МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ЗАПОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

МАТЕМАТИЧНИЙ ФАКУЛЬТЕТ
Кафедра програмної інженерії

КВАЛІФІКАЦІЙНА РОБОТА БАКАЛАВРА

на тему: **«РОЗРОБКА БАГАТОРІВНЕВОГО ХМАРНОГО ТЕЛЕГРАМ-БОТА ЗАСОБАМИ AIOGRAM ТА FASTAPI, З ВИКОРИСТАННЯМ VERTEXAI»**

Виконав: студент 4 курсу, групи 6.1212-пі-1
спеціальності 121 інженерія програмного забезпечення
(шифр і назва спеціальності)
освітньої програми програмна інженерія
(назва освітньої програми)
В.І. Сєргєєв
(ініціали та прізвище)
Керівник доцент кафедри програмної інженерії, PhD Столярова А.В.
(посада, вчене звання, науковий ступінь, прізвище та ініціали)
Рецензент завідувач кафедри фундаментальної та прикладної математики, професор, д.т.н. Гребенюк С.М.
(посада, вчене звання, науковий ступінь, прізвище та ініціали)

Запоріжжя – 2026

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ЗАПОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

Факультет математичний

Кафедра програмної інженерії

Рівень вищої освіти бакалавр

Спеціальність 121 інженерія програмного забезпечення
(шифр і назва)

Освітня програма програмна інженерія

ЗАТВЕРДЖУЮ
Завідувач кафедри програмної інженерії, к.ф.-м.н., доцент

_____ Лісняк А.О.
(підпис)

“ ____ ” _____ 2025 р.

З А В Д А Н Н Я

НА КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТОВІ

Сєргєєву Вадиму Ігоровичу
(прізвище, ім’я та по-батькові)

1. Тема роботи Розробка багаторівневого хмарного Телеграм-бота засобами Aiogram та FastAPI, з використанням VertexAI

керівник роботи Столярова Анастасія Валеріївна, PhD
(прізвище, ім’я та по-батькові, науковий ступінь, вчене звання)

затверджені наказом ЗНУ від « 02 » грудня 2025 року № 1741-с

2. Строк подання студентом роботи 01.06.2026 р.

3. Вихідні дані до роботи 1. Постановка задачі.
2. Перелік літератури.

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)
1. Основні теоретичні відомості та аналіз предметної області.
2. Проєктування системи.
3. Реалізація та тестування.

5. Перелік графічного матеріалу (з точним зазначенням обов’язкових креслень)
презентація за темою доповіді

6. Консультанти розділів роботи

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

7. Дата видачі завдання 09.12.2025 р.

К А Л Е Н Д А Р Н И Й П Л А Н

№	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1.	Розробка плану роботи.	19.12.2025	
2.	Збір вихідних даних.	12.01.2026	
3.	Обробка методичних та теоретичних	05.02.2026	
	джерел.		
4.	Розробка першого та другого розділу.	18.03.2026	
5.	Розробка третього розділу.	21.05.2026	
6.	Оформлення та нормоконтроль	25.05.2026	
	кваліфікаційної роботи бакалавра.		
7.	Захист кваліфікаційної роботи.	16.06.2026	

Студент _____ В.І. Сєргєєв
(підпис) (ініціали та прізвище)

Керівник роботи _____ А.В. Столярова
(підпис) (ініціали та прізвище)

Нормоконтроль пройдено

Нормоконтролер _____ А.В. Столярова
(підпис) (ініціали та прізвище)

РЕФЕРАТ

Кваліфікаційна робота бакалавра «Розробка багаторівневого хмарного Телеграм-бота засобами Aiogram та FastAPI, з використанням VertexAI»: 55 с., 29 рис., 16 табл., 13 джерел.

TELEGRAM-БОТ, ВЕБІНТЕРФЕЙС, КОНТЕЙНЕРИЗАЦІЯ, ОБЛІК СНУ, AI-РЕКОМЕНДАЦІЇ, ХМАРНА АРХІТЕКТУРА, AIOGRAM, FASTAPI, POSTGRESQL, SUPABASE, VERTEXAI.

Об’єкт дослідження – процес обліку, інтелектуального аналізу та оптимізації режиму сну користувачів з використанням хмарних технологій та моделей штучного інтелекту.

Мета роботи: розробити багаторівневий хмарний Telegram-бот із вебінтерфейсом, серверною частиною на FastAPI, централізованою базою даних та інтеграцією VertexAI для формування персоналізованих рекомендацій щодо режиму сну.

Методи дослідження – аналіз предметної області та аналогів, об’єктно-орієнтоване проєктування, моделювання архітектури та даних, проєктування REST API, контейнеризація, інтеграційне та модульне тестування.

У кваліфікаційній роботі розглянуто проблему створення зручної системи для регулярного обліку сну користувача, яка поєднує швидку взаємодію через Telegram-бота з повноцінним вебінтерфейсом для перегляду історії, статистики та налаштувань. Проведено аналіз наявних рішень для моніторингу сну, формування звичок і використання чат-ботів, на основі якого визначено функціональну прогалину: відсутність єдиної системи, що одночасно підтримує Telegram-інтерфейс, вебдоступ, централізоване сховище даних, нагадування, статистику та ШІ-рекомендації.

SUMMARY

Bachelor’s qualifying paper “Development of a Multi-Tier Cloud Telegram Bot Using Aiogram and FastAPI, with the Integration of VertexAI”: 55 pages, 29 figures, 16 tables, 13 references.

AI RECOMMENDATIONS, AIOGRAM, CLOUD ARCHITECTURE, CONTAINERIZATION, FASTAPI, POSTGRESQL, SLEEP TRACKING, SUPABASE, TELEGRAM BOT, VERTEXAI, WEB INTERFACE.

Object of research is the process of tracking, intelligent analysis, and optimization of users' sleep patterns using cloud technologies and artificial intelligence models.

The objective of the work is to develop a multi-tier cloud-based Telegram bot with a web interface, a FastAPI-powered backend, a centralized database, and VertexAI integration to generate personalized sleep schedule recommendations.

Research methods include domain and competitive analysis, object-oriented design, data and architecture modeling, REST API design, containerization, as well as integration and unit testing.

The qualification thesis addresses the challenge of creating an intuitive system for regular user sleep tracking that combines rapid interaction via a Telegram bot with a comprehensive web interface for viewing history, statistics, and configurations. An analysis of existing solutions for sleep monitoring, habit formation, and chatbot utilization was conducted, which helped identify a functional gap: the lack of a single system that simultaneously supports a Telegram interface, web access, centralized data storage, reminders, statistics, and AI-driven recommendations.

ЗМІСТ

Завдання на кваліфікаційну роботу……..2
Реферат 4
Summary 5
Вступ..... 8
1 Основні теоретичні відомості та аналіз предметної області 11
1.1 Багаторівневий хмарний Telegram-бот 11
1.1.1 Telegram-бот як рівень представлення системи 11
1.1.2 Багаторівнева архітектура..... 11
1.1.3 Хмарна інфраструктура..... 13
1.2 Аналіз аналогів..... 13
1.2.1 Комерційні трекери сну..... 14
1.2.2 Чат-боти для формування звичок..... 14
1.2.3 Технічні аналоги 15
1.3 Заглиблення у предметну область..... 16
1.4 Функціональні вимоги..... 18
1.5 Нефункціональні вимоги..... 19
1.5.1 Безпека 19
1.5.2 Продуктивність та оптимізація..... 20
1.5.3 Зручність використання 21
2 Проєктування системи..... 22
2.1 Проєктування архітектури та розгортання системи..... 22
2.2 Проєктування моделі даних 25
2.3 Проєктування REST API та сценаріїв обробки запитів 27
2.4 Проєктування Telegram-бота 31
2.5 Проєктування вебінтерфейсу..... 33
2.6 Проєктування ШІ-підсистеми та фонових процесів 35
3 Реалізація та тестування 38

3.1 Середовище розробки та організація проєкту 38
3.2 Реалізація серверної частини 40
3.3 Реалізація клієнтських інтерфейсів 43
3.3.1 Telegram-бот 43
3.3.2 Вебінтерфейс 45
3.4 Тестування 48
Висновки 52
Перелік посилань 54

ВСТУП

Сон є одним із базових чинників, що впливають на працездатність, самопочуття та загальний режим життя людини [1], тому в сучасних умовах зростає потреба у цифрових інструментах для самоспостереження, планування режиму дня та формування корисних звичок. У межах обраної предметної області доцільним є розроблення багаторівневої хмарної системи, яка поєднує Telegram-бота для швидкого внесення даних і отримання нагадувань, вебінтерфейс для перегляду історії, статистики та налаштувань, серверну частину, централізовану базу даних і зовнішній ШІ-сервіс [2]. На відміну від окремих мобільних застосунків або простих ботів-нагадувачів, така система дає змогу не лише фіксувати окремі показники, а й накопичувати історію, аналізувати динаміку та використовувати дані в різних каналах взаємодії, що обумовлює актуальність даного проєкту.

Мета – розробити багаторівневий хмарний Telegram-бот із вебінтерфейсом, серверною частиною на FastAPI, централізованою базою даних та інтеграцією VertexAI для формування персоналізованих рекомендацій щодо режиму сну.

Досягнення поставленої мети передбачає реалізацію наступних **завдань**:

- проаналізувати предметну область обліку сну та існуючі рішення для моніторингу сну, формування звичок і роботи з нагадуваннями;
- дослідити принципи побудови багаторівневих хмарних систем із Telegram-інтерфейсом, вебінтерфейсом і централізованим API;
- розробити серверну частину, клієнтські інтерфейси, механізми збереження даних, нагадувань і формування рекомендацій;
- спроєктувати архітектуру системи, модель даних, REST API, сценарії роботи Telegram-бота, вебінтерфейс, ШІ-підсистему та фонові процеси;

– протестувати розроблений функціонал за допомогою модульного та інтеграційного тестування.

Об’єкт дослідження – процес обліку, інтелектуального аналізу та оптимізації режиму сну користувачів з використанням хмарних технологій та моделей штучного інтелекту.

Предмет дослідження – архітектурні, програмні та інтеграційні засоби реалізації Telegram-бота, вебінтерфейсу, REST API, централізованої бази даних і ШІ-підсистеми для формування персоналізованих рекомендацій щодо режиму сну.

Методи дослідження – аналіз предметної області та аналогів, об’єктно-орієнтоване проєктування, моделювання архітектури та даних, проєктування REST API, контейнеризація, інтеграційне та модульне тестування.

Для реалізації проєкту було обрано мову Python, фреймворк FastAPI для створення серверної частини та Aiogram для розробки Telegram-бота. Збереження даних організовано за допомогою PostgreSQL у хмарному середовищі Supabase, що забезпечує централізовану роботу з профілями користувачів, сесіями сну, цілями, нагадуваннями та службовими записами. Вебінтерфейс реалізовано як окремий клієнтський компонент, що взаємодіє із серверною частиною через REST API, а для формування персоналізованих рекомендацій використано інтеграцію з VertexAI. У результаті розроблено інформаційну систему для регулярного обліку сну, перегляду історії та статистики, налаштування цілей і нагадувань, а також отримання рекомендацій, що надає практичне значення роботі як програмному рішенню для самоспостереження за режимом сну.

Основні положення і результати кваліфікаційної роботи доповідались, обговорювалися та знайшли схвалення на науково-практичній конференції: Сімнадцятій Всеукраїнській, Двадцять четвертій регіональній науковій конференції молодих дослідників «Актуальні проблеми математики та інформатики» (Запоріжжя, 23-24 квітня 2026 р.), за результатами якої опубліковано тези.

Кваліфікаційна робота складається зі вступу, трьох розділів, висновків та переліку посилань. Загальний обсяг роботи становить 55 сторінок. У переліку посилань 13 використаних джерел.

Перший розділ присвячений основним теоретичним відомостям та аналізу предметної області. У другому розділі наведено проєктування системи, проілюстровано архітектуру, модель даних, та сценарії обробки запитів. В третьому розділі описано реалізацію серверної частини, Telegram-бота, вебінтерфейсу та тестування розробленого функціоналу.

1 ОСНОВНІ ТЕОРЕТИЧНІ ВІДОМОСТІ ТА АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Багаторівневий хмарний Telegram-бот

Telegram-бот можна розглядати як програмного агента, що взаємодіє з клієнтом через відповідний API та виконує конкретні дії в залежності від отриманих команд, повідомлень або натискань кнопок.

1.1.1 Telegram-бот як рівень представлення системи

У межах кваліфікаційної роботи було прийнято рішення розглядати бота не як окремий елемент, у якому зосереджена вся логіка застосунку, а як один з рівнів представлення багаторівневої хмарної системи, де основна бізнес-логіка, інтеграція штучного інтелекту та робота с базою даних винесені до серверної частини [3–4]. Такий підхід дозволяє спростити підтримку та відокремити інтерфейс, що відкриває можливість використання кількох каналів взаємодії.

Перевага рішення полягає в централізації даних, а логіка обробки ключових функцій не дублюється у різних частинах системи. У разі зміни бізнес-правил достатньо оновити серверну частину, після чого зміни будуть доступні для всіх клієнтів.

1.1.2 Багаторівнева архітектура

Після аналізу можливих підходів до побудови системи було обрано модульну багаторівневу архітектуру, оскільки вона дозволяє розділити

відповідальність між компонентами та забезпечити гнучкість подальшого розвитку продукту. Ця перевага є критичною для запланованого функціоналу.

У результаті проєкт було поділено на наступні рівні: представлення, прикладний рівень, бізнес-логіка, дані та зовнішні ресурси.

Рівень представлення у розробленій системі реалізований через Telegram-бота та вебінтерфейс. прикладний рівень створено засобами Python фреймворку FastAPI, який виконує обробку HTTP-запитів та надає централізований API для клієнтів.

Рівень бізнес-логіки відповідає за облік сесій сну, аналіз повторюваних патернів, формування рекомендацій та роботу з нагадуваннями. Зовнішні ресурси представлені інтеграцією сервісів Google Cloud, зокрема VertexAI [5].

Для *рівня даних* було використано СУБД PostgreSQL із використанням сервісу Supabase, що забезпечує хмарний доступ до сховища, зручне адміністрування та можливість централізованого збереження інформації користувачів [6].

У таблиці 1.1 наведено відповідність рівнів багаторівневої архітектури компонентам розробленої системи.

Таблиця 1.1 – Відповідність рівнів архітектури

Рівень системи	**Реалізація у застосунку**
Представлення	Вебінтерфейс, Telegram-бот засобами фреймворку Aiogram
Прикладний рівень	FastAPI routers, централізований API
Бізнес-логіка	Аналіз сну, рекомендації, нагадування
Дані	PostgreSQL/Supabase
Зовнішні ресурси	Google Cloud, VertexAI

Отже, обрана архітектура дозволяє не дублювати логіку між Telegram-ботом і вебінтерфейсом, а використовувати спільний backend та єдину базу даних.

