = $patient->hash_file_name;

    if (isset($photo)) {
        if (isset($hashFileName)) {
            Storage::disk('patient_avatars')->delete($hashFileName);
        }
        $imagePath = $photo->store('/', 'patient_avatars');
        if ($imagePath) {
            $originalFileName = $photo->getClientOriginalName();
            $hashFileName = basename($imagePath);
        }
    }
    $patient->update([
        'full_name' => $request->full_name ?? $patient->full_name,
        'phone' => $request->phone ?? $patient->phone,
        'email' => $request->email ?? $patient->email,
        'gender' => $request->gender ?? $patient->gender,
        'birth_date' => $request->birth_date ?? $patient->birth_date,
        'medical_info' => $request->medical_info ?? $patient->medical_info,
        'anamnesis_id' => $request->anamnesis_id ?? $patient->anamnesis_id,
        'original_file_name' => $originalFileName,
        'hash_file_name' => $hashFileName
    ]);
    return response()->api($patient);
}

```

```
}
```

```
public function showVisitHistory(PatientsShowVisitHistory $request, Patient $patient) {
    $limit = $request->get('limit') ?? 10;
    $sortBy = $request->get('sort_by');
    $query = $patient->visits()
        ->join('service_visit', 'visits.id', '=', 'service_visit.visit_id')
        ->groupBy('visits.id', 'visits.date')
        ->select(DB::raw('visits.id, visits.date, SUM(service_visit.total_cost) as service_cost'));
    if (isset($sortBy)) {
        $direction = $request->get('sort_desc');
        $query = $query->orderBy($request-
>get('sort_by'), (isset($direction) && json_decode($direction)) ? 'DESC' : 'ASC');
    }
    $history = $query->orderBy('visits.created_at', 'DESC')
        ->paginate($limit);
    return response()->api($history);
}
```

```
public function createVisitHistoryRecord(PatientsCreateVisitHistoryRecord $request, Patient $patient)
{
    $visit = $patient->visits()->create([
        'date' => $request->date
    ]);
    $serviceCost = 0;
    foreach ($request->services as $service) {
        $visit->services()->attach($service['id'], [
            'service_count' => $service['service_count'],
            'total_cost' => $service['total_cost']
        ]);
        $serviceCost += $service['total_cost'];
    }
    $patient->update([
```

```

        'total_accrued' => $patient->total_accrued + $serviceCost
    });
    return response()->api($visit);
}

```

```

public function showPaymentHistory(PatientsShowPaymentHistory $request, Patient $patient) {
    $limit = $request->get('limit') ?? 10;
    $payments = $patient->payments()
        ->when($request->get('sort_by'), function(Builder $query) use (&$request) {
            $direction = $request->get('sort_desc');
            $query->orderBy($request-
>get('sort_by'), (isset($direction) && json_decode($direction)) ? 'DESC' : 'ASC');
        })
        ->orderBy('created_at', 'DESC')
        ->paginate($limit);
    return response()->api($payments);
}

```

```

public function createPaymentHistoryRecord(PatientsCreatePaymentHistoryRecord $request, Patient
$patient) {
    $payment = $patient->payments()->create([
        'amount' => $request->amount,
        'date' => $request->date,
        'notes' => $request->notes ?? null,
    ]);
    $patient->update([
        'total_paid' => $patient->total_paid + $request->amount
    ]);
    return response()->api($payment);
}

```

```

public function destroy(PatientsDestroy $request, Patient $patient)
{

```

```

    $patient->delete();
    return response()->api($patient);
}
}

```

```
<?php
```

```

namespace App\Http\Controllers\Api;
use App\Http\Controllers\Controller;
use App\Http\Requests\Services\Index as ServicesIndex;
use App\Http\Requests\Services\Store as ServicesStore;
use App\Http\Requests\Services\Update as ServicesUpdate;
use App\Http\Requests\Services\Destroy as ServicesDestroy;
use App\Models\Service;
use Illuminate\Database\Eloquent\Builder;

```

```
class ServicesController extends Controller
```

```

{
    public function index(ServicesIndex $request)
    {
        $limit = $request->get('limit') ?? 10;
        $services = Service::query()
            ->when($request->get('q'), function(Builder $query) use (&$request) {
                $query->where('name', 'LIKE', '%'. $request->get('q') .'%');
            })
            ->when($request->get('sort_by'), function(Builder $query) use (&$request) {
                $direction = $request->get('sort_desc');
                $query->orderBy($request-
>get('sort_by'), (isset($direction) && json_decode($direction)) ? 'DESC' : 'ASC');
            })
            ->orderBy('created_at', 'DESC')
            ->paginate($limit);
        return response()->api($services);
    }
}

```

