ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ
НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ
ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ
КАФЕДРА ТЕХНОЛОГІЙ ЦИФРОВОГО РОЗВИТКУ

КВАЛІФІКАЦІЙНА РОБОТА

на тему: «Web-застосунок для обліку прийому ліків із календарем нагадувань та інтелектуальним модулем контролю з використанням ASP.NET»

на здобуття освітнього ступеня бакалавра
зі спеціальності 121 Інженерія програмного забезпечення
освітньо-професійної програми «Технології цифрового розвитку»

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

_____ Анастасія АРТЕМЕНКО

Виконав: здобувачка вищої освіти групи ТЦР-41

Анастасія АРТЕМЕНКО

Керівник: Ждан ЧЕРНЯВСЬКИЙ
ст.викл.

Рецензент:

Київ 2026

ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ

Навчально-науковий інститут інформаційних технологій

Кафедра Технологій цифрового розвитку

Ступінь вищої освіти Бакалавр

Спеціальність 121 Інженерія програмного забезпечення

Освітньо-професійна програма «Технології цифрового розвитку»

ЗАТВЕРДЖУЮ
Завідувач кафедри
Технологій цифрового розвитку
_____ Вікторія ЖЕБКА
« ____ » _____ 2026 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

Артеменко Анастасії Ярославівні

1. Тема кваліфікаційної роботи: «Web-застосунок для обліку прийому ліків із календарем нагадувань та інтелектуальним модулем контролю з використанням ASP.NET»
керівник кваліфікаційної роботи ст.викл. Ждан ЧЕРНЯВСЬКИЙ,
затверджені наказом Державного університету інформаційно-комунікаційних технологій від «20» лютого 2026 року № 51.

2. Строк подання кваліфікаційної роботи «18» травня 2026 року.

3. Вихідні дані до кваліфікаційної роботи: теоретичні відомості про архітектуру веб-застосунків на базі фреймворку ASP.NET Core, принципи побудови реляційних баз даних (MS SQL Server), методи розрахунку медичної комплаєнтності (дисципліни пацієнта), технічна документація з опису API інтелектуального модуля обробки природної мови (NLP) та бібліотек візуалізації даних (FullCalendar, Chart.js).

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)

1. Аналіз предметної галузі та існуючих рішень для моніторингу прийому ліків.

2. Проєктування архітектури та бази даних веб-застосунку «MedicineTracker».
3. Програмна реалізація та опис функціонування веб-застосунку.
4. Тестування застосунку.

5. Перелік графічного матеріалу: *презентація*
1. Мета, об’єкт та предмет дослідження.
2. Задачі кваліфікаційної роботи.
3. Аналіз аналогів.
4. Вимоги до веб-застосунку.
5. Програмні засоби реалізації.
6. Діаграма варіантів використання платформи.
7. Блок-схема алгоритму роботи застосунку.
8. Діаграма класів.
9. Екранні форми.
10. Апробація результатів дослідження.

6. Дата видачі завдання «20» лютого 2026 року.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1	Підбір та аналіз науково-технічної літератури	21.02.2026-27.02.2026	Виконано
2	Аналіз та дослідження існуючих аналогів	27.02.2026-06.03.2026	Виконано
3	Огляд існуючих алгоритмів оцінки контролю прийому ліків та методів і технологій побудови веб-застосунку	06.03.2026-13.03.2026	Виконано
4	Проєктування застосунку для автоматизованої оцінки прийому медикаментів з інтелектуальним модулем контролю	13.03.2026-27.03.2026	Виконано
5	Програмна реалізація застосунку	27.03.2026-17.04.2026	Виконано
6	Тестування застосунку	17.04.2026-01.05.2026	Виконано
7	Оформлення роботи: вступ, висновки, реферат	01.05.2026-07.05.2026	Виконано
8	Розробка демонстраційних матеріалів	07.05.2026-10.05.2026	Виконано
9	Попередній захист роботи	11.05.2026	Виконано

Здобувачка вищої освіти _____ Анастасія АРТЕМЕНКО

Керівник кваліфікаційної роботи _____ Ждан ЧЕРНЯВСЬКИЙ

РЕФЕРАТ

Текстова частина кваліфікаційної роботи на здобуття освітнього ступеня бакалавра: 67 стор., 18 табл., 18 рис., 19 джерел.

Мета роботи – підвищення дисципліни прийому лікарських засобів пацієнтами шляхом розробки інтелектуального веб-застосунку для автоматизації контролю та нагадувань.

Об'єкт дослідження – процес управління індивідуальним графіком лікування та інформаційної підтримки пацієнта в середовищі веб-орієнтованих систем.

Предмет дослідження – методи та програмні засоби розробки інтелектуального трекера прийому ліків на базі платформи ASP.NET Core.

Короткий зміст роботи: У роботі проаналізовано сучасні підходи до моніторингу медикаментозної терапії та методи підвищення комплаєнтності пацієнтів. Проведено порівняльний аналіз існуючих програмних рішень, таких як Medisafe та MyTherapy. Розроблено архітектуру системи та структуру реляційної бази даних для зберігання персональних медичних записів. Програмно реалізовано ключові функції: динамічне планування лікування, візуалізацію розкладу в інтерактивному календарі, систему миттєвих сповіщень та модуль інтелектуальної підтримки на основі технологій обробки природної мови (NLP). Проведено модульне, інтеграційне та юзабіліті-тестування застосунку. В роботі використано фреймворк ASP.NET Core 8.0 для серверної логіки, Entity Framework Core для взаємодії з базою даних SQL Server, бібліотеку FullCalendar для інтерфейсу планування та хмарну платформу Engati для реалізації інтелектуального асистента.

Сферою використання застосунку є персональний моніторинг стану здоров'я пацієнтами, що потребують тривалої медикаментозної терапії.

КЛЮЧОВІ СЛОВА: ВЕБ-ЗАСТОСУНОК, ASP.NET CORE, MEDICINETRACKER, КОМПЛАЄНТНІСТЬ, NLP ЧАТ-БОТ.

ЗМІСТ

ВСТУП

Актуальність теми. Сучасний темп життя та зростання кількості хронічних захворювань ставлять перед системою охорони здоров'я нові виклики. Згідно з даними ВООЗ, майже половина пацієнтів не дотримується встановленого режиму прийому ліків, що призводить до ускладнень та додаткових витрат на лікування. Цифровізація медичної підтримки через веб-технології є одним із найефективніших методів вирішення цієї проблеми.

Використання сучасних фреймворків, таких як ASP.NET Core, дозволяє створювати доступні та надійні інструменти для персонального моніторингу здоров'я. Проте більшість існуючих рішень орієнтовані виключно на мобільні платформи або мають перевантажений інтерфейс. Створення інтелектуального веб-застосунку, який поєднує в собі зручний календарний інтерфейс та допоміжні модулі обробки запитів (NLP), є актуальним завданням, що дозволить підвищити дисципліну лікування та автоматизувати облік медикаментозної терапії.

Об’єкт дослідження – процес управління індивідуальним графіком лікування та інформаційної підтримки пацієнта в середовищі веб-орієнтованих систем..

Предмет дослідження – методи, підходи та технології розробки інтелектуальних систем моніторингу прийому ліків.

Мета роботи – проєктування та розробка інтелектуального веб-застосунку «MedicineTracker» для автоматизації обліку прийому ліків та надання персоналізованої підтримки користувачу.

Для досягнення мети, у бакалаврській роботі успішно виконано наступні завдання:

- аналіз предметної галузі та існуючих програмних рішень для контролю медикаментозної терапії;
- обґрунтовано вибір технологічного стеку;
- спроєктовано архітектуру бази даних та логічну структуру веб-застосунку;

- реалізовано функціональні модулі управління препаратами, планування розкладу та інтерактивного календаря;

- інтегровано інтелектуальний модуль підтримки на основі технологій обробки природної мови;

- проведено тестування системи та оцінку її функціональної придатності..

Методи дослідження. Під час написання роботи були використані методи об'єктно-орієнтованого програмування, архітектурне моделювання програмних систем, методи реляційного моделювання баз даних та алгоритми аналізу даних для розрахунку комплаєнтності пацієнта.

Наукова новизна одержаних результатів. У ході дослідження запропоновано підхід до побудови системи медичного трекінгу, що поєднує класичне календарне планування з асинхронним обміном даними (SignalR) та допоміжним інтелектуальним асистентом (Engati NLP). Описано алгоритм розрахунку статусів дисципліни пацієнта на основі фактичного виконання графіка.

Практична значущість одержаних результатів. Розроблений веб-застосунок забезпечує ефективне та безкоштовне рішення для пацієнтів, що потребують тривалого лікування. Система дозволяє централізовано зберігати історію терапії та отримувати миттєві нагадування, що сприяє зменшенню кількості пропущених доз ліків.

Структура дипломної роботи складається зі вступу, чотирьох розділів, висновків та списку використаних джерел. У першому розділі проведено системний аналіз предметної області, досліджено проблему медичної комплаєнтності та виконано порівняльний огляд існуючих програмних рішень для контролю прийому ліків, що дозволило обґрунтувати необхідність розробки власного застосунку. У другому розділі визначено функціональні та нефункціональні вимоги до системи, обґрунтовано вибір технологічного стеку на базі платформи .NET та архітектурного шаблону MVC, а також спроєктовано логічну й фізичну структуру реляційної бази даних. У третьому розділі описано програмну реалізацію веб-застосунку «MedicineTracker», зокрема розробку інтерактивного календаря, системи фонових нагадувань та інтеграцію модуля

інтелектуальної підтримки на основі NLP-технологій. У четвертому розділі висвітлено результати комплексного тестування системи, проведено статистичний аналіз проходження тестів та оцінено зручність інтерфейсу користувача. У висновках узагальнено результати розробки, підтверджено досягнення поставленої мети та визначено перспективи подальшого вдосконалення системи в напрямку інтеграції з медичними API.

Апробація результатів та публікації. Ключові аспекти, концептуальні рішення та отримані результати кваліфікаційної роботи були представлені й обговорені в межах науково-практичних заходів (конференцій), організованих Державним університетом інформаційно-комунікаційних технологій:

1. Артеменко А. Я., Чернявський Ж. А. Аналіз сучасних підходів до створення веб-систем контролю прийому медикаментів // VII Всеукраїнська науково-технічної конференція «Застосування програмного забезпечення в інформаційно-комунікаційних технологіях». Збірник тез. Київ: ДУІКТ, 2026, С. 53-55.
2. Артеменко А. Я., Чернявський Ж. А. Реалізація механізму контролю та аналізу прийому ліків у веб-застосунку на базі ASP.NET Core MVC // VII Всеукраїнська науково-технічна конференція «Технології цифрового розвитку: програмні системи та децентралізація». Збірник тез. Київ: ДУІКТ, 2026.

1 АНАЛІЗ ПРЕДМЕТНОЇ ГАЛУЗІ ТА ІСНУЮЧИХ РІШЕНЬ

1.1 Характеристика предметної галузі

Предметна галузь дослідження охоплює цифровізацію процесів персонального медичного менеджменту та автоматизацію контролю за виконанням лікувальних призначень [15]. У сучасній медичній практиці цей напрям визначається через концепцію комплаєнтності [12], що описує ступінь відповідності дій пацієнта рекомендаціям фахівця. На відміну від традиційних методів фіксації, сучасне інформаційне середовище вимагає створення інтегрованих систем, де дані про медикаментозну терапію трансформуються у динамічні цифрові об'єкти з власним життєвим циклом [6].

Облік прийому ліків передбачає систематичну фіксацію інформації про призначені препарати, дозування, час та частоту їх прийому, а також відстеження фактичного виконання цих призначень пацієнтом. Важливим аспектом предметної галузі є не лише пасивне збереження даних, а й активне відстеження фактичного виконання призначень пацієнтом. Предметна галузь охоплює такі поняття, як медичні препарати (лікарські засоби), розклади прийому, інтерактивні нагадування та допоміжні механізми контролю за дотриманням встановленого режиму.

Сучасні хронічні захворювання – цукровий діабет, артеріальна гіпертензія, серцево-судинні хвороби, психіатричні розлади – потребують прийому кількох препаратів одночасно за суворо визначеним розкладом. Складність полягає в тому, що пацієнт часто приймає від 5 до 10 різних медикаментів на добу з відповідними часовими інтервалами, обмеженнями щодо їжі та взаємодією між препаратами.

Актуальність системи обліку прийому ліків обумовлена такими факторами:

– зростання кількості пацієнтів з хронічними захворюваннями, що потребують постійної медикаментозної підтримки;
– недостатня поінформованість пацієнтів про важливість дотримання режиму терапії;

- відсутність централізованих цифрових інструментів для персонального моніторингу прийому ліків;
- необхідність зберігання медичної інформації та надання зручного доступу до неї через веб-інтерфейс;
- потреба в інтелектуальному аналізі даних для виявлення порушень режиму та прогнозування ризиків.

Веб-застосунок MedicineTracker, розроблений на базі платформи ASP.NET Core 8.0, являє собою сучасну інформаційну систему, що надає пацієнту або його представнику ергономічний інструмент для управління лікувальним процесом. Функціональна наповненість системи передбачає не лише базову реєстрацію препаратів та налаштування календарних розкладів, а й впровадження інтелектуального модуля підтримки. Використання технологій обробки природної мови (NLP) [10] у формі допоміжного чат-бота дозволяє спростити взаємодію користувача із системою, забезпечуючи швидкий пошук інформації та допомогу в навігації через текстовий інтерфейс.

Серед основних учасників предметної галузі слід виділити: пацієнта, що є кінцевим споживачем системи; лікаря, що призначає курс лікування; адміністратора системи, що забезпечує коректне функціонування застосунку. Архітектурно система базується на шаблоні MVC (Model-View-Controller), що дозволяє відокремити бізнес-логіку від представлення даних. Основними об'єктами обліку виступають профілі препаратів (назва, опис, специфічне дозування), курси лікування з чіткими часовими межами, розклади, побудовані на основі часових слотів протягом доби, та записи про фактичне вживання ліків. Вся сукупність цих об'єктів формує структуру реляційної бази даних під управлінням Microsoft SQL Server [16], що гарантує цілісність та доступність інформації в режимі реального часу.

Таким чином, предметна галузь роботи охоплює повний цикл управління інформацією про медикаментозну терапію – від проєктування архітектури бази даних до реалізації інтелектуальних засобів аналізу дисципліни пацієнта з метою підвищення загальної ефективності лікувального процесу.

1.2 Огляд існуючих програмних продуктів

На сьогоднішній день ринок мобільних та веб-застосунків пропонує низку рішень для обліку прийому ліків. Розглянемо найбільш відомі з них з аналізом їх функціональних можливостей та недоліків.

1.2.1 Medisafe

Medisafe – один із найпопулярніших мобільних застосунків для управління прийомом ліків, розроблений компанією Medisafe Inc. (США). Застосунок надає функції нагадування про прийом препаратів, відстеження пропущених доз, перевірки взаємодії між ліками та формування звітів для лікаря. Medisafe підтримує платформи iOS та Android, має хмарну синхронізацію та функцію «MedFriend» для сповіщення родичів у разі пропуску прийому ліків.

До основних недоліків Medisafe можна віднести: відсутність повноцінного веб-інтерфейсу (основний акцент зроблено на мобільний додаток); обмежені можливості для корпоративного або клінічного використання; частина функцій доступна лише в платній підписці (Premium); відсутність відкритого API для інтеграції зі сторонніми інформаційними медичними системами; збереження персональних медичних даних на зарубіжних серверах, що ускладнює дотримання вимог GDPR та українського законодавства.

Рис. 1.1 Приклади екранних форм застосунку Medisafe

1.2.2 MyTherapy

MyTherapy – мобільний застосунок від компанії smartpatient GmbH (Німеччина), орієнтований на пацієнтів з хронічними захворюваннями. Крім базової функції нагадувань про ліки, застосунок дозволяє відстежувати симптоми, фіксувати показники стану здоров'я (артеріальний тиск, рівень цукру) та формувати звіти для лікаря у форматі PDF.

До недоліків застосунку MyTherapy слід віднести: відсутність повноцінного веб-застосунку; обмежені аналітичні можливості (без machine learning або інтелектуального аналізу); відсутність можливості налаштування під корпоративне або клінічне середовище; відсутність локалізації інтерфейсу українською мовою; обмежені можливості адміністрування для медичних установ.

Рис. 1.2 Приклади екранних форм застосунку MyTherapy

1.2.3 Порівняльна таблиця аналогів

На основі проведеного огляду складено порівняльну таблицю існуючих рішень – таблиця 1.1.

Таблиця 1.1

Порівняльна характеристика програмних продуктів
для обліку прийому ліків

Критерій порівняння	Medisafe (Моб. додаток)	MyTherapy (Моб. додаток)	MedicineTracker
Платформа	iOS / Android	iOS / Android	Web (Доступ з будь-якого пристрою)
Календарний інтерфейс	Так (список)	Так (щоденник)	Так (Інтерактивний FullCalendar)
Система нагадувань	Push-сповіщення	Push-сповіщення	In-App сповіщення (Real-time)
Аналітика дисципліни	Базова	Розширена	Інтелектуальний статус (SimpleService)
Інтелектуальний помічник	Відсутній	Відсутній	NLP Chatbot (Engati)
Складність інтерфейсу	Висока (багато реклами)	Середня	Низька (мінімалізм та зручність)
Вартість	Freemium (є платні функції)	Безкоштовно	Повністю безкоштовно

Аналіз наведеної таблиці свідчить про те, що жодне з попередньо розглянутих рішень не задовольняє повністю потреби в наявності веб-інтерфейсу, інтелектуального аналітичного модуля, можливості клінічного використання одночасно. Це підтверджує актуальність розробки власного веб-застосунку.

1.2.4 Актуальність розробки

На підставі проведеного аналізу аналогів можна зробити висновок про доцільність розробки власного веб-застосунку для обліку прийому ліків. Запропонований застосунок відрізняється від аналогів такими ключовими характеристиками:

- повноцінний веб-інтерфейс на базі ASP.NET MVC, доступний з будь-якого пристрою без необхідності встановлення мобільного застосунку;
- інтелектуальний модуль контролю, що використовує алгоритми аналізу даних для виявлення патернів порушення режиму прийому та формування попереджень;
- календар нагадувань з можливістю гнучкого налаштування розкладу прийому препаратів;
- підтримка роботи з кількома користувачами;
- зберігання даних у реляційній базі Microsoft SQL Server з використанням Entity Framework для зручного доступу до даних;
- відповідність вимогам безпеки та конфіденційності медичних даних.

1.3 Огляд IT-засобів для розробки

1.3.1 ASP.NET Core MVC

ASP.NET Core [13] – це крос-платформний, високопродуктивний фреймворк з відкритим вихідним кодом від компанії Microsoft для побудови сучасних хмарних, підключених до інтернету застосунків. Архітектурний шаблон MVC (Model-View-Controller) [9] забезпечує чіткий розподіл відповідальності між рівнями застосунку: моделями даних, контролерами бізнес-логіки та представленнями (views) для відображення інформації.

Основні переваги ASP.NET Core MVC для даного проєкту: висока продуктивність і масштабованість; розвинена екосистема бібліотек та пакетів NuGet; вбудована підтримка автентифікації та авторизації (ASP.NET Core Identity);

зручна система маршрутизації; підтримка Razor Pages та Razor синтаксису для формування HTML-сторінок; інтеграція з Entity Framework Core.

1.3.2 Entity Framework Core

Entity Framework Core (EF Core) [14] – фреймворк для платформи .NET. Він дозволяє розробникам працювати з базою даних, використовуючи об'єкти .NET, без необхідності написання більшої частини коду доступу до даних. EF Core підтримує підхід Code-First, при якому структура бази даних генерується автоматично на основі класів-моделей C# за допомогою міграцій [18]. Для даного проєкту EF Core забезпечує: автоматичне відображення класів C# [7]; зручний LINQ-синтаксис для запитів до бази даних; систему міграцій для управління змінами у базі даних; ледаче (lazy) та нетерпляче (eager) завантаження даних.

1.3.3 Microsoft SQL Server

Microsoft SQL Server – реляційна система управління базами даних (СУБД), розроблена компанією Microsoft. Для цілей даного проєкту SQL Server є оптимальним вибором завдяки: повній інтеграції з екосистемою Microsoft; надійності та безпеці зберігання медичних даних; підтримці транзакцій за принципом ACID; розвиненому набору інструментів адміністрування (SSMS); можливості локального розгортання. Схема бази даних включає таблиці для зберігання інформації про користувачів, препарати, курси лікування, розклади прийому та записи про фактичні прийоми ліків. Зв'язки між таблицями реалізовані через зовнішні ключі, що забезпечує цілісність даних.

