

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДЕРЖАВНИЙ УНІВЕРСИТЕТ «ЖИТОМИРСЬКА ПОЛІТЕХНІКА»
ФАКУЛЬТЕТ ІНФОРМАЦІЙНО-КОМП'ЮТЕРНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

ПОЯСНЮВАЛЬНА ЗАПИСКА

до кваліфікаційної роботи освітнього ступеня «бакалавр»
за спеціальністю 121 «Інженерія програмного забезпечення»
(освітня програма «Інженерія програмного забезпечення»)

на тему:

**«Вебзастосунок для моніторингу психологічного
здоров'я людини»**

Виконав студент групи ІПЗ-21-1
СВІРГУН Анастасія Володимирівна

Керівник роботи:
САВЦЬКИЙ Роман Святославович

Рецензент:
ЛОКТИКОВА Тамара Миколаївна

Житомир – 2025

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДЕРЖАВНИЙ УНІВЕРСИТЕТ «ЖИТОМИРСЬКА ПОЛІТЕХНІКА»
ФАКУЛЬТЕТ ІНФОРМАЦІЙНО-КОМП'ЮТЕРНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

«ЗАТВЕРДЖУЮ»

Зав. кафедри інженерії програмного
забезпечення

Тетяна Вакалюк

«23» лютого 2025 р.

ЗАВДАННЯ
на кваліфікаційну роботу

Здобувач вищої освіти: **СВІРГУН Анастасія Володимирівна**

Керівник роботи: **САВІЦЬКИЙ Роман Святославович**

Тема роботи: **«Вебзастосунок для моніторингу психологічного здоров'я людини»**,
затверджена Наказом закладу вищої освіти від **«14» лютого 2025 р., №61/с**

Вихідні дані для роботи: Об'єктом дослідження є система для моніторингу психологічного здоров'я, яка надає всі шляхи для пошуку лікаря та керування протікання терапії.

Предметом дослідження є спрощення процесу адміністрування амбулаторій для людей з психологічними проблемами та автоматизація процесу лікування. Дослідження направлене на вивчення технологій та методів, задля забезпечення автоматизованого управління процесами терапії та адміністрування закладами лікування.

Консультанти з бакалаврської кваліфікаційної роботи із зазначенням розділів, що їх стосуються:

Розділ	Консультант	Завдання видав	Завдання прийняв
1	Савіцький Р.С.	14 лютого 2025 р.	14 лютого 2025 р.
2	Савіцький Р.С.	14 лютого 2025 р.	14 лютого 2025 р.
3	Савіцький Р.С.	14 лютого 2025 р.	14 лютого 2025 р.

РЕФЕРАТ

Кваліфікаційна робота освітнього ступеня «бакалавр» складається з вебзастосунку для моніторингу психологічного здоров'я людини та пояснювальної записки. Пояснювальна записка до кваліфікаційної роботи містить 56 сторінок, 22 ілюстрацій та 24 таблиць. Загальний обсяг 96 сторінок.

Метою роботи є розробка вебзастосунку для моніторингу психологічного здоров'я людини. Система включає модулі для управління пацієнтами, лікарями, адміністраторами з використанням інтеграційних модулів.

У роботі поставлені основні завдання на розробку системи. Також було проведено порівняльний аналіз аналогічних систем. Було обрано клієнт-серверну архітектуру, яка працює в зв'язці з RESTful інтерфейсом. Приведені основні сценарії використання системи, логіка взаємодії між модулями системи, діаграма класів та алгоритми роботи ключових модулів, спроектована база даних з використанням PostgreSQL. Клієнтська частина розроблена за допомогою фреймворку React, а серверна частина за допомогою Spring Boot.

КЛЮЧОВІ СЛОВА: ПАЦІЄНТ, ЛІКАР, ПСИХОЛОГІЯ, МЕНТАЛЬНЕ ЗДОРОВ'Я, ТЕРАПІЯ, ВІДСЛІДКОВУВАННЯ СТАТИСТИКИ ЗДОРОВ'Я.

					ІПЗ.КР.Б – 121 – 25 – ПЗ			
Змн.	Арк.	№ докум.	Підпис	Дата	Веб застосунок для моніторингу психологічного здоров'я людини	Літ.	Арк.	Аркуші
Розроб.		Свіргун А.В.						
Керівник		Савіцький Р.В.					3	96
Н. контр.		Локтікова Т.М.				Житомирська політехніка, група ІПЗ-21-1		
Зав. каф.		Вакалюк Т.А.						

ABSTRACT

The bachelor's thesis consists of a website for tracking people's mental health and an explanatory note. The explanatory note to the qualification work contains 56 pages, 22 illustrations and 24 tables. The total volume is 96 pages.

The goal of the work is to implement a website for tracking people's mental health. The system contains modules for managing patients, doctors, admins while using integration services.

The paper sets out the main tasks for the development of the system. A comparative analysis of similar systems was also conducted. A client-server architecture was chosen that works in conjunction with RESTful interface. The main scenarios of using the system, the logic of interaction between the system modules, the diagram and algorithms of the key modules are presented, and the database is designed using PostgreSQL. The client side was developed using React, and the server side was developed with Spring Boot.

KEY WORDS: PATIENTS, DOCTORS, PSYCHOLOGY, MENTAL HEALTH, THERAPY, HEALTH STATS MANAGING.

					ІПЗ.КР.Б – 121 – 25 – ПЗ			
Змн.	Арк.	№ докум.	Підпис	Дата	Веб застосунок для моніторингу психологічного здоров'я людини	Літ.	Арк.	Аркуші
Розроб.		Свіргун А.В.						
Керівник		Савіцький Р.В.					4	96
Н. контр.		Локтікова Т.М.				Житомирська політехніка, група ІПЗ-21-1		
Зав. каф.		Вакалюк Т.А.						

ЗМІСТ

РЕФЕРАТ	3
ABSTRACT	4
ПЕРЕЛІК УМОВНИХ СКОРОЧЕНЬ	7
ВСТУП.....	8
РОЗДІЛ 1. АНАЛІЗ НАПРЯМКІВ ВИКОРИСТАННЯ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ	10
1.1 Постановка задачі	10
1.2 Аналіз аналогів програмного продукту	11
1.3 Вибір архітектури застосунку	16
1.4 Обґрунтування вибору інструментальних засобів	18
Висновки до першого розділу	20
РОЗДІЛ 2. ПРОЄКТУВАННЯ ВЕБЗАСТОСУНКУ ДЛЯ МОНІТОРИНГУ ПСИХОЛОГІЧНОГО ЗДОРОВ'Я	22
2.1 Визначення варіантів використання та об'єктно-орієнтованої структури системи	22
2.2 Розробка бази даних системи	26
2.3 Проектування та реалізація алгоритмів роботи системи	32
2.4 Реалізація функціоналу вебзастосунку для моніторингу психологічного здоров'я	35
Висновки до другого розділу	39
РОЗДІЛ 3. ІНТЕРФЕЙС ТА ПОРЯДОК РОБОТИ З ВЕБЗАСТОСУНКОМ	40
3.1 Структура інтерфейсу вебзастосунку для моніторингу психологічного здоров'я	40
3.2 Статистика зборів та загальна статистика платформи	46
3.3 Тестування платформи	47
Висновки до третього розділу	52
ВИСНОВКИ.....	54
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ	55

ДОДАТКИ.....	58
Додаток А.....	59
Додаток Б	60
Додаток В.....	61
Додаток Г	68

ПЕРЕЛІК УМОВНИХ СКОРОЧЕНЬ

API — Application Programming Interface

БД — база даних

CDN — Content Delivery Network

CRUD — Create, Read, Update, Delete

DTO — Data Transfer Object

HTML — HyperText Markup Language

HTTP — Hypertext Transfer Protocol

JSON — JavaScript Object Notation

JWT — JSON Web Token

JPQL — Java Persistence Query Language

MVC — Model-View-Controller

ООН — Організація Об'єднаних Націй

ПТСР — посттравматичний стресовий розлад

СКБД — система керування базами даних

Spring Boot JPA — Spring Boot Java Persistence API (підсистема ORM у Spring Boot)

Stripe SDK — набір бібліотек Stripe Software Development Kit

VS Code — Visual Studio Code

ВСТУП

Актуальність теми. Триваюча повномасштабна війна в Україні спричинила масовий психологічний стрес серед населення. За даними ООН, понад 10 млн українців потребують психосоціальної підтримки через травми, втрату рідних, переселення або постійні загрози життю. Для військових, які щодня стикаються з екстремальними умовами, ризик розвитку ПТСР, тривожних розладів та депресії є критично високим.

Створення вебзастосунку для відслідковування психологічного здоров'я — це не лише відповідь на актуальні виклики, але й інвестиція в майбутнє України. Поєднання технології з гуманітарною допомогою у додатку забезпечує доступність, оперативність та системність у боротьбі з психологічними наслідками війни. Для військових це може стати життєво важливим інструментом, який врятує тисячі від ран, які не помітні для звичайного ока.

Метою кваліфікаційної роботи є розробка застосунку, який допоможе організовувати візит до лікаря, запланувати терапії, відстежити перебіг лікування та слідкувати за показником сну, емоцій, активності та серцебиття.

Практичними завданнями кваліфікаційної роботи є:

- аналіз предметної області, вивчення програмних рішень у сфері трекінгу психофізіологічних показників, формулювання ключових вимоги користувачів та медичних консультантів до майбутньої платформи;
- визначення та аргументація добірки інструментальних засобів і технологій для реалізації фронтенду, серверної частини, сховища даних та інтеграцій зі сторонніми сервісами;
- розроблення повного переліку функціональних і нефункціональних вимог, формулювання варіантів сценаріїв використання залежно від ролі користувача — пацієнт, лікар, адміністратор;
- проектування структури бази даних для зберігання інформації про облікові записи, профілі пацієнтів, настроєві записи, біометричні показники, нагадування, платежі та журнали аудиту;

– розробка серверної частини сайту з використання Java Spring Boot, а саме Spring Data JPA для роботи з PostgreSQL, Spring Security для аутентифікації та авторизації;

– забезпечення інтерфейсу, що пропонує користувачеві зручну навігацію, динамічні графіки статистики та можливість перегляду персональних рекомендацій;

– створення й реалізація основних модулів платформи, включаючи запис і редагування настрою, імпорт даних Google Fit, аналітичний блок для обчислення трендів, систему нагадувань і нотифікацій електронною поштою, керування підписками і платежами через Stripe;

– проведення комплексного тестування клієнтської й серверної частин за допомогою набору тест-кейсів та модульних перевірок, фіксування й усунення виявлених дефектів, підтвердивши відповідність розробленого продукту заявленим вимогам і коректності роботи.

Об’єктом дослідження є система для моніторингу психологічного здоров’я, яка надає шляхи для пошуку лікаря та керування перебігом терапії.

Предметом дослідження є покращення процесу адміністрування амбулаторій для людей з психологічними проблемами та автоматизація процесу лікування. Дослідження направлене на вивчення технологій та методів, задля забезпечення автоматизованого управління процесами терапії та адміністрування закладами лікування.

У майбутньому система може допомогти користувачам поліпшити психологічне здоров’я. Основним функціоналом є процес налагодження спілкування між пацієнтом та лікарем, проте звичайний користувач також зможе переглядати свою статистику, щоб зрозуміти чи є у нього сховані проблеми. Це допоможе полегшити лікування та надати простий доступ до всіх функцій.

За тематикою дипломної роботи були опубліковані тези [1-2].

РОЗДІЛ 1. АНАЛІЗ НАПРЯМКІВ ВИКОРИСТАННЯ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

1.1 Постановка задачі

Кінцевий продукт роботи — це вебзастосунок для моніторингу психологічного здоров'я, який має допомогти в полегшенні процесу терапії через автоматизацію та спрощення збору життєво важливих даних.

Вебзастосунок має такий архітектурний поділ на модулі:

- модуль управління користувачами містить функціонал реєстрації, логіну, рольової політики (пацієнт, лікар, адмін) та налаштувань безпеки. Використовує Spring Security та зберігає дані у PostgreSQL [3].
- модуль пацієнта реалізує збір даних (сон, харчування, настроїв), відстежування терапії, оплат та комунікацію з лікарем;
- модуль лікаря забезпечує керування пацієнтами, планування терапевтичних сесій та формування діагнозів;
- модуль адміністратора відповідає за керування користувачами (додавання/видалення лікарів чи пацієнтів), генерацію фінансових звітів та логування дій;
- модуль інтеграцій відповідає таким API: Google [4] для отримання статистики по здоров'ю пацієнта та платіжні інтеграції через Stripe [5]. Вони допоможуть отримати біометричні дані, дані про сон та кроки, серцебиття та кілокалорії, що синхронізуються з базою даних та оновлюватиметься у профілі пацієнта, допоможуть проводити оплати терапії, виставлення рахунків та вивід історії платежів.

Основними завданнями розробки буде розробка дизайну вебзастосунку, побудова архітектури системи, розробка основного функціоналу, тестування та вдосконалення проєкту.

Клієнтська частина буде більше налаштована для пацієнтів та лікарів. Для кожного з них буде зроблено персоналізований дизайн з функціями доступними відповідно ролям користувачів. Окремо передбачено сторінки для

адміністратора з повноцінним доступом до даних застосунку.

У серверній частині передбачено використання JWT-токену [6] для авторизації та розподілення доступу відповідно до ролей користувачів. Чітке розділення HTTP-рівня та бізнес-логіки забезпечують окремі шари контролерів, сервісів і репозиторіїв. Для додаткових функцій буде проводитися інтеграція Google та Stripe, що допоможе додати функціонал статистики здоров'я та оплати підписки.

Клієнтська частина буде повноцінно розкривати всі функції починаючи від реєстрації нового користувача, редагування власних даних, корегування оплат для пацієнтів, керуванням перебігу терапій та розповсюдження даних серед інших користувачів, безперервної можливості спілкування між пацієнтами та лікарями та закінчуючи можливістю користувача відслідковувати власного стану.

Окремо від програми буде проводитися функціональне тестування за допомогою Postman [7], що допоможе відслідковувати перебіг розробки, точність виконання завдань та правильне підключення сторонніх API. Також буде проводитися ручне тестування, де тестувальник буде переглядати всю платформу з повною перевіркою можливостей сайту.

У кінці кінців, результатом розробки стане повноцінна платформа для медичних центрів для лікування психологічних захворювань та проведення профілактичних зустрічей та відслідковування. Це допоможе спростити процес для відслідковування стану пацієнта, надання рекомендацій від лікаря та менеджменту процесу оплати та спілкування.

1.2 Аналіз аналогів програмного продукту

Платформа для моніторингу психологічного здоров'я розрахована на спрощення спілкування між пацієнтом та психологом/психотерапевтом/психіатром. Додаток має охоплювати всі сфери, які впливають на психологічний стан пацієнта. Людина має бути впевненою, що вебзастосунок надає професійну допомогу для ментальних проблем. Однією з

наданих функцій вебзастосунку є відслідковування перепадів стану пацієнта та загального перебігу терапії.

Існує вдосталь аналогів системи, які мають в собі переваги та недоліки у використанні. Розглянемо декілька з них для кращого розуміння ринку:

- Moodbeam — це пристрій у вигляді браслета, який дозволяє користувачам відстежувати свій настрій протягом дня [8];
- Headspace — популярний додаток для медитації та уважності, який допомагає зменшити стрес і тривогу [9];
- MOZHNA — це платформа для вибору психолога та замовлення психологічної консультації [10];
- «Spokiy» — мобільний додаток для медитацій та покращення психологічного стану [11];
- «Rozмова» — мобільний додаток, який надає користувачам можливість спілкування з психологами та отримання підтримки [12].

Всі вищеперелічені застосунки мають за мету допомогти людям покращити свій психологічний стан. Проте, кожен з них має різний функціонал чи підхід. Деякі додатки зв'язують людей зі спеціалістами, інші ж – налаштовані на самостійний процес зцілення. У таблиці 1.1 показано порівняння всіх функцій, які присутні у програмах.

Таблиця 1.1

	Moodbeam	Headspace	MOZHNA	Spokiy	Rozмова	HealthApp
Відстеження настрою	+	-	-	-	-	+
Медитації та релаксація	-	+	-	+	-	-
Пошук психолога	-	-	+	-	+	+
Онлайн-консультації	-	+	+	-	+	+
Техніки самодопомоги	-	+	-	+	-	-
Вправи на уважність	-	+	-	+	-	-
Доступ до статей та ресурсів	-	+	-	-	+	-

Відслідковування статистики Google Fit	-	-	-	-	-	+
Двомовність	-	-	-	-	-	+
Перегляд оплат	-	-	-	-	-	+

Кожен із розглянутих додатків для моніторингу та підтримки психологічного здоров'я має унікальний набір функцій, спрямованих на різні потреби користувачів. Вибір відповідного інструменту залежить від індивідуальних потреб: відстеження настрою, медитації, негайна підтримка через чат-бот або доступ до професійних психологічних послуг. Розглянемо більш детально такі застосунки: Moodbeam, Headspace та Rozmova.

Moodbeam — це проєкт, який першочергово був створений для відслідковування перепадів настрою дітей. Загальний принцип роботи полягає у натисканні на кнопки (жовта для радісних емоцій, тоді як для негативних є синя) на браслеті, який після цього надсилатиме отриманні дані у додаток.



Рисунок 1.1 – Вигляд браслету moodbeam

Якщо ж людина не вносить зміни у настрої, то браслет вібрує для нагадування, щоб користувач додав дані про свій стан. Також є додаткові функції по відслідковуванню режиму сну та вимірюванню кількості кроків за день, що

також допомагає зрозуміти внутрішній стан. На рис. 1.1 та 1.2 показано вигляд девайсу та сумісної програми.

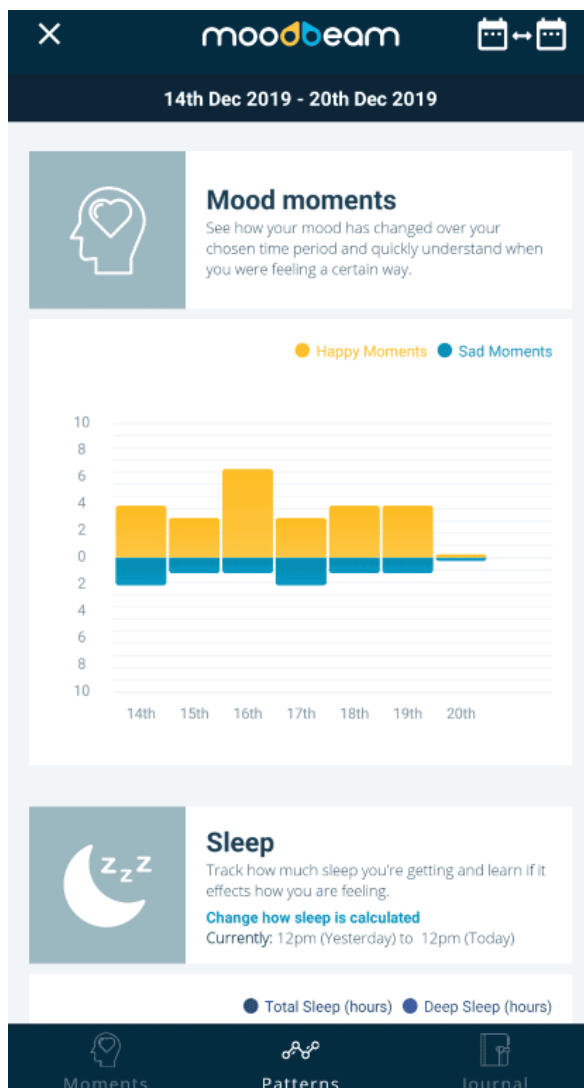


Рисунок 1.2 – Програма для синхронізації даних з браслетом

Headspace — це вебресурс, який розрахований для поверхневої допомоги через медитації. У застосунку наявні різноманітні практики для сну, заспокоєння негативних емоцій, тривожності та інших аспектів життя. Платформа надає перелік різноманітних ресурсів, щоб користувачі могли знайти запис медитації/афірмації чи пораду у будь-якій ситуації. Також цей простір за додаткову плату надає консультації з психологічного здоров'я у власного тренера. Нічого не вказує, що це заміняє справжнього лікаря, проте, це може допомогти у повсякденних проблемах. На рис. 1.3 зображено план та перелік всього, що відкривається з підпискою.