```

public function store(ServicesStore $request)
{
    $service = Service::create([
        'name' => $request->name,
        'price' => $request->price,
    ]);
    return response()->api($service);
}

```

```

public function update(ServicesUpdate $request, Service $service)
{
    $service->update([
        'name' => $request->name ?? $service->name,
        'price' => $request->price ?? $service->price,
    ]);
    return response()->api($service);
}

```

```

public function destroy(ServicesDestroy $request, Service $service)
{
    $service->delete();
    return response()->api($service);
}
}

```

```

<?php
namespace App\Http\Controllers\Api;
use App\Http\Controllers\Controller;
use App\Http\Requests\Users\Index as UsersIndex;
use App\Http\Requests\Users\Store as UsersStore;
use App\Http\Requests\Users\Show as UsersShow;
use App\Http\Requests\Users\Update as UsersUpdate;

```

```

use App\Http\Requests\Users\Destroy as UsersDestroy;
use App\Models\User;
use Illuminate\Database\Eloquent\Builder;
use Illuminate\Auth\Events\Registered;
use Illuminate\Support\Facades\Storage;
use Illuminate\Support\Facades\Hash;
use Illuminate\Support\Str;

class UsersController extends Controller
{
    public function index(UsersIndex $request)
    {
        $limit = $request->get('limit') ?? 10;
        $users = User::query()
            ->when($request->get('q'), function(Builder $query) use (&$request) {
                $query->where('full_name', 'LIKE', '%'. $request->get('q') .'%');
            })
            ->when($request->get('sort_by'), function(Builder $query) use (&$request) {
                $direction = $request->get('sort_desc');
                $query->orderBy($request->
>get('sort_by'), (isset($direction) && json_decode($direction)) ? 'DESC' : 'ASC');
            })
            ->where('id', '!=', auth()->user()->id)
            ->orderBy('created_at', 'DESC')
            ->paginate($limit);
        $users->getCollection()->transform(function ($item, $key) {
            if (isset($item->hash_file_name)) {
                $item->image_path = Storage::disk('user_avatars')->url($item->hash_file_name);
            }
            return $item;
        });
        return response()->api($users);
    }
}

```

```

public function store(UsersStore $request) {
    $photo = $request->file('photo');
    $originalFileName = null;
    $hashFileName = null;
    if (isset($photo)) {
        $imagePath = $photo->store('/', 'user_avatars');
        if ($imagePath) {
            $originalFileName = $photo->getClientOriginalName();
            $hashFileName = basename($imagePath);
        }
    }
    $user = User::create([
        'full_name' => $request->full_name,
        'phone' => $request->phone,
        'email' => $request->email,
        'password' => Hash::make($request->password),
        'api_token' => uniqid(Str::random(60)),
        'original_file_name' => $originalFileName,
        'hash_file_name' => $hashFileName
    ]);
    $user->assignRole('user');
    event(new Registered($user));
    return response()->api($user);
}

public function show(UsersShow $request, User $user) {
    if (isset($user->hash_file_name)) {
        $user->image_path = Storage::disk('user_avatars')->url($user->hash_file_name);
    }
    return response()->api($user);
}

```

```

public function update(UsersUpdate $request, User $user) {
    $photo = $request->file('photo');
    $originalFileName = $user->original_file_name;
    $hashFileName = $user->hash_file_name;
    if (isset($photo)) {
        if (isset($hashFileName)) {
            Storage::disk('user_avatars')->delete($hashFileName);
        }
        $imagePath = $photo->store('/', 'user_avatars');
        if ($imagePath) {
            $originalFileName = $photo->getClientOriginalName();
            $hashFileName = basename($imagePath);
        }
    }
    $user->update([
        'full_name' => $request->full_name ?? $user->full_name,
        'phone' => $request->phone ?? $user->phone,
        'email' => $request->email ?? $user->email,
        'password' => $request->new_password ? Hash::make($request->new_password) : $user->password,
        'original_file_name' => $originalFileName,
        'hash_file_name' => $hashFileName
    ]);
    return response()->api($user);
}

public function destroy(UsersDestroy $request, User $user)
{
    $user->delete();
    return response()->api($user);
}
}

```