1.1.3 Хмарна інфраструктура

Хмарна модель у межах цієї роботи передбачає винесення окремих компонентів системи до зовнішніх сервісів: база даних розміщується у Supabase, ШІ-функціональність реалізується за допомогою Google Cloud, а серверна частина та Telegram-бот можуть бути розгорнуті як контейнеризовані компоненти. Такий підхід дозволяє масштабувати окремі частини системи незалежно одна від одної.

Окремою перевагою хмарної моделі є можливість незалежного масштабування компонентів. Це дає можливість окремо розвивати та оновлювати базу даних, серверну частину, Telegram-бота і ШІ-сервіс. Це знижує зв’язаність між частинами системи та спрощує її подальшу підтримку. Крім того, використання контейнеризації дає змогу уніфікувати середовище запуску застосунку та спростити перенесення системи між локальним, тестовим і хмарним середовищем [7].

Таким чином, абстрактною предметною областю цього проєкту є побудова багаторівневого хмарного бота, у якому месенджер виступає каналом взаємодії для користувача, а основна логіка реалізується на серверному рівні. Далі має сенс розглянути вже існуючі рішення, які вирішують подібні задачі.

1.2 Аналіз аналогів

Для обґрунтування доцільності розробленої системи необхідно розглянути схожі проєкти, які пов’язані з обліком сну, формуванням корисних звичок, та ШІ-рекомендаціями. Подібний аналіз дозволить визначити, які функції вже реалізовані в наявних продуктах, а які залишаються недостатньо представленими. Загально аналоги можна поділити на три групи: комерційні трекери сну, Telegram-боти для формування звичок та технічні аналоги зі схожими засобами розробки.

1.2.1 Комерційні трекери сну

Комерційні трекери, що представлені на ринку можуть бути реалізовані у різних формах, це можуть бути як вузькоспеціалізовані додатки, наприклад Sleep Cycle та Pillow, так і масштабні системи для моніторингу стану тіла, наприклад Samsung Health та Google Fit [8–11].

Рішення на кшталт Sleep Cycle є прикладами застосунків, що орієнтовані на аналіз сну за допомогою смартфона. Такі додатки мають розвинену аналітику, однак переважно функціонують лише у межах власної мобільної екосистеми, та не мають інших каналів взаємодії з користувачем.

Аналоги, що націлені на більш широкий моніторинг здоров’я, на кшталт Google Fit, окрім сну можуть відстежувати більше показників (фізичну активність, стан пульсу). Проте, оскільки ці системи не спеціалізуються на відстеженні саме якості сну відповідний функціонал у них значно менший. Також варто зазначити, що для повноцінного використання подібних рішень необхідні окремі аксесуари які входять до екосистеми додатку.

1.2.2 Чат-боти для формування звичок

Чат-боти для формування звичок і нагадувань мають перевагу у простоті використання, оскільки користувач взаємодіє з ними у звичному середовищі месенджера. Такі рішення можуть бути корисними для щоденних нагадувань, фіксації виконаних дій або підтримки регулярності певної поведінки. Однак їх функціональність зазвичай обмежується простими сценаріями: створенням нагадування, отриманням повідомлення та підтвердженням виконання дії.

Для задачі обліку сну цього недостатньо, оскільки система має не лише нагадувати користувачу про необхідність дотримання режиму, а й накопичувати історичні дані, аналізувати динаміку, формувати статистику та надавати персоналізовані рекомендації. Крім того, більшість простих чат-

ботів не мають повноцінного вебінтерфейсу та централізованої моделі даних, яка дозволяє використовувати один профіль користувача в різних каналах взаємодії.

1.2.3 Технічні аналоги

Технічні аналоги, побудовані на основі Aiogram та FastAPI, є важливими з точки зору вибору архітектурного підходу. Вони демонструють, що Telegram-бот може бути реалізований не як самостійний монолітний застосунок, а як клієнт до серверного API. У такому випадку бот відповідає за взаємодію з користувачем, а серверна частина забезпечує обробку даних, авторизацію, доступ до бази даних та виконання бізнес-логіки.

Водночас такі рішення зазвичай мають загальний характер і не орієнтовані саме на предметну область обліку сну. Вони можуть бути корисними як технічний орієнтир, однак не вирішують задачі аналізу якості сну, формування персоналізованих порад, роботи з цілями сну та побудови статистики. Тому розроблена система поєднує технічну модель Telegram-бота як клієнта до API з конкретною предметною логікою системи.

У таблиці 1.2 подано наочний аналіз та порівняння аналогів.

Таблиця 1.2 – Порівняння аналогів

Критерій	**Sleep Cycle**	**Google Fit**	**StreaksBot**	**Open-source**	**Розроблена система**
Telegram-інтерфейс	Відсутній	Відсутній	Наявний	Наявний	Наявний
Вебінтерфейс	Відсутній	Відсутній	Відсутній	Наявний	Наявний
Облік сну	Наявний	Наявний	Частково наявний	Відсутній	Наявний
ШІ-функції	Відсутні	Відсутні	Відсутні	Відсутні	Наявні
Нагадування	Наявні	Наявні	Наявні	Відсутні	Наявні
Єдина база даних	Закрита інформація	Закрита інформація	Наявна	Наявна	Наявна

Проведений аналіз показує, що існуючі рішення частково покривають окремі аспекти обліку сну, нагадувань або взаємодії з користувачем. Водночас вони не поєднують у межах однієї системи Telegram-інтерфейс, вебінтерфейс, централізоване сховище даних, AI-рекомендації, статистику та можливість подальшого розширення архітектури. Саме ця функціональна та архітектурна прогалина обґрунтовує доцільність розробки системи.

1.3 Заглиблення у предметну область

Конкретною предметною областю проєкту є облік, аналіз і покращення режиму сну користувача за допомогою інформаційної системи. Більшість користувачів часто оцінюють якість власного сну суб’єктивно та нерегулярно, системне збереження цієї інформації вирішує проблему.

Основна мета розробленого застосунку – надати користувачу інструмент для самоспостереження та отримання інформативних рекомендацій. У якості акторів системи виступають адміністратор і користувач.

Користувач взаємодіє із системою через наявні канали, додає записи про сон, встановлює цілі, отримує статистику та персоналізовані поради.

Адміністратор має доступ до розширеного функціоналу і службових функцій системи, зокрема перегляд загальної статистики усіх користувачів і журналу аудиту. На рисунку 1.1 зображені наявні основні сценарії взаємодії користувача і адміністратора.

До меж розробленої системи входять: Telegram-бот, вебінтерфейс, серверне API, база даних, модуль статистики, модуль нагадувань, адміністративний функціонал та інтеграція з ШІ-сервісом. Поза межами системи залишаються медична діагностика, інтеграція з фітнес пристроями, автоматичне зчитування біометричних показників та платіжна функціональність. Такий поділ дозволяє чітко визначити призначення системи як інструмента самоспостереження, а не як медичного програмного забезпечення.

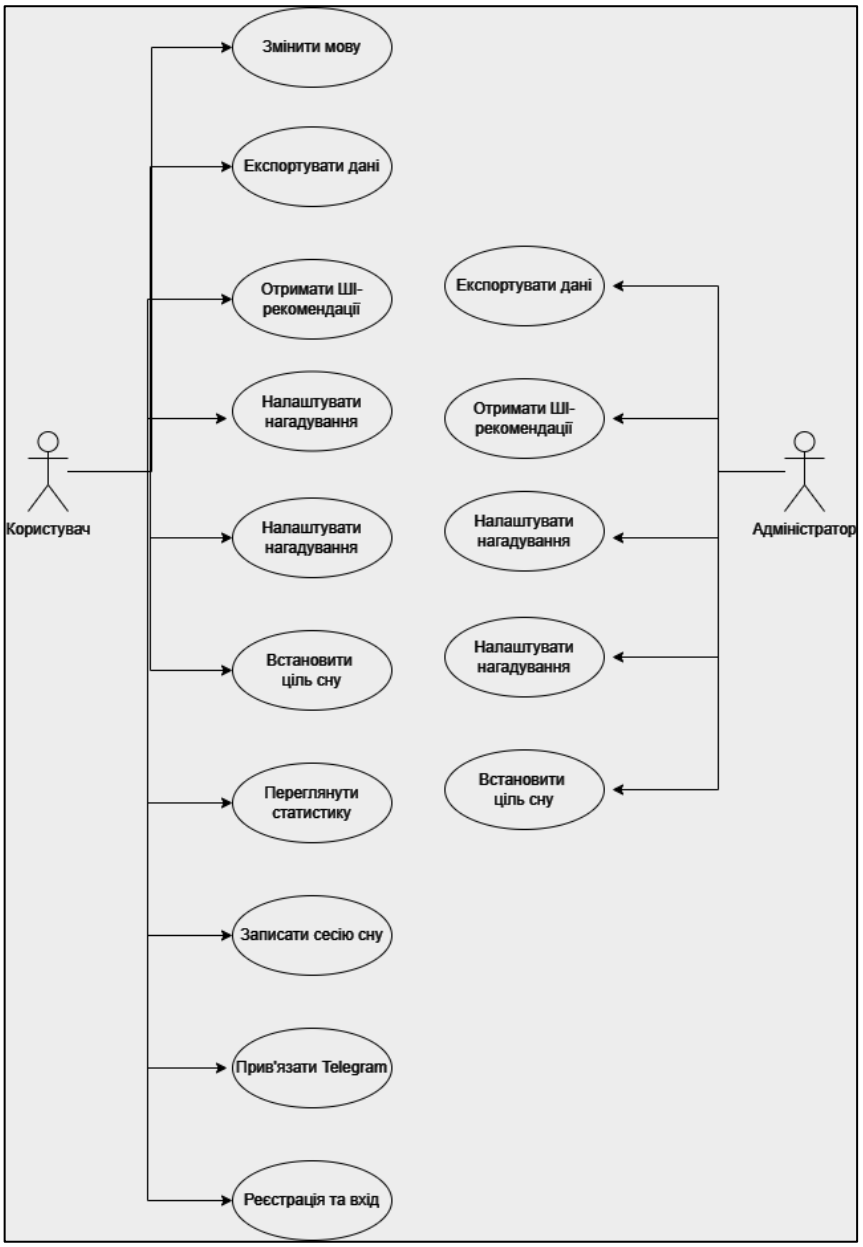


Рисунок 1.1 – Use Case діаграма

Базовим сценарієм роботи користувача є внесення запису про сон. Користувач вказує час початку та завершення сну, оцінку якості та за потреби додає нотатку. Після збереження запису система може використати ці дані для розрахунку тривалості сну, побудови статистики та формування рекомендацій.

Окремий сценарій пов’язаний із формуванням персоналізованої рекомендації На основі збережених записів система аналізує режим користувача та передає узагальнені дані до зовнішнього ШІ ресурсу, який генерує відповідну рекомендацію.

Основними важливими інформаційними сутностями системи є користувач, сесія сну, ціль сну, нагадування та журнал аудиту. Сутність користувача зберігає дані профілю та зв’язок з Telegram. Сесія сну містить інформацію про тривалість, якість і нотатки. Ціль сну задає бажаний діапазон тривалості, нагадування відповідає за регулярну комунікацію з користувачем, а журнал аудиту використовується для фіксації адміністративних дій.

Таким чином програма конкретизує загальну багаторівневу модель у сфері обліку та аналізу. Визначені сценарії та сутності є основою для формування системи.

1.4 Функціональні вимоги

Функціональні вимоги, наведені в таблиці 1.3, визначають основні можливості, які повинна надавати система. Вони формуються на основі сценаріїв використання, описаних у попередньому підрозділі.

Таблиця 1.3 – Основні функціональні вимоги

Назва вимоги	**Опис**	**Пріоритет**
Реєстрація користувача	Можливість створення облікового запису	Високий
Авторизація користувача	Система повинна забезпечувати вхід користувача	Високий
Прив’язка Telegram	Реалізація зв’язування Telegram-акаунта з профілем користувача	Високий
Додавання сесій сну	Користувач повинен мати можливість додавати записи про сон	Високий
Перегляд історії сну	Система має відображати список попередніх сесій	Високий
Статистика	Можливість побачити та відслідкувати тижневу і місячну динаміку	Високий
ШІ-поради	Формування персоналізованих рекомендацій через VertexAI	Високий
Зміна мови	Система повинна підтримувати українську та англійську мови	Середній
Журнал аудиту	Зберігання інформації про адміністративні дії	Середній
Експорт даних	Можливість експортувати дані у CSV/JSON	Низький

Найважливішою функцією системи є створення записів про сон, оскільки саме ці дані використовуються для історії, статистики, аналізу патернів і формування рекомендацій.

Єдність профілю між вебінтерфейсом і чат-ботом забезпечується механізмом прив’язки Telegram-акаунта. Після цього записи з різних каналів належать одному користувачу.

Окреме значення має вимога формування ШІ-порад, оскільки вона відрізняє систему від звичайного журналу обліку сну. На основі накопичених записів система може сформувати узагальнений контекст і передати його до VertexAI для створення рекомендації, зрозумілої для користувача.

Сформульовані функціональні вимоги визначають набір дій, які система має підтримувати для користувача й адміністратора. Наступним кроком є визначення нефункціональних вимог, які описують якість реалізації цих можливостей.

1.5 Нефункціональні вимоги

Нефункціональні вимоги визначають якісні характеристики проєкту. На відміну від функціональних вимог, вони не описують окремі сценарії, а встановлюють вимоги до способу реалізації системи.

1.5.1 Безпека

Для захисту облікових даних паролі користувачів не повинні зберігатися у відкритому вигляді. Замість цього використовується хешування, що знижує ризики у випадку несанкціонованого доступу до бази даних. Доступ до захищених шляхів має здійснюватися лише після автентифікації користувача. При цьому система повинна перевіряти, що користувач працює лише зі своїми записами сну, цілями та нагадуваннями.

Основні механізми забезпечення безпеки системи узагальнено в таблиці 1.4.

Таблиця 1.4 – Нефункціональні вимоги безпеки

Формулювання вимоги	**Механізм реалізації**
Захист паролів	Хешування
Захист API	JWT
Контроль доступу	Перевірка за ідентифікатором користувача
Валідація вводу	Схеми бібліотеки Pydantic

Окрему увагу необхідно приділити адміністративним функціям, оскільки вони передбачають доступ до ширшого набору даних. Для таких дій доцільно використовувати перевірку ролі користувача та фіксацію операцій, що дозволяє відстежувати зміни в системі.

1.5.2 Продуктивність та оптимізація

Продуктивність системи є важливою, оскільки користувач взаємодіє з нею через декілька каналів зв’язку та очікує швидкої відповіді від кожної з них. Типові операції, такі як додавання сесії сну, перегляд записів або отримання базової статистики, повинні виконуватися без помітної затримки.

Таблиця 1.5 відображає бажані показники для основних процесів та способи, за допомогою яких вони можуть бути забезпечені.

Таблиця 1.5 – Бажані показники оптимізації

Показник	**Бажане значення**	**Спосіб забезпечення**
Відповідь на простий запит	До 1 секунди	Оптимізовані запити, індекси
Відповідь чат-бота	Без помітної затримки	Асинхронна обробка
Запит до ШІ	До 30 секунд	Кешування та обмеження запитів

Найбільший ризик затримки пов’язаний із ШІ-функціями, оскільки вони залежать від зовнішнього сервісу. Для таких запитів доцільно застосовувати кешування, обмеження частоти звернень і fallback-відповіді.