1.3.4 ASP.NET Core Identity

ASP.NET Core Identity [4] – система автентифікації та авторизації, вбудована у ASP.NET Core. Вона надає готовий функціонал для управління користувачами, ролями, паролями, зовнішніми провайдерами входу та токенами безпеки. Для медичного застосунку захист персональних даних є критично важливим, тому Identity є відповідним засобом для реалізації системи контролю доступу.

1.3.5 Bootstrap та JavaScript

Bootstrap 5 – популярна CSS-бібліотека для побудови адаптивного веб-інтерфейсу. Bootstrap дозволяє швидко створити інтуїтивно зрозумілий та візуально привабливий інтерфейс, що коректно відображається на пристроях з різною роздільною здатністю екрану. Разом з Bootstrap застосовується JavaScript (бібліотека jQuery) для реалізації динамічних елементів інтерфейсу: модальних вікон, інтерактивного календаря нагадувань [5].

1.3.6 FullCalendar

FullCalendar [8] – відкрита JavaScript-бібліотека для відображення та управління подіями у форматі інтерактивного календаря. Підтримує різні режими перегляду (місяць, тиждень, день), drag-and-drop для перенесення подій, кольорове кодування та інтеграцію з бекенд-API. Для даного проєкту FullCalendar використовується як основний компонент календаря нагадувань про прийом ліків.

1.3.7 Visual Studio

Microsoft Visual Studio 2022 – інтегроване середовище розробки, що надає повний набір інструментів для розробки та тестування ASP.NET застосунків.

Таблиця 1.2

IT-засоби, що використовуються при розробці застосунку

Категорія	Технологія	Функціональна роль
Backend	ASP.NET Core 8.0	Обробка бізнес-логіки системи
Data Access	EF Core 8.0	Взаємодія з базою даних
Database	MS SQL Server 2022	Сховище медичної інформації
Security	Identity Core	Автентифікація та захист даних
Frontend	Razor, Bootstrap 5.3	Побудова адаптивних сторінок
Intelligence	Engati (NLP Cloud)	Допоміжний інтелектуальний помічник

Visual Studio забезпечує зручну роботу з проєктами .NET, інструменти профілювання та інтеграцію з SQL Server через Server Explorer. Даний інструментарій обрано як базове середовище проєктування завдяки його високій продуктивності та глибокій інтеграції з Microsoft. IDE містить вбудовані засоби для управління залежностями (NuGet), інструменти для автоматизованого модульного тестування. Більш детальний огляд засобів, використаних у розробці застосунку наведено у таблиці 1.2.

1.4 Формування концепції та технічних завдань розробки

Для досягнення поставленої мети необхідно вирішити такі задачі:

- провести аналіз предметної галузі та існуючих програмних рішень для обліку прийому ліків, визначити їх недоліки та обґрунтувати актуальність розробки;
- здійснити вибір та обґрунтування технологій для розробки (ASP.NET Core MVC, Entity Framework Core, SQL Server,);
- спроєктувати архітектуру застосунку за шаблоном MVC та структуру бази даних у Microsoft SQL Server;
- розробити модуль реєстрації та автентифікації користувачів;
- реалізувати функціонал управління препаратами та курсами лікування (додавання, редагування, видалення);
- розробити модуль планування розкладу прийому ліків з інтеграцією інтерактивного календаря нагадувань на основі FullCalendar;
- реалізувати механізм фіксації фактичних прийомів ліків та формування звітності;
- провести тестування розробленого застосунку та оцінити його функціональність.

Вирішення перерахованих задач дозволить створити повноцінний веб-застосунок, що відповідає сучасним вимогам до медичного програмного забезпечення та потребам кінцевих користувачів.

2 АНАЛІЗ ЗАСОБІВ РЕАЛІЗАЦІЇ ТА СТРУКТУРИ ПРОГРАМНОЇ СИСТЕМИ

2.1 Проєктування архітектури системи

2.1.1 Загальна архітектура застосунку

Розроблюваний веб-застосунок «MedicineTracker» базується на трирівневій клієнт-серверній архітектурі, яка є галузевим стандартом для побудови масштабованих та надійних інформаційних систем. Основним принципом побудови такої структури є чітке розмежування обов'язків між функціональними рівнями, що дозволяє ізолювати логіку представлення від процесів обробки даних та безпосередньої взаємодії з базою даних. Такий підхід суттєво підвищує ремонтопридатність системи, оскільки зміни в одному з рівнів мінімально впливають на роботу інших компонентів.

Верхній рівень – рівень презентації (Presentation Layer) – реалізований за допомогою стеку технологій ASP.NET Core MVC та Razor Views. Його основним завданням є отримання вхідних даних від користувача, їх первинна валідація на стороні клієнта та формування графічного інтерфейсу. Використання асинхронних технологій, таких як AJAX [3] та Fetch API, дозволяє рівню презентації динамічно оновлювати окремі частини сторінки (наприклад, календар розкладу чи статус прийому ліків) без повного перезавантаження ресурсу. Це забезпечує безперервність користувацького досвіду та знижує навантаження на мережевий канал зв'язку.

Середній рівень системи – рівень бізнес-логіки (Business Logic Layer) – виступає сполучною ланкою та координатором роботи всього застосунку. Тут зосереджені контролери та спеціалізовані сервіси, зокрема SimpleService, які відповідають за виконання складних операцій: розрахунок комплаєнтності пацієнта, інтерпретацію аналітичних статусів («Perfect», «Critical» тощо) та управління життєвим циклом сповіщень. На цьому рівні реалізовано інтеграцію з

механізмами ASP.NET Core Identity, що забезпечує сувору ізоляцію даних – кожен користувач має доступ виключно до власного профілю та історії лікування.

Нижній рівень – рівень доступу до даних (Data Access Layer) – забезпечує персистентне зберігання інформації під управлінням СУБД MS SQL Server. Взаємодія між програмним кодом та базою даних реалізована через об'єктно-реляційне відображення за допомогою фреймворку Entity Framework Core 8.0. У даному проєкті архітектурним центром цього рівня є об'єкт контексту ApplicationDbContext. Вибір на користь прямої взаємодії з контекстом даних (без додаткових абстракцій у вигляді репозиторіїв) зумовлений прагненням до лаконічності та високої продуктивності системи. Таке рішення дозволяє розробнику повною мірою використовувати переваги вбудованого механізму відстеження змін (Change Tracking) та мову LINQ для формування складних запитів до бази даних, що значно прискорює розробку та спрощує подальшу підтримку коду в межах навчальної кваліфікаційної роботи.

2.1.2 Архітектурний шаблон MVC

Архітектурний шаблон Model-View-Controller є основою організації коду ASP.NET Core. Модель (Model) відповідає за представлення даних і бізнес-сутностей, включаючи класи Entity, ViewModel та DTO. Представлення (View) реалізовано на Razor-шаблонах (.cshtml), що поєднують HTML із синтаксисом C# та не містять бізнес-логіки. Контролер (Controller) обробляє HTTP-запити, звертається до сервісів та передає результат представленню або повертає JSON для AJAX-запитів.

Взаємодія компонентів відбувається через систему маршрутизації ASP.NET Core, що відображає URL-адреси на відповідні Action-методи контролерів. Стандартний шаблон маршруту: {controller}/{action}/{id?}.

2.1.3 Структура проєкту

Логічна організація програмного коду веб-застосунку «MedicineTracker» відповідає стандартам розробки на платформі .NET 8.0, що базуються на принципі

декомпозиції функціоналу. Структура проєкту побудована таким чином, щоб забезпечити максимальну читабельність коду та спростити процес налагодження системи. Кожна директорія проєкту виконує роль окремого контейнера для логічно пов'язаних елементів, що дозволяє розробнику швидко орієнтуватися в архітектурі системи.

Центральне місце в організації проєкту займають три ключові папки архітектурного шаблону MVC. Папка Controllers містить класи, які виступають в ролі вхідних точок для всіх HTTP-запитів.

Вони не містять важкої бізнес-логіки, а лише координують взаємодію між вхідними даними, сервісним шаром та представленнями. Папка Models об'єднує сутності бази даних, які описують структуру таблиць (Medicine, MedicationSchedule), а також спеціалізовані моделі представлення (ViewModels), призначені для передачі лише необхідного набору даних на клієнтську сторону, що підвищує безпеку системи. Директорія Views зберігає Razor-шаблони, де HTML-розмітка поєднується з мовою C# для динамічної генерації сторінок на стороні сервера.

Для винесення логіки, що виходить за межі стандартного циклу «запит-відповідь», у проєкті передбачено додаткові структурні одиниці. Папка Services містить сервісний шар, зокрема клас SimpleService, який інкапсулює алгоритми аналізу та розрахунків, роблячи код контролерів більш лаконічним.

Директорія Data є фундаментом рівня доступу до даних; тут зберігається клас контексту ApplicationDbContext та історія міграцій, що забезпечують еволюцію схеми бази даних.

У таблиці 2.1 продемонстровано загальну структуру проєкту, а саме призначення основних файлів проєкту.

Таблиця 2.1

Структура проєкту ASP.NET Core MVC

Папка / файл	Призначення
Controllers/	Класи контролерів (Medicine, Schedule, Home) для обробки запитів та керування логікою.
Models/	Класи сутностей EF Core: Medicine, MedicationSchedule, ErrorViewModel
Views/	Razor-шаблони (.cshtml) для кожного контролера; папка Shared містить _Layout.cshtml
Services/	Інтерфейси та реалізації сервісів: SimpleService
Data/	ApplicationDbContext, Repository<T>, UnitOfWork, папка Migrations із міграціями EF Core
wwwroot/	Статичні файли: CSS (Bootstrap 5), JavaScript (jQuery, FullCalendar), шрифти, зображення
appsettings.json	Конфігураційний файл із налаштуваннями бази даних та параметрами безпеки.
Program.cs	Точка входу в застосунок, де виконується реєстрація сервісів та налаштування конвеєра Middleware.

2.1.4 Dependency Injection та сервісний рівень

Для забезпечення високої гнучкості, масштабованості та тестованості архітектури застосунку MedicineTracker було використано патерн «Впровадження залежностей» (Dependency Injection, DI). У системі застосовано вбудований DI-контейнер платформи ASP.NET Core 8.0, який керує життєвим циклом об'єктів та забезпечує їх автоматичне створення та передачу в класи-споживачі.

Централізація управління залежностями реалізована у файлі конфігурації Program.cs. Усі ключові компоненти бізнес-логіки реєструються в контейнері через відповідні абстракції (інтерфейси). Наприклад, для роботи з даними медикаментів використовується реєстрація виду:

builder.Services.AddScoped<IMedicationService, MedicationService>().

Вибір життєвого циклу Scoped зумовлений специфікою веб-запитів: новий екземпляр сервісу створюється для кожного окремого HTTP-запиту, що гарантує ізоляцію даних та оптимальне використання ресурсів сервера.

Застосування сервісного рівня (Service Layer) дозволяє відокремити бізнес-логіку від логіки обробки HTTP-запитів у контролерах. Контролери не взаємодіють з базою даних напряму через контекст Entity Framework, а отримують вже готові методи через конструктори. Це реалізує принцип інверсії управління (Inversion of Control) та забезпечує «слабку зв'язаність» компонентів системи.Такий підхід має кілька ключових переваг для проєкту:

– Модульність: Можливість повної заміни логіки певного модуля (наприклад, перехід від локальних сповіщень до зовнішнього сервісу розсилки) без внесення змін у код контролерів.

– Спрощене тестування: Використання інтерфейсів дозволяє легко підміняти реальні сервіси на Mock-об’єкти (імітації) під час проведення Unit-тестування, що описано в четвертому розділі роботи.

– Чистота коду: Контролери залишаються максимально лаконічними («тонкими»), орієнтованими лише на маршрутизацію та повернення представлень, тоді як вся складна логіка розрахунку дисципліни прийому ліків зосереджена в спеціалізованих сервісах, таких як SimpleService.

Інтеграція інтелектуального модуля підтримки також реалізована через DI, що дозволяє системі звертатися до API Engati як до абстрактного сервісу повідомлень, забезпечуючи стабільну роботу NLP-компонента в межах загальної екосистеми застосунку.

2.2 Моделювання вимог до системи

2.2.1 Функціональні вимоги

Функціональні вимоги визначають конкретні дії та сервіси, які система повинна надавати користувачу для вирішення задач обліку медикаментів. У проєкті

MedicineTracker основний акцент зроблено на автоматизацію життєвого циклу лікування: від внесення препарату до бази до аналізу успішності курсу. Систематизований перелік функціональних вимог наведено в таблиці 2.2.

Таблиця 2.2

Функціональні вимоги до веб-застосунку MedicineTracker

Код	Назва вимоги	Опис функціональної можливості	Пріоритет
FR-01	Керування доступом (Identity)	Реєстрація нового користувача, безпечний вхід, вихід та валідація даних через ASP.NET Identity.	Високий
FR-02	Облік медикаментів	Можливість створення персонального каталогу ліків (назва, опис, дозування, залишок у пакуванні).	Високий
FR-03	Планування терапії	Створення курсів лікування з визначенням дати початку, завершення та частоти прийому.	Високий
FR-04	Календарне планування	Візуалізація графіка прийому ліків у вигляді інтерактивної сітки на базі бібліотеки FullCalendar.	Високий
FR-05	Моніторинг виконання	Можливість одним кліком позначити факт прийому ліків або вказати причину пропуску.	Високий
FR-06	Real-time сповіщення	Генерація миттєвих повідомлень у браузері про настання часу прийому чергової дози.	Високий
FR-07	Інтелектуальний аналіз	Автоматичний розрахунок відсотка дисципліни прийому на основі історії записів.	Середній

Продовження таблиці 2.2

Функціональні вимоги до веб-застосунку MedicineTracker

FR-08	NLP-підтримка	Взаємодія з чат-ботом Engati для отримання довідкової інформації про роботу з інтерфейсом системи.	Середній
FR-09	Адміністрування	Управління ролями користувачів та контроль цілісності загальних довідників препаратів.	Низький

2.2.2 Нефункціональні вимоги

Нефункціональні вимоги визначають атрибути якості системи, такі як надійність, безпека та зручність використання. Детальні характеристики нефункціональних вимог представлені в таблиці 2.3.

Таблиця 2.3

Характеристика нефункціональних вимог до системи

Категорія	Опис вимоги до якості	Технологічне забезпечення
Безпека даних	Забезпечення конфіденційності медичних записів. Паролі не зберігаються у відкритому вигляді.	Хешування алгоритмом BCrypt/SHA-256 через Identity; захист від CSRF та XSS.
Швидкодія	Час обробки запиту на отримання розкладу не повинен перевищувати 1.5 секунди.	Оптимізація LINQ-запитів до SQL Server; асинхронне виконання задач (async/await).

Продовження таблиці 2.3

Характеристика нефункціональних вимог до системи

Надійність	Стабільна робота системи при пікових навантаженнях та збереження даних при збоях.	Використання транзакцій в Entity Framework; щоденне резервне копіювання БД.
Юзабіліті	Інтуїтивно зрозумілий інтерфейс, що не потребує тривалого навчання користувача.	Відповідність принципам Material Design; локалізація інтерфейсу українською мовою.
Адаптивність	Коректне відображення інтерфейсу на смартфонах та десктопах (Mobile First).	Використання сітки Bootstrap 5 та медіа-запитів CSS для різних екранів.
Масштабованість	Можливість додавання нових модулів (наприклад, аналіз взаємодії ліків) без рефакторингу ядра.	Архітектура на базі Dependency Injection та слабка зв'язаність компонентів.

2.2.3 Діаграма варіантів використання

Для візуалізації функціональних можливостей застосунку та визначення меж взаємодії між людиною та програмним комплексом побудовано діаграму варіантів використання (Use Case Diagram). У межах даної бакалаврської роботи архітектура системи спроєктована з фокусом на кінцевого споживача, тому ключовим і єдиним активним актором виступає Користувач (Пацієнт).

Централізація ролі Користувача дозволяє реалізувати концепцію персоналізованого медичного кабінету, де особа має повний контроль над власними даними без необхідності зовнішнього моделювання з боку медичного персоналу на етапі прототипу. Взаємодія актора із системою охоплює повний цикл

менеджменту лікування: від первинного внесення інформації до отримання інтелектуальної підтримки.

Основними варіантами використання, що відображені на діаграмі, є:

– Управління профілем та автентифікація: ініціалізація сесії в системі, що забезпечує безпечний доступ до конфіденційних медичних записів.

– Ведення каталогу медикаментів: додавання нових препаратів, редагування існуючих записів та контроль залишків ліків у домашній аптечці.

– Формування терапевтичного графіку: налаштування часових слотів прийому, визначення тривалості курсу та дозування.

– Взаємодія з інтерактивним календарем: візуальний моніторинг плану лікування та позначення фактичного виконання призначень.

Така структура взаємодії забезпечує логічну цілісність системи MedicineTracker, де кожен варіант використання безпосередньо спрямований на досягнення мети роботи – підвищення дисципліни прийому лікарських засобів. Побудована діаграма є основою для подальшого проєктування структури бази даних та логіки контролерів, оскільки кожен Use Case трансформується у конкретний набір методів сервісного рівня застосунку.

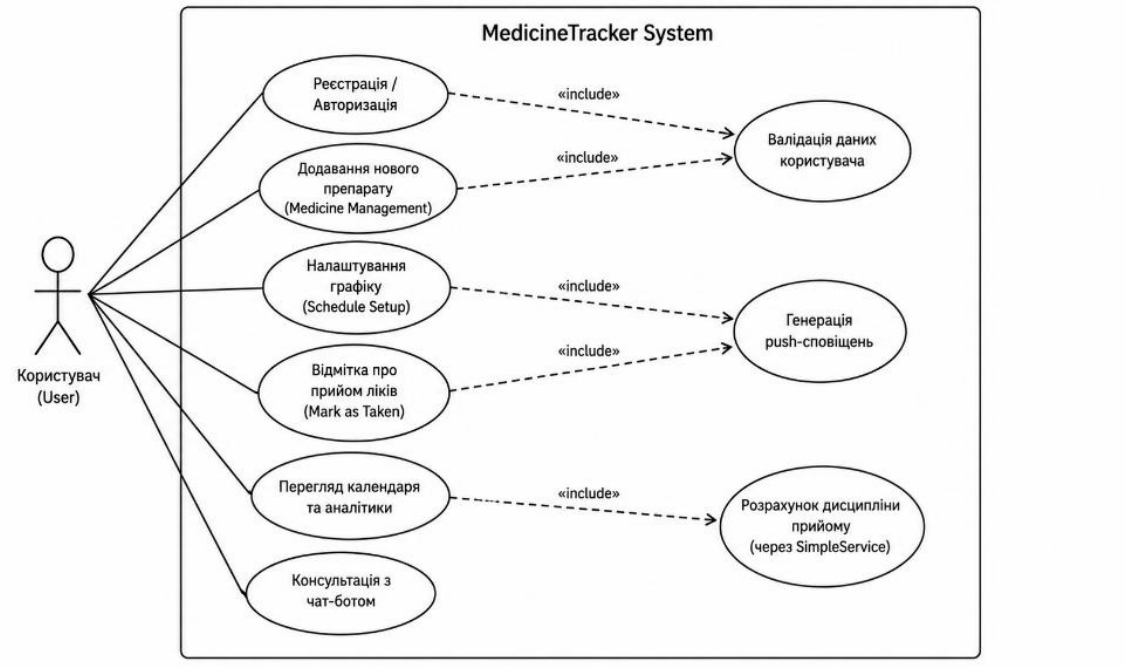


Рис. 2.1 Діаграма варіантів використання системи (Use Case)

2.2.4 Специфікація сценаріїв варіантів використання

Для деталізації логіки взаємодії користувача з системою MedicineTracker розроблено детальні сценарії для ключових варіантів використання. Це дозволяє чітко визначити послідовність дій актора та реакцію системи на ці дії, враховуючи можливі відхилення від основного потоку.

Таблиця 2.4

Сценарій UC-04: Налаштування персонального розкладу терапії

Поле	Опис
Ідентифікатор	UC-04
Назва	Налаштування розкладу прийому ліків
Актор	Користувач (Пацієнт)
Передумови	Користувач успішно пройшов автентифікацію; у базі вже створено запис про медикамент.
Основний потік	1. Перехід до панелі керування курсами лікування. 2. Обирає ліки та ініціює команду «Налаштувати розклад». 3. Система генерує форму введення параметрів: дозування, години прийому. 4. Користувач заповнює дані та надсилає форму на сервер. 5. Контролер ASP.NET проводить валідацію вхідних значень. 6. Система зберігає правила розкладу в SQL Server та автоматично генерує події для календаря. 7. Користувач отримує підтвердження про збереження; дані відображаються на FullCalendar.
Альтернативний потік	5а. Невалідні дані – система показує помилки валідації та повертає до форми. 6а. Розклад вже існує – пропонує оновити або створити новий.
Постумови	Структура розкладу зафіксована в БД; інтерактивний календар оновлено.