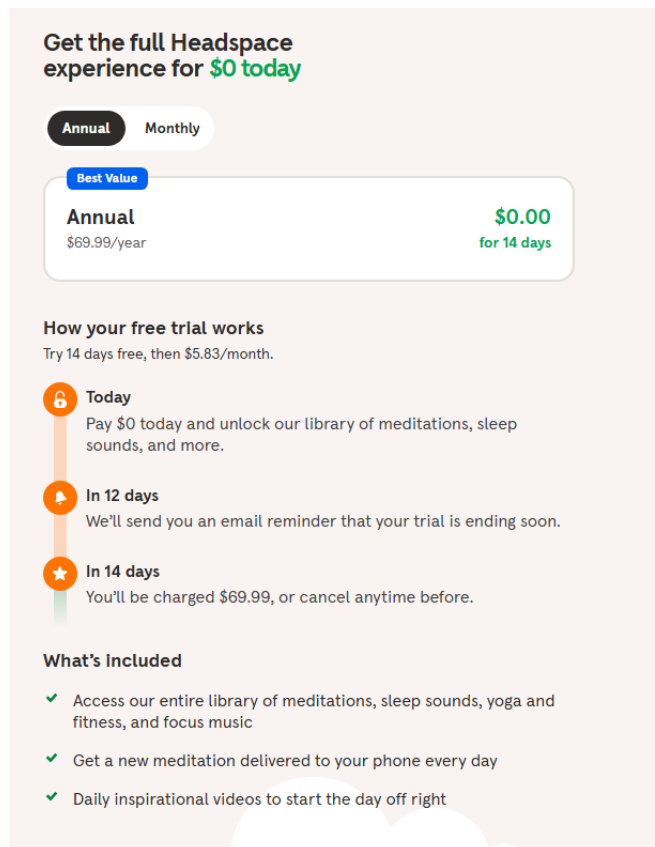


Рисунок 1.3 – Перелік послуг від Headspace

Rozmova — це застосунок, що допомагає знайти спеціаліста через персоналізовану анкету. Там представлений каталог лікарів, де показано його направленість, освіта, кваліфікація, цінності, а також статті та відеозвернення. Для кращого спілкування присутній вбудований чат, але ви завжди можете перейти на інший, більш зручний, месенджер. На рис. 1.4 показано вигляд вебсайту з переліком фахівців.

Після деталізованого аналізу застосунків можна створити розгорнутий перелік функцій, які зможуть повноцінно охопити всі аспекти у лікуванні. По-перше, платформа має надавати можливість налагодження спілкування з лікарями (безперервне спілкування між користувачами, створення майбутніх терапевтичних сесій, додавання спеціалістів у власному профілі). По-друге, персоналізований кабінет у кожного користувача з можливістю редагування даних. Також кожен користувач має мати доступ до різноманітних трекерів та загальної статистики щодо здоров'я. Останньою з майбутніх функцій має бути відслідкування оплати користувачів.

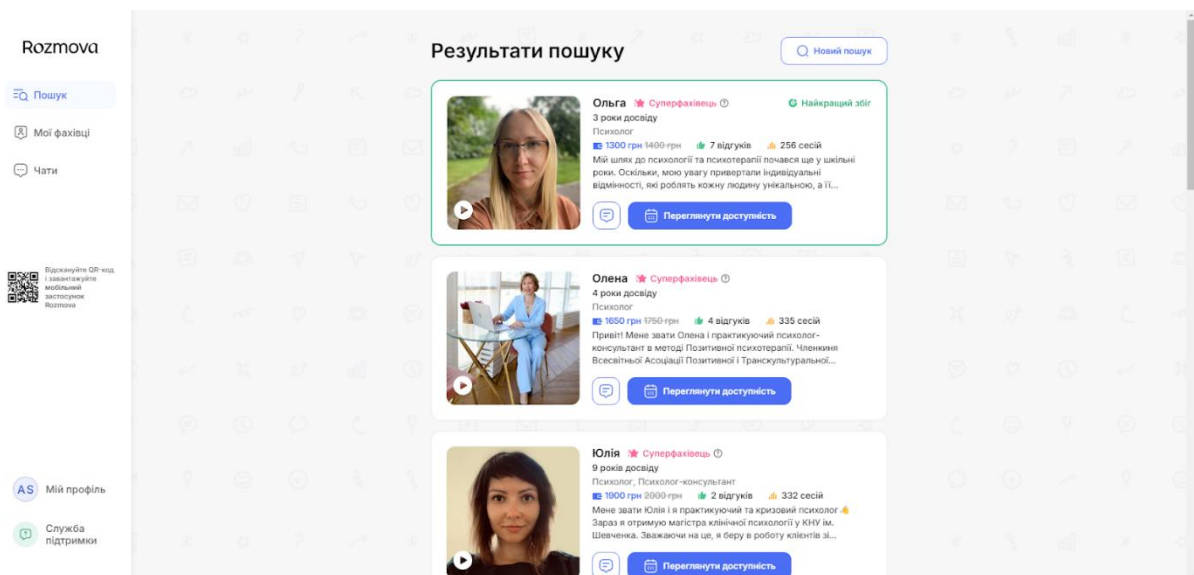


Рисунок 1.4 – Вигляд персонального кабінету у Rozmova

1.3 Вибір архітектури застосунку

Застосунок для моніторингу психологічного здоров'я містить різні типи даних (стан сну, настрій, активність, ритм серця, платежі тощо), інтеграції та кілька ролей, такі як: пацієнти, лікарі, адміністратори. Це вказує на потенційно чималу кількість функціональних модулів.

Застосунок має обробляти чимало пацієнтів, надавати можливість листування та надсилання нагадувань на електронну пошту, зберігати історію консультацій, вести оплати. У перспективі зростання кількості користувачів необхідне масштабування — і на рівні обчислень, і на рівні бази даних.

Є потреба взаємодіяти з Google та Stripe для налаштування додаткових функцій. Це вимагає чітко визначених точок входу, які можуть обслуговуватися різними компонентами системи.

У зв'язку з всіма попереднім вимогами було обрано клієнт-серверну архітектуру (рис. 1.5). Дана архітектура чітко відокремлює відповідальності. React-клієнт рендерить інтерфейс, виконує валідацію на стороні користувача й надсилає невеликі JSON-запити, тим самим зменшуючи навантаження на процесор серверної машини. Spring Boot-сервер займається тільки бізнес-логікою, зберіганням даних та інтеграціями: контролери приймають HTTP-запит, сервіси реалізують правила домену, а репозиторії інкапсулюють доступ до

PostgreSQL. Такий поділ спрощує тестування — кожен шар можна ізолювати й перевірити окремо.

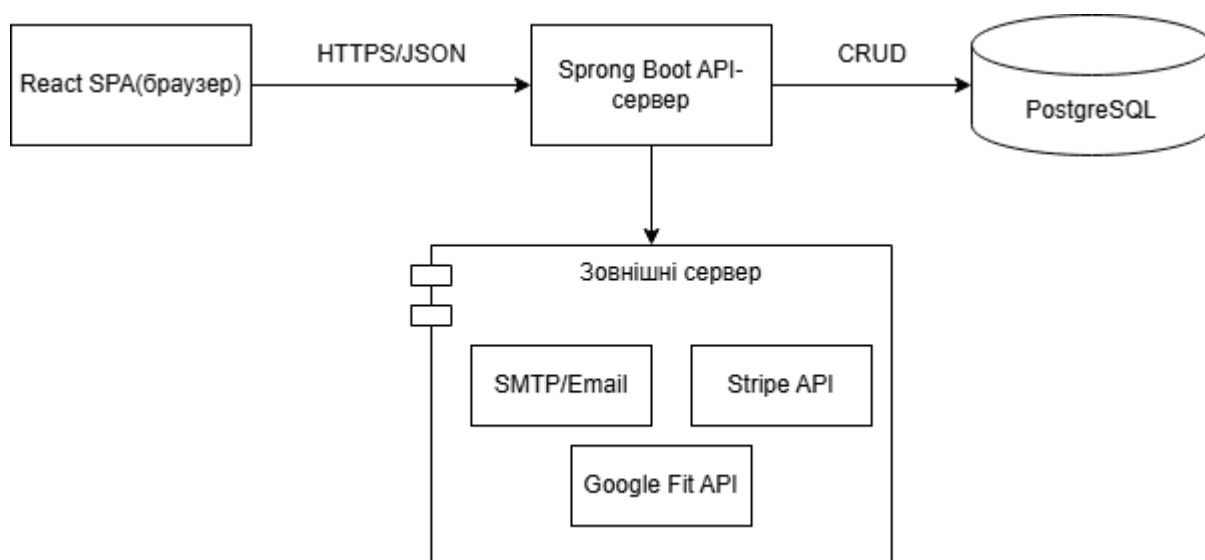


Рисунок 1.5 – Схема клієнт-серверної архітектури вебзастосунку

Зовнішні інтеграції підключено як окремі клієнтські компоненти сервісного шару. Google Fit синхронізує біометрію за допомогою офіційного REST-SDK; Stripe приймає платежі через SDK та webhook-контролер; JavaMailSender відправляє e-mail-нагадування й сповіщення. Усі вони використовують чітко визначені точки входу API-серверу, тому додавання нових сервісів (наприклад, іншого постачальника платіжних чи медичних даних) не потребує змін клієнтської частини та не порушує існуючі контракти.

Обрана модель масштабування базується на безстанних JWT-сесійних токенах: кожен запит містить власний маркер, тому будь-який реплікований екземпляр Spring Boot може його обробити. PostgreSQL підтримує потокову реплікацію, тож зростання обсягу даних користувачів вирішується горизонтальним масштабуванням читачів бази або шардингом. Витрати на інфраструктуру залишаються помірними, оскільки CDN роздає статичний фронт, а серверна частина споживає ресурси лише під час бізнес-операцій, не беручи на себе рендеринг UI.

Загалом клієнт-серверна архітектура забезпечує ясність коду, легкість супроводу й готовність до майбутнього розширення функціоналу, що критично

для платформи, яка оперує конфіденційними психологічними даними та має інтегруватись із різними зовнішніми системами.

1.4 Обґрунтування вибору інструментальних засобів

На початковій стадії розробки головним завданням був вибір стеку технологій, який би дозволив ефективно керувати даними, забезпечувати високу продуктивність та масштабованість системи, а також надавати зручний інтерфейс для користувача. Крім того, важливим фактором було дотримання стандартів безпеки та конфіденційності, оскільки йдеться про обробку чутливих персональних даних. Окремий блок у проєкті становить інтеграція Google API, яка покликана отримати додаткові біометричні показники, синхронізувати їх з основною системою.

Основою бекенд-архітектури став Java з використанням Spring Boot [13], який спростив створення застосунків завдяки “starter-пакетам” та вбудованим конфігураціям. Наявність Spring Data JPA полегшує роботу з базою даних, а Spring Security дає можливість впроваджувати різні механізми аутентифікації та авторизації. Більша частина завдань застосунку уже реалізовані в репозиторії як повноцінні абстракції Spring. Інші конкуренти не можуть надати такого самого покриття, що збільшить час на написання коду у лічені рази.

Spring Data JPA бере на себе всі типові операції зі збереження й вибірки даних, забезпечує транзакційну цілісність і надає зручні засоби для формування запитів, тож розробникові не потрібно писати SQL вручну. Spring Security централізує питання доступу: від аутентифікації користувачів до перевірки прав на рівні окремих методів чи API-маршрутів, водночас підтримуючи інтеграцію з зовнішніми постачальниками та безпечне зберігання облікових даних. Разом ці модулі знімають більшість рутинних задач, пов’язаних із роботою з базою та захистом ресурсів, і дозволяють зосередитися на бізнес-логіці платформи.

Для розробки інтерфейсу було обрано React, адже його віртуальна DOM-структура дає змогу забезпечити швидку реакцію інтерфейсу на дії користувачів [14]. Також при збільшенні складності UI React буде мати високу

продуктивність. Іншою перевагою була наявність великої кількості бібліотек та готових компонентів, що допомогло пришвидшити розробку та полегшити підтримку застосунку. Додатково було прийнято до уваги, що застосунок у майбутньому може бути розширений до мобільного застосунку, через що вибір на користь React допомагав зменшити кількість переробки фронтенду.

Щодо ж інтеграційних сервісів та API, то для функціоналу програми потрібні були Stripe для проведення оплат підписки пацієнтів, отриманням статистики операцій та можливість контролю транзакцій та Google API [15] для отримання біометричних даних, їх відслідковування та підключення додаткових функцій по типу реєстрація/входу в аккаунт.

Для забезпечення надійного зберігання даних про пацієнтів, лікарів, платежів та біометричні показники необхідна спеціалізована база даних, яка відповідає всім потребам системи. Тому проведемо порівняння декількох СКБД. Порівняльна таблиця з основними характеристиками обраних баз даних наведена нижче (табл. 1.2).

Таблиця 1.2

	MySQL	Microsoft SQL Server	Oracle	MongoDB	PostgreSQL
Логічна модель даних	Реляційна	Реляційна	Реляційна	Документно-орієнтована	Реляційна
Ліцензія	Open Source (GPL)	Пропрієтарна	Пропрієтарна	SSPL	Open Source (PostgreSQL License)
Області застосування	Малий та середній бізнес	Корпоративний сектор	Переважно великий бізнес	Великий, середній та малий бізнес	Великий, середній та малий бізнес
Складність налаштування	Низька, однак складно підтримувати складні кластери	Висока (складна інфраструктура, ліцензії)	Дуже висока (потребує кваліфікованих DBA, дорогі ліцензії)	Середня (кластеризація проста, проте потрібне розуміння NoSQL)	Середня (налаштування реплікації потребує деяких знань)

Продовження таблиці 1.2

Платформи	Windows, Linux, macOS	Windows, Linux(обмежано)	Windows, Linux, Unix-системи	Windows, Linux, macOS	Windows, Linux, macOS, Unix-системи
Мова маніпуляції даними	SQL	T-SQL	PL/SQL	MQL (MongoDB Query Language)	SQL (розширена підтримка)

На підставі табл 1.2 можна визначити, що PostgreSQL є найбільш вдалим вибором для даного проєкту щодо моніторингу психологічного здоров'я. Він забезпечує гнучке та порівняно просте масштабування, а також повноцінні транзакції. Також PostgreSQL не поступається комерційним рішенням, і водночас залишається безкоштовною СКБД з відкритим кодом, що гарантує надійне зберігання медичних і фінансових даних із можливістю масштабування й аналітичної обробки в реальному часі.

Щодо ж вибору на користь реляційної бази даних, а не документно-орієнтованої, то це було зроблено через використання декількох різних ролей, мапінг яких був би надзвичайно складним без використання колонок з посиланням на необхідні інші значення. Також так мінімалізується кількість помилок при збереженні даних та збільшується ефективність виклику значень, адже у реляційних базах даних є можливість надати каскадний зв'язок між стовпцями у різних таблицях.

Висновки до першого розділу

У першому розділі було виконано постановку задачі на кваліфікаційну роботу та визначено практичні завдання проєкту. Проведено дослідження ринку та аналогів, завдяки чому було визначено основні функціональні можливості системи: налагодження спілкування з лікарями, персоналізований кабінет у кожного користувача, доступ до різноманітних трекерів, відслідкування дотримання правил та оплати.

Також було продемонстровано, обґрунтовано та доведено вибір архітектури для розробки системи. Було обрано клієнт-серверну архітектуру, яка

відокремлює інтерфейс користувача від бізнес-логіки, спрощує масштабування, полегшує тестування й забезпечує гнучке підключення зовнішніх сервісів.

Для розробки клієнтської частини було використано мову програмування React та мову гіперрозмітки HTML. Для серверної частини – Java з використанням Spring Boot, а в якості бази даних PostgreSQL. З інтеграційних сервісів та API було використано Stripe та Google API.

РОЗДІЛ 2. ПРОЄКТУВАННЯ ВЕБЗАСТОСУНКУ ДЛЯ МОНІТОРИНГУ ПСИХОЛОГІЧНОГО ЗДОРОВ'Я

2.1 Визначення варіантів використання та об'єктно-орієнтованої структури системи

Основна мета полягає у підтримці здоров'я користувачів шляхом моніторингу стану психоемоційного та фізичного здоров'я через збір даних про сон, харчування, настрій та активність. Платформа надає можливість пацієнтам зручно вносити свої дані, переглядати статистику змін, а лікарям — аналізувати ці дані та здійснювати професійну підтримку. Основною метою є створення умов для покращення загального стану здоров'я, профілактики психологічних проблем та підвищення якості життя пацієнтів.

Внутрішні користувачі

Внутрішні користувачі мають різні обов'язки відповідно до ролей, що приведено нижче (табл. 2.1).

Таблиця 2.1

Внутрішні користувачі

Актор	Короткий опис
Адмін	Управляє всією системою, переглядає загальну статистику, створює нових користувачів з роллю "лікар".
Лікар	Переглядає статистику власних пацієнтів, створює терапевтичні сесії, спілкується з іншими користувачами та може керувати своїм профілем.
Пацієнт	Керує власним профілем від самого початку, може додавати дані про свій стан, має такий самий доступ до терапій та переписок, що і лікар.

На рис. 2.1 зображено побудовано діаграму варіантів використання системи. Деякий функціонал було опущено через його незначні прояви. Для прикладу, пацієнт має першочергову інструкцію по використанню сайту, що показано тільки один раз.

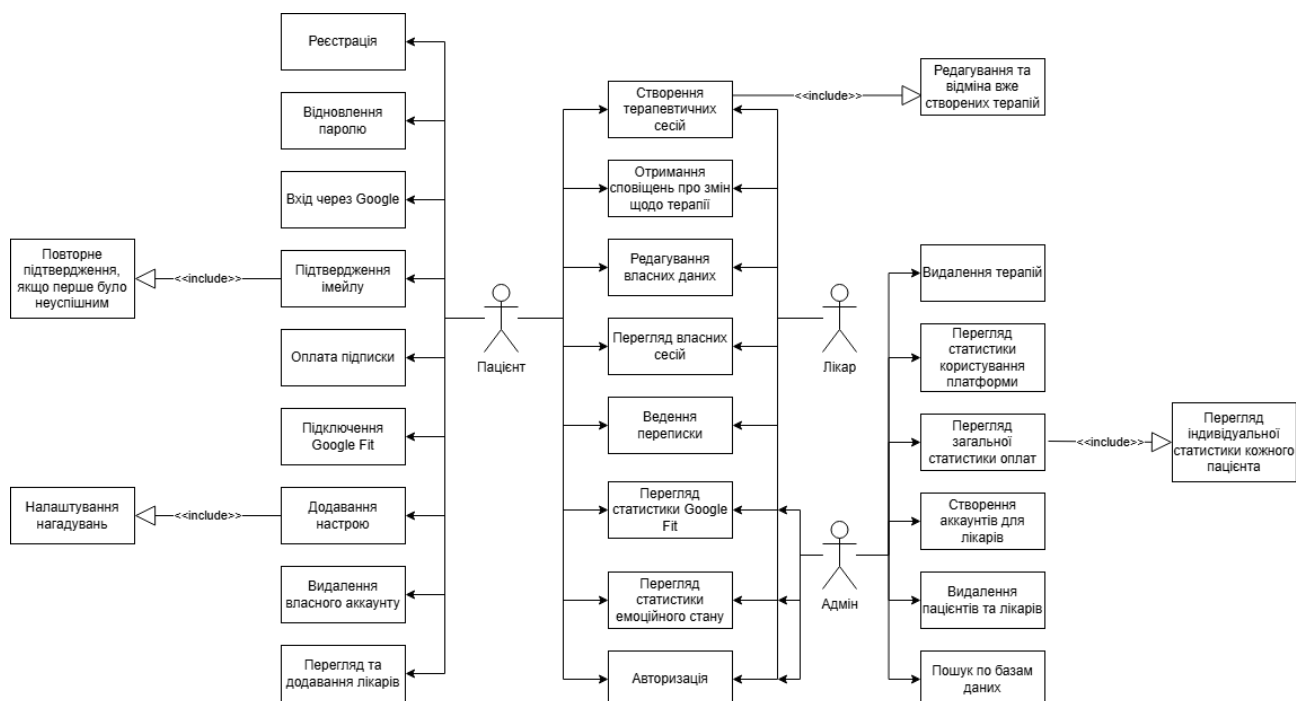


Рисунок 2.1 – Діаграма базових варіантів використання

Було виділено декілька основних прецедентів для кожної ролі. Кожен з них описано у таблицях 2.2 – 2.6.

Таблиця 2.2

Ім'я варіанту використання: Авторизація		Рівень важливості: дуже високий
Основний актор: Адмін, Лікар, Пацієнт		
Інші учасники: -		
Короткий опис:	Дозволяє користувачам входити у систему через спеціальну форму.	
Передумови:	Користувач має мати вже створений аккаунт та використовувати необхідну форму.	
Основний потік:	Після створення аккаунту кожен з користувачів матиме можливість входу у систему. Форма має відмінності відповідно до актору. У Пацієнтів форма буде розширена з додатковими функціями відновлення паролю, створення аккаунту та входу за допомогою Google.	
Альтернативні/виключні потоки:	Якщо сталася помилка, то користувач має бути проінформований, чому саме вона сталася.	

Таблиця 2.3

Ім'я варіанту використання: підключення Google Fit		Рівень важливості: високий
Основний актор: Пацієнт		
Інші учасники: Лікар, Адмін		

Продовження таблиці 2.3

Короткий опис:	Сценарій надає користувачу можливість підключити сторонній API для отримання та контролю додаткової статистики.
Передумови:	Пацієнт повинен мати Google аккаунт, доступ до програми Google Fit на іншому пристрої, де буде підключений смарт-браслет, або користувач має можливість мануально вводити дані.
Основний потік:	Варіант використання дозволяє Пацієнту підключити Google Fit до свого аккаунту, щоб усі ролі з доступом до статистики могли переглянути перебіг його стану.
Альтернативні/виключні потоки:	Якщо сталася помилка, то користувач має бути проінформований у чому саме вона сталася.

