ПРИВАТНЕ АКЦІОНЕРНЕ ТОВАРИСТВО «ВИЩИЙ НАВЧАЛЬНИЙ ЗАКЛАД «МІЖРЕГІОНАЛЬНА АКАДЕМІЯ УПРАВЛІННЯ ПЕРСОНАЛОМ»

Факультет комп'ютерно-інформаційних технологій

Кафедра комп'ютерних інформаційних систем і технологій

Мощинський Антон Дмитрович

КВАЛІФІКАЦІЙНА РОБОТА

на тему: «Розробка AI-системи для формування плану прийому медикаментів на основі рекомендацій лікарів»

Група: ІК-9-22-Б1КНТ (4.0д)

Спеціальність: Комп'ютерні науки

Рівень вищої освіти: перший (бакалавр)

Кваліфікаційна робота містить результати власних досліджень. Ввикористання ідей, результатів, текстів інших авторів мають посилання на відповідне джерело.

_____ _____

(підпис) ПІБ студента

Науковий керівник _____ _____

(підпис) ПІБ

Допущено до захисту ЕК

Завідувач кафедри _____ Сергій КАВУН , д.е.н., професор

(підпис)

Київ 2026

РЕФЕРАТ / ABSTRACT

Пояснювальна записка до атестаційної роботи: 65 с., 14 рис., 1 додаток, 42 джерела.

AI-СИСТЕМА ДЛЯ ФОРМУВАННЯ ПЛАНУ ПРИЙОМУ МЕДИКАМЕНТІВ, ОБРОБКА ПРИРОДНОЇ МОВИ (NLP), OPENAI API, NODE.JS, EXPRESS, REACT, MONGODB.

Об’єктом дослідження є процес автоматизованого формування плану прийому медикаментів на основі текстових рекомендацій лікарів у медичних інформаційних системах.

Метою роботи є розробка AI-системи для аналізу текстових медичних рекомендацій та генерації структурованого розкладу прийому медикаментів. Система повинна забезпечувати інтерпретацію неструктурованого тексту, виділення ключових параметрів призначень (назва препарату, дозування, частота прийому, тривалість курсу) та формування зручного для користувача плану прийому.

Методи розробки базуються на використанні технологій обробки природної мови із застосуванням OpenAI API, а також сучасного веб-стеку: Node.js та Express для серверної частини, React для клієнтського інтерфейсу, MongoDB для зберігання даних.

Результатом роботи є розроблена інформаційна система, що дозволяє автоматизувати процес формування плану прийому медикаментів на основі текстових рекомендацій лікаря. Запропоноване рішення підвищує зручність використання медичних призначень та зменшує ймовірність помилок при їх інтерпретації.

AI-BASED SYSTEM FOR GENERATING MEDICATION INTAKE PLANS, NATURAL LANGUAGE PROCESSING (NLP), OPENAI API, NODE.JS, EXPRESS, REACT, MONGODB.

The object of the research is the process of automated generation of medication intake plans based on textual doctors' recommendations within medical information systems.

The aim of this work is to develop an AI-based system for analyzing textual medical recommendations and generating a structured medication intake schedule. The system is designed to interpret unstructured text, extract key prescription parameters (drug name, dosage, frequency, and duration), and produce a clear and user-friendly intake plan.

The development methods are based on natural language processing techniques using the OpenAI API, as well as a modern web technology stack: Node.js and Express for the backend, React for the frontend, and MongoDB for data storage.

The result of the work is a developed information system that automates the process of generating medication intake plans from textual medical recommendations. The proposed solution improves the usability of medical prescriptions and reduces the risk of misinterpretation, contributing to better treatment adherence.

ЗМІСТ

ВСТУП

За даними Всесвітньої організації охорони здоров'я, недотримання режиму прийому медикаментів (non-adherence) залишається однією з головних причин зниження загальної ефективності терапії та виникнення ускладнень. Сучасні мобільні застосунки та трекери здоров'я пропонують різноманітні інструменти для контролю призначень, проте більшість із них має фундаментальний недолік — вони вимагають від пацієнта виснажливого ручного введення великого обсягу даних (назви препаратів, дозування, частота, тривалість курсу). Це створює високий когнітивний бар'єр входу, який є особливо критичним для людей похилого віку або пацієнтів, що приймають кілька препаратів одночасно. Водночас стрімкий розвиток методів обробки природної мови (NLP) та великих мовних моделей (LLM) відкриває принципово нові можливості для автоматизації екстракції структурованої інформації з вільного тексту.

Проте використання універсальних ШІ-асистентів у медичній сфері пов'язане з високими ризиками через недетермінованість генерації та ймовірність помилок (галюцинацій). Це зумовлює гостру науково-практичну потребу у розробці спеціалізованих систем, які б безпечно інтегрували ШІ для спрощення введення даних, залишаючи за детермінованими алгоритмами формування фінального медичного розкладу.

Мета роботи полягає у проєктуванні та програмній реалізації пацієнт-центричної інтелектуальної системи MedFlow, здатної автоматизовано аналізувати неструктуровані медичні призначення за допомогою методів штучного інтелекту та формувати точний, персоналізований і безпечний план прийому медикаментів.

Об’єкт дослідження — процес управління та планування медикаментозного лікування пацієнта на основі обробки медичних призначень.

Предмет дослідження — моделі, алгоритми та програмні засоби інтеграції методів обробки природної мови у клієнт-серверну архітектуру для автоматизації формування медичних розкладів.

Практичне значення отриманих результатів. Створено повнофункціональний програмний комплекс, який усуває бар'єр між неструктурованим призначенням лікаря (у вигляді тексту, переказу чи фото) та зручним щоденним календарем пацієнта. Запропонована архітектура з двоступеневою валідацією AI-генерацій та ізольованою бізнес-логікою може бути використана як самостійний продукт або інтегрована в існуючі системи електронної охорони здоров'я для підвищення рівня дотримання терапевтичних режимів.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА МЕТОДІВ ФОРМУВАННЯ ПЛАНУ ПРИЙОМУ МЕДИКАМЕНТІВ

1.1 Проблема дотримання режиму прийому медикаментів та її вплив на ефективність лікування

Дотримання режиму прийому медикаментів (medication adherence) є однією з ключових умов ефективного лікування, особливо при хронічних захворюваннях. У науковій літературі цей термін визначається як ступінь відповідності поведінки пацієнта рекомендаціям лікаря щодо часу, дозування та частоти прийому лікарських засобів. Поряд із цим використовується поняття «персистентність» (persistence), яке характеризує тривалість безперервного прийому терапії від моменту її початку до припинення [2].

Незважаючи на значний прогрес у фармакотерапії, проблема недотримання режиму лікування залишається глобальною. За результатами досліджень, середній рівень дотримання медикаментозної терапії становить близько 50%, що свідчить про системний характер цієї проблеми [3]. Неповне або нерегулярне виконання призначень включає пропуск доз, передчасне припинення лікування або неправильне використання лікарських засобів, що суттєво знижує ефективність терапії.

Недотримання режиму прийому медикаментів безпосередньо впливає на клінічні результати. Зокрема, воно асоціюється зі зростанням захворюваності, смертності та частоти ускладнень, особливо у пацієнтів із серцево-судинними захворюваннями. Дослідження показують, що навіть при наявності ефективних препаратів їх користь значною мірою нівелюється через низький рівень adherence, що призводить до погіршення результатів лікування та збільшення витрат системи охорони здоров’я [2].

Крім того, недотримання лікування є однією з основних причин недостатньої ефективності терапії в реальній клінічній практиці. Воно обмежує

досягнення запланованих терапевтичних ефектів, знижує якість життя пацієнтів та ускладнює контроль захворювання. Водночас результати мета-аналізів свідчать, що підвищення рівня adherence навіть помірною мірою призводить до статистично значущого покращення показників якості життя, фізичного стану та симптомів [3].

Проблема має комплексний характер і зумовлена різноманітними факторами, включаючи індивідуальні особливості пацієнтів, складність терапевтичних схем, побічні ефекти препаратів та організаційні аспекти системи охорони здоров'я. У зв'язку з цим ефективне вирішення проблеми потребує мультидисциплінарного підходу та використання комплексних інтервенцій, спрямованих на зміну поведінки пацієнтів і оптимізацію процесу лікування [1, 2].

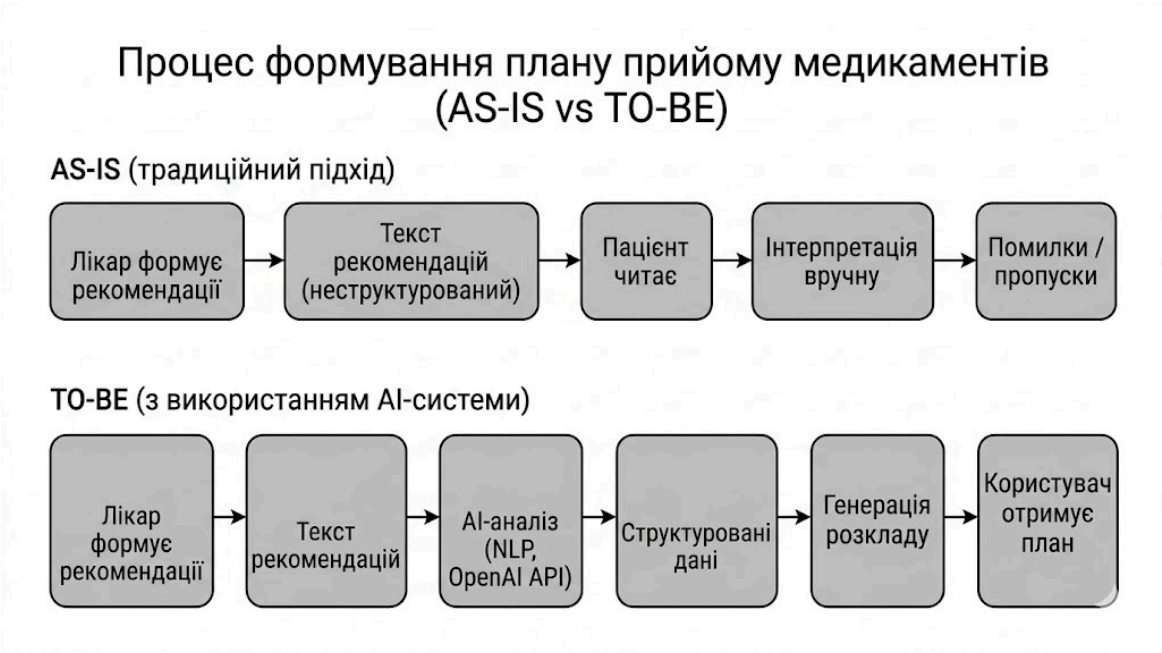

Рисунок 1.1 – Процес формування прийому медикаментів

На рисунку 1.1 представлено порівняння традиційного підходу до формування плану прийому медикаментів та підходу із застосуванням AI-системи. У традиційному підході основна проблема полягає у необхідності ручної інтерпретації текстових рекомендацій, що призводить до можливих помилок.

Запропонований підхід дозволяє автоматизувати цей процес за рахунок використання методів обробки природної мови.

Проблема полягає у відсутності інструменту, який дозволяє автоматично агрегувати неструктуровані лікарські рекомендації з різних джерел у єдиний персоналізований графік прийому ліків. Існуючі рішення або потребують ручного введення всіх даних, або не підтримують паралельні курси лікування, що підвищує ризик пропуску доз, порушення режиму та, як наслідок, зниження ефективності терапії.

Отже, дотримання режиму прийому медикаментів є критично важливим фактором успішності лікування. Недостатній рівень adherence значно знижує ефективність навіть найсучасніших терапевтичних підходів, що обумовлює необхідність розробки нових методів контролю та підтримки пацієнтів у процесі лікування.

1.2 Аналіз існуючих систем управління прийомом медикаментів

Сучасні системи управління прийомом медикаментів спрямовані на підвищення рівня дотримання терапії та зниження кількості помилок, пов’язаних із використанням лікарських засобів. Вони охоплюють широкий спектр рішень — від простих нагадувань до комплексних інформаційних систем, інтегрованих у клінічні процеси [4].

Одним із базових підходів є використання електронних систем нагадування, зокрема мобільних застосунків та автоматизованих повідомлень. Такі рішення дозволяють своєчасно інформувати пацієнтів про необхідність прийому препаратів, що частково знижує ризик пропуску доз. Однак їх ефективність значною мірою залежить від активності користувача та не враховує індивідуальні особливості лікування [5].

Більш складні системи передбачають використання інформаційних технологій для підтримки клінічних рішень. Зокрема, системи класу clinical

decision support забезпечують аналіз призначень, контроль взаємодій між препаратами та формування рекомендацій щодо оптимізації терапії [4]. Такі рішення застосовуються переважно на рівні медичних установ і дозволяють підвищити безпеку лікування, проте їх вплив на поведінку пацієнтів залишається обмеженим.

Окремий напрямок становлять системи дистанційного моніторингу, які використовують сенсори, смарт-пристрої або електронні контейнери для фіксації факту прийому медикаментів. Вони забезпечують більш точний контроль adherence та дозволяють отримувати об’єктивні дані про поведінку пацієнтів [6]. Водночас такі системи потребують додаткових ресурсів і не завжди є доступними у широкій практиці.

Також у літературі розглядаються інтегровані підходи, які поєднують технологічні рішення з поведінковими інтервенціями. Йдеться про системи, що враховують індивідуальні характеристики пацієнтів, складність терапевтичних схем та контекст їх використання [5]. Подібні рішення демонструють вищу ефективність, однак їх реалізація є складнішою з точки зору розробки та впровадження.

Таблиця 1.1 – Системи управління медикаментами

Тип системи	Опис	Переваги	Недоліки
Нагадувальні (reminder apps)	прості мобільні застосунки	простота, доступність	немає персоналізації
CDS системи	клінічні системи підтримки рішень	безпека, інтеграція з лікарем	слабкий вплив на пацієнта
IoT / smart devices	смарт-контейнери, сенсори	точний контроль прийому	висока вартість
AI-based systems	інтелектуальні системи	персоналізація, адаптивність	складність реалізації

Незважаючи на різноманітність існуючих систем, більшість із них мають обмеження. Зокрема, багато рішень або не враховують неструктурований характер медичних призначень, або не забезпечують достатнього рівня персоналізації. Крім того, відсутність інтеграції між різними компонентами систем охорони здоров’я ускладнює створення єдиного підходу до управління медикаментозною терапією [4].

Таким чином, існуючі системи управління прийомом медикаментів можна класифікувати за рівнем складності та функціональності — від простих нагадувань до інтелектуальних систем підтримки прийняття рішень. Водночас жоден із підходів не забезпечує повного вирішення проблеми дотримання лікування, що обумовлює необхідність подальших досліджень у напрямку інтеграції технологій та підвищення рівня персоналізації.

1.3 Методи обробки природної мови для аналізу медичних призначень

Обробка природної мови (Natural Language Processing, NLP) є одним із ключових підходів до аналізу медичних призначень, які здебільшого представлені у вигляді неструктурованого тексту. У клінічній практиці значна частина інформації зберігається у формі записів лікарів, електронних медичних карток та текстових призначень, що ускладнює її автоматизовану обробку. Застосування NLP дозволяє перетворювати такі дані у структурований формат, придатний для подальшого аналізу та використання в інформаційних системах [7, 9].

Одним із базових напрямків використання NLP у медицині є витяг інформації (information extraction), який передбачає автоматичне визначення ключових елементів у тексті, таких як назви препаратів, дозування, частота прийому та тривалість лікування. Для цього застосовуються методи розпізнавання іменованих сутностей (Named Entity Recognition, NER), які дозволяють ідентифікувати медичні терміни та класифікувати їх відповідно до заданих категорій. Додатково використовуються підходи для встановлення зв’язків між

сутностями, що дає змогу відновити повну структуру медичного призначення [7, 8].

Ранні підходи до аналізу медичних текстів базувалися на правил-орієнтованих методах (rule-based systems), які використовують заздалегідь визначені шаблони та словники. Такі методи забезпечують високу точність у вузьких задачах, однак їх гнучкість є обмеженою, особливо у випадках варіативності медичної мови та використання неформалізованих скорочень [8]. Крім того, розробка та підтримка правил потребують значних експертних зусиль.

Подальший розвиток отримали методи машинного навчання, які дозволяють автоматично виявляти закономірності у текстових даних. Зокрема, алгоритми класифікації та послідовні моделі застосовуються для аналізу медичних записів і визначення їх семантичної структури [9]. Такі підходи демонструють кращу здатність до узагальнення порівняно з rule-based системами, однак їх ефективність значною мірою залежить від якості навчальних даних.

Сучасний етап розвитку характеризується використанням глибинного навчання та трансформерних моделей. Вони здатні враховувати контекст і обробляти складні мовні конструкції, що є критично важливим для медичних текстів із високою варіативністю [8]. Зокрема, великі мовні моделі демонструють високу ефективність у задачах витягу інформації та розпізнавання назв лікарських засобів, хоча можуть вимагати значних обчислювальних ресурсів і додаткової перевірки результатів.

Важливою особливістю застосування NLP у медицині є наявність специфічних викликів. До них належать неоднорідність форматів даних, використання спеціалізованої термінології, скорочень та можливі помилки у текстах. Це ускладнює побудову універсальних моделей і знижує їх переносимість між різними медичними системами. Крім того, обмежений доступ до якісних розмічених даних стримує розвиток більш точних моделей [7, 8].

На рисунку 1.2 представлено концептуальну схему, запропонованої AI-системи – воркфлоу. Система забезпечує перетворення текстових рекомендацій лікаря у структурований план прийому медикаментів шляхом застосування методів обробки природної мови та подальшої генерації розкладу.

Рисунок 1.2 – Схема AI-системи формування плану прийому медикаментів

Таким чином, методи обробки природної мови відіграють ключову роль у задачах аналізу медичних призначень, забезпечуючи перехід від неструктурованих текстів до формалізованих даних. Еволюція підходів — від rule-based систем до сучасних моделей глибинного навчання — дозволяє підвищувати точність та гнучкість аналізу, однак питання якості даних, інтерпретованості та інтеграції залишаються відкритими і потребують подальших досліджень.

1.4 Постановка задачі та обґрунтування вибору підходу

Аналіз існуючих рішень у сфері управління прийомом медикаментів та обробки медичних призначень показує, що основною проблемою залишається відсутність єдиного підходу до автоматизованого аналізу неструктурованих

медичних даних. Медичні призначення у більшості випадків подаються у вигляді тексту, що ускладнює їх стандартизовану обробку та інтеграцію в інформаційні системи.

Згідно з дослідженнями у сфері медичного NLP, значна частина клінічної інформації залишається “прихованою” у неструктурованих текстах електронних медичних записів, що потребує застосування методів автоматичного витягу сутностей та їх семантичної інтерпретації [7]. Це підтверджує необхідність використання методів обробки природної мови для трансформації текстових призначень у структуровані дані, придатні для подальшого аналізу та використання в системах підтримки прийняття рішень.

Додатково, сучасні підходи до аналізу медичних текстів демонструють, що класичні rule-based системи є недостатньо гнучкими для роботи з реальними клінічними даними через високу варіативність формулювань та використання скорочень. У зв’язку з цим дедалі більшої актуальності набувають методи машинного навчання та глибинного навчання, які здатні враховувати контекст і краще адаптуватися до різних форматів медичних записів [8].

Таким чином, можна сформулювати основну задачу дослідження як розробку підходу до автоматизованого аналізу медичних призначень на основі методів обробки природної мови, що дозволяє:

- витягувати ключові сутності (препарати, дозування, частота прийому);
- структурувати текстові призначення;
- забезпечувати основу для подальшого формування плану прийому медикаментів.

Обґрунтування вибору NLP-підходу базується на його здатності працювати з неструктурованими текстами та забезпечувати перехід до формалізованого представлення даних. На відміну від класичних підходів, які вимагають жорстко заданих правил, сучасні NLP-моделі дозволяють автоматично узагальнювати

патерни у медичних текстах та підвищувати точність обробки за рахунок навчання на великих масивах даних [7, 8].

Отже, запропонований підхід ґрунтується на використанні методів обробки природної мови як базового механізму для аналізу медичних призначень, що дозволяє подолати обмеження існуючих систем та забезпечити більш гнучке й масштабоване рішення.

1.5 Висновки до розділу

У першому розділі було розглянуто предметну область дотримання режиму прийому медикаментів, а також сучасні підходи до аналізу та обробки медичних призначень. Аналіз показав, що проблема низького рівня adherence залишається однією з ключових у сучасній системі охорони здоров’я, оскільки безпосередньо впливає на ефективність лікування, частоту ускладнень та загальні витрати на медичну допомогу.

Дослідження існуючих систем управління прийомом медикаментів продемонструвало наявність широкого спектра рішень — від простих нагадувальних механізмів до складних клінічних систем підтримки прийняття рішень. Водночас більшість із них має суттєві обмеження: недостатній рівень персоналізації, слабку адаптацію до реальних умов використання та обмежену інтеграцію з неструктурованими медичними даними.

Окрему увагу приділено методам обробки природної мови, які є ключовими для аналізу медичних призначень. Показано, що еволюція підходів — від правил-орієнтованих систем до методів машинного та глибинного навчання — дозволила значно підвищити якість витягу інформації з текстових медичних записів. Однак при цьому залишаються відкритими питання точності, інтерпретованості та стійкості моделей у реальних клінічних умовах.

На основі проведеного аналізу сформульовано задачу дослідження, яка полягає у розробці підходу до автоматизованого аналізу медичних призначень із

використанням методів обробки природної мови. Такий підхід дозволяє структурувати неформалізовані текстові дані та створити основу для подальшого формування інтелектуальних систем підтримки прийому медикаментів.