1.5.3 Зручність використання

Зручність використання розробленої системи визначається тим, наскільки швидко користувач може виконати типові дії.

Наявність Telegram-інтерфейсу знижує поріг входу, оскільки користувач може взаємодіяти із системою у звичному месенджері без необхідності щоразу відкривати окремий вебзастосунок.

Вебінтерфейс доповнює чат-бота, оскільки дозволяє зручніше переглядати статистику, графіки та історію записів [12]. Таким чином, один інтерфейс краще підходить для простих дій, а інший для аналізу даних.

Зручність також забезпечується тим, що користувач може обирати найбільш відповідний канал взаємодії: Telegram-бот для швидких дій і вебінтерфейс для детального перегляду інформації. Такий підхід підвищує гнучкість системи та робить її придатною для регулярного використання.

Отже, нефункціональні вимоги проєкту охоплюють безпеку, продуктивність та зручність використання. Їх виконання є важливим для того, щоб система була не лише функціонально повною, а й придатною для регулярного використання.

2 ПРОЄКТУВАННЯ СИСТЕМИ

На етапі проєктування було визначено, які програмні компоненти входять до складу системи, які дані вони обробляють і через які інтерфейси взаємодіють між собою. Проєктні рішення сформовано таким чином, щоб забезпечити підтримку двох каналів доступу без створення окремих незалежних систем для кожного з них.

2.1 Проєктування архітектури та розгортання системи

Проєктування архітектури проєкту виконано з урахуванням необхідності одночасної роботи декількох сервісів: вебінтерфейсу, Telegram-бота, серверного API, бази даних, кешу та зовнішньої ШІ-платформи. Такий поділ дає змогу описувати кожен компонент через його відповідальність, залежності та роль у загальному сценарії роботи.

У програмі основні виконувані частини представлені окремими контейнерами:

– Frontend відповідає за відображення вебінтерфейсу;
– сервер обробляє HTTP-запити;
– Telegram-бот взаємодіє з користувачами у месенджері;
– Redis використовується для кешування.

Дані користувачів зберігаються у PostgreSQL, розміщеній у Supabase, а генерація рекомендацій виконується через VertexAI.

На рисунку 2.1 показано, що користувач може взаємодіяти із системою двома способами. У першому випадку запити надходять до інтерфейсу або безпосередньо до серверного API, в іншому варіанті повідомлення обробляються Telegram Bot API та сервісом бота. Сервер звертається до бази даних, Redis і VertexAI, тоді як бот додатково використовує Telegram API для надсилання відповідей користувачу.

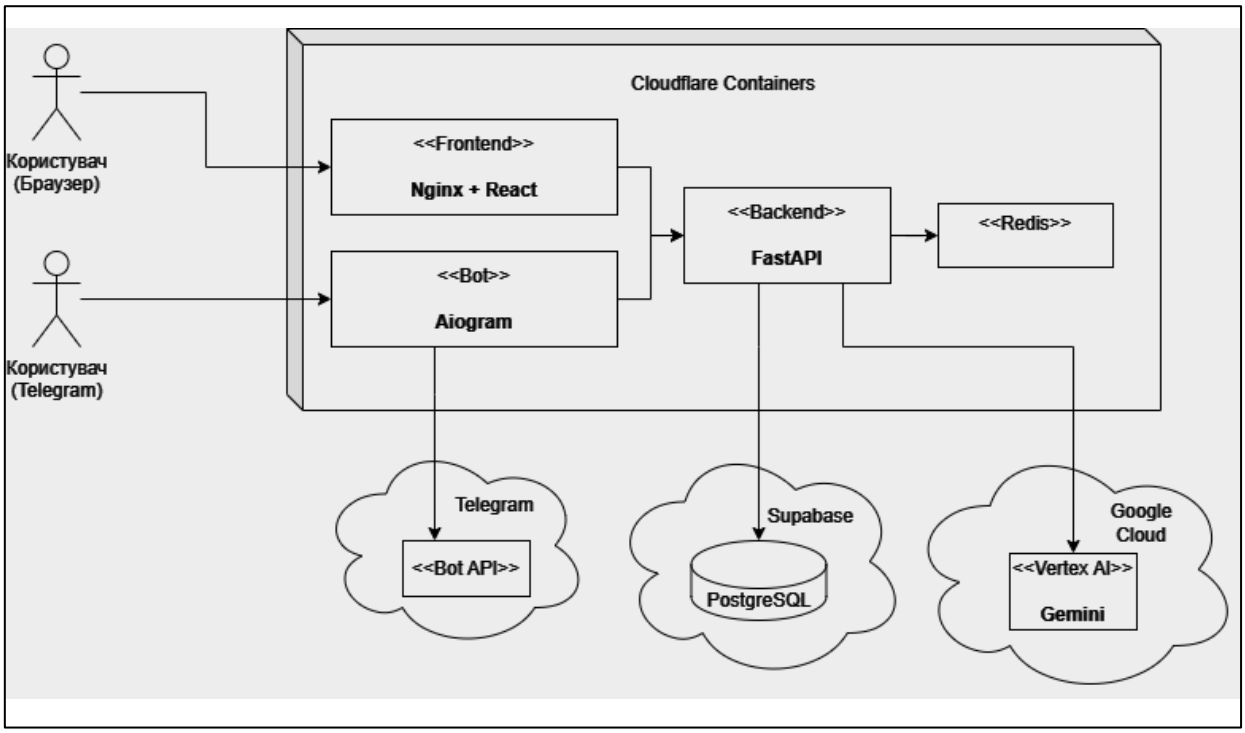


Рисунок 2.1 – Діаграма розгортання

Для деталізації внутрішньої організації було побудовано діаграму компонентів (див. рис. 2.2). Вона показує логічну участь кожного компонента у виконанні функцій системи.

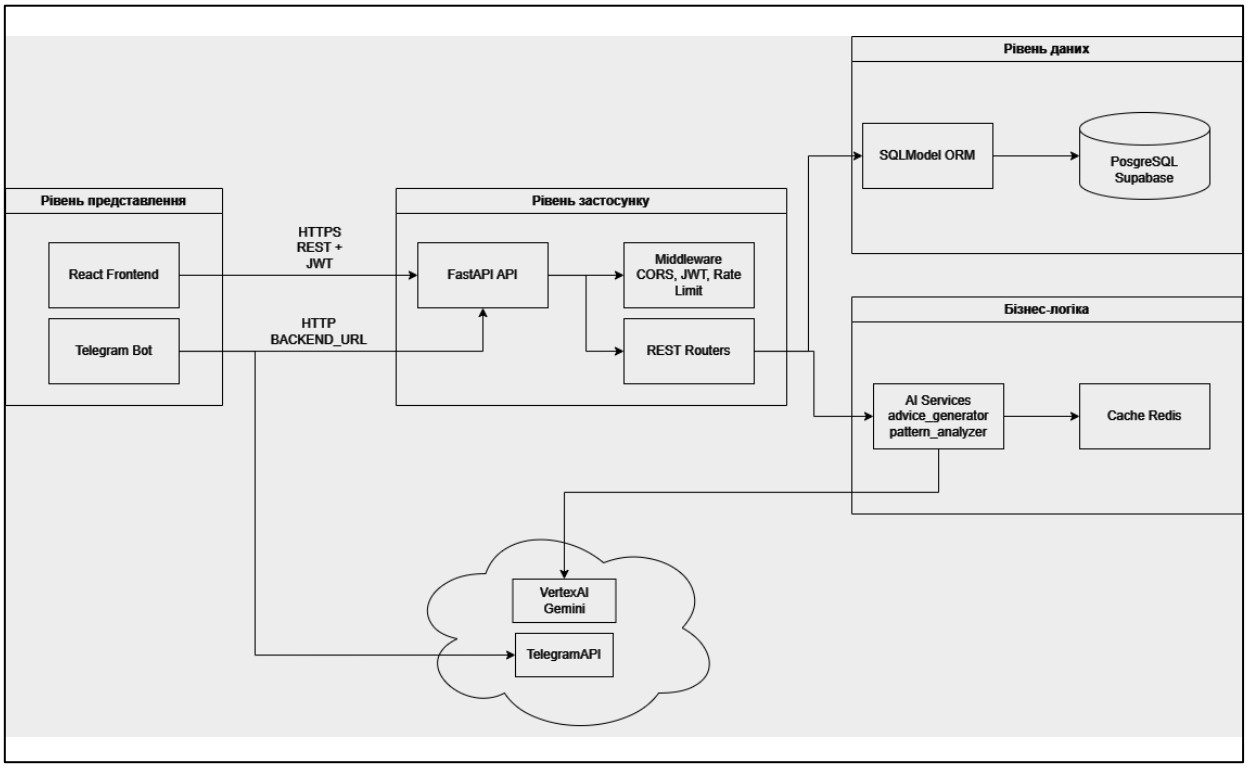


Рисунок 2.2 – Діаграма компонентів системи

Клієнтські частини системи не працюють безпосередньо з базою даних. Щоб зменшити кількість точок доступу до даних і спростити контроль над логікою роботи вебінтерфейс і Telegram-бот передають запити до FastAPI серверу, де виконується перевірка доступу, обробка даних і виклик відповідних сервісів.

У межах проєкту сервер виступає центральним координаційним елементом. Саме він перевіряє користувача, обробляє сесії сну, формує статистику, працює з нагадуваннями та ініціює звернення до ШІ-сервісів. Основні компоненти системи перелічено у таблиці 2.1. Структура модулів зазначена у таблиці 2.2.

Таблиця 2.1 – Основні компоненти системи

Компонент	**Призначення**	**Основні залежності**
React Frontend	Вебінтерфейс користувача та адміністратора	Сервер
Telegram Bot	Обробка команд і повідомлень у Telegram	Сервер, Telegram API
FastAPI Backend	Центральна обробка запитів і бізнес-логіки	PostgreSQL, Redis, VertexAI
PostgreSQL	Постійне збереження даних	Сервер, частково Bot
Redis	Кешування результатів ШІ-запитів	Сервер
VertexAI	Генерація рекомендацій і аналітичних відповідей	Сервер
Telegram API	Доставка повідомлень користувачам	Telegram Bot

Таблиця 2.2 – Структура програмних модулів

Модуль	**Що містить**	**Для чого**
app/routers/	REST-маршрути	Обробка HTTP-запитів
app/domain/	Моделі та схеми	Опис сутностей і DTO
app/services/	Сервіси бізнес-логіки	AI, кеш, аналітика, повідомлення
app/bot/	Обробники Telegram	Команди, меню, FSM-сценарії
app/workers/	Фонові задачі	Планувальник нагадувань
app/middleware/	Проміжна обробка	CORS, авторизація, rate limit

Отже, архітектура розробленого застосунку спроєктована як набір взаємопов’язаних компонентів, кожен з яких виконує окрему функцію. Розгортання у вигляді окремих сервісів дозволяє ізолювати клієнтські інтерфейси, сервер, кеш, базу даних і зовнішні інтеграції. Далі доцільно визначити, які дані зберігаються у системі та як вони пов’язані між собою.

2.2 Проєктування моделі даних

Модель даних (див. рис. 2.3) побудована навколо користувача та його записів про сон. Основними сутностями є користувач, сесія сну, ціль сну, нагадування та журнал аудиту. Ці сутності відповідають основним сценаріям:

– створенню профілю;
– фіксації сну;
– налаштуванню цілей;
– отриманню нагадувань;
– контролю адміністративних дій.

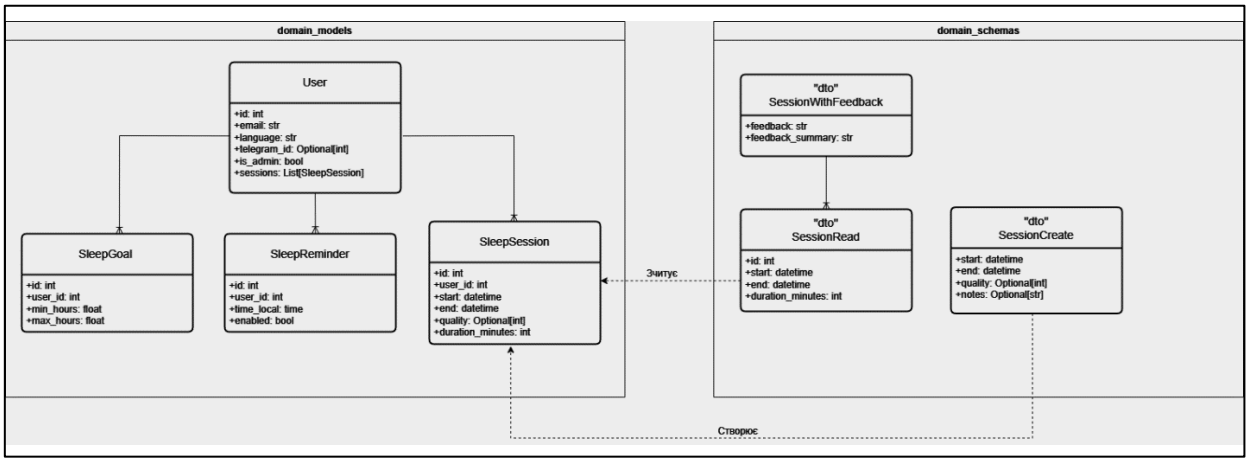


Рисунок 2.3 – Доменна модель даних системи

Центральною сутністю є User. Від неї залежать записи SleepSession, SleepGoal і SleepReminder. Зв’язок один-до-багатьох дозволяє одному користувачу мати багато сесій сну, декілька записів цілей і декілька

налаштувань нагадувань. У таблиці 2.3 перелічені усі основні сутності бази даних.

Таблиця 2.3 – Основні сутності бази даних

Сутність	**Призначення**	**Ключові поля**
User	Профіль користувача	id, email, hashed_password, telegram_id, is_admin
SleepSession	Окремий запис сну	id, user_id, start, end, quality, duration_minutes, notes
SleepGoal	Цільова тривалість сну	id, user_id, min_hours, max_hours, created_at
SleepReminder	Налаштування нагадувань	id, user_id, time_local, tz, enabled, last_sent_date
AuditLog	Журнал службових дій	id, user_id, action, resource_type, details, created_at

Поле user_id у пов’язаних таблицях використовується для розмежування даних різних користувачів. Це дозволяє системі вибирати лише ті записи, які належать поточному користувачу.

Для сесії сну зберігаються не лише введені користувачем значення, а й обчислене поле duration_minutes. Щоб уникнути ситуації, коли різні клієнтські інтерфейси по-різному розраховують тривалість воно формується на сервері на основі часу початку та завершення сну.

Сутність SleepGoal передбачає можливість зберігати декілька цілей для одного користувача. Поточна ціль може визначатися як останній створений запис. Таке рішення було прийнято для розширення статистики і можливості аналізу динаміки зміни цілей кожного клієнта.