Таблиця 2.4

Ім'я варіанту використання: оплата підписки		Рівень важливості: дуже високий
Основний актор: Пацієнт		
Інші учасники: Адмін		
Короткий опис:	Надає Пацієнту можливість оплати підписки на сервер.	
Передумови:	Підключення Stripe API у проєкті, ініціалізація в системі, Пацієнт має мати дійсну банківську картку.	
Основний потік:	Перед відкриттям доступу до основних сторінок кожен пацієнт буде проходити перевірку на виявлення, чи була проведена оплата. У випадку, коли вона не пройшла, то буде відкрита сторінка з вибором підписки та формою для оплати.	
Альтернативні/виключні потоки:	Якщо сталася помилка, то користувач має бути проінформований у чому саме вона сталася.	

Таблиця 2.5

Ім'я варіанту використання: отримання сповіщення про зміни щодо терапії		Рівень важливості: високий
Основний актор: Пацієнт, Лікар		
Інші учасники: -		
Короткий опис:	Отримання сповіщень на імейл про зміни у графіку проведення терапевтичних сесій.	
Передумови:	Лікар та Пацієнт мають мати дійсні імейли та зв'язок у застосунку.	

Продовження таблиці 2.5

Основний потік:	При будь-якій взаємодії з календарем щодо сесій кожен з задіяних користувачів буде отримувати повідомлення про те, що було виконано. Буде передбачено декілька підвидів імейлів: створення терапій, зміна стану (видалення, початок, закінчення) та попередження за годину до початку.
Альтернативні/виключні потоки:	Якщо сталася помилка, то користувач має бути проінформований у чому саме вона сталася.

Таблиця 2.6

Ім'я варіанту використання: створення аккаунтів для лікарів		Рівень важливості: дуже високий
Основний актор: Адмін		
Інші учасники: Лікар		
Короткий опис:	Надає Адміну можливість створення нових користувачів з роллю Лікар.	
Передумови:	Ініціалізація у систему, наявність незадіяних даних для реєстрації.	
Основний потік:	Варіант використання дозволяє Адміністратору створювати користувачів з роллю Лікар після того як людину було прийнято на посаду. Адмін створює незаповнений аккаунт для авторизації у систему, де Лікар у подальшому має надати відсутні дані.	
Альтернативні/виключні потоки:	Якщо сталася помилка, то користувач має бути проінформований у чому саме вона сталася.	

Функціональні вимоги

1. Система повинна забезпечувати авторизацію та реєстрацію користувачів через email і OAuth (Google).
2. Реалізація ролей користувачів: адміністратор, лікар, пацієнт.
3. Збір та збереження даних пацієнтів щодо якості сну, ритму серцебиття, кількості кроків за день, кількості вивільнених кілокалорій та настрою.
4. Забезпечення зв'язку між пацієнтом і лікарем через платформу.
5. Можливість лікарям аналізувати стан пацієнтів.
6. Відображення статистики змін у стані здоров'я пацієнтів у вигляді графіків або таблиць.

Нефункціональні вимоги

Сприйняття:

- час, необхідний для освоєння інтерфейсу вебсайту — 3–5 хвилин для нових користувачів, менше 2-х хвилин для досвідчених;
- час відповіді на дії користувача — не більше 3 секунд;
- інтерфейс повинен бути інтуїтивно зрозумілим, адаптованим для пацієнтів без спеціальної підготовки.

Надійність:

- платформа повинна бути доступною 24/7;
- автоматичне відновлення роботи після збоїв сервера.

Продуктивність:

- підтримка великої кількості одночасних користувачів (пацієнтів і лікарів);
- стабільна робота при високому навантаженні на сервер.

Системні вимоги

Вимоги до середовища виконання (мінімальна комплектація пристроїв користувачів): 4 ГБ оперативної пам'яті, двоядерний процесор із частотою не менше 2.0 GHz, стабільне з'єднання з інтернетом.

Вимоги до серверної частини: використання бази даних PostgreSQL для збереження даних пацієнтів, використання Spring Boot для серверної логіки та забезпечення безпечної авторизації через JWT, підтримка сервером технологій OAuth2 для аутентифікації через Google.

2.2 Розробка бази даних системи

Для збереження файлів та інформації у додатку використовуються PostgreSQL — це система управління базами даних, яка базується на реляційній моделі.

PostgreSQL — це потужна відкрита реляційна система управління базами даних (СУБД), розроблена спільнотою з відкритим вихідним кодом і поширювана під ліцензією PostgreSQL. Вона підтримує повний набір стандартних SQL-конструкцій, включно з транзакціями ACID, складними JOIN-

запитами та підзапитами, а також багаторівневими механізмами контролю паралельності. Завдяки своїй модульній архітектурі, PostgreSQL легко налаштовується під різні робочі навантаження — від невеликих однокористувацьких додатків до масштабних корпоративних систем з тисячами одночасних підключень.

Серед ключових переваг PostgreSQL — підтримка розширень (PostGIS, pg_trgm, pgcrypto тощо), що дозволяє безпосередньо вбудовувати додаткову функціональність у саму СУБД, не вдаючись до сторонніх сервісів. Система знайшла широке застосування у фінансових установах, державних проектах і медичних платформах, де SQL-надійність, збереження цілісності даних і можливість масштабування мають критичне значення. Наявність потокової реплікації й розпізнавальних засобів аварійного відновлення гарантує безперервність сервісу та швидке повернення до роботи після збоїв.

У проєкті моніторингу психологічного здоров'я PostgreSQL використовується для зберігання складної, зв'язаної моделі даних — облікових записів користувачів, їхніх профілів, історії вимірювань та журналів подій. Жорсткі обмеження цілісності (foreign key, not null, unique) захищають від неконсистентності, а матеріалізовані подання та індекси — від тривалих запитів в аналітичних розділах. В результаті платформа отримує надійну і продуктивну основу для збирання й обробки медичних та поведінкових даних.

На рисунку 2.2 показана схема бази даних.

У клієнт–серверному проєкті, де всі операції створення йдуть через єдиний Spring Boot API, використання автоінкрементних bigint-ключів було обрано заради максимальної продуктивності та компактності індексів у PostgreSQL. Оскільки генерація нових записів централізована на сервері, немає потреби в незалежному розподіленому створенні ідентифікаторів, яке забезпечують UUID. Таким чином bigint гарантує послідовні значення без фрагментації індексів, пришвидшує операції вставки й пошуку, полегшує налагодження й аналіз логів.

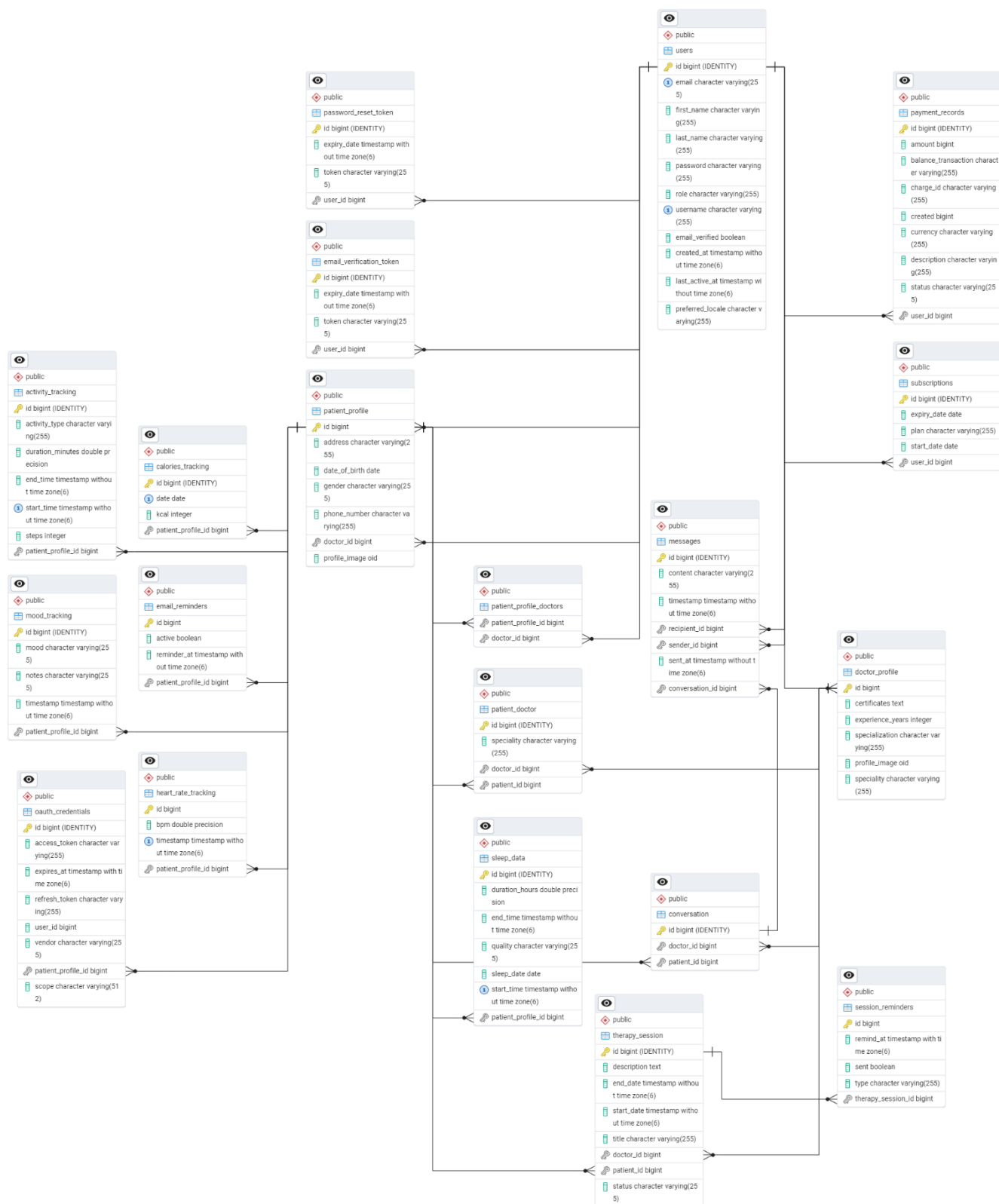


Рисунок 2.2 – Діаграма бази даних

База даних включає користувачів, профілі лікарів і пацієнтів, дані про здоров'я пацієнта, повідомлення та платежі. Опис бази даних та її основних таблиць представлений у таблицях у 2.7-2.21.

Таблиця 2.7

Опис таблиці users

Поле	Тип	Призначення
id	bigint	Ідентифікатор користувача
email	varchar(255)	Email користувача
first_name	varchar(255)	Ім'я користувача
last_name	varchar(255)	Прізвище користувача
password	varchar(255)	Пароль користувача
role	varchar(255)	Роль користувача (PATIENT, DOCTOR, ADMIN)
username	varchar(255)	Нікнейм користувача
email_verified	boolean	Статус підтвердження електронної пошти
created_at	timestamp	Дата створення акаунта
last_active_at	timestamp	Дата останньої активності
preferred_locale	varchar(255)	Код локального інтерфейсу для визначення мови

Таблиця 2.8

Опис таблиці patient_profile

Поле	Тип	Призначення
id	bigint	Ідентифікатор профілю пацієнта
address	varchar(255)	Адреса пацієнта
date_of_birth	date	Дата народження пацієнта
gender	varchar(255)	Стать пацієнта
phone_number	varchar(255)	Телефонний номер пацієнта
doctor_id	bigint	FK → doctor_profile
profile_image	oid	Фото профілю пацієнта

Таблиця 2.9

Опис таблиці patient_profile

Поле	Тип	Призначення
id	bigint	Ідентифікатор профілю пацієнта
address	varchar(255)	Адреса пацієнта
date_of_birth	date	Дата народження пацієнта
gender	varchar(255)	Стать пацієнта

Таблиця 2.10

Опис таблиці doctor_profile

Поле	Тип	Призначення
id	bigint	Ідентифікатор профілю лікаря
certificate	text	Сертифікат лікаря
experience_years	integer	Стаж роботи

Продовження таблиці 2.10

specialization	varchar(255)	Спеціалізація лікаря
profile_image	oid	Фото профілю лікаря
speciality	varchar(255)	Додаткова спеціалізація

Таблиця 2.11

Опис таблиці activity_tracking

Поле	Тип	Призначення
id	bigint	Ідентифікатор запису активності
activity_type	varchar(255)	Тип (walking, cycling ...)
duration_minutes	double precision	Тривалість, хв
end_time	timestamp	Кінець активності
start_time	timestamp	Початок активності
steps	integer	Кількість кроків
patient_profile_id	bigint	FK → patient_profile

Таблиця 2.12

Опис таблиці sleep_data

Поле	Тип	Призначення
id	bigint	Ідентифікатор запису сну
duration_hours	double precision	Тривалість сну (години)
end_time	timestamp	Час завершення сну
quality	varchar(255)	Якість сну
start_time	timestamp	Час початку сну
patient_profile_id	bigint	FK → patient_profile
sleep_date	date	Дата сну

Таблиця 2.13

Опис таблиці calories_tracking

Поле	Тип	Призначення
id	bigint	Ідентифікатор запису кілокалорій
date	date	Дата споживання
kcal	integer	Калорії
patient_profile_id	bigint	FK → patient_profile

Таблиця 2.14

Опис таблиці heart_rate_tracking

Поле	Тип	Призначення
id	bigint	Ідентифікатор запису серцебиття
bpm	double precision	Пульс (уд./хв)
timestamp	timestamp	Час вимірювання
patient_profile_id	bigint	FK → patient_profile

Таблиця 2.15

Опис таблиці mood_tracking

Поле	Тип	Призначення
id	bigint	Ідентифікатор запису настрою
mood	varchar(255)	Оцінка настрою
notes	varchar(255)	Додаткові нотатки
patient_profile_id	bigint	FK → patient_profile
timestamp	timestamp	Час запису

Таблиця 2.16

Опис таблиці messages

Поле	Тип	Призначення
id	bigint	Ідентифікатор повідомлення
content	varchar(255)	Текст повідомлення
timestamp	timestamp	Час відправлення
recipient_id	bigint	FK → users
sender_id	bigint	FK → users
sent_at	timestamp	Фактичний час відправки
conversation_id	bigint	FK → conversation

Таблиця 2.17

Опис таблиці therapy_sessions

Поле	Тип	Призначення
id	bigint (IDENTITY)	Сеанс терапії
description	text	Опис сеансу
end_date	timestamp	Кінець
start_date	timestamp	Початок
title	varchar(255)	Заголовок
doctor_id	bigint	FK → users
patient_id	bigint	FK → users
status	varchar(255)	Статус

Таблиця 2.18

Опис таблиці oauth_credentials

Поле	Тип	Призначення
id	bigint	Ідентифікатор OAuth запису
access_token	varchar(255)	Токен доступу
expire_at	timestamp	Дата завершення дії токена
refresh_token	varchar(255)	Refresh-token
user_id	bigint	FK → users
vendor	varchar(255)	Назва провайдера
patient_profile_id	bigint	FK → patient_profile
scope	varchar(512)	Охоплені дозволи

Таблиця 2.19

Опис таблиці payment_records

Поле	Тип	Призначення
id	bigint	Ідентифікатор платежу
amount	bigint	Сума платежу
balance_transaction	varchar(255)	Транзакція в балансі
charge_id	varchar(255)	Ідентифікатор списання
created	bigint	Unix-time створення
currency	varchar(255)	Валюта
description	varchar(255)	Опис платежу
status	varchar(255)	Статус платежу
user_id	bigint	FK → users

Таблиця 2.20

Опис таблиці subscriptions

Поле	Тип	Призначення
id	bigint	Ідентифікатор підписки
expiry_date	date	Дата завершення підписки
plan	varchar(255)	Тип підписки
start_date	date	Дата початку підписки
user_id	bigint	FK → users

Таблиця 2.21

Опис таблиці session_reminders

Поле	Тип	Призначення
id	bigint	Первинний ключ
remind_at	timestamp	Коли нагадати
sent	boolean	Чи вже надіслано
type	varchar(255)	Тип
therapy_session_id	bigint	FK → therapy_session

2.3 Проектування та реалізація алгоритмів роботи системи

Додаток є простим у використанні, має всі основні функції системи моніторингу здоров'я, що дозволяє користувачам легко взаємодіяти з платформою.

У Додатку А показана діаграма активності платформи. На якій представлені всі ролі, які розподіляються користувачам після авторизації. У всіх унікальний набір функцій, хоч вони можуть бути і схожими у виконанні.

На представлений діаграмі зображено початкову точку входу користувача до платформи моніторингу психологічного здоров'я та подальші варіанти його взаємодії залежно від обраної ролі й стану облікового запису. Спершу

відбувається авторизація: новий користувач може зареєструватися через форму реєстрації або скористатися Google Oauth, а вже зареєстрований просто вводить логін і пароль. Якщо виникає потреба скинути пароль, реалізована відповідна гілка – користувач ініціює відновлення, отримує лист із токеном і задає новий пароль.

Далі діаграма розгалужується за ролями. Для адміністратора передбачені операції управління — перегляд статистики, реєстрація й модерація лікарів та пацієнтів, керування підписками й платежами. Лікар бачить власний дашборд із узагальненою статистикою пацієнтів, може створювати й призначати терапевтичні сесії, надсилати їм повідомлення та налаштовувати нагадування. Для пацієнта реалізовано перелік дій з особистим трекінгом: підключення Google Fit, додавання лікарів до профілю, заповнення щоденних записів настрою й біометрії, налаштування нагадувань.

У діаграмі також передбачена опція першого входу — “have a tour” — яка демонструє користувачеві короткий гід функціоналом платформи. Після цього система повертає його до основного меню згідно з роллю. Вкінці гілки з відслідковування емоцій та будь-яких дій з терапією передбачено крок “have a reminder” – підтвердження запланованих сповіщень, які надійдуть у встановлений час або при досягненні певних аналітичних порогів.

Уся логіка оформлена єдиним потоком, який гарантує: незалежно від способу входу чи ролі, користувач завжди потрапляє до відповідної секції застосунку, отримує доступ лише до дозволених операцій і бачитиме релевантні дані та інструменти для своєї ролі. Така схема спрощує розуміння поведінки системи і полегшує тестування всіх можливих сценаріїв (див. Додаток А).

Застосунок вже було розділено на модулі. Для презентації побудуємо діаграму компонентів (рис. 2.3.).

На даній діаграмі показано такі компоненти: Analytics, Messaging, Scheduler, Mood Tracking, Users, Notification, Admin. Users відповідає за модуль користувача, який надає можливість авторизації та редагування даних користувача.

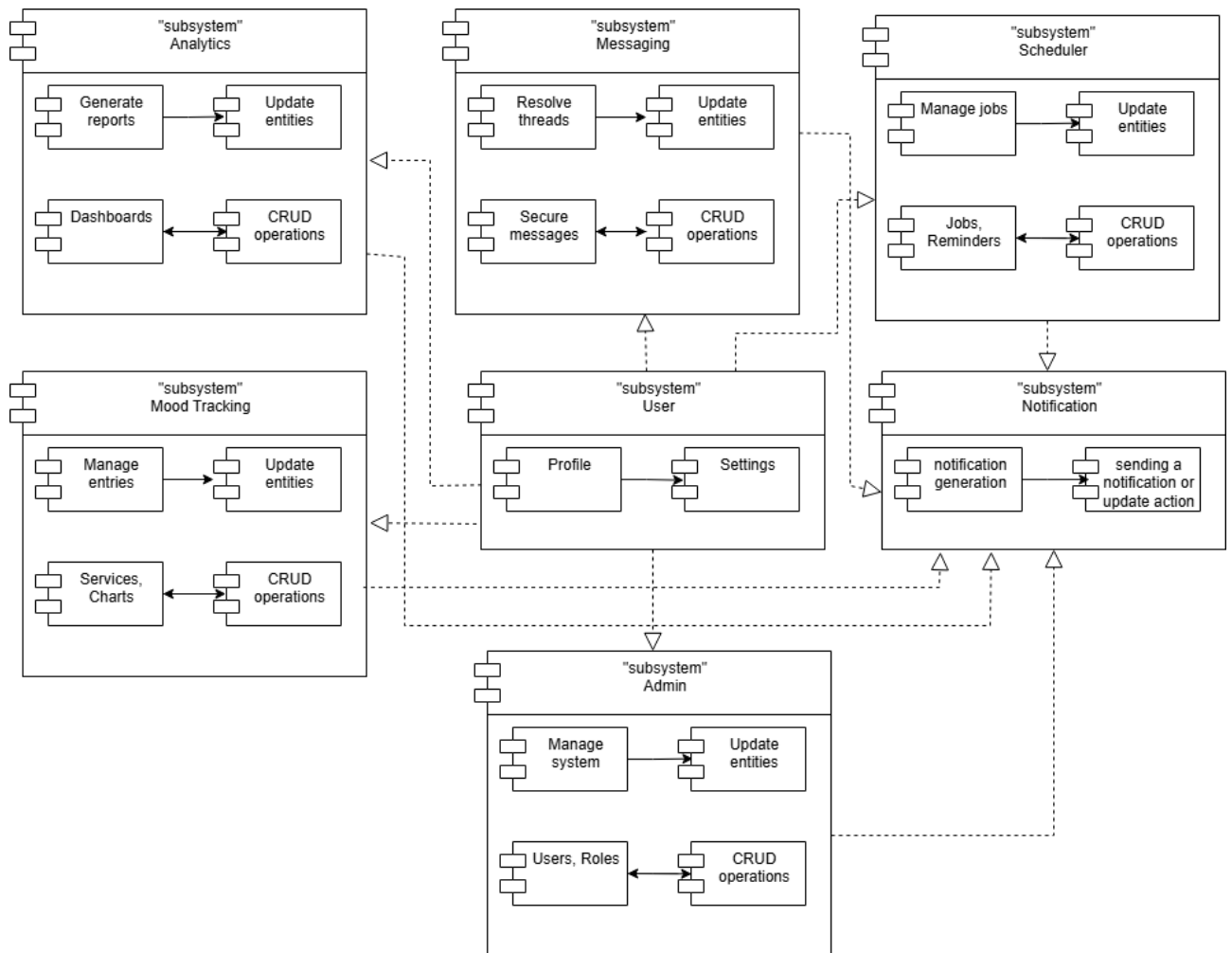


Рисунок 2.3 – Діаграма компонентів додатку

Analytics відповідає за збір загальної інформації та статистики щодо кожного з користувачів. Messaging — це модуль, який відповідає за безпечне надсилання повідомлень між пацієнтами та лікарями. Scheduler допомагає надсилати заплановані імейли щодо терапії чи оновлення даних про здоров'я. Mood Tracking — це модуль, який відокремлений від непередставленого модуль Google Fit, адже він відповідає тільки за збір емоцій, які користувач самостійно надсилає через необхідну форму. Admin — це модуль адміна, який збирає всі дані та керує користувачами платформи. Останнім представленим модулем є Notification, який відповідає за надсилання імейлів щодо зміну стану терапії.