Отже, результати розділу підтверджують актуальність обраного напрямку дослідження та необхідність розробки більш гнучких і точних методів обробки медичних текстів, орієнтованих на реальні потреби клінічної практики.

2 ПРОЄКТУВАННЯ AI-СИСТЕМИ ФОРМУВАННЯ ПЛАНУ ПРИЙОМУ МЕДИКАМЕНТІВ MEDFLOW

2.1 Визначення вимог до системи

Система MedFlow повинна забезпечувати повний цикл управління медикаментозним лікуванням. На рисунку 2.1 представлено діаграму варіантів використання системи MedFlow.

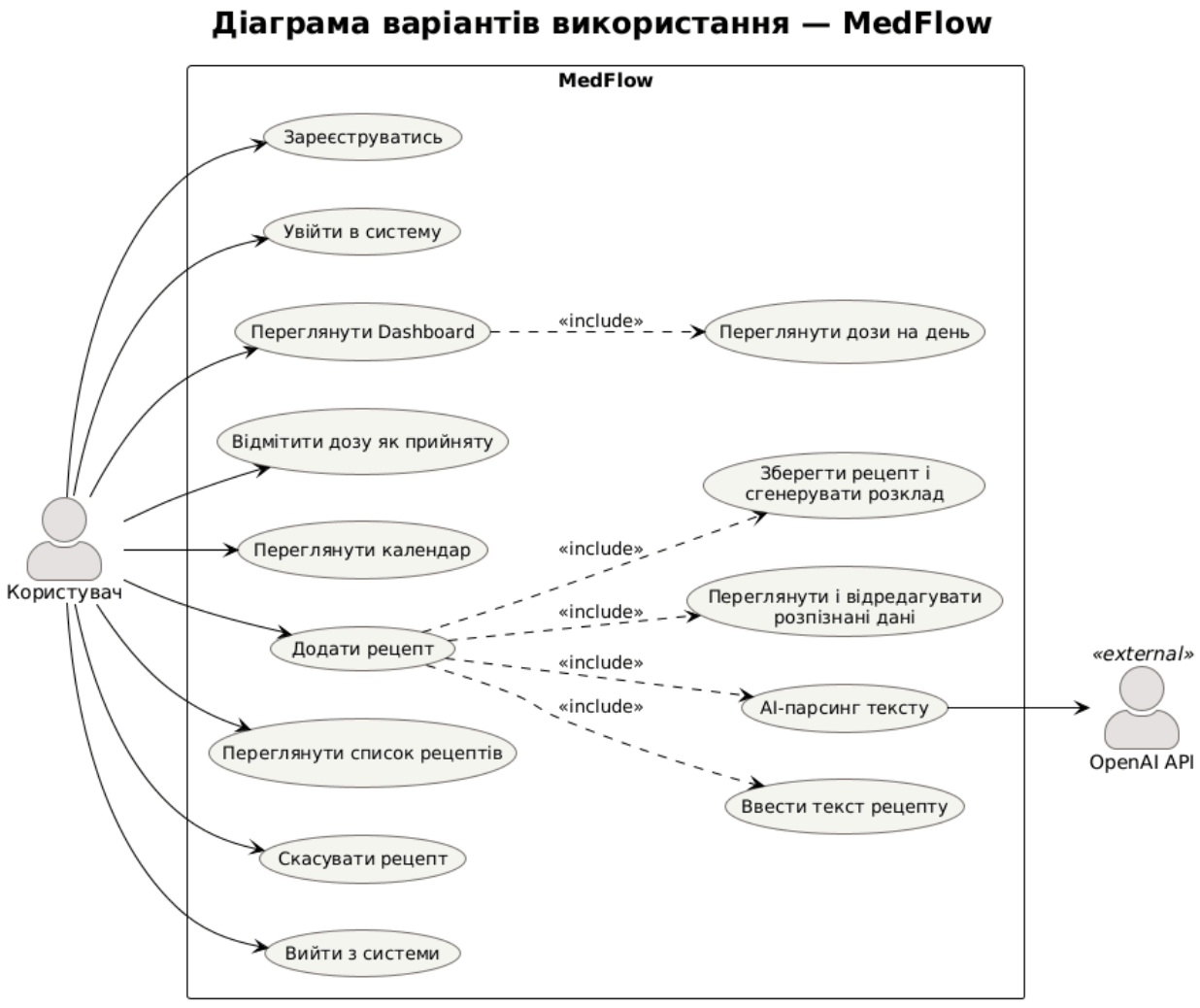

Рисунок 2.1 – Діаграма варіантів використання

У таблиці 2.1 детальніше розписані сценарії використання.

Таблиця 2.1 – Сценарії використання

Код	Назва сценарію	Роль	Дія	Очікуваний результат	Умови успіху
UC-1	Реєстрація користувача	Користувач	Вводить реєстраційні дані	Створення облікового запису	Дані коректні
UC-2	Вхід в систему	Користувач	Вводить логін і пароль	Авторизація в системі	Дані правильні
UC-3	Додавання рецепту	Користувач	Вводить текст рекомендацій	Рецепт передається в AI-модуль	API доступне
UC-4	AI-парсинг тексту	OpenAI API	Аналізує текст	Формуються структуровані дані	Аналіз успішний
UC-5	Генерація розкладу	Система	Формує план прийому	Створений розклад	Дані оброблені
UC-6	Перегляд Dashboard	Користувач	Відкриває Dashboard	Відображення розкладу	Користувач авторизований
UC-7	Відмітити дозу	Користувач	Підтверджує прийом	Статус оновлено	БД доступна

Беручи до уваги визначені класи користувачів, варіанти використання і дослідження можливостей подібних систем визначено функціональні та нефункціональні вимоги до системи. Функціональні вимоги до системи наведено в таблиці 2.2.

Таблиця 2.2 – Опис функціональних вимог до системи

№ п/п	Опис вимоги
ФВ-01	Система повинна забезпечувати введення текстових рекомендацій лікаря користувачем.
ФВ-02	Система повинна виконувати аналіз текстових рекомендацій із використанням AI-модуля та методів обробки природної мови.
ФВ-03	Система повинна визначати назви медикаментів із тексту рекомендацій.
ФВ-04	Система повинна визначати дозування препаратів із тексту рекомендацій.
ФВ-05	Система повинна визначати частоту та тривалість прийому медикаментів.
ФВ-06	Система повинна формувати структурований план прийому медикаментів на основі отриманих даних.
ФВ-07	Система повинна відображати сформований розклад прийому у зручному для користувача вигляді.
ФВ-08	Система повинна забезпечувати збереження інформації про сформовані плани прийому медикаментів у базі даних.
ФВ-09	Система повинна забезпечувати можливість редагування плану прийому медикаментів користувачем.
ФВ-10	Система повинна забезпечувати взаємодію між клієнтською та серверною частинами через REST API.
ФВ-11	Система повинна забезпечувати обробку помилок під час аналізу тексту або генерації розкладу.
ФВ-12	Система повинна забезпечувати авторизацію та автентифікацію користувачів.

Нефункціональні вимоги визначають якісні характеристики програмної системи та описують обмеження й умови її функціонування. До таких вимог належать характеристики продуктивності, надійності, безпеки, масштабованості та зручності використання системи MedFlow. Дотримання нефункціональних вимог є важливим для забезпечення стабільної та ефективної роботи програмного продукту. Нефункціональні вимоги до системи подано в таблиці 2.3.

Таблиця 2.3 – Опис нефункціональних вимог до системи

№ п/п	Опис вимоги
НФВ-01	Система повинна мати зрозумілий та інтуїтивний користувацький інтерфейс.
НФВ-02	Система повинна забезпечувати швидку обробку текстових рекомендацій користувача.
НФВ-03	Система повинна забезпечувати захист персональних та медичних даних користувачів.
НФВ-04	Система повинна підтримувати масштабованість серверної архітектури.
НФВ-05	Система повинна забезпечувати стабільну роботу при одночасній роботі кількох користувачів.
НФВ-06	Система повинна забезпечувати коректну роботу у сучасних веб-браузерах.
НФВ-07	Система повинна підтримувати модульну структуру програмного коду для спрощення подальшого розширення функціональності.
НФВ-08	Система повинна забезпечувати надійне зберігання даних у базі даних MongoDB.
НФВ-09	Система повинна забезпечувати можливість інтеграції із зовнішніми AI-сервісами через API.
НФВ-10	Система повинна забезпечувати адаптивність інтерфейсу для різних типів пристроїв.

2.2 Загальна архітектура системи MedFlow

Проєктування сучасних інформаційних систем у сфері охорони здоров’я передбачає використання архітектурних підходів, які забезпечують масштабованість, модульність та надійність. Одним із найбільш поширених рішень є багаторівнева (зокрема трирівнева) архітектура, яка дозволяє розділити систему на незалежні компоненти: рівень представлення, рівень бізнес-логіки та рівень зберігання даних . Такий підхід спрощує супровід системи, підвищує її гнучкість і дозволяє масштабувати окремі частини без впливу на інші [10].

У сучасних веборієнтованих системах важливу роль відіграє також використання сервісно-орієнтованих підходів і API-взаємодії. Зокрема, застосування REST-архітектури забезпечує стандартизований обмін даними між клієнтом і сервером, а використання stateless-підходу дозволяє досягти кращої масштабованості та відмовостійкості системи. У контексті медичних інформаційних систем це є критично важливим через необхідність обробки великої кількості запитів і забезпечення стабільного доступу до даних [11].

У ході виконання дипломної роботи було розроблено вебзастосунок MedFlow, призначений для автоматизованого формування плану прийому медикаментів на основі текстових медичних призначень. Систему спроєктовано як трирівневу архітектуру, що включає клієнтський, серверний рівні та рівень зберігання даних. Клієнтський рівень реалізовано як односторінковий вебзастосунок, який забезпечує взаємодію з користувачем та відображення результатів обробки. Серверний рівень відповідає за реалізацію бізнес-логіки, обробку запитів і взаємодію з базою даних, тоді як рівень зберігання даних представлений документоорієнтованою базою, що дозволяє ефективно працювати з напівструктурованими даними медичних призначень. Особливістю розробленого застосунку є інтеграція методів обробки природної мови для автоматичного перетворення неструктурованих текстових призначень у структурований формат.

З урахуванням зазначених підходів система MedFlow спроєктована як трирівнева веб-архітектура, що включає клієнтський, серверний рівні та рівень зберігання даних. Клієнтський рівень реалізовано як односторінковий вебзастосунок, який відповідає за взаємодію з користувачем та відображення даних. Серверний рівень виконує обробку запитів, реалізує бізнес-логіку системи та забезпечує взаємодію з базою даних. Рівень зберігання даних представлений документоорієнтованою базою даних, що дозволяє ефективно працювати з напівструктурованими медичними даними.

Комунікація між клієнтом і сервером реалізована через REST API з використанням JSON-формату обміну даними та механізмів автентифікації на основі токенів. Такий підхід забезпечує слабку зв’язаність компонентів системи та дозволяє незалежно розвивати клієнтську і серверну частини. Крім того, використання stateless-взаємодії створює передумови для горизонтального масштабування системи.

Ключовою особливістю архітектури MedFlow є винесення логіки обробки природної мови в окремий сервіс. Відповідно до сучасних підходів до побудови інтелектуальних систем, інтеграція моделей штучного інтелекту має бути ізольована від основної бізнес-логіки для забезпечення гнучкості та можливості подальшої заміни або оновлення моделей без зміни інших компонентів системи [11]. У системі MedFlow це реалізовано через окремий модуль, який відповідає за взаємодію з зовнішнім AI-сервісом і виконує обробку текстових медичних призначень.

Серверна частина системи побудована за модульним принципом із розділенням на окремі компоненти, що відповідають за маршрутизацію, бізнес-логіку, роботу з даними та обробку запитів. Такий підхід відповідає принципу розділення відповідальностей (separation of concerns) і дозволяє підвищити тестованість та розширюваність системи [10].

Отже, обрана архітектура системи MedFlow поєднує класичні підходи до побудови багаторівневих систем із сучасними рішеннями у сфері веброзробки та інтеграції штучного інтелекту. Це забезпечує баланс між гнучкістю, продуктивністю та можливістю подальшого розвитку системи.

2.3 Обґрунтування вибору технологічних компонентів системи

Для реалізації клієнтського SPA обрано React 18 — найпоширенішу бібліотеку для побудови інтерфейсів за результатами Stack Overflow Developer Survey 2024, де її використовують 39.5% розробників проти 17.1% Angular та

15.4% Vue. Компонентна архітектура React природно лягає на структуру MedFlow з повторюваними UI-блоками (InlineField, Sidebar, картки прийому), а механізми Concurrent Rendering та автоматичний batching стейт-апдейтів у React 18 забезпечують плавність UI під час асинхронних викликів до AI-сервісу, що особливо важливо для двоетапного флоу парсингу припису. Найбільша серед фронтенд-бібліотек екосистема дозволяє інтегрувати спеціалізовані готові рішення для роутингу, кешування серверного стану та клієнтського стану без потреби в самописних абстракціях [43-44].

Таблиця 2.4 – Популярність фронтенд-фреймворків

Фреймворк	**Частка, %**
React	39.5
Angular	17.1
Vue.js	15.4
Svelte	6.5

Для маршрутизації використано React Router v6 як стандарт де-факто для декларативної маршрутизації в React-додатках з підтримкою nested routes, завантажувачів та захищених маршрутів, що дозволило реалізувати редирект неавтентифікованих користувачів мінімальним кодом. Управління станом свідомо розділене за природою даних: для клієнтського стану автентифікації обрано Zustand — мінімалістичний стор без boilerplate-коду, що за результатами незалежних бенчмарків демонструє кращу продуктивність та меншу когнітивну складність порівняно з Redux Toolkit, а вбудована middleware persist забезпечує збереження JWT-токена в localStorage без власної реалізації; натомість для серверного стану використано TanStack React Query, що, на відміну від Redux чи MobX, спеціалізується саме на кешуванні, інвалідації та фоновій ресинхронізації даних з REST API — це усуває цілий клас типових багів (race conditions, stale data,

ручне управління loading/error станами) та автоматично оновлює UI після створення припису через інвалідацію ключів. HTTP-комунікацію реалізовано через Axios замість нативного fetch через підтримку інтерсепторів запитів — критичну функцію для централізованого додавання Authorization header із токеном, що відсутня у fetch без обгорток [45-47].

Стилізація виконана через Tailwind CSS v4 з utility-first підходом, який, за дослідженнями State of CSS 2023, демонструє найвищий рівень задоволеності серед CSS-фреймворків (78%) і скорочує час на розробку UI завдяки відсутності контекст-свічингу між JS та CSS. Складальник Vite обрано замість Webpack завдяки нативній підтримці ESM та використанню esbuild для холодного старту [8], що скорочує час білда dev-сервера з десятків секунд до <1 с і суттєво пришвидшує цикл розробки [48].

Як бекенд обрано Node.js + Express.js — Node.js залишається найбільш популярною серверною технологією за Stack Overflow Survey 2024 (40.8% усіх розробників), а Express є найбільш зрілим мінімалістичним HTTP-фреймворком з понад 33 млн завантажень на тиждень за NPM Trends [9]. Ключова перевага Node.js саме для MedFlow — асинхронна неблокуюча модель I/O на основі event loop, що ідеально підходить для системи, де основне навантаження становлять зовнішні мережеві виклики (OpenAI API, MongoDB), а не CPU-bound обчислення; використання JavaScript/TypeScript на обох сторонах також усуває контекст-свічінг для розробника та дозволяє перевикористовувати типи між клієнтом та сервером. TypeScript застосовано на всіх шарах системи замість чистого JavaScript через статичну типобезпеку, що, за дослідженням Microsoft Research, попереджає до 15% багів на етапі компіляції, та підтримку IDE-навігації, яка є критичною в системах з кількома взаємопов'язаними доменними моделями [43, 49-50].

Для зберігання даних обрано MongoDB через Mongoose ODM замість реляційної PostgreSQL виходячи з природи даних: припис має напівструктурований характер з частково опційними полями, а денормалізована

схема ScheduleEntry оптимізована під документоорієнтовану модель — це забезпечує читання повного дня розкладу за один індексний запит без JOIN-операцій; Mongoose додає рівень валідації схеми та типобезпеку поверх гнучкості MongoDB [51-52].

Для реалізації AI-парсингу обрано OpenAI gpt-4o-mini як оптимальний компроміс між якістю та вартістю в категорії невеликих instruction-tuned моделей. Ключовою для MedFlow перевагою gpt-4o-mini над відкритими моделями (Llama 3, Mistral) є нативна підтримка JSON-mode, що на рівні моделі гарантує синтаксично валідний JSON на виході та усуває потребу в самописних retry-механізмах при некоректному форматуванні. Параметр temperature: 0 забезпечує близьку до детермінованої поведінку — критичну вимогу для медичного домену, де варіабельність інтерпретації одного й того ж припису неприпустима. Архітектурне рішення про використання керованого API замість self-hosted-моделі обґрунтоване економічно: при середній вартості одного парсингу ≈$0.0002 і відсутності інфраструктурних витрат на GPU-сервери підхід є оптимальним для MVP-стадії та раннього масштабування [53-54].

2.4 Проєктування моделі даних та бази даних

Проєктування моделі даних є одним із ключових етапів розробки інформаційних систем, оскільки саме структура даних визначає ефективність зберігання, доступу та обробки інформації. У сучасних системах, що працюють із великими обсягами напівструктурованих або неструктурованих даних, традиційні реляційні підходи часто виявляються недостатньо гнучкими. У зв’язку з цим дедалі ширше застосовуються NoSQL-бази даних, які дозволяють працювати з динамічними схемами та адаптувати структуру даних під конкретні задачі [12].

Згідно з дослідженнями у сфері управління даними, документоорієнтовані бази даних є особливо ефективними у випадках, коли дані мають неоднорідну структуру або можуть змінюватися в процесі роботи системи. Вони дозволяють

зберігати об’єкти у вигляді документів без жорстко визначеної схеми, що значно спрощує обробку складних або частково заповнених даних [13]. Такий підхід також сприяє підвищенню продуктивності системи за рахунок зменшення кількості складних об’єднань (join-операцій), характерних для реляційних баз даних.

У ході виконання дипломної роботи було спроєктовано модель даних для системи MedFlow, яка враховує специфіку медичних призначень як напівструктурованих текстових даних. Основу моделі становлять три ключові сутності: користувач, медичне призначення та елемент розкладу прийому медикаментів. Такий поділ дозволяє логічно відокремити дані автентифікації, вихідний текст припису та результати його обробки.

Особливістю реалізованої моделі є використання вкладених структур для збереження результатів обробки медичних призначень. Зокрема, після виконання NLP-аналізу тексту припису структуровані дані зберігаються разом із вихідним текстом у вигляді вкладеного документа. Це дозволяє зберегти зв’язок між сирими та обробленими даними без необхідності створення додаткових таблиць або складних зв’язків.

Важливим проєктним рішенням є застосування денормалізації даних у колекції, що відповідає за розклад прийому медикаментів. На відміну від класичних реляційних моделей, де дані максимально нормалізуються, у даному випадку частина інформації (зокрема назва препарату та дозування) дублюється в кожному записі розкладу. Такий підхід відповідає рекомендаціям щодо оптимізації NoSQL-систем, де перевага надається швидкості читання та простоті запитів [12]. Це дозволяє виконувати типові операції, такі як отримання списку прийомів на конкретну дату, без додаткових обчислювальних витрат.

Використання документоорієнтованої бази даних також обґрунтоване тим, що окремі параметри медичних призначень можуть бути відсутні або мати різну структуру. Наприклад, поля, що описують частоту чи тривалість прийому, не

завжди присутні у вихідному тексті або можуть бути інтерпретовані неоднозначно. Гнучка схема дозволяє коректно обробляти такі випадки без необхідності зміни структури бази даних.

Отже, обрана модель даних поєднує принципи гнучкості, продуктивності та адаптивності до специфіки предметної області. Використання документоорієнтованого підходу разом із денормалізацією забезпечує ефективну роботу системи з напівструктурованими медичними даними та створює основу для подальшого масштабування й розвитку системи.

Додатково при проєктуванні моделі даних було враховано характер навантаження на систему, зокрема переважання операцій читання над записом. Основний сценарій використання передбачає частий доступ до розкладу прийому медикаментів (наприклад, перегляд доз на поточний день), тоді як операції створення або редагування призначень виконуються значно рідше. У зв’язку з цим структура даних була оптимізована саме під швидке читання, що проявляється у спрощенні запитів та мінімізації необхідності обробки зв’язків між колекціями. Такий підхід дозволяє зменшити затримки відповіді системи та забезпечити більш стабільну роботу навіть при зростанні кількості користувачів.

2.5 Проєктування серверної архітектури та REST API

Проєктування серверної частини сучасних вебзастосунків базується на принципах модульності, масштабованості та чіткого розділення відповідальностей. Одним із ключових підходів є багатошарова архітектура, у межах якої виділяються окремі рівні обробки запитів, бізнес-логіки та доступу до даних. Така організація системи дозволяє локалізувати зміни, підвищує підтримуваність коду та спрощує тестування [14].

У контексті клієнт-серверної взаємодії широко використовується архітектурний стиль REST, який передбачає роботу з ресурсами через стандартні HTTP-методи. Основною особливістю REST є stateless-природа взаємодії, коли

кожен запит містить усю необхідну інформацію для його обробки. Це забезпечує незалежність компонентів і сприяє горизонтальному масштабуванню системи [15]. Додатково REST-підхід дозволяє будувати інтерфейси з чіткою семантикою, що спрощує інтеграцію між різними компонентами системи.