Поле telegram_id у таблиці користувачів використовується для прив’язки Telegram-акаунта до профілю в системі. Завдяки цьому записи, створені через чат-бота, будуть пов’язані з тим самим користувачем, який працює через вебінтерфейс.

Отже, модель даних охоплює основні об’єкти, необхідні для роботи проєкту. Вона дозволяє зберігати історію сну, персональні налаштування

користувача, зв’язок із Telegram та службову інформацію. Наступним кроком є опис REST API, через який ці дані створюються, змінюються та отримуються клієнтськими частинами системи.

2.3 Проєктування REST API та сценаріїв обробки запитів

REST API у розробленому застосунку виконує роль основного інтерфейсу доступу до функцій серверної частини. Через нього канали взаємодії виконують свої головні функції.

Для спрощення структури було прийнято рішення групувати API за функціональними областями. Перелік областей можна побачити у таблиці 2.4.

Таблиця 2.4 – Групи REST API

Група маршрутів	**Призначення**	**Приклад функції**
/auth	Авторизація та прив’язка Telegram	Вхід, реєстрація, створення bot-link
/sessions	Робота із записами сну	Створення, перегляд, видалення
/goals	Цілі користувача	Додавання та перегляд цільової тривалості
/reminders	Нагадування	Створення, оновлення, вимкнення
/stats	Статистика	Агрегація записів сну
/advice	Поради	Генерація AI-рекомендації
/ai-analysis	Розширений аналіз	Патерни та прогноз якості сну
/admin	Адміністрування	Перегляд системних даних
/audit	Журнал подій	Отримання записів аудиту

Доступ до більшості маршрутів потребує JWT-токен. Після входу користувача сервер формує токен, який надалі передається у запитах до захищених шляхів. Для адміністративних маршрутів додатково перевіряється наявність відповідної ролі. Більш детально модель автентифікації та авторизації представлена на рисунку 2.4.

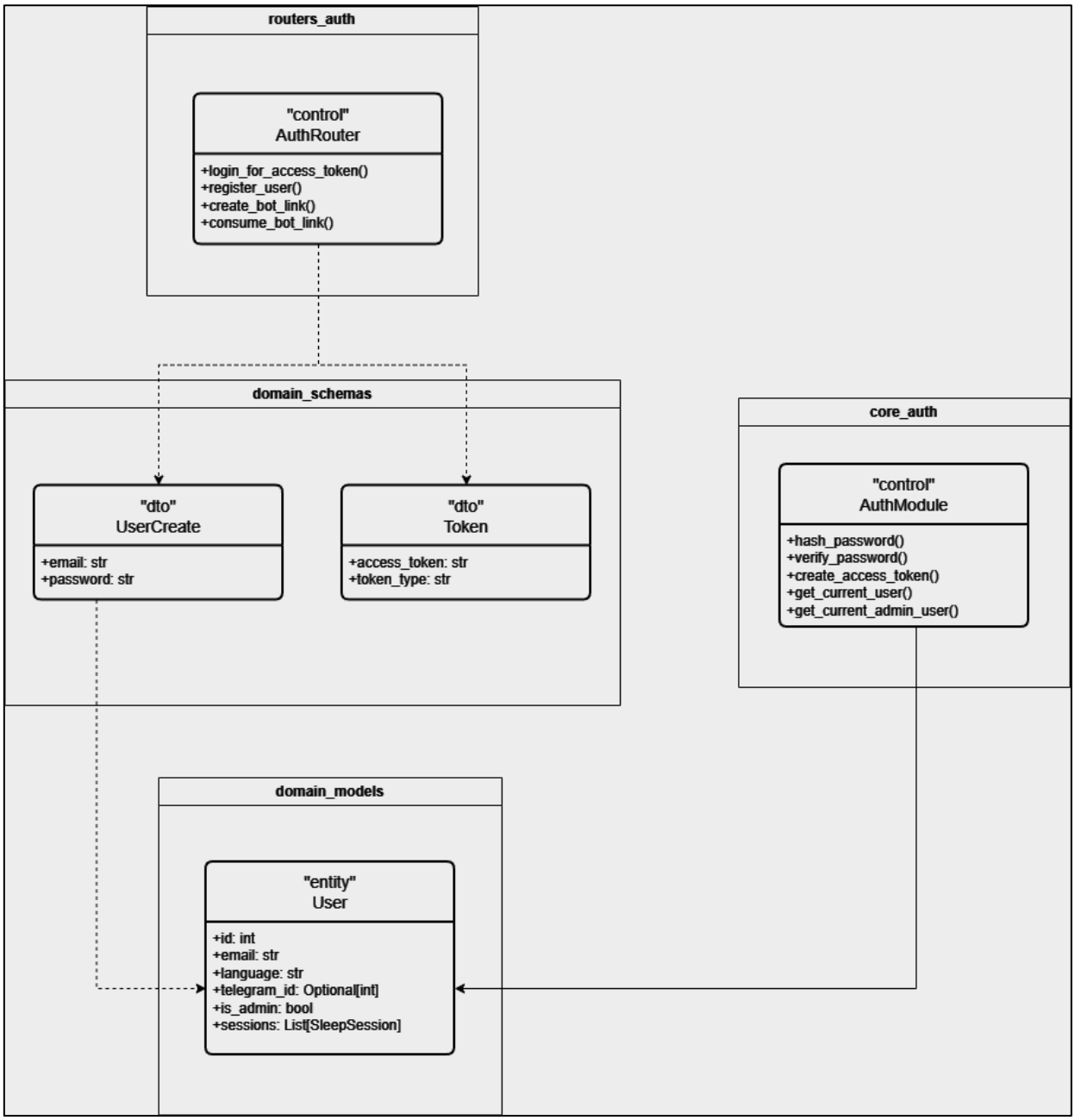


Рисунок 2.4 – Модель автентифікації та авторизації

На рисунку продемонстровано, що модуль авторизації пов’язаний із моделлю користувача, DTO-схемами та маршрутизатором авторизації. Умовний клас AuthModule об’єднує функції хешування пароля, перевірки пароля, створення токена та отримання поточного користувача.

Одним із ключових сценаріїв є створення сесії сну (див. рис. 2.5). Він є показовим, оскільки включає введення даних, перевірку доступу, валідацію, створення запису в базі даних і генерацію ШІ-коментаря. Сценарій починається з введення часу початку, часу завершення, оцінки якості та

нотатки. Після надсилання форми сервер перевіряє JWT-токен, знаходить користувача, виконує валідацію часових меж і створює об’єкт SleepSession. Якщо для користувача задано ціль сну, вона може бути використана під час формування ШІ-фідбеку. Для деталізації варіантів використання сценарію нижче надано рисунок 2.6.

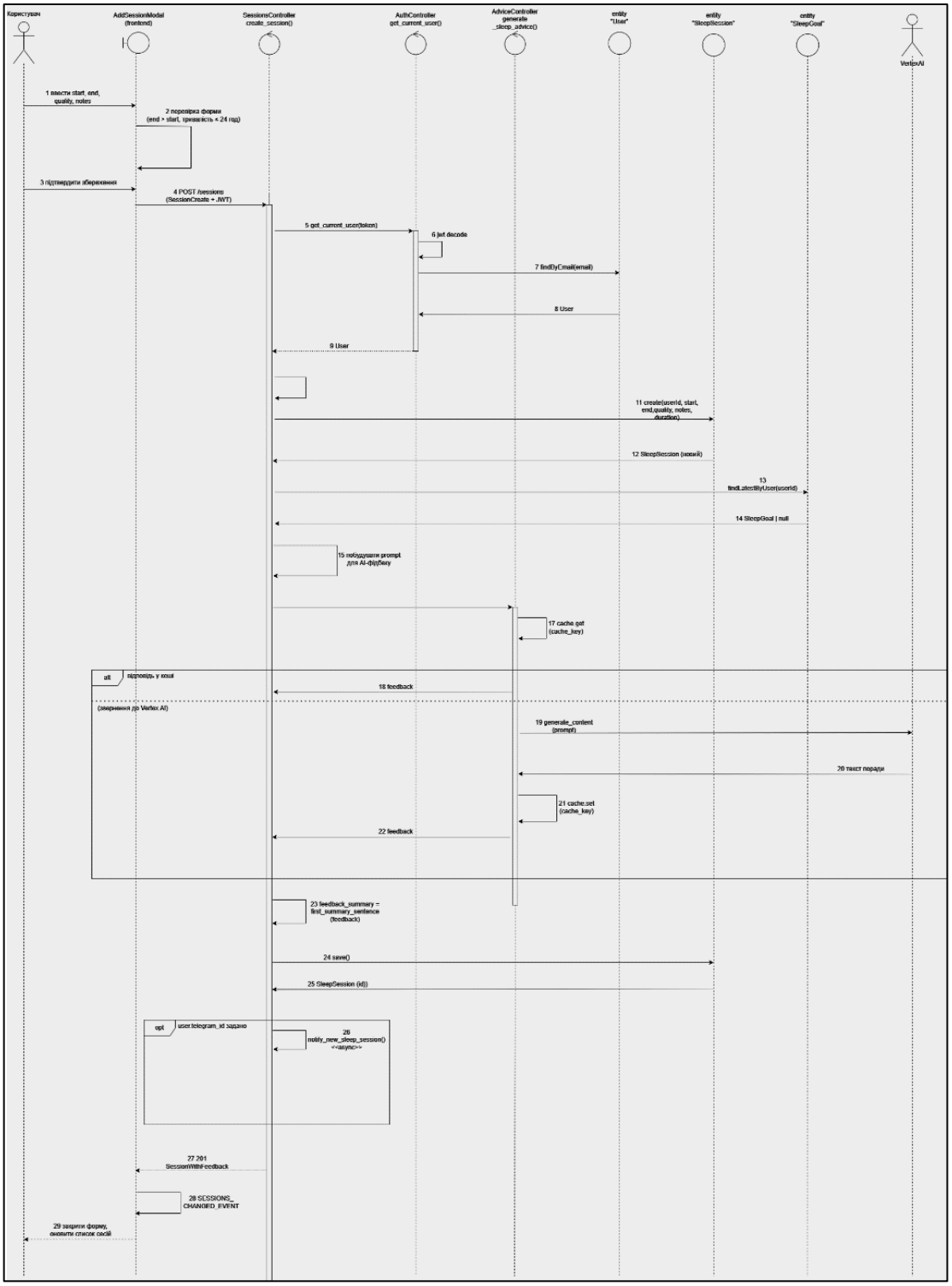


Рисунок 2.5 – Послідовність створення сесії сну

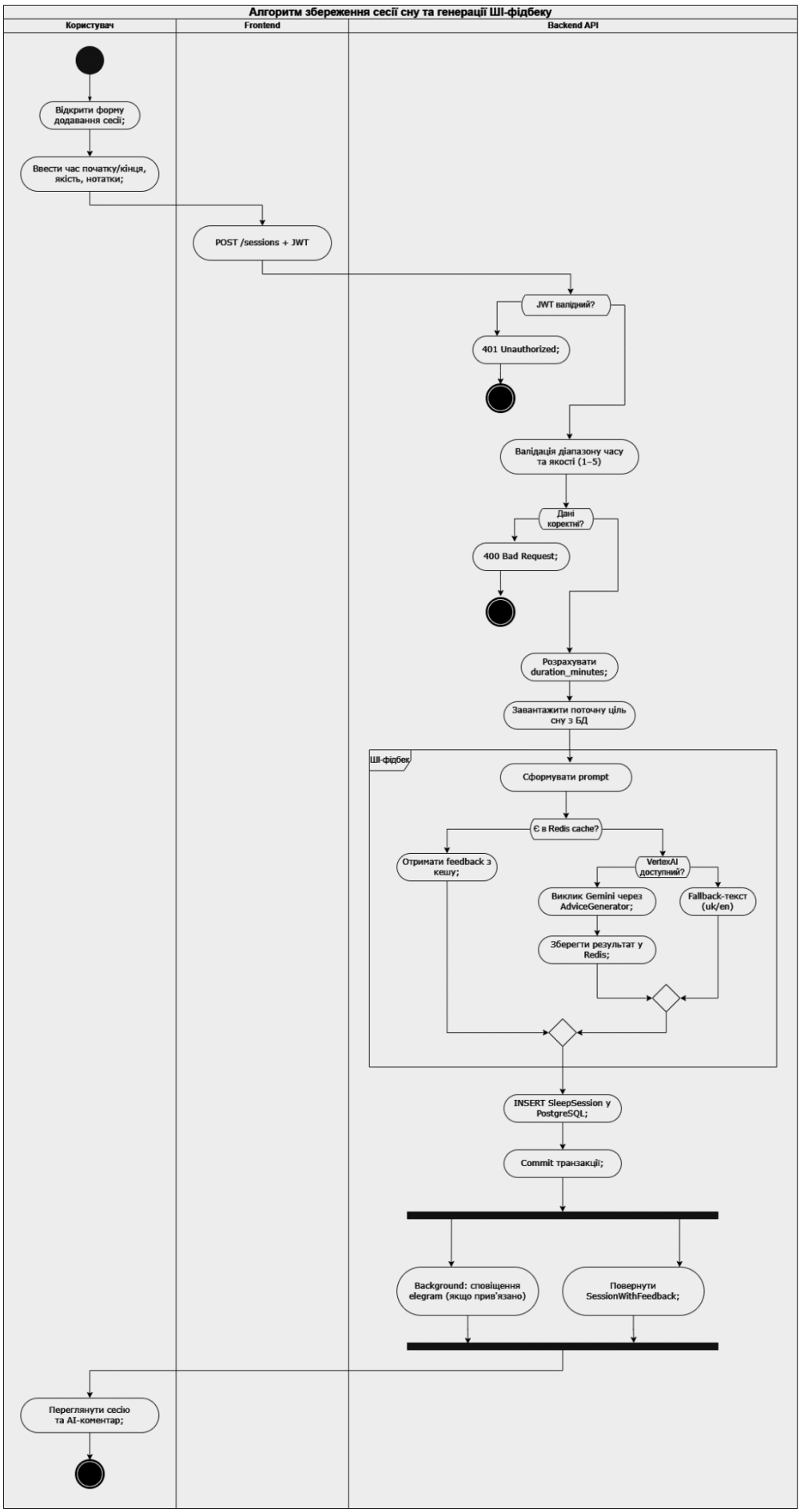


Рисунок 2.6 – Алгоритм збереження сесії сну та генерації ШІ-фідбеку

У разі успішної перевірки система розраховує тривалість сну, формує запит для ШІ-сервісу, отримує або генерує фідбек і лише після цього зберігає сесію в базі даних. Якщо токен недійсний або дані некоректні, запит завершується помилкою авторизації або валідації відповідно. У таблиці 2.5 надано перелік типових помилок API.