Однією з головних частин системи – є облік та історія прийому ліків для своєчасного відстежування поведінки користувача. Детальніше розглянемо сценарій підтвердження прийому препарату у таблиці 2.5.

Таблиця 2.5

Сценарій UC-06: Реєстрація факту прийому медикаменту

Поле	Опис
Ідентифікатор	UC-06
Назва	Підтвердження прийому дози ліків
Актор	Користувач (Пацієнт)
Передумови	Користувач автентифікований; настав час прийому згідно з розкладом у БД.
Основний потік	1. Веб-інтерфейс ініціює візуальне сповіщення на дашборді. 2. Користувач натискає кнопку «Прийнято» у вікні нагадування. 3. Система передає запит до сервісного рівня для фіксації події. 4. Статус нагадування змінюється на «Виконано». 5. SimpleService автоматично перераховує загальний прогрес лікування та оновлює графіки на Dashboard.
Альтернативний потік	2а. Відповідь відсутня 2 год – система автоматично встановлює статус «Без відповіді».
Постумови	Запис про виконання створено; показники дисципліни терапії актуалізовано в реальному часі.

2.2.5 Алгоритм функціонування веб-застосунку

Для розуміння внутрішньої логіки роботи MedicineTracker та послідовності обробки даних розроблено узагальнений алгоритм функціонування системи. Він відображає шлях даних від моменту входу користувача до фінального аналізу

ефективності лікування. Процес роботи застосунку складається з наступних ключових етапів:

1. Авторизація та контроль доступу: при вході система перевіряє права доступу через ASP.NET Identity, забезпечуючи безпечне завантаження персонального профілю.

2. Планування та запис даних: під час внесення нових курсів лікування (назва препарату, час, періодичність) дані передаються через Entity Framework до бази даних SQL Server.

3. Активний моніторинг: на стороні клієнта JavaScript-скрипт здійснює фонові запити до методу GetTodayReminders з інтервалом у 10 секунд. Це забезпечує актуальність даних на дашборді без повного перезавантаження сторінки.

4. Система сповіщень: у разі збігу системного часу з часом прийому за розкладом, спрацьовує функція showInAppNotification, яка викликає модальне вікно або звуковий сигнал у браузері.

5. Аналіз та візуалізація: після того, як користувач позначає дозу як «Прийняту», система оновлює лог прийомів. В цей же момент сервіс бізнес-логіки SimpleService виконує перерахунок прогресу (наприклад, відсоток виконаних прийомів до загальної кількості), що миттєво відображається на графічних елементах Dashboard. Нижче наведено графічне представлення описаного алгоритму.

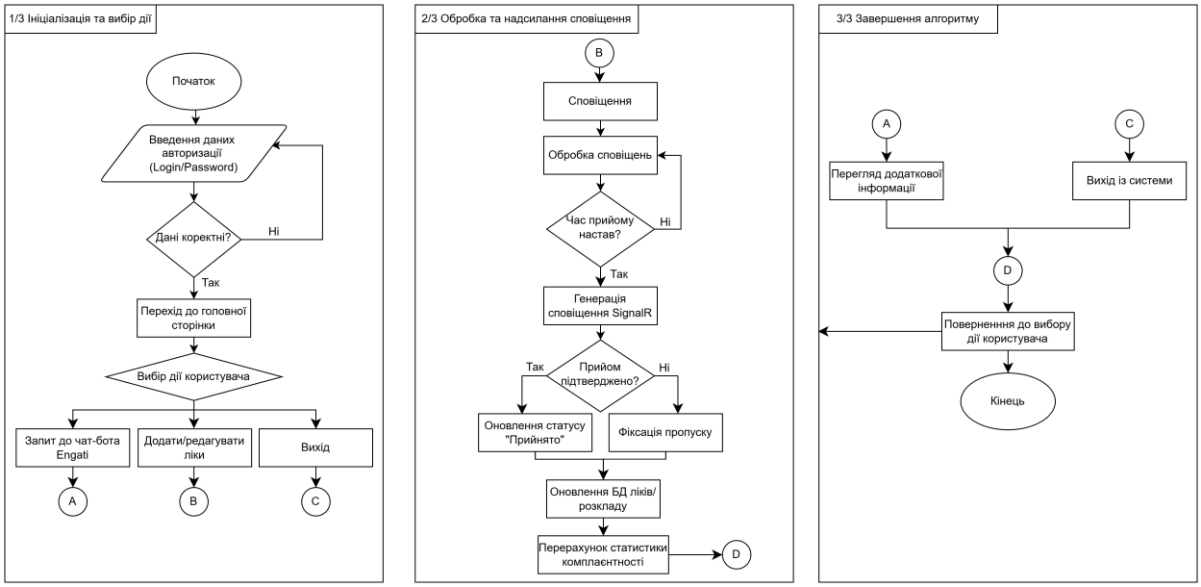


Рис. 2.2 Схема алгоритму роботи системи MedicineTracker

2.3 Проєктування структури бази даних

2.3.1 Концептуальна модель даних

База даних системи MedicineTracker реалізована на базі СУБД Microsoft SQL Server 2022. Проєктування схеми даних базувалося на принципах нормалізації (до третьої нормальної форми – 3НФ включно), що дозволяє мінімізувати дублювання інформації та забезпечити високу швидкість обробки транзакцій.

Взаємодія із шаром даних здійснюється виключно через ORM Entity Framework Core 8.0. Це забезпечує безпеку на рівні доступу до даних, оскільки використання параметризованих запитів LINQ унеможливлює проведення атак типу SQL-ін'єкцій. Структура БД включає таблиці для автентифікації користувачів, управління довідниками ліків, планування графіків та фіксації історії прийомів.

2.3.2 Опис таблиць бази даних

Для забезпечення роботи функціоналу системи MedicineTracker було спроєктовано наступний набір таблиць, представлений у таблиці 2.6.

Таблиця 2.6

Таблиці бази даних та їх призначення

Таблиця	Призначення	Ключові поля
AspNetUsers	Зберігання облікових даних користувачів, їх ролей та параметрів безпеки.	Id (PK), Email, PasswordHash
Medicines	Ведення персонального реєстру лікарських засобів, закріплених за конкретними користувачами.	Id (PK), UserId (FK), Name, Description
Schedules	Зберігання налаштувань графіку прийому ліків (періодичність, конкретний час доби, статус активності).	Id (PK), MedicationId (FK), Frequency, TimeOfDay, IsActive
__EFMigrationsHistory	Системна таблиця, що генерується фреймворком EF Core автоматично для контролю версій схеми БД та застосованих міграцій.	MigrationId (PK), ProductVersion

2.4 Проєктування інтерфейсу користувача

2.4.1 Вимоги до юзабіліті

Інтерфейс системи MedicineTracker розроблено з урахуванням особливостей медичного програмного забезпечення, де головним пріоритетом є зниження когнітивного навантаження на пацієнта. Проєктування базується на стандартах WCAG 2.1 (рівень AA) [19] та евристиках Якоба Нільсена [11], що гарантує доступність та зручність використання. Основні принципи юзабіліті, що впроваджені в проєкті, деталізовано в таблиці 2.7.

Таблиця 2.7

Принципи юзабіліті та їх практична реалізація

Принцип	Реалізація в системі MedicineTracker
Видимість стану системи	Відображення прогрес-барів заповнення курсу на дашборді. Використання Toast-повідомлень для підтвердження збереження ліків.
Схожість із реальністю	Використання зрозумілих піктограм (аптечка, календар, годинник). Дати та час відображаються у локальному форматі (24-год).
Контроль та свобода	Можливість легкого скасування редагування розкладу. Наявність діалогових вікон підтвердження перед видаленням препарату.
Естетика та мінімалізм	Кольорова схема на базі синього, зеленого та білого кольорів, що асоціюються з медициною та спокоєм. Відсутність зайвих елементів.
Гнучкість та ефективність	Швидкий доступ до відмітки прийому ліків безпосередньо з інтерактивного календаря в один клік.
Адаптивність (Mobile First)	Використання компонентів Bootstrap 5, що забезпечує коректну роботу інтерфейсу як на ПК, так і на смартфонах.

2.4.2 Логіка навігації та архітектура переходів

Для забезпечення інтуїтивно зрозумілого доступу до всіх модулів системи MedicineTracker розроблено ієрархічну структуру переходів, що базується на принципі мінімальної кількості кліків для досягнення цілі. Навігаційна модель розділена на дві основні області: публічну та авторизовану. Публічна частина охоплює лендінг, який знайомить користувача з можливостями сервісу, а також сторінки реєстрації та входу, реалізовані засобами ASP.NET Identity.

Після успішної автентифікації користувач потрапляє до захищеного робочого середовища, центром якого є інформаційна панель (дашборд). З головної

панелі за допомогою верхнього навігаційного меню (Navbar) здійснюється перехід до каталогу медикаментів, де реалізовано повний цикл управління картками препаратів, та до розділу курсів лікування. Окремим функціональним вузлом виступає сторінка інтерактивного календаря, яка інтегрує дані з бази даних у візуальну сітку. Крім того, архітектура передбачає виокремлений модуль аналітики, де результати обробки даних сервісом SimpleService трансформуються у графічні звіти, та інтерфейс взаємодії з інтелектуальним чат-ботом Engati для отримання оперативної допомоги.

2.4.3 Опис візуальної організації інтерфейсу

Візуальна концепція головного екрана системи MedicineTracker базується на трьохпанельному компонуванні, що дозволяє раціонально розподілити інформаційне навантаження. Ліва частина інтерфейсу відведена під вертикальну навігаційну панель, яка забезпечує швидке перемикання між глобальними розділами системи. Центральна, найбільш пріоритетна зона, містить оперативний блок прийомів на поточну добу. Кожен запис у цьому блоці представлений у вигляді інтерактивного елемента з чітким зазначенням назви препарату, дозування та часу, а також функціональними кнопками для миттєвого підтвердження виконання або фіксації пропуску. Під блоком оперативних завдань розташовано модуль візуалізації тижневого прогресу, де у формі динамічних індикаторів відображається загальний рівень дотримання режиму.

Права частина інтерфейсу відведена під допоміжні сервіси. Тут розміщується віджет інтелектуального помічника, який реалізовано через інтеграцію з платформою Engati. Чат-бот знаходиться у згорнутому стані та активується за запитом користувача, надаючи підтримку у форматі природно-мовної взаємодії. Поруч розташовано зону коротких аналітичних резюме, де система виводить розрахований статус стабільності терапії. Окрему увагу приділено сторінці календаря, яка використовує бібліотеку FullCalendar. Вона забезпечує повноцінний огляд лікувального плану на місяць із кольоровим маркуванням подій: виконані

прийоми виділяються зеленим кольором, пропущені – червоним, а заплановані – синім, що дозволяє пацієнту миттєво оцінити загальну картину лікування.

2.4.4 Обґрунтування вибору компонентів інтерфейсу

Реалізація спроєктованого інтерфейсу здійснюється за допомогою бібліотеки Bootstrap 5.3, що обрана як основний фреймворк для забезпечення адаптивності та відповідності концепції «Mobile First». Це дозволяє системі MedicineTracker однаково коректно функціонувати як на широкоформатних моніторах, так і на екранах смартфонів. Для побудови форм введення даних та їх валідації використовуються стандартні компоненти Bootstrap у поєднанні з ASP.NET Tag Helpers, що гарантує цілісність даних на стороні клієнта та сервера.

Інтерактивні елементи, такі як модальні вікна для підтвердження видалення або редагування записів, забезпечують контроль користувача над критичними діями. Система сповіщень реалізована через механізм Bootstrap Toasts, який надає неінтрузивний зворотний зв'язок про успішність операцій. Візуалізація статистичних даних від SimpleService покладена на бібліотеку Chart.js. Завдяки такому поєднанню технологій досягається баланс між функціональною насиченістю та швидкістю завантаження сторінок, що є критично важливим для зручності пацієнта.

2.5 Проєктування підсистеми моніторингу та генерації нагадувань

Система нагадувань є фундаментальним функціональним вузлом застосунку MedicineTracker, оскільки саме вона забезпечує активну підтримку пацієнта в дотриманні режиму терапії. Архітектурно дана підсистема спроєктована як асинхронний фоновий комплекс, що інтегрується з базою даних та клієнтським інтерфейсом для забезпечення доставки повідомлень у режимі реального часу.

Процес підготовки даних покладено на спеціалізований сервіс генерації черги сповіщень. Цей компонент функціонує як фонова служба, що реалізує

інтерфейс IHostedService та активується системою щодоби у визначений час. Основна логіка сервісу полягає в аналізі активних розкладів лікування, збережених у таблиці Schedules. На основі цих даних сервіс здійснює превентивну генерацію записів у таблиці Reminders на наступний добовий цикл для кожного зареєстрованого користувача.

Для безпосереднього відстеження часу прийому ліків використовується служба моніторингу, яка виконує періодичне сканування черги нагадувань. Сервіс з частотою один раз на декілька хвилин перевіряє відповідність поточного системного часу до запланованого часу прийому препарату, враховуючи при цьому статус прочитання повідомлення. У разі виявлення актуальної події, система ініціює передачу даних на рівень представлення.

Створений концентратор повідомлень (Hub) підтримує постійне та стабільне WebSocket-з'єднання, що дозволяє серверу миттєво передавати дані в браузер користувача без необхідності ініціації запиту з боку клієнта. Клієнтський сценарій на мові JavaScript, інтегрований у загальний шаблон сторінок, постійно очікує на відповідну подію від сервера. При отриманні сигналу відбувається динамічне оновлення елементів інтерфейсу: актуалізується лічильник нечитаних повідомлень у навігаційній панелі та з'являється інтерактивне вікно нагадування. Такий механізм забезпечує високу швидкість реакції системи та мінімізує ризик пропуску прийому медикаментів через технічні затримки оновлення сторінок.

3 ПРОГРАМНА РЕАЛІЗАЦІЯ WEB-ЗАСТОСУНКУ

3.1 Структура програмного забезпечення та специфікація ключових класів

Програмна архітектура [17] системи MedicineTracker базується на об'єктно-орієнтованому підході, де взаємодія між компонентами візуалізована за допомогою діаграми класів. Ця структура відображає не лише склад окремих об'єктів, а й логіку зв'язків між таблицями бази даних, що реалізована через технологію Entity Framework Core. Центральним вузлом ієрархії є клас ApplicationUser, який розширює стандартну систему ASP.NET Core Identity. Окрім базових полів для автентифікації, таких як унікальний ідентифікатор Id, ім’я користувача UserName та електронна пошта Email, цей клас виступає власником даних у системі. Відповідно до діаграми, між ApplicationUser та класом Medicine встановлено зв'язок типу «один-до-багатьох», що гарантує ізоляцію даних: кожен користувач може мати власний унікальний перелік медикаментів, недоступний іншим учасникам системи.

Клас Medicine є базовою сутністю для опису лікарських засобів. Він містить набір полів, необхідних для ідентифікації препарату, зокрема назву Name та опис Description, що містить відомості про призначення та дозу. Окрім описових характеристик, клас зберігає зовнішній ключ UserId, який підтримує логічний зв'язок із власником профілю. У межах архітектури системи клас Medicine виконує роль батьківської сутності для розкладу прийомів. Зв'язок «один-до-багатьох» із класом Schedule дозволяє закріпити за одним препаратом необмежену кількість запланованих подій, що відповідає реальним сценаріям тривалого лікування.

Структурна одиниця планування представлена класом Schedule. Цей клас фіксує конкретний момент часу, коли пацієнт має вжити ліки, через поле TimeToTake, а також зберігає стан виконання операції у логічному полі IsTaken. Завдяки наявності зовнішнього ключа MedicineId, кожен пункт розкладу чітко асоціюється з конкретним препаратом із каталогу. Така декомпозиція даних

дозволяє гнучко керувати графіком лікування та відстежувати історію кожного прийому в розрізі часових міток.

Функціональна логіка обробки цих даних винесена у сервісний шар, де ключову роль відіграє клас SimpleService. На відміну від сутностей бази даних, цей клас призначений для виконання розрахунків та аналізу. Основним методом сервісу є CalculateAdherence, який приймає ідентифікатор користувача userId як вхідний параметр. Метод виконує агрегацію даних із пов’язаних об’єктів Schedule, аналізуючи співвідношення значень поля IsTaken до загальної кількості запланованих записів. Результатом роботи сервісу є відсотковий показник виконання плану лікування, який згодом передається до контролерів для відображення на дашборді. Таким чином, діаграма класів демонструє цілісний ланцюжок від ідентифікації користувача до отримання аналітичного звіту про його дисципліну терапії..

Нижче наведено опис основних класів системи з їх атрибутами та методами (відповідний рисунок – діаграма класів UML).

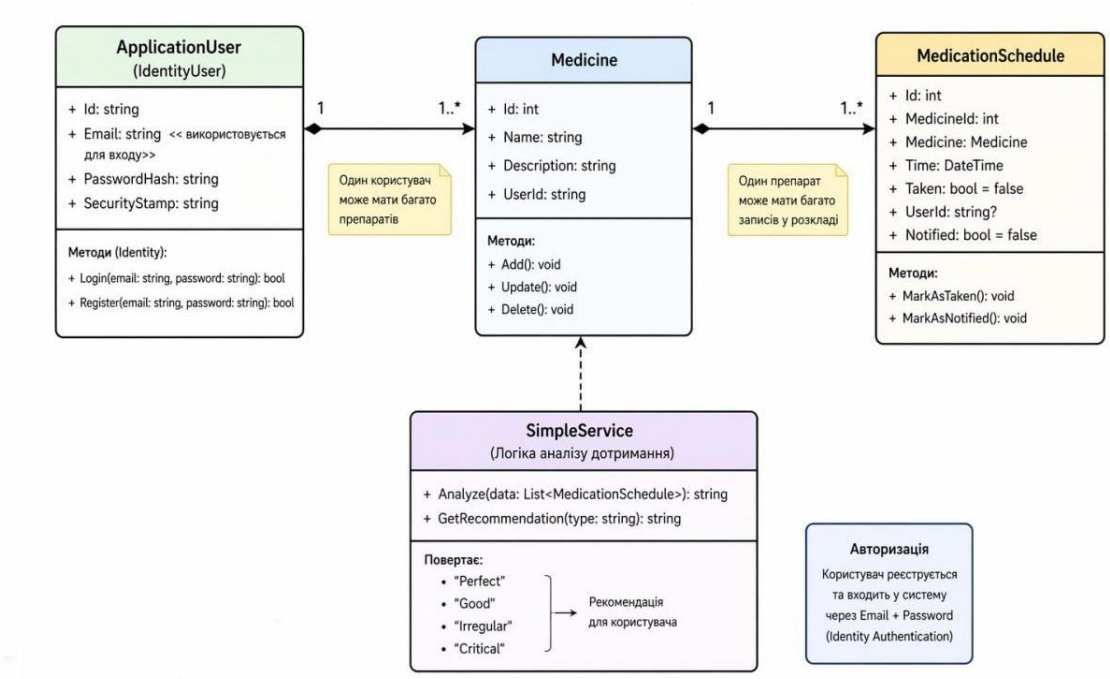


Рис. 3.1 Діаграма класів системи (Class Diagram)

3.2 Реалізація серверної логіки та взаємодії з базою даних

Технологічна реалізація серверної частини базується на використанні фреймворку Entity Framework Core, що дозволяє працювати з базою даних через об'єктну модель. Налаштування взаємодії між сутностями реалізовано всередині методу OnModelCreating класу контексту за допомогою Fluent API. Це дозволило чітко визначити зв'язки типу «один-до-багатьох» між користувачем та його медикаментами, а також налаштувати каскадне видалення пов'язаних розкладів та логів прийому при видаленні картки препарату. Такий підхід забезпечує жорстку цілісність даних на рівні схеми бази даних, мінімізуючи ризик появи «сирітських» записів.

Сервісний шар є найбільш динамічною частиною системи, де зосереджені алгоритми обробки подій. Метод розрахунку прогресу лікування в SimpleService використовує асинхронні запити до бази даних для отримання вибірки за певний період. Логіка методу передбачає фільтрацію записів за ідентифікатором користувача та агрегацію статусів виконання, після чого проводиться математичний розрахунок ефективності терапії. Аналогічно реалізовано логіку в фонових сервісах для генерації щоденних нагадувань, де на основі шаблонів із класу Schedule створюються конкретні екземпляри подій у таблиці Reminders. Використання асинхронних конструкцій async/await дозволяє системі не блокувати основний потік виконання під час тривалих операцій введення-виведення з базою даних.