```
<?php
namespace App\Http\Controllers\Api;
use App\Http\Controllers\Controller;
use App\Http\Requests\Visits\ShowServices as VisitsShowServices;
use App\Models\Visit;

class VisitsController extends Controller
{
    public function showServices(VisitsShowServices $request, Visit $visit) {
        $services = $visit->services()
            ->get();
        return response()->api($services);
    }
}
```

ДОДАТОК Б

Код клієнтської частини

```

<template>
  <div>
    <h1 class="form-title">Login</h1>
    <v-card class="form-card elevation-2">
      <v-card-text class="card-content">
        <v-form>
          <v-text-field
            label="Email"
            name="email"
            prepend-icon="email"
            type="email"
            v-model="email"
            maxlength="255"
            :error-messages="emailErrors"
            @blur="$v.email.$touch()"
            @keyup.native.enter="submit"
          ></v-text-field>
          <v-text-field
            label="Password"
            name="password"
            prepend-icon="lock"
            :append-icon="showPassword ? 'mdi-eye' : 'mdi-eye-off'"
            @click:append="showPassword = !showPassword"
            :type="showPassword ? 'text' : 'password'"
            v-model="password"
            maxlength="20"
            :error-messages="passwordErrors"
            @blur="$v.password.$touch()"
  
```

```

    @keyup.native.enter="submit"
  ></v-text-field>

```

```

<div class="remember">
  <v-checkbox
    label="Remember me"
    color="primary"
    hide-details
    v-model="remember"
    class="mt-2"
  ></v-checkbox>
  <router-link :to="{ name: 'auth.forgot-password' }">Forgot Password?</router-link>
</div>
</v-form>

```

```

</v-card-text>

```

```

<v-card-actions class="card-actions">

```

```

  <v-btn color="primary"
    class="form-submit"
    :loading="isLoading"
    @click.prevent="submit">Login</v-btn>

```

```

  <p>Are you new? <router-link :to="{ name: 'auth.register' }">Sign up</router-link></p>

```

```

</v-card-actions>

```

```

</v-card>

```

```

</div>

```

```

</template>

```

```

<script>

```

```

  import { required, email, minLength } from 'vuelidate/lib/validators';

```

```

  export default {

```

```

    name: "Login",

```

```

    data() {

```

```

      return {

```

```

        email: "",

```

```

    password: "",
    remember: false,
    showPassword: false,
    isLoading: false,
  }
},
validations: {
  email: {
    required,
    email
  },
  password: {
    required,
    minLength: minLength(8)
  }
},
computed: {
  emailErrors() {
    const errors = this.$store.state.auth.validationErrors.email || [];
    if (!this.$v.email.$dirty) return errors;
    !this.$v.email.required && errors.push('The e-mail is required. ');
    !this.$v.email.email && errors.push('The email must be a valid email address. ');
    return errors;
  },
  passwordErrors() {
    const errors = this.$store.state.auth.validationErrors.password || [];
    if (!this.$v.password.$dirty) return errors;
    !this.$v.password.required && errors.push('The password is required. ');
    !this.$v.password.minLength && errors.push('Password must be at least 8 characters long');
    return errors;
  },
},
methods: {

```

```

submit() {
  this.$v.$touch();
  if (!this.$v.$invalid) {
    const {email, password, remember} = this;

    this.isLoading = true;
    this.$store.dispatch('auth/login', {data: {email, password}, remember})
      .then(() => {
        this.$store.dispatch('auth/clearValidationErrors');
        if (this.$route.params.nextUrl != null) {
          this.$router.push(this.$route.params.nextUrl);
        } else {
          this.$router.push({ name: 'dashboard.patients' });
        }
      })
      .finally(() => {
        this.isLoading = false;
      });
  }
},
},
}
</script>

```

```

<style scoped>
.form-title {
  color: #fff;
  text-transform: uppercase;
  margin-bottom: 5px;
}
.form-card {
  padding: 30px 40px;
}

```

```

.card-content {
  padding: 0;
}
.card-actions {
  padding: 0;
  flex-wrap: wrap;
  justify-content: center;
}
.form-submit {
  width: 100%;
  margin: 20px 0 25px;
}
.remember {
  display: flex;
  justify-content: space-between;
  align-items: flex-end;
  font-size: 1rem;
}
</style>

<template>
  <v-container fluid>
    <v-row class="ma-0" justify="center">
      <v-col cols="12" class="py-0">
        <v-card>
          <v-toolbar flat color="white">
            <v-text-field
              flat
              solo
              prepend-icon="search"
              placeholder="Type something"
              v-model="search"
              hide-details

```