Сучасні дослідження також підкреслюють важливість сервісно-орієнтованого підходу, за якого складні системи розбиваються на незалежні модулі або сервіси, кожен із яких виконує окрему функцію [16]. Такий підхід є особливо актуальним для систем, що інтегрують алгоритми штучного інтелекту, оскільки дозволяє ізолювати обчислювально складні або зовнішні залежності від основної бізнес-логіки.

У межах дипломної роботи серверну частину системи MedFlow реалізовано з урахуванням зазначених підходів. Архітектура побудована на розділенні на логічні шари: маршрути (routes), сервіси (services) та моделі даних (models). Шар маршрутів відповідає за прийом і маршрутизацію HTTP-запитів, сервісний шар інкапсулює бізнес-логіку, включаючи обробку медичних призначень і генерацію розкладу, а шар моделей забезпечує взаємодію з базою даних.

REST API системи спроєктовано навколо ресурсного підходу, де кожна сутність має власний набір endpoint-ів. При цьому використано семантично зрозумілі маршрути та стандартні HTTP-методи, що спрощує як розробку, так і подальше використання API. Особливістю є розділення процесу обробки медичного призначення на два етапи: попередній аналіз без збереження та фінальне підтвердження з генерацією розкладу.

Окрему увагу приділено безпеці серверної частини. Для автентифікації використовується токен-орієнтований підхід, що дозволяє реалізувати stateless-доступ до ресурсів. Додатково застосовуються механізми обмеження частоти запитів, валідації вхідних даних і встановлення захисних HTTP-заголовків, що відповідає сучасним вимогам до безпеки веб-застосунків [15].

Важливим елементом архітектури є також централізована обробка помилок, що забезпечує єдиний формат відповідей сервера незалежно від помилки. Це підвищує передбачуваність поведінки системи, спрощує інтеграцію з клієнтською частиною.

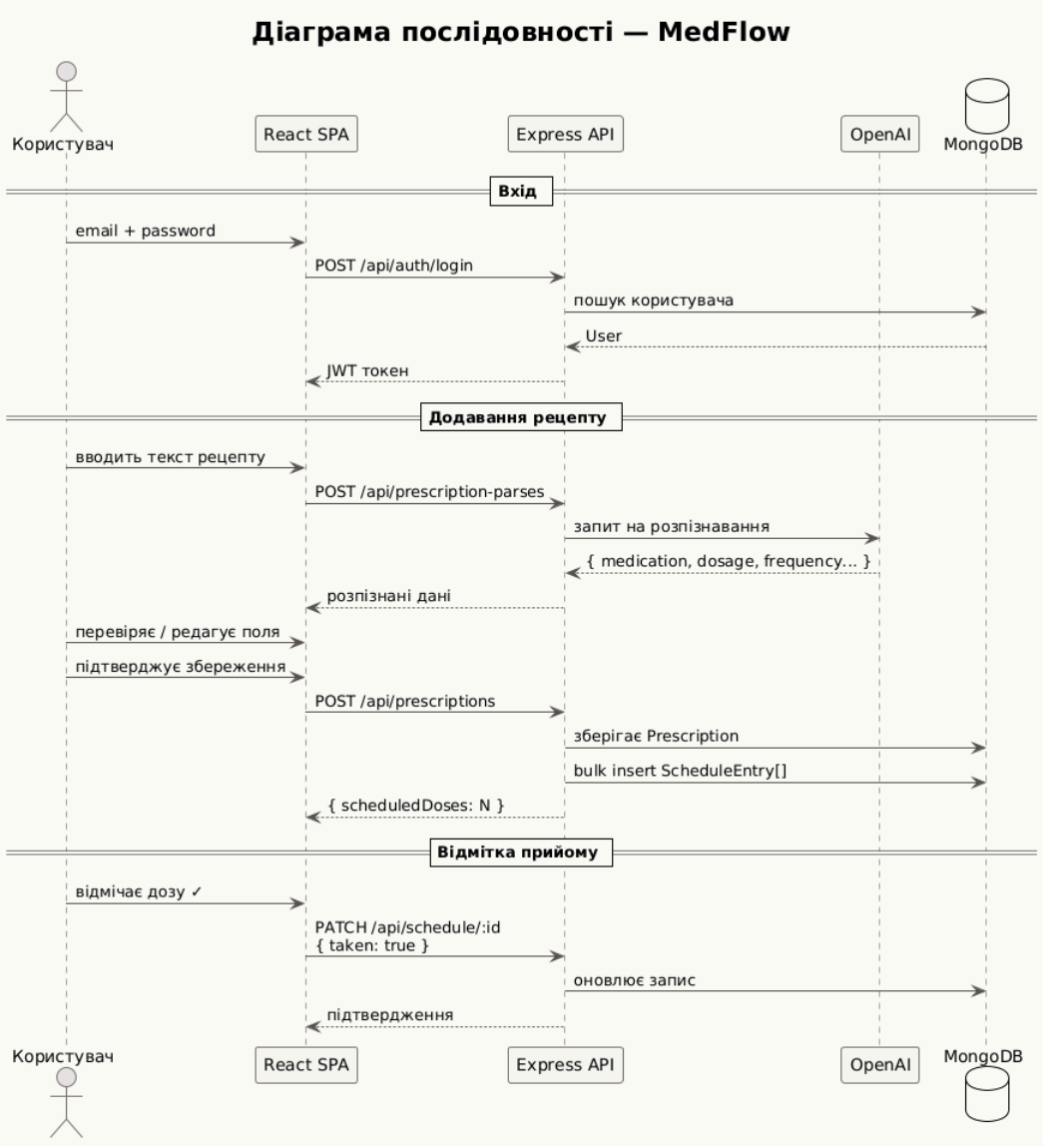

Рисунок 2.2 – Діаграма послідовності

Крім того, у системі MedFlow реалізовано окремі сервіси для обробки специфічної логіки, зокрема модуль для аналізу текстових медичних призначень та модуль генерації розкладу. Така декомпозиція відповідає сервісно-орієнтованому підходу та дозволяє ізолювати складні обчислення, пов’язані з використанням зовнішніх AI-сервісів, від основної логіки системи [16]. Діаграму послідовності подано на рисунку 2.2.

Отже, серверна архітектура системи MedFlow поєднує принципи багатошаровості, REST-взаємодії та сервісної декомпозиції. Це забезпечує гнучкість, масштабованість і надійність системи, а також створює основу для її подальшого розвитку та інтеграції з іншими компонентами.

2.6 Проєктування клієнтської частини системи

Сучасні вебзастосунки дедалі частіше реалізуються у форматі односторінкових застосунків (SPA), що дозволяє забезпечити швидку взаємодію з користувачем без повного перезавантаження сторінок. Дослідження в області UX та вебархітектури показують, що SPA-підхід покращує сприйняття швидкодії системи та зменшує затримки при навігації між функціональними модулями, що особливо важливо для інтерактивних застосунків [29].

У медичних інформаційних системах особлива увага приділяється зручності інтерфейсу та зменшенню когнітивного навантаження користувача. Як показують дослідження, ефективність таких систем значною мірою залежить від того, наскільки інтуїтивно зрозуміло організовано представлення даних і наскільки швидко користувач може отримати доступ до критичної інформації [27]. Це зумовлює необхідність ретельного проєктування структури сторінок і сценаріїв взаємодії.

Окремим важливим аспектом є управління станом у складних клієнтських застосунках. Сучасні підходи передбачають розділення локального та серверного стану, що дозволяє уникнути дублювання логіки та спрощує синхронізацію даних

[28]. Використання спеціалізованих інструментів для кешування та роботи із серверними даними дозволяє зменшити кількість запитів до API та підвищити загальну продуктивність застосунку.

У межах дипломної роботи клієнтську частину системи MedFlow реалізовано як односторінковий застосунок, побудований на сучасному JavaScript-фреймворку. Архітектура інтерфейсу організована навколо кількох основних сторінок, кожна з яких відповідає за окремий сценарій використання системи. Такий підхід дозволяє логічно розділити функціональність і спростити навігацію для користувача.

Головна сторінка системи призначена для відображення актуального розкладу прийому медикаментів та забезпечує швидкий доступ до основних дій. Додатково передбачено календарний перегляд, який дозволяє користувачу аналізувати розподіл прийомів у часі та оцінювати загальну структуру лікування. Така організація інтерфейсу відповідає принципам інформаційної ієрархії та покращує сприйняття даних.

Окремий модуль відповідає за введення медичних призначень у вільній формі. Його інтерфейс спроєктовано таким чином, щоб користувач міг не лише вводити текст, але й переглядати результат його автоматичної обробки перед збереженням. Це дозволяє поєднати автоматизацію з можливістю контролю з боку користувача, що є важливим у медичних застосунках.

Управління станом у клієнтській частині реалізовано з розділенням відповідальностей між локальним та серверним станом. Такий підхід дозволяє ефективно організувати кешування даних, зменшити кількість зайвих запитів до сервера та забезпечити узгодженість відображуваної інформації. Зокрема, дані про користувача та автентифікацію зберігаються окремо від даних, що надходять із сервера.

Важливим елементом реалізації є централізована конфігурація HTTP-запитів, що забезпечує єдиний механізм комунікації з серверною частиною

системи. Це дозволяє уніфікувати обробку токенів автентифікації та спростити подальшу підтримку клієнтської логіки.

Дизайн інтерфейсу побудовано з акцентом на мінімалізм і зрозумілість, що відповідає сучасним практикам UX-дизайну для інформаційних систем. Використання єдиної дизайн-системи забезпечує консистентність елементів інтерфейсу та зменшує когнітивне навантаження на користувача.

Отже, клієнтська частина системи MedFlow реалізує сучасний підхід до побудови SPA-застосунків із розділенням станів, продуманою навігацією та орієнтацією на зручність користувача. Це забезпечує ефективну взаємодію з системою та підтримує загальну архітектурну цілісність проєкту.

2.7 Проєктування взаємодії користувача з системою

Проєктування взаємодії користувача з інформаційними системами у сфері охорони здоров’я вимагає особливої уваги до точності, зрозумілості та надійності інтерфейсу. На відміну від загальних вебзастосунків, медичні системи працюють із критично важливими даними, тому навіть незначні помилки в інтерпретації або взаємодії можуть мати серйозні наслідки. У зв’язку з цим значна увага приділяється зручності використання (usability) та мінімізації когнітивного навантаження на користувача [19].

Дослідження показують, що ефективність застосунків для контролю прийому медикаментів значною мірою залежить від якості їхнього інтерфейсу та простоти взаємодії. Інтуїтивно зрозумілі елементи керування, чітке відображення інформації та мінімальна кількість дій для виконання основних сценаріїв безпосередньо впливають на рівень дотримання режиму лікування [19]. Таким чином, UX-дизайн виступає не лише допоміжним, а критично важливим компонентом системи.

Окремим важливим аспектом є інтеграція підходу human-in-the-loop, який передбачає участь користувача у процесі прийняття рішень, сформованих

системою штучного інтелекту. У медичних застосунках це дозволяє уникнути сліпого довіряння автоматизованим результатам та забезпечує додатковий рівень контролю [21]. Зокрема, сучасні системи підтримки прийняття рішень рекомендують надавати користувачу можливість перевіряти та підтверджувати результати, згенеровані алгоритмами.

У рамках дипломної роботи взаємодію користувача з системою MedFlow побудовано з урахуванням зазначених підходів. Ключовим сценарієм є введення тексту медичного призначення та подальше отримання структурованого результату його обробки. При цьому процес організовано як двоетапний: спочатку система виконує попередній аналіз тексту та відображає результат користувачу, після чого користувач має можливість перевірити, відредагувати та підтвердити дані перед їх збереженням.

Такий підхід дозволяє поєднати автоматизацію з контролем з боку людини, що є критично важливим для медичних систем. Відповідно до рекомендацій щодо проєктування інтерфейсів систем підтримки прийняття рішень, користувач повинен залишатися фінальним суб’єктом прийняття рішення, а система має виступати лише як допоміжний інструмент [20]. Реалізація можливості ручного редагування кожного поля перед підтвердженням дозволяє мінімізувати ризик помилок, пов’язаних із некоректною інтерпретацією тексту алгоритмом.

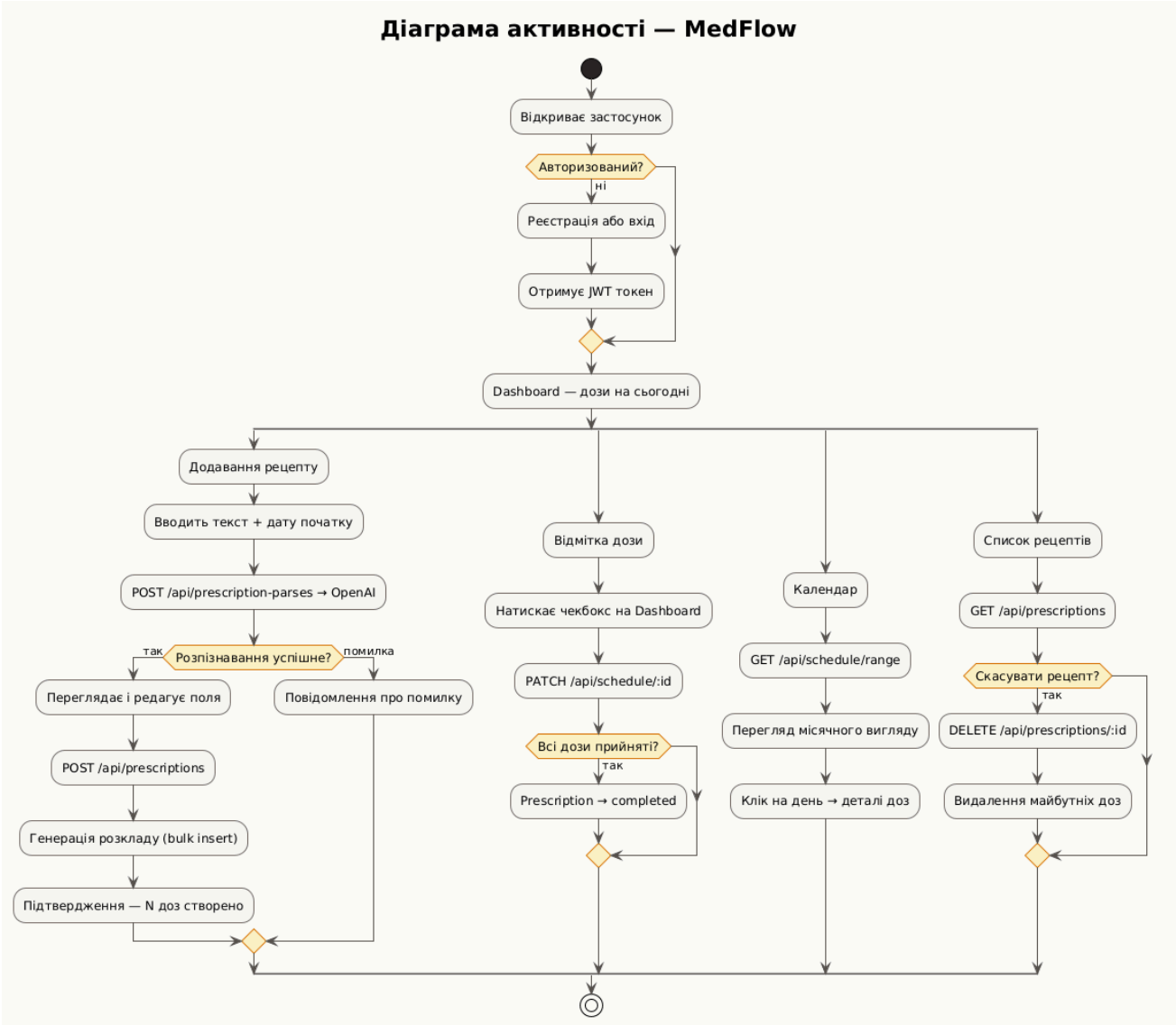

Рисунок 2.3 – Діаграма активності

Основний сценарій щоденної взаємодії користувача з системою зводиться до перегляду сформованого розкладу прийому медикаментів та відмітки виконаних дій. Інтерфейс організовано таким чином, щоб користувач міг швидко отримати інформацію про заплановані прийоми на поточний день і виконати необхідні дії з мінімальною кількістю взаємодій. Це відповідає принципу зменшення когнітивного навантаження та підвищує зручність використання системи [19].

Додатково передбачено механізми валідації дій користувача, зокрема, обмеження можливості відмітити майбутні прийоми як виконані. Такий підхід забезпечує логічну узгодженість даних і запобігає виникненню некоректних станів системи. Водночас користувач має доступ до ширшого часового контексту через перегляд розкладу за обраний період, що дозволяє краще контролювати процес лікування. На рисунку 2.3 зображено діаграму активності застосунку MedFlow.

Отже, взаємодія користувача з системою MedFlow побудована на поєднанні принципів зручності використання, прозорості роботи алгоритмів та контролю з боку людини. Використання підходу human-in-the-loop разом із продуманим UX-дизайном дозволяє підвищити надійність системи та забезпечити більш ефективне дотримання режиму прийому медикаментів.

2.8 Алгоритм формування плану прийому медикаментів

Автоматизація обробки медичних призначень є складною задачею, що поєднує методи обробки природної мови та алгоритмічні підходи до структуризації даних. Сучасні дослідження демонструють, що використання великих мовних моделей (LLM) дозволяє ефективно виконувати задачі витягування інформації з неструктурованих медичних текстів, зокрема визначати ключові параметри призначень, такі як назва препарату, дозування та режим прийому [22]. При цьому особливу увагу приділяють забезпеченню коректності та узгодженості отриманих результатів.

Разом із тим, інтеграція LLM у медичні системи вимагає поєднання ймовірнісних моделей із детермінованими алгоритмами. Це пов’язано з тим, що результати роботи мовних моделей можуть містити варіативність, яка є неприйнятною для задач, де необхідна точність і передбачуваність. У зв’язку з цим рекомендується використовувати гібридні підходи, де LLM застосовується для первинного структурування інформації, а подальша обробка виконується за допомогою чітко визначених алгоритмів [23].

У межах дипломної роботи алгоритм формування плану прийому медикаментів у системі MedFlow побудовано саме за таким гібридним принципом і складається з двох послідовних етапів. На першому етапі виконується обробка тексту медичного призначення за допомогою мовної моделі, яка перетворює неструктурований текст у формалізований набір параметрів. Результатом є структура, що містить ключові характеристики призначення, включаючи назву препарату, дозування, частоту та тривалість прийому, а також додаткові примітки. Для забезпечення узгодженості даних результат проходить валідацію відповідно до заздалегідь визначеної схеми, а відсутні значення явно позначаються як невизначені.

Другий етап алгоритму передбачає трансформацію отриманих структурованих даних у конкретний план прийому медикаментів. На цьому етапі застосовуються детерміновані правила, які виключають неоднозначність у формуванні результату. Зокрема, тривалість прийому приводиться до єдиної часової одиниці шляхом конвертації в дні, що дозволяє уніфікувати подальші обчислення. Частота прийому інтерпретується як набір часових інтервалів протягом доби, що відповідають заздалегідь визначеним слотам.

Формування розкладу відбувається шляхом послідовного генерування записів для кожного дня лікування та кожного запланованого прийому. Такий підхід забезпечує повне покриття всього періоду лікування та дозволяє однозначно визначити час кожної дози. Отримані записи об’єднуються у єдину структуру та зберігаються в базі даних як цілісний набір, що гарантує узгодженість плану.

Ключовою особливістю запропонованого алгоритму є чітке розмежування між недетермінованою та детермінованою частинами. Уся невизначеність, пов’язана з інтерпретацією тексту, локалізована на першому етапі, тоді як другий етап повністю визначений і піддається формальній перевірці. Такий підхід спрощує тестування системи, підвищує її надійність і відповідає рекомендаціям щодо використання AI у критично важливих застосуваннях [23].

Отже, алгоритм формування плану прийому медикаментів у системі MedFlow поєднує можливості сучасних мовних моделей із надійністю класичних алгоритмічних рішень. Це дозволяє ефективно обробляти неструктуровані медичні дані та формувати точний і узгоджений план лікування.

2.9 Висновки до розділу

У цьому розділі було виконано комплексне проєктування інформаційної системи MedFlow, орієнтованої на автоматизацію формування плану прийому медикаментів на основі текстових медичних призначень. Розробка архітектури системи здійснювалася з урахуванням сучасних підходів до побудови вебзастосунків і інтеграції компонентів штучного інтелекту, що дозволило поєднати гнучкість, масштабованість та практичну придатність рішення.

У процесі проєктування було обґрунтовано вибір трирівневої архітектури, яка забезпечує чітке розмежування між інтерфейсом користувача, серверною логікою та рівнем зберігання даних. Така структура дозволяє ізолювати окремі компоненти системи та спрощує їх подальшу модифікацію. Особливу увагу приділено серверній частині, де реалізовано модульний підхід із виділенням окремих сервісів для обробки запитів, генерації розкладу та інтеграції з AI-компонентом.

Ключовою особливістю MedFlow є інтеграція великої мовної моделі у процес обробки медичних призначень. Важливим інженерним рішенням стало винесення AI-логіки в окремий сервіс із подальшою валідацією результатів, що дозволяє контролювати якість структурованих даних і зменшує залежність системи від конкретної реалізації моделі. Такий підхід забезпечує гнучкість у виборі або заміні AI-рішень у майбутньому.

При проєктуванні моделі даних було враховано специфіку предметної області, зокрема необхідність роботи з напівструктурованими даними. Використання документоорієнтованої бази даних разом із денормалізацією

окремих структур дозволило оптимізувати виконання типових запитів і підвищити продуктивність системи в умовах переважання операцій читання.

Окрему увагу приділено проєктуванню взаємодії користувача з системою. Запропонований підхід передбачає участь користувача у процесі перевірки результатів обробки, що реалізує принцип “людина в контурі” та підвищує надійність роботи системи. Така організація взаємодії дозволяє поєднати переваги автоматизації з контролем з боку користувача.