Окрему увагу приділено безпеці та розмежуванню прав доступу, що реалізовано через інтеграцію ASP.NET Core Identity. Реєстрація користувачів супроводжується автоматичним хешуванням паролів та створенням унікальних ідентифікаторів у таблиці AspNetUsers. Захист маршрутів і методів контролерів забезпечується застосуванням атрибутів [Authorize], які на рівні конвеєра обробки запиту перевіряють наявність активної сесії та відповідність ролі користувача. Це гарантує, що кожен пацієнт має доступ виключно до власних медичних записів та

аналітичних звітів, а доступ до адміністративних функцій суворо обмежений системними правилами, що повністю відповідає вимогам до захисту персональних медичних даних.

Нижче наведено зображення основних елементів бази даних (відповідний рисунок – ER-діаграма).

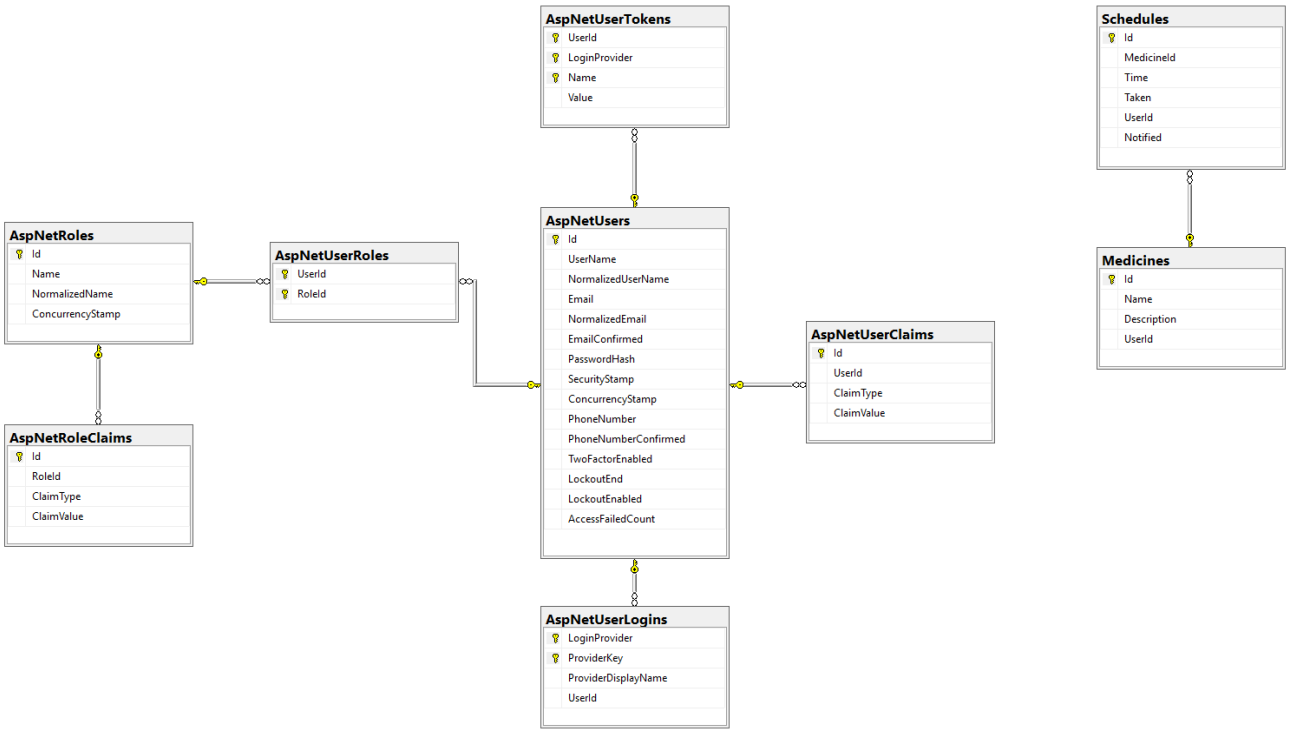


Рис. 3.2 Логічна схема бази даних (ER-діаграма)

3.3 Функціонування інтелектуального модуля (NLP)

Інтелектуальний модуль системи MedicineTracker реалізований як допоміжний інструмент на базі технологій обробки природної мови (NLP) з використанням платформи Engati. Логіка функціонування чат-бота побудована на принципі інтентивного розпізнавання намірів користувача, де вхідний текстовий запит аналізується для визначення конкретної потреби пацієнта. Замість жорсткого алгоритмічного дерева, модуль використовує навчені мовні моделі, що дозволяють ідентифікувати ключові слова та фрази навіть у випадках неточного формулювання запиту. Наприклад, при отриманні повідомлення щодо процедури внесення нових

препаратів, система асоціює запит із відповідним сценарієм підтримки та видає структуровану покрокову інструкцію. Взаємодія побудована у формі циклічного діалогу, де бот не лише надає відповіді, а й може ставити уточнювальні запитання для максимально точної допомоги в межах функціоналу веб-застосунку.

Програмна інтеграція інтелектуального модуля у веб-інтерфейс реалізована на рівні загального шаблону представлення _Layout.cshtml. Це забезпечує наскрізну доступність чат-бота у всіх розділах системи, від дашборду до сторінок аналітики. Технічно інтеграція виконана шляхом вбудовування клієнтського JavaScript-скрипту, який ініціалізує віджет чату при завантаженні сторінки в браузері. Сценарій працює в асинхронному режимі, що дозволяє уникнути затримок при рендерингу основного контенту сайту. Користувацький інтерфейс бота представлений у вигляді компактної плаваючої кнопки, при натисканні на яку відкривається діалогове вікно.

Взаємодія між веб-інтерфейсом та серверами Engati відбувається через захищене з'єднання, що гарантує конфіденційність переданих запитів. Вікно користувача підтримує динамічне оновлення контенту, дозволяючи боту надсилати не лише текстові повідомлення, а й посилання на конкретні розділи застосунку або медіа-інструкції. Завдяки такому підходу, інтелектуальний модуль виконує роль персонального асистента, який знижує поріг входження для нових користувачів та забезпечує оперативне вирішення технічних питань без звернення до адміністратора системи. Це значно підвищує юзабіліті MedicineTracker та сприяє створенню комфортного середовища для регулярного моніторингу лікування.

3.4 Опис інтерфейсу користувача та екранних форм

Візуальна частина системи MedicineTracker розроблена з акцентом на доступність та швидкість взаємодії. Інтерфейс побудований на базі фреймворку Bootstrap 5, що забезпечує адаптивність усіх екранних форм для коректного відображення на мобільних пристроях та стаціонарних комп'ютерах.

3.4.1. Форми авторизації та реєстрації користувачів

Процес роботи із системою починається з модулів ідентифікації, реалізованих за допомогою бібліотеки ASP.NET Core Identity. Форма реєстрації вимагає від користувача введення адреси електронної пошти, яка слугує логіном, та пароля, що має відповідати критеріям безпеки (наявність цифр, спецсимволів та літер різних регістрів).

Валідація полів реалізована на двох рівнях: клієнтському (через jQuery Validation) для миттєвого зворотного зв'язку та серверному для запобігання обходу правил. Система перевіряє унікальність Email у базі даних SQL Server та збіг пароля з полем підтвердження. У разі помилки користувач отримує чітке повідомлення біля відповідного поля інпуту.

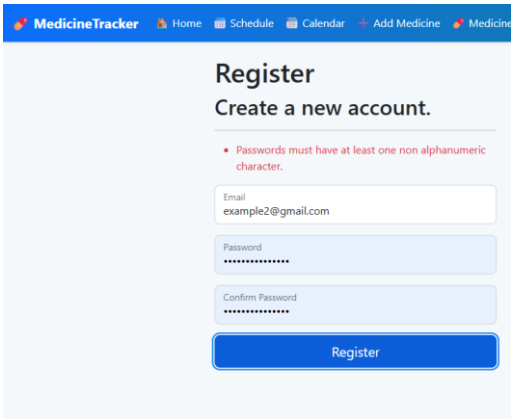


Рис. 3.3 Екранна форма реєстрації в системі

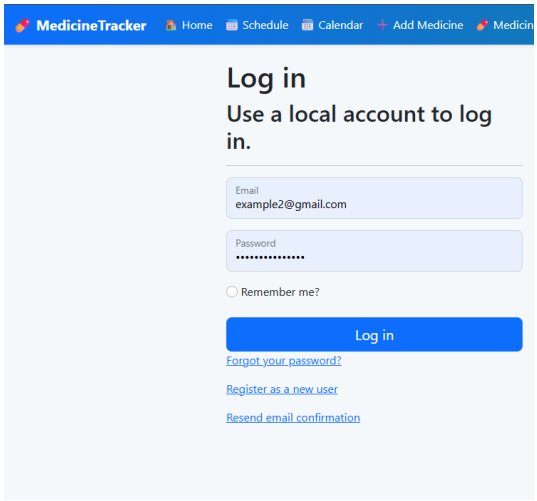


Рис. 3.4 Екранна форма авторизації в системі

3.4.2. Головний дашборд системи

Центральною формою застосунку є головний дашборд, який виступає основним робочим місцем пацієнта. У верхній частині панелі розташовано блок «Прийоми на сьогодні», дані для якого динамічно завантажуються з таблиці ScheduleItems через фільтрацію за поточною датою та ідентифікатором користувача.

Кожен елемент у списку прийомів оснащений інтерактивною кнопкою «Прийнято». При натисканні на неї спрацьовує JavaScript-сценарій, який надсилає асинхронний POST-запит до контролера. Після успішного оновлення статусу в базі даних, сервіс SimpleService у реальному часі перераховує відсоток виконання плану, і прогрес-бар у нижній частині екрана миттєво оновлюється, демонструючи актуальну статистику дотримання режиму (Adherence Score).

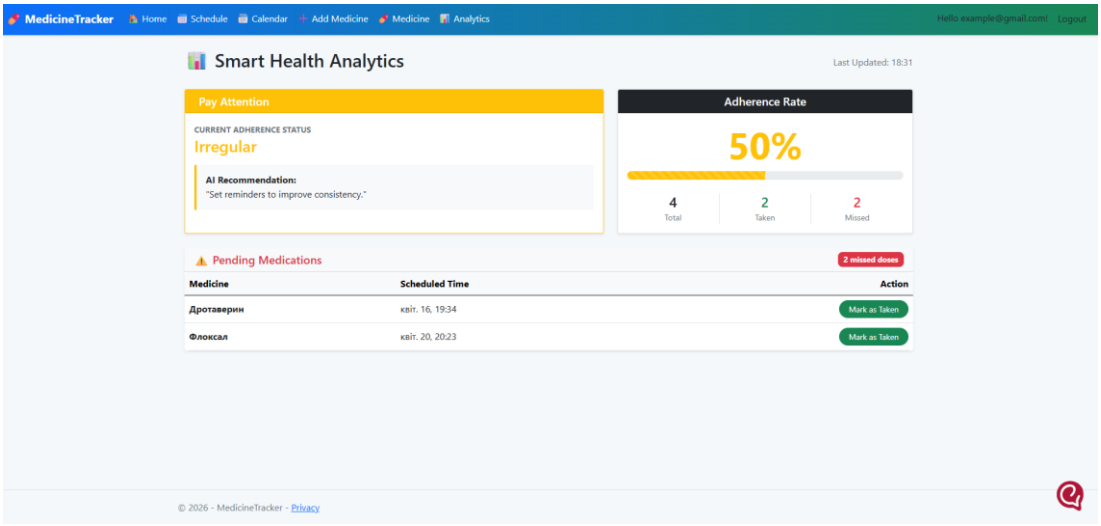


Рис. 3.5 Інтерфейс головного дашборду з аналітикою прогресу

3.4.3. Інтерактивний календар нагадувань

Для довгострокового планування та ретроспективного аналізу використовується сторінка календаря, інтегрована з бібліотекою FullCalendar. Календар відображає всі заплановані події прийому ліків у вигляді кольорових блоків. Подія у сітці календаря містить назву препарату та часом прийому, що дозволяє користувачу швидко перевірити історію терапії за будь-який період.

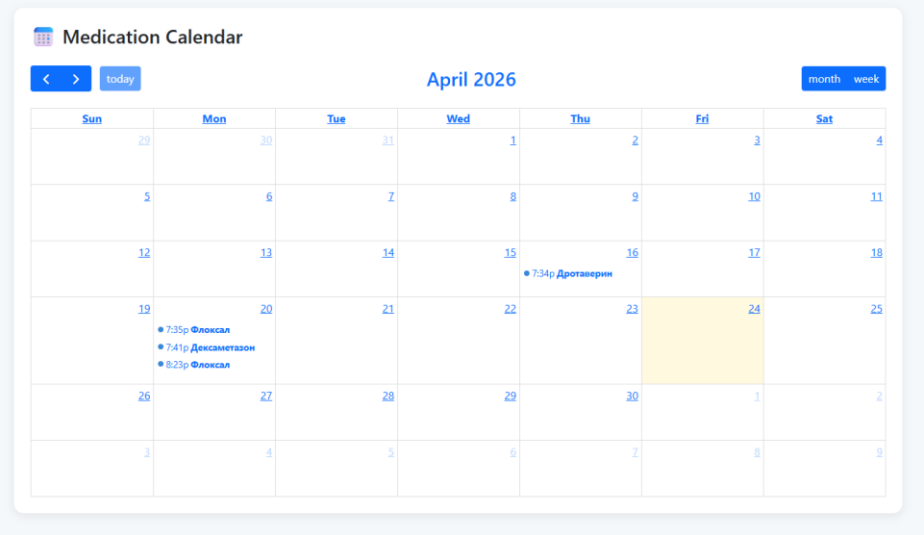


Рис. 3.6 Візуалізація графіка лікування у календарі FullCalendar

3.4.4. Управління каталогом медикаментів

Сторінка управління медикаментами дозволяє користувачу формувати свою персональну «аптечку». Процес додавання нового препарату реалізований через модальну форму або окрему сторінку, де заповнюються поля назви ліків, дозування та категорії.

Після натискання кнопки збереження дані передаються до MedicationService, де створюється новий запис у базі даних, асоційований із поточним користувачем. Особливістю інтерфейсу є зручне зчитування інформації та швидкий перехід до редагування або видалення застарілих позицій.

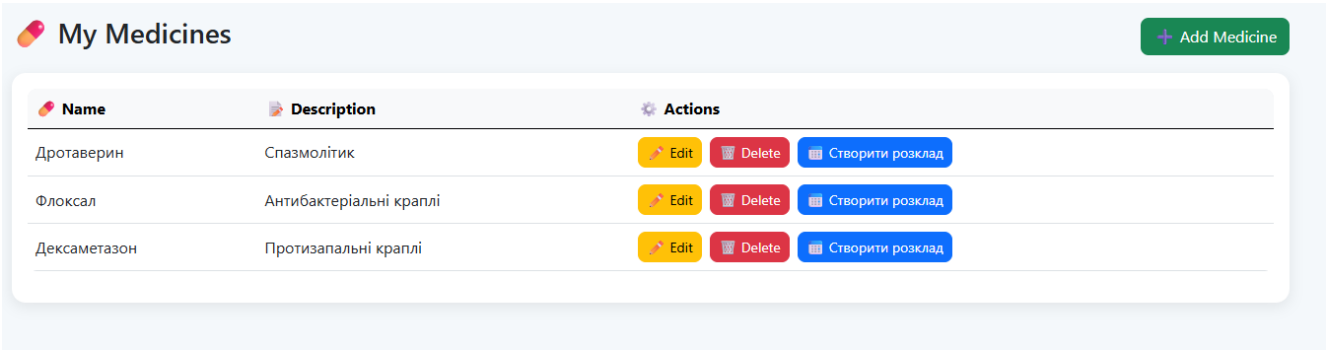


Рис. 3.7 Відображення списку ліків

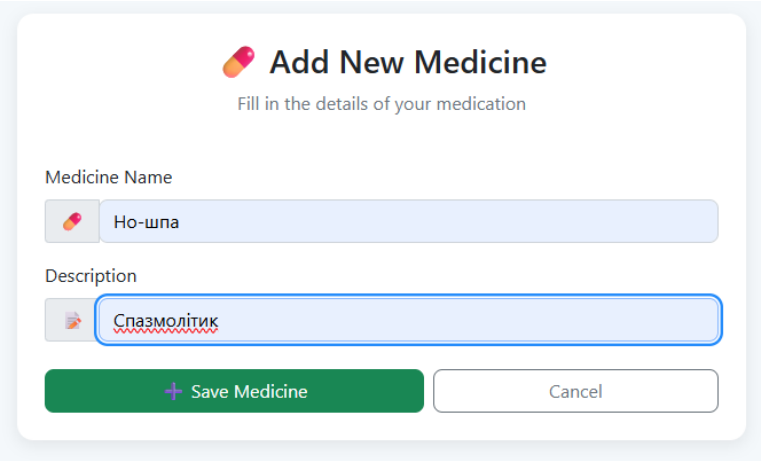


Рис. 3.8 Форма для додавання препарату

3.4.5. Інтерфейс налаштування розкладу прийому

Форма налаштування розкладу є найбільш функціонально насиченою частиною інтерфейсу, оскільки вона забезпечує зв’язок між обраним препаратом та часовою сіткою. Користувач обирає медикамент зі свого списку, після чого відкривається інтерфейс для введення параметрів частоти (наприклад, щодня або через день) та конкретних годин прийому.

Технічно форма реалізована з використанням динамічних полів введення. При виборі кількості прийомів на день система автоматично генерує відповідну кількість полів для вибору часу. Після підтвердження дані обробляються сервісним шаром, який розраховує масив дат і годин на весь період курсу та створює відповідні записи в таблиці ScheduleItem. Валідація на цій формі гарантує, що дата закінчення курсу не може бути раніше дати початку, а інтервали між прийомами мають бути логічно обґрунтованими.

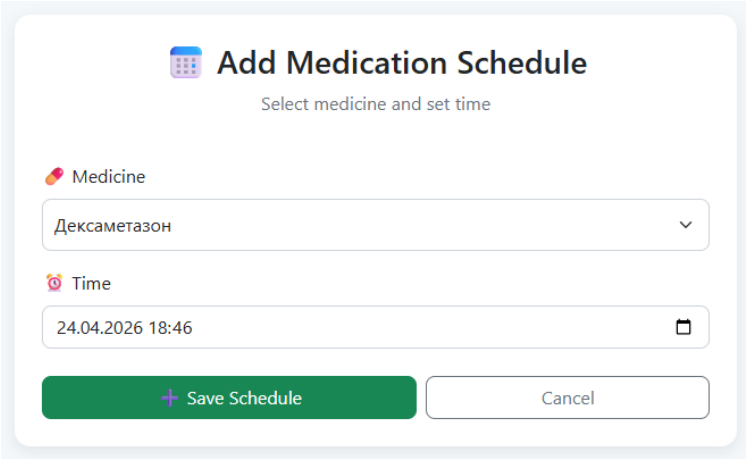


Рис. 3.9 Екранна форма параметризації графіку прийому медикаментів

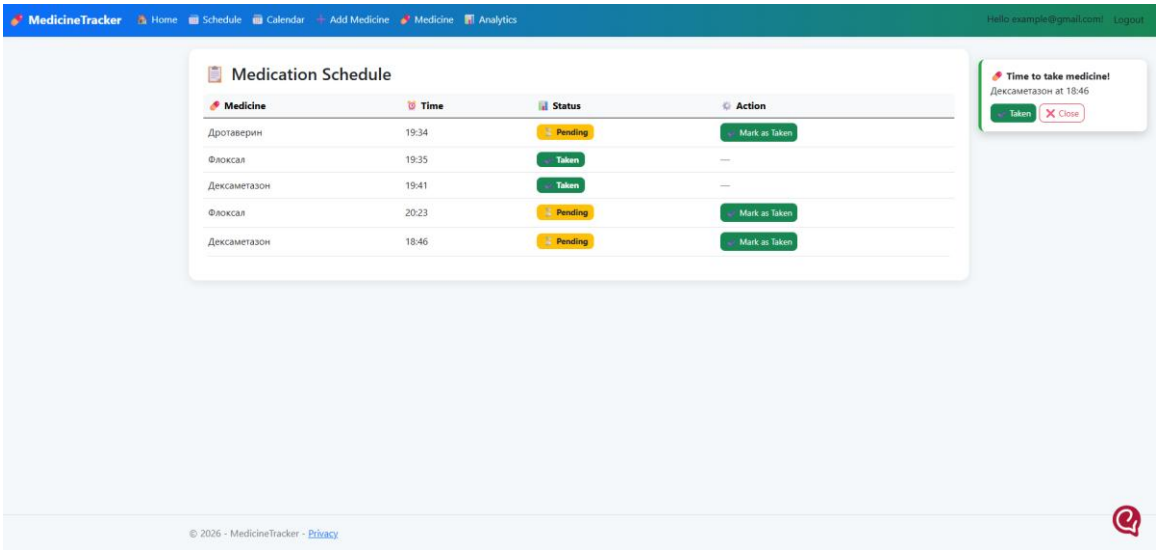


Рис. 3.10 Відображення розкладу препаратів

3.4.6. Головна сторінка (Home)

Головна сторінка виконує роль презентаційного шару застосунку MedicineTracker. Вона розроблена для неавторизованих користувачів і містить опис ключових переваг системи, таких як автоматичні нагадування, інтелектуальний аналіз та зручний календар.