Для кращого розуміння збору даних про стан пацієнта розглянемо алгоритм роботи модуля, який відповідає за збір даних про настрій (див. Додаток Б).

Користувач проходить авторизацію, відкриває сторінку MoodTrackerPage.jsx і надсилає до MoodTrackingController запит POST /api/mood-tracking, що містить JSON-об'єкт MoodTrackingRequestDto. Контролер передає DTO у MoodTrackingServiceImpl, де профіль пацієнта буде прочитано через PatientProfileRepository, створюється новий екземпляр сутності MoodTracking, зберігається у MoodTrackingRepository і конвертується назад у DTO за допомогою MoodTrackingMapper; далі контролер повертає клієнтові відповідь 201 Created із даними щойно створеного запису. Після цього фронтенд робить запит GET /api/mood-tracking?userId=..., контролер викликає MoodTrackingServiceImpl.getByUser, сервіс читає потрібні записи у MoodTrackingRepository, мапить їх у список DTO і повертає для відображення історії настрою. Коли користувач редагує існуючий запис, браузер надсилає PUT /api/mood-tracking/{id}; сервіс знаходить відповідну сутність, оновлює й зберігає її, а оновлений DTO повертається у відповідь. Після кожного створення або оновлення запису MoodTrackingServiceImpl викликає AnalyticsServiceImpl.recalcStats: якщо нові середні значення виходять за порогові межі, EmailReminderServiceImpl формує й надсилає користувачеві тривожне сповіщення, а якщо показники залишаються у нормі, сповіщення не генеруються. Окремо ReminderScheduler щодня запускає sendDailyReminders у EmailReminderServiceImpl, який через EmailReminderRepository знаходить користувачів без записів за останню добу й розсилає їм електронні листи-нагадування. Таким чином у процесі задіяні фронтенд, контролер, сервісний шар, репозиторії, мапер, сервіс аналітики та дві гілки сповіщень — негайна (alert) і відкладена (daily reminder).

2.4 Реалізація функціоналу вебзастосунку для моніторингу психологічного здоров'я

Реалізація функціоналу вебзастосунку для моніторингу психологічного здоров'я ґрунтується на цілісній, але шарованій архітектурі. У центрі застосунку — єдиний Spring Boot-сервер, що виступає API-сервером, і React-клієнт, який

живе окремою збіркою на CDN. Внутрішня будова бекенду слідує класичній схемі Controller → Service → Repository → Entity [16], а всі допоміжні класи розкладені по пакунках config, scheduler, security, listener та util. Контролери приймають HTTP-запити, валідовані за допомогою DTO-класів [17] — вони існують окремо від сутностей бази даних і виконують роль транспорту між фронтом та доменною моделлю. У сервісному шарі концентрується бізнес-логіка; тут працюють транзакції з анотацією @Transactional [18], а також стратегія обмеження доступу через @PreAuthorize [24]. Репозиторії є інтерфейсами Spring-Data JPA, тож за них планувальником CRUD-операцій стає саме середовище Spring, а ми зберігаємо лише сигнатури методів, покладаючись на генерацію JPQL за конвенцією назв.

У кодовій базі використано низку патернів, що роблять реалізацію прозорою й розширюваною. Доступ до даних інкапсулює патерн «Repository» [19], що дозволив винести SQL-деталі з бізнес-коду. Пара «DTO ↔ Entity» у зв'язці з ModelMapper реалізує «Data Transfer Object» і «Mapper» [20-21], гарантуючи, що зовнішній контракт не протече всередину схеми БД. У класах конфігурації застосовано «Factory / Builder» [22]. MailConfig створює екземпляр JavaMailSender, а SecurityConfig налаштовує ланцюжок фільтрів із JWT-фільтром, який виступає прикладом патерна «Decorator». Кронові задачі з нагадуваннями оформлено через @Scheduled, що повторює ідею патерна «Scheduler» [23], а AuditListener з @EntityListeners показує «Observer» [25] — він реагує на кожен життєвий цикл сутності і складає журнал у AuditLog. Для коду шифрування паролів використано «Strategy» [26]: через PasswordEncoder можемо підставити BCrypt або Argon 2, не змінюючи жодного іншого класу. RestTemplate реалізовано як «Singleton» [27], який повторно використовується для всіх зовнішніх HTTP-запитів. Це гарантує, що для виконання запитів до інших сервісів застосунок працює з одним об'єктом, а не створює його щоразу заново. Нарешті контролери наслідують «Template Method» [28]: BaseController містить загальний обробник помилок і пагінації, а конкретні контролери перевизначають лише бізнес-частину.

Підсистема платежів для платних консультацій реалізована через Stripe SDK (див. Додаток В). Конфігураційний клас `StripeConfig` встановлює `Stripe.apiKey`, витягуючи секрет із `application.yaml`. У `PaymentController` викликом `paymentService.createCheckoutSession()` створюється `Session`; метод повертає `sessionUrl`, який фронт отримує і відкриває у вікні Stripe Checkout. `PaymentServiceImpl` формує параметри сесії — ідентифікатор прайсу, домен підтвердження і `mode=»payment»`. Після успішного платежу Stripe надсилає `webhook` на `/api/v1/payments/webhook`; у `StripeWebhookController` підпис події перевіряється методом `Webhook.constructEvent(payload, sigHeader, endpointSecret)`. Якщо подія `invoice.payment_succeeded`, сервіс витягає `customer email`, підвантажує `User` через `UserRepository`, створює `PaymentRecord` і одночасно продовжує `Subscription` на місяць. Таким чином цикл «сесія → оплатив → webhook → сутність `PaymentRecord`» повністю закривається в бекенді без ручних тригерів.

Інтеграція з Google реалізується у двох площинах (див. Додаток Г). По-перше, для соціального входу використано Spring Security OAuth 2.0. У `SecurityConfig` додано `oauth2Login()`, а після успішного кол-беку `OAuth2SuccessHandler` перевіряє, чи існує рядок із `provider = «google»` у таблиці `OAuthCredentials`. Якщо ні, заводиться `User`, далі `PatientProfile`, а відповідний токен зберігається в `OAuthCredentials` для майбутніх запитів. По-друге, існує фоновий збирач даних з Google Fit: `GoogleFitSyncJob` позначений як `@Scheduled(cron=»0 0 */6 * * *»)`. У середині нього `GoogleFitClient` — це обгортка над `com.google.api.services.fitness.Fitness`, яка викликає `endpoint «users.dataSources.datasets.get»` та дістає `heart-rate` і `sleep`-датасети. Отримані значення перетворюються на DTO `HeartRateTrackingDto` і `SleepDataDto`, а далі через відповідні сервіси зберігаються як сутності `HeartRateTracking` і `SleepData`. Валідація дубля знаходиться у `HeartRateTrackingRepository.existsByPatientProfileAndTimestamp`, щоб уникнути повторних імпортів.

Структурно бекенд ділиться на декілька пакетів. Пакет `controllers` містить

					ІПЗ.КР.Б – 121 – 25 – ПЗ	
						37

по одному REST-класу на кожну бізнес область, наприклад MoodTrackingController, HeartRateTrackingController, PaymentController. Усі вони повертають або DTO, або сторінку Swagger-документації. Пакет services поділено на інтерфейси й Impl-класи; інтерфейс публікує контракт, а реалізація містить @Service і потрібні залежності. Пакет repositories — це Spring-Data інтерфейси, які працюють тільки з конкретною сутністю. Пакет entities налічує двадцять дві JPA-моделі; у них описані зв'язки @ManyToOne, @OneToMany і @ManyToMany, причому всі ключові FK позначені @OnDelete(cascade=CASCADE) для коректного ланцюжка видалення. Додаткові пакети: listener тримає AuditListener та ActiveUserTracker, scheduler — ReminderJob і GoogleFitSyncJob, security — JWT-фільтр, OAuth конфіг і рольову ієрархію, config — MailConfig, StripeConfig, GoogleFitConfig та ModelMapperConfig.

Завдяки такому поділу кожен шар має лише одну причину для зміни: контролери змінюються, коли змінюється REST-контракт; сервіси — коли змінюються бізнес-правила; репозиторії — коли змінюється модель даних. Це відповідає принципу Single Responsibility та полегшує юніт-тестування.

Фонові задачі не вимагають зовнішнього брокера повідомлень: Spring Scheduler ізолює логіку в один JVM-процес, а мережеві сесії залишаються статичними завдяки JWT, тож горизонтальне масштабування відбувається просто додаванням контейнерів. Нагадування та алерти надсилаються через NotificationService, який інтерфейсно абстрагований, що дає змогу замінити за необхідності SMTP на Push-шину Firebase, реалізацію можна переключити в конфігураційному файлі без змін решти коду.

Таким чином у реалізації вебзастосунку для моніторингу психологічного здоров'я використано перевірені часом патерни проєктування, чітко розмежовані шари відповідальності й інтеграції з зовнішніми сервісами Stripe та Google, які під'єднано через офіційні SDK, обгорнуті у власні сервіси й захищені механізмами валідації та журналювання. У підсумку проєкт отримав гнучку, безпечну й масштабовану основу, що дозволяє впевнено розвивати нові модулі

й при цьому не ламати вже існуючий, перевірений бойовими навантаженнями код.

Висновки до другого розділу

В процесі проєктування вебзастосунку для моніторингу психологічного здоров'я були проаналізовані усі варіанти використання програмного продукту відповідно до вимог програмного забезпечення. На основі отриманого результату була спроектована структура платформи. Проєктування включало в себе моделювання структури проєкту та взаємодії між їх складовими частинами.

Також було сконфігуровано базу даних для зберігання значень перед потраплянням їх на блокчейн, та для даних які будуть зберігатися на серверній частині додатку, таких як користувачі та категорії.

Було описано алгоритм роботи з додатком та послідовні пункти варіантів використання.

Також був наведений опис основного функціоналу додатку – це створення збору користувачем, його підтвердження адміністрацією, були розглянуті основні функції по взаємодії користувача зі збором.

Було описано застосування загальноприйнятих патернів для чіткого розмежування відповідальностей і спрощення розширення функціоналу. Було наведено опис інкапсуляції доступу до даних через репозиторії, обміну між шарами за допомогою DTO і маперів та створення конфігурованих компонентів через фабричні класи. Було виконано реалізацію фонового запуску завдань за розкладом і обробки подій життєвого циклу сутностей як підписників, а також застосовано безпечне шифрування, повторне використання HTTP-клієнта та уніфіковану обробку помилок у контролерах на основі патернів «Strategy», «Singleton» та «Template Method».

РОЗДІЛ 3. ІНТЕРФЕЙС ТА ПОРЯДОК РОБОТИ З ВЕБЗАСТОСУНКОМ

3.1 Структура інтерфейсу вебзастосунку для моніторингу психологічного здоров'я

Першочерговою сторінкою для користувача буде форма входу на сервіс. Користувач буде мати на вибір форми, які розрізняються тим, які саме ролі можуть у них входити та додатковим функціоналом. У Пацієнта він буде розширений можливістю відновити пароль, зареєструватися у вебзастосунку та входженням у систему за допомогою Google Auth. На рис. 3.1 та рис. 3.2 показані вигляди для даних форм.

Пацієнт Лікар

Увійти

Електронна пошта admin3@gmail.com

Пароль

Увійти

[Забули пароль?](#)

[Не зареєстровані? Рєєстрація](#)

[Увійти через Google](#)

 Увійти через Google

Рисунок 3.1 – Форма входу для Пацієнту

Рисунок 3.2 – Форма входу для Адміна та Лікаря

У випадку не сплати підписки буде показане вікно для обрання плану оплати та форми для заповнення банківськими даними. Після успішного проведення операції Пацієнту буде наданий доступ до його сторінки з туром для ознайомлення, де він буде мати змогу переглядати останні дані, які були вивантажені з Google Fit (рис. 3.3-3.4). Також на першій сторінці буде форма з введенням даних щодо свого настрою. Для точного вимірювання стану користувач також може налаштувати нагадування на імейл (форма на рис. 3.5).

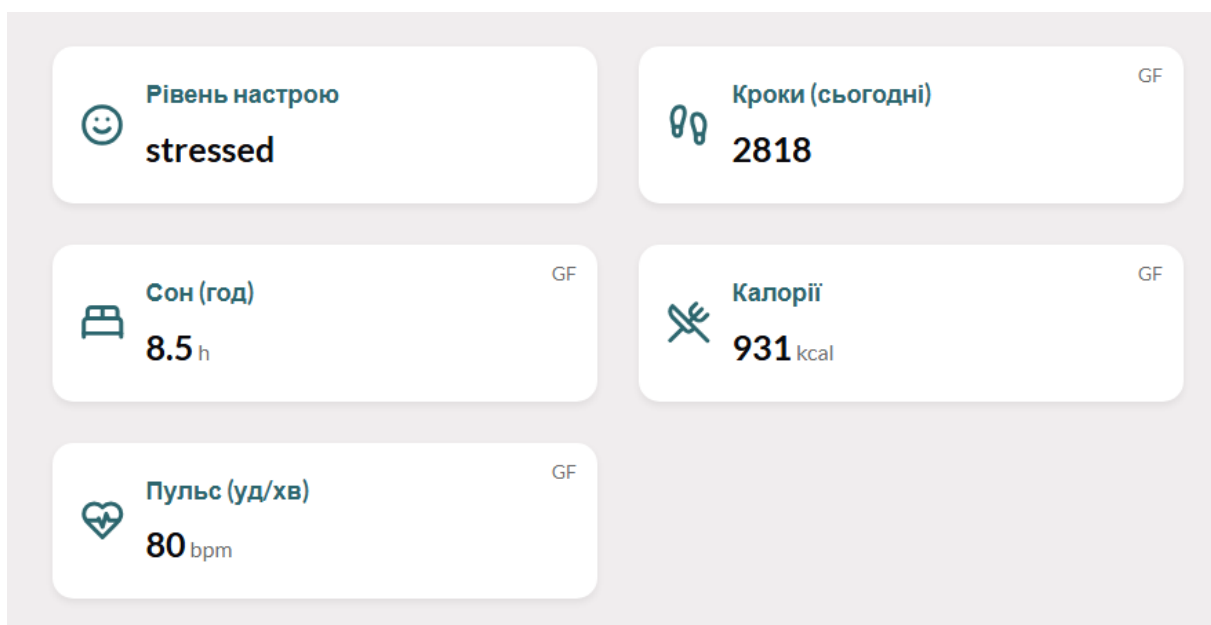


Рисунок 3.3 – Картки з статистикою за останній день

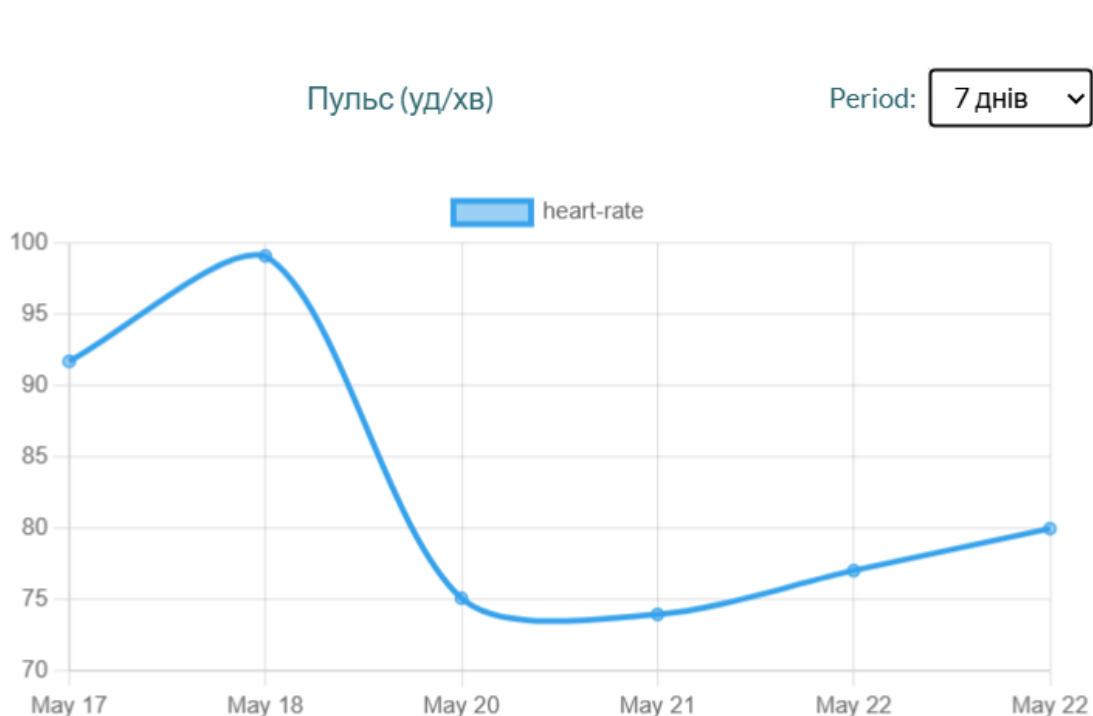


Рисунок 3.4 – Графік статистики здоров'я за обраний період

Оновити свій настрій

☐ Надіслати сповіщення електронною поштою

Рисунок 3.5 – Секція для виконання операцій з настроєм

Далі користувач може перейти на будь-яку з запропонованих сторінок: Профіль (початкова сторінка), Терапія, Повідомлення, Лікар, Налаштування. На

сторінці терапії представлений поточний календарний місяць, де користувач може обрати дати для перегляду запланованих сесій терапії або для створення нової. На рис. 3.6 показана форма для заповнення під час додавання сесії.

Рисунок 3.6 – Форма для редагування або створення терапії

На сторінці Повідомлення представлені всі чати з лікарями або пацієнтами з базовими функціями перегляду та надсилання повідомлень. Також додана сторінка, де Пацієнт може переглядати всіх зареєстрованих Лікарів та може додавати їх як своїх. На рис. 3.7 показано частина сторінки зі всіма Лікарями.

Рисунок 3.7 - Перелічень всіх лікарів на сторінці Пацієнта

Також у користувача з роллю Пацієнт буде можливість змінити власні дані, підключити Google Fit та переглянути власний план підписки на сторінці налаштувань (рис. 3.8).

Особиста інформація



Ім'я користувача

Ім'я

Прізвище

Дата народження

Стать

Рисунок 3.8 – Частина форми з особистими даними Пацієнта

Для користувачів з роллю Лікар при початковому заході буде показано сторінку, де вони повинні заповнити данні, які не були внесенні Адміном. Він матиме доступ до сторінки з профілем, де одним з функціоналом є перегляд даних про пацієнта (рис. 3.9).

Всього пацієнтів

Ім'я Прізвище	Останній сеанс	Сон (год)	Кроки (сьогодні)	Пульс (уд/хв)	Калорії	Рівень настр
Windows Ubuntu	2025-05-24T12:37:00	8.5 h	2818	79.97893524169922	931	stressed
Anastasiia Svirhun	2025-05-09T17:27:00	— h	—	—	—	—

Рисунок 3.9 – Таблиця зі всіма Пацієнтами Лікаря

Всі інші сторінки нічим не відрізняються у Лікаря та Пацієнта, проте користуватися ними повноцінно Лікарі зможуть тільки у випадку, коли будуть

мати з'єднаного пацієнта. Єдина відмінність полягає на сторінці Налаштувань, де Лікар може тільки редагувати власні дані.