```

        class="hidden-sm-and-down mr-4"
    ></v-text-field>

    <v-btn color="primary" link :to="{ name: 'dashboard.create-patient' }">
        <v-icon left>mdi-plus</v-icon>
        Add new
    </v-btn>
</v-toolbar>
<v-divider></v-divider>
<v-card-text class="pa-0">
    <v-data-table
        :headers="headers"
        :items="patients"
        :server-items-length="totalPatients"
        :options.sync="options"
        :loading="isLoading"
        :items-per-page="15"
        :footer-props="footerProps"
        class="elevation-1"
    >
        <template v-slot:item.avatar="{ item }">
            <v-avatar size="32">
                
            </v-avatar>
        </template>
        <template v-slot:item.action="{ item }">
            <v-menu left offset-y origin="center center" :nudge-bottom="10" transition="scale-
transition">
                <template v-slot:activator="{ on }">
                    <v-btn text icon v-on="on">
                        <v-icon>mdi-dots-vertical</v-icon>
                    </v-btn>
                </template>
                <v-list class="pa-0">

```

```

<template v-for="(menuItem, index) in patientActions">
  <v-divider
    v-if="menuItem.divider"
    :key="index"
  ></v-divider>
  <v-list-item
    v-else
    @click="menuItem.click ? menuItem.click(item.id) : null"
    :key="index"
    ripple="ripple"
  >
    <v-list-item-icon v-if="menuItem.icon" class="mr-4">
      <v-icon>{{ menuItem.icon }}</v-icon>
    </v-list-item-icon>
    <v-list-item-content>
      <v-list-item-title>{{ menuItem.title }}</v-list-item-title>
    </v-list-item-content>
    </v-list-item>
  </template>
</v-list>
</v-menu>
</template>
</v-data-table>
</v-card-text>
</v-card>
</v-col>
</v-row>
</v-container>
</template>

<script>
  import moment from 'moment';
  const MALE = 0;

```

```
const FEMALE = 1;

export default {
  name: "Patients",
  data() {
    return {
      search: null,
      isLoading: false,
      options: {},
      footerProps: {
        'items-per-page-options': [15, 30, 45]
      },
      headers: [
        {
          text: 'ID',
          align: 'left',
          sortable: false,
          value: 'id',
        },
        {
          text: 'Avatar',
          align: 'left',
          sortable: false,
          value: 'avatar',
        },
        {
          text: 'Full name',
          align: 'left',
          sortable: true,
          value: 'full_name',
        },
        {
          text: 'Phone',
          align: 'left',
```

```

    sortable: false,
    value: 'phone',
  },
  {
    text: 'Email',
    align: 'left',
    sortable: false,
    value: 'email',
  },
  {
    text: 'Gender',
    align: 'left',
    sortable: false,
    value: 'gender',
  },
  {
    text: 'Birth date',
    align: 'left',
    sortable: true,
    value: 'birth_date',
  },
  {
    text: 'Actions',
    align: 'left',
    sortable: false,
    value: 'action',
  },
],
patientActions: [
  {
    icon: "mdi-eye",
    click: (id) => {
      this.$router.push({name: 'dashboard.patient-profile', params: { id }});
    }
  }
]

```

```

    },
    title: "Show patient profile",
  },
  { divider: true },
  {
    icon: "mdi-account-details",
    click: (id) => {
      this.$router.push({name: 'dashboard.visit-history', params: { id }});
    },
    title: "Show visit history",
  },
  { divider: true },
  {
    icon: "mdi-account-cash",
    click: (id) => {
      this.$router.push({name: 'dashboard.payment-history', params: { id }});
    },
    title: "Show payment history",
  },
  { divider: true },
  {
    icon: "mdi-delete",
    click: (id) => {
      this.deletePatient(id);
    },
    title: "Delete patient",
  },
]
}
},
computed: {
  patients() {
    if (this.$store.state.patients.patients && this.$store.state.patients.patients.length) {

```

```

return this.$store.state.patients.patients.map(patient => {
  let gender = null;
  if (patient.gender || patient.gender === 0) {
    gender = (+patient.gender === MALE) ? 'male' : (+patient.gender === FEMALE) ? 'female'
: null;
  }
  return {
    'id': patient.id,
    'full_name': patient.full_name,
    'phone': patient.phone,
    'email': patient.email || null,
    'gender': gender,
    'birth_date': patient.birth_date ? moment(patient.birth_date).format('DD-MM-
YYYY') : null,
    'image_path': patient.image_path || '/storage/images/no-profile-image.png',
  };
});
}
return [];
},
totalPatients() {
  return this.$store.state.patients.pagination.total || 0;
}
},
watch: {
  options: {
    handler() {
      this.loadPatients();
    },
    deep: true,
  },
},
search() {
  this.options.page = 1;

```