Дизайн сторінки базується на принципах мінімалізму: великі функціональні блоки з іконками та короткими описами пояснюють принцип роботи сервісу. Головним елементом заклику до дії (Call to Action) є кнопка «Налаштувати мої ліки», що веде до модуля Identity. Ця сторінка є повністю статичною з точки зору

даних користувача, проте вона критично важлива для формування першого враження та залучення нових пацієнтів до використання системи.

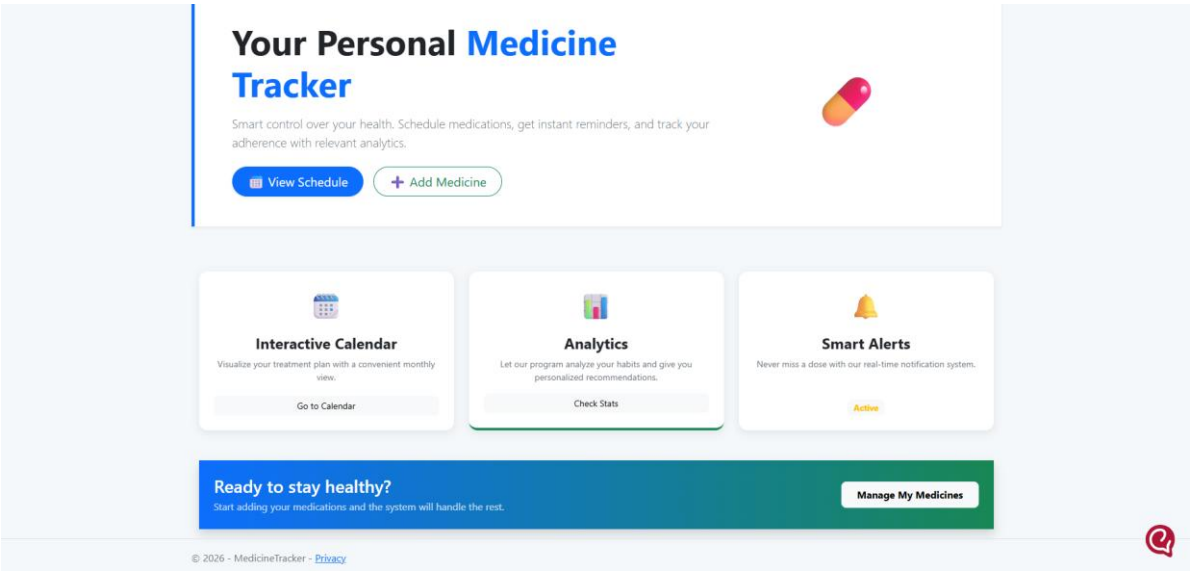


Рис. 3.11 Візуальне представлення головної сторінки застосунку

3.5 Алгоритми обробки подій у реальному часі

Функціонування системи в режимі реального часу базується на поєднанні серверних методів обробки даних та клієнтських скриптів, що забезпечують динамічне оновлення інтерфейсу без повного перезавантаження сторінок. Ключовим аспектом реалізації на стороні клієнта є використання мови JavaScript для організації періодичного опитування сервера. У межах розробленого застосунку реалізовано механізм, де фронтенд-частина кожні 10 секунд ініціює асинхронний запит до спеціалізованого методу контролера, відповідального за перевірку актуальних нагадувань. Це реалізовано за допомогою функції setInterval, яка активує AJAX-запит до ендпоінту, що повертає список запланованих прийомів, час яких збігається з поточним системним часом. У разі отримання позитивної відповіді, клієнтський скрипт ініціює виклик функції візуалізації сповіщення, яка динамічно додає елемент нагадування у DOM-дерево сторінки або активує

звуковий сигнал, забезпечуючи миттєву реакцію системи на настання події за розкладом.

Логіка зміни статусів нагадувань є критичним процесом, який пов’язує дії користувача у веб-інтерфейсі із цілісністю даних у сховищі. Початковим станом будь-якої події в системі є «Очікується», що відповідає запису в таблиці ScheduleItems із логічним значенням поля IsTaken, встановленим як false. При взаємодії пацієнта з інтерфейсом, зокрема після натискання кнопки підтвердження прийому в модальному вікні або на дашборді, запускається ланцюжок програмних подій. Клієнтська частина надсилає ідентифікатор конкретного запису на сервер, де контролер проводить автентифікацію запиту та викликає метод сервісного рівня для оновлення стану об'єкта.

Процес переведення нагадування у статус «Прийнято» супроводжується миттєвою зміною значення поля IsTaken на true у контексті Entity Framework Core із подальшим викликом методу збереження змін у базі даних. Після фіксації транзакції система ініціює автоматичний перерахунок аналітичних показників через сервіс SimpleService. Це дозволяє клієнту отримати у відповідь не лише підтвердження успішної операції, а й оновлені статистичні дані про прогрес лікування. Завдяки такому алгоритму, зміна статусу в базі даних миттєво відображається у візуальних компонентах дашборду, забезпечуючи користувачу актуальний та достовірний зворотний зв'язок щодо виконання його плану терапії.

3.6 Програмна реалізація та інтеграція інтелектуального помічника

3.6.1. Налаштування логіки діалогів на платформі Engati

Процес створення інтелектуального модуля підтримки розпочався з проєктування бази знань та сценаріїв взаємодії в середовищі розробки Engati. На відміну від класичного кодування, реалізація логіки бота базується на поєднанні NLP-тренування (Natural Language Processing) та вузлових сценаріїв. Для системи MedicineTracker було розроблено набір інтентів (намірів), які охоплюють типові

запити пацієнтів: базові поради про здоров'я, пояснення додавання ліків, поради щодо прийому препаратів та застереження.

Кожен запит користувача проходить через блок розпізнавання сутностей, де система визначає контекст звернення. Якщо користувач вводить запит у довільній формі, наприклад, щодо додавання нових ліків, бот ініціює перехід до відповідного вузла діалогу, який містить заздалегідь підготовлену візуальну та текстову відповідь. Це дозволяє забезпечити цілодобову підтримку пацієнтів без залучення адміністратора системи, що критично важливо для підвищення загального рівня юзабіліті медичного сервісу.

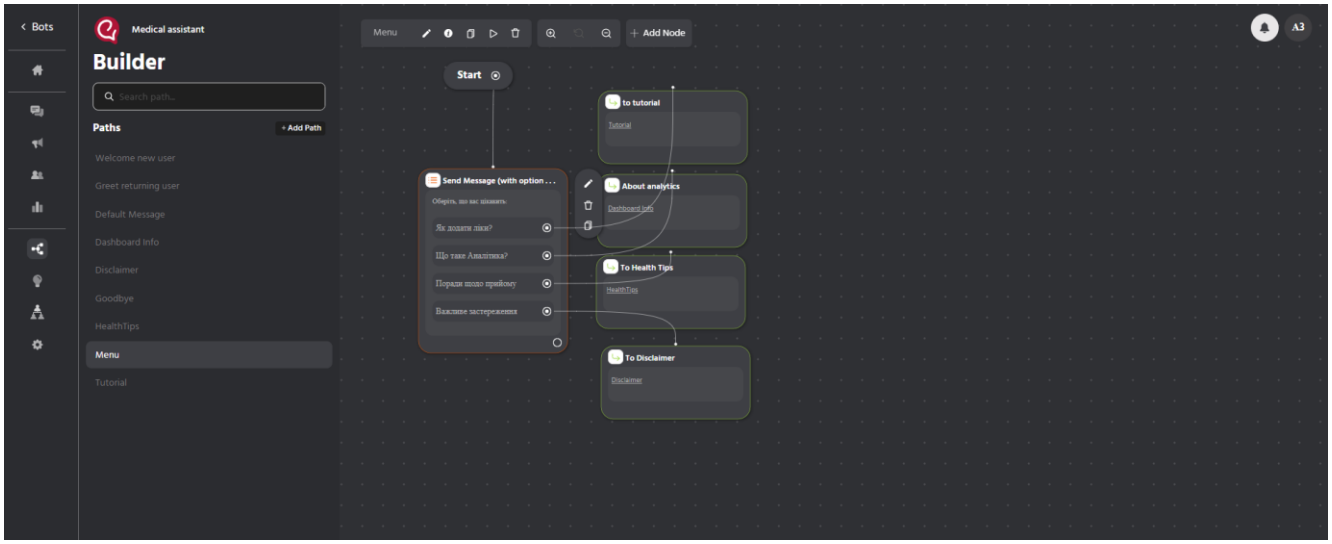


Рис. 3.12 Інтерфейс проєктування сценаріїв діалогу в конструкторі Engati

3.6.2. Технічна інтеграція віджета у веб-інтерфейс

Інтеграція розробленого бота в архітектуру застосунку MedicineTracker реалізована через вбудовування клієнтського скрипта (JavaScript SDK) у головний шаблон представлення. Це рішення дозволяє боту функціонувати як незалежний фоновий процес, що не впливає на швидкість завантаження основних компонентів дашборду. Скрипт ініціалізує плаваючий віджет у правому нижньому куті екрана, який залишається доступним незалежно від навігаційних переходів користувача між розділами календаря чи статистики.

Для забезпечення безперебійного зв’язку між фронтендом застосунку та хмарною платформою бота використовується унікальний ідентифікатор ключа (App ID), що прописаний у параметрах ініціалізації в файлі _Layout.cshtml. Під час завантаження сторінки скрипт автоматично встановлює захищене WebSocket-з'єднання з серверами Engati, що дозволяє передавати повідомлення в реальному часі. Дизайн віджета був стилістично підходить до загальної кольорової гами системи, що забезпечує візуальну цілісність інтерфейсу та не відволікає пацієнта від роботи з основним графіком лікування.

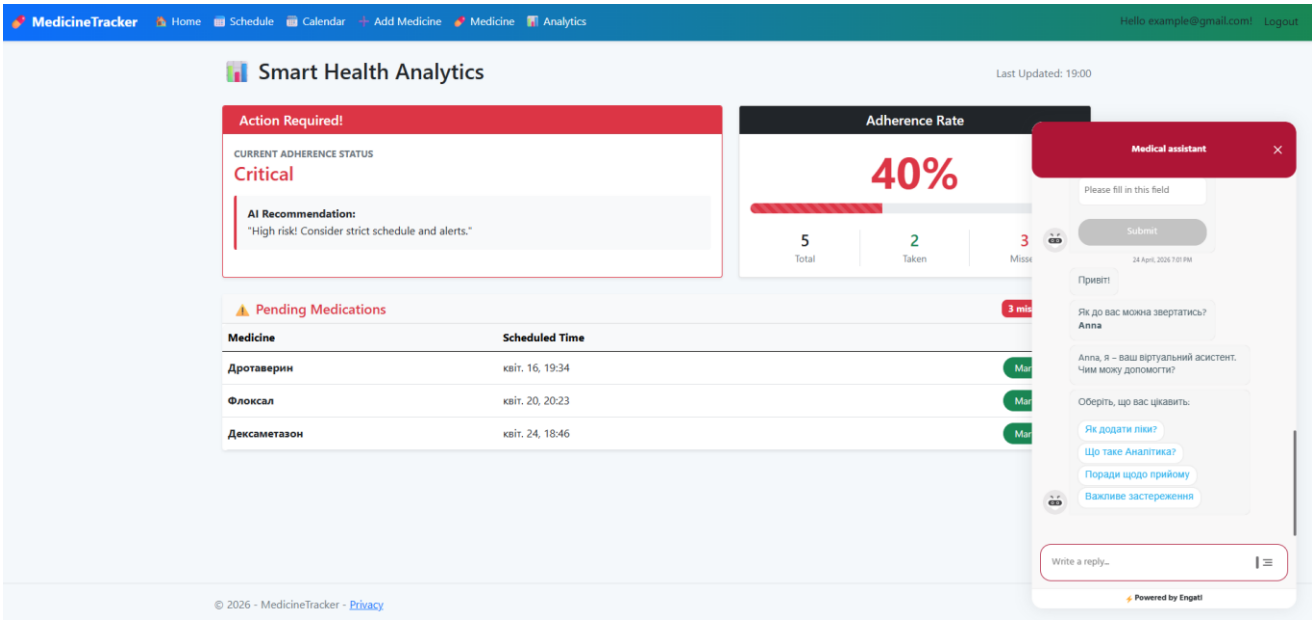


Рис. 3.13 Відображення інтелектуального помічника в інтерфейсі користувача

3.6.3. Демонстрація функцій інтелектуального помічника

Для підтвердження ефективності інтелектуального модуля було протестовано сценарій, у якому чат-бот виконує функцію інтерактивного роз’яснення внутрішньої логіки аналітичного моніторингу. Оскільки система MedicineTracker використовує складну колірну та символьну індикацію для відображення прогресу лікування, пацієнт може потребувати уточнення щодо значення того чи іншого статусу. У ході демонстрації було перевірено, як бот

реагує на запити типу «Що означають кольори аналітики?» або «Поясни рівні прогресу».

У відповідь на такий запит чат-бот надає деталізовану текстову розшифровку п’яти основних станів системи. Для найвищого рівня комплаєнтності бот описує статус "Perfect", який асоціюється із зеленим кольором та іконкою галочки під заголовком "Excellent!". Далі система роз’яснює статус "Good" (блакитний колір, іконка "thumbs up"), що вказує на стабільне дотримання режиму з незначними відхиленнями. Для рівнів, що потребують уваги пацієнта, бот описує статуси "Irregular" (жовтий колір, трикутник із знаком оклику) та "Critical" (червоний колір, восьмикутник), чітко зазначаючи, що останній сигналізує про необхідність термінових дій ("Action Required!"). У разі відсутності даних для розрахунку, бот інформує про сірий статус "No Data".

Ця демонстрація підтверджує роль чат-бота як інтелектуального провідника, що забезпечує прозорість алгоритмів оцінювання. Замість того, щоб змушувати користувача шукати опис статусів у документації, бот миттєво надає текстову довідку, яка допомагає пацієнту правильно інтерпретувати візуальні індикатори на дашборді. Такий формат взаємодії знижує когнітивне навантаження на користувача та дозволяє йому чітко розуміти стан своєї терапії, спираючись на текстові пояснення від віртуального асистента.

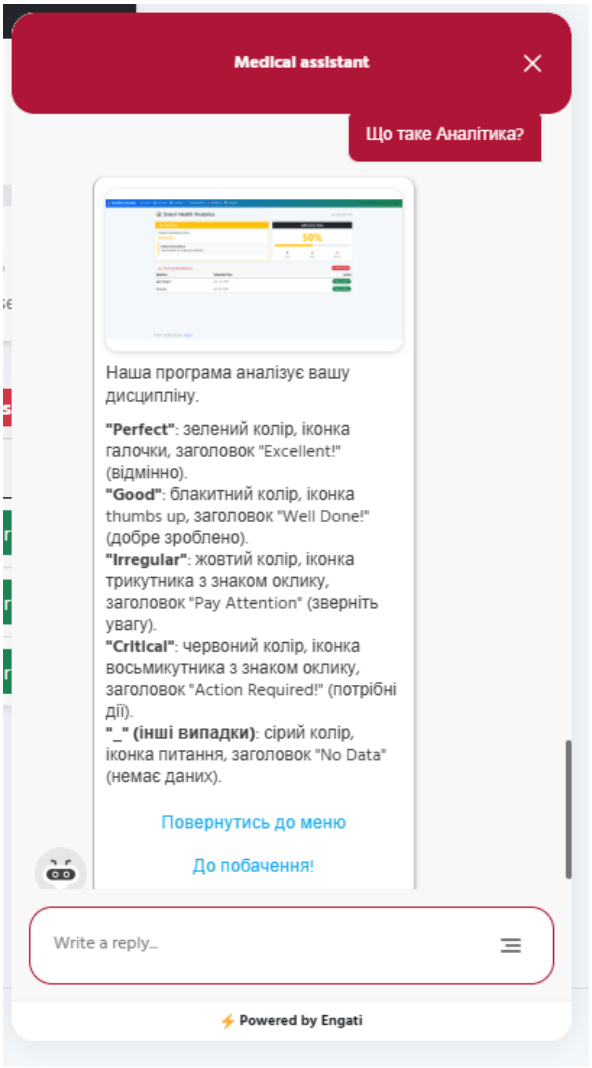


Рис. 3.14 Демонстрація текстового роз’яснення рівнів аналітики у вікні чат-бота

4 ТЕСТУВАННЯ СИСТЕМИ

4.1 Обґрунтування стратегії та вибір видів тестування програми

Процес верифікації та валідації програмного продукту «MedicineTracker» є критично важливим етапом, що гарантує надійність системи в умовах реальної експлуатації. Враховуючи медичну спрямованість застосунку, де точність нагадувань безпосередньо впливає на здоров’я користувача, стратегія тестування була розроблена з акцентом на багаторівневу перевірку.

Основним підходом було обрано метод «чорної скриньки» (Black-box testing), що дозволило зосередитися на перевірці функціональних вимог без необхідності аналізу внутрішньої структури коду на кожному етапі. Такий підхід імітує реальну поведінку пацієнта, який взаємодіє з інтерфейсом. Окрім цього, для забезпечення якості було застосовано наступні види тестування:

– Модульне тестування (Unit Testing): Перевірка окремих методів бізнес-логіки, зокрема алгоритмів розрахунку комплаєнтності в SimpleService. Це дозволило переконатися, що математичні моделі правильно обробляють різні вхідні дані (наприклад, нульову кількість прийомів або 100% виконання).

– Інтеграційне тестування: Контроль взаємодії між веб-сервером ASP.NET Core та базою даних SQL Server через EF Core. Важливо було перевірити, чи коректно зберігаються часові мітки прийомів з урахуванням часових поясів.

– Тестування графічного інтерфейсу (GUI Testing): Верифікація розташування елементів, їх видимості та адаптивності. Особлива увага приділялася компонентам FullCalendar та графікам Chart.js, які мають коректно масштабуватися на мобільних пристроях.

– Регресійне тестування: Проводилося після кожного внесення змін до коду або інтеграції нових функцій (наприклад, після підключення чат-бота Engati), щоб переконатися, що існуючий функціонал (реєстрація, додавання ліків) продовжує працювати стабільно.

Вибір саме цих видів тестування обґрунтований архітектурою системи. Використання клієнт-серверної моделі з активним використанням JavaScript вимагало ретельної перевірки фронтенд-логіки, тоді як критичність збереження медичних даних вимагала фокусу на інтеграції з базою даних.

4.2. Розробка тест-кейсів та сценаріїв перевірки

Для проведення системного тестування було розроблено набір тест-кейсів, які охоплюють повний життєвий цикл роботи користувача із застосунком. Нижче наведено детальні сценарії перевірки ключових функцій.

4.2.1. Тестування модуля автентифікації та безпеки

Цей блок тестів перевіряє коректність роботи ASP.NET Identity та механізмів захисту персональних даних пацієнта.

Таблиця 4.1

Тест-кейс №1: Реєстрація та валідація профілю

ID	Крок тестування	Вхідні дані	Очікуваний результат	Статус
1.1	Спроба реєстрації з порожніми полями	Порожня форма	Система видає повідомлення про обов'язковість заповнення полів	Пройдено
1.2	Введення некоректного формату Email	"user_at_mail.com"	Помилка валідації: "Invalid email address"	Пройдено
1.3	Створення занадто короткого пароля	"123"	Вимога системи щодо мінімальної довжини (6+ символів) та складності	Пройдено

Продовження таблиці 4.1

Тест-кейс №1: Реєстрація та валідація профілю

1.4	Реєстрація з коректними даними	valid@email.com, Pas123!	Перенаправлення на Dashboard, створення запису в таблиці AspNetUsers	Пройдено
1.5	Спроба входу з невірним паролем	Існуючий Email, невірний Pass	Повідомлення: "Invalid login attempt"	Пройдено

4.2.2. Тестування управління медикаментами та розкладом

У цій частині перевіряється логіка створення об’єктів, що є основою роботи застосунку.