На головній сторінці у Адміністратора показана статистика щодо використання сайту (рис. 3.10) та всі наявні записи щодо здоров'я Пацієнтів. На сторінках Користувачі, Пацієнти та Терапія показана інформація довідникового характеру, що була отримана з бази даних. Схоже присутнє і на сторінці Лікарі, проте функціонал там ще і додатково розширено за допомогою форми додавання користувача (рис. 3.11). Останньою сторінкою є Управління платіжками, де є можливість переглядати бухгалтерську статистику, всі оплати та індивідуальні дані.

Нові реєстрації за день

Нові реєстрації за день

Часовий відрізок	Кількість
2025-05-12	1

Нові реєстрації за тиждень

Часовий відрізок	Кількість
2025-W17	6
2025-W20	1

Нові реєстрації за місяць

Часовий відрізок	Кількість
2025-04	6
2025-05	1

Рисунок 3.10 – Статистика реєстрацій на сайті

Створити лікаря

Ім'я користувача

Електронна пошта

Пароль

Ім'я

Прізвище

Зберегти

Рисунок 3.11 – Форма додавання Лікарів

3.2 Статистика зборів та загальна статистика платформи

На сторінці кожного користувача присутня статистика щодо здоров'я. Доступ до неї представлений у різних формах через обмеження по ролі. Для прикладу, Пацієнт може бачити тільки власні дані, тоді як Лікар має доступ до статистики кожного Пацієнта, який підключений до них. На рис. 3.12 представлений вигляд статистики на сторінці Адміна.

Тренд калорій

ID пацієнта	Часовий відрізок	Значення
67	2025-05-23	931

Тренд кроків

ID пацієнта	Часовий відрізок	Значення
67	2025-05-23	2818

Рисунок 3.12 – Таблиця з даними калорій та кроків всіх пацієнтів

Окрім даної статистики на панелі адміністратора можна також дізнатися загальні значення про користування сайтом. Це представлено на рис. 3.13.



Рисунок 3.13 – Частина панелі адміністратора

На сторінці з управління платіжками можна побачити загальний прибуток, активні підписки та невдалі платежі у вигляді таких самих карток. Також при натисканні на одну з оплат буде відкрита додаткова панель з особистою статистикою вибраного користувача(рис. 3.14).

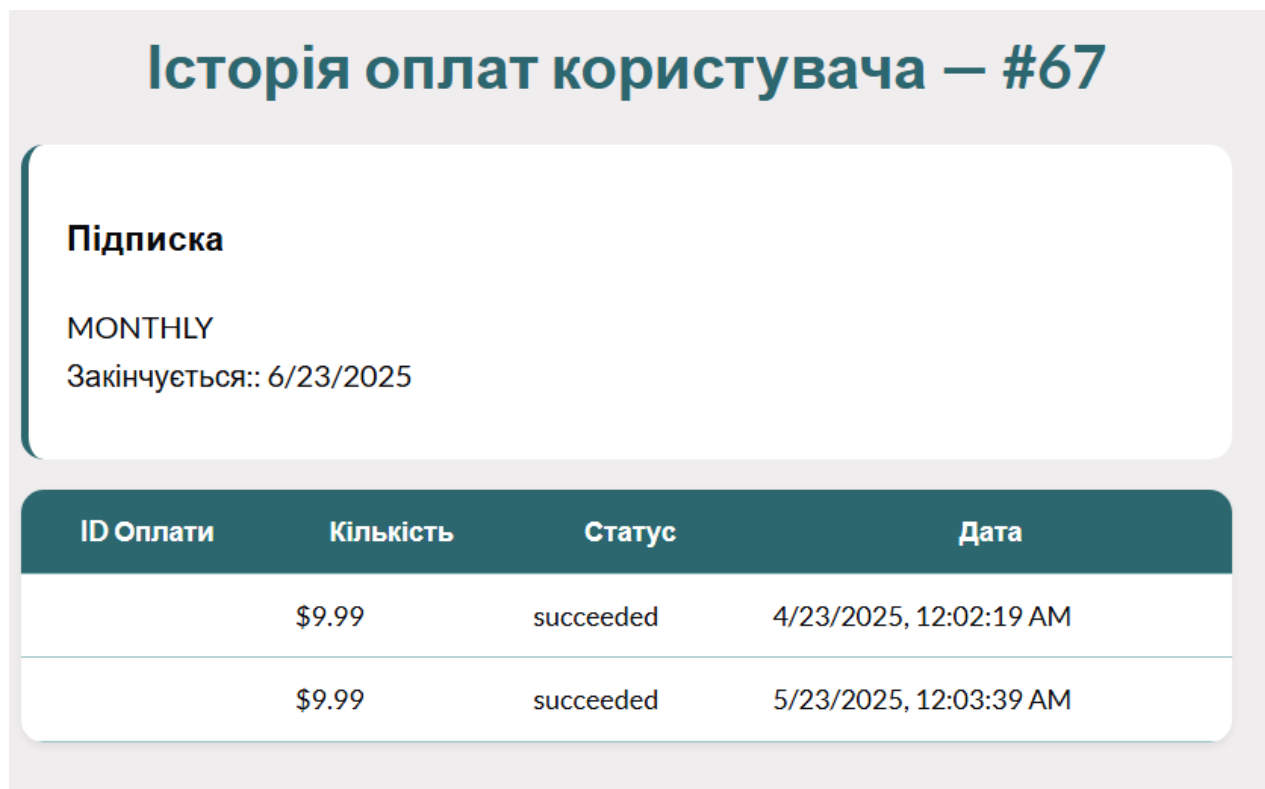


Рисунок 3.14 – Історія оплати користувача

3.3 Тестування платформи

Тестування платформи «Health Monitor» спрямоване на підтвердження коректності критичних бізнес-процесів — від автентифікації та створення записів настрою до інтеграцій зі Stripe і Google Fit. Для цього було сформовано набір наскрізних тест-кейсів, кожен з яких відтворює реальну користувацьку дію й перевіряє усі задіяні шари: клієнтський інтерфейс, REST-контролери, сервісну логіку, репозиторії та зовнішні API. Таблиця 3.1 фіксує опис сценарію, вхідні дані, очікувану та фактичну поведінку й статус проходження, що дозволяє швидко оцінити стабільність функціоналу та визначити області, які потребують додаткової уваги.

Таблиця 3.1

Test Case Description	Test Steps	Test Data	Expected Results	Actual Results	Pass/Fail
Авторизація зареєстрованого користувача через форму логіну	Відкрити сторінку /login, увести коректну адресу електронної пошти й пароль, натиснути Sign In, перевірити наявність JWT у local storage	Email: user@test.com Пароль: Valid_Pa55	Після підтвердження запиту POST /api/v1/auth/login сервер повертає JWT, React-Router автоматично робить редирект на «/dashboard»	Як і очікувалось	Pass
Авторизація нового користувача через Google OAuth2	Натиснути кнопку Sign in with Google, пройти кол-бек /login/oauth2/code/google, дозволити скоупи профілю і e-mail	Google Account test.oauth@gmail.com доданий у список тестерів	У таблиці users створюється новий запис, у oauth_credentials зберігається provider=google, бекенд повертає власний JWT та refresh-cookie, фронт переходить на /dashboard	Як і очікувалось	Pass
Створення запису настрою MoodTracking	На сторінці Mood Tracker натиснути Add Entry, у формі вибрати «anxious», у полі notes ввести «перед екзаменом», натиснути Save	mood=»anxious» notes=»перед екзаменом»	POST /api/v1/moods повертає 201, у базі mood_tracking з'являється рядок з mood=anxious, notes заповнено, patient_profile_id дорівнює чинному користувачеві	Як і очікувалось	Pass

Продовження табл. 3.1

Перерахунок аналітики після запису настрою	Після попереднього кейса викликати GET /api/v1/analytics/overview, перевірити середнє значення настрою	patient_id=текущ ий	JSON відповіді містить field averageMood, яке відображає значення «anxious» у середньому ваговому показнику за поточну дату	Як і очікувало сь	Pass
Генерація алерту при перевищенні порогу	В адміні сторінці Alert Rules встановити threshold = 10 для шкали тривоги, створити 3 послідовні записи «extreme_anxiety», перевірити пошту	три записи extreme_anxiety	NotificationService надсилає e-mail із темою «High anxiety alert», у таблиці audit_log фіксується запис action=ALERT_SENT	Як і очікувало сь	Pass
Редагування існуючого запису настрою	Клікнути іконку Edit біля останнього запису, змінити mood на «relieved», зберегти	mood=»relieved»	PUT /api/v1/moods/{id} повертає 200, mood_tracking .row оновлено, аналітика перерахована, компонент React оновив рядок списку без перезавантаження сторінки	Як і очікувало сь	Pass
Валідація обов'язково го поля mood	Відкрити форму Add Entry, залишити mood порожнім, натиснути Save	mood=» » notes=«test»	Фронт показує червону підказку «Mood is required», запит на бекенд не відправляється	Як і очікувало сь	Pass

Продовження табл. 3.1

Поштове нагадування без записів за добу	Змінити системний час на +24 h від останнього mood-entry, викликати вручну endpoint /api/v1/scheduler/run-reminders	user id поточного профілю	ReminderService створює запис у email_reminder, NotificationService надсилає лист «Please log your mood today», у email_reminder.sent встановлюється true	Як і очікувалося	Pass
Підключення Stripe Checkout	На сторінці Billing натиснути Subscribe, дочекатися перенаправлення на Stripe Checkout, увести тестову карту 4242 4242 4242 4242, CVV 123, завершити платіж	plan_price_id із Stripe Dashboard	Stripe повертає session_id, після оплати Stripe webhook /invoice.payment_succeeded викликає PaymentService, у таблиці payment_record створено запис amount=9.99, subscription продовжено	Як і очікувалося	Pass
Відміна платежу Stripe	На кроці Checkout натиснути Cancel, повернутися на /billing	N/A	Сесія Checkout переходить у статус canceled, у бекенді немає записів payment_record, фронт показує повідомлення «Payment canceled»	Як і очікувалося	Pass

Продовження табл. 3.1

Імпорт Google Fit heart-rate для тестового користувача	У консолі Google Fit запустити симулятор, створити dataset heart-rate = 80 bpm, дочекатися запуску GoogleFitSyncJob	користувач OAuth test.oauth@gmail.com	У таблиці heart_rate_tracking з'являється рядок bpm=80, timestamp відповідає dataset, patient_profile_id = тестовий акаунт	Як і очікувалося	Pass
Неможливість імпорту даних не-тестового акаунта	Виконати OAuth з продакшн-акаунтом, спробувати отримати heart rate	будь-який акаунт не в списку тестерів	Google API повертає 403, GoogleFitSyncJob логгує помилку, у базі не створюється записів, статус credentials → revoked	Як і очікувалося	Pass
Створення приватного повідомлення пацієнт–лікар	На сторінці Messages обрати лікаря Dr Smith, надіслати текст «Need advice»	senderId=patient recipientId=doctor	POST /api/v1/messages повертає 201, у таблиці message створено новий рядок, лікар бачить повідомлення у своєму інбоксі	Як і очікувалося	Pass
Відповідь лікаря на повідомлення	У лікарському кабінеті натиснути Reply, увести «Schedule session», Send	senderId=doctor recipientId=patient	message.row створений з відповідними id, NotificationService генерує e-mail «New secure message», пацієнт бачить його після F5	Як і очікувалося	Pass

Закінчення табл. 3.1

Створення профілю лікаря	У формі Doctor Profile ввести speciality=«psychologist» experienceYears=10 , натиснути Save	...	DoctorProfileRepository.save створює рядок speciality=psychologist, experienceYears=10, userId doc	Як і очікувалось	Pass
Оновлення профілю пацієнта	В особистих налаштуваннях змінити gender на «non-binary», birthDate = 1998-05-12	gender non-binary	PatientProfileRepository.update оновлює gender і birthDate, GET /api/v1/profile повертає нові значення	Як і очікувалось	Pass
Обробка токена відновлення пароля	Викликати endpoint /api/v1/auth/forgot з email існуючого користувача, перейти по посиланню з листа, задати новий пароль	email=user@test.com	PasswordResetToken створено, після POST /api/v1/auth/reset token стає використаним , пароль bcrypt-хешується, користувач входить з новим паролем	Як і очікувалось	Pass
Відтворення аудиту дій користувача	Зробити будь-яку CRUD-операцію настрою, отримати GET /api/v1/audit?userId=X	action=MoodCreated	AuditListener створює рядок action=MoodCreated, GET повертає запис зі штампом часу й ір, фронт таблицю виводить історію	Як і очікувалось	Pass

Висновки до третього розділу

У третьому розділі було описано структуру інтерфейсу веб-застосунку для моніторингу психологічного здоров'я: головне меню містить навігацію між

дашбордом із загальною статистикою, сторінкою введення та історії настрою, розділом моніторингу біометричних показників (сон, активність, ритм серця) і панеллю адміністрування з керуванням користувачами та налаштуваннями нагадувань. На дашборді реалізовано динамічні графіки, які відображають середнє значення настрою за обраний період, максимальні та мінімальні показники ритму серця, тривалість сну й кількість активностей. Було наведено статистику зборів даних, що включає загальну кількість збережених записів настрою, імпортованих з Google Fit біометричних метрик, відправлених електронних нагадувань та успішно оброблених платежів через Stripe, а також ключові метрики платформи — час безвідмовної роботи, середнє навантаження на API та кількість одночасних сесій користувачів.

Було наведено опис набору тест-кейсів, якими перевірено критичні сценарії роботи системи: аутентифікацію користувачів, створення й аналітику записів настрою, інтеграцію зі Stripe Checkout, імпорт даних Google Fit, роботу системи сповіщень і керування ролями (пацієнт, лікар, адміністратор). Було виконано верифікацію очікуваних та фактичних результатів для кожного кейсу, усі тести пройдено успішно. Таким чином поточна версія платформи повністю відповідає заявленим вимогам і готова до наступного етапу верифікації зовнішніх API.

ВИСНОВКИ

Під час виконання роботи над проектом було спроектовано та реалізовано вебзастосунок для моніторингу психологічного здоров'я із клієнт-серверною архітектурою, що включає React-клієнт і Spring Boot-сервер. Було проведено дослідження предметної області та аналіз існуючих рішень у сфері цифрового трекінгу настрою, біометрії та телемедицини. На основі отриманих даних сформовано набір функціональних і нефункціональних вимог, які лягли в основу дизайну системи.

Проектування платформи включало моделювання основних сценаріїв використання для пацієнтів, лікарів і адміністраторів, розробку структури бази даних у PostgreSQL, а також продумане розмежування шарів Controller → Service → Repository → Entity із використанням DTO для захисту доменної моделі. Було реалізовано інтеграції зі Stripe для обробки платежів і з Google Fit API для імпорту біометричних даних, а також налагоджено систему нагадувань і сповіщень через Spring Scheduler та JavaMailSender.

Було продемонстровано основні потоки роботи з платформою: реєстрацію та аутентифікацію (включно з JWT та OAuth2 через Google), введення і редагування записів настрою й біометрії, перегляд динамічних графіків аналітики, керування профілями користувачів, налаштування нагадувань і оплати підписок. Описано інтерфейс користувача з інтуїтивно зрозумілим меню, сторінками загальної статистики, історії та адмін-панеллю. Інтерфейс вирізняється простотою сприйняття та адаптивністю для різних носіїв мови.

Результатом роботи є готова до експлуатації платформа моніторингу психологічного здоров'я, що забезпечує автоматизацію збору й аналізу настрою, біометричних показників, оплати сервісу та управління сповіщеннями. Система пройшла комплексне тестування клієнтської та серверної частин і відповідає заявленим вимогам до функціональності, продуктивності й безпеки.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Свіргун А. В., Савіцький Р. С. Вплив технологій на надання психологічної допомоги : матеріали Міжнар. Наук.-практ. Конф., Житомир, 24–25 квіт. 2025 р. / А. В. Свіргун, Р. С. Савіцький. – Кам’янець-Подільський : ЖТУ, 2025. – 1 електрон. Опт. Диск (PDF). – Режим доступу: <https://conf.ztu.edu.ua/wp-content/uploads/2025/04/povnyj-tekst.pdf>
2. Svirhun A. V., Savitsky R. S. THE USAGE OF THE INTERNET OF THINGS FOR REMOTE MONITORING OF MENTAL HEALTH : подано до друку / A. V. Svirhun, R. S. Savitsky. – 2025.
3. PostgreSQL Global Development Group. PostgreSQL Documentation: [Електронний ресурс]. – Режим доступу: <https://www.postgresql.org/docs/>
4. Google Developers. Google APIs Explorer : [Електронний ресурс]. – Режим доступу: <https://developers.google.com/apis-explorer>
5. Stripe, Inc. Stripe API Reference : [Електронний ресурс]. – Режим доступу: <https://stripe.com/docs/api>
6. JSON Web Token. Introduction to JWT : [Електронний ресурс]. – Режим доступу: <https://jwt.io/introduction>
7. Postman, Inc. Postman API Client : [Електронний ресурс]. – Режим доступу: <https://www.postman.com/product/api-client/>
8. Moodbeam Ltd. Moodbeam Official Site : [Електронний ресурс]. – Режим доступу: <https://moodbeam.co.uk>
9. Headspace Health. Headspace : [Електронний ресурс]. – Режим доступу: <https://www.headspace.com> (дата звернення: 29.05.2025). – Назва з екрана.
10. ГО «Можна разом». Платформа «МОЖНА» : [Електронний ресурс]. – Режим доступу: <https://mozhnarazom.org>
11. Spokiy. Додаток «Spokiy» : [Електронний ресурс]. – Режим доступу: <https://spokiy.app>

12. Розмова. Сервіс «Rozmova» : [Електронний ресурс]. – Режим доступу: <https://rozmova.info>
13. Pivotal Software. Spring Boot Reference Documentation : [Електронний ресурс]. – Режим доступу: <https://docs.spring.io/spring-boot/docs/current/reference/html/>
14. Meta Platforms, Inc. React Documentation : [Електронний ресурс]. – Режим доступу: <https://react.dev>
15. Google LLC. Google Fit REST API : [Електронний ресурс]. – Режим доступу: <https://developers.google.com/fit/rest>
16. Spring Framework. Reference Documentation (Layers and DAO) : [Електронний ресурс]. – Режим доступу: <https://docs.spring.io/spring-framework/docs/current/reference/html/data-access.html>
17. Fowler M. Patterns of Enterprise Application Architecture. – Boston : Addison-Wesley, 2003. – 557 с. – doi:10.5555/579257
18. Spring Framework. Declarative Transaction Management : [Електронний ресурс]. – Режим доступу: <https://docs.spring.io/spring-framework/docs/current/reference/html/data-access.html#transaction>
19. Stonebraker M. The End of an Architectural Era (It's Time for a Complete Rewrite) / M. Stonebraker // SIGMOD Record. – 2005. – Т. 34, № 4. – С. 87–92. – doi:10.1145/1107499.1107500.
20. Vernon V. Implementing Domain-Driven Design. – Boston : Addison-Wesley Professional, 2013. – 352 с. – doi:10.5555/2901730.
21. Gamma E., Helm R., Johnson R., Vlissides J. Design Patterns : Elements of Reusable Object-Oriented Software. – Boston : Addison-Wesley, 1995. – 395 с. – doi:10.5555/186897
22. MapStruct. User Guide : [Електронний ресурс]. – Режим доступу: <https://mapstruct.org/documentation/>
23. Spring Framework. Task Scheduling and @Scheduled : [Електронний ресурс]. – Режим доступу: <https://docs.spring.io/spring-framework/docs/current/reference/html/integration.html#scheduling>

24. Spring Security. Method Security with @PreAuthorize : [Электронный ресурс]. – Режим доступа: <https://docs.spring.io/spring-security/reference/servlet/authorization/method-security.html>

25. Oracle. Java™ Tutorials – The Observer Pattern : [Электронный ресурс]. – Режим доступа: <https://docs.oracle.com/javase/tutorial/uiswing/events/observer.html>

26. Oracle. Java™ Tutorials – The Strategy Pattern : [Электронный ресурс]. – Режим доступа: <https://docs.oracle.com/javase/tutorial/extra/generics/strategy.html>

27. Oracle. Java™ Tutorials – The Singleton Pattern : [Электронный ресурс]. – Режим доступа: <https://docs.oracle.com/javase/tutorial/essential/concurrency/singleton.html>

28. Oracle. Java™ Tutorials – The Template Method Pattern : [Электронный ресурс]. – Режим доступа: <https://docs.oracle.com/javase/tutorial/rmi/design.html#template>