```

        this.loadPatients();
    }
},
methods: {
    loadPatients() {
        let params = {
            q: this.search,
            page: this.options.page || 1,
            limit: this.options.itemsPerPage || 15,
            sort_by: this.options.sortBy || [],
            sort_desc: this.options.sortDesc || [],
        };
        this.isLoading = true;
        this.$store.dispatch('patients/loadPatients', { params })
            .finally(() => {
                this.isLoading = false;
            });
    },
    deletePatient(id) {
        if (confirm('Are you sure you want to delete this patient?')) {
            this.$store.dispatch('patients/deletePatient', id)
                .then(() => {
                    this.loadPatients();
                });
        }
    },
}
}
</script>

```

```
<template>
```

```
  <v-container fluid>
```

```
    <v-row class="ma-0" justify="center">
```

```

<v-col cols="12" class="py-0">
  <v-btn fab dark color="primary" class="add-event-button" @click="dialog = true">
    <v-icon>mdi-plus</v-icon>
  </v-btn>
  <event-form
    :dialog="dialog"
    :event="editedEvent"
    :selected-id="selectedId"
    @on-create="eventCreated"
    @on-update="eventUpdated"
    @on-close="closeForm"
  ></event-form>

  <v-card>
    <v-toolbar flat color="white" height="64">
      <v-btn outlined class="mr-4" @click="setToday">
        Today
      </v-btn>
      <v-btn fab text small class="mr-1" @click="prev">
        <v-icon small>mdi-chevron-left</v-icon>
      </v-btn>
      <v-btn fab text small class="mr-4" @click="next">
        <v-icon small>mdi-chevron-right</v-icon>
      </v-btn>

      <v-toolbar-title>{{ title }}</v-toolbar-title>
      <div class="flex-grow-1"></div>
      <v-menu offset-y origin="center center" :nudge-bottom="10" transition="scale-
transition">
        <template v-slot:activator="{ on }">
          <v-btn outlined v-on="on">
            <span>{{ typeToLabel[type] }}</span>
            <v-icon right>mdi-menu-down</v-icon>

```

```

        </v-btn>
    </template>
    <v-list>
        <v-list-item @click="type = 'day'">
            <v-list-item-title>Day</v-list-item-title>
        </v-list-item>
        <v-list-item @click="type = 'week'">
            <v-list-item-title>Week</v-list-item-title>
        </v-list-item>
        <v-list-item @click="type = 'month'">
            <v-list-item-title>Month</v-list-item-title>
        </v-list-item>
    </v-list>
</v-menu>
</v-toolbar>
<v-divider></v-divider>
<v-progress-linear indeterminate color="primary" height="2" v-if="isLoading"></v-progress-
linear>
<v-card-text>
    <v-calendar
        class="calendar"
        ref="calendar"
        v-model="focus"
        color="primary"
        :events="events"
        :event-name="getEventName"
        :event-color="(event) => getStatusColor(event.status)"
        :event-margin-bottom="3"
        :now="today"
        :type="type"
        :weekdays="[1,2,3,4,5,6,0]"
        @click:event="showEvent"
        @click:more="viewDay"

```

```

        @click:date="viewDay"
        @change="updateRange"
    ></v-calendar>
    <v-menu
        v-model="selectedOpen"
        :close-on-content-click="false"
        :activator="selectedElement"
        offset-x
    >
        <v-card max-width="400px" outlined class="event-card">
            <v-card-content>
                <v-list-item three-line>
                    <v-list-item-content class="pt-4 pb-6">
                        <v-chip class="event-card__status mb-
4" small :color="getStatusColor(selectedEvent.status)" text-color="white">{{ selectedEvent.status }}</v-
chip>
                        <v-list-item-title class="headline mb-
1">{{ selectedEvent.patient && selectedEvent.patient.full_name ? selectedEvent.patient.full_name : 'Un
known patient'}}</v-list-item-title>
                        <v-list-item-subtitle class="event-card__period"><v-icon small>mdi-clock-
outline</v-icon>{{ getPeriod(selectedEvent.start, selectedEvent.end) }}</v-list-item-subtitle>
                    </v-list-item-content>
                    <v-list-item-avatar tile size="80" color="grey">
                        <v-
img :src="selectedEvent.patient && selectedEvent.patient.image_path ? selectedEvent.patient.image_p
ath : '/storage/images/no-profile-image.png'"></v-img>
                    </v-list-item-avatar>
                </v-list-item>
                <v-list-item class="event-card__info" two-line v-
if="selectedEvent.user && selectedEvent.user.full_name">
                    <v-list-item-content class="pa-0">
                        <v-list-item-title>Attending doctor</v-list-item-title>
                        <v-list-item-subtitle>{{ selectedEvent.user.full_name }}</v-list-item-subtitle>

```