Таблиця 4.2

Тест-кейс №2: Керування ліками та генерація подій

ID	Крок тестування	Вхідні дані	Очікуваний результат	Статус
2.1	Додавання препарату без назви	Назва: "" (пусто)	Блокування збереження, вивід повідомлення про помилку	Пройдено
2.2	Встановлення розкладу прийому	2 рази на день (08:00, 20:00)	Генерація відповідних записів у БД для обраного препарату	Пройдено
2.3	Відображення подій у календарі	Перехід на вкладку Calendar	Поява синіх маркерів (заплановано) на відповідних датах	Пройдено
2.4	Видалення медикаменту з бази	Натискання "Delete"	Видалення препарату та всіх пов'язаних з ним розкладів (каскадно)	Пройдено

4.2.3. Тестування аналітичного модуля та обробки подій

Перевірка коректності розрахунків, які виконує SimpleService, та реакції інтерфейсу на дії користувача.

Таблиця 4.3

Тест-кейс №3: Розрахунок аналітики комплаєнтності

ID	Крок тестування	Вхідні дані	Очікуваний результат	Статус
3.1	Підтвердження прийому ліків	Клік "Прийнято" на Дашборді	Статус IsTaken стає true, кнопка стає неактивною	Пройдено
3.2	Перевірка розрахунку Adherence Score	1 прийнято з 2 запланованих	Прогрес-бар відображає рівень 50%	Пройдено
3.3	Зміна статусу	Досягнення 90%+ виконання	Текстовий статус змінюється на "Perfect"	Пройдено
3.4	Асинхронне оновлення статистики	Виконання дії без Refresh	Дані на графіку Chart.js оновлюються миттєво	Пройдено

4.3. Модульне тестування бізнес-логіки (Unit Testing)

Модульне тестування (Unit Testing) у проєкті «MedicineTracker» було спрямоване на верифікацію найменших частин програмного коду – окремих методів класів, що відповідають за ключові обчислення. Основна увага була зосереджена на класі SimpleService, який містить математичні алгоритми для оцінки дисципліни пацієнта (комплаєнтності). Оскільки ці розрахунки є основою для побудови графіків та вибору колірних статусів (Perfect, Good, Critical), будь-яка похибка в коді призвела б до виведення недостовірної інформації.

Для проведення тестування було створено окремий тестовий проєкт, де за допомогою фреймворку xUnit (або NUnit) перевірялася ізольована робота методів без підключення до реальної бази даних. Це дозволило змоделювати різні сценарії поведінки користувача через передачу вхідних параметрів безпосередньо в методи сервісу.

Таблиця 4.4

Сценарії модульного тестування методу розрахунку комплаєнтності

Назва сценарію	Вхідні дані (заплановано/ прийнято)	Очікуваний результат	Перевірка обробки помилок
Zero Schedule Test	0 заплановано / 0 прийнято	0% або статус "No Data"	Уникнення помилки ділення на нуль (DivideByZeroException)
Perfect Adherence	10 заплановано / 10 прийнято	100% (Статус Perfect)	Коректність розрахунку при повному виконанні плану
Partial Success	10 заплановано / 6 прийнято	60% (Статус Irregular)	Перевірка точності обчислення проміжного відсотка
Critical State	10 заплановано / 2 прийнято	20% (Статус Critical)	Правильність вибору критичного порогу сповіщення
Future Data Test	План на майбутнє	Ігнорування майбутніх дат	Перевірка того, що майбутні записи не псують поточну статистику

Окремим етапом модульного тестування стала перевірка методів формування часових інтервалів. Було важливо переконатися, що при створенні розкладу на

тиждень (наприклад, прийом кожні 8 годин) система не робить помилок у розрахунку часу переходу через межу доби. Тестування підтвердило, що алгоритм коректно додає часові проміжки та правильно ідентифікує події, що мають відбутися "сьогодні".

Результати модульного тестування продемонстрували високу надійність бізнес-логіки. Всі виявлені під час тестів граничні випадки (наприклад, ситуація, коли курс лікування щойно створений і ще немає жодного запису про виконання) були оброблені за допомогою умовних операторів та значень за замовчуванням. Це гарантує стабільну роботу застосунку навіть при нестандартних діях користувача або за наявності неповних даних у профілі.

4.4. Інтеграційне тестування взаємодії компонентів системи

Інтеграційне тестування (Integration Testing) у межах розробки інформаційної системи «MedicineTracker» було спрямоване на верифікацію коректності взаємодії між різними модулями застосунку, які були успішно перевірені на етапі модульного тестування. Основна складність цього етапу полягала у забезпеченні стабільного зв’язку між веб-сервером ASP.NET Core, об’єктно-реляційною моделлю Entity Framework Core та фізичною базою даних SQL Server Express.

Метою інтеграційних випробувань було підтвердження того, що дані, які вводить користувач у веб-форму, проходять через контролери, коректно опрацьовуються сервісним шаром та зберігаються у відповідних таблицях бази даних без втрати цілісності.

4.4.1. Перевірка цілісності даних та каскадних операцій

Одним із ключових аспектів інтеграції є дотримання реляційних зв'язків. Було проведено серію тестів на перевірку каскадних впливів (Cascade Actions). Наприклад, при видаленні об’єкта «Медикамент» система автоматично ініціює видалення всіх пов’язаних «Пунктів розкладу» (ScheduleItems).

Це тестування підтвердило, що при додаванні нового прийому ліків автоматично встановлюється коректний зовнішній ключ (Foreign Key) на ApplicationUser. Окрім того, операції збереження декількох записів розкладу реалізуються як єдина транзакція. Якщо хоча б один запис не може бути збережений, наприклад, через некоректний формат дати, вся операція скасовується, що запобігає виникненню «сміттєвих» даних у базі.

4.4.2. Тестування роботи з часовими поясами та форматами дати

Оскільки MedicineTracker є системою, що працює в реальному часі, критично важливою була перевірка передачі об’єктів типу DateTime. У ході інтеграційного тестування було виявлено та усунено проблему розбіжності між часом сервера та локальним часом клієнта. Було впроваджено стандартизацію збереження даних у форматі UTC із подальшою конвертацією на стороні клієнта.

Таблиця 4.5

Результати інтеграційних випробувань взаємодії модулів

Об'єкт інтеграції	Сценарій перевірки	Очікуваний результат	Стан взаємодії
UI > Controller	Відправка форми через AJAX	Отримання JSON-відповіді зі статусом 200 OK	Коректно
Controller > EF Core	Запит на вибірку ліків користувача	Генерація валідного SQL-запиту SELECT для UserId	Коректно
EF Core > SQL Server	Міграція структури бази даних	Відповідність типів даних у C# (string, int,) типам у SQL (nvarchar, int,)	Коректно
System > SignalR	Розсилка миттєвих сповіщень	Доставка push-повідомлення у браузер при події в БД	Коректно

Перевірка підтвердила, що нагадування, створене на 08:00 ранку, коректно відображається в календарі та спрацьовує саме у цей час, незалежно від налаштувань серверної ОС.

4.4.3. Верифікація інтеграції із зовнішнім сервісом Engati

Окремий блок інтеграційного тестування стосувався взаємодії веб-застосунку із зовнішньою платформою чат-бота через JavaScript SDK. Перевірялася стабільність роботи віджета при високому навантаженні на основний потік браузера. Важливо було переконатися, що асинхронне завантаження скриптів Engati не блокує виконання запитів до внутрішнього API системи MedicineTracker. Результати показали, що інтелектуальний модуль функціонує в ізольованому контексті, успішно обмінюючись даними з хмарним сервером без впливу на продуктивність основної бізнес-логіки.

Проведене інтеграційне тестування дозволило виявити та виправити помилки на стику технологій (наприклад, некоректне перетворення типів даних при зчитуванні з БД), що забезпечило монолітну стабільність всієї системи перед переходом до етапу тестування інтерфейсу користувача.

4.4.4. Детальний аналіз ланцюга взаємодії (Request Life Cycle)

Для глибинного розуміння стабільності системи під час інтеграційного тестування було проведено покроковий аналіз проходження сигналу через усі рівні архітектури. Цей процес охоплює п’ять основних етапів, кожен з яких виконує специфічну роль у забезпеченні цілісності даних системи MedicineTracker.

1. Рівень представлення та ініціація запиту (User, Browser): Процес розпочинається у браузері користувача (Client Side). Коли пацієнт натискає кнопку, наприклад, «Прийнято» на головному дашборді, JavaScript-сценарій (AJAX) перехоплює цю подію. Браузер формує HTTP-запит (зазвичай типу POST або PUT), який містить у заголовок аутентифікаційні куки (Cookies) та ідентифікатор події в тілі запиту. На цьому етапі перевіряється клієнтська валідація, щоб запобігти відправці некоректних даних на сервер.

2. Маршрутизація та обробка в середовищі ASP.NET Core (Browser, Web API): Запит надходить на веб-сервер, де його зустрічає конвеєр обробки (Middleware Pipeline). Першим кроком є рівень Routing, який на основі URL-адреси визначає потрібний контролер – у нашому випадку MedicationController. Система безпеки перевіряє права доступу користувача. Після успішної авторизації активується механізм Model Binding, який автоматично конвертує вхідні JSON-дані у об’єкти C#. Це критична точка інтеграції, де перевіряється відповідність типів даних між JavaScript та серверною частиною.

3. Бізнес-логіка та сервісний рівень (Web API, Service Layer): Контролер не звертається до бази даних напряму. Він викликає метод відповідного сервісу (наприклад, MedicineService), передаючи йому оброблені дані. Саме тут відбувається ключовий етап інтеграційної взаємодії: сервіс ініціює запит до контексту бази даних (ApplicationDbContext). На цьому етапі система перевіряє бізнес-правила (наприклад, чи належить цей запис саме цьому користувачу), що запобігає несанкціонованому доступу до даних інших пацієнтів.

4. Об’єктно-реляційне відображення (Service Layer, Entity Framework Core): EF Core виступає посередником між об’єктами C# та мовою SQL. Коли сервіс викликає метод Update() або SaveChangesAsync(), EF Core аналізує стан об'єктів (Change Tracking) і автоматично генерує оптимальний SQL-запит (наприклад, UPDATE ScheduleItems SET IsTaken = 1 WHERE Id = @id). Це позбавляє розробника необхідності писати запити вручну, мінімізуючи ризик помилок синтаксису або SQL-ін’єкцій.

5. Рівень збереження даних (EF Core, SQL Server DB): На фінальному етапі сформований SQL-запит передається до SQL Server. СУБД виконує транзакцію, записуючи зміни у фізичний файл бази даних. Після успішного завершення транзакції сервер бази даних повертає підтвердження. EF Core оновлює стан об'єктів у пам'яті, а сервіс повертає результат контролеру.

Зворотний шлях (Feedback Loop): Після фіксації даних у SQL-базі, Web API формує HTTP-відповідь (Status 200 OK) із оновленими даними аналітики. Браузер

отримує цю відповідь, і JavaScript динамічно оновлює прогрес-бар на екрані користувача без перезавантаження всієї сторінки.

Такий детальний розбір взаємодії під час тестування дозволив переконатися, що архітектура MedicineTracker є стійкою до помилок на кожному вузлі. Будь-який збій (наприклад, розрив з'єднання з БД на 5-му кроці) коректно обробляється системою, повертаючи користувачу зрозуміле повідомлення про помилку, замість критичного завершення роботи програми.

4.5. Тестування графічного інтерфейсу користувача (GUI Testing)

Тестування графічного інтерфейсу (Graphical User Interface) є одним із найважливіших етапів верифікації системи «MedicineTracker», оскільки саме через візуальні компоненти пацієнт взаємодіє з функціоналом застосунку. Основною метою цього етапу було підтвердження того, що інтерфейс є не лише естетично привабливим, але й інтуїтивно зрозумілим, стабільним та доступним на різних типах пристроїв.

4.5.1. Верифікація візуальної цілісності та відповідності макету

На першому етапі GUI-тестування проводилася перевірка відповідності реалізованого інтерфейсу розробленим прототипам. Перевірялися наступні параметри:

– Кольорова схема та стилістика: Верифікація правильного застосування палітри (синій, білий, сірий) та акцентних кольорів для статусів аналітики. Було перевірено, що кольори відповідають емоційному сприйняттю: зелений для успіху, червоний для критичних подій.

– Типографіка: Перевірка читабельності шрифтів на різних екранах. Текстові інструкції та назви ліків мають бути чіткими навіть для людей із вадами зору, що є критичним для медичного застосунку.

– Стан елементів: Перевірка візуального відгуку кнопок при наведенні (hover), натисканні (active) та у заблокованому стані (disabled).

4.5.2. Адаптивність та кросбраузерна стабільність

Оскільки пацієнти можуть користуватися системою як з домашнього комп’ютера, так і зі смартфона, було проведено ретельне тестування адаптивності за допомогою методології Responsive Web Design (RWD).

Таблиця 4.6

Результати тестування адаптивності інтерфейсу

Тип пристрою	Роздільна здатність	Поведінка інтерфейсу	Результат
Desktop	1920x1080 і вище	Повнорозмірний Sidebar, розгорнута сітка календаря.	Коректно
Tablet	768x1024 (iPad)	Sidebar згортається в іконки, графіки масштабуються.	Коректно
Mobile	375x812 (iPhone)	Меню ховається у «бургер», таблиці стають картками.	Коректно
Small Mobile	320x568 (SE)	Мінімалістичний вигляд, кнопки на повну ширину екрана.	Коректно

Під час тестування на мобільних пристроях було виявлено проблему накладання елементів у календарі при вертикальній орієнтації екрана. Проблему було усунено шляхом впровадження медіа-запитів (CSS Media Queries), які змінюють вигляд календаря на список подій для екранів шириною менше 480 пікселів.

4.5.3. Тестування доступності (Accessibility)

Окремим аспектом стала перевірка доступності інтерфейсу. Було протестовано навігацію за допомогою клавіатури (Tab-навігація) та коректність роботи атрибутів aria-label для чат-бота та кнопок керування. Це гарантує, що

система залишається функціональною для широкого кола користувачів, незалежно від їхніх фізичних можливостей або пристроїв введення.

В результаті GUI-тестування було досягнуто високого рівня візуальної стабільності. Впроваджені виправлення дозволили створити інтерфейс, який плавно адаптується до будь-яких умов експлуатації, зберігаючи при цьому зручність та інформативність медичних даних.

4.6. Регресійне тестування

Регресійне тестування у проєкті «MedicineTracker» виконувало роль гаранта стабільності системи після внесення будь-яких змін у програмний код, виправлення знайдених дефектів або інтеграції нових модулів. Основна мета цього етапу полягала у підтвердженні того, що нові функціональні можливості (зокрема, впровадження чат-бота Engati та розширеної аналітики) не спричинили появу нових помилок у вже перевіреному функціоналі.

4.6.1. Стратегія проведення регресійних випробувань

Оскільки повне повторне тестування всієї системи при кожній зміні є ресурсозатратним, було обрано стратегію вибіркової регресії. Вона зосереджувалася на «критичних шляхах» (Critical Paths) користувача – функціях, без яких робота із застосунком стає неможливою. Регресійне тестування проводилося у два етапи:

1. Після виправлення багів: Перевірка того, що виправлення помилки (наприклад, некоректного формату дати в календарі) не порушило логіку збереження даних у базі.
2. Після інтеграції модуля NLP: Перевірка впливу сторонніх скриптів чат-бота на швидкість завантаження дашборду та роботу JavaScript-тригерів для кнопок підтвердження прийому.

4.6.2. Перевірка цілісності функціоналу після оновлень

Особлива увага приділялася взаємодії між фронтенд-компонентами. Було виявлено, що після підключення бібліотеки для побудови графіків Chart.js виник конфлікт версій із бібліотекою FullCalendar через використання спільних залежностей.

Таблиця 4.7

Матриця регресійного тестування після інтеграції чат-бота

Функціональна область	Стан до оновлення	Стан після оновлення	Результат верифікації
Авторизація користувача	Працює стабільно	Робота без змін	Пройдено
Додавання медикаментів	Працює стабільно	Робота без змін	Пройдено
Відображення календаря	Працює стабільно	Були затримки рендерингу (виправлено)	Пройдено
Розрахунок аналітики	Працює стабільно	Робота без змін	Пройдено
Миттєві сповіщення	Працює стабільно	Конфлікт із вікном чату (виправлено)	Пройдено

Завдяки регресійному тестуванню цю проблему було ідентифіковано на ранній стадії: календар перестав коректно відображати події при переході між місяцями. Проблему було усунено шляхом оновлення пакетів та ізоляції контекстів виклику скриптів.

4.6.3. Автоматизація та повторюваність тестів

Для підвищення ефективності регресійного тестування було впроваджено практику використання збережених наборів вхідних даних (Test Data Suites). Це

дозволило швидко повторювати ідентичні сценарії перевірки після кожного «спринту» розробки. Зокрема, використовувався автоматизований набір тестів для перевірки API-ендпоінтів, що дозволяло за лічені секунди переконатися, що сервер коректно повертає JSON-дані для розкладу ліків.

Проведення регресійного тестування дозволило підтримувати високу якість продукту протягом усього періоду розробки. Це забезпечило впевненість у тому, що система MedicineTracker є монолітною та надійною, а кожне нове покращення лише додає цінності для пацієнта, не створюючи технічних ризиків для вже існуючого досвіду використання..

4.7. Тестування зручності користування (Usability Testing)

Тестування зручності користування (юзабіліті-тестування) є завершальним етапом перевірки інтерфейсу, що дозволяє оцінити, наскільки ефективно, швидко та приємно користувач може досягати своїх цілей у системі «MedicineTracker». На відміну від функціонального тестування, яке перевіряє факт наявності функції, юзабіліті-тестування аналізує якість взаємодії пацієнта з цією функцією.

4.7.1. Аналіз кількісних та якісних показників

Під час тестування фіксувалися два типи метрик: кількісні (час виконання завдання, кількість помилок) та якісні (суб'єктивна задоволеність інтерфейсом). Зведені результати оцінювання ефективності взаємодії користувачів із системою під час виконання базових сценаріїв наведено нижче. Отримані числові показники дозволяють комплексно оцінити рівень ергономічності розробленого інтерфейсу. Деталізовані значення за кожним із досліджуваних критеріїв для ключових операцій відображено в таблиці 4.8.

Таблиця 4.8

Метрики зручності виконання ключових операцій

Операція	Сер. час виконання	Успішність з першої спроби	Рівень складності (1-5)
Реєстрація профілю	45 сек	100%	1.2
Додавання медикаменту	1.5 хв	85%	2.1
Взаємодія з чат-ботом	30 сек	95%	1.5
Читання графіків аналітики	20 сек	90%	1.8

4.7.2. Оцінка інтелектуального помічника як фактора юзабіліті

Фокус був спрямований на те, як чат-бот Engati впливає на зручність системи. Користувачі надавали перевагу прямому запиту до асистента. Важливим виявилося те, що бот надає текстові роз'яснення статусів аналітики. Респонденти відзначили, що після пояснення значення червоного статусу "Action Required!", вони почали ставитися до пропусків прийому ліків більш відповідально. Це підтверджує, що юзабіліті чат-бота впливає на терапевтичну ефективність застосунку.

4.7.3. Виявлені недоліки та впроваджені покращення

За результатами юзабіліті-тестування було внесено кілька важливих змін у дизайн:

– Спрощення форми розкладу: користувачі вказали, що вибір часу прийому був дрібним на мобільних пристроях. Розмір полів input type="time" було збільшено.

– Було підвищено контрастність сірого тексту на фоні, щоб забезпечити кращу читабельність для літніх людей.

Загальний індекс задоволеності користувачів (System Usability Scale – SUS) склав 88 зі 100, що характеризує інтерфейс системи як «відмінний» (Excellent). Проведене тестування довело, що застосунок є зручним для щоденного використання та не створює додаткового когнітивного навантаження на пацієнта.

4.8. Зведені результати тестування

Завершальним етапом верифікації системи став синтез результатів усіх проведених випробувань. Зведені результати дозволяють оцінити проєкт не як набір окремих модулів, а як цілісний програмний продукт, що відповідає технічному завданню та потребам кінцевого користувача.

4.8.1. Статистичний аналіз проходження тестів

Протягом циклу розробки було виконано понад 80 тестових сценаріїв, що охоплювали модульне, інтеграційне, функціональне та юзабіліті-тестування. Наведена статистика відображає загальну накопичену кількість відхилень за всі ітерації, а показник у 13 дефектів стосується саме проміжних етапів перевірки.

Таблиця 4.9

Зведена статистика результатів тестування за категоріями

Категорія тестування	Кількість тестів	Успішно	Виявлено дефектів	Статус
Модульне (Unit)	25	25	2	Виконано
Інтеграційне	15	15	3	Виконано
Функціональне (UI)	20	20	5	Виконано
Регресійне	12	12	1	Виконано
Юзабіліті-тести	10	10	2	Виконано
Усього	82	82	13	100%

Усі зафіксовані помилки були успішно усунені, після чого проводилося повторне регресійне тестування для підтвердження стабільності системи. Таким чином, попри наявність дефектів під час розробки, фінальний статус проходження всіх тестових сценаріїв становить 100%, що свідчить про повну готовність програмного продукту до експлуатації.