ДОДАТКИ

					ІІЗ.КР.Б – 121 – 25 – ІІЗ	
						58

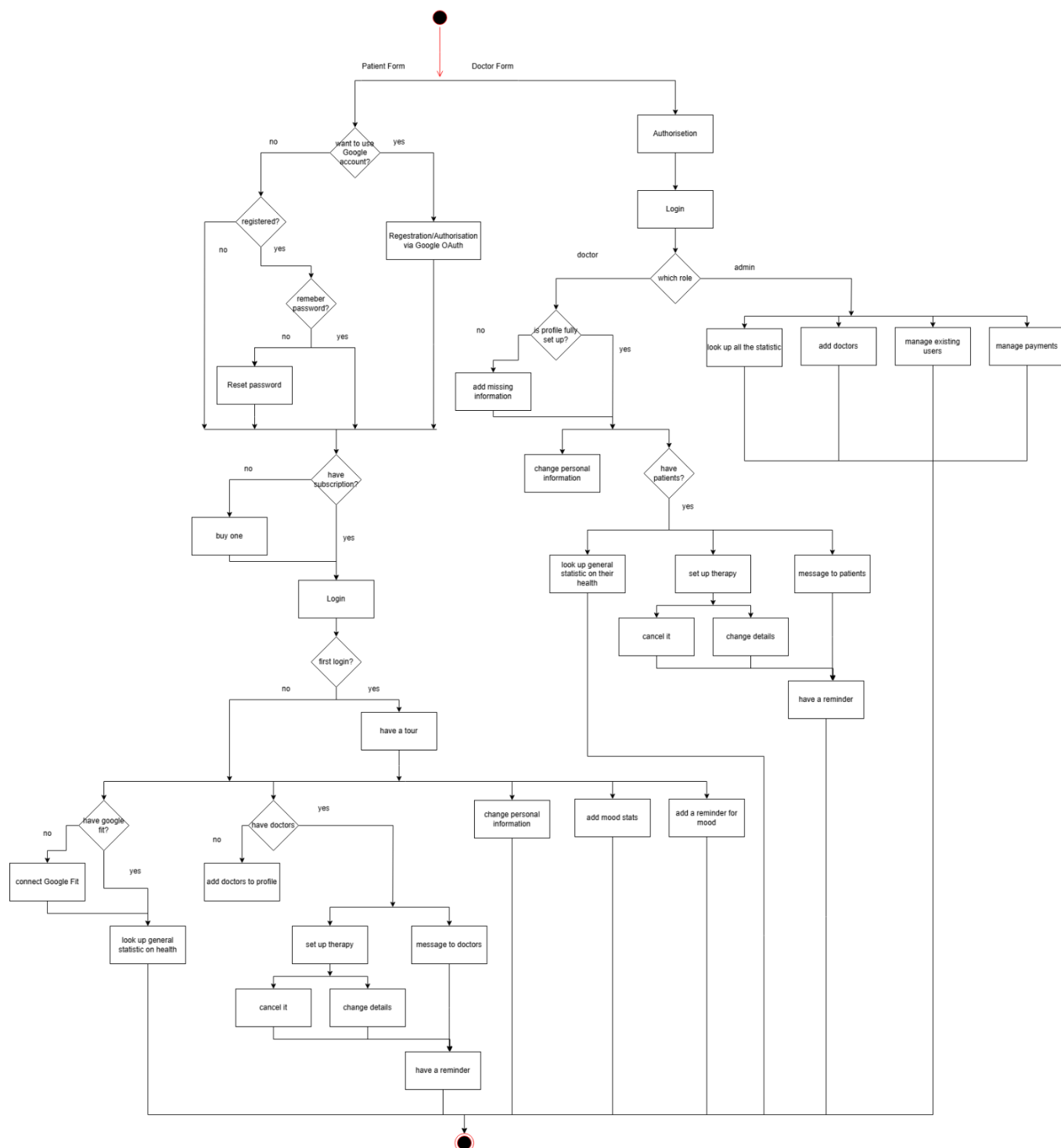


Рисунок А.1 - Діаграма активності платформи

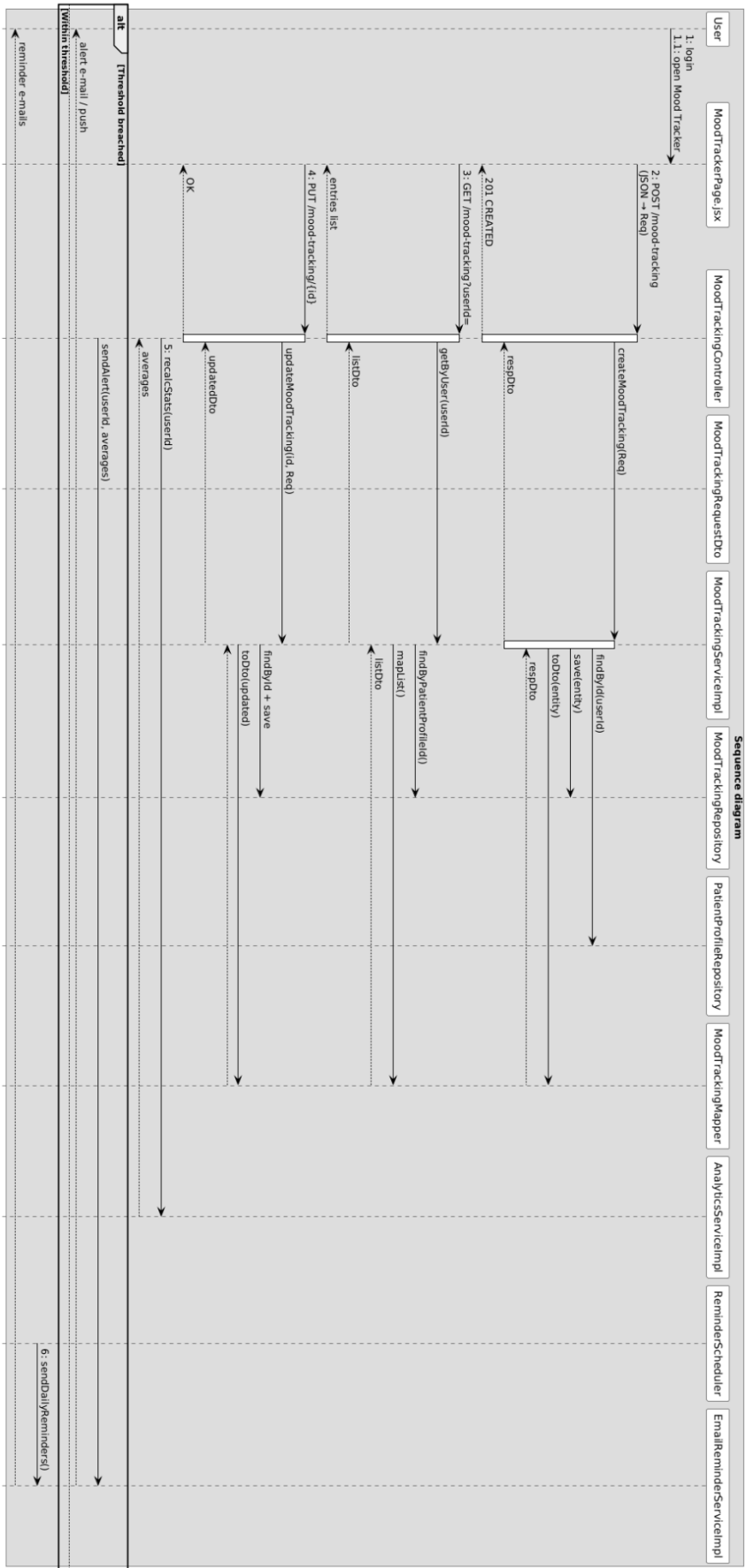


Рисунок Б.1 – Діаграма послідовності

PaymentController.java

```

package com.example.health_monitor.controller;

import com.example.health_monitor.dto.PaymentResponse;
import com.example.health_monitor.dto.SubscriptionDto;
import com.example.health_monitor.service.PaymentService;
import com.stripe.exception.StripeException;
import com.stripe.model.Charge;

import java.util.List;
import java.util.Map;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.ResponseEntity;
import org.springframework.security.access.prepost.PreAuthorize;
import org.springframework.web.bind.annotation.*;

@RestController
@RequestMapping("/api/payments")
public class PaymentController {

    @Autowired
    private PaymentService paymentService;

    @PostMapping("/charge")
    public ResponseEntity<?> charge(@RequestBody Map<String, Object> payload) {
        try {
            String token = (String) payload.get("token");
            Double amount = Double.valueOf(payload.get("amount").toString());
            Long userId = Long.valueOf(payload.get("userId").toString());
            Charge charge = paymentService.chargeCreditCard(token, amount, userId);

            PaymentResponse response = new PaymentResponse(
                charge.getId(),
                charge.getStatus(),
                charge.getBalanceTransaction(),
                charge.getDescription(),
                charge.getCurrency(),
                charge.getAmount(),
                charge.getCreated(),
                userId);

            return ResponseEntity.ok(response);
        } catch (StripeException e) {
            return ResponseEntity.badRequest().body(e.getMessage());
        }
    }
}

```

```

    @GetMapping("/subscription")
    public ResponseEntity<SubscriptionDto> getSubscription(@RequestParam Long
userId) {
        SubscriptionDto dto = paymentService.getSubscription(userId);
        return dto != null
            ? ResponseEntity.ok(dto)
            : ResponseEntity.noContent().build(); // 204
    }

    @GetMapping("/history")
    public ResponseEntity<List<PaymentResponse>> getHistory(@RequestParam Long
userId) {
        List<PaymentResponse> hist = paymentService.getPaymentHistory(userId);
        return ResponseEntity.ok(hist);
    }

    @GetMapping("/all")
    @PreAuthorize("hasRole('ADMIN')")
    public List<PaymentResponse> allCharges() {
        return paymentService.listAll();
    }

    @GetMapping("/stats")
    @PreAuthorize("hasRole('ADMIN')")
    public Map<String, Long> stats() {
        return paymentService.dashboardStats();
    }
}

```

PaymentResponse.java

```

package com.example.health_monitor.dto;

import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.NoArgsConstructor;

@Data
@NoArgsConstructor
@AllArgsConstructor
public class PaymentResponse {
    private String id;
    private String status;
    private String balanceTransaction;
    private String description;
    private String currency;
    private Long amount;
    private Long created;
    private Long userId;
}

```

PaymentRecordRepository.java

```
package com.example.health_monitor.repository;

import com.example.health_monitor.entity.PaymentRecord;
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.data.jpa.repository.Query;
import org.springframework.data.repository.query.Param;

import java.util.List;

public interface PaymentRecordRepository extends JpaRepository<PaymentRecord, Long>
{

    List<PaymentRecord> findByUserId(Long userId);

    @Query("SELECT COALESCE(SUM(p.amount),0) FROM PaymentRecord p WHERE p.status = :status")
    Long sumAmountByStatus(@Param("status") String status);

    @Query("SELECT COUNT(p) FROM PaymentRecord p WHERE p.status = :status")
    Long countByStatus(@Param("status") String status);
}
```

PaymentService.java

```
package com.example.health_monitor.service;

import com.example.health_monitor.dto.PaymentResponse;
import com.example.health_monitor.dto.SubscriptionDto;
import com.stripe.exception.StripeException;
import com.stripe.model.Charge;
import com.stripe.model.Token;

import java.util.List;
import java.util.Map;

public interface PaymentService {

    Token createToken(String cardNumber, String expMonth, String expYear, String cvc) throws StripeException;

    Charge chargeCreditCard(String token, double amount, Long userId) throws StripeException;

    SubscriptionDto getSubscription(Long userId);

    List<PaymentResponse> getPaymentHistory(Long userId);

    List<PaymentResponse> listAll();

    Map<String, Long> dashboardStats();
}
```

PaymentServiceImpl.java

```
package com.example.health_monitor.service.impl;

import com.example.health_monitor.dto.PaymentResponse;
import com.example.health_monitor.dto.SubscriptionDto;
import com.example.health_monitor.entity.PaymentRecord;
import com.example.health_monitor.entity.Subscription;
import com.example.health_monitor.entity.User;
import com.example.health_monitor.enums.Plan;
import com.example.health_monitor.repository.PaymentRecordRepository;
import com.example.health_monitor.repository.SubscriptionRepository;
import com.example.health_monitor.repository.UserRepository;
import com.example.health_monitor.service.PaymentService;
import com.stripe.Stripe;
import com.stripe.exception.StripeException;
import com.stripe.model.Charge;
import com.stripe.model.Token;
import com.stripe.param.ChargeCreateParams;
import com.stripe.param.TokenCreateParams;

import jakarta.annotation.PostConstruct;
import lombok.RequiredArgsConstructor;
import org.springframework.beans.factory.annotation.Value;

import org.springframework.stereotype.Service;

import java.time.LocalDate;
import java.time.temporal.ChronoUnit;
import java.util.HashMap;
import java.util.List;
import java.util.Map;
import java.util.Optional;
import java.util.stream.Collectors;

@Service
@RequiredArgsConstructor
public class PaymentServiceImpl implements PaymentService {

    private final SubscriptionRepository subscriptionRepo;
    private final PaymentRecordRepository paymentRepo;
    private final UserRepository userRepo;

    @Value("${stripe.secretKey}")
    private String stripeSecretKey;

    @PostConstruct
    public void init() {
        Stripe.apiKey = stripeSecretKey;
    }
}
```

```

    @Override
    public Token createToken(String cardNumber, String expMonth, String
expYear, String cvc)
        throws StripeException {

        TokenCreateParams.Card card = TokenCreateParams.Card.builder()
            .setNumber(cardNumber)
            .setExpMonth(expMonth)
            .setExpYear(expYear)
            .setCvc(cvc)
            .build();

        TokenCreateParams params = TokenCreateParams.builder()
            .setCard(card)
            .build();

        return Token.create(params);
    }

    @Override
    public Charge chargeCreditCard(String token, double amount, Long userId)
throws StripeException {
        ChargeCreateParams params = ChargeCreateParams.builder()
            .setAmount((Long) (amount))
            .setCurrency("usd")
            .setSource(token)
            .build();

        Charge charge = Charge.create(params);

        // record the payment
        User user = userRepo.findById(userId)
            .orElseThrow(() -> new RuntimeException("User not
found: " + userId));

        PaymentRecord rec = new PaymentRecord();
        rec.setUser(user);
        rec.setChargeId(charge.getId());
        rec.setStatus(charge.getStatus());
        rec.setBalanceTransaction(charge.getBalanceTransaction());
        rec.setDescription(charge.getDescription());
        rec.setCurrency(charge.getCurrency());
        rec.setAmount(charge.getAmount());
        rec.setCreated(charge.getCreated());
        paymentRepo.save(rec);

        Long cents = charge.getAmount();
        Plan plan;
        if (cents == 999L) {
            plan = Plan.MONTHLY;

```

```

        } else if (cents == 9999L) {
            plan = Plan.YEARLY;
        } else {
            throw new IllegalArgumentException("Unrecognized payment
amount (cents): " + cents);
        }

        LocalDate today = LocalDate.now();
        Optional<Subscription> maybe =
subscriptionRepo.findByUser_Id(userId);

        Subscription sub;
        if (maybe.isPresent()) {
            sub = maybe.get();
            LocalDate start = today.isBefore(sub.getExpiryDate())
                ? sub.getExpiryDate()
                : today;
            sub.setStartDate(start);
            sub.setExpiryDate(plan == Plan.MONTHLY
                ? start.plus(1, ChronoUnit.MONTHS)
                : start.plus(1, ChronoUnit.YEARS));
            sub.setPlan(plan);
        } else {
            sub = new Subscription();
            sub.setUser(user);
            sub.setPlan(plan);
            sub.setStartDate(today);
            sub.setExpiryDate(plan == Plan.MONTHLY
                ? today.plus(1, ChronoUnit.MONTHS)
                : today.plus(1, ChronoUnit.YEARS));
        }
        subscriptionRepo.save(sub);

        return charge;
    }

    @Override
    public SubscriptionDto getSubscription(Long userId) {
        return subscriptionRepo.findByUser_Id(userId)
            .map(s -> new SubscriptionDto(
                s.getPlan().name(),
                s.getStartDate(),
                s.getExpiryDate()))
            .orElse(null);
    }

    @Override
    public List<PaymentResponse> getPaymentHistory(Long userId) {
        return paymentRepo.findById(userId).stream()

```

```

        .map(r -> new PaymentResponse(
            r.getChargeId(),
            r.getStatus(),
            r.getBalanceTransaction(),
            r.getDescription(),
            r.getCurrency(),
            r.getAmount(),
            r.getCreated(),
            r.getUser().getId()))
        .collect(Collectors.toList());
    }

    @Override
    public Map<String, Long> dashboardStats() {
        Long total = paymentRepo.sumAmountByStatus("succeeded");
        Long failed = paymentRepo.countByStatus("failed");
        Long active =
subscriptionRepo.countActiveSubscriptions(LocalDate.now());
        Map<String, Long> map = new HashMap<>();
        map.put("total", total);
        map.put("failed", failed);
        map.put("activeSubs", active);
        return map;
    }

    @Override
    public List<PaymentResponse> listAll() {
        return paymentRepo.findAll()
            .stream()
            .map(p -> new PaymentResponse(
                p.getChargeId(),
                p.getStatus(),
                p.getBalanceTransaction(),
                p.getDescription(),
                p.getCurrency(),
                p.getAmount(),
                p.getCreated(),
                p.getUser().getId()))
            .collect(Collectors.toList());
    }
}

```

SecurityConfig.java

```

package com.example.health_monitor.config;

import com.example.health_monitor.security.JwtAuthFilter;
import com.example.health_monitor.security.OAuth2LoginSuccessHandler;
import com.example.health_monitor.service.CustomOAuth2UserService;
import com.example.health_monitor.service.CustomUserDetailsService;

import jakarta.servlet.http.HttpServletRequest;

import java.util.List;
import java.util.Locale;
import java.util.Properties;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.context.MessageSource;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.context.support.ReloadableResourceBundleMessageSource;
import org.springframework.core.env.Environment;
import org.springframework.http.HttpMethod;
import org.springframework.http.HttpStatus;
import org.springframework.mail.javamail.JavaMailSender;
import org.springframework.mail.javamail.JavaMailSenderImpl;
import org.springframework.security.config.Customizer;
import org.springframework.security.authentication.AuthenticationManager;
import
org.springframework.security.config.annotation.authentication.builders.Authenticati
onManagerBuilder;
import org.springframework.security.config.annotation.web.builders.HttpSecurity;
import org.springframework.security.config.http.SessionCreationPolicy;
import org.springframework.security.core.session.SessionRegistry;
import org.springframework.security.core.session.SessionRegistryImpl;
import org.springframework.security.crypto.password.PasswordEncoder;
import
org.springframework.security.oauth2.client.registration.ClientRegistrationRepositor
Y;
import
org.springframework.security.oauth2.client.web.DefaultOAuth2AuthorizationRequestRes
olver;
import
org.springframework.security.oauth2.client.web.OAuth2AuthorizationRequestResolver;
import
org.springframework.security.oauth2.core.endpoint.OAuth2AuthorizationRequest;
import org.springframework.security.oauth2.core.endpoint.OAuth2ParameterNames;
import org.springframework.security.web.SecurityFilterChain;
import org.springframework.web.cors.UrlBasedCorsConfigurationSource;
import org.springframework.web.servlet.LocaleResolver;

```

```

import org.springframework.web.servlet.i18n.AcceptHeaderLocaleResolver;
import org.springframework.web.cors.CorsConfiguration;
import org.springframework.web.cors.CorsConfigurationSource;
import org.springframework.security.web.authentication.HttpStatusEntryPoint;
import
org.springframework.security.web.authentication.UsernamePasswordAuthenticationFilter;

@Configuration
public class SecurityConfig {

    private final CustomUserDetailsService customUserDetailsService;
    private final CustomOAuth2UserService customOAuth2UserService;
    private final PasswordEncoder passwordEncoder;
    private final ClientRegistrationRepository clientRegistrationRepository;

    @Autowired
    private JwtAuthFilter jwtAuthFilter;

    @Autowired
    private OAuth2LoginSuccessHandler oAuth2LoginSuccessHandler;

    public SecurityConfig(CustomUserDetailsService customUserDetailsService,
        CustomOAuth2UserService customOAuth2UserService,
        PasswordEncoder passwordEncoder, ClientRegistrationRepository
clientRegistrationRepository) {
        this.customUserDetailsService = customUserDetailsService;
        this.customOAuth2UserService = customOAuth2UserService;
        this.passwordEncoder = passwordEncoder;
        this.clientRegistrationRepository = clientRegistrationRepository;
    }

    @SuppressWarnings("removal")
    @Bean
    public AuthenticationManager authenticationManager(HttpSecurity http) throws
Exception {
        return http.getSharedObject(AuthenticationManagerBuilder.class)
            .userDetailsService(customUserDetailsService)
            .passwordEncoder(passwordEncoder)
            .and()
            .build();
    }

    @Bean
    public SecurityFilterChain filterChain(HttpSecurity http) throws Exception {
        http
            .cors(Customizer.withDefaults())
            .csrf(csrf -> csrf.disable())
            .sessionManagement(sm ->
sm.sessionCreationPolicy(SessionCreationPolicy.STATELESS))