```

        </v-list-item-content>
    </v-list-item>
    <v-list-item class="event-card__info" two-line v-if="selectedEvent.details">
        <v-list-item-content class="pa-0">
            <v-list-item-title>Details</v-list-item-title>
            <v-list-item-subtitle>{{ selectedEvent.details }}</v-list-item-subtitle>
        </v-list-item-content>
    </v-list-item>
</v-card-content>
<v-divider></v-divider>
<v-card-actions>
    <v-btn text color="primary" @click.prevent="editEvent(selectedEvent)">Edit</v-
btn>

    <v-spacer></v-spacer>
    <v-btn
text color="primary" @click.prevent="deleteEvent(selectedEvent)">Delete</v-btn>
</v-card-actions>
</v-card>
</v-menu>
</v-card-text>
</v-card>
</v-col>
</v-row>
{{selectedEvent}}
</v-container>
</template>

<script>
import moment from 'moment';
import EventForm from "../EventForm";
export default {
    name: "Calendar",
    components: {

```

```

    EventForm,
  },
  data () {
    return {
      isLoading: false,
      today: moment().format('YYYY-MM-DD'),
      focus: moment().format('YYYY-MM-DD'),
      type: 'month',
      typeToLabel: {
        month: 'Month',
        week: 'Week',
        day: 'Day',
      },
      start: null,
      end: null,
      selectedId: null,
      selectedEvent: {},
      editedEvent: {},
      selectedElement: null,
      selectedOpen: false,
      dialog: false,
    }
  },
  created () {
    this.loadEvents();
  },
  computed: {
    events () {
      if (this.$store.state.events.events && this.$store.state.events.events.length) {
        return this.$store.state.events.events.map(event => {
          return {
            'id': event.id,
            'name': (event.patient && event.patient.full_name) ? event.patient.full_name : null,

```

```

        'start': event.start,
        'end': event.end,
        'status': event.status,
        'details': event.details || null,
        'patient': event.patient || null,
        'user': event.user || null
    };
});
}

```

```

return [];

```

```

},

```

```

title () {

```

```

    const { start, end } = this;

```

```

    if (!start || !end) {

```

```

        return "";

```

```

    }

```

```

const startMonth = moment(start.date).format('MMMM');

```

```

const endMonth = moment(end.date).format('MMMM');

```

```

const suffixMonth = startMonth === endMonth ? "" : endMonth;

```

```

const startYear = start.year;

```

```

const endYear = end.year;

```

```

const suffixYear = startYear === endYear ? "" : endYear;

```

```

const startDay = moment(start.date).format('Do');

```

```

const endDay = moment(end.date).format('Do');

```

```

switch (this.type) {

```

```

    case 'month':

```

```

        return `${startMonth} ${startYear}`;

```

```

    case 'week':

```

```

        return suffixYear ? `${startMonth} ${startDay} ${startYear} -`

```

```

        `${suffixMonth} ${endDay} ${suffixYear}` : `${startMonth} ${startDay} -`

```

```

        `${suffixMonth} ${endDay} ${startYear}`;

```

```

    case 'day':

```

```

        return `${startMonth} ${startDay} ${startYear}`;
    }
    return "";
}
},
methods: {
    loadEvents () {
        this.isLoading = true;
        this.$store.dispatch('events/loadEvents')
            .finally(() => {
                this.isLoading = false;
            });
    },
    viewDay ({ date }) {
        this.focus = date;
        this.type = 'day';
    },
    setToday () {
        this.focus = this.today;
    },
    prev () {
        this.$refs.calendar.prev();
    },
    next () {
        this.$refs.calendar.next();
    },
    updateRange ({ start, end }) {
        this.start = start;
        this.end = end;
    },
    getEventName (event) {
        return event.start.time + ' - ' + event.end.time + ' ' + event.input.name;
    },

```

