Міністерство освіти і науки України
Харківський національний університет радіоелектроніки

Факультет комп'ютерних наук (або центр післядипломної освіти, або навчально-науковий центр заочної форми навчання)
(повна назва)

Кафедра програмної інженерії
(повна назва)

КВАЛІФІКАЦІЙНА РОБОТА
Пояснювальна записка

рівень вищої освіти перший (бакалаврський)

Програмна система для планування та моніторингу виконання особистих задач і досягнень. Тестування
(тема)

Виконав:
здобувач _4__ року навчання
групи ПЗПІ-21-1

Маріам АЛМАКАДМА
(Власне ім'я, ПРІЗВИЩЕ)

Спеціальність 121 – Інженерія програмного забезпечення
(код і повна назва спеціальності)
Тип програми освітньо-професійна
Освітня програма Програмна інженерія
(повна назва освітньої програми)

Керівник проф. кафедри ПІ Зоя ДУДАР
(посада, Власне ім'я, ПРІЗВИЩЕ)

Допускається до захисту
Зав. кафедри _____ Кирило СМЕЛЯКОВ
(підпис) (Власне ім'я, ПРІЗВИЩЕ)

2025 р.

Харківський національний університет радіоелектроніки

Факультет комп'ютерних наук (або центр післядипломної освіти, або навчально-науковий центр заочної форми навчання)
Кафедра програмної інженерії
Рівень вищої освіти перший (бакалаврський)
Спеціальність 121 – Інженерія програмного забезпечення
Тип програми Освітньо-професійна
Освітня програма Програмна Інженерія
(шифр і назва)

ЗАТВЕРДЖУЮ:
Зав. кафедри _____
(підпис)
«___» _____ 2025 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

здобувачеві Алмакадма Маріам Ібрагімівні
(прізвище, ім'я, по батькові)

1. Тема роботи Програмна система для планування та моніторингу виконання особистих задач і досягнень. Тестування
Затверджена наказом по університету від 19.05.2025р. № 397Ст

2. Термін подання здобувачем роботи до екзаменаційної комісії 11.06.2025

3. Вихідні дані до роботи Протестувати програмну систему для планування та моніторингу виконання особистих задач і досягнень, що включає серверну частину, клієнтський частину та мобільний застосунок. Для реалізації тестування використати інструменти та підходи, зокрема NUnit для модульного тестування серверної частини, Swagger для перевірки API, Cypress для автоматизованого тестування вебінтерфейсу, Android Studio, емулятор та фізичний мобільний пристрій для тестування мобільного застосунку.

4. Перелік питань, що потрібно опрацювати в роботі
Вступ, аналіз предметної галузі, формування вимог до програмної системи, архітектура та проєктування тестування програмного забезпечення, опис прийнятих програмних рішень та їх тестування, висновки, додатки.

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів роботи	Термін виконання етапів роботи	Примітка
1	Аналіз предметної галузі	08.04.2025	*виконано*
2	Створення тест плану	10.04.2025	*виконано*
5	Тестування ПЗ	27.05.2025	*виконано*
6	Оформлення пояснювальної записки	31.05.2025	*виконано*
7	Підготовка презентації та доповіді	04.06.2025	*виконано*
8	Попередній захист	05.06.2025	*виконано*
9	Нормоконтроль, рецензування	06.06.2025	*виконано*
10	Здача роботи у електронний архів	07.06.2025	*виконано*
11	Допуск до захисту у зав. кафедри	08.06.2025	*виконано*

Дата видачі завдання « 8 » « квітня » 2025р.

Здобувач _____
(підпис)

Керівник роботи _____ проф. кафедри ПІ Зоя ДУДАР
(підпис) (посада, Власне ім’я, ПРІЗВИЩЕ)

РЕФЕРАТ

Пояснювальна записка до кваліфікаційної роботи бакалавра, 107 с., 28 рисунків, 11 таблиці, 13 використаних джерел, 6 додатків.

ГЕЙМІФІКАЦІЯ, ПРОДУКТИВНІСТЬ, СИСТЕМА ДОСЯГНЕНЬ, ТЕСТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ, ЦИФРОВИЙ ПОМІЧНИК.

Об’єкт розробки – програмна система для планування та моніторингу особистих задач і досягнень, що включає серверну, клієнтську та мобільну частини.