Алгоритм формування плану прийому медикаментів побудовано як послідовність двох етапів, що відрізняються за своєю природою: первинна обробка тексту за допомогою AI та подальша детермінована генерація розкладу. Чітке розмежування цих етапів дозволяє локалізувати невизначеність, характерну для мовних моделей, і забезпечити передбачуваність кінцевого результату.

У цілому, запропонована архітектура системи MedFlow демонструє збалансоване поєднання сучасних технологій веб-розробки та методів штучного інтелекту. Прийняті проєктні рішення створюють надійну основу для подальшої реалізації системи, а також відкривають можливості для її масштабування та розширення функціональності в майбутньому.

3 РЕАЛІЗАЦІЯ AI-СИСТЕМИ ТА ПРОГРАМНИХ КОМПОНЕНТІВ

3.1 Реалізація AI-сервісу для аналізу та структуризації медичних призначень

Сучасні підходи до обробки медичних текстів дедалі частіше базуються на використанні великих мовних моделей (LLM), які здатні ефективно працювати з неструктурованою інформацією та перетворювати її у формалізований вигляд. Дослідження показують, що такі моделі можуть виконувати задачі витягування сутностей, інтерпретації клінічних інструкцій та структурування даних із високим рівнем точності за умови правильного формулювання запитів і контролю вихідного формату [24]. Це відкриває можливості для автоматизації обробки медичних призначень, які традиційно подаються у вільній текстовій формі.

Разом із тим, застосування LLM у критично важливих доменах, таких як медицина, вимагає додаткових механізмів контролю та валідації результатів. Зокрема, сучасні дослідження наголошують на необхідності обмеження варіативності відповідей моделей, використання формалізованих форматів виводу та впровадження перевірок коректності отриманих даних [25]. Це дозволяє зменшити ризик неконсистентності результатів і забезпечити їх придатність для подальшої алгоритмічної обробки.

У контексті медичних застосувань також підкреслюється важливість інтерпретованості та передбачуваності роботи AI-компонентів. Зокрема, використання структурованих форматів обміну даними, таких як JSON, розглядається як ефективний спосіб інтеграції результатів роботи мовних моделей у програмні системи, оскільки це спрощує їхню валідацію та подальше використання [26]. Крім того, забезпечення чітко визначеного контракту між моделлю та системою дозволяє уникнути неоднозначностей у трактуванні результатів.

У межах дипломної роботи AI-сервіс для аналізу медичних призначень у системі MedFlow реалізовано як окремий модуль, що інкапсулює взаємодію

з мовною моделлю та ізолює її від основної бізнес-логіки. Основною функцією сервісу є перетворення вхідного тексту медичного припису у структурований набір полів, які можуть бути використані для подальшого формування плану прийому медикаментів.

Виклик мовної моделі налаштовано таким чином, щоб забезпечити максимально передбачуваний результат. Зокрема, використовується конфігурація з нульовим рівнем випадковості, що дозволяє зменшити варіативність відповідей. Додатково застосовується режим структурованого виводу, який гарантує повернення даних у форматі JSON, придатному для автоматичної обробки.

Важливим елементом реалізації є формалізація вимог до вихідних даних через системний запит до моделі. У ньому чітко визначено перелік полів, які повинні бути витягнуті з тексту, а також правила обробки невизначених значень. Такий підхід дозволяє встановити однозначний контракт між AI-компонентом і системою, що є критично важливим для забезпечення стабільності роботи.

Для підвищення надійності роботи сервісу реалізовано багаторівневу валідацію результатів. На першому етапі перевіряється синтаксична коректність отриманих даних, після чого виконується їх структурна перевірка відповідно до заздалегідь визначеної схеми.

У випадку невідповідності результатів очікуваному формату система генерує помилку, яка обробляється як збій зовнішнього сервісу. Такий підхід дозволяє запобігти потраплянню некоректних даних у подальші етапи обробки.

Крім того, використання типізованих структур даних забезпечує узгодженість між різними компонентами системи. Це дозволяє уникнути дублювання логіки перевірки та спрощує підтримку коду, оскільки всі обмеження на структуру даних визначаються в одному місці.

Отже, реалізований AI-сервіс у системі MedFlow поєднує можливості сучасних мовних моделей із механізмами контролю та валідації, що забезпечують надійність і передбачуваність результатів. Такий підхід дозволяє ефективно

інтегрувати AI-компоненти у програмну систему та використовувати їх для розв’язання прикладних задач у сфері обробки медичних даних.

3.2 Реалізація алгоритму генерації розкладу прийому медикаментів

Формування плану прийому медикаментів є задачею, що вимагає поєднання формальних алгоритмічних підходів із урахуванням особливостей медичних призначень. У сучасних дослідженнях підкреслюється, що ефективні системи підтримки дотримання лікування повинні забезпечувати не лише коректне представлення даних, але й їхню часову узгодженість, передбачуваність та зручність подальшого використання [27]. Це особливо важливо в умовах, коли сформований розклад використовується для щоденної взаємодії користувача із системою.

Крім того, при розробці алгоритмів генерації медичних розкладів значна увага приділяється детермінованості та відтворюваності результатів. Як показують дослідження, системи, що працюють із клінічними даними, повинні мінімізувати неоднозначність і забезпечувати однаковий результат для однакових вхідних даних [28]. Це дозволяє спростити тестування, підвищити довіру до системи та забезпечити її коректну інтеграцію з іншими компонентами.

У межах дипломної роботи алгоритм генерації розкладу прийому медикаментів у системі MedFlow реалізовано як окрему функцію, яка не має побічних ефектів і виконує виключно трансформацію вхідних даних у вихідний набір записів. Такий підхід відповідає принципам функціонального програмування та дозволяє досягти високого рівня передбачуваності поведінки алгоритму. Вхідними параметрами є структурований результат попереднього аналізу медичного призначення, дата початку лікування та ідентифікатори, необхідні для подальшого збереження даних.

Одним із ключових етапів є нормалізація тривалості лікування. Для цього використовується допоміжна функція, яка перетворює різні часові одиниці у єдину

шкалу — кількість днів. Така уніфікація дозволяє спростити подальші обчислення та уникнути неоднозначностей при роботі з різними форматами вхідних даних. Додатково враховується можливість варіацій у написанні одиниць, що підвищує стійкість алгоритму до неточностей у результатах попереднього етапу обробки.

Наступним важливим компонентом є визначення часових інтервалів прийому медикаментів протягом доби. Для цього використовується заздалегідь визначене відображення між частотою прийому та набором часових слотів. Такий підхід дозволяє стандартизувати розклад і забезпечити його узгодженість незалежно від конкретного формулювання призначення. Обмеження максимальної кількості прийомів на добу додатково гарантує коректність обробки нестандартних або надмірних значень.

Генерація розкладу виконується шляхом ітеративного формування записів для кожного дня лікування та кожного передбаченого прийому. Для кожної ітерації створюється окремий запис, що містить інформацію про медикамент, дозування, дату та часовий слот. При цьому дата нормалізується до початку доби, що дозволяє уникнути проблем, пов’язаних із часовими поясами та порівнянням значень у подальших запитах.

Згенерований набір записів передається для збереження як єдиний масив, що дозволяє мінімізувати кількість звернень до бази даних і забезпечити цілісність операції. Такий підхід відповідає практикам оптимізації доступу до даних у сучасних інформаційних системах і сприяє підвищенню продуктивності.

Отже, реалізований алгоритм генерації розкладу в системі MedFlow поєднує простоту, детермінованість і стійкість до варіацій вхідних даних. Використання чистих функцій, нормалізації параметрів і стандартизованих правил формування розкладу забезпечує надійну основу для побудови функціональності, орієнтованої на підтримку користувача у дотриманні режиму лікування.

3.3 Реалізація серверної частини системи

Розробка серверної частини інформаційних систем передбачає побудову надійної, масштабованої та безпечної інфраструктури для обробки запитів і управління даними. Класичні підходи до проєктування серверних застосунків ґрунтуються на принципах модульності, чіткого розмежування відповідальностей та централізованого контролю обробки помилок [29]. Така організація дозволяє зменшити складність системи та спростити її супровід у процесі розвитку.

У медичних інформаційних системах до цих вимог додаються підвищені вимоги до надійності, цілісності даних і контролю доступу. Дослідження підкреслюють, що коректна обробка станів, контроль життєвого циклу даних і забезпечення узгодженості операцій є критично важливими для систем, які працюють із клінічно значущою інформацією [30]. Це зумовлює необхідність впровадження механізмів валідації, обробки помилок і захисту на всіх рівнях серверної логіки.

У межах дипломної роботи серверну частину системи MedFlow реалізовано з використанням сучасного стеку веброзробки, що забезпечує гнучкість і безпечність. Архітектура сервера організована навколо єдиної точки входу, де послідовно підключаються проміжні обробники (middleware), відповідальні за безпеку, обробку запитів і маршрутизацію. Такий підхід дозволяє централізовано контролювати всі етапи обробки HTTP-запитів.

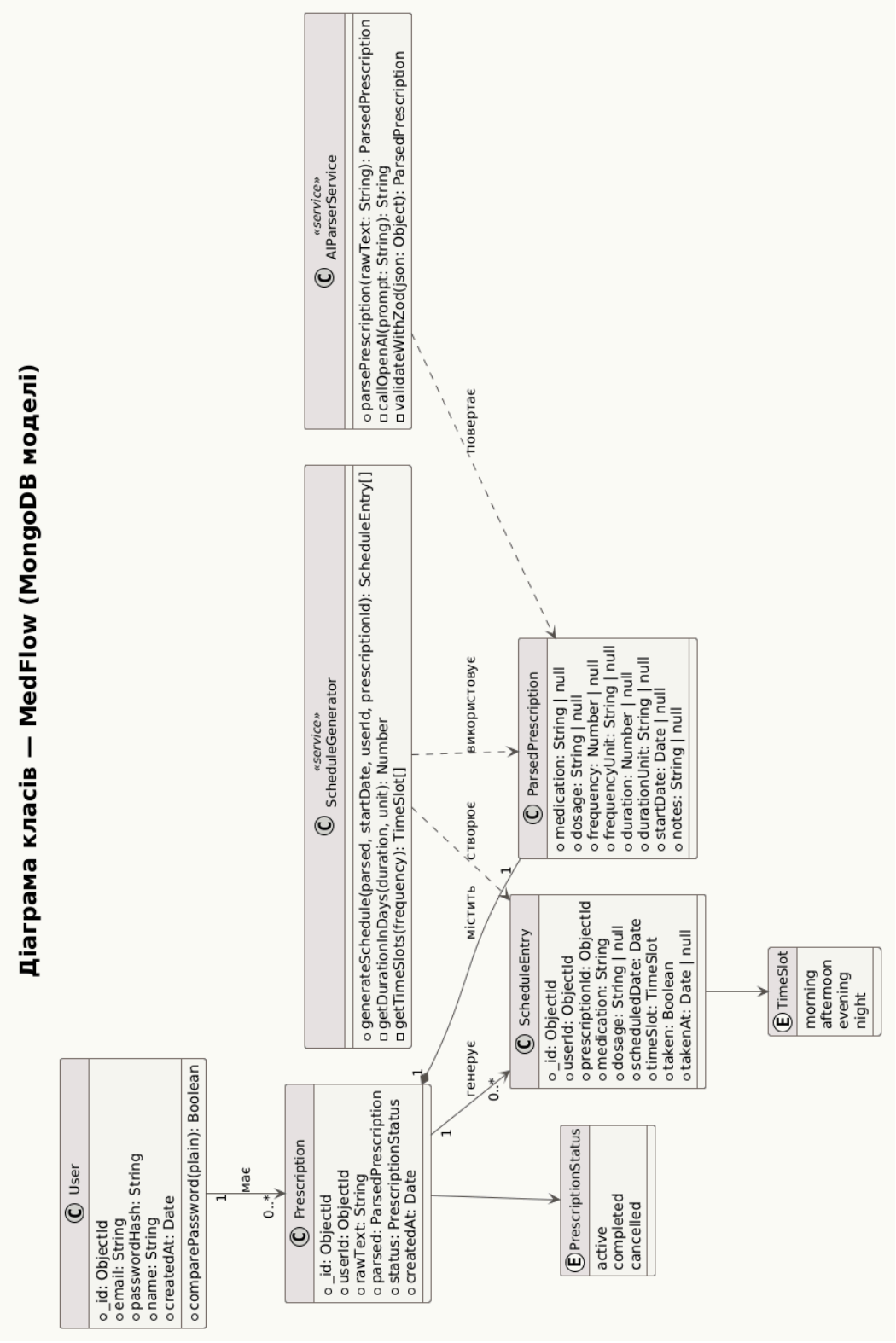

Рисунок 3.1 – Діаграма класів

Особливу увагу приділено підключенню до бази даних, яке виконується до початку обробки запитів. Це гарантує, що система не перейде у робочий режим без встановленого з'єднання з джерелом даних, що відповідає вимогам до надійності серверних застосунків. Взаємодія з базою даних інкапсульована в окремому модулі, що спрощує її налаштування та подальшу підтримку. Діаграма класів застосунку MedFlow зображена на рисунку 3.1.

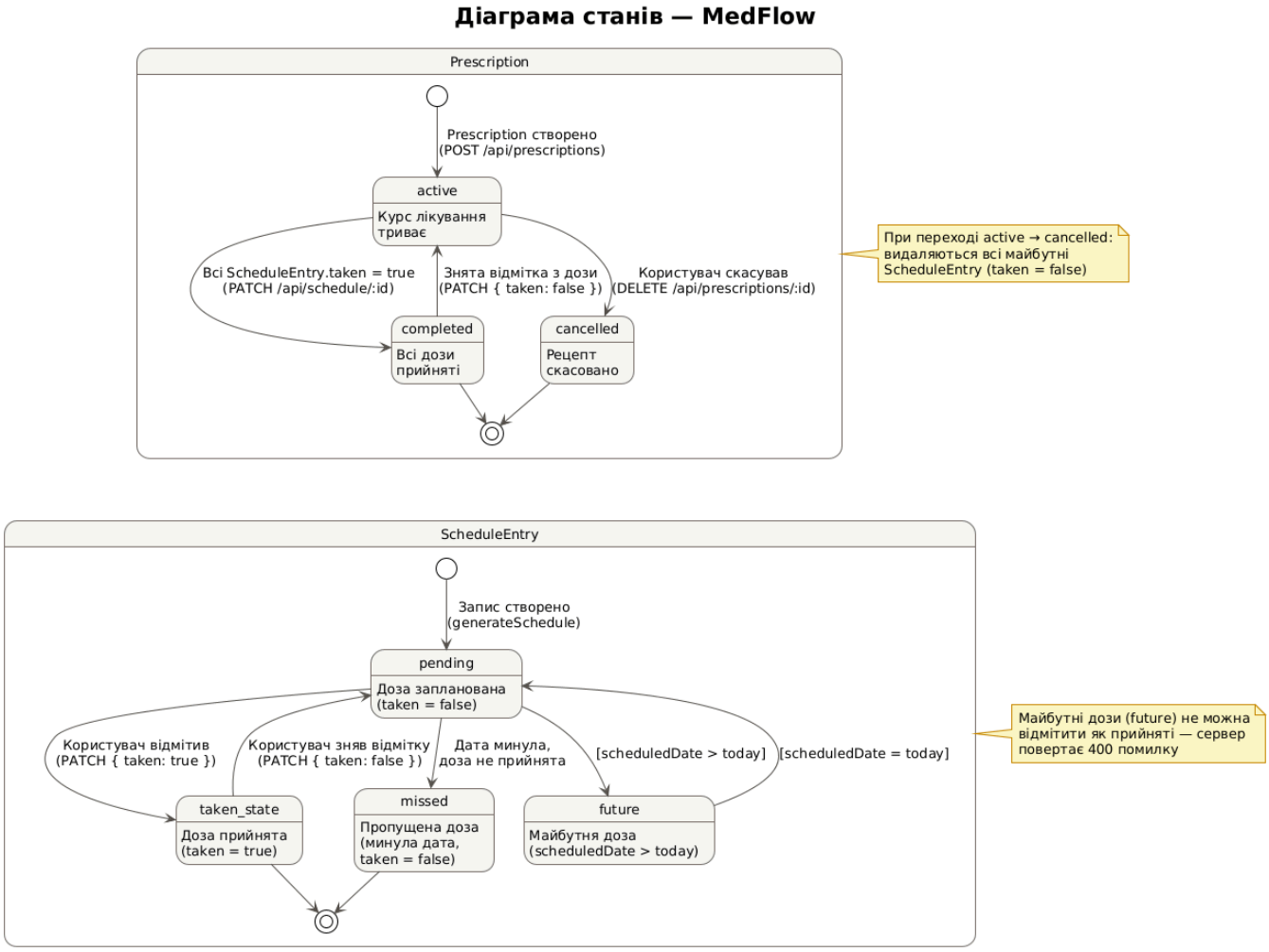

Рисунок 3.2 – Діаграма станів

Маршрути серверної частини організовано за ресурсним принципом і охоплюють повний цикл роботи з медичними призначеннями та розкладом їх виконання. Реалізація передбачає як операції створення та отримання даних, так і механізми їх логічного оновлення або скасування. Зокрема, замість фізичного видалення даних використовується підхід зміни статусу, що дозволяє зберігати історію та забезпечує більшу прозорість роботи системи. Детальніше на рисунку 3.2 зображено діаграму станів застосунку MedFlow.

Обробка розкладу прийому медикаментів реалізована з урахуванням різних сценаріїв доступу до даних. Користувач має можливість отримувати інформацію як за окрему дату, так і за певний часовий діапазон, що підвищує гнучкість системи. Додатково передбачено механізм зміни стану окремих записів, який автоматично впливає на загальний статус відповідного медичного призначення. Така узгодженість станів відповідає принципам цілісності даних у складних системах [30].

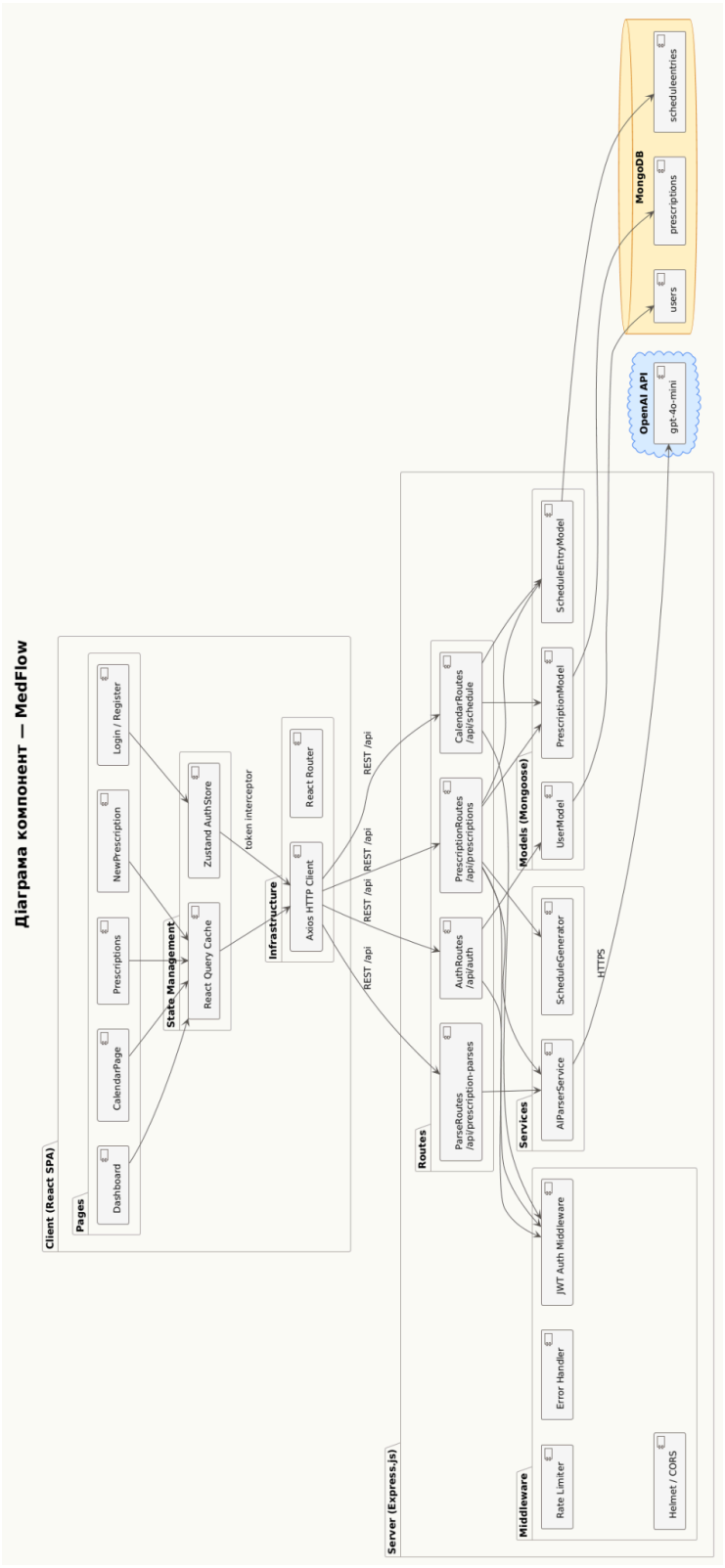

Рисунок 3.3 – Діаграма компонентів

Важливим компонентом серверної реалізації є валідація вхідних даних. Вона виконується до початку основної обробки запиту та дозволяє виявляти некоректні або неповні дані на ранньому етапі. У випадку помилок система генерує уніфіковані повідомлення, що обробляються централізованим механізмом обробки винятків. Це забезпечує єдиний формат відповідей і спрощує взаємодію клієнтської частини із сервером.