Таблиця 2.5 – Групи REST API

Код	**Причина**	**Приклад ситуації**
400	Некоректні вхідні дані	Кінець сну раніше за початок
401	Відсутня авторизація	Немає JWT-токена
403	Недостатньо прав	Звичайний користувач відкриває admin API
404	Дані не знайдено	Запит до неіснуючої сесії
429	Перевищено ліміт	Забагато запитів за короткий проміжок часу
500	Внутрішня помилка	Помилка сервера або зовнішнього сервісу

У результаті REST API визначає правила взаємодії між клієнтськими інтерфейсами та серверною частиною. Через нього виконуються основні операції з даними, перевіряється доступ користувача та запускаються додаткові сервіси, зокрема генерація ШІ-фідбеку. Далі розглядається проєктування Telegram-інтерфейсу як одного з клієнтів цієї API-структури.

2.4 Проєктування Telegram-бота

Telegram-бот спроєктовано як інтерфейс для швидкого виконання основних дій без відкриття вебдодатку. Найбільш важливими для бота є короткі сценарії:

– додавання сесії сну;
– перегляд стислої статистики;
– отримання поради;
– налаштування цілі або нагадування.

Структура бота (див. табл. 2.6) поділяється на окремі обробники, кожен з яких відповідає за власну групу команд.

Таблиця 2.6 – Структура Telegram-бота

Група обробників	**Призначення**	**Приклад дії**
auth	Початок роботи та прив’язка акаунта	Команда «/start»
sleep	Введення сесії сну	Покрокове додавання запису
goals	Робота з цілями	Задання бажаної тривалості сну
reminders	Нагадування	Встановлення часу повідомлення
stats	Коротка статистика	Перегляд середньої тривалості
advice	ШІ-поради	Отримання рекомендації
language	Локалізація	Зміна мови
admin	Адміністративні дії	Перегляд службової інформації

Враховуючи особливості розміщення у месенджері, для введення сесії сну доцільно використовувати покроковий сценарій. Користувач не заповнює велику форму, як у вебінтерфейсі, а послідовно відповідає на повідомлення бота.

Сценарій додавання сесії починається з вибору відповідної команди або кнопки меню. Після цього бот запитує час початку сну, потім час завершення, далі пропонує оцінити якість сну та за бажанням додати нотатку. Після підтвердження введені значення передаються до шляху «/sessions». У відповідь бот отримує створений запис і короткий ШІ-коментар, який відображається користувачу у повідомленні. Стани сценарію описано в таблиці 2.7.

Для зручності користувача в боті передбачаються кнопки швидкого доступу. Вони зменшують кількість ручного введення команд і роблять сценарії більш зрозумілими. Особливо це корисно для повторюваних дій, наприклад додавання сну або перегляду статистики. Повідомлення бота формуються з урахуванням мови, збереженої у профілі користувача. Якщо

користувач змінює мову в одному інтерфейсі, це налаштування враховується і в інших сценаріях взаємодії.

Таблиця 2.7 – Стани сценарію додавання сесії сну в Telegram-боті

Стан	**Дія користувача**	**Результат**
Вибір дії	Натискає кнопку додавання сесії	Бот починає сценарій
Введення початку	Вказує час початку сну	Значення тимчасово зберігається
Введення завершення	Вказує час завершення	Перевіряється часовий діапазон
Оцінка якості	Обирає оцінку	Формується повний набір даних
Нотатка	Додає текст або пропускає	Дані готові до відправлення
Підтвердження	Підтверджує запис	Бот надсилає запит до API

Отже, Telegram-бот спроєктовано як інтерфейс для швидких сценаріїв взаємодії. Його структура базується на окремих групах обробників і покрокових сценаріях введення даних. Основна перевага такого підходу полягає у тому, що користувач може виконувати базові дії без переходу до вебінтерфейсу.

2.5 Проєктування вебінтерфейсу

Вебінтерфейс системи призначений для сценаріїв, які потребують більшого екранного простору:

– перегляд історії;
– аналіз статистики;
– керування записами;
– налаштування цілей та нагадувань;
– виконання адміністративних дій.

Під час проєктування вебінтерфейсу основну увагу приділено структурі сторінок і маршрутам (див. табл. 2.8). Одразу після авторизації користувач

потрапляє до основної панелі, з якої може перейти до записів сну, цілей, нагадувань або адміністративного розділу за наявності відповідних прав.

Таблиця 2.8 – Структура вебінтерфейсу

Елемент	**Призначення**	**Основні дії**
Login	Вхід або створення акаунта	Авторизація, реєстрація
Dashboard	Основний екран користувача	Коротка статистика, огляд останніх даних
Sessions	Історія сесій сну	Додавання, перегляд, видалення
Goals	Цілі сну	Створення або перегляд цільової тривалості
Reminders	Нагадування	Налаштування часу та активності
Admin	Панель адміністратора	Перегляд службових даних
TelegramLinkModal	Прив’язка Telegram	Зв’язування Telegram з акаунтом

Dashboard виконує роль стартової панелі після входу. На ній розмішено короткі показники: останню сесію, середню тривалість сну, оцінку якості та загальну динаміку. Детальніша робота із записами переноситься на сторінку Sessions.

У розділі Sessions користувач працює з історією записів та формою додавання нової сесії. Для користувача важливо бачити не лише сирі дані, а й обчислену тривалість, оцінку якості та, за потреби, короткий ШІ-коментар.

Goals і Reminders використовуються для персональних налаштувань. У Goals користувач задає бажаний діапазон тривалості сну, а у Reminders – час отримання повідомлень. Ці дані надалі використовуються сервером під час аналізу записів і роботи планувальника.

Адміністративна частина повинна бути доступна лише користувачам із відповідною роллю. Її призначення полягає не у звичайному використанні застосунку, а у перегляді службової інформації, контролі користувачів і аналізі загального стану системи.

Клієнтська частина не виконує складних обчислень самостійно. Вона надсилає запити до API, отримує підготовлені дані та відображає їх у зручній

формі. Це стосується як списків з записами, так і статистики, ШІ-рекомендацій та адміністративних даних.

Вебінтерфейс доповнює Telegram-бота сценаріями, що потребують ширшого візуального представлення Його структура орієнтована на розділення за окремими сторінками, що спрощує навігацію та подальшу реалізацію.

2.6 Проєктування ШІ-підсистеми та фонових процесів

ШІ-підсистема призначена для формування текстових рекомендацій на основі даних про сон користувача. Вона не замінює статистику, а доповнює її поясненням у зрозумілій для користувача формі.

Для генерації рекомендації сервер формує запит, до якого можуть входити тривалість останньої сесії, оцінка якості, нотатка користувача, поточна ціль сну та агреговані показники за попередні періоди. При цьому до ШІ-сервісу не повинні передаватися технічні або конфіденційні дані, які не потрібні для формування відповіді.

На рисунку 2.7 показано основні сервіси, які беруть участь у генерації рекомендацій та аналітичній обробці. AdviceGenerator відповідає за створення текстових порад, PatternAnalyzer за пошук повторюваних особливостей у записах, а CacheService за збереження результатів, які можуть бути використані повторно.

Кешування необхідне для зменшення кількості однакових або дуже схожих звернень до VertexAI. Якщо рекомендація для певного набору даних уже була сформована, система може повернути збережений результат. Це прискорює відповідь і зменшує залежність від зовнішнього сервісу.

У випадку недоступності ШІ-сервісу система не повинна завершувати сценарій помилкою, якщо основний запис сну вже може бути створений [13]. Для цього передбачається fallback-відповідь у вигляді короткого стандартного

тексту українською або англійською мовою. Завдяки цьому користувач усе одно отримує результат основної дії.

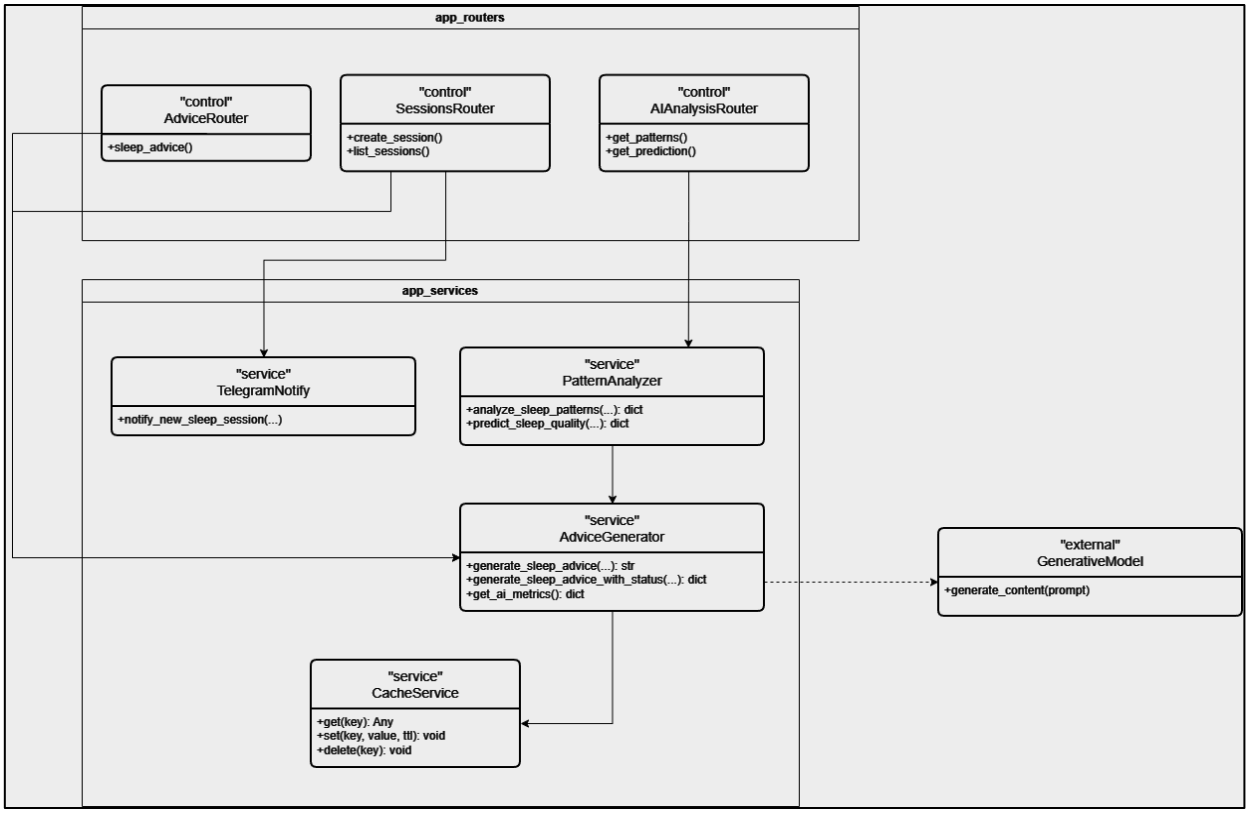


Рисунок 2.7 – Підсистема ШІ-сервісів і кешування

Окремим фоновим процесом є планувальник нагадувань (див. рис. 2.8). На відміну від звичайних API-запитів, він запускається не за прямою дією користувача, а автоматично через певний проміжок часу. Його завдання полягає в тому щоб перевірити активні нагадування та надіслати повідомлення користувачам у потрібний момент.

Алгоритм роботи планувальника складається з вибірки активних нагадувань, перевірки локального часу користувача, порівняння з поточним часом і контролю поля last_sent_date. Останнє потрібне для того, щоб одне й те саме нагадування не було надіслане користувачу декілька разів за день.

Зазначені сервіси відповідають за пояснення та рекомендації, кешування підвищує швидкодію, а планувальник забезпечує автоматичне надсилання нагадувань. Разом ці елементи роблять систему не лише сховищем записів, а й інструментом регулярної взаємодії з користувачем.

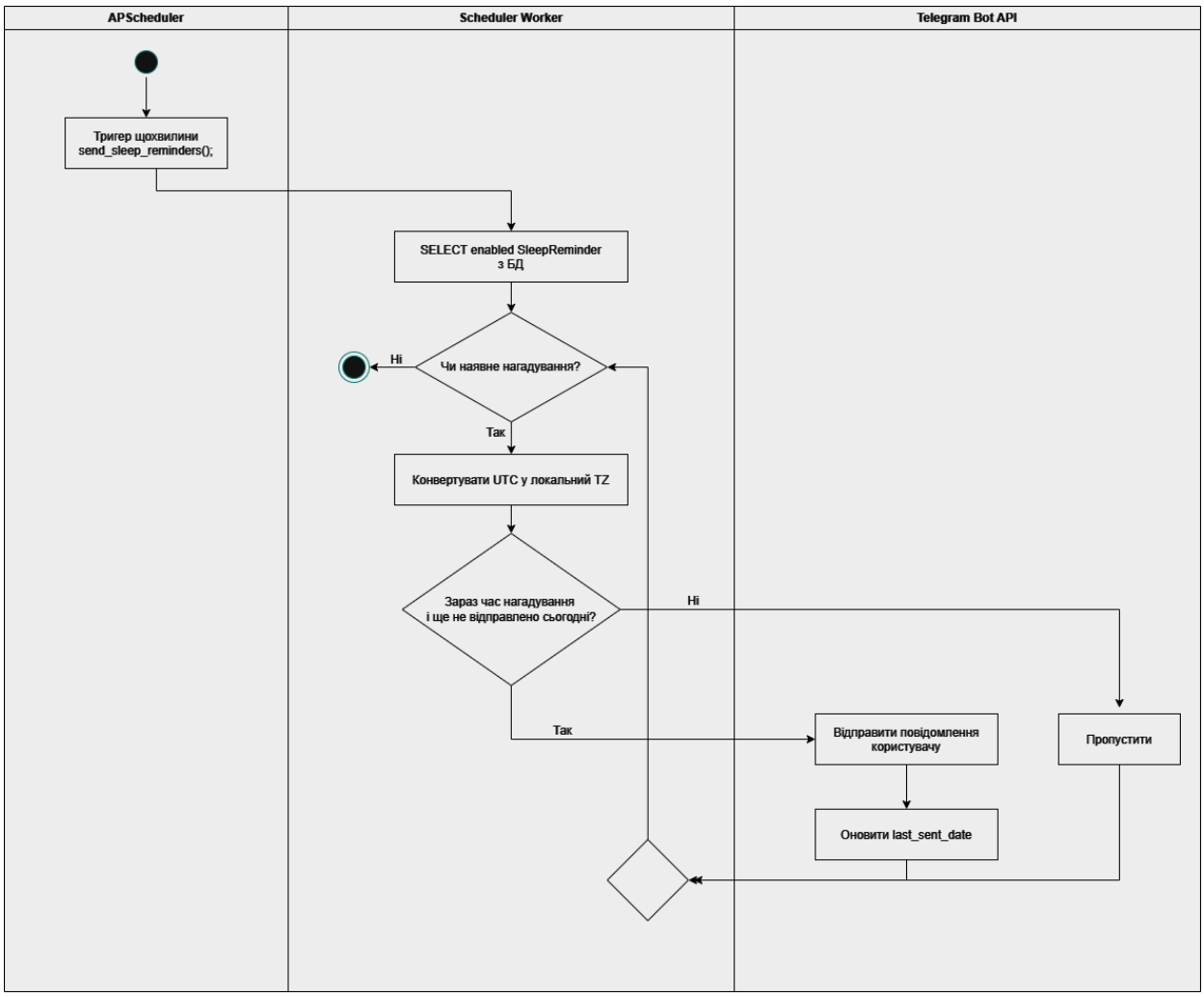


Рисунок 2.8 – Алгоритм надсилання нагадувань користувачу

Спроєктована хмарна архітектура, доменна модель даних та специфікація REST API сформували вичерпну конструктивну базу, що дозволяє перейти до безпосередньої програмної реалізації компонентів системи та їхнього комплексного тестування.