```

```

        .authorizeHttpRequests(auth -> auth
            .requestMatchers("/login**",
                "/oauth2/**",
                "/connect/**",
                "/api/auth/**",
                "/api/public/**",
                "/api/users/**")
            .permitAll()
            .requestMatchers("/api/admin/**").hasRole("ADMIN")
            .requestMatchers("/api/doctor/**").hasAnyRole("DOCTOR",
"ADMIN")
            .requestMatchers("/api/patients/**").hasAnyRole("PATIENT",
"ADMIN")
            .requestMatchers("/api/therapy-
sessions/**").hasAnyRole("PATIENT", "DOCTOR", "ADMIN")
            .requestMatchers("/api/sleep-
data/**").hasAnyRole("PATIENT", "ADMIN", "DOCTOR")
            .requestMatchers("/api/mood-tracking/**")
            .hasAnyRole("PATIENT", "ADMIN", "DOCTOR")
            .requestMatchers("/api/calories-tracking/**")
            .hasAnyRole("PATIENT", "ADMIN", "DOCTOR")
            .requestMatchers("/api/heart-rate", "/api/heart-rate/**")
            .hasAnyRole("PATIENT", "DOCTOR", "ADMIN")
            .anyRequest().authenticated())
        .exceptionHandling(ex -> ex
            .authenticationEntryPoint(new
HttpStatusEntryPoint(HttpStatus.UNAUTHORIZED)))
        .oauth2Login(oauth2 -> oauth2
            .authorizationEndpoint(
                a ->
a.authorizationRequestResolver(offlineGoogle(clientRegistrationRepository)))
            .userInfoEndpoint(userInfo ->
userInfo.userService(customOAuth2UserService))
            .successHandler(oAuth2LoginSuccessHandler))
        .addFilterBefore(jwtAuthFilter,
UsernamePasswordAuthenticationFilter.class)
        .httpBasic(Customizer.withDefaults());
    return http.build();
}

@Bean
public SessionRegistry sessionRegistry() {
    return new SessionRegistryImpl();
}

@Bean
public CorsConfigurationSource corsConfigurationSource() {
    var configuration = new CorsConfiguration();
    configuration.setAllowedOrigins(List.of("http://localhost:3000"));
    configuration.setAllowedMethods(List.of(

```

```

        HttpMethod.GET.name(),
        HttpMethod.POST.name(),
        HttpMethod.PUT.name(),
        HttpMethod.DELETE.name(),
        HttpMethod.OPTIONS.name()));
configuration.setAllowedHeaders(List.of("*"));
configuration.setAllowCredentials(true);

var source = new UrlBasedCorsConfigurationSource();
source.registerCorsConfiguration("/*", configuration);
return source;
}

@Bean
public JavaMailSender javaMailSender(Environment env) {
    JavaMailSenderImpl mailer = new JavaMailSenderImpl();
    mailer.setHost(env.getProperty("spring.mail.host"));
    mailer.setPort(env.getProperty("spring.mail.port", Integer.class));
    mailer.setUsername(env.getProperty("spring.mail.username"));
    mailer.setPassword(env.getProperty("spring.mail.password"));

    Properties props = mailer.getJavaMailProperties();
    props.put("mail.transport.protocol", "smtp");
    props.put("mail.smtp.auth", "true");
    props.put("mail.smtp.starttls.enable", "true");
    props.put("mail.smtp.starttls.required", "true");
    props.put("mail.debug", "true");

    return mailer;
}

```

```

@Bean
public OAuth2AuthorizationRequestResolver offlineGoogle(
    ClientRegistrationRepository repo) {

    DefaultOAuth2AuthorizationRequestResolver delegate = new
DefaultOAuth2AuthorizationRequestResolver(repo,
    "/oauth2/authorization");

    return new OAuth2AuthorizationRequestResolver() {

        @Override
        public OAuth2AuthorizationRequest resolve(HttpServletRequest request) {
            return customize(delegate.resolve(request));
        }

        @Override
        public OAuth2AuthorizationRequest resolve(HttpServletRequest request,
            String clientRegistrationId) {
            return customize(delegate.resolve(request, clientRegistrationId));
        }
    };
}

```

```

    }

    private OAuth2AuthorizationRequest customize(OAuth2AuthorizationRequest
original) {
        if (original == null) {
            return null;
        }

        String registrationId = (String) original.getAttribute(
            OAuth2ParameterNames.REGISTRATION_ID);

        if (!"google".equals(registrationId)) {
            return original;
        }

        return OAuth2AuthorizationRequest.from(original)
            .additionalParameters(params -> {
                params.put("access_type", "offline");
                params.put("prompt", "consent");
            })
            .build();
    }
};
}

@Bean
public MessageSource messageSource() {
    ReloadableResourceBundleMessageSource ms = new
ReloadableResourceBundleMessageSource();
    ms.setBasename("classpath:i18n/messages");
    ms.setDefaultEncoding("UTF-8");
    ms.setFallbackToSystemLocale(false);
    return ms;
}

@Bean
public LocaleResolver localeResolver() {
    AcceptHeaderLocaleResolver lr = new AcceptHeaderLocaleResolver();
    lr.setDefaultLocale(Locale.ENGLISH);
    return lr;
}
}

```

OAuth2LoginSuccessHandler.java

```

package com.example.health_monitor.security;

import com.example.health_monitor.entity.*;
import com.example.health_monitor.enums.Role;
import com.example.health_monitor.repository.*;
import com.example.health_monitor.service.DataFetchService;
import jakarta.servlet.ServletException;

```

```

import jakarta.servlet.http.HttpServletRequest;
import jakarta.servlet.http.HttpServletResponse;
import lombok.RequiredArgsConstructor;

import org.slf4j.LoggerFactory;
import org.springframework.beans.factory.annotation.Value;
import org.springframework.http.HttpHeaders;
import org.springframework.http.ResponseCookie;
import org.springframework.security.core.Authentication;
import org.springframework.security.oauth2.client.OAuth2AuthorizedClient;
import org.springframework.security.oauth2.client.OAuth2AuthorizedClientService;
import
org.springframework.security.oauth2.client.authentication.OAuth2AuthenticationToken
;
import org.springframework.security.oauth2.core.user.DefaultOAuth2User;
import
org.springframework.security.web.authentication.SimpleUrlAuthenticationSuccessHandl
er;
import org.springframework.stereotype.Component;
import org.springframework.web.client.HttpClientErrorException;

import java.io.IOException;
import java.time.Instant;

import org.slf4j.Logger;

@Component
@RequiredArgsConstructor
public class OAuth2LoginSuccessHandler extends
SimpleUrlAuthenticationSuccessHandler {

    private final UserRepository userRepository;
    private final JwtUtil jwtUtil;
    private final OAuth2AuthorizedClientService clientService;
    private final DataFetchService dataFetchService;
    private final OAuthCredentialsRepository credentialRepository;
    private final PatientProfileRepository patientProfileRepository;
    private static final Logger logger =
LoggerFactory.getLogger(OAuth2LoginSuccessHandler.class);

    @Value("${frontend.url:http://localhost:3000}")
    private String frontendUrl;

    @Override
    public void onAuthenticationSuccess(HttpServletRequest request,
        HttpServletResponse response,
        Authentication authentication) throws IOException, ServletException {

        if (authentication instanceof OAuth2AuthenticationToken oauthToken
            && "google".equals(oauthToken.getAuthorizedClientRegistrationId()))

```

```

{

    DefaultOAuth2User oauthUser = (DefaultOAuth2User)
authentication.getPrincipal();
    String email = oauthUser.getAttribute("email");

    User user = userRepository.findByEmail(email);
    if (user == null) {
        user = new User();
        user.setEmail(email);
        user.setUsername(email.split("@")[0]);
        user.setRole(Role.PATIENT);
        user.setEmailVerified(true);
        userRepository.save(user);
    }

    OAuth2AuthorizedClient client = clientService.loadAuthorizedClient(
        oauthToken.getAuthorizedClientRegistrationId(),
        oauthToken.getName());

    String accessToken = client.getAccessToken().getTokenValue();
    String refreshToken = client.getRefreshToken() != null
        ? client.getRefreshToken().getTokenValue()
        : null;
    Instant expiresAt = client.getAccessToken().getExpiresAt();
    String scopeString = String.join(" ",
        client.getAccessToken().getScopes());

    PatientProfile profile = user.getPatientProfile();
    if (profile == null) {
        profile = new PatientProfile(user);
        profile = patientProfileRepository.save(profile);
        user.setPatientProfile(profile);
    }

    OAuthCredentials creds = credentialRepository
        .findByIdAndVendor(user.getId(), "GOOGLE_FIT")
        .orElse(null);

    if (creds == null) {
        creds = new OAuthCredentials(user, "GOOGLE_FIT");
    }

    creds.setPatientProfile(profile);
    creds.setAccessToken(accessToken);
    creds.setExpiresAt(expiresAt);
    creds.setVendor("GOOGLE_FIT");
    creds.setScope(scopeString);
    if (refreshToken != null)
        creds.setRefreshToken(refreshToken);
}

```

```

        credentialRepository.save(creds);
        try {
            dataFetchService.fetchAndSaveForUser(creds);
        } catch (HttpClientErrorException e) {
            logger.warn("Google Fit fetch failed ({}): {}"; continuing login",
                e.getStatusCode(), e.getResponseBodyAsString());
        }
    }

    DefaultOAuth2User oauthUser = (DefaultOAuth2User)
authentication.getPrincipal();
    String email = oauthUser.getAttribute("email");

    User user = userRepository.findByEmail(email);
    if (user == null) {
        user = new User();
        user.setEmail(email);
        user.setUsername(email.split("@")[0]);
        user.setRole(Role.PATIENT);
        user.setEmailVerified(true);
        user = userRepository.save(user);
    }

    String accessToken = jwtUtil.generateAccessToken(user.getEmail(),
user.getId(), user.getRole());
    String refreshToken = jwtUtil.generateRefreshToken(user.getEmail(),
user.getId(), user.getRole());

    ResponseCookie accessCk = ResponseCookie.from("access", accessToken)
        .httpOnly(true)
        .secure(false)
        .path("/").maxAge(jwtUtil.getAccessExpirationMs() / 1000)
        .sameSite("Lax").build();
    response.addHeader(HttpHeaders.SET_COOKIE, accessCk.toString());

    ResponseCookie refreshCk = ResponseCookie.from("refreshToken",
refreshToken)
        .httpOnly(true)
        .secure(false)
        .path("/api/auth/refresh")
        .maxAge(jwtUtil.getRefreshExpirationMs() / 1000)
        .sameSite("Strict").build();
    response.addHeader(HttpHeaders.SET_COOKIE, refreshCk.toString());

    String redirectPath;
    if (user.getRole() == Role.PATIENT) {
        redirectPath = "/patient/profile";
    } else if (user.getRole() == Role.DOCTOR) {
        redirectPath = "/doctor/profile";
    }

```

```

    } else if (user.getRole() == Role.ADMIN) {
        redirectPath = "/admin/dashboard";
    } else {
        redirectPath = "/";
    }

    response.sendRedirect(frontendUrl + redirectPath);
}
}

```

OAuthCredentialsController.java

```

package com.example.health_monitor.controller;

import com.example.health_monitor.dto.OAuthCredentialsDto;
import com.example.health_monitor.service.OAuthCredentialsService;
import lombok.RequiredArgsConstructor;
import org.springframework.web.bind.annotation.*;

import java.util.List;

@RestController
@RequestMapping("/api/oauth-credentials")
@RequiredArgsConstructor
public class OAuthCredentialsController {

    private final OAuthCredentialsService oauthService;

    @GetMapping("/{id}")
    public OAuthCredentialsDto getOAuthCredentialsById(@PathVariable Long id) {
        return oauthService.getOAuthCredentialsById(id);
    }

    @PostMapping
    public OAuthCredentialsDto createOAuthCredentials(@RequestBody
    OAuthCredentialsDto dto) {
        return oauthService.createOAuthCredentials(dto);
    }

    @PutMapping("/{id}")
    public OAuthCredentialsDto updateOAuthCredentials(@PathVariable Long id,
    @RequestBody OAuthCredentialsDto dto) {
        return oauthService.updateOAuthCredentials(id, dto);
    }

    @DeleteMapping("/{id}")
    public void deleteOAuthCredentials(@PathVariable Long id) {
        oauthService.deleteOAuthCredentials(id);
    }

    @GetMapping
    public List<OAuthCredentialsDto> list(

```

```

        @RequestParam(required = false) Long userId,
        @RequestParam(required = false) String vendor) {
    if (userId != null && vendor != null) {
        return oauthService.findByUserAndVendor(userId, vendor);
    }
    return oauthService.getAllOAuthCredentials();
}
}

```

SleepDataController.java

```

package com.example.health_monitor.controller;

import com.example.health_monitor.dto.SleepDataDto;
import com.example.health_monitor.service.SleepDataService;
import lombok.RequiredArgsConstructor;

import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.*;

import java.time.LocalDateTime;
import java.util.Comparator;
import java.util.List;

@RestController
@RequestMapping("/api/sleep-data")
@RequiredArgsConstructor
public class SleepDataController {

    private final SleepDataService sleepService;

    @GetMapping("/{id}")
    public ResponseEntity<SleepDataDto> getSleepDataById(@PathVariable Long id) {
        return ResponseEntity.ok(sleepService.getSleepDataById(id));
    }

    @GetMapping
    public ResponseEntity<?> list(
        @RequestParam(required = false) Long userId,
        @RequestParam(defaultValue = "false") boolean latest) {

        List<SleepDataDto> data = (userId != null)
            ? sleepService.getByPatient(userId)
            : sleepService.getAllSleepData();

        if (latest) {
            return data.stream()
                .max(Comparator.comparing(
                    SleepDataDto::getEndTime,
                    Comparator.nullsLast(LocalDateTime::compareTo)))
                .map(ResponseEntity::ok)

```

```

        .orElseGet(() -> ResponseEntity.ok().build());
    }

    return ResponseEntity.ok(data);
}

@PostMapping
public ResponseEntity<SleepDataDto> createSleepData(@RequestBody SleepDataDto
dto) {
    return ResponseEntity.status(HttpStatus.CREATED)
        .body(sleepService.createSleepData(dto));
}

@PutMapping("/{id}")
public ResponseEntity<SleepDataDto> updateSleepData(@PathVariable Long id,
@RequestBody SleepDataDto dto) {
    return ResponseEntity.ok(sleepService.updateSleepData(id, dto));
}

@PostMapping("/quality")
public String gradeSleepData(@RequestBody SleepDataDto dto) {
    return sleepService.evaluateQuality(dto);
}

@DeleteMapping("/{id}")
public void deleteSleepData(@PathVariable Long id) {
    sleepService.deleteSleepData(id);
}
}

```

GoogleFitSleepResponse.java

```

package com.example.health_monitor.dto;

import com.fasterxml.jackson.annotation.JsonIgnoreProperties;
import lombok.Data;
import lombok.NoArgsConstructor;

import java.util.List;

@Data
@NoArgsConstructor
@JsonIgnoreProperties(ignoreUnknown = true)
public class GoogleFitSleepResponse {

    private List<Session> session;

    @Data
    @JsonIgnoreProperties(ignoreUnknown = true)
    public static class Session {
        private String id;
        private Long startTimeMillis;
    }
}

```

```

        private Long endTimeMillis;
        private int activityType;
    }
}

```

GoogleTokenResponse.java

```

package com.example.health_monitor.dto;

import com.fasterxml.jackson.annotation.JsonProperty;

public class GoogleTokenResponse {

    @JsonProperty("access_token")
    private String accessToken;

    @JsonProperty("refresh_token")
    private String refreshToken;

    @JsonProperty("expires_in")
    private Long expiresIn;

    @JsonProperty("scope")
    private String scope;

    @JsonProperty("token_type")
    private String tokenType;

    public GoogleTokenResponse() {
    }

    public String getAccessToken() {
        return accessToken;
    }

    public void setAccessToken(String accessToken) {
        this.accessToken = accessToken;
    }

    public String getRefreshToken() {
        return refreshToken;
    }

    public void setRefreshToken(String refreshToken) {
        this.refreshToken = refreshToken;
    }

    public Long getExpiresIn() {
        return expiresIn;
    }

    public void setExpiresIn(Long expiresIn) {
    }
}

```

```

        this.expiresIn = expiresIn;
    }

    public String getScope() {
        return scope;
    }

    public void setScope(String scope) {
        this.scope = scope;
    }

    public String getTokenType() {
        return tokenType;
    }

    public void setTokenType(String tokenType) {
        this.tokenType = tokenType;
    }
}

```

OAuthCredentialsDto.java

```
package com.example.health_monitor.dto;
```

```
import java.time.Instant;
```

```
import lombok.AllArgsConstructor;
```

```
import lombok.Data;
```

```
import lombok.NoArgsConstructor;
```

```
@Data
```

```
@NoArgsConstructor
```

```
@AllArgsConstructor
```

```
public class OAuthCredentialsDto {
```

```
    private Long id;
```

```
    private String vendor;
```

```
    private String accessToken;
```

```
    private String refreshToken;
```

```
    private Instant expiresAt;
```

```
}
```

SleepDataDto.java

```
package com.example.health_monitor.dto;
```

```
import lombok.AllArgsConstructor;
```

```
import lombok.Data;
```

```
import lombok.NoArgsConstructor;
```

```
import java.time.LocalDate;
```

```
import java.time.LocalDateTime;
```

```
@Data
```

```
@NoArgsConstructor
```

					ИПЗ.КР.Б – 121 – 25 – ИПЗ	
						80

```

@ALLArgsConstructor
public class SleepDataDto {
    private Long id;
    private LocalDateTime startTime;
    private LocalDateTime endTime;
    private Double durationHours;
    private String quality;
    private LocalDate sleepDate;
    private Long patientProfileId;
}

OAuthCredentials.java

package com.example.health_monitor.entity;

import jakarta.persistence.*;
import lombok.ALLArgsConstructor;
import lombok.Data;
import lombok.NoArgsConstructor;

import java.time.Instant;

import com.fasterxml.jackson.annotation.JsonIgnore;

@Data
@ALLArgsConstructor
@NoArgsConstructor
@Entity
@Table(name = "oauth_credentials")
public class OAuthCredentials {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    @Column(name = "user_id", nullable = false)
    private Long userId;

    @Column(name = "vendor", nullable = false)
    private String vendor;

    @Column(name = "access_token", nullable = false)
    private String accessToken;

    @Column(name = "refresh_token")
    private String refreshToken;

    @Column(name = "expires_at")
    private Instant expiresAt;

    @OneToOne(fetch = FetchType.LAZY)
    @JoinColumn(name = "patient_profile_id")

```

```

    @JsonIgnore
    private PatientProfile patientProfile;

    @Column(length = 512)
    private String scope;

    public OAuthCredentials(User user, String vendor) {
        this.userId = user.getId();
        this.vendor = vendor;
    }
}

```

SleepData.java

```

package com.example.health_monitor.entity;

import jakarta.persistence.*;
import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.NoArgsConstructor;

import java.time.LocalDate;
import java.time.LocalDateTime;

import com.fasterxml.jackson.annotation.JsonIgnore;

@Data
@NoArgsConstructor
@AllArgsConstructor
@Entity
@Table(name = "sleep_data", uniqueConstraints = @UniqueConstraint(name =
"sleep_patient_start", columnNames = {
    "patient_profile_id", "start_time" }))
public class SleepData {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    @ManyToOne(fetch = FetchType.LAZY)
    @JoinColumn(name = "patient_profile_id", nullable = false)
    @JsonIgnore
    private PatientProfile patientProfile;

    @Column(name = "start_time", nullable = false)
    private LocalDateTime startTime;

    @Column(name = "end_time", nullable = false)
    private LocalDateTime endTime;

    @Column(name = "quality")

```

```

    private String quality;

    @Column(name = "duration_hours")
    private Double durationHours;

    @Column(name = "sleep_date")
    private LocalDate sleepDate;
}

```

OAuthCredentialsRepository.java

```

package com.example.health_monitor.repository;

import com.example.health_monitor.entity.OAuthCredentials;
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.stereotype.Repository;

import java.util.Optional;

@Repository
public interface OAuthCredentialsRepository extends JpaRepository<OAuthCredentials,
    Long> {

    Optional<OAuthCredentials> findByPatientProfileId(Long patientProfileId);

    Optional<OAuthCredentials> findByUserIdAndVendor(Long userId, String vendor);
}

```

SleepDataRepository.java

```

package com.example.health_monitor.repository;

import com.example.health_monitor.entity.SleepData;
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.data.jpa.repository.Query;
import org.springframework.data.repository.query.Param;
import org.springframework.stereotype.Repository;

import java.time.LocalDate;
import java.util.List;

@Repository
public interface SleepDataRepository extends JpaRepository<SleepData, Long> {

    List<SleepData> findByPatientProfileId(Long patientProfileId);

    @Query(value = "SELECT patient_profile_id           AS patient_id, " +
        "          DATE(start_time)                   AS period, " +
        "          AVG(duration_hours)                 AS avg_sleep " +
        "FROM      sleep_data " +
        "WHERE     start_time >= :since " +
        "GROUP    BY 1, 2 " +
        "ORDER    BY 1, 2", nativeQuery = true)