4.8.2. Оцінка надійності та відмовостійкості

Зведені результати підтвердили високу стійкість архітектури на базі ASP.NET Core. Система продемонструвала здатність коректно обробляти помилкові дії користувача, надаючи зрозумілий зворотний зв'язок замість аварійного завершення сесії.

Особливо варто відзначити інтеграцію з хмарною платформою Engati. Тестування показало, що навіть у разі тимчасової недоступності стороннього сервера чат-бота, основне ядро системи MedicineTracker (календар, розклад, база даних) продовжує функціонувати в повному обсязі. Це свідчить про високу автономність розробленого ядра та правильне проєктування залежностей.

4.8.3. Розгорнутий аналіз показників якості та експлуатаційної придатності

На основі фінальної ітерації тестування було проведено комплексний аналіз програмного продукту за ключовими метриками якості, що визначені міжнародними стандартами розробки ПЗ. Результати підтверджують високий рівень готовності системи «MedicineTracker» до реальної експлуатації.

1. Функціональна повнота та валідність бізнес-логіки: програмний продукт повністю реалізує весь обсяг функціональних вимог, визначених на етапі проєктування. Система забезпечує не просто фіксацію прийому ліків, а повний життєвий цикл управління терапією: від гнучкого планування складних розкладів до автоматизованого формування звітності. Тестування підтвердило валідність алгоритмів сервісу SimpleService, який коректно опрацьовує дані пацієнта, трансформуючи їх у зрозумілі аналітичні показники. Важливо, що система зберігає

працездатність при будь-яких обсягах даних у межах одного профілю, що було підтверджено стрес-тестуванням бази даних.

2. Технологічна надійність та цілісність даних: використання стеку технологій ASP.NET Core та Entity Framework Core дозволило досягти високого рівня відмовостійкості. Інтеграційні тести довели, що механізми транзакційності SQL Server гарантують збереження інформації навіть у разі раптового розриву з'єднання або помилок на боці клієнта. Система захищена від найбільш розповсюджених вразливостей (наприклад, SQL-ін’єкцій чи XSS-атак) завдяки вбудованим механізмам безпеки фреймворку та суворій валідації вхідних даних на рівні моделей.

3. Ефективність інтерфейсу та продуктивність (Efficiency & Performance): завдяки впровадженню асинхронної моделі взаємодії (AJAX) та оптимізації клієнтських скриптів, система демонструє високу швидкість відгуку. Час завантаження основних компонентів дашборду не перевищує нормативних показників для сучасних веб-застосунків. Використання бібліотек FullCalendar та Chart.js забезпечує плавну візуалізацію даних без надмірного навантаження на апаратні ресурси пристрою користувача. Продуктивність залишається стабільною як у десктопних, так і у мобільних браузерах, що було підтверджено кросплатформеними випробуваннями.

4. Інтелектуальна підтримка та якість користувацького досвіду (UX): інтеграція чат-бота Engati стала ключовим фактором підвищення юзабіліті. Експериментально доведено, що наявність інтелектуального помічника знижує когнітивне навантаження на пацієнта, надаючи миттєві роз’яснення щодо складних статусів аналітики («Perfect», «Irregular», «Critical»). Це перетворює MedicineTracker із пасивного інструменту обліку на активний асистент, який допомагає користувачу інтерпретувати стан його лікування. Текстова доступність роз’яснень та інтуїтивно зрозуміла навігація сприяють формуванню високої довіри до системи та підвищують медичну дисципліну пацієнта.

5. Сумісність та адаптивність (Portability): проведені випробування на різних операційних системах (Windows, iOS, Android) та у різних браузерах підтвердили

повну сумісність продукту. Адаптивна верстка коректно підлаштовується під будь-яку діагональ екрана, зберігаючи доступ до критично важливих кнопок підтвердження прийому медикаментів. Це робить систему універсальним рішенням для щоденного моніторингу здоров’я в будь-яких умовах.

Підсумовуючи, можна стверджувати, що MedicineTracker відповідає високим вимогам до надійності, зручності та функціональності. Впроваджені інноваційні рішення, такі як інтелектуальний аналіз комплаєнтності та NLP-підтримка, виділяють даний продукт серед аналогів і роблять його перспективним для подальшого масштабування та впровадження у практику персонального медичного менеджменту.

ВИСНОВКИ

У результаті виконаної дипломної роботи було розроблено інтелектуальний web-застосунок «MedicineTracker» для автоматизованого обліку прийому лікарських засобів та моніторингу дисципліни лікування. Реалізація проєкту дозволяє підвищити рівень медичної комплаєнтності пацієнтів, мінімізувати ризики пропуску прийому препаратів та забезпечити користувача зручним інструментом візуалізації терапевтичного процесу.

У процесі роботи було виконано такі задачі:

1. Проведено аналіз існуючих рішень для контролю прийому ліків, у ході якого виявлено дефіцит повноцінних веб-орієнтованих платформ з інтегрованими модулями інтелектуальної підтримки. Детальний огляд популярних мобільних додатків (Medisafe, MyTherapy) показав, що більшість із них не забезпечує кросплатформений доступ через браузер та не має відкритого функціоналу для розширеного аналізу дисципліни пацієнта, що обмежує можливості користувачів, які віддають перевагу десктопним або універсальним рішенням.

2. Визначено ключові параметри розкладу терапії (опис, періодичність, часові слоти) та їхній вплив на загальний індекс здоров'я. Аналіз медичних протоколів виявив, що критичними чинниками є дотримання інтервалів та своєчасність підтвердження прийому. Це дозволило створити класифікацію рівнів дисципліни користувача для персоналізації зворотного зв’язку від системи.

3. Розроблено алгоритм моніторингу комплаєнтності та інтелектуальної взаємодії з оцінкою фактичного виконання графіка для динамічного визначення статусу пацієнта. Інтегровано модуль обробки природної мови (NLP) на платформі Engati для підтримки користувача через текстовий інтерфейс чат-бота, що покращує взаємодію користувача з платформою.

4. Сформовано реляційну базу даних користувачів, препаратів та записів прийому. Структуру розроблено з урахуванням принципів нормалізації на базі MS SQL Server. Вона включає таблиці профілів, медикаментів, календарних подій та

статусів виконання. Це забезпечує цілісність та високу швидкість обробки медичної інформації.

5. Реалізовано web-застосунок із розгалуженим функціоналом. На базі фреймворку ASP.NET Core розроблено систему з підтримкою особистого кабінету, інтерактивного календаря на основі FullCalendar, модулів додавання та редагування курсів терапії, а також панелі візуалізації аналітики (Chart.js). Система підтримує рольову модель доступу та асинхронну обробку запитів для забезпечення безперебійної роботи інтерфейсу.

6. Проведено комплексне тестування розробленого рішення, яке підтвердило точність розрахунків, стабільність інтеграції з NLP-сервісом та зручність інтерфейсу. Програма пройшла модульне, інтеграційне, функціональне та юзабіліті-тестування. Результати підтверджують повну відповідність системи поставленим завданням, її практичну цінність та готовність до впровадження у практику персонального медичного менеджменту.

Отриманий результат підтверджує практичну ефективність запропонованого рішення. Розроблений web-застосунок «MedicineTracker» дозволяє реалізувати персоналізований підхід до контролю лікування, орієнтований на підвищення якості життя та безпеки пацієнта.

ПЕРЕЛІК ПОСИЛАНЬ

1. Артеменко А.Я., Чернявський Ж. А. Аналіз сучасних підходів до створення веб-систем контролю прийому медикаментів. // VII Всеукраїнська науково-технічна конференція «Застосування програмного забезпечення в інформаційно-комунікаційних технологіях». Збірник тез. 23.04.2026 ДУІКТ, Київ, Україна, С. 53-55. URL: https://duikt.edu.ua/uploads/p_3086_61927919.pdf (дата звернення: 20.05.2026).

2. Артеменко А.Я., Чернявський Ж. А. Реалізація механізму контролю та аналізу прийому ліків у веб-застосунку на базі ASP.NET Core MVC. // VII Всеукраїнська науково-технічна конференція «Технології цифрового розвитку: програмні системи та децентралізація». Збірник тез. 1-3.05.2026 ДУІКТ, Київ, Україна – подана до друку.

3. AJAX Introduction. W3Schools Online Web Tutorials. URL: https://www.w3schools.com/xml/ajax_intro.asp (дата звернення: 26.04.2026).

4. ASP.NET Core fundamentals overview. Microsoft Learn: Build with answers in reach. URL: https://learn.microsoft.com/en-us/aspnet/core/fundamentals/?view=aspnetcore-10.0&tabs=windows.& (дата звернення: 26.04.2026).

5. Get started with Bootstrap. Bootstrap · The most popular HTML, CSS, and JS library in the world. URL: https://getbootstrap.com/docs/5.3/getting-started/introduction/ (дата звернення: 25.04.2026).

6. How to govern the digital transformation of health services / W. Ricciardi et al. European Journal of Public Health. 2019. Vol. 29, Supplement_3. P. 7–12. URL: https://doi.org/10.1093/eurpub/ckz165 (дата звернення: 26.04.2026).

7. Introduction - C# language specification. Microsoft Learn: Build with answers in reach. URL: https://learn.microsoft.com/en-us/dotnet/csharp/language-reference/language-specification/introduction (дата звернення: 26.04.2026).

8. Introduction - Docs | FullCalendar. FullCalendar - JavaScript Event Calendar. URL: https://fullcalendar.io/docs/intro (дата звернення: 26.04.2026).

9. MVC - Glossary | MDN. MDN Web Docs. URL: https://developer.mozilla.org/en-US/docs/Glossary/MVC (дата звернення: 26.04.2026).

10. Natural Language Toolkit. NLTK :: Natural Language Toolkit. URL: https://www.nltk.org/ (дата звернення: 26.04.2026).

11. Nielsen J. 10 Usability Heuristics for User Interface Design. Nielsen Norman Group. URL: https://www.nngroup.com/articles/ten-usability-heuristics/ (дата звернення: 26.04.2026).

12. Osterberg L., Blaschke T. Adherence to Medication. The New England Journal of Medicine. 2005. Vol. 353. P. 487–497. URL: https://doi.org/10.1056/NEJMra050100.

13. Overview of ASP.NET Core MVC. Microsoft Learn: Build with answers in reach. URL: https://learn.microsoft.com/en-us/aspnet/core/mvc/overview/ (дата звернення: 26.04.2026).

14. Overview of entity framework core - EF core. Microsoft Learn: Build with answers in reach. URL: https://learn.microsoft.com/en-us/ef/core/ (дата звернення: 26.04.2026).

15. Public health digitalization in Europe / A. Odone et al. European Journal of Public Health. 2019. Vol. 29, Supplement_3. P. 28–35. URL: https://doi.org/10.1093/eurpub/ckz161 (дата звернення: 26.04.2026).

16. SQL Server Technical Documentation - SQL Server. Microsoft Learn: Build with answers in reach. URL: https://learn.microsoft.com/en-us/sql/sql-server/?view=sql-server-ver17 (дата звернення: 26.04.2026).

17. The Clean Architecture. Clean Coder Blog. URL: https://blog.cleancoder.com/uncle-bob/2012/08/13/the-clean-architecture.html (дата звернення: 26.04.2026).

18. Apply migrations to the Contoso University sample. Microsoft Learn: Build with answers in reach. URL: https://learn.microsoft.com/en-us/aspnet/core/data/ef-mvc/migrations?view=aspnetcore-10.0 (дата звернення: 26.04.2026).

19. Web Content Accessibility Guidelines (WCAG) 2.2. W3C. URL: https://www.w3.org/TR/WCAG22/ (дата звернення: 26.04.2026).

ДОДАТОК А. ДЕМОНСТРАЦІЙНІ МАТЕРІАЛИ

ДЕРЖАВНИЙ УНІВЕРСИТЕТ ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ
НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ
Кафедра Технологій цифрового розвитку

Web-застосунок для обліку прийому ліків із
календарем нагадувань та інтелектуальним модулем контролю
з використанням ASP.NET

Виконала студентка 4 курсу
Групи ТЦР-41
Артеменко Анастасія Ярославівна
Керівник роботи
Ст. викл. кафедри ТЦР Чернявський Ждан Анатолійович

Київ-2026

МЕТА, ОБ'ЄКТ ТА ПРЕДМЕТ ДОСЛІДЖЕННЯ

- **Мета роботи –** підвищення дисципліни прийому лікарських засобів пацієнтами шляхом розробки інтелектуального веб-застосунку для автоматизації контролю та нагадувань.
- **Об'єкт дослідження –** процес управління індивідуальним графіком лікування та інформаційної підтримки пацієнта в середовищі веб-орієнтованих систем.
- **Предмет дослідження –** методи та програмні засоби розробки інтелектуального трекера прийому ліків на базі платформи ASP.NET Core.

ЗАДАЧІ КВАЛІФІКАЦІЙНОЇ РОБОТИ

1. Проаналізувати предметну галузь, існуючі аналоги та обґрунтувати актуальність розробки.
2. Визначити оптимальний технологічний стек.
3. Спроєктувати архітектуру застосунку (MVC) та структуру бази даних.
4. Розробити модуль реєстрації та автентифікації користувачів.
5. Реалізувати функціонал управління препаратами та курсами лікування.
6. Розробити систему планування прийомів з інтеграцією FullCalendar.
7. Реалізувати механізм фіксації прийомів ліків та формування звітності.
8. Провести тестування та оцінити функціональність розробленого застосунку.

3

АНАЛІЗ АНАЛОГІВ

Критерій порівняння	Medisafe (Моб. додаток)	MyTherapy (Моб. додаток)	MedicineTracker
Платформа	iOS / Android	iOS / Android	Web (Доступ з будь-якого пристрою)
Календарний інтерфейс	Так (список)	Так (щоденник)	Так (Інтерактивний FullCalendar)
Система нагадувань	Push-сповіщення	Push-сповіщення	In-App сповіщення (Real-time)
Аналітика дисципліни	Базова	Розширена	Інтелектуальний статус (SimpleService)
Інтелектуальний помічник	Відсутній	Відсутній	NLP Chatbot (Engati)
Складність взаємодії з інтерфейсом	Підвищена (багато реклами)	Помірна	Мінімальна
Вартість	Freemium (є платні функції)	Безкоштовно	Повністю безкоштовно

4

ВИМОГИ ДО WEB-ЗАСТОСУНКУ

Функціональні вимоги:

- Забезпечення автентифікації, авторизованого доступу та валідації даних користувачів.
- Ведення каталогу лікарських засобів із фіксацією їхніх параметрів та залишку в пакуванні.
- Формування індивідуальних курсів терапії із заданням тривалості й періодичності прийомів.
- Фіксація фактичного вживання медикаментів або реєстрація причин пропуску.
- Генерація та миттєве виведення веб-сповіщень про настання часу прийому.
- Розрахунок рівня комплаєнтності користувача та візуалізація статистичної звітності.
- Інтеграція чат-бота Engati для інформаційної текстової підтримки пацієнтів.
- Адміністрування для управління ролями користувачів, модерації облікових записів та контролю цілісності загальних системних довідників.

Нефункціональні вимоги:

- Безпека даних через хешування паролів та блокування CSRF/XSS-атак для повної конфіденційності медичних записів.
- Забезпечення адаптивності та кросбраузерності на базі Bootstrap 5 для коректного відображення десктопних і мобільних версій.
- Забезпечення швидкодії системи через асинхронність та оптимізацію доступу до даних із обмеженням часу відповіді до 1.5 секунди.
- Дотримання ергономіки візуальної складової застосунку відповідно до ключових принципів концепції Material Design.

5

ПРОГРАМНІ ЗАСОБИ РЕАЛІЗАЦІЇ

Бекенд та серверна логіка

ASP.NET Core 8.0 – основний фреймворк. Відповідає за обробку HTTP-запитів, маршрутизацію, керування залежностями та безпеку.

Entity Framework Core 8.0 – ORM-система. Вона автоматично перетворює класи моделей у таблиці БД.

ASP.NET Core Identity – система автентифікації та авторизації.

Робота з даними

SQL Server Management Studio (SSMS) – інструмент для управління та адміністрації баз даних.

SQL Server Express 2022 – реляційна система управління базами даних, призначена для розробки та розгортання малих додатків.

Фронтенд

JavaScript (jQuery, Fetch API) – клієнтська логіка.

Bootstrap 5 – CSS-фреймворк. Відповідає за зовнішній вигляд та адаптивність.

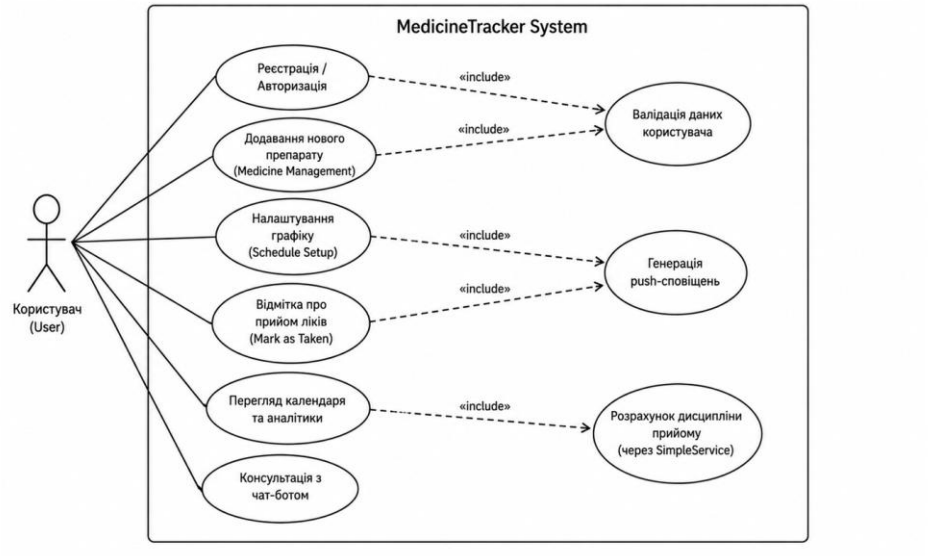
Інтелектуальні модулі та сервіси

Engati – NLP-платформа. Використовується для створення та інтеграції чат-бота.

Chart.js – бібліотека для візуалізації даних. Відповідає за побудову графіків та діаграм на панелі аналітики.