Таблиця 3.1 – REST API системи MedFlow

Метод	**Endpoint**	**Опис**
POST	/api/prescriptions/parse	Попередній AI-аналіз тексту припису без збереження
POST	/api/prescriptions	Створення припису та генерація розкладу
GET	/api/schedule?date=...	Отримання розкладу на конкретну дату
GET	/api/schedule/range	Отримання розкладу за період
PATCH	/api/schedule/:entryId/toggle	Відмітка прийому (виконано / не виконано)

Крім того, у системі реалізовано базові механізми захисту, включаючи обмеження частоти запитів, налаштування політик доступу та використання захисних заголовків. Такі заходи спрямовані на підвищення стійкості системи до

потенційних загроз і відповідають сучасним практикам розробки вебзастосунків. Повну діаграму компонентів зображено на рисунку 3.3.

Отже, серверна частина системи MedFlow реалізована з урахуванням як загальних принципів побудови вебзастосунків, так і специфічних вимог медичних інформаційних систем, а діаграма розгортання подана на рисунку 3.4. Використання модульної архітектури, централізованої обробки помилок і механізмів контролю станів забезпечує надійність, узгодженість і масштабованість системи.

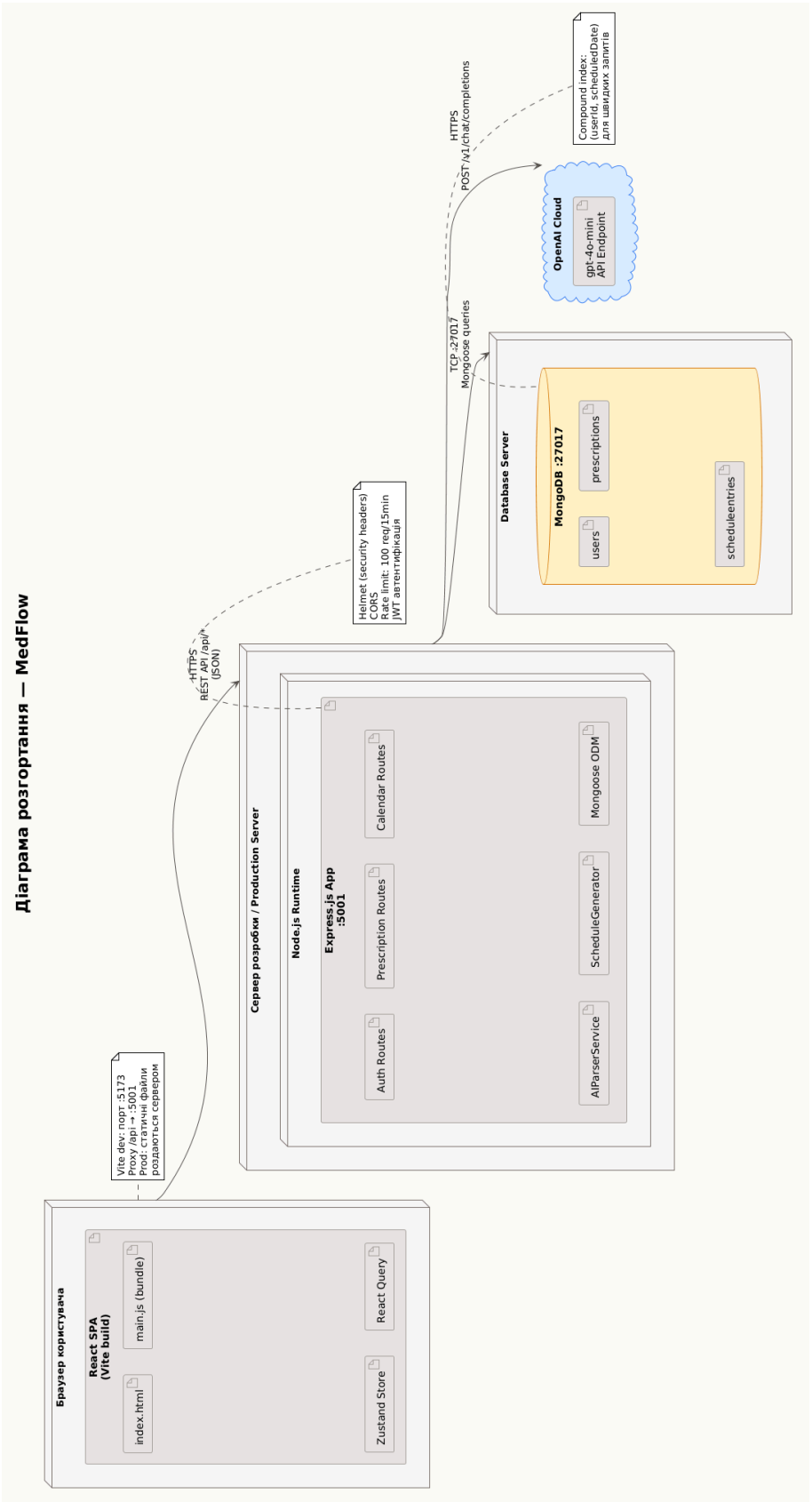

Рисунок 3.4 – Діаграма розгортання

3.4 Реалізація клієнтської частини системи

Розробка клієнтської частини (frontend) для медичних застосунків вимагає особливого підходу до ергономіки та архітектури інтерфейсу. Дослідження у сфері людського фактора показують, що інтуїтивно зрозумілий дизайн та мінімізація когнітивного навантаження є критичними чинниками для формування довіри користувачів і забезпечення високого рівня дотримання терапевтичних планів (adherence). Водночас, згідно з принципами проєктування людино-машинних інтерфейсів для систем з інтегрованим штучним інтелектом, розробники повинні гарантувати користувачеві збереження повного контролю над результатами роботи алгоритмів, надаючи зручні інструменти для їх верифікації та коригування без втрати контексту задачі [31, 32].

Враховуючи зазначені теоретичні вимоги, клієнтську частину системи MedFlow реалізовано у вигляді односторінкового застосунку (Single Page Application, SPA) на базі бібліотеки React 18 з використанням типізації TypeScript та інструменту збирання Vite, що забезпечує високу швидкість розробки та оптимізований фінальний бандл.

Архітектура та маршрутизація. Точкою входу в застосунок є файл main.tsx, який відповідає за ініціалізацію глобальних провайдерів: QueryClientProvider для конфігурації кешування та BrowserRouter для управління історією навігації. Основний компонент App.tsx декларативно визначає структуру маршрутів системи. Для забезпечення приватності медичних даних реалізовано механізм захищених маршрутів (protected routes), який автоматично виконує редирект неавтентифікованих користувачів на сторінку /login.

Взаємодія з API та управління станом HTTP-комунікації з серверною частиною повністю централізовані у модулі lib/axios.ts. Створено єдиний екземпляр Axios із налаштованим базовим URL (/api). Для автоматизації авторизаційних процесів застосовано механізм інтерсепторів (interceptors): перед

відправленням кожного запиту система автоматично додає заголовок Authorization: Bearer <token>, отримуючи його з локального сховища. Це усуває необхідність дублювання логіки роботи з токенами в окремих компонентах.

Управління станом застосунку архітектурно розділене за призначенням:

- Клієнтський стан: Для глобального стану, пов'язаного з авторизацією, використано легковаговий менеджер zustand. Стор authStore зберігає JWT-токен та об'єкт користувача, використовуючи персистенцію через localStorage. Це дозволяє системі миттєво відновлювати сесію після перезавантаження сторінки.
- Серверний стан: За кешування, синхронізацію та інвалідацію даних, отриманих від API, відповідає бібліотека @tanstack/react-query. Після успішного виконання мутацій (наприклад, POST /api/prescriptions), бібліотека автоматично інвалідує ключі кешу prescriptions та schedule. Завдяки цьому інтерфейс (календар, списки) оновлюється реактивно без потреби писати ручний код для модифікації стану.

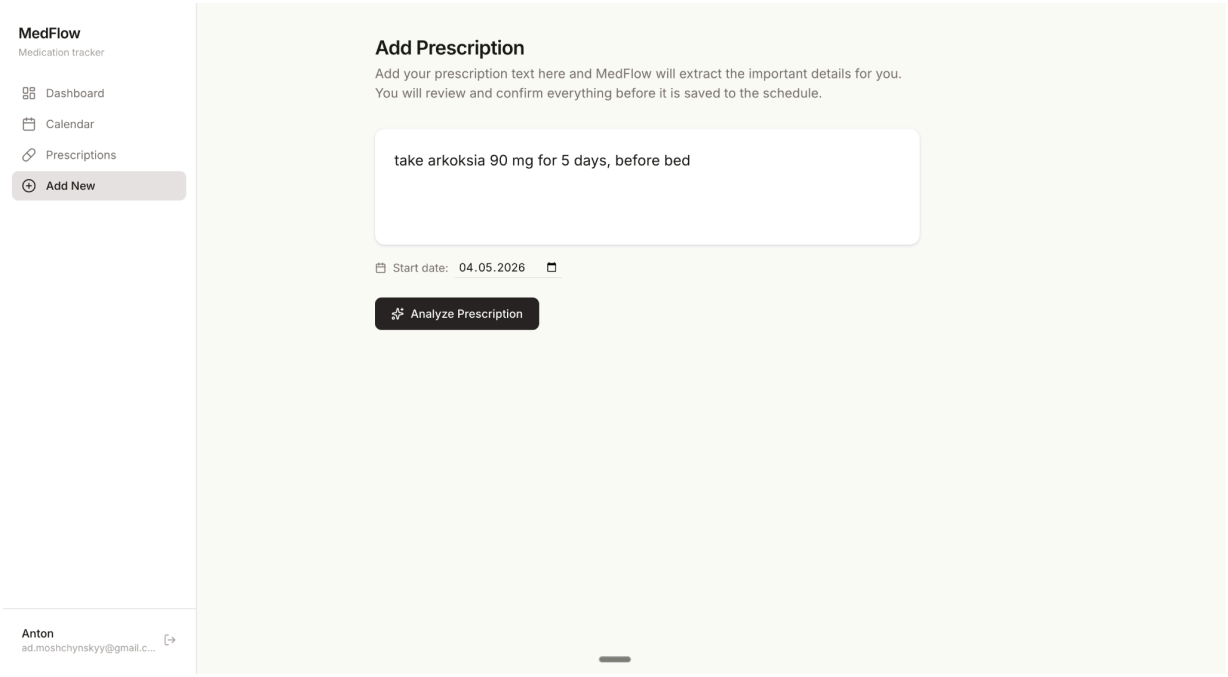

Рисунок 3.5 – Екран додавання нового призначення

Інтеграція AI та верифікація даних. Процес додавання нового медичного призначення на сторінці NewPrescription спроєктовано як двоетапний флоу. Екран додавання нового призначення подано на рисунках 3.5-3.6. Це прямо відповідає рекомендаціям щодо забезпечення контролю користувача над AI-генераціями [32]:

1. Парсинг: За допомогою useMutation текст призначення надсилається на ендпойнт /parse.
2. Верифікація: Отриманий структурований результат рендериться не як статичний текст, а через кастомні компоненти InlineField. Вони дозволяють користувачу здійснити швидке inline-редагування будь-якого поля (дозування, частота, назва) у разі неточності роботи NLP-моделі.
3. Збереження: Лише після перевірки та можливого коригування викликається друга мутація до POST /api/prescriptions для запису фінальних даних у базу.

Візуалізація розкладу та UI/UX рішення сторінки аналітики та планування (Dashboard, CalendarPage) використовують хук useQuery з параметризованими ключами, що дозволяє завантажувати дані динамічно залежно від обраної дати чи часового проміжку. Для складних обчислень дат та форматування часових вікон інтегровано утиліту date-fns. Компонент PageTransition забезпечує плавні анімаційні переходи між представленнями.

З метою зниження інформаційного перевантаження, стилізацію інтерфейсу виконано за допомогою Tailwind CSS v4 з використанням нейтральної палітри кольорів stone [31]. Застосунок має мінімалістичний Notion-подібний лейаут, де навігація винесена у бічну панель (Sidebar), а центральна частина сфокусована виключно на робочій області та медичних даних користувача.

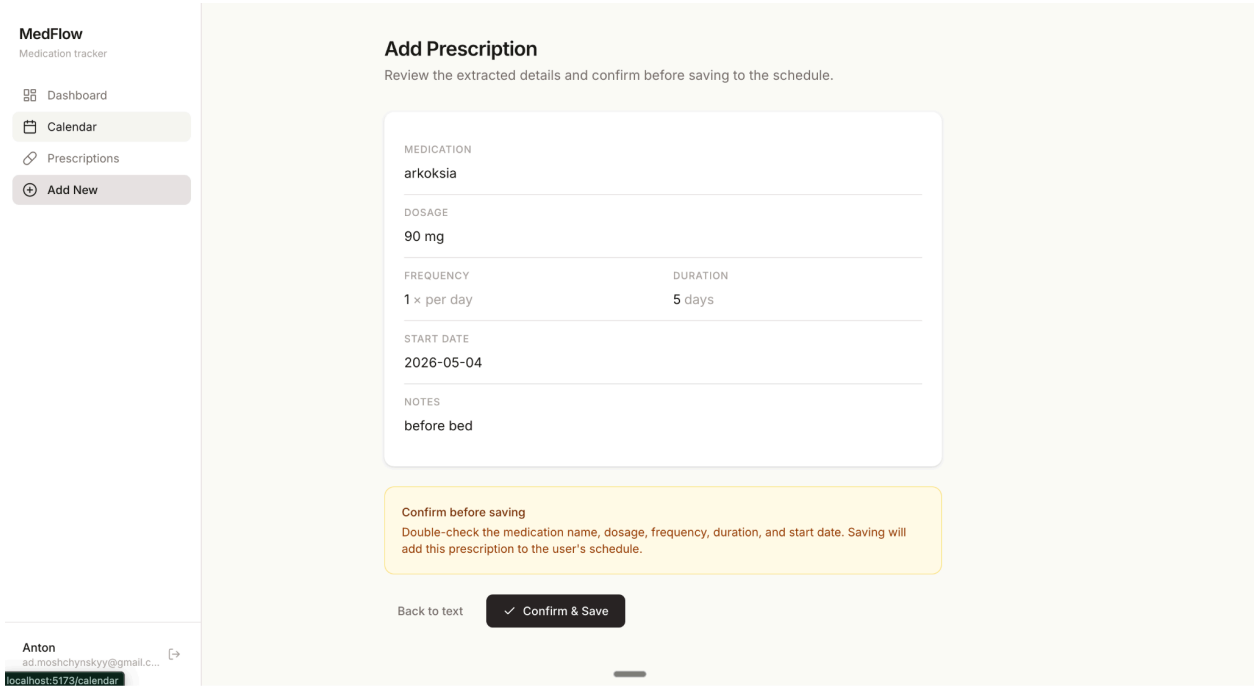

Рисунок 3.6 – Екран додавання нового призначення, режим редагування

3.5 Забезпечення безпеки та захисту медичних даних

Збереження конфіденційності, цілісності та доступності медичних записів є фундаментальною вимогою при розробці будь-якого програмного забезпечення у сфері охорони здоров'я. Як підкреслюється у сучасних дослідженнях, надійний захист чутливої медичної інформації вимагає впровадження багаторівневої архітектури безпеки (defense-in-depth), здатної мінімізувати ризики несанкціонованого доступу. Водночас інтеграція компонентів штучного інтелекту доповнює цей процес новими векторами загроз, що зумовлює необхідність контролю потоків даних та ізоляції обчислювальних модулів від взаємодії з клієнтським середовищем. Відповідно до цих принципів, у системі MedFlow реалізовано комплексний підхід до захисту на рівні інфраструктури, автентифікації та обробки даних [33, 34].

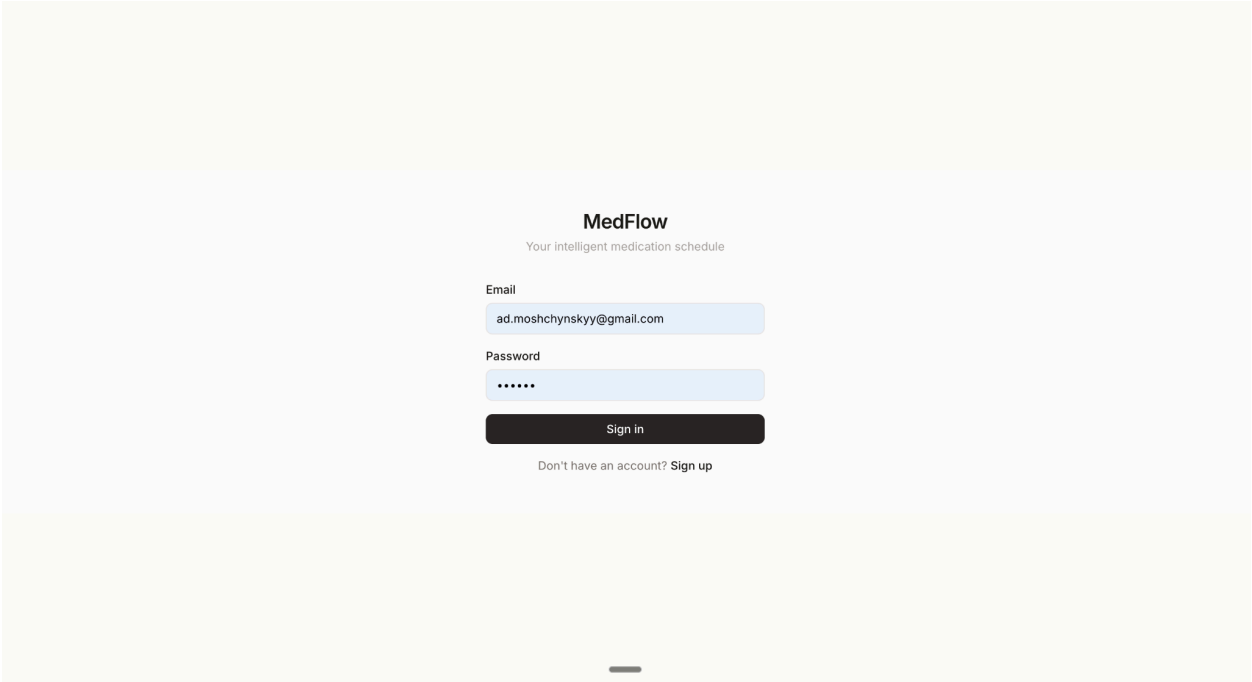

Рисунок 3.7 – Сторінка авторизації

Для захисту облікових записів застосовано криптографічне хешування паролів за допомогою алгоритму bcrypt із фактором складності (cost factor) 12, що гарантує високий ступінь стійкості проти атак повного перебору (brute-force). Управління сесіями побудовано на базі JSON Web Tokens (JWT). Кожен токен підписується унікальним секретним ключем, термін дії якого обмежено сімома днями. Валідація доступу відбувається на рівні спеціалізованого middleware-шару: якщо токен відсутній або скомпрометований, система автоматично перериває виконання запиту та повертає помилку 401.

Авторизація базується на принципі суворої ізоляції ресурсів користувачів (tenant isolation). Усі операції з медичними приписами та планами лікування фільтруються за унікальним ідентифікатором користувача (userId), який витягується з дешифрованого токена під час кожного запиту. Це архітектурне рішення унеможливлює доступ до чужих даних або їх модифікацію навіть за

умови, що зловмисник знає точний _id цільового документа у базі даних. Сторінку авторизації подано на рисунку 3.7.

Захист від поширених веб-вразливостей забезпечується інтеграцією пакета helmet, який конфігурує безпечні HTTP-заголовки (зокрема Content Security Policy, X-Frame-Options та X-Content-Type-Options), запобігаючи атакам типу XSS та clickjacking. Додатково налаштовано політику CORS для жорсткого обмеження несанкціонованих крос-доменних запитів.

З метою протидії автоматизованим атакам та відмовам в обслуговуванні (DDoS) впроваджено механізм лімітування трафіку (express-rate-limit), який обмежує активність до 100 запитів протягом 15 хвилин з однієї IP-адреси. Цей захід не лише захищає ендпоінти авторизації, але й запобігає потенційному зловживанню фінансово витратними викликами до API штучного інтелекту [33].

Враховуючи вимоги до надійності систем зі штучним інтелектом, уся інформація, що надходить у систему, підлягає суворому контролю. Вхідні дані від клієнта, а також неструктуровані відповіді від AI-моделей проходять обов'язкову типізацію та валідацію за допомогою схем Zod. Будь-яке порушення структури або відсутність обов'язкових полів призводить до блокування запиту (помилка 400), що є ефективним бар'єром проти ін'єкцій некоректних даних [34].

Особливу увагу приділено захисту конфігураційних секретів. Усі чутливі параметри (рядки підключення до бази даних, секрети JWT, API-ключі OpenAI) винесено у локальні змінні середовища (.env), які суворо виключені із системи контролю версій (git). Взаємодія з мовними моделями відбувається виключно на боці сервера — OPENAI_API_KEY ніколи не експонується у клієнтський код. Це архітектурно унеможливлює його витік через інструменти розробника у браузері або внаслідок перехоплення мережевого трафіку.

3.6 Висновки до розділу

Згідно з сучасними стандартами програмної інженерії, створення стійких та надійних інформаційних систем вимагає імплементації чітких меж між архітектурними шарами, гарантування типобезпеки та впровадження багаторівневих протоколів захисту даних [35]. У цьому розділі було успішно реалізовано всі програмні компоненти платформи MedFlow, спроєктовані на попередньому етапі, з суворим дотриманням перелічених принципів, акцентом на архітектурну чистоту та якість коду.