```

getStatusColor (status) {
  switch (status) {
    case "Запланировано":
      return "#2196F3";
    case "Подтверждено":
      return "#673AB7";
    case "Выполнено":
      return "#8BC34A";
    case "Пропущено":
      return "#FF5252";
    case "Перенесено":
      return "#FFC107";
  }
  return "#2196F3";
},
getPeriod (start, end) {
  return moment(start).format('D MMMM, dddd ') + moment(start).format('HH:mm') + ' - ' + moment(end).format('HH:mm');
},
editEvent (event) {
  this.$store.dispatch('events/setPatients', [event.patient]);
  this.selectedId = event.id;
  this.editedEvent = event;
  this.dialog = true;
},
deleteEvent (event) {
  if (confirm('Are you sure you want to delete this event?')) {
    this.$store.dispatch('events/deleteEvent', event.id)
      .then(() => {
        this.selectedOpen = false;
        this.loadEvents();
      });
  }
}

```

```

},
showEvent ({ nativeEvent, event }) {
  const open = () => {
    this.selectedEvent = event;
    this.selectedElement = nativeEvent.target;
    setTimeout(() => this.selectedOpen = true, 10)
  };
  if (this.selectedOpen) {
    this.selectedOpen = false;
    setTimeout(open, 10);
  } else {
    open();
  }
  nativeEvent.stopPropagation();
},
eventCreated () {
  this.loadEvents();
  this.closeForm();
},
eventUpdated (data) {
  this.selectedEvent = Object.assign({}, this.selectedEvent, data);
  this.loadEvents();
  this.closeForm();
},
closeForm () {
  this.dialog = false;
  setTimeout(() => {
    this.selectedId = null;
    this.editedEvent = {};
  }, 300);
},
},
}

```

```
</script>
```

```
<style scoped lang="scss">
```

```
.add-event-button {
  position: fixed;
  bottom: 45px;
  right: 25px;
  z-index: 99;
}
```

```
.calendar {
  min-height: 650px;
}
```

```
.event-card__status {
  flex: initial;
}
```

```
.event-card__period .v-icon {
  margin-right: 5px;
  margin-top: -2px;
}
```

```
.event-card__info {
  min-height: 0;
  padding-top: 0;
  padding-bottom: 20px;
  &:last-child {
    padding-bottom: 24px;
  }
}
```

```
</style>
```

```
<template>
```

```
<v-container fluid>
```

```
<v-row class="ma-0" justify="center">
```

```
<v-col cols="8" class="py-0">
```

```

<v-card>
  <v-toolbar flat color="white" height="64">
    <chart-filters @filters-changed="(filters) => loadChartData(filters)"/>
  </v-toolbar>
  <v-divider></v-divider>
  <v-card-text>
    <line-chart :chart-data="statisticData" :options="statisticOptions" v-if="!statisticLoading"/>
    <div class="chart-progress" v-else>
      <v-progress-circular
        :size="50"
        color="primary"
        indeterminate
      ></v-progress-circular>
    </div>
  </v-card-text>
</v-card>
</v-col>
<v-col cols="4" class="py-0">
  <v-card>
    <v-toolbar flat color="white" height="64">
      <h3 class="dashboard-card__title">Patients ratio</h3>
      <v-spacer></v-spacer>
      <v-select
        class="ratio-type"
        dense
        outlined
        v-model="ratioType"
        :items="ratioTypeOptions"
        hide-details
        item-text="name"
        item-value="value"
        placeholder="Ratio type"
        :menu-props="{ offsetY: true, nudgeBottom: 10 }"

```

```

    >
    <template v-slot:item="{ item }">
      <span class="subtitle-1">{{ item.name }}</span>
    </template>
  </v-select>
</v-toolbar>
<v-divider></v-divider>
<v-card-text>
  <doughnut-chart :chart-data="ratioData" :options="ratioOptions" v-if="!ratioLoading"/>
  <div class="chart-progress" v-else>
    <v-progress-circular
      :size="50"
      color="primary"
      indeterminate
    ></v-progress-circular>
  </div>
</v-card-text>
</v-card>
</v-col>
</v-row>
</v-container>
</template>

<script>
import ChartFilters from './ChartFilters.vue';
import LineChart from './Charts/LineChart.vue';
import DoughnutChart from './Charts/DoughnutChart.vue';
const colors = {
  "Visits": '#dc62f8',
  "Payments": '#aef849',
  "Services Cost": '#f8a421',
};
export default {

```