Мета розробки – забезпечити якісне тестування програмного забезпечення, що допомагає користувачам підвищувати ефективність, формувати продуктивні звички і підтримувати мотивацію через контроль цілей і візуалізацію прогресу.

Метод рішення – аналіз предметної області, визначення функціональних вимог, розробка тестової документації (тест-план, чек-листи, тест-кейси), а також функціональне і нефункціональне тестування (продуктивність, зручність, сумісність) із застосуванням NUnit, Swagger, Cypress, Android Studio та емуляторів.

У результаті – проведено комплексне тестування системи на рівнях серверу, веб- та мобільного застосунку. Створено повну тестову документацію, визначено стратегії тестування окремих модулів та їх взаємодії. Надано рекомендації щодо підвищення зручності, надійності та користувацького досвіду.

ABSTRACT

GAMIFICATION, PRODUCTIVITY, ACHIEVEMENT SYSTEM, SOFTWARE TESTING, DIGITAL ASSISTANT.

The object of development is a software system for planning and monitoring personal tasks and achievements, which includes server-side, client-side, and mobile components.

The goal of the development is to ensure high-quality software testing that helps users improve efficiency, develop productive habits, and maintain motivation through goal tracking and progress visualization.

The method involves analysing the subject area, defining functional requirements, developing test documentation (test plan, checklists, test cases), as well as conducting functional and non-functional testing (performance, usability, compatibility) using NUnit, Swagger, Cypress, Android Studio, and emulators.

As a result, comprehensive testing of the system was performed at the server, web, and mobile application levels. Complete test documentation was created, testing strategies for individual modules and their interaction were defined. Recommendations were provided to improve usability, reliability, and overall user experience.

ЗМІСТ

Перелік скорочень 8

Вступ 9

1 Аналіз предметної галузі 10

1.1 Аналіз предметної галузі 10

1.2 Аналіз конкурентів 12

1.3 Виявлення та вирішення проблем 20

1.4 Постановка задачі 21

1.5 Цільова аудиторія 22

2 Формування вимог до програмної системи 23

2.1 Постановка мети 23

2.2 Загальний опис 23

2.3 Припущення та залежності 24

3 Архітектура та проєктування тестування програмного забезпечення 26

3.1 Моделювання прецедентів і користувацьких сценаріїв 26

3.2 Візуалізація процесу тестування 27

3.3 Структура системи та технології 29

3.4 Створення плану тестування 29

3.5 Розробка чек-листів 30

3.6 Розробка тест-кейсів 34

4 Опис прийнятих програмних рішень та їх тестування 39

4.1 Тестування серверної частини 39

4.1.1 Тестування функціональності сервісного рівня 39

4.1.2 Валідація вхідних даних у DTO 41

4.1.3 Інтеграційні тести для репозиторіїв 43

4.1.4 Використання автоматизованих тестів у регресійному тестуванні 44
4.1.4. Ручне тестування API 45
4.1.5 Перевірка продуктивності сервера 47
4.2 Заходи безпеки 49
4.3 Тестування клієнтської частини 50
4.4 Тестування мобільного застосунку 56
4.5 Життєвий цикл дефекту 57
Висновки 63
Перелік джерел посилання 63
Додаток А Звіт результатів перевірки на унікальність тексту в базі ХНУРЕ 66
Додаток Б Слайди презентації 68
Додаток В Тест план 77
Додаток Г Результати навантажувального тестування 99
Додаток Д Тест-кейси 103
Додаток Е Аналіз продуктивності застосунку 106

ПЕРЕЛІК СКОРОЧЕНЬ

XR – Extended Reality

VR – Virtual Reality

AR – Augmented Reality

MR – Mixed Reality

CSV – Comma Separated Values

ADHD – Attention Deficit Hyperactivity Disorder

СДУГ – синдром дефіциту уваги та гіперактивності

ВСТУП

У сучасному світі, де інформаційне навантаження невпинно зростає, дедалі більше людей постають перед труднощами в організації особистого часу, формуванні продуктивних звичок та збереженні мотивації. Відповідно до психологічних досліджень, нестача внутрішньої дисципліни та зовнішньої підтримки часто призводить до прокрастинації, емоційного виснаження й зниження особистої ефективності. У цьому контексті зростає потреба в