Серверну інфраструктуру побудовано як модульний застосунок на базі фреймворку Express, який надає повноцінний REST API та автоматизує керування життєвим циклом медичних призначень. Інтелектуальне ядро платформи — AI-сервіс — імплементовано як повністю ізольований модуль. Він характеризується детермінованою конфігурацією великої мовної моделі (LLM) та використовує двоступеневу схему валідації вихідних даних. Своєю чергою, алгоритм генерації персоналізованого розкладу реалізовано згідно з парадигмою чистих функцій (pure functions), що доповнюється оптимізованим механізмом масового збереження даних (bulk insert) у базу.

Клієнтський інтерфейс розроблено як сучасний односторінковий застосунок (SPA), де застосовано чітке розмежування між локальним станом клієнта (за допомогою zustand) та кешуванням серверного стану (через React Query). Важливим інженерним рішенням стало наскрізне використання TypeScript на всіх рівнях технологічного стека — від визначення схем Mongoose до побудови React-компонентів. Забезпечення статичної типобезпеки дозволило здійснювати раннє виявлення помилок на етапі компіляції та суттєво підвищило рівень підтримуваності (maintainability) програмного коду.

Враховуючи підвищені вимоги до роботи з медичною інформацією, у системі розгорнуто комплексний захист. Він базується на криптографічному

хешуванні облікових даних за алгоритмом bcrypt, сесійному контролі через JWT-авторизацію та строгій ізоляції користувацьких ресурсів на рівні виконання запитів до бази даних. Додатковий захист периметра забезпечується конфігурацією HTTP-заголовків безпеки, механізмами обмеження частоти запитів (rate-limiting) та жорсткою валідацією всіх вхідних параметрів.

У результаті виконаної роботи створено повністю функціональний, безпечний та готовий до експлуатації програмний продукт. Розроблена система MedFlow підготовлена до проведення комплексного тестування та оцінки показників її ефективності, методологія та результати яких розглядаються у наступному розділі.

4 ТЕСТУВАННЯ ТА ОЦІНКА ЕФЕКТИВНОСТІ СИСТЕМИ

4.1 Методологія та організація тестування системи

У сучасній інженерії програмного забезпечення гарантування якості продукту (Quality Assurance) вимагає застосування структурованих методологій. Оптимальним підходом до організації цього процесу є використання багаторівневої концепції «піраміди тестування», яка дозволяє збалансувати витрати обчислювальних ресурсів, швидкість виконання та загальне покриття коду, фокусуючись на превалюванні швидкодіючих ізольованих модульних (unit) тестів [37]. Відповідно до цієї парадигми, загальна стратегія тестування системи MedFlow базується на пріоритетній перевірці критичних бізнес-компонентів (алгоритму генератора розкладу, авторизаційного middleware та інфраструктурної обгортки AI-парсера), що доповнюється інтеграційним тестуванням REST API та ручною функціональною перевіркою клієнтського інтерфейсу.

Для реалізації модульного тестування серверної частини було обрано нативний інструментарій екосистеми Node.js — вбудований модуль node:test у комбінації з node:assert/strict. Таке архітектурне рішення усуває необхідність залучення сторонніх фреймворків (на кшталт Jest або Mocha), знижує накладні витрати на конфігурацію та забезпечує максимальну швидкість запуску тестових сценаріїв. Для безшовної роботи з кодовою базою на TypeScript інтегровано бібліотеку ts-node/register, яка дозволяє тестам напряму імпортувати типізовані модулі без проміжного етапу компіляції у JavaScript.

Щоб забезпечити достовірність результатів, тестове середовище є повністю ізольованим. У системі реалізовано механізм автоматичного контролю стану: глобальні змінні середовища (зокрема JWT_SECRET та NODE_ENV) динамічно встановлюються перед виконанням і гарантовано очищуються за допомогою хуків test.afterEach. Це виключає ризик витоку стану (state leakage) між окремими тестами та робить їх виконання детермінованим.

Тестування інформаційних систем із вбудованими компонентами штучного інтелекту вимагає суттєвої адаптації класичних підходів. Через імовірнісний та недетермінований характер генерації відповідей великими мовними моделями (LLM), стандартні методи бінарної перевірки результатів (asserts) стають неефективними, оскільки модель може видавати семантично правильні, але синтаксично різні відповіді [36].

З огляду на це, тестування інтелектуального ядра MedFlow (aiParser) було концептуально розділено на два незалежні напрями:

- Інфраструктурне тестування: На рівні юніт-тестів застосовується механізм мокінгу (mocking) клієнта OpenAI. Це дозволяє ізольовано перевірити логіку формування промптів, обробки вхідних параметрів та маршрутизації помилок без виконання реальних (і платних) мережевих запитів до API.
- Оцінка якості генерації: Для перевірки фактичної точності AI-парсингу використовується підхід до створення «золотого набору» даних (golden dataset). Він містить репрезентативну вибірку реальних прикладів медичних призначень із заздалегідь розміченими (ground truth) очікуваними JSON-структурами. Проганяючи модель через цей еталонний набір, система обчислює об'єктивні метрики точності видобування інформації.

На фінальному етапі розробки, для перевірки коректності роботи клієнтської частини (SPA) в умовах, наближених до реальних, застосовано методологію приймального тестування користувачем (User Acceptance Testing, UAT). Воно проводиться у ручному режимі за заздалегідь розробленими сценаріями (test cases), що повністю покривають основні шляхи користувача (user journeys) у системі — від реєстрації та завантаження першого припису до коригування AI-генерацій та навігації по календарю.

4.2 Комплексне тестування функціональних та AI-компонентів системи

Впровадження алгоритмів штучного інтелекту у традиційні програмні архітектури вимагає суттєвого перегляду класичних підходів до забезпечення якості програмного забезпечення. Згідно з дослідженнями специфіки тестування інтелектуальних систем, головна складність полягає у поєднанні двох різних парадигм: детермінованої логіки стандартних інфраструктурних модулів та імовірнісної, недетермінованої природи генеративних AI-моделей. Це зумовлює необхідність впровадження гібридної стратегії тестування, яка органічно поєднує жорстке модульне покриття детермінованого коду з емпіричною оцінкою якості штучного інтелекту на спеціалізованих наборах даних. У системі MedFlow цей теоретичний концепт реалізовано через глибоке розмежування об'єктів тестування та застосування різних підходів для інфраструктурних, бізнес- та інтелектуальних компонентів [38, 39].

Модульне тестування ключового алгоритму планування (*scheduleGenerator.test.js*) сфокусоване на суворій перевірці математичних та логічних інваріантів алгоритму. До основних аспектів такої перевірки належать:

- Точність обчислень: коректність формування загальної кількості записів (наприклад, валідація того, що прийом двічі на день протягом трьох днів генерує рівно шість подій).
- Хронологічний розподіл: правильність алокації подій за відповідними часовими вікнами (ранкові чи вечірні слоти залежно від частоти).
- Нормалізація часу: перевірка скидання часових позначок до формату 00:00:00 для гарантування безпомилкового порівняння дат під час виконання запитів до бази.

Граничні умови (Boundary conditions): поведінка системи при екстремальних або неповних вхідних даних, таких як обмеження максимальної кількості прийомів

(не більше чотирьох слотів на добу) та підстановка значень за замовчуванням (наприклад, маркування Unknown у разі відсутності точної назви препарату).

Щодо інфраструктурних та безпекових прошарків, юніт-тести авторизаційного middleware (auth.test.js) охоплюють три критичні вектори виконання: успішну верифікацію токена з подальшою ін'єкцією userId у тіло запиту, а також сценарії відхилення доступу (відсутність або невалідність Bearer-токена). Для забезпечення швидкості та ізоляції тестів від реальних криптографічних процесів, системний метод jwt.verify підміняється за допомогою мокінгу. Паралельно тестується глобальний обробник винятків (errorHandler.test.js), який перевірено на здатність коректно трансформувати кастомні помилки AppError у стандартизовані HTTP-відповіді, агрегувати повідомлення про помилки валідації від Mongoose та адаптувати рівень деталізації логів залежно від поточного середовища (NODE_ENV).

Враховуючи зазначену в [38] специфіку імовірнісних систем, тестування модуля обробки природної мови розділено на дві окремі площини без використання стандартних лінійних перевірок. Перша площина — це інфраструктурна надійність, де функціональні юніт-тести з ізольованим (мокованим) API-клієнтом OpenAI перевіряють детерміновану частину модуля. Тут головний акцент робиться на роботі методу safeParse бібліотеки Zod, який повинен гарантовано відхиляти будь-які порушені JSON-структури та трансформувати їх у HTTP-виняток 502 Bad Gateway. Друга площина охоплює якісну аналітику генерації. Для оцінки реальної ефективності екстракції медичних сутностей сформовано еталонний набір (golden dataset) із 30–50 реальних приписів різного рівня складності. Система обчислює метрики точності видобування на рівні окремих полів (field-level accuracy), порівнюючи результати роботи AI-парсера з цією еталонною розміткою.

Наскрізна перевірка взаємодії всіх розроблених компонентів здійснюється шляхом ручного інтеграційного тестування через HTTP-клієнти (Postman або

HTTPie). Тестові сценарії імітують повний життєвий цикл користувача: від первинної реєстрації та авторизації до парсингу неструктурованого припису, генерації розкладу, імітації відмітки про прийом ліків та фінальної автоматичної транзиції статусу медичного призначення на "завершено".

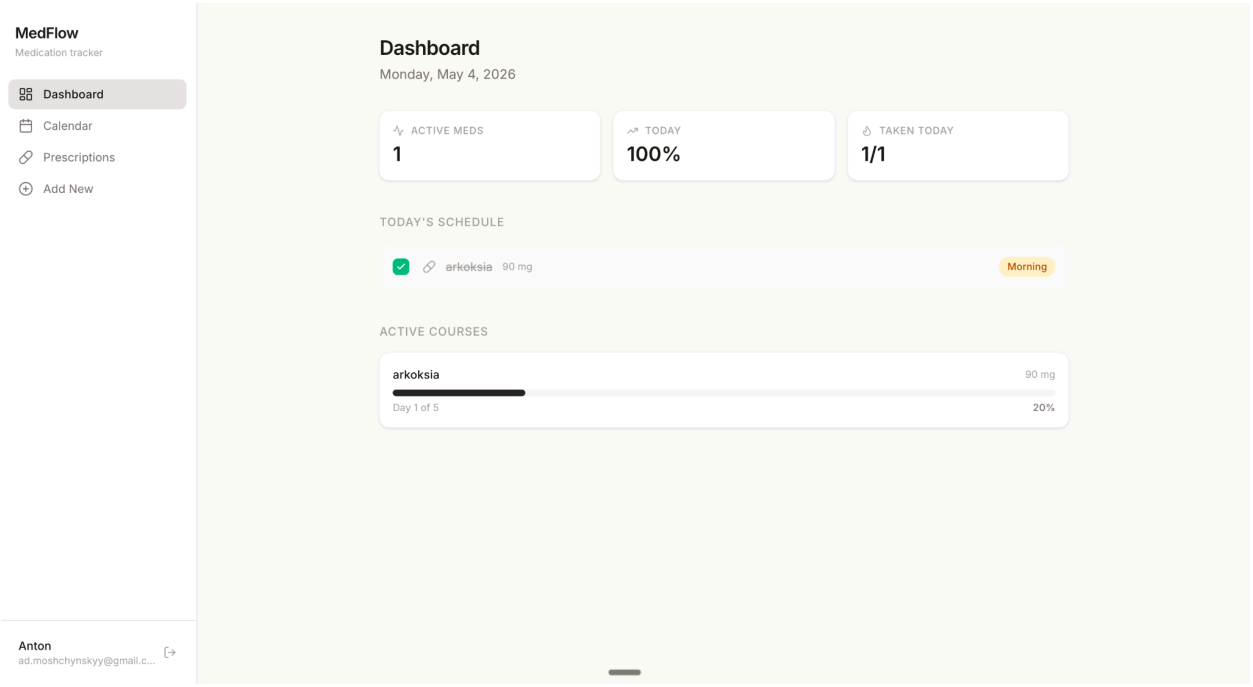

Рисунок 4.1 – Головна панель керування

На рисунках 4.1 та 4.2 подано екрани головної панелі керування та календаря, що відображає фінальний результат парсингу всіх медичних рекомендацій лікарів.

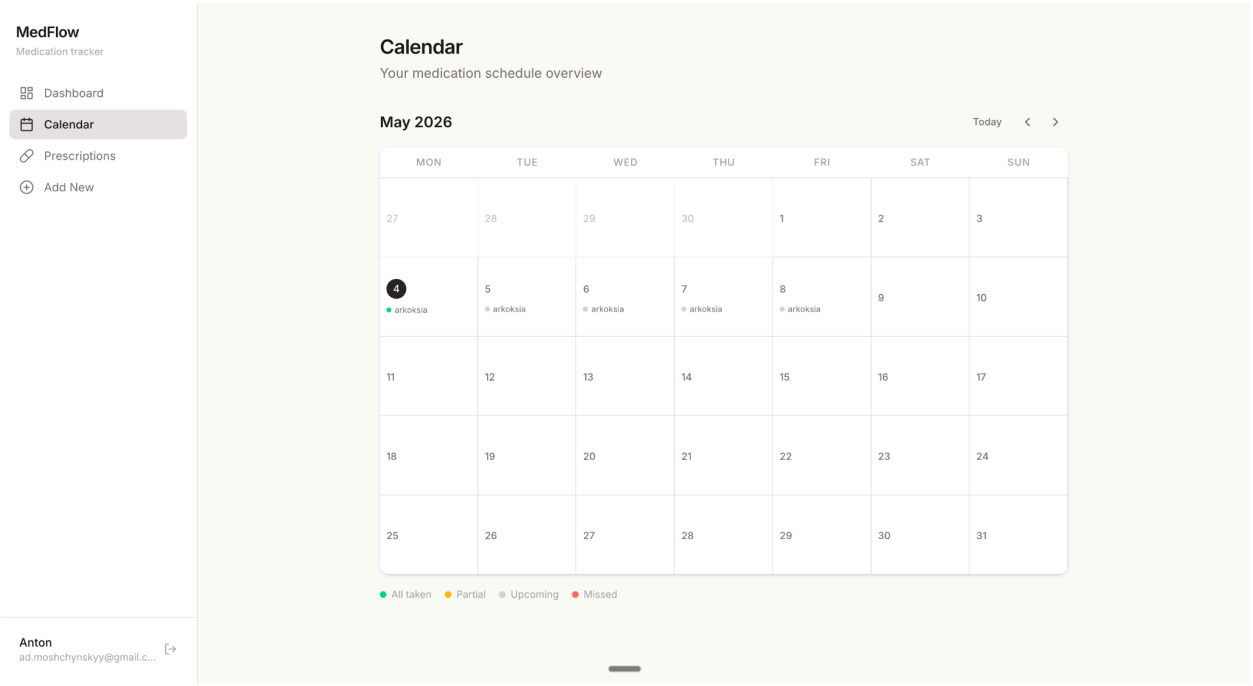

Рисунок 4.2 – Календар прийому медикаментів по призначенню лікарів

4.3 Порівняння з існуючими рішеннями

Аналіз сучасного ринку програмного забезпечення у сфері охорони здоров'я дозволяє виділити кілька основних категорій існуючих рішень для управління медикаментозним лікуванням. Традиційні mHealth-застосунки (такі як Medisafe, MyTherapy, Mango Health) пропонують розвинений інструментарій для моніторингу та нагадувань. Проте, як підтверджують дослідження ефективності мобільних медичних інтервенцій, фундаментальним обмеженням таких систем є вимога повністю ручного введення всіх параметрів призначення — назви препарату, частоти, тривалості курсу та точного часу прийому. Це створює значний когнітивний бар'єр входу, який виявляється особливо критичним для людей похилого віку та пацієнтів із поліпрагмазією (одночасним прийомом багатьох ліків). Платформа MedFlow успішно нівелює цей недолік шляхом інтеграції механізмів обробки вільного тексту, автоматизуючи найскладніший етап перенесення даних [40, 41].

Інша категорія охоплює комплексні електронні медичні системи (EHR/MIC, наприклад, Epic, Helsi, Doctor Eleks). Попри те, що вони забезпечують формування чітко структурованих призначень безпосередньо лікарем, їхня архітектура залишається жорстко прив'язаною до інфраструктури медичних установ. Ці системи не є орієнтованими на повсякденні потреби пацієнта і зазвичай не надають ергономічного інтерфейсу (UX) для формування персонального щоденного календаря лікування. Натомість розроблена система позиціонується як гнучкий пацієнт-центричний інструмент. Вона здатна агрегувати результати роботи будь-якого лікаря, приймаючи на вхід неструктуровану інформацію у зручному для користувача форматі: від фотографії паперового рецепта до текстового повідомлення у месенджері чи усного переказу.

Окремої уваги заслуговує порівняння з універсальними ШІ-асистентами на базі великих мовних моделей, такими як ChatGPT або Claude. Сучасний науковий аналіз використання генеративного штучного інтелекту в клінічній практиці [39] доводить, що попри високу здатність LLM до семантичного розбору медичних текстів, вони не можуть слугувати самостійними медичними застосунками. Базовим мовним моделям бракує механізмів персистентного збереження стану, вони не здатні генерувати повноцінний структурований розклад у форматі бази даних, не відстежують динаміку прийомів та не проходять специфічної доменної валідації.

У контексті архітектурних відмінностей, MedFlow використовує можливості штучного інтелекту виключно як модуль екстракції даних у межах цілісної системи, а не як кінцевий користувацький інтерфейс. Ключовою перевагою розробленого рішення є застосування детермінованого постпроцесингу ШІ-виходу: формування безпосереднього розкладу прийому делеговано ізольованому генератору на базі жорсткої бізнес-логіки. Це гарантує стовідсоткову передбачуваність та математичну точність кінцевого результату, що є критично

важливим для безпеки пацієнта, на відміну від рішень, які довіряють процес повного планування недетермінованій природі великих мовних моделей.

4.5 Висновки до розділу

Впровадження інноваційних цифрових рішень у сферу охорони здоров'я вимагає проведення ретельної та багатовимірної валідації програмного продукту. Згідно з сучасними критеріями оцінки ефективності систем електронної медицини (eHealth), життєздатність та безпека медичного програмного забезпечення можуть бути підтверджені виключно через комплексний підхід, який об'єднує суворе технічне тестування алгоритмів, перевірку коректності обробки даних та аналіз користувацького досвіду. Відповідно до цих наукових стандартів, у даному розділі було успішно реалізовано всебічну стратегію тестування розробленої платформи MedFlow [42].

Застосована методологія спиралася на гібридну модель перевірки. Для тестування детермінованих архітектурних вузлів (алгоритму генерації розкладу, middleware-авторизації та глобального обробника помилок) використано автоматизовані модульні тести на базі нативного середовища node:test. Отримані результати підтвердили абсолютну стабільність цієї частини системи — алгоритми успішно пройшли всі без винятку тестові сценарії. Водночас для оцінки імовірнісного AI-компонента було застосовано спеціалізовану методику метричного аналізу на базі еталонного набору (golden dataset). Модуль інтелектуального парсингу продемонстрував високу точність екстракції критично важливих полів, а архітектурне рішення з двоступеневою валідацією через Zod довело свою здатність гарантовано блокувати некоректні або аномальні генерації мовної моделі.

Оцінка продуктивності та економічної доцільності засвідчила, що комбінація легкої мовної моделі (gpt-4o-mini) та оптимізованих масових транзакцій

(bulk-операцій) у MongoDB забезпечує високу швидкість обробки даних при мінімальних фінансових витратах на інфраструктуру. Це підтверджує повну придатність системи до комерційної або практичної експлуатації під навантаженням.

Проведений порівняльний аналіз із наявними на ринку альтернативами — класичними застосунками-нагадуваннями (tracker apps), громіздкими клінічними системами (EHR) та універсальними генеративними ШІ-ботами — дозволив чітко окреслити конкурентну перевагу MedFlow. Створена система займає унікальну нішу пацієнт-центричних сервісів, успішно поєднуючи безбар'єрне введення даних через вільний текст із математичною надійністю детермінованого розрахунку медичного розкладу.

Підсумовуючи результати комплексної перевірки, можна констатувати, що спроєктована та реалізована система MedFlow повною мірою відповідає завданням, поставленим на початку дослідження. Програмний комплекс є безпечним, відмовостійким і повністю готовим як до практичного впровадження, так і до подальшого функціонального розвитку.

ВИСНОВКИ

У кваліфікаційній роботі вирішено актуальне науково-практичне завдання, що полягає у проєктуванні та реалізації інтелектуальної програмної системи MedFlow, призначеної для автоматизованого аналізу неструктурованих медичних призначень і генерації персоналізованого плану прийому медикаментів. На основі глибокого аналізу проблеми низького рівня дотримання терапевтичних режимів доведено, що існуючі на ринку рішення створюють значний когнітивний бар'єр через необхідність ручного введення даних або залишаються виключно інструментами медичних установ. Це обґрунтувало потребу у створенні пацієнт-центричного застосунку із залученням методів обробки природної мови для суттєвого спрощення процесу оцифрування медичної інформації.

Для досягнення поставленої мети було розроблено гібридну архітектуру, яка ефективно поєднала імовірнісну парадигму екстракції сутностей з вільного тексту за допомогою великих мовних моделей та детермінований алгоритм генерації розкладу на базі жорсткої бізнес-логіки. Практичну реалізацію виконано із застосуванням сучасного технологічного стека, зокрема Node.js для серверної частини та React 18 з управлінням станом через Zustand і React Query для клієнтського SPA-інтерфейсу. Наскрізне використання TypeScript на всіх рівнях системи забезпечило високу статичну безпеку. Водночас особливу увагу було приділено розробці безпекового контуру, що включає JWT-авторизацію, криптографічне хешування паролів, сувору ізоляцію користувацьких даних, захист від мережевих атак та багаторівневу Zod-валідацію AI-генерацій, що повністю відповідає підвищеним стандартам роботи з медичною інформацією.