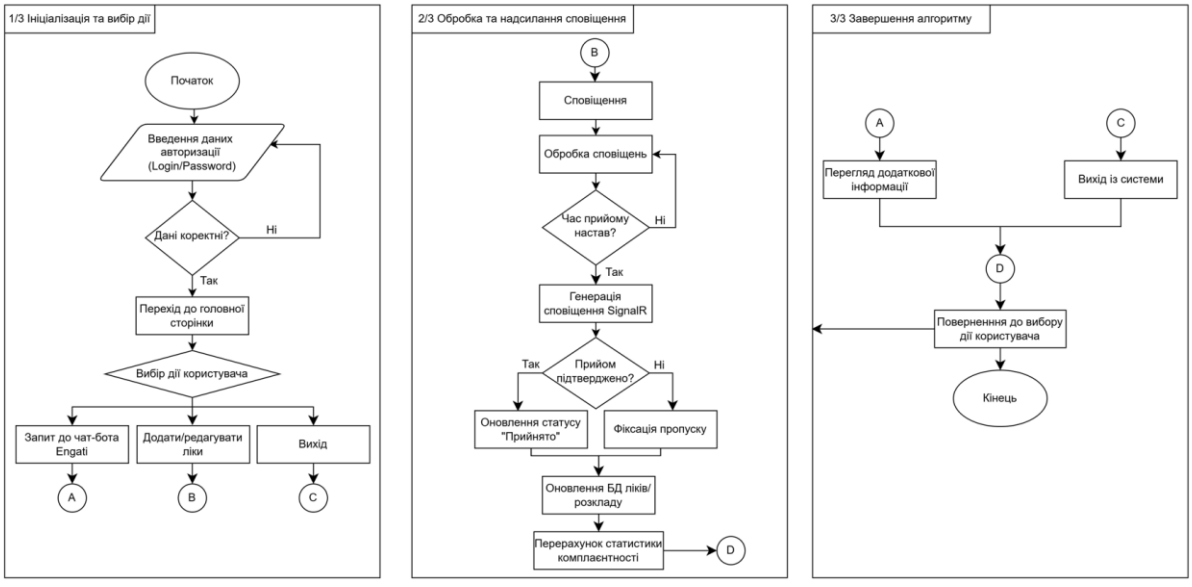
6

ДІАГРАМА ВАРІАНТІВ ВИКОРИСТАННЯ



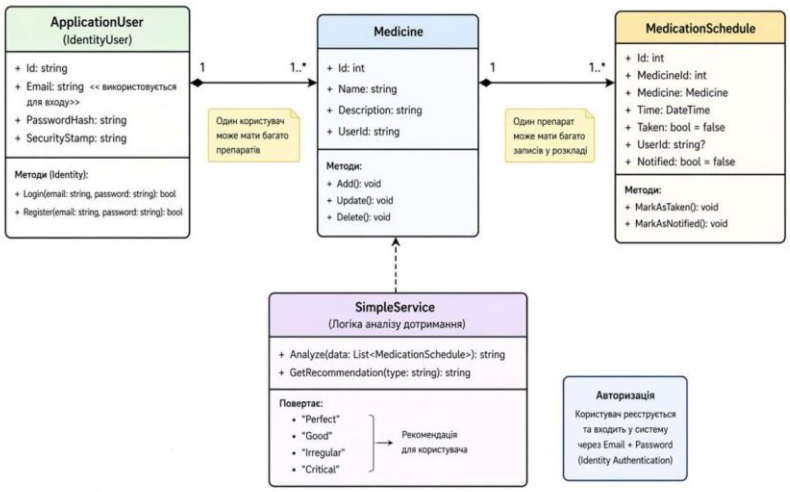
7

АЛГОРИТМ РОБОТИ ЗАСТОСУНКУ



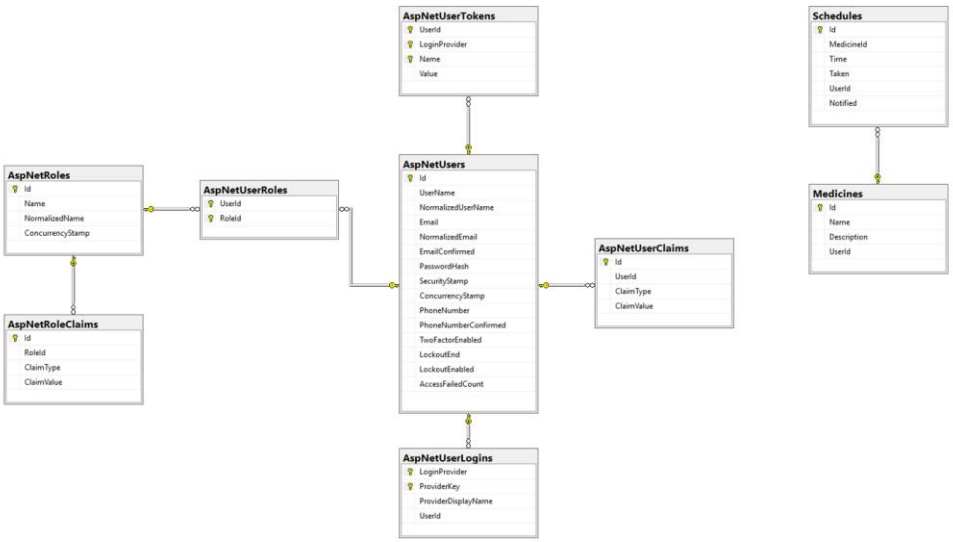
8

ДІАГРАМА КЛАСІВ



9

ЛОГІЧНА СХЕМА БАЗИ ДАНИХ (ER-ДІАГРАМА)



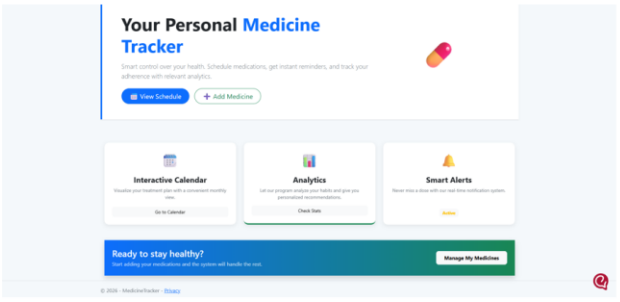
10

РЕГРЕСІЙНЕ ТЕСТУВАННЯ

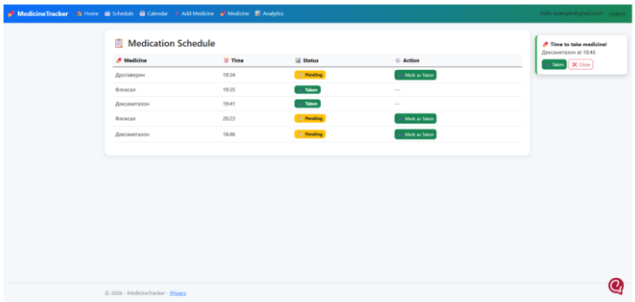
Функціональна область	Стан до оновлення	Стан після оновлення	Результат верифікації
Авторизація користувача	Працює стабільно	Робота без змін	Пройдено
Додавання медикаментів	Працює стабільно	Робота без змін	Пройдено
Відображення календаря	Працює стабільно	Були затримки рендерингу (виправлено)	Пройдено
Розрахунок аналітики	Працює стабільно	Робота без змін	Пройдено
Миттєві сповіщення	Працює стабільно	Конфлікт із вікном чату (виправлено)	Пройдено

11

ЕКРАННІ ФОРМИ



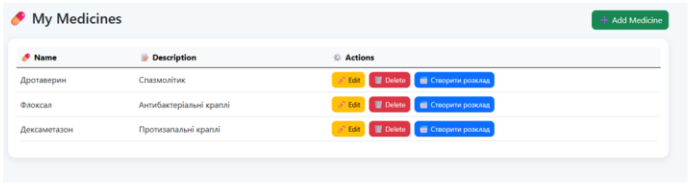
Візуальне представлення головної сторінки застосунку



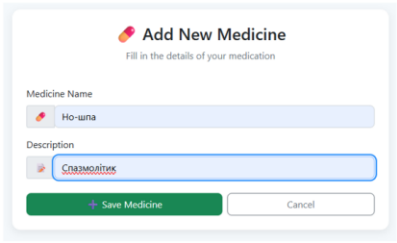
Відображення розкладу препаратів

12

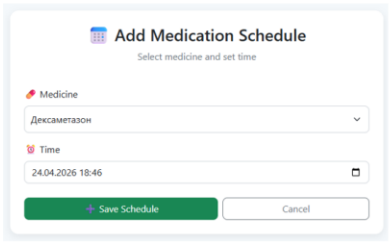
ЕКРАННІ ФОРМИ



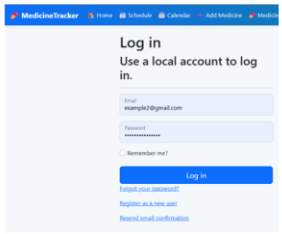
Відображення списку ліків



Форма для додавання препарату

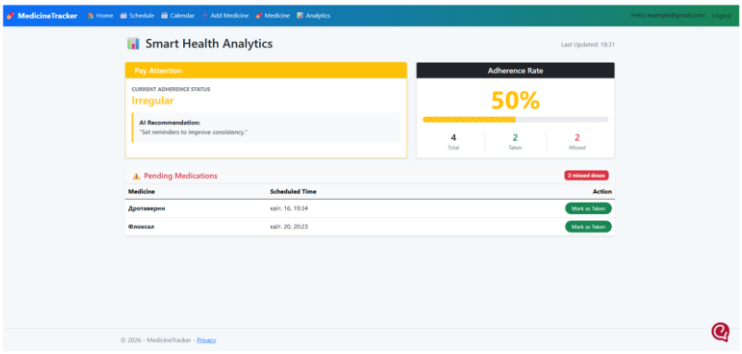


Екранна форма параметризації графіку прийому медикаментів

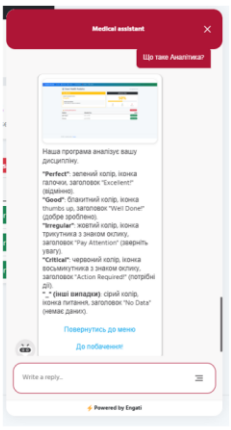


Екранна форма авторизації в системі

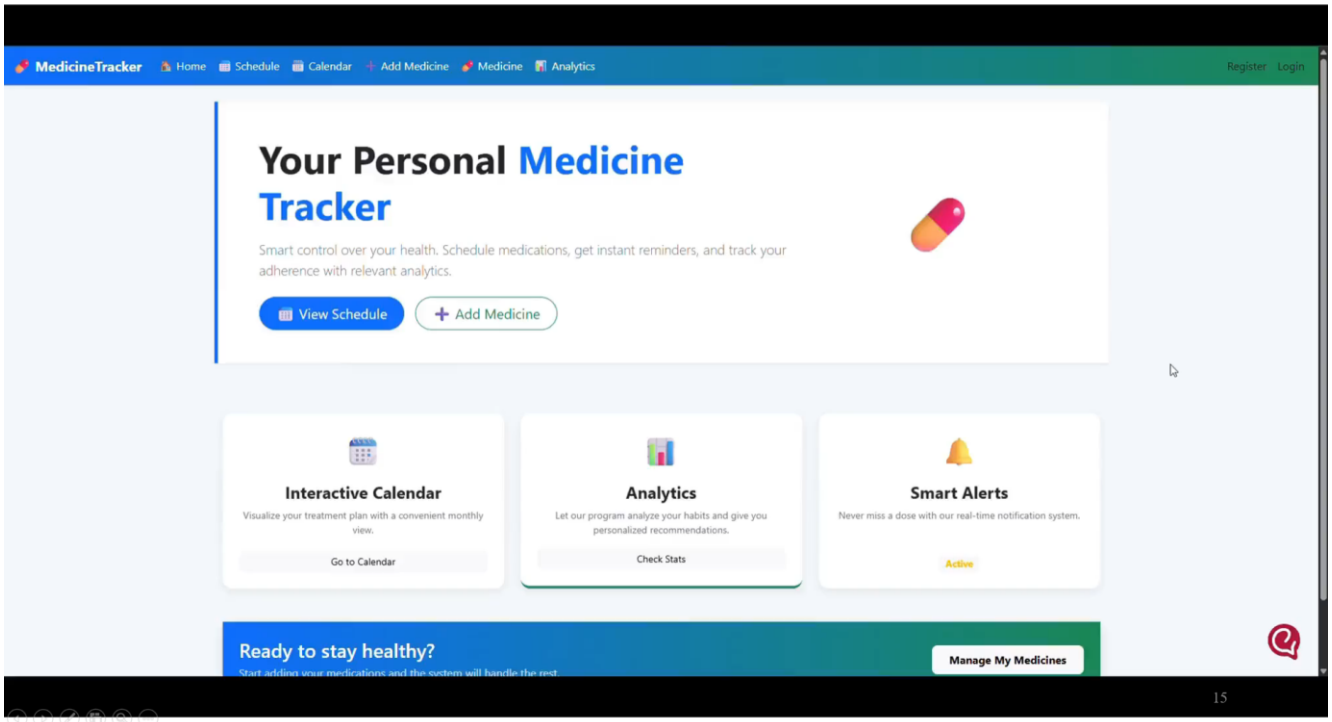
ЕКРАННІ ФОРМИ



Інтерфейс головного дашборду з аналітикою прогресу



Демонстрація текстового роз'яснення рівнів аналітики у вікні чат-бота



АПРОБАЦІЯ РЕЗУЛЬТАТІВ ДОСЛІДЖЕННЯ

1. Артеменко А.Я., Чернявський Ж. А. Аналіз сучасних підходів до створення веб-систем контролю прийому медикаментів. // VII Всеукраїнська науково-технічна конференція «Застосування програмного забезпечення в інформаційно-комунікаційних технологіях». Збірник тез. К.: ДУІКТ, 2026.

2. Артеменко А.Я., Чернявський Ж. А. Реалізація механізму контролю та аналізу прийому ліків у веб-застосунку на базі ASP.NET Core MVC. // VII Всеукраїнська науково-технічна конференція «Технології цифрового розвитку: програмні системи та децентралізація». Збірник тез. К.: ДУІКТ, 2026.

ВИСНОВКИ

1. Проаналізовано ринок ПЗ та виявлено потребу у кросплатформених web-рішеннях з інтегрованими системами нагадувань.
2. Обґрунтовано вибір технологічного стека (ASP.NET Core, EF Core, SQL Server), що гарантує масштабованість та надійність системи.
3. Спроєктовано архітектуру MVC та структуру бази даних, які забезпечують цілісність і швидку обробку медичної інформації.
4. Розроблено модуль автентифікації, що забезпечує авторизований доступ до функцій та захист персональних даних.
5. Впроваджено інструментарій для адміністрування ліків та формування персоналізованих терапевтичних планів.
6. Інтегровано календар FullCalendar для ефективного керування графіком лікування та візуального контролю прийомів.
7. Забезпечено механізм фіксації вживання препаратів із можливістю моніторингу комплаєнтності та аналізу статистики.
8. Підтверджено стабільність і готовність застосунку до практичного використання за результатами комплексного тестування.

ДЯКУЮ ЗА УВАГУ!

ДОДАТОК Б. ЛІСТИНГИ ОСНОВНИХ МОДУЛІВ

```
using Microsoft.AspNetCore.Mvc;
using Microsoft.AspNetCore.Authorization;
using Microsoft.EntityFrameworkCore;
using MedicineTracker.Data;
using MedicineTracker.Models;
using System.Security.Claims;

namespace MedicineTracker.Controllers
{
    [Authorize]
    public class MedicineController : Controller
    {
        private readonly ApplicationDbContext _context;

        public MedicineController(ApplicationDbContext
context)
        {
            _context = context;
        }

        // Список ліків
        public IActionResult Index()
        {
            var userId =
User.FindFirstValue(ClaimTypes.NameIdentifier);

            var medicines = _context.Medicines
                .Where(m => m.UserId == userId)
                .ToList();

            return View(medicines);
        }

        // Форма створення
        public IActionResult Create()
        {
            return View();
        }

        // Додавання ліків
        [HttpPost]
        [ValidateAntiForgeryToken]
        public IActionResult Create(Medicine medicine)
        {
            medicine.UserId =
User.FindFirstValue(ClaimTypes.NameIdentifier);

            _context.Medicines.Add(medicine);
            _context.SaveChanges();

            return RedirectToAction(nameof(Index));
        }

        // Редагування (GET)
        public IActionResult Edit(int id)
        {
            var userId =
User.FindFirstValue(ClaimTypes.NameIdentifier);

            var medicine = _context.Medicines
                .FirstOrDefault(m => m.Id == id && m.UserId ==
userId);

            if (medicine == null)
                return NotFound();

            return View(medicine);
        }

        // Редагування (POST)
        [HttpPost]
        [ValidateAntiForgeryToken]
        public IActionResult Edit(int id, Medicine medicine)
        {
            var userId =
User.FindFirstValue(ClaimTypes.NameIdentifier);

            if (id != medicine.Id)
                return NotFound();

            var existing = _context.Medicines
                .AsNoTracking()
                .FirstOrDefault(m => m.Id == id && m.UserId ==
userId);

            if (existing == null)
                return NotFound();

            medicine.UserId = userId;

            _context.Update(medicine);
            _context.SaveChanges();

            return RedirectToAction(nameof(Index));
        }

        // Видалення (GET)
        public IActionResult Delete(int id)
        {
            var userId =
User.FindFirstValue(ClaimTypes.NameIdentifier);

            var medicine = _context.Medicines
                .FirstOrDefault(m => m.Id == id && m.UserId ==
userId);

            if (medicine == null)
                return NotFound();

            return View(medicine);
        }

        // Видалення (POST)
        [HttpPost, ActionName("Delete")]
        [ValidateAntiForgeryToken]
        public IActionResult DeleteConfirmed(int id)
        {
            var userId =
User.FindFirstValue(ClaimTypes.NameIdentifier);
```

```
            var medicine = _context.Medicines
                .FirstOrDefault(m => m.Id == id && m.UserId ==
userId);

            if (medicine != null)
            {
                _context.Medicines.Remove(medicine);
                _context.SaveChanges();
            }

            return RedirectToAction(nameof(Index));
        }

        // Деталі
        public IActionResult Details(int id)
```

```
using Microsoft.AspNetCore.Mvc;
using Microsoft.AspNetCore.Authorization;
using Microsoft.EntityFrameworkCore;
using MedicineTracker.Data;
using MedicineTracker.Models;
using System.Security.Claims;
using MedicineTracker.Services;

namespace MedicineTracker.Controllers
{
    [Authorize]
    public class ScheduleController : Controller
    {
        private readonly ApplicationDbContext _context;

        public ScheduleController(ApplicationDbContext context)
        {
            _context = context;
        }

        // INDEX (БЕЗ ML)
        public IActionResult Index()
        {
            var userId =
User.FindFirstValue(ClaimTypes.NameIdentifier);

            var schedules = _context.Schedules
                .Include(s => s.Medicine)
                .Where(s => s.UserId == userId)
                .ToList();

            return View(schedules);
        }

        // CREATE (GET)
        public IActionResult Create()
        {
            var userId =
User.FindFirstValue(ClaimTypes.NameIdentifier);

            ViewBag.Medicines = _context.Medicines
                .Where(m => m.UserId == userId)
                .ToList();

            return View();
        }

        // CREATE (POST)
        [HttpPost]
        [ValidateAntiForgeryToken]
        public IActionResult Create(MedicationSchedule
schedule)
        {
            var userId =
User.FindFirstValue(ClaimTypes.NameIdentifier);

            schedule.UserId = userId;

            _context.Schedules.Add(schedule);
            _context.SaveChanges();

            return RedirectToAction(nameof(Index));
        }

        // CALENDAR
        public IActionResult Calendar()
        {
            return View();
        }

        // EVENTS
        public JsonResult GetEvents()
        {
            var userId =
User.FindFirstValue(ClaimTypes.NameIdentifier);

            var events = _context.Schedules
                .Include(s => s.Medicine)
                .Where(s => s.UserId == userId)
                .Select(s => new
                {
                    title = s.Medicine.Name,
                    start = s.Time
                })
                .ToList();

            return Json(events);
        }

        // MARK AS TAKEN
        public IActionResult MarkAsTaken(int id)
        {
            var userId =
User.FindFirstValue(ClaimTypes.NameIdentifier);

            var schedule = _context.Schedules
                .FirstOrDefault(s => s.Id == id && s.UserId ==
userId);

            if (schedule == null)
                return NotFound();

            return View(schedule);
        }
    }
}
```

```
            return NotFound();

        schedule.Taken = true;
        _context.SaveChanges();

        return RedirectToAction(nameof(Index));
    }

    //  DASHBOARD
    public IActionResult Dashboard()
    {
        var userId =
User.FindFirstValue(ClaimTypes.NameIdentifier);

        var schedules = _context.Schedules
            .Include(s => s.Medicine)
            .Where(s => s.UserId == userId)
            .ToList();

        var ai = new SimpleService();

        var type = ai.Analyze(schedules);
        var rec = ai.GetRecommendation(type);

        ViewBag.Type = type;
        ViewBag.Recommendation = rec;

        return View(schedules);
    }

    // REMINDERS API
    public JsonResult GetTodayReminders()
    {
        var userId =
User.FindFirstValue(ClaimTypes.NameIdentifier);
        var now = DateTime.Now;

        // ШУКАЄМО: не прийняті, не сповіщені, час яких
вже настав або настане протягом 1 хв
        var reminders = _context.Schedules
            .Include(s => s.Medicine)
            .Where(s => s.UserId == userId &&
                        !s.Taken &&
                        !s.Notified &&
                        s.Time <= now.AddMinutes(1))
            .ToList();

        if (reminders.Any())
        {
            foreach (var item in reminders)
            {
                item.Notified = true;
            }
            _context.SaveChanges();
        }

        return Json(reminders.Select(r => new
        {
            id = r.Id,
            title = r.Medicine.Name,
            time = r.Time // передаємо об'єкт дати для
коректного парсингу в JS
        }));
    }
  }
}
```

```
using MedicineTracker.Models;

namespace MedicineTracker.Services
{
  public class SimpleService
  {
    public string Analyze(List<MedicationSchedule> data)
    {
      if (data == null || data.Count == 0)
        return "No data";

      int total = data.Count;
      int taken = data.Count(x => x.Taken);
      int missed = total - taken;

      double rate = (double)taken / total;

      if (rate > 0.9)
        return "Perfect";
      if (rate > 0.7)
        return "Good";
      if (rate > 0.4)
        return "Irregular";
      return "Critical";
    }

    public string GetRecommendation(string type)
    {
      return type switch
      {
        "Perfect" => "Excellent adherence. Keep it up!",
        "Good" => "You're doing well, try to avoid missing
doses.",
        "Irregular" => "Set reminders to improve
consistency.",
        "Critical" => "High risk! Consider strict schedule and
alerts.",
        _ => "No recommendation"
      };
    }
  }
}
```