```

```

    List<Object[]> avgDailySleep(@Param("since") LocalDate since);
}

```

OAuthCredentialsServiceImpl.java

```

package com.example.health_monitor.service.impl;

import com.example.health_monitor.dto.OAuthCredentialsDto;
import com.example.health_monitor.entity.OAuthCredentials;
import com.example.health_monitor.mapper.OAuthCredentialsMapper;
import com.example.health_monitor.repository.OAuthCredentialsRepository;
import com.example.health_monitor.service.OAuthCredentialsService;
import lombok.RequiredArgsConstructor;
import org.springframework.stereotype.Service;
import org.springframework.transaction.annotation.Transactional;

import java.util.List;
import java.util.stream.Collectors;

@Service
@RequiredArgsConstructor
@Transactional
public class OAuthCredentialsServiceImpl implements OAuthCredentialsService {

    private final OAuthCredentialsRepository repository;
    private final OAuthCredentialsMapper mapper;

    @Override
    public List<OAuthCredentialsDto> getAllOAuthCredentials() {
        return repository.findAll().stream()
            .map(mapper::toDto)
            .collect(Collectors.toList());
    }

    @Override
    public OAuthCredentialsDto getOAuthCredentialsById(Long id) {
        return mapper.toDto(repository.findById(id)
            .orElseThrow(() -> new RuntimeException("OAuth Credentials not
found"))));
    }

    @Override
    public OAuthCredentialsDto createOAuthCredentials(OAuthCredentialsDto
oauthCredentialsDto) {
        OAuthCredentials entity = mapper.toEntity(oauthCredentialsDto);
        return mapper.toDto(repository.save(entity));
    }

    @Override
    public List<OAuthCredentialsDto> findByUserAndVendor(Long userId, String
vendor) {
        return repository

```

```

        .findByUserIdAndVendor(userId, vendor)
        .stream()
        .map(mapper::toDto)
        .collect(Collectors.toList());
    }

    @Override
    public OAuthCredentialsDto updateOAuthCredentials(Long id, OAuthCredentialsDto
oauthCredentialsDto) {
        OAuthCredentials entity = repository.findById(id)
            .orElseThrow(() -> new RuntimeException("OAuth Credentials not
found"));

        entity.setAccessToken(oauthCredentialsDto.getAccessToken());
        entity.setRefreshToken(oauthCredentialsDto.getRefreshToken());
        entity.setVendor(oauthCredentialsDto.getVendor());
        entity.setExpiresAt(oauthCredentialsDto.getExpiresAt());

        return mapper.toDto(repository.save(entity));
    }

    @Override
    public void deleteOAuthCredentials(Long id) {
        repository.deleteById(id);
    }
}

```

SleepDataServiceImpl.java

```

package com.example.health_monitor.service.impl;

import com.example.health_monitor.dto.SleepDataDto;
import com.example.health_monitor.entity.*;
import com.example.health_monitor.mapper.SleepDataMapper;
import com.example.health_monitor.repository.*;
import com.example.health_monitor.service.SleepDataService;
import com.example.health_monitor.util.SleepQualityCalculator;
import lombok.RequiredArgsConstructor;
import org.springframework.stereotype.Service;
import org.springframework.transaction.annotation.Transactional;

import java.util.List;

@Service
@RequiredArgsConstructor
@Transactional
public class SleepDataServiceImpl implements SleepDataService {

    private final SleepDataRepository repository;
    private final PatientProfileRepository patientRepo;
    private final SleepDataMapper mapper;
}

```

```

@Override
@Transactional(readOnly = true)
public List<SleepDataDto> getAllSleepData() {
    return repository.findAll().stream().map(mapper::toDto).toList();
}

@Override
@Transactional(readOnly = true)
public SleepDataDto getSleepDataById(Long id) {
    return mapper.toDto(repository.findById(id)
        .orElseThrow(() -> new IllegalArgumentException("Sleep data not
found"))));
}

@Override
@Transactional(readOnly = true)
public List<SleepDataDto> getByPatient(Long patientProfileId) {
    return repository
        .findByPatientProfileId(patientProfileId)
        .stream()
        .map(mapper::toDto)
        .toList();
}

@Override
public SleepDataDto createSleepData(SleepDataDto dto) {

    PatientProfile pp = patientRepo.findById(dto.getPatientProfileId())
        .orElseThrow(() -> new IllegalArgumentException(
            "Patient not found id=" + dto.getPatientProfileId()));

    SleepData entity = mapper.toEntity(dto);
    entity.setPatientProfile(pp);

    double hours = SleepQualityCalculator.hours(dto.getStartTime(),
dto.getEndTime());
    entity.setDurationHours(hours);
    entity.setQuality(SleepQualityCalculator.grade(hours));

    return mapper.toDto(repository.save(entity));
}

@Override
public SleepDataDto updateSleepData(Long id, SleepDataDto dto) {

    SleepData entity = repository.findById(id)
        .orElseThrow(() -> new IllegalArgumentException("Sleep data not
found")));

    entity.setStartTime(dto.getStartTime());

```

```

        entity.setEndTime(dto.getEndTime());

        double hours = SleepQualityCalculator.hours(dto.getStartTime(),
dto.getEndTime());
        entity.setDurationHours(hours);
        entity.setQuality(SleepQualityCalculator.grade(hours));

        entity.setSleepDate(dto.getSleepDate() != null ? dto.getSleepDate() :
dto.getStartTime().toLocalDate());

        return mapper.toDto(entity);
    }

    @Override
    public void deleteSleepData(Long id) {
        repository.deleteById(id);
    }

    @Override
    @Transactional(readOnly = true)
    public String evaluateQuality(SleepDataDto dto) {
        return SleepQualityCalculator.grade(
            SleepQualityCalculator.hours(dto.getStartTime(),
dto.getEndTime()));
    }
}

```

CustomOAuth2UserService.java

```

package com.example.health_monitor.service;

import com.example.health_monitor.dto.PatientProfileDto;
import com.example.health_monitor.entity.User;
import com.example.health_monitor.enums.Role;
import com.example.health_monitor.repository.UserRepository;
import lombok.RequiredArgsConstructor;
import org.springframework.security.core.authority.SimpleGrantedAuthority;
import org.springframework.security.core.userdetails.UsernameNotFoundException;
import org.springframework.security.crypto.password.PasswordEncoder;
import org.springframework.security.oauth2.client.userinfo.*;
import org.springframework.security.oauth2.core.*;
import org.springframework.security.oauth2.core.user.*;
import org.springframework.stereotype.Service;
import org.springframework.transaction.annotation.Transactional;

import java.util.Collections;
import java.util.UUID;

@Service
@RequiredArgsConstructor
@Transactional
public class CustomOAuth2UserService

```

```

implements OAuth2UserService<OAuth2UserRequest, OAuth2User> {

private final UserRepository userRepository;
private final PasswordEncoder passwordEncoder;
private final PatientProfileService patientProfileService;

@Override
public OAuth2User loadUser(OAuth2UserRequest userRequest)
    throws OAuth2AuthenticationException {

    OAuth2UserService<OAuth2UserRequest, OAuth2User> delegate = new
DefaultOAuth2UserService();
    OAuth2User oAuth2User = delegate.loadUser(userRequest);

    String email = oAuth2User.getAttribute("email");
    if (email == null) {
        throw new UsernameNotFoundException(
            "Email not found from Google OAuth2 provider");
    }

    User user = userRepository.findByEmail(email);
    if (user == null) {
        user = new User();
        user.setEmail(email);

        String rawPwd = UUID.randomUUID().toString();
        user.setPassword(passwordEncoder.encode(rawPwd));

        user.setRole(Role.PATIENT);

        user.setFirstName(oAuth2User.getAttribute("given_name"));
        user.setLastName(oAuth2User.getAttribute("family_name"));

        String baseUsername = email.substring(0, email.indexOf('@'));
        user.setUsername(baseUsername);

        user.setEmailVerified(true);

        user = userRepository.save(user);

        PatientProfileDto p = new PatientProfileDto();
        p.setUserId(user.getId());

        patientProfileService.createPatientProfile(p);
    }

    SimpleGrantedAuthority authority = new SimpleGrantedAuthority("ROLE_" +
user.getRole());

    return new DefaultOAuth2User(

```

```

        Collections.singleton(authority),
        oAuth2User.getAttributes(),
        "email");
    }
}

```

DataFetchService.java

```

package com.example.health_monitor.service;

import com.example.health_monitor.dto.*;
import com.example.health_monitor.entity.ActivityTracking;
import com.example.health_monitor.entity.CaloriesTracking;
import com.example.health_monitor.entity.HeartRateTracking;
import com.example.health_monitor.entity.OAuthCredentials;
import com.example.health_monitor.entity.SleepData;
import com.example.health_monitor.repository.ActivityTrackingRepository;
import com.example.health_monitor.repository.CaloriesTrackingRepository;
import com.example.health_monitor.repository.HeartRateTrackingRepository;
import com.example.health_monitor.repository.OAuthCredentialsRepository;
import com.example.health_monitor.repository.SleepDataRepository;
import com.example.health_monitor.util.SleepQualityCalculator;

import lombok.extern.slf4j.Slf4j;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.beans.factory.annotation.Value;
import org.springframework.dao.DataIntegrityViolationException;
import org.springframework.http.*;
import org.springframework.scheduling.annotation.Scheduled;
import org.springframework.stereotype.Service;
import org.springframework.util.LinkedMultiValueMap;
import org.springframework.util.MultiValueMap;
import org.springframework.web.client.RestTemplate;

import java.time.Instant;
import java.time.LocalDate;
import java.time.LocalDateTime;
import java.time.ZoneOffset;
import java.time.temporal.ChronoUnit;
import java.util.List;
import java.util.Map;

@Slf4j
@Service
public class DataFetchService {

    private static final String DT_KCAL = "com.google.calories.expended";

    @Autowired
    private ActivityTrackingRepository activityRepository;

```

```

@Autowired
private HeartRateTrackingRepository heartRepository;

@Autowired
private OAuthCredentialsRepository credentialRepository;

@Autowired
private SleepDataRepository sleepDataRepository;

@Autowired
private CaloriesTrackingRepository caloriesTrackingRepository;

@Autowired
private RestTemplate restTemplate;

@Value("${spring.security.oauth2.client.registration.google.client-id}")
private String googleFitClientId;

@Value("${spring.security.oauth2.client.registration.google.client-secret}")
private String googleFitClientSecret;

/**
 * Викликається кожен день о першій ночі
 */
@Scheduled(cron = "0 0 1 * * ?")
public void fetchAllUsersData() {
    List<OAuthCredentials> allCreds = credentialRepository.findAll();
    for (OAuthCredentials creds : allCreds) {
        fetchUserData(creds);
    }
}

private void fetchUserData(OAuthCredentials creds) {
    if (!"GOOGLE_FIT".equalsIgnoreCase(creds.getVendor())) {
        return;
    }

    if (isTokenExpired(creds)) {
        refreshToken(creds);
    }

    Long endMillis = System.currentTimeMillis();
    Long startMillis = endMillis - (24 * 60 * 60 * 1000);

    fetchSleepDataFromGoogleFit(creds, startMillis, endMillis);

    fetchCaloriesDataFromGoogleFit(creds, startMillis, endMillis);

    fetchActivityDataFromGoogleFit(creds, startMillis, endMillis);
}

```

```

        fetchHeartRateDataFromGoogleFit(creds, startMillis, endMillis);
    }

    private void fetchCaloriesDataFromGoogleFit(OAuthCredentials creds, Long
startMillis, Long endMillis) {
        Map<String, Object> payload = Map.of(
            "aggregateBy", List.of(Map.of("dataTypeName", DT_KCAL)),
            "bucketByTime", Map.of("durationMillis", 86_400_000L),
            "startTimeMillis", startMillis,
            "endTimeMillis", endMillis);

        GoogleFitCaloriesResponse rs = restTemplate.postForEntity(
            "https://www.googleapis.com/fitness/v1/users/me/dataset:aggregate",
            new HttpEntity<>(payload, bearerJson(creds.getAccessToken())),
            GoogleFitCaloriesResponse.class).getBody();

        if (rs == null || rs.getBucket() == null)
            return;

        rs.getBucket().forEach(b -> {

            Integer kcal = b.getDataset().stream()
                .flatMap(d -> d.getPoint().stream())
                .flatMap(p -> p.getValue().stream())
                .map(v -> v.getFpVal() == null ? null :
v.getFpVal().intValue())
                .filter(v -> v != null)
                .findFirst().orElse(null);

            if (kcal == null)
                return;

            LocalDate day =
Instant.ofEpochMilli(Long.parseLong(b.getStartTimeMillis()))
                .atZone(ZoneOffset.UTC).toLocalDate();

            CaloriesTracking ct = new CaloriesTracking();
            ct.setPatientProfile(creds.getPatientProfile());
            ct.setDate(day);
            ct.setKcal(kcal);

            try {
                caloriesTrackingRepository.save(ct);
            } catch (DataIntegrityViolationException dup) {
                log.debug("Duplicate kcal ignored: {}", dup.getMessage());
            }
        });
    }
}

```

```

private void fetchSleepDataFromGoogleFit(OAuthCredentials creds,
    Long startTimeMillis,
    Long endTimeMillis) {
    String url = "https://www.googleapis.com/fitness/v1/users/me/sessions"
        + "?startTime=" + toIso8601(startTimeMillis)
        + "&endTime=" + toIso8601(endTimeMillis);

    HttpHeaders headers = buildBearerHeaders(creds.getAccessToken());
    HttpEntity<Void> entity = new HttpEntity<>(headers);

    ResponseEntity<GoogleFitSleepResponse> response =
restTemplate.exchange(url, HttpMethod.GET, entity,
    GoogleFitSleepResponse.class);

    GoogleFitSleepResponse body = response.getBody();
    if (body == null || body.getSession() == null) {
        return;
    }

    body.getSession().forEach(session -> {
        if (session.getActivityType() == 72) {
            LocalDateTime startLdt =
toLocalDateTime(session.getStartTimeMillis());
            LocalDateTime endLdt = toLocalDateTime(session.getEndTimeMillis());

            SleepData sd = new SleepData();

            sd.setPatientProfile(creds.getPatientProfile());
            sd.setStartTime(startLdt);
            sd.setEndTime(endLdt);
            double hrs = ChronoUnit.SECONDS.between(startLdt, endLdt) / 3600.0;
            sd.setDurationHours(hrs);
            sd.setQuality(SleepQualityCalculator.grade(hrs));
            sd.setSleepDate(startLdt.toLocalDate());

            try {
                sleepDataRepository.save(sd);
            } catch (DataIntegrityViolationException dup) {
                log.debug("Duplicate sleep ignored: {}", dup.getMessage());
            }
        }
    });
}

private void fetchActivityDataFromGoogleFit(OAuthCredentials creds,
    Long startMillis, Long endMillis) {

    String url =
"https://www.googleapis.com/fitness/v1/users/me/dataset:aggregate";
    HttpHeaders headers = buildBearerHeaders(creds.getAccessToken());

```

```

        headers.setContentType(MediaType.APPLICATION_JSON);

        Map<String, Object> payload = Map.of(
            "aggregateBy", List.of(Map.of("dataTypeName",
                "com.google.step_count.delta")),
            "bucketByTime", Map.of("durationMillis", 86_400_000),
            "startTimeMillis", startMillis,
            "endTimeMillis", endMillis);

        GoogleFitActivityResponse rs = restTemplate.postForEntity(
            url, new HttpEntity<>(payload, headers),
            GoogleFitActivityResponse.class).getBody();

        if (rs == null || rs.getBucket() == null)
            return;

        rs.getBucket().forEach(b -> b.getDataset().forEach(ds ->
            ds.getPoint().forEach(pt -> {

                ActivityTracking entity = new ActivityTracking();
                entity.setPatientProfile(creds.getPatientProfile());
                entity.setStartTime(instant(pt.getStartTimeNanos()).atZone(ZoneOffset.UTC).toLocalDateTime());
                entity.setEndTime(instant(pt.getEndTimeNanos()).atZone(ZoneOffset.UTC).toLocalDateTime());
                entity.setActivityType("WALKING");
                entity.setSteps(pt.getValue().get(0).getIntVal());
                entity.setDurationMinutes(null);

                try {
                    activityRepository.save(entity);
                } catch (DataIntegrityViolationException dupe) {
                    log.debug("Duplicate activity ignored: {}", dupe.getMessage());
                }
            }
        ));

        private static Instant instant(Long nanos) {
            return Instant.ofEpochMilli(nanos / 1_000_000);
        }

        private void fetchHeartRateDataFromGoogleFit(OAuthCredentials creds,
            Long startMillis, Long endMillis) {

            log.debug("Fetching HR: start={}, end={}, tokenScopes={}", startMillis,
                endMillis, creds.getScope());
            log.debug("Nutrition window: {} / {}", Instant.ofEpochMilli(startMillis),
                Instant.ofEpochMilli(endMillis));
            String url =
                "https://www.googleapis.com/fitness/v1/users/me/dataset:aggregate";

```

```

    Map<String, Object> payload = Map.of(
        "aggregateBy", List.of(Map.of("dataTypeName",
"com.google.heart_rate.bpm")),
        "bucketByTime", Map.of("durationMillis", 60_000L),
        "startTimeMillis", startMillis,
        "endTimeMillis", endMillis);

    var rs = restTemplate.postForEntity(
        url,
        new HttpEntity<>(payload, bearerJson(creds.getAccessToken())),
        GoogleFitHeartRateResponse.class).getBody();

    if (rs == null)
        return;

    rs.getBucket().forEach(b -> b.getDataset().forEach(ds ->
ds.getPoint().forEach(pt -> {

        LocalDateTime ts = nanosToUtc(pt.getStartTimeNanos());
        Double bpm = pt.getValue().getFirst().getFpVal();

        HeartRateTracking hr = new HeartRateTracking();
        hr.setPatientProfile(creds.getPatientProfile());
        hr.setTimestamp(ts);
        hr.setBpm(bpm);

        try {
            heartRepository.save(hr);
        } catch (DataIntegrityViolationException dup) {
            log.debug("Duplicate HR ignored: {}", dup.getMessage());
        }
    })));
}

private HttpHeaders bearerJson(String token) {
    HttpHeaders h = new HttpHeaders();
    h.setBearerAuth(token);
    h.setContentType(MediaType.APPLICATION_JSON);
    return h;
}

private LocalDateTime nanosToUtc(Long nanos) {
    return Instant.ofEpochMilli(nanos / 1_000_000L)
        .atZone(ZoneOffset.UTC)
        .toLocalDateTime();
}

private boolean isTokenExpired(OAuthCredentials creds) {
    return creds.getExpiresAt() != null &&
Instant.now().isAfter(creds.getExpiresAt());
}

```

```

}

private void refreshToken(OAuthCredentials creds) {
    HttpHeaders headers = new HttpHeaders();
    headers.setContentType(MediaType.APPLICATION_FORM_URL_ENCODED);

    MultiValueMap<String, String> form = new LinkedMultiValueMap<>();
    form.set("grant_type", "refresh_token");
    form.set("client_id", googleFitClientId);
    form.set("client_secret", googleFitClientSecret);
    form.set("refresh_token", creds.getRefreshToken());

    HttpEntity<MultiValueMap<String, String>> request = new HttpEntity<>(form,
headers);

    ResponseEntity<GoogleTokenResponse> response = restTemplate.postForEntity(
        "https://oauth2.googleapis.com/token",
        request,
        GoogleTokenResponse.class);

    GoogleTokenResponse tokenResp = response.getBody();
    if (tokenResp == null || tokenResp.getAccessToken() == null) {
        return;
    }
    if (tokenResp != null && tokenResp.getAccessToken() != null) {
        creds.setAccessToken(tokenResp.getAccessToken());
        Long expiresIn = tokenResp.getExpiresIn();
        creds.setExpiresAt(Instant.now().plusSeconds(expiresIn));
        if (tokenResp.getRefreshToken() != null) {
            creds.setRefreshToken(tokenResp.getRefreshToken());
        }
        if (tokenResp.getScope() != null) {
            creds.setScope(tokenResp.getScope());
        }
        credentialRepository.save(creds);
    }
}

private HttpHeaders buildBearerHeaders(String accessToken) {
    HttpHeaders headers = new HttpHeaders();
    headers.setBearerAuth(accessToken);
    return headers;
}

private String toIso8601(Long millis) {
    Instant instant = Instant.ofEpochMilli(millis);
    return instant.toString();
}

```

```

        private LocalDateTime toLocalDateTime(Long millis) {
            return
Instant.ofEpochMilli(millis).atZone(ZoneOffset.UTC).toLocalDateTime();
        }

        public void fetchAndSaveForUser(OAuthCredentials creds) {
            if (!"GOOGLE_FIT".equalsIgnoreCase(creds.getVendor()))
                return;
            if (isTokenExpired(creds))
                refreshToken(creds);

            Long end = System.currentTimeMillis();
            Long start = end - 86_400_000;

            fetchSleepDataFromGoogleFit(creds, start, end);
            fetchCaloriesDataFromGoogleFit(creds, start, end);
            fetchActivityDataFromGoogleFit(creds, start, end);
            fetchHeartRateDataFromGoogleFit(creds, start, end);
        }
    }
}

```

SleepDataService.java

```

package com.example.health_monitor.service;

import com.example.health_monitor.dto.SleepDataDto;

import java.util.List;

public interface SleepDataService {
    List<SleepDataDto> getAllSleepData();

    List<SleepDataDto> getByPatient(Long patientProfileId);

    SleepDataDto getSleepDataById(Long id);

    SleepDataDto createSleepData(SleepDataDto sleepDataDto);

    SleepDataDto updateSleepData(Long id, SleepDataDto sleepDataDto);

    void deleteSleepData(Long id);

    String evaluateQuality(SleepDataDto dto);
}

```