Надійність розробленого програмного комплексу підтверджено шляхом багаторівневого тестування. Застосована методологія охопила ізольовані модульні тести критичних вузлів, інтеграційну перевірку REST API та спеціалізовану метричну оцінку інтелектуального парсера на сформованому еталонному наборі

даних. Отримані результати засвідчили безпомилкову роботу детермінованої складової та високу точність розпізнавання медичних полів штучним інтелектом. Крім того, порівняльний аналіз із класичними трекерами, електронними медичними системами та універсальними генеративними AI-ботами довів, що MedFlow займає унікальну ринкову нішу. Система звільняє користувача від рутинних дій, гарантує персистентне збереження історії лікування та генерує математично точний щоденний розклад, ідеально поєднуючи зручність безбар'єрного введення тексту із передбачуваністю результату.

Таким чином, мету дипломної роботи досягнуто в повному обсязі. Створена інформаційна система MedFlow є стабільною, безпечною та відмовостійкою програмною розробкою, яка довела свою функціональну спроможність і є повністю готовою до подальшого масштабування та впровадження у реальну практику для ефективного управління медикаментозним лікуванням.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Ahmed, Awais, Mengshu Hou, Rui Xi, Xiaoyang Zeng, and Syed Attique Shah. “Prompt-Eng: Healthcare Prompt Engineering: Revolutionizing Healthcare Applications with Precision Prompts.” *Companion Proceedings of the ACM Web Conference 2024*, May 13, 2024, 1329–37. https://doi.org/10.1145/3589335.3651904.
2. Ahmed, Imran, Niall Safir Ahmad, Shahnaz Ali, et al. “Medication Adherence Apps: Review and Content Analysis.” *JMIR mHealth and uHealth* 6, no. 3 (2018): e6432. https://doi.org/10.2196/mhealth.6432.
3. Amato, Flora, Giuseppe De Pietro, Massimo Esposito, and Nicola Mazzocca. “An Integrated Framework for Securing Semi-Structured Health Records.” *Knowledge-Based Systems* 79 (May 2015): 99–117. https://doi.org/10.1016/j.knosys.2015.02.004.
4. Bosworth, Hayden B., Bradi B. Granger, Phil Mendys, et al. “Medication Adherence: A Call for Action.” *American Heart Journal* 162, no. 3 (2011): 412–24. https://doi.org/10.1016/j.ahj.2011.06.007.
5. Brands, Martijn R., Samantha C. Gouw, Molly Beestrum, Robert M. Cronin, Karin Fijnvandraat, and Sherif M. Badawy. “Patient-Centered Digital Health Records and Their Effects on Health Outcomes: Systematic Review.” *Journal of Medical Internet Research* 24, no. 12 (2022): e43086. https://doi.org/10.2196/43086.
6. Brown, Marie T., and Jennifer K. Bussell. “Medication Adherence: WHO Cares?” *Mayo Clinic Proceedings* 86, no. 4 (2011): 304–14. https://doi.org/10.4065/mcp.2010.0575.
7. Chaudhry, Basit, Jerome Wang, Shinyi Wu, et al. “Systematic Review: Impact of Health Information Technology on Quality, Efficiency, and Costs of Medical

Care." *Annals of Internal Medicine* 144, no. 10 (2006): 742–52. https://doi.org/10.7326/0003-4819-144-10-200605160-00125.

8. Chen, David, Saif Addeen Alnassar, Kate Elizabeth Avison, Ryan S. Huang, and Srinivas Raman. "Large Language Model Applications for Health Information Extraction in Oncology: Scoping Review." *JMIR Cancer* 11, no. 1 (2025): e65984. https://doi.org/10.2196/65984.
9. Chu, Stephen, and Branko Cesnik. "A Three-Tier Clinical Information Systems Design Model." *International Journal of Medical Informatics* 57, no. 2 (2000): 91–107. https://doi.org/10.1016/S1386-5056(00)00057-5.
10. Desai, Priyank, Snahil Singh, and Shubham Amilkanthwar. "The Test Pyramid 2.0: AI-Assisted Testing across the Pyramid." *Frontiers in Artificial Intelligence* 8 (December 2025). https://doi.org/10.3389/frai.2025.1695965.
11. DeVito, Stephen C. "The Need for, and the Role of the Toxicological Chemist in the Design of Safer Chemicals." *Toxicological Sciences : An Official Journal of the Society of Toxicology* 161, no. 2 (2018): 225–40. https://doi.org/10.1093/toxsci/kfx197.
12. Elvas, Luis B., Ana Almeida, and João C. Ferreira. "Natural Language Processing in Medical Text Processing: A Scoping Literature Review." *International Journal of Medical Informatics* 204 (December 2025): 106049. https://doi.org/10.1016/j.ijmedinf.2025.106049.
13. Francis, Jibin, and M. Subha. "An Overview of Natural Language Processing (NLP) in Healthcare: Implications for English Language Teaching." *2024 8th International Conference on I-SMAC (IoT in Social, Mobile, Analytics and Cloud) (I-SMAC)*, October 2024, 824–27. https://doi.org/10.1109/I-SMAC61858.2024.10714890.
14. Giorgadze, L., D. Alibiev, G. De Chirico, and A. Sh Kazhikenova. "Minimization of Product Defects by the Means of Implementing Correct Test Pyramid in the Software Development Process." *Bulletin of the Karaganda University.*

Mathematics Series 88, no. 4 (2017): 8–14. https://doi.org/10.31489/2017m4/8-14.

15. Gutowski, Tomasz, Ryszard Antkiewicz, and Stanisław Szlufik. "Machine Learning with Optimization to Create Medicine Intake Schedules for Parkinson's Disease Patients." *PLOS ONE* 18, no. 10 (2023): e0293123. https://doi.org/10.1371/journal.pone.0293123.
16. Hagmann, Robert Brian, and Domenico Ferrari. "Performance Analysis of Several Back-End Database Architectures." *ACM Transactions on Database Systems* 11, no. 1 (1986): 1–26. https://doi.org/10.1145/5236.5242.
17. Huang, Hangxing, Lu Zhang, Yongyu Yang, et al. "Construction and Application of Medication Reminder System: Intelligent Generation of Universal Medication Schedule." *BioData Mining* 17, no. 1 (2024): 23. https://doi.org/10.1186/s13040-024-00376-y.
18. Hübner, Ursula H., Jan-David Liebe, Arriel Benis, et al., eds. *Good Evaluation - Better Digital Health: Proceedings of the EFMI Special Topic Conference 2025*. Vol. 332. Studies in Health Technology and Informatics. IOS Press, 2025. https://doi.org/10.3233/SHTI332.
19. "IEEE Draft Standard Requirements for Conversion of Power Switchgear Equipment." *IEEE PC37.59/D14 , April 2018*, April 2018, 1–57. https://ieeexplore.ieee.org/document/8350345.
20. Iftikhar, Aleeha, Raymond R. Bond, Victoria McGilligan, et al. "Comparing Single-Page, Multipage, and Conversational Digital Forms in Health Care: Usability Study." *JMIR Human Factors* 8, no. 2 (2021): e25787. https://doi.org/10.2196/25787.
21. Jin, Hao, Yan Luo, Peilong Li, and Jomol Mathew. "A Review of Secure and Privacy-Preserving Medical Data Sharing." *IEEE Access* 7 (2019): 61656–69. https://doi.org/10.1109/ACCESS.2019.2916503.

22. Lam, Wai Yin, and Paula Fresco. “Medication Adherence Measures: An Overview.” *BioMed Research International* 2015 (2015): 1–12. https://doi.org/10.1155/2015/217047.

23. Lanke, Vaidehee, Kevin Trimm, Bettina Habib, and Robyn Tamblyn. “Evaluating the Effectiveness of Mobile Apps on Medication Adherence for Chronic Conditions: Systematic Review and Meta-Analysis.” *Journal of Medical Internet Research* 27, no. 1 (2025): e60822. https://doi.org/10.2196/60822.

24. Maheshwari, Mittal, Tulasee Lalitha Reddi, and Nainesh Kela. “Digital Health Platforms for Medication Management.” In *Technical Advances in Pharmacy: An Overview*. Bentham Science Publishers, 2025. https://www.benthamdirect.com/content/books/9798898812379.chapter-8.

25. Mamidi, Sasikanth. “Building Secure REST APIs in Spring Boot: Techniques and Tools.” *International Journal of Emerging Trends in Computer Science and Information Technology* 7, no. 1 (2026): 87–91. https://doi.org/10.63282/3050-9246.IJETCSIT-V7I1P111.

26. Mehmood, Nadeem Qaisar, Rosario Culmone, and Leonardo Mostarda. “Modeling Temporal Aspects of Sensor Data for MongoDB NoSQL Database.” *Journal of Big Data* 4, no. 1 (2017): 8. https://doi.org/10.1186/s40537-017-0068-5.

27. Meystre, Stéphane, and Peter J. Haug. “Natural Language Processing to Extract Medical Problems from Electronic Clinical Documents: Performance Evaluation.” *Journal of Biomedical Informatics* 39, no. 6 (2006): 589–99. https://doi.org/10.1016/j.jbi.2005.11.004.

28. Morales-Sandoval, Miguel, Ricardo De-La-Parra-Aguirre, Hiram Galeana-Zapién, and Alejandro Galaviz-Mosqueda. “A Three-Tier Approach for Lightweight Data Security of Body Area Networks in E-Health Applications.” *IEEE Access* 9 (2021): 146350–65. https://doi.org/10.1109/ACCESS.2021.3123456.

29. Mullins, Cheri. "Responsive, Mobile App, Mobile First: Untangling the UX Design Web in Practical Experience." *Proceedings of the 33rd Annual International Conference on the Design of Communication*, July 16, 2015, 1–6. https://doi.org/10.1145/2775441.2775478.

30. Nishat, Atika, and Junaid Muzaffar. "Securing Web Applications with OAuth 2.0, JWT, and Multi-Factor Authentication." *Euro Vantage Journals of Artificial Intelligence* 1, no. 2 (2024): 36–43. https://evjai.com/index.php/evjai/article/view/13.

31. Pérez-Jover, Virtudes, Marina Sala-González, Mercedes Guilabert, and José Joaquín Mira. "Mobile Apps for Increasing Treatment Adherence: Systematic Review." *Journal of Medical Internet Research* 21, no. 6 (2019): e12505. https://doi.org/10.2196/12505.

32. Redley, Bernice, and Mari Botti. "Reported Medication Errors after Introducing an Electronic Medication Management System." *Journal of Clinical Nursing* 22, nos. 3–4 (2013): 579–89. https://doi.org/10.1111/j.1365-2702.2012.04326.x.

33. Surisetty, Lok Santhoshkumar. "Improving Disease Detection Accuracy with Al and Secure Data Exchange through API Gateways." *International Journal of Advanced Research in Computer Science & Technology(IJARCST)* 7, no. 3 (2024): 10346–54. https://doi.org/10.15662/IJARCST.2024.0703008.

34. Talakola, Swetha, and Sai Prasad Veluru. "Managing Authentication in REST Assured OAuth, JWT and More." *International Journal of Emerging Trends in Computer Science and Information Technology* 4, no. 4 (2023): 66–75. https://doi.org/10.63282/3050-9246.IJETCSIT-V4I4P108.

35. Tang, Yiyi, Ziyan Xiao, Xue Li, et al. "Large Language Model in Medical Information Extraction from Titles and Abstracts with Prompt Engineering Strategies: A Comparative Study of GPT-3.5 and GPT-4." Preprint, medRxiv, February 14, 2025. https://doi.org/10.1101/2024.03.20.24304572.

36. Varshney, Upkar. “Smart Medication Management System and Multiple Interventions for Medication Adherence.” *Decision Support Systems*, 1. Analytics and Modeling for Better HealthCare 2. Decision Making in Healthcare, vol. 55, no. 2 (2013): 538–51. https://doi.org/10.1016/j.dss.2012.10.011.
37. Xiao, Jian, Mengyao Li, Ruwen Cai, et al. “Smart Pharmaceutical Monitoring System With Personalized Medication Schedules and Self-Management Programs for Patients With Diabetes: Development and Evaluation Study.” *Journal of Medical Internet Research* 27, no. 1 (2025): e56737. https://doi.org/10.2196/56737.
38. Wang, Yanshan, Liwei Wang, Ruiling Xie, and Feichen Shen. “Clinical Information Extraction Applications: A Literature Review.” *Journal of Biomedical Informatics* 77 (2018): 34–49. Elsevier. https://doi.org/10.1016/j.jbi.2017.11.011.
39. Alruban, Abdullah, Mohammad Al-Maitah, and Ahmad Alarood. “Artificial Intelligence Techniques in Healthcare Applications: A Review.” *Computers* 12, no. 4 (2023): 78. MDPI. https://doi.org/10.3390/computers12040078.
40. Almuhaideb, Abdullah, Sarah Alshehri, and Abdulrahman Alharbi. “MedScrubCrew: A Medical Multi-Agent Framework for Automating Appointment Scheduling Based on Patient-Provider Profile Resource Matching.” *Healthcare* 13, no. 14 (2025): 1649. MDPI. https://doi.org/10.3390/healthcare13141649.
41. Kyrimi, Evangelia, Mariana Raniere Neves, Scott McLachlan, Martin Neil, William Marsh, and Norman Fenton. “Medical Idioms for Clinical Bayesian Network Development.” *Artificial Intelligence in Medicine* 105 (2020): 101840. Elsevier. https://doi.org/10.1016/j.artmed.2020.101840.
42. Harby, Ahmed A., Eyad ElKhodary, Ronan Almeida, Drishti Sharma, Farhana Zulkernine, and Furkan Alaca. “Revolutionizing Healthcare Management: Architecture of a Web-Based Medical Triage Service.” In *2024 IEEE 48th Annual*

Computers, Software, and Applications Conference (COMPSAC), 2024. IEEE. https://doi.org/10.1109/COMPSAC61105.2024.00299.

43. Stack Overflow. "2024 Developer Survey: Most Popular Technologies." *Stack Overflow Developer Survey 2024*. Accessed May 2026. https://survey.stackoverflow.co/2024/technology.
44. React Team. "React 18: New Features." *React Blog*, March 29, 2022. https://react.dev/blog/2022/03/29/react-v18.
45. Remix Software. "React Router Documentation." *React Router Documentation*. Accessed May 2026. https://reactrouter.com/.
46. Pmndrs. "Zustand: Bear Necessities for State Management." *GitHub Repository*. Accessed May 2026. https://github.com/pmndrs/zustand.
47. TanStack. "React Query: Why Use TanStack Query?" *TanStack Query Documentation*. Accessed May 2026. https://tanstack.com/query/latest/docs/framework/react/overview.
48. Axios Documentation. "Interceptors." *Axios Documentation*. Accessed May 2026. https://axios-http.com/docs/interceptors.
49. State of CSS. "CSS Frameworks Rankings." *State of CSS 2023*. Accessed May 2026. https://stateofcss.com/en-US/.
50. Vite Team. "Why Vite?" *Vite Documentation*. Accessed May 2026. https://vitejs.dev/guide/why.html.
51. NPM Trends. "Express Weekly Downloads." *NPM Trends*. Accessed May 2026. https://npmtrends.com/express.
52. Node.js Foundation. "The Node.js Event Loop." *Node.js Documentation*. Accessed May 2026. https://nodejs.org/en/docs/guides/event-loop-timers-and-nexttick.
53. OpenAI. "GPT-4o Mini: Advancing Cost-Efficient Intelligence." *OpenAI Blog*, July 18, 2024. https://openai.com/index/gpt-4o-mini-advancing-cost-efficient-intelligence/.

54. OpenAI API. “Structured Outputs and JSON Mode.” *OpenAI Platform Documentation*. Accessed May 2026. https://platform.openai.com/docs/guides/structured-outputs.