3 РЕАЛІЗАЦІЯ ТА ТЕСТУВАННЯ

Розділ присвячено практичній реалізації спроєктованої системи, організації середовища запуску та перевірці працездатності основних сценаріїв.

3.1 Середовище розробки та організація проєкту

Реалізація системи виконувалася після визначення технологічного стеку, який відповідає архітектурним рішенням другого розділу. Для розробки елементів системи було обрано наступний стек:

- для серверної частини використано Python 3.12, FastAPI, Pydantic/SQLModel та Alembic;
- Telegram-інтерфейс реалізовано засобами фреймворку Aiogram 3;
- вебклієнт розроблено за допомогою React 18, Vite та Rechart;
- дані зберігаються у PostgreSQL для локальної перевірки та Supabase для хмарної робочої версії;
- для кешування обрано Redis;
- інтеграцію VertexAI реалізовано сервісами Google Cloud.

Для локальної розробки створено конфігурацію Docker Compose, яка одночасно запускає основні компоненти системи. Параметри передаються через змінні середовища без жорсткого кодування секретів. Схема локального розгортання наведена на рисунку 3.1.

У хмарному середовищі frontend, backend, bot та Redis розміщуються як Cloudflare Containers.

Порівняння способів розгортання основних компонентів системи наведено в таблиці 3.1.

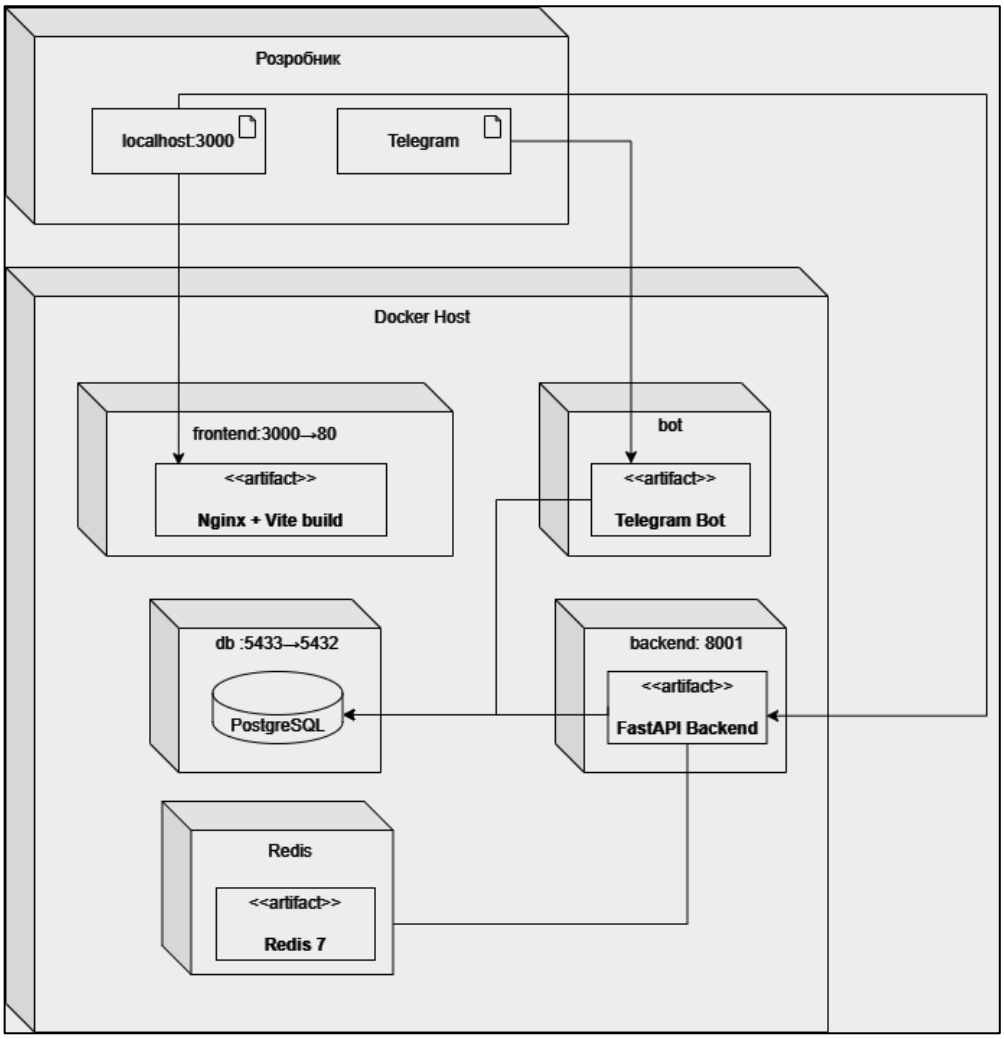


Рисунок 3.1 – Локальне розгортання системи через Docker Compose

Таблиця 3.1 – Порівняння локального та хмарного середовищ

Компонент	**Локально**	**Хмарне середовище**
Frontend	Docker, порт 3000	Cloudflare Containers
Backend	Docker, порт 8001	Cloudflare Containers
Telegram-бот	Контейнер	Cloudflare Containers
Redis	Контейнер redis	Cloudflare Containers
VertexAI	Google Cloud API	Google Cloud API
PostgreSQL	Контейнер db	Supabase

Підготовлене середовище дозволяє розробляти, локально перевіряти та розгортати систему в хмарі без зміни логіки застосунку. Далі розглянуто реалізацію серверної частини, яка є спільним ядром для обох клієнтських інтерфейсів.

3.2 Реалізація серверної частини

Серверна частина об’єднує конфігурацію, механізми безпеки, доменну логіку та REST API на FastAPI. Централізовані налаштування зосереджено в класі Settings, який зчитує параметри зі змінних середовища, а саме: підключення до БД; JWT-секрет; GCP; Redis; CORS. Фрагмент наведено на рисунку 3.2.

```
class Settings(BaseSettings):
    APP_NAME: str = "Sleep Tracker"
    JWT_SECRET: str = "dev-secret"
    DATABASE_URL: str = "postgresql+psycopg2://..."
    GCP_PROJECT: str
    REDIS_URL: str | None = None
    CORS_ORIGINS: str = "http://localhost:3000,..."
    model_config = SettingsConfigDict(env_file=str(_CONFIG_ROOT / ".env"))
```

Рисунок 3.2 – Фрагмент класу Settings

Автентифікація реалізована через bcrypt-хешування паролів та JWT. Фрагмент коду автентифікації наведено на рисунку 3.3. Систему авторизації утворюють такі елементи:

- модуль «app/core/auth.py» містить функції для створення і отримання токена;
- роутер «app/routers/auth.py» включає в себе шляхи «/auth/register» та «/auth/token»;
- у «app/main.py» підключено middleware з захисними HTTP-заголовками та логуванням.

Доменні сутності реалізовано в «app/domain/models.py» за допомогою SQLModel, а зміни схеми міграціями Alembic. Сутність SleepSession містить час сну, якість за п’ятибальною шкалою, нотатки та duration_minutes (див. рис. 3.4).

```
def get_current_user(
    session: Session = Depends(get_session),
    credentials: HTTPAuthorizationCredentials | None = Depends(bearer),
) -> User:
    if credentials is None:
        raise HTTPException(status_code=401, detail="Not authenticated")
    payload = jwt.decode(credentials.credentials, settings.JWT_SECRET, ...)
    user = session.exec(select(User).where(User.email == payload.get("sub"))).first()
    if not user:
        raise HTTPException(status_code=401, detail="User not found")
    return user
```

Рисунок 3.3 – Фрагмент JWT-автентифікації

```
class SleepSession(SQLModel, table=True):
    user_id: int = Field(foreign_key="user.id")
    start: datetime
    end: datetime
    quality: Optional[int] = Field(default=None, ge=1, le=5)
    duration_minutes: int = 0
```

Рисунок 3.4 – Фрагмент моделі SleepSession

REST API інкапсулює бізнес-логіку та слугує єдиним контрактом для Telegram-бота і вебінтерфейсу. Ключові реалізації:

– прив’язка Telegram виконана у «/auth/bot-link» та «auth/bot-link/consume» використовуються для генерації одноразового коду та заповнення поля User.telegram_id;
– валідація і розрахунок тривалості сесії сну реалізовано за шляхом «POST /sessions» (див. рис. 3.5);
– цілі, нагадування, статистику і експорт спроєктовано у модулях goals.py, reminders.py, stats.py і export.py відповідно;
– виклик LLM-моделі реалізовано у файлі advice_generator.py (див. рис. 3.6).

```
class SleepSession(SQLModel, table=True):
    user_id: int = Field(foreign_key="user.id")
    start: datetime
    end: datetime
    quality: Optional[int] = Field(default=None, ge=1, le=5)
    duration_minutes: int = 0
```

Рисунок 3.5 – Фрагмент «POST /sessions» з ШІ-фідбеком

```
def generate_sleep_advice(prompt: str, cache_key: str | None = None, language: str = "uk") ->
str:
    if cache_key and (cached := cache.get(cache_key)):
        return cached
    init_vertex_client()
    model = GenerativeModel(settings.GEMINI_MODEL or "gemini-1.5-flash")
    response = model.generate_content(prompt)
    text = strip_ai_markdown(response.text or "")
    if cache_key:
        cache.set(cache_key, text, ttl=3600)
    return text or _FALLBACK_ADVICE_UK
```

Рисунок 3.6 – Фрагмент інтеграції з VertexAI

Для перевірки коректності роботи маршруту створення сесії сну було використано інтерактивну документацію Swagger, яка автоматично формується FastAPI на основі описаних маршрутів і DTO-схем. Такий спосіб перевірки дозволяє швидко проаналізувати структуру запиту, статус відповіді та набір даних, які сервер повертає після успішного створення запису.

На рисунку 3.7 наведено приклад відповіді API, у якій відображено створену сесію сну, розраховану тривалість і текстовий фідбек, сформований системою.



Рисунок 3.7 – Приклад відповіді API при створенні сесії

Модуль планувальника нагадувань реалізовано на APScheduler. Зведена відповідність функцій системи їх реалізації наведена в таблиці 3.2.

Таблиця 3.2 – Відповідність функцій системи реалізації API

Функціональність	**Шлях**	**Модуль**
Реєстрація та автентифікація користувача	«/auth/register», «/auth/token»	«auth.py»
Прив’язка Telegram-акаунта	«/auth/bot-link»	«auth.py»
Облік сесій сну	«/sessions»	«sessions.py»
Цілі сну	«/goals/*»	«goals.py»
Нагадування про сон	«/reminders/*»	«reminders.py»
Статистика сну	«/stats/*»	«stats.py»
AI-поради тааналіз патернів	«/advice», «/ai-«analysis/*», «/coach/report»	«advice_generator.py»
Локалізація	«/language/*»	«language.py»

Отже, серверна частина реалізує повний набір функцій, включно з інтеграцією VertexAI. Далі розглянуто клієнтські інтерфейси, що використовують цей API.

3.3 Реалізація клієнтських інтерфейсів

Система має два канали представлення Telegram-бот і вебінтерфейс. Обидва є тонкими клієнтами до спільного REST API і не дублюють бізнес-логіку на стороні клієнта.

3.3.1 Telegram-бот

Обраний канал взаємодії реалізовано як клієнтську частину що відповідає за сценарій діалогу, формування клавіатур та обробку вводу користувача. Для передавання даних бот виконує HTTP-запити до backend за адресою BACKEND_URL. Фрагмент відправлення даних про сон на сервер наведено на рисунку 3.8.

```
headers = {"Authorization": f"Bearer {token}"}
data = {"start": start, "end": end, "quality": quality}
r = await asyncio.to_thread(
    requests.post, f"{BACKEND_URL}/sessions", json=data, headers=headers
)
if r.status_code == 201:
    feedback = r.json().get("feedback")
    await callback.message.answer(format_sleep_saved_html(..., feedback))
```

Рисунок 3.8 – Надсилання сесії сну на сервер

Telegram-інтерфейс не містить власного шару збереження даних, а використовує централізований API, що відповідає архітектурному підходу.

Користувацький сценарій взаємодії з ботом починається з головного меню. Зовнішній вигляд стартового екрана, який відкривається командою «/start», наведено на рисунку 3.9.

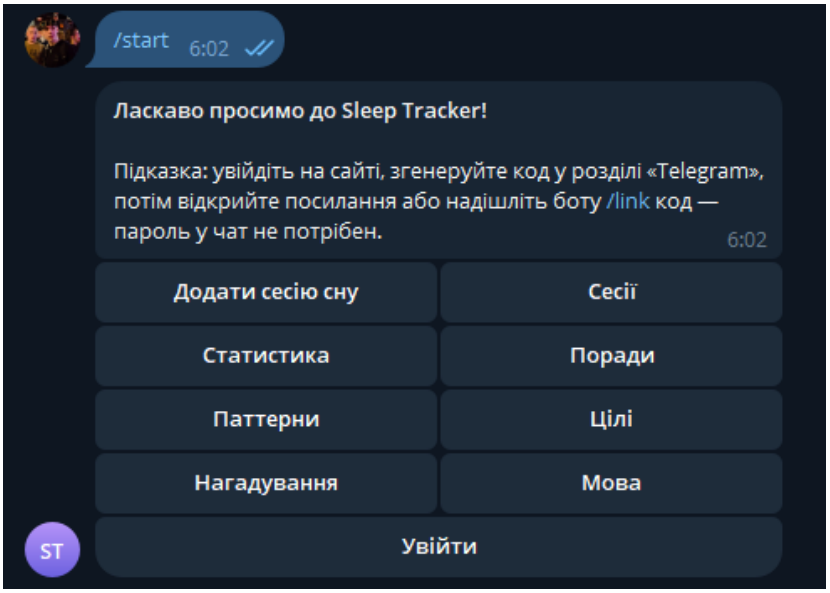


Рисунок 3.9 – Головне меню Telegram-бота

Основні функції згруповано в одному вікні, що спрощує навігацію та зменшує кількість кроків до ключових дій.

Під час додавання сесії сну бот використовує покроковий сценарій введення даних. Приклад вибору дати та часу наведено на рисунку 3.10.