```
components: {  
  ChartFilters,  
  LineChart,  
  DoughnutChart  
},  
data() {  
  return {  
    statisticLoading: false,  
    ratioLoading: false,  
    statisticData: {  
      labels: [],  
      datasets: [],  
    },  
    ratioData: {  
      labels: [],  
      datasets: [],  
    },  
    statisticOptions: {  
      responsive: true,  
      maintainAspectRatio: false,  
      animation: {  
        duration: 0  
      },  
      hover: {  
        animationDuration: 0  
      },  
      responsiveAnimationDuration: 0  
    },  
    ratioOptions: {  
      responsive: true,  
      maintainAspectRatio: false,  
    },  
    ratioType: null,  
  }  
}
```

```

ratioTypeOptions: [
  {
    name: 'Age',
    value: 0
  },
  {
    name: 'Gender',
    value: 1
  }
]
},
created() {
  this.ratioType = 0;
},
watch: {
  ratioType(value) {
    this.loadRatio(value);
  }
},
methods: {
  loadChartData(filters) {
    this.statisticLoading = true;
    Promise.resolve(this.getVisits(filters))
      .then(() => {
        return this.getPayments(filters);
      })
      .then(() => {
        return this.getServicesCost(filters);
      })
      .finally(() => {
        this.statisticLoading = false;
      });
  }
}

```

```

},
getVisits(filters) {
  return new Promise((resolve, reject) => {
    this.$store.dispatch("statistics/loadChartData", {
      chartType: 'visits', params: filters
    }).then(() => {
      let chartData = this.$store.state.statistics.chartsData['visits'];
      if (chartData) {
        this.setChartData(chartData, 'Visits');
      }
      resolve();
    });
  });
},
getPayments(filters) {
  return new Promise((resolve, reject) => {
    this.$store.dispatch("statistics/loadChartData", {
      chartType: 'payments', params: filters
    }).then(() => {
      let chartData = this.$store.state.statistics.chartsData['payments'];
      if (chartData) {
        this.setChartData(chartData, 'Payments');
      }
      resolve();
    });
  });
},
getServicesCost(filters) {
  return new Promise((resolve, reject) => {
    this.$store.dispatch("statistics/loadChartData", {
      chartType: 'services-cost', params: filters
    }).then(() => {
      let chartData = this.$store.state.statistics.chartsData['services-cost'];

```

```

    if (chartData) {
      this.setChartData(chartData, 'Services Cost');
    }
    resolve();
  });
});
},
setChartData(data, label) {
  let labelName = label || 'Unknown';
  let index = this.statisticData.datasets.findIndex(set => set.label === labelName);
  if (index >= 0) {
    let set = this.statisticData.datasets[index];
    this.statisticData.datasets = this.statisticData.datasets.slice(0, index).concat([Object.assign({},
set, {
      data: data ? Object.values(data) : [],
    }, ...this.statisticData.datasets.slice(index + 1))]);
  } else {
    let color = colors[labelName] || '#877D87';
    let dataq = {
      label: labelName,
      backgroundColor: color,
      borderColor: color,
      data: data ? Object.values(data) : [],
      fill: false,
      spanGaps: true,
    };
    this.statisticData.datasets = [...this.statisticData.datasets, dataq]
  }
  this.statisticData.labels = Object.keys(data);
},
loadRatio(type) {
  this.ratioLoading = true;
  this.$store.dispatch("statistics/loadRatio", {

```

```

      params: { type }
    }).then(() => {
      let chartData = this.$store.state.statistics.ratio;
      if (chartData) {
        this.setRatio(chartData);
      }
    })
    .finally(() => {
      this.ratioLoading = false;
    });
  },
  setRatio(data) {
    const colors = ['#0aadd0', '#e43e6e', '#01c498', '#ffbe07', '#4e40de', '#7f2084'];
    let labels = [];
    let datasets = [
      {
        backgroundColor: [],
        data: []
      }
    ];
    Object.keys(data).forEach((key, index) => {
      labels.push(key);
      datasets[0].backgroundColor.push(colors[index]);
      datasets[0].data.push(data[key]);
    });

    this.ratioData = Object.assign({}, this.ratioData, {
      labels,
      datasets
    });
  }
}

```

```
</script>
```

```
<style scoped>
```

```
  .chart-progress {
```

```
    display: flex;
```

```
    justify-content: center;
```

```
    align-items: center;
```

```
    min-height: 400px;
```

```
  }
```

```
  .ratio-type {
```

```
    max-width: 235px;
```

```
  }
```

```
</style>
```