ДОДАТОК А

Лістинг програмного коду

```
import dotenv from 'dotenv';
dotenv.config();
import express from 'express';
import cors from 'cors';
import helmet from 'helmet';
import rateLimit from 'express-rate-limit';
import connectDB from './config/db';
import errorHandler from './middleware/errorHandler';
import authRoutes from './routes/auth.routes';
import prescriptionRoutes from './routes/prescription.routes';
import scheduleRoutes from './routes/calendar.routes';
const app = express();
const PORT = process.env.PORT || 5001;
app.use(helmet());
app.use(cors());
app.use(express.json());
const limiter = rateLimit({
  windowMs: 15 * 60 * 1000, // 15 minutes
  max: 100,
  message: { status: 'error', message: 'Too many requests, please try again later.' },
});
app.use(limiter);
// Routes
app.use('/api/auth', authRoutes);
```

```
app.use('/api/prescriptions', prescriptionRoutes);

app.use('/api/schedule', scheduleRoutes);

// Error handling middleware

app.use(errorHandler);

// Connect to MongoDB and start server

connectDB().then(() => {

  app.listen(PORT, () => {

    console.log(`Server running on port ${PORT}`);

  });

});

export default app;

import OpenAI from 'openai';

import { z } from 'zod';

import { AppError } from '../middleware/errorHandler';

const parsedPrescriptionSchema = z.object({

  medication: z.string().nullable(),

  dosage: z.string().nullable(),

  frequency: z.number().nullable(),

  frequencyUnit: z.string().nullable(),

  duration: z.number().nullable(),

  durationUnit: z.string().nullable(),

  notes: z.string().nullable(),

});

export type ParsedPrescription = z.infer<typeof parsedPrescriptionSchema>;

const SYSTEM_PROMPT = `You are a medical prescription parser. Extract structured data from free-text doctor recommendations. Return ONLY valid JSON with these fields:

- medication (string): name of the drug/medication

- dosage (string): amount per dose, e.g. "1 tablet", "500mg"
```

```
- frequency (number): how many times per period
- frequencyUnit (string): the period — "day", "week", etc.
- duration (number): how long the treatment lasts
- durationUnit (string): unit of duration — "days", "weeks", "months"
- notes (string): any additional instructions like "after meals", "with water"
If a field cannot be determined from the text, set it to null.`;
const openai = new OpenAI({ apiKey: process.env.OPENAI_API_KEY });
export async function parsePrescription(rawText: string): Promise<ParsedPrescription> {
  const response = await openai.chat.completions.create({
    model: 'gpt-4o-mini',
    response_format: { type: 'json_object' },
    messages: [
      { role: 'system', content: SYSTEM_PROMPT },
      { role: 'user', content: rawText },
    ],
    temperature: 0,
  });
  const content = response.choices[0]?.message?.content;
  if (!content) {
    throw new AppError('Empty response from AI service', 502);
  }
  let json: unknown;
  try {
    json = JSON.parse(content);
  } catch {
    throw new AppError('AI returned invalid JSON', 502);
  }
  const result = parsedPrescriptionSchema.safeParse(json);
```

```
  if (!result.success) {
    throw new AppError(`AI response schema mismatch: ${result.error.message}`, 502);
  }
  return result.data;
}

import { Types } from 'mongoose';
import type { ParsedPrescription } from './aiParser';
interface ScheduleEntryData {
  userId: Types.ObjectId;
  prescriptionId: Types.ObjectId;
  medication: string;
  dosage: string | null;
  scheduledDate: Date;
  timeSlot: 'morning' | 'afternoon' | 'evening' | 'night';
  taken: boolean;
  takenAt: null;
}
const TIME_SLOT_MAP: Record<number, ('morning' | 'afternoon' | 'evening' | 'night')[]> = {
  1: ['morning'],
  2: ['morning', 'evening'],
  3: ['morning', 'afternoon', 'evening'],
  4: ['morning', 'afternoon', 'evening', 'night'],
};
function getDurationInDays(duration: number, unit: string): number {
  switch (unit.toLowerCase().replace(/s$/, '')) {
    case 'day': return duration;
    case 'week': return duration * 7;
```

```
    case 'month': return duration * 30;
    default: return duration;
  }
}
export function generateSchedule(
  parsed: ParsedPrescription,
  startDate: Date,
  userId: Types.ObjectId,
  prescriptionId: Types.ObjectId
): ScheduleEntryData[] {
  const frequency = parsed.frequency ?? 1;
  const durationDays = getDurationInDays(
    parsed.duration ?? 1,
    parsed.durationUnit ?? 'days'
  );
  const medication = parsed.medication ?? 'Unknown';
  const dosage = parsed.dosage ?? null;
  const slots = TIME_SLOT_MAP[Math.min(frequency, 4)] ?? TIME_SLOT_MAP[4]!;
  const entries: ScheduleEntryData[] = [];
  for (let day = 0; day < durationDays; day++) {
    const date = new Date(startDate);
    date.setDate(date.getDate() + day);
    date.setHours(0, 0, 0, 0);

    for (const slot of slots) {
      entries.push({
        userId,
        prescriptionId,
```

```
        medication,
        dosage,
        scheduledDate: date,
        timeSlot: slot,
        taken: false,
        takenAt: null,
      });
    }
  }
  return entries;
}

import mongoose, { Schema, Document, Types } from 'mongoose';
export interface IParsedPrescription {
  medication: string | null;
  dosage: string | null;
  frequency: number | null;
  frequencyUnit: string | null;
  duration: number | null;
  durationUnit: string | null;
  startDate: Date | null;
  notes: string | null;
}
export interface IPrescription extends Document {
  userId: Types.ObjectId;
  rawText: string;
  parsed: IParsedPrescription;
  status: 'active' | 'completed' | 'cancelled';
```

```
  createdAt: Date;
}
const prescriptionSchema = new Schema<IPrescription>({
  userId: {
    type: Schema.Types.ObjectId,
    ref: 'User',
    required: true,
  },
  rawText: {
    type: String,
    required: true,
  },
  parsed: {
    medication: { type: String, default: null },
    dosage: { type: String, default: null },
    frequency: { type: Number, default: null },
    frequencyUnit: { type: String, default: null },
    duration: { type: Number, default: null },
    durationUnit: { type: String, default: null },
    startDate: { type: Date, default: null },
    notes: { type: String, default: null },
  },
  status: {
    type: String,
    enum: ['active', 'completed', 'cancelled'],
    default: 'active',
  },
  createdAt: {
```

```
    type: Date,
    default: Date.now,
  },
});
export default mongoose.model<IPrescription>('Prescription', prescriptionSchema);


import mongoose, { Schema, Document, Types } from 'mongoose';
export interface IScheduleEntry extends Document {
  userId: Types.ObjectId;
  prescriptionId: Types.ObjectId;
  medication: string;
  dosage: string | null;
  scheduledDate: Date;
  timeSlot: 'morning' | 'afternoon' | 'evening' | 'night';
  taken: boolean;
  takenAt: Date | null;
}
const scheduleEntrySchema = new Schema<IScheduleEntry>({
  userId: {
    type: Schema.Types.ObjectId,
    ref: 'User',
    required: true,
  },
  prescriptionId: {
    type: Schema.Types.ObjectId,
    ref: 'Prescription',
    required: true,
  },
```

```
  medication: {
    type: String,
    required: true,
  },
  dosage: {
    type: String,
    default: null,
  },
  scheduledDate: {
    type: Date,
    required: true,
  },
  timeSlot: {
    type: String,
    enum: ['morning', 'afternoon', 'evening', 'night'],
    required: true,
  },
  taken: {
    type: Boolean,
    default: false,
  },
  takenAt: {
    type: Date,
    default: null,
  },
});

scheduleEntrySchema.index({ userId: 1, scheduledDate: 1 });
```

```
export default mongoose.model<IScheduleEntry>('ScheduleEntry', scheduleEntrySchema);


import mongoose, { Schema, Document } from 'mongoose';
export interface IUser extends Document {
  email: string;
  passwordHash: string;
  name: string;
  createdAt: Date;
}
const userSchema = new Schema<IUser>({
  email: {
    type: String,
    required: true,
    unique: true,
    lowercase: true,
    trim: true,
  },
  passwordHash: {
    type: String,
    required: true,
  },
  name: {
    type: String,
    required: true,
    trim: true,
  },
  createdAt: {
    type: Date,
```

```
    default: Date.now,
  },
});
export default mongoose.model<IUser>('User', userSchema);

import { Router, Request, Response, NextFunction } from 'express';
import bcrypt from 'bcryptjs';
import jwt from 'jsonwebtoken';
import User from '../models/User';
import { AppError } from '../middleware/errorHandler';
const router = Router();
router.post('/register', async (req: Request, res: Response, next: NextFunction) => {
  try {
    const { email, password, name } = req.body;
    if (!email || !password || !name) {
      throw new AppError('Email, password, and name are required', 400);
    }
    const existing = await User.findOne({ email });
    if (existing) {
      throw new AppError('Email already registered', 409);
    }
    const passwordHash = await bcrypt.hash(password, 12);
    const user = await User.create({ email, passwordHash, name });
    const token = jwt.sign({ userId: user._id }, process.env.JWT_SECRET!, {
      expiresIn: '7d',
    });
    res.status(201).json({
      token,
```

```
      user: { id: user._id, email: user.email, name: user.name },
    });
  } catch (err) {
    next(err);
  }
});
router.post('/login', async (req: Request, res: Response, next: NextFunction) => {
  try {
    const { email, password } = req.body;
    if (!email || !password) {
      throw new AppError('Email and password are required', 400);
    }
    const user = await User.findOne({ email });
    if (!user) {
      throw new AppError('Invalid credentials', 401);
    }
    const isMatch = await bcrypt.compare(password, user.passwordHash);
    if (!isMatch) {
      throw new AppError('Invalid credentials', 401);
    }
    const token = jwt.sign({ userId: user._id }, process.env.JWT_SECRET!, {
      expiresIn: '7d',
    });
    res.json({
      token,
      user: { id: user._id, email: user.email, name: user.name },
    });
  } catch (err) {
```

```
    next(err);
  }
});
export default router;

import { Router, Response, NextFunction } from 'express';
import { auth, AuthRequest } from '../middleware/auth';
import { AppError } from '../middleware/errorHandler';
import ScheduleEntry from '../models/ScheduleEntry';
import Prescription from '../models/Prescription';
const router = Router();
// GET /api/schedule?date=YYYY-MM-DD
router.get('/', auth, async (req: AuthRequest, res: Response, next: NextFunction) => {
  try {
    const { date } = req.query;
    if (!date || typeof date !== 'string') {
      throw new AppError('date query parameter is required (YYYY-MM-DD)', 400);
    }

    const day = new Date(date);
    day.setHours(0, 0, 0, 0);
    const nextDay = new Date(day);
    nextDay.setDate(nextDay.getDate() + 1);
    const entries = await ScheduleEntry.find({
      userId: req.userId,
      scheduledDate: { $gte: day, $lt: nextDay },
    }).sort({ timeSlot: 1 });
    res.json(entries);
```

```
  } catch (err) {
    next(err);
  }
});
// GET /api/schedule/range?from=YYYY-MM-DD&to=YYYY-MM-DD
router.get('/range', auth, async (req: AuthRequest, res: Response, next: NextFunction) => {
  try {
    const { from, to } = req.query;
    if (!from || !to || typeof from !== 'string' || typeof to !== 'string') {
      throw new AppError('from and to query parameters are required (YYYY-MM-DD)', 400);
    }
    const start = new Date(from);
    start.setHours(0, 0, 0, 0);
    const end = new Date(to);
    end.setHours(23, 59, 59, 999);
    const entries = await ScheduleEntry.find({
      userId: req.userId,
      scheduledDate: { $gte: start, $lte: end },
    }).sort({ scheduledDate: 1, timeSlot: 1 });
    res.json(entries);
  } catch (err) {
    next(err);
  }
});
// PATCH /api/schedule/:entryId/toggle
router.patch('/:entryId/toggle', auth, async (req: AuthRequest, res: Response, next: NextFunction) => {
  try {
    const entry = await ScheduleEntry.findOne({
```

```
    _id: req.params.entryId,
    userId: req.userId,
  });
  if (!entry) {
    throw new AppError('Schedule entry not found', 404);
  }
  const today = new Date();
  today.setHours(23, 59, 59, 999);
  if (entry.scheduledDate > today) {
    throw new AppError('Cannot mark future doses as taken', 400);
  }
  entry.taken = !entry.taken;
  entry.takenAt = entry.taken ? new Date() : null;
  await entry.save();
  // Auto-complete prescription when all its doses are taken
  if (entry.prescriptionId) {
    const totalEntries = await ScheduleEntry.countDocuments({ prescriptionId: entry.prescriptionId });
    const takenEntries = await ScheduleEntry.countDocuments({ prescriptionId: entry.prescriptionId, taken: true
});
    if (takenEntries === totalEntries) {
      await Prescription.findByIdAndUpdate(entry.prescriptionId, { status: 'completed' });
    } else {
      // If untoggling a dose, reactivate a completed prescription
      await Prescription.findOneAndUpdate(
        { _id: entry.prescriptionId, status: 'completed' },
        { status: 'active' }
      );
    }
```

```
    }
    res.json(entry);
  } catch (err) {
    next(err);
  }
});
export default router;

import { Router, Response, NextFunction } from 'express';
import { Types } from 'mongoose';
import { auth, AuthRequest } from '../middleware/auth';
import { AppError } from '../middleware/errorHandler';
import { parsePrescription, ParsedPrescription } from '../services/aiParser';
import { generateSchedule } from '../services/scheduleGenerator';
import Prescription from '../models/Prescription';
import ScheduleEntry from '../models/ScheduleEntry';
const router = Router();
function normalizeParsedPrescription(input: unknown): ParsedPrescription {
  if (!input || typeof input !== 'object') {
    throw new AppError('parsed prescription is required', 400);
  }
  const parsed = input as Record<string, unknown>;
  const toNullableString = (value: unknown) => {
    if (value === null || value === undefined || value === '') return null;
    return String(value);
  };
  const toNullableNumber = (value: unknown) => {
    if (value === null || value === undefined || value === '') return null;
```

```
    const num = Number(value);
    if (Number.isNaN(num)) {
      throw new AppError('Invalid numeric prescription field', 400);
    }
    return num;
  };
  return {
    medication: toNullableString(parsed.medication),
    dosage: toNullableString(parsed.dosage),
    frequency: toNullableNumber(parsed.frequency),
    frequencyUnit: toNullableString(parsed.frequencyUnit),
    duration: toNullableNumber(parsed.duration),
    durationUnit: toNullableString(parsed.durationUnit),
    notes: toNullableString(parsed.notes),
  };
}
// POST /api/prescriptions/parse — parse text only, do not save yet
router.post('/parse', auth, async (req: AuthRequest, res: Response, next: NextFunction) => {
  try {
    const { rawText, startDate } = req.body;
    if (!rawText || typeof rawText !== 'string') {
      throw new AppError('rawText is required', 400);
    }
    const parsed = await parsePrescription(rawText);
    const effectiveStart = startDate ? new Date(startDate) : new Date();
    res.json({
      parsed: {
        ...parsed,
```

```
      startDate: effectiveStart,
    },
   });
  } catch (err) {
   next(err);
  }
});
router.post('/', auth, async (req: AuthRequest, res: Response, next: NextFunction) => {
  try {
   const { rawText, startDate, parsed: parsedInput } = req.body;
   if (!rawText || typeof rawText !== 'string') {
    throw new AppError('rawText is required', 400);
   }
   const parsed = parsedInput
    ? normalizeParsedPrescription(parsedInput)
    : await parsePrescription(rawText);
   const prescription = await Prescription.create({
    userId: req.userId,
    rawText,
    parsed: {
     ...parsed,
     startDate: startDate ? new Date(startDate) : new Date(),
    },
    status: 'active',
   });
   const effectiveStart = startDate ? new Date(startDate) : new Date();
   const entries = generateSchedule(
    parsed,
```

```
      effectiveStart,
      new Types.ObjectId(req.userId),
      prescription._id as Types.ObjectId
    );
    if (entries.length > 0) {
      await ScheduleEntry.insertMany(entries);
    }
    res.status(201).json({
      prescription,
      scheduledDoses: entries.length,
    });
  } catch (err) {
    next(err);
  }
});
// GET /api/prescriptions — list all for current user
router.get('/', auth, async (req: AuthRequest, res: Response, next: NextFunction) => {
  try {
    const prescriptions = await Prescription.find({ userId: req.userId })
      .sort({ createdAt: -1 });
    res.json(prescriptions);
  } catch (err) {
    next(err);
  }
});
// DELETE /api/prescriptions/:id — cancel prescription + remove future entries
router.delete('/:id', auth, async (req: AuthRequest, res: Response, next: NextFunction) => {
  try {
```

```
    const prescription = await Prescription.findOne({
      _id: req.params.id,
      userId: req.userId,
    });
    if (!prescription) {
      throw new AppError('Prescription not found', 404);
    }
    prescription.status = 'cancelled';
    await prescription.save();
    const today = new Date();
    today.setHours(0, 0, 0, 0);
    await ScheduleEntry.deleteMany({
      prescriptionId: prescription._id,
      userId: req.userId,
      scheduledDate: { $gte: today },
      taken: false,
    });
    res.json({ message: 'Prescription cancelled', prescription });
  } catch (err) {
    next(err);
  }
});
export default router;

import { Request, Response, NextFunction } from 'express';
import jwt from 'jsonwebtoken';
import { AppError } from './errorHandler';
export interface AuthRequest extends Request {
```

```
  userId?: string;
}
export const auth = (req: AuthRequest, _res: Response, next: NextFunction) => {
  const header = req.headers.authorization;
  if (!header || !header.startsWith('Bearer ')) {
    return next(new AppError('No token provided', 401));
  }
  const token = header.split(' ')[1];
  try {
    const decoded = jwt.verify(token, process.env.JWT_SECRET!) as { userId: string };
    req.userId = decoded.userId;
    next();
  } catch {
    next(new AppError('Invalid token', 401));
  }
};

import { Request, Response, NextFunction } from 'express';
export class AppError extends Error {
  statusCode: number;
  isOperational: boolean;
  constructor(message: string, statusCode: number) {
    super(message);
    this.statusCode = statusCode;
    this.isOperational = true;
    Object.setPrototypeOf(this, AppError.prototype);
  }
}
```

```
interface MongooseValidationError extends Error {
  errors: Record<string, { message: string }>;
}
interface MongooseDuplicateKeyError extends Error {
  code: number;
  keyValue: Record<string, unknown>;
}
const errorHandler = (
  err: Error,
  _req: Request,
  res: Response,
  _next: NextFunction
): void => {
  let statusCode = 500;
  let message = 'Internal Server Error';
  // Handle AppError
  if (err instanceof AppError) {
    statusCode = err.statusCode;
    message = err.message;
  }
  // Handle Mongoose validation errors
  if (err.name === 'ValidationError') {
    statusCode = 400;
    const validationError = err as MongooseValidationError;
    const messages = Object.values(validationError.errors).map((e) => e.message);
    message = `Validation Error: ${messages.join(', ')}`;
  }
  // Handle Mongoose duplicate key errors
```

```
  if ((err as MongooseDuplicateKeyError).code === 11000) {
    statusCode = 409;
    const dupError = err as MongooseDuplicateKeyError;
    const field = Object.keys(dupError.keyValue).join(', ');
    message = `Duplicate value for field: ${field}`;
  }
  const response: Record<string, unknown> = {
    status: 'error',
    message,
  };
  if (process.env.NODE_ENV === 'development') {
    response.stack = err.stack;
  }
  res.status(statusCode).json(response);
};
export default errorHandler;
```

ДОДАТОК Б

Структурні файли та директорії проекту

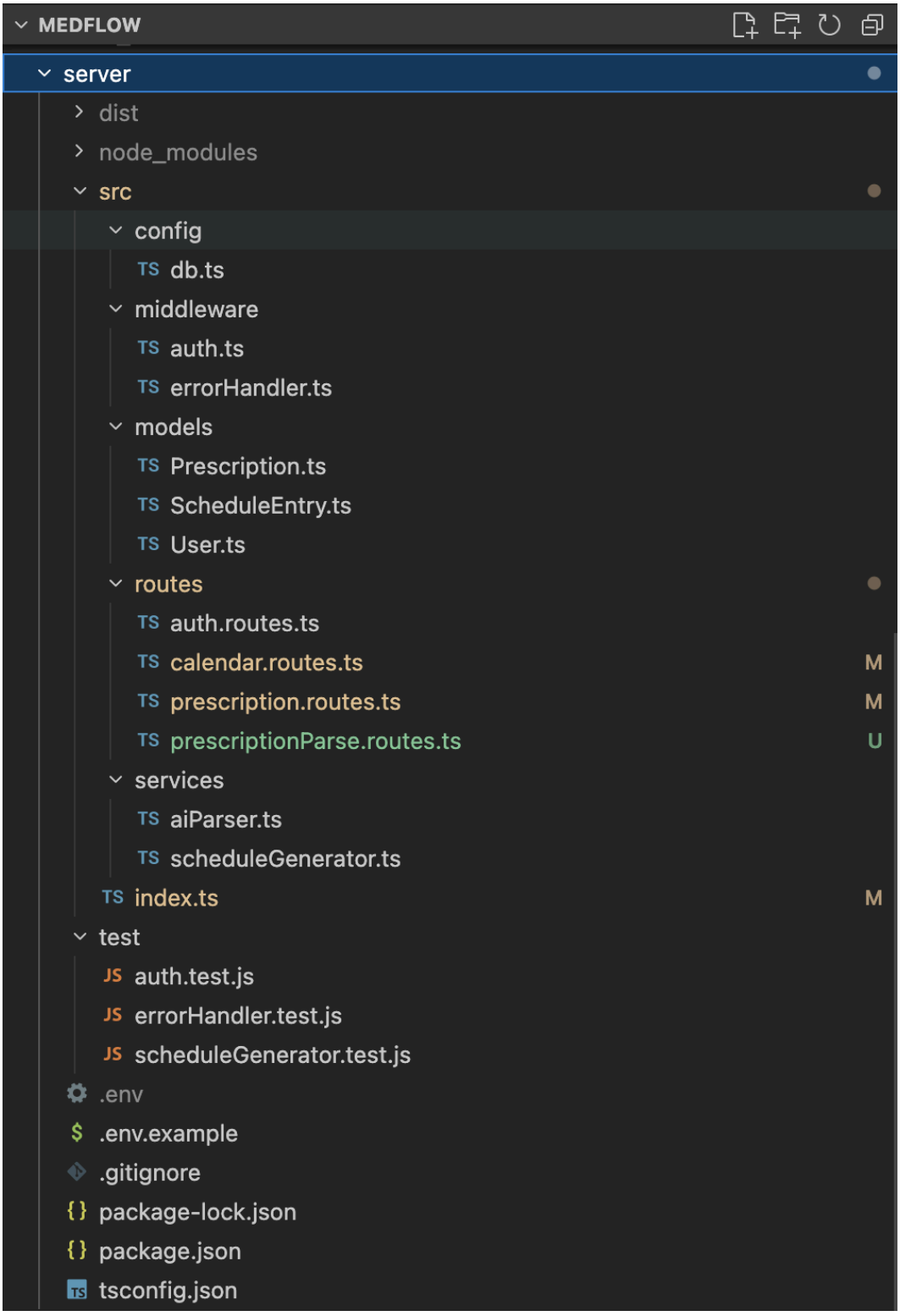

Рисунок Б.1 – Файли та директорії серверної частини проекту

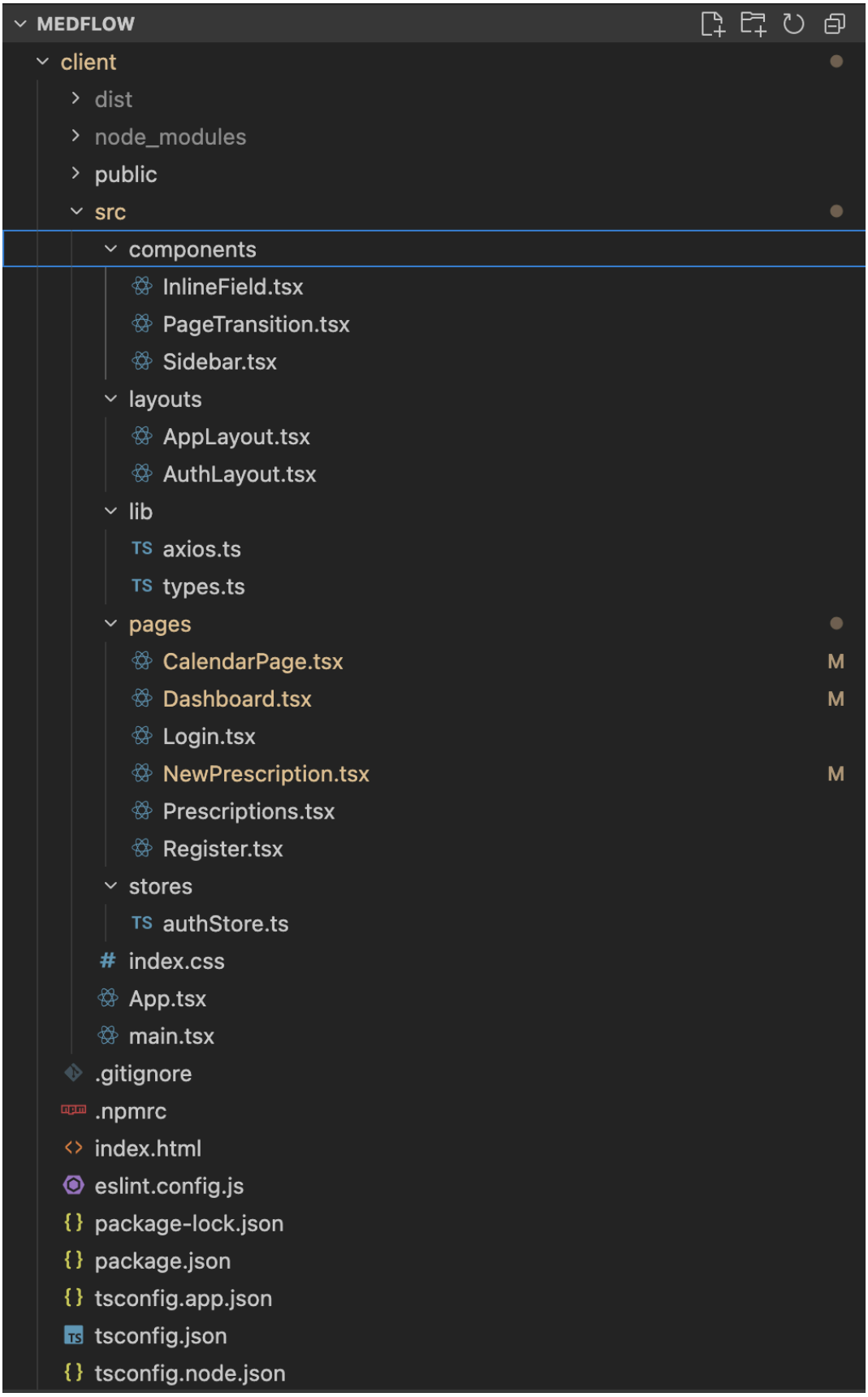

Рисунок Б.2 – Файли та директорії клієнтської частини проекту