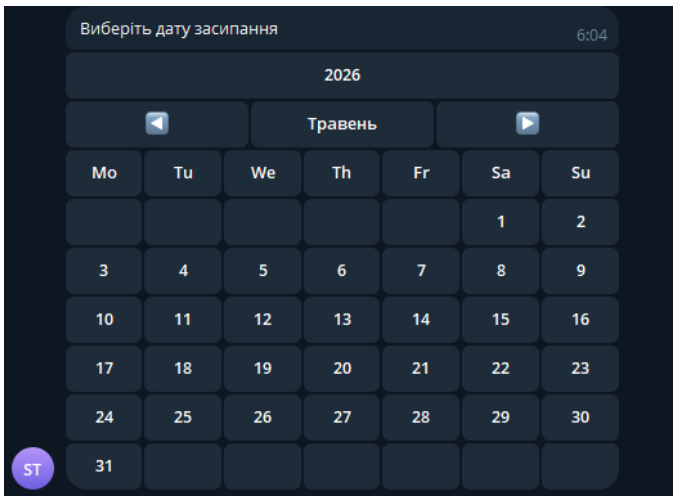


Рисунок 3.10 – Вибір дати та часу сну в Telegram-боті

Інтерфейс введення структуровано та орієнтовано на мінімізацію помилок користувача під час заповнення даних. Після збереження сесії користувач отримує текстовий зворотний зв’язок на основі аналізу даних. Приклад відповіді показано на рисунку 3.11.

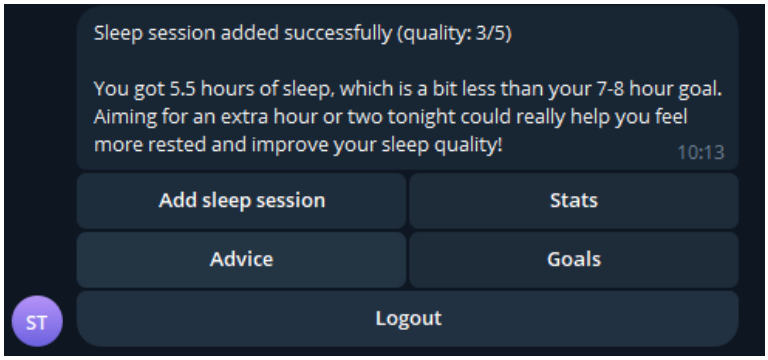


Рисунок 3.11 – Відповідь ШІ після збереження сесії англійською мовою

Telegram-бот забезпечує не лише фіксацію даних, а й оперативне представлення аналітичного результату в тому самому каналі взаємодії.

3.3.2 Вебінтерфейс

Вебінтерфейс реалізовано як SPA-застосунок на React. Сторінка входу поєднує реєстрацію та авторизацію, а після успішного входу токен доступу

використовується для запитів до API. Зовнішній вигляд сторінки входу наведено на рисунку 3.12.

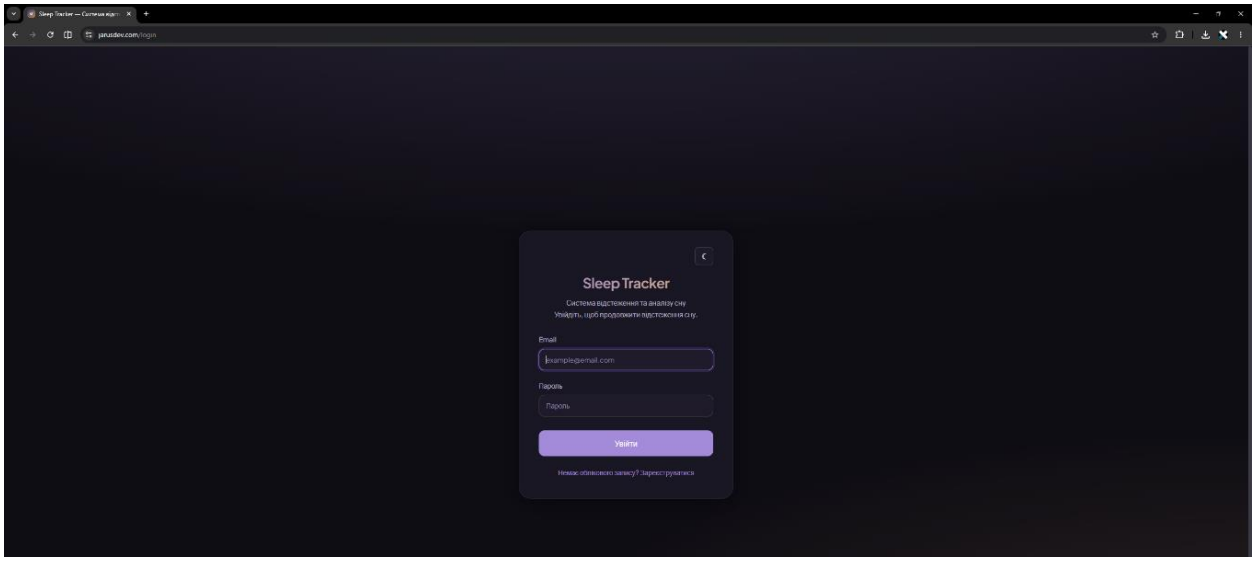


Рисунок 3.12 – Сторінка входу вебінтерфейсу

Вебклієнт підтримує єдиний сценарій доступу до системи, синхронізований із серверними механізмами автентифікації.

На головній сторінці вебінтерфейсу відображаються ключові показники та графік статистики сну. Приклад наведено на рисунку 3.13.

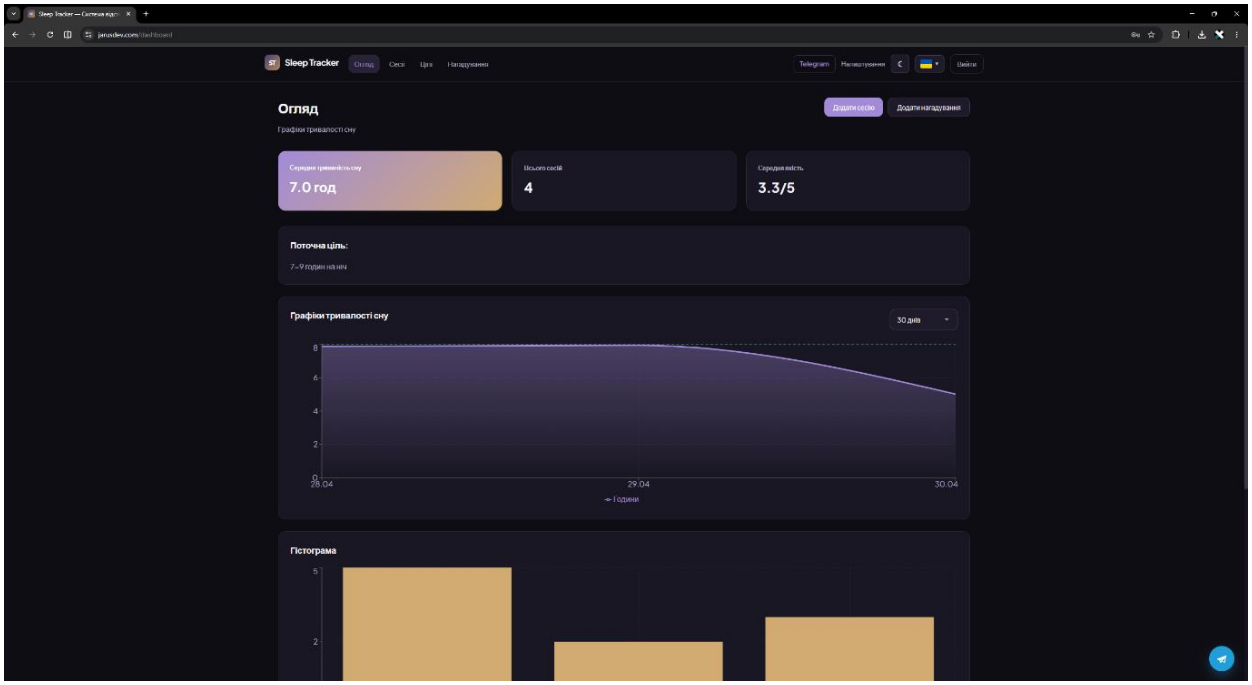


Рисунок 3.13 – Головна сторінка вебінтерфейсу

Інтерфейс орієнтований на візуальне представлення динаміки сну та швидкий доступ до аналітичної інформації.

Для керування даними користувача в системі реалізовано сторінки сесій, цілей, нагадувань та модальне вікно прив’язки Telegram. Відповідний приклад наведено на рисунку 3.14.

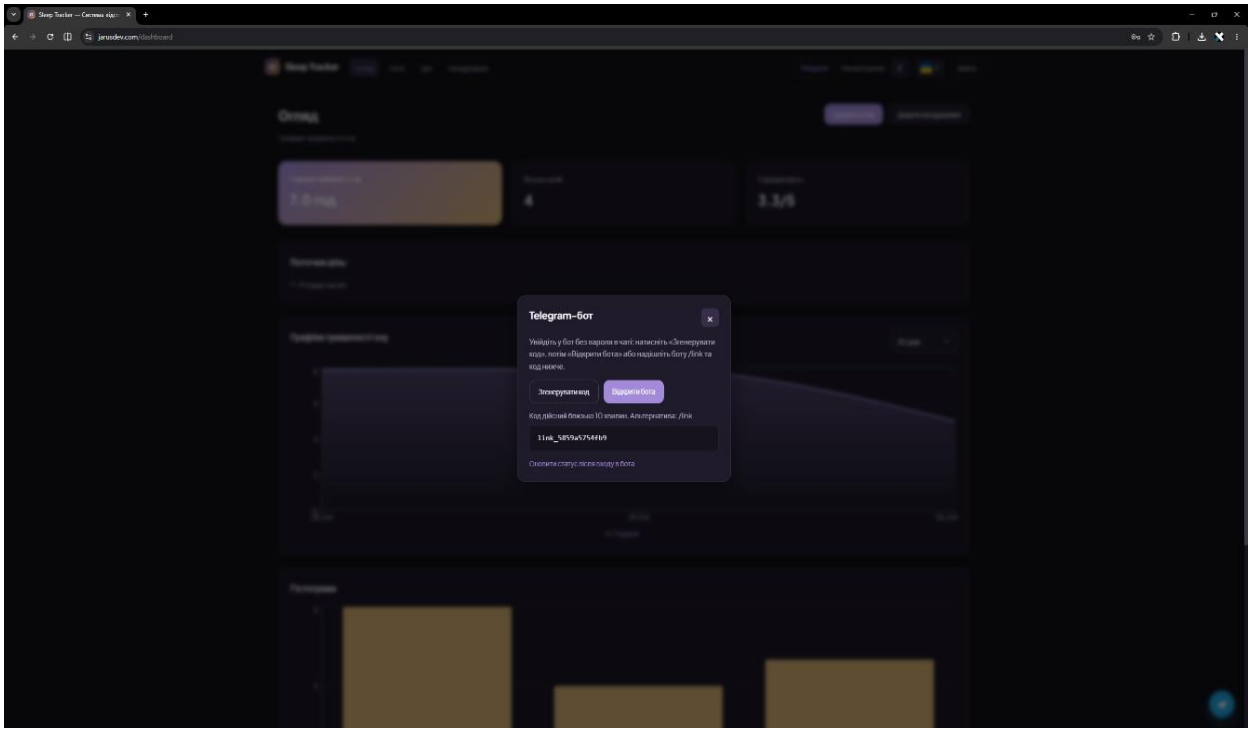


Рисунок 3.14 – Модальне вікно прив’язки Telegram

Окремий режим роботи передбачено для адміністраторів системи. Візуальне представлення адміністративної панелі зображено на рисунку 3.15.

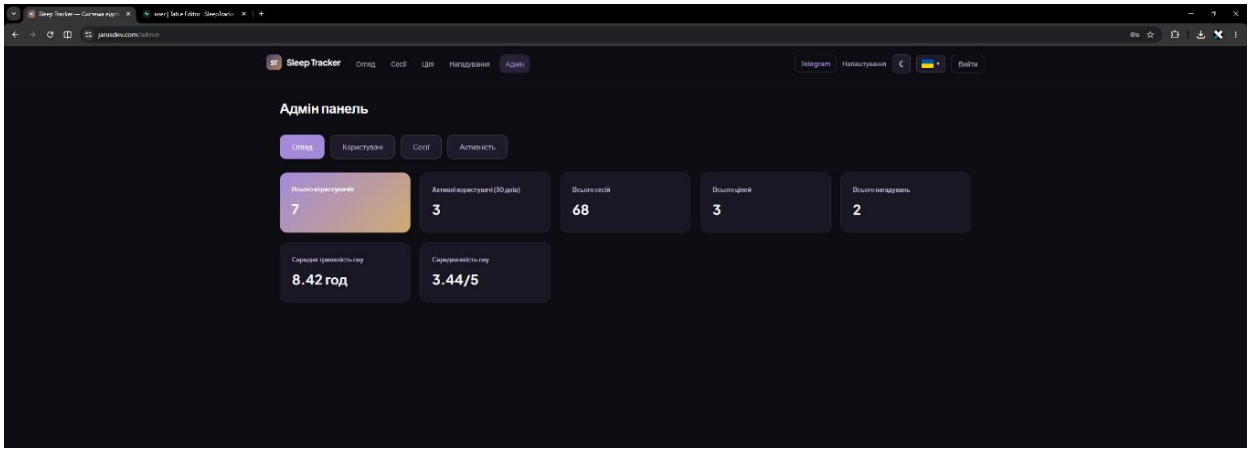


Рисунок 3.15 – Адміністративна панель

Таким чином, реалізовані клієнтські інтерфейси забезпечують повноцінну взаємодію із системою в двох каналах доступу та використовують єдину серверну модель обробки даних. Далі розглянуто результати тестування, що підтверджують коректність реалізації.

3.4 Тестування

Перевірка реалізації виконувалася за допомогою автоматизованого тестування на рівні серверного API та окремих сервісів. Метою тестування було підтвердити коректність основних користувацьких сценаріїв, стабільність обробки даних та працездатність ключових механізмів безпеки. Для цього у проєкті використано інструменти pytest та pytest-cov, а також тестовий сценарій у GitHub Actions.

Ізоляція тестового середовища реалізована шляхом створення in-memory бази SQLite, підміни залежностей доступу до бази даних та використання тестових сутностей, наприклад test_user. Фрагмент конфігурації тестового середовища наведено на рисунку 3.16.

```
@pytest.fixture
def test_user(test_session):
    user = User(
        email="test@example.com",
        hashed_password=hash_password("testpassword123"),
        is_verified=True,
        is_admin=False,
        language="uk"
    )
    test_session.add(user)
    test_session.commit()
    test_session.refresh(user)
    return user
```

Рисунок 3.16 – Фрагмент pytest

Описаний підхід дозволяє перевіряти логіку незалежно від інфраструктури та зовнішніх сервісів.

Усього в межах проєкту реалізовано 110 автоматизованих тестів, з яких 96 належать до інтеграційних, а 14 до модульних. Для ілюстрації в таблиці 3.3 наведено вибрані приклади тестів із коротким описом перевіюваних сценаріїв.

Таблиця 3.3 – Приклади автоматизованих тестів

Тип тесту	**Назва тесту**	**Короткий опис**
integration	test_create_session_success	Перевіряє успішне створення сесії сну через API та наявність поля feedback у відповіді.
integration	test_bot_link_create_and_consume	Перевіряє повний цикл прив’язки Telegram-акаунта: генерація коду, погашення, оновлення telegram_id.
integration	test_advice_success	Перевіряє формування персоналізованої поради на основі даних сесій сну з використанням mock AI-виклику.
unit	test_validate_time_range_too_long	Перевіряє валідацію часового інтервалу та відхилення надто довгої сесії сну.
unit	test_cache_expired_entry_returns_none	Перевіряє коректну поведінку кешу.
unit	test_start_scheduler_registers_jobs	Перевіряє, що під час запуску планувальника реєструються потрібні фонові задачі.

Як приклад інтеграційного тесту розглянуто сценарій створення сесії сну, у якому перевіряються статус відповіді, структура JSON та коректність розрахованої тривалості сну. Відповідний фрагмент тесту наведено на рисунку 3.17.

```
def test_create_session_success(client, auth_headers):
    data = {"start": "2025-01-01T22:00:00", "end": "2025-01-02T07:00:00",
            "quality": 4, "notes": "Good sleep"}
    response = client.post("/sessions", json=data, headers=auth_headers)
    assert response.status_code == 201
    assert "feedback" in response.json()
    assert response.json()["session"]["duration_minutes"] == 540
```

Рисунок 3.17 – Фрагмент тесту test_create_session_success

Тест перевіряє не лише успішне створення запису, а й ключові атрибути відповіді API, що безпосередньо впливають на клієнтські сценарії.

Поряд із базовими сценаріями перевірялися і граничні випадки:

– валідація некоректної оцінки якості сну;
– доступ без токена;
– цикл прив’язки Telegram-акаунта;
– генерація AI-порад у режимі mock без звернення до зовнішнього сервісу.

Результат виконання повного тестового набору наведено на рисунку 3.18.

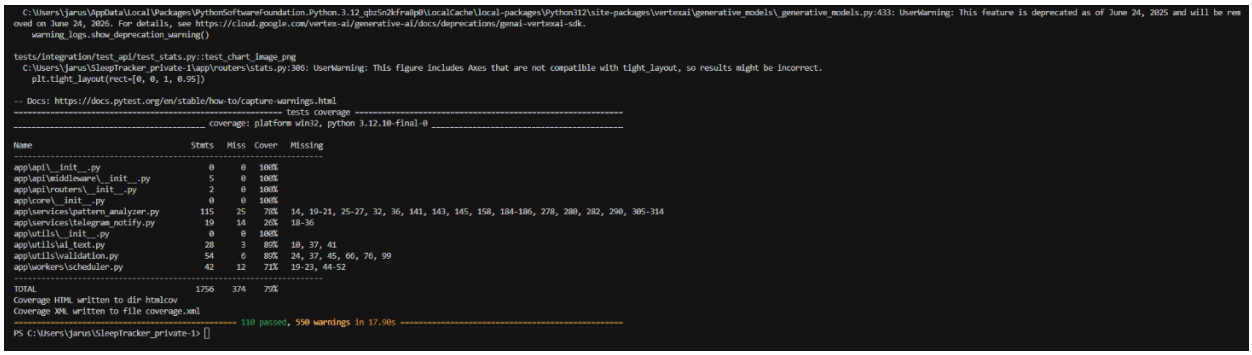


Рисунок 3.18 – Результат pytest у терміналі

Для контролю покриття коду та стабільності збірки в CI використовується звіт htmlcov і GitHub Actions (див. рис. 3.19 та 3.20).

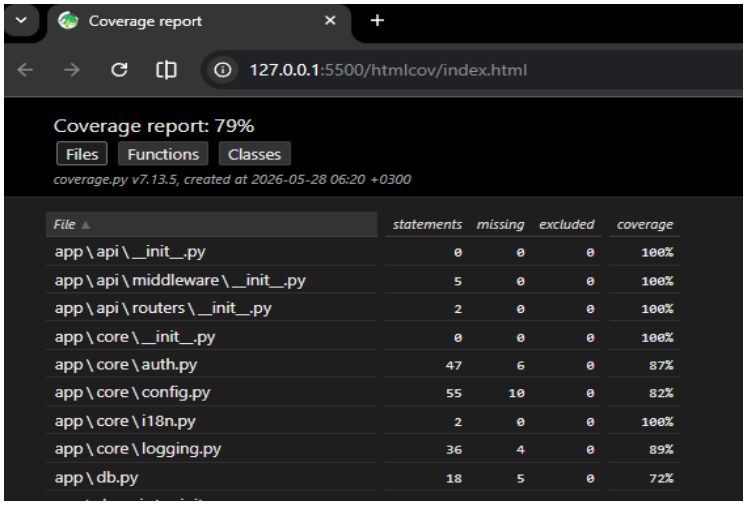
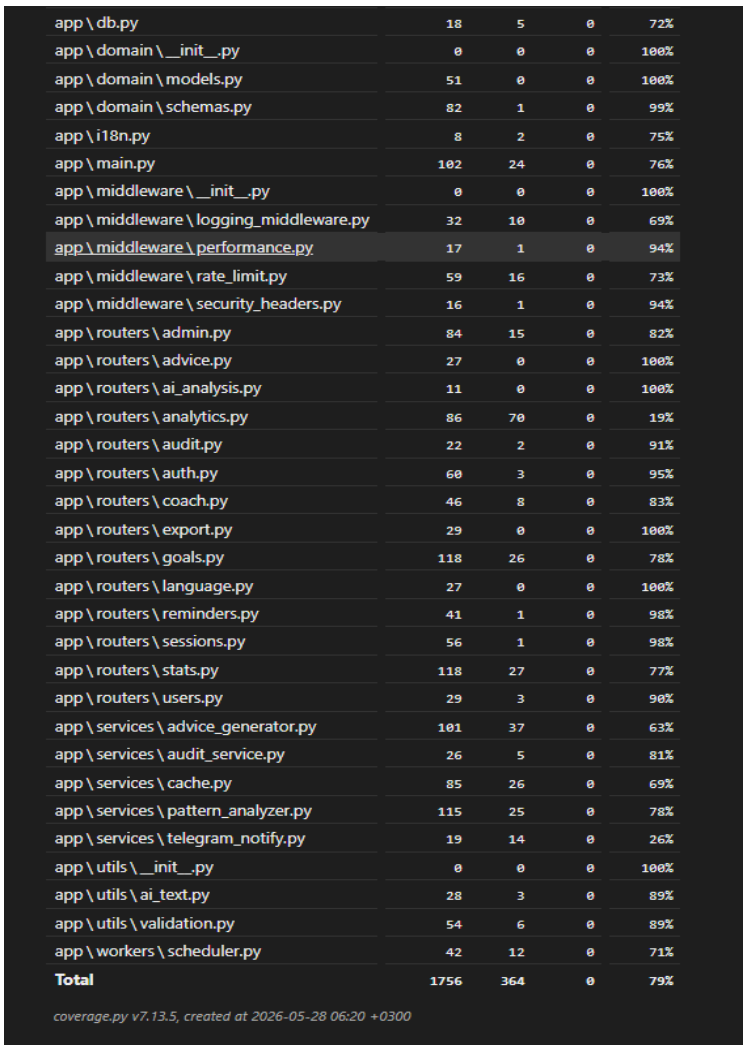


Рисунок 3.19 – Звіт покриття (частина 1)



app \ db.py	18	5	0	72%
app \ domain \ __init__.py	0	0	0	100%
app \ domain \ models.py	51	0	0	100%
app \ domain \ schemas.py	82	1	0	99%
app \ i18n.py	8	2	0	75%
app \ main.py	102	24	0	76%
app \ middleware \ __init__.py	0	0	0	100%
app \ middleware \ logging_middleware.py	32	10	0	69%
app \ middleware \ performance.py	17	1	0	94%
app \ middleware \ rate_limit.py	59	16	0	73%
app \ middleware \ security_headers.py	16	1	0	94%
app \ routers \ admin.py	84	15	0	82%
app \ routers \ advice.py	27	0	0	100%
app \ routers \ ai_analysis.py	11	0	0	100%
app \ routers \ analytics.py	86	70	0	19%
app \ routers \ audit.py	22	2	0	91%
app \ routers \ auth.py	60	3	0	95%
app \ routers \ coach.py	46	8	0	83%
app \ routers \ export.py	29	0	0	100%
app \ routers \ goals.py	118	26	0	78%
app \ routers \ language.py	27	0	0	100%
app \ routers \ reminders.py	41	1	0	98%
app \ routers \ sessions.py	56	1	0	98%
app \ routers \ stats.py	118	27	0	77%
app \ routers \ users.py	29	3	0	90%
app \ services \ advice_generator.py	101	37	0	63%
app \ services \ audit_service.py	26	5	0	81%
app \ services \ cache.py	85	26	0	69%
app \ services \ pattern_analyzer.py	115	25	0	78%
app \ services \ telegram_notify.py	19	14	0	26%
app \ utils \ __init__.py	0	0	0	100%
app \ utils \ ai_text.py	28	3	0	89%
app \ utils \ validation.py	54	6	0	89%
app \ workers \ scheduler.py	42	12	0	71%
Total	1756	364	0	79%

coverage.py v7.13.5, created at 2026-05-28 06:20 +0300

Рисунок 3.20 – Звіт покриття (частина 1)

Отримані результати свідчать про коректність основних API-контрактів, стабільність ключових сценаріїв і відповідність реалізації спроєктованій архітектурі.

ВИСНОВКИ

У кваліфікаційній роботі було розроблено багаторівневу хмарну систему для обліку та аналізу режиму сну користувача з використанням Telegram-бота, вебінтерфейсу, серверної частини на FastAPI та інтеграції VertexAI. У результаті виконання роботи в повному обсязі досягнуто поставленої мети: створено програмне рішення, яке забезпечує реєстрацію користувачів, авторизацію, прив’язку Telegram-акаунта, додавання сесій сну, перегляд історії, формування статистики, налаштування цілей і нагадувань, а також генерацію персоналізованих рекомендацій.

Було досліджено предметну область, пов’язану з обліком сну та використанням чат-ботів як каналу взаємодії з користувачем. Розглянуто комерційні трекери сну, боти для формування звичок і технічні аналоги, побудовані на основі Telegram-ботів та серверного API. Аналіз показав, що наявні рішення зазвичай покривають лише окремі функції: мобільний моніторинг сну, прості нагадування або технічну взаємодію з API. Водночас вони не поєднують у межах однієї системи Telegram-інтерфейс, вебінтерфейс, централізовану базу даних, статистику, нагадування та AI-рекомендації. Це обґрунтувало доцільність розроблення власної системи.

Було спроєктовано архітектуру та основні компоненти програмного продукту. Систему поділено на рівень представлення, прикладний рівень, бізнес-логіку, рівень даних і зовнішні сервіси. Такий підхід дав змогу відокремити Telegram-бота та вебінтерфейс від серверної логіки, зосередивши обробку даних у FastAPI backend. Було визначено основні сутності предметної області: користувач, сесія сну, ціль сну, нагадування та журнал аудиту. Також спроєктованою REST API, сценарії роботи Telegram-бота, структуру вебінтерфейсу, механізм авторизації, підсистему AI-порад, кешування та фонові процеси для надсилання нагадувань.

Також було описано реалізацію системи та перевірку її працездатності. Серверну частину реалізовано засобами Python, FastAPI, SQLModel, Alembic і

Pydantic. Telegram-бот створено на основі Aiogram із використанням покрокових сценаріїв введення даних. Вебінтерфейс реалізовано за допомогою React і орієнтовано на перегляд історії сну, статистики, налаштувань профілю та адміністративних функцій. Для збереження даних використано PostgreSQL/Supabase, для кешування – Redis, для генерації рекомендацій – VertexAI. Розгортання системи передбачає використання контейнеризованих компонентів, що спрощує перенесення між локальним і хмарним середовищем.

Під час тестування було перевірено ключові сценарії роботи системи, зокрема реєстрацію та авторизацію користувача, створення сесій сну, валідацію даних, обмеження доступу без токена, прив’язку Telegram-акаунта, роботу статистики та генерацію ШІ-порад у тестовому режимі. Наявність модульних та інтеграційних тестів підтверджує коректність основних API-контрактів і стабільність реалізації в межах заявленого функціоналу.

Практична цінність розробленої системи полягає у тому, що користувач отримує єдиний інструмент для регулярного самоспостереження за режимом сну. Telegram-бот забезпечує швидке внесення даних і отримання нагадувань, тоді як вебінтерфейс надає зручний доступ до історії, статистики та налаштувань. Інтеграція VertexAI розширює функціональність системи, оскільки дозволяє не лише накопичувати записи, а й формувати зрозумілі текстові рекомендації на основі даних користувача.

Подальший розвиток системи може передбачати інтеграцію з фітнес-пристроями, розширення аналітики, додавання більш гнучких налаштувань нагадувань, покращення адміністративної панелі, реалізацію мобільної версії вебінтерфейсу та поглиблення ШІ-аналізу на основі більшої кількості історичних даних. Отже, розроблена система є функціонально завершеним прототипом багаторівневого хмарного застосунку, який може бути розширений і адаптований до реального використання.

ПЕРЕЛІК ПОСИЛАНЬ

1. Irish L. A., Kline C. E., Gunn H. E., Buysse D. J. et al. The role of sleep hygiene in promoting public health: A review of empirical evidence. *Sleep Medicine Reviews*. 2015. Vol. 22. P. 23–36. URL: https://doi.org/10.1016/j.smrv.2014.10.001 (дата звернення: 28.01.2026).

2. Telegram Bot API. *Telegram*. URL: https://core.telegram.org/bots/api (дата звернення: 12.02.2026).

3. Aiogram Developers. Aiogram Documentation. *Aiogram*. URL: https://docs.aiogram.dev/en/v3.28.2/ (дата звернення: 12.02.2026).

4. FastAPI Documentation. *FastAPI*. URL: https://fastapi.tiangolo.com/ (дата звернення: 13.02.2026).

5. Vertex AI Documentation. *Google Cloud*. URL: https://docs.cloud.google.com/vertex-ai/docs (дата звернення: 20.02.2026).

6. Supabase Docs. *Supabase*. URL: https://supabase.com/docs (дата звернення: 16.02.2026).

7. Docker Compose Documentation. *Docker Docs*. URL: https://docs.docker.com/compose/ (дата звернення: 11.04.2026).

8. Sleep Cycle – Sleep Tracker App. *Sleep Cycle*. URL: https://sleepcycle.com/ (дата звернення: 19.01.2026).

9. Pillow: Sleep Tracker, Monitor & Alarm Clock. *Pillow*. URL: https://pillow.app/ (дата звернення: 20.01.2026).

10. Find your sleep information in Google Fit. *Google Fit Help*. URL: https://support.google.com/fit/answer/9399034 (дата звернення: 21.01.2026).

11. Samsung Health. *Samsung*. URL: https://www.samsung.com/us/apps/samsung-health/ (дата звернення: 22.01.2026).

12. React Documentation. *React*. URL: https://react.dev/ (дата звернення: 11.03.2026).

13. Schürch S. Understanding the Tech Stack Behind Generative AI. *Towards*

Data Science. URL: https://towardsdatascience.com/tech-stack-generative-ai/ (дата звернення: 20.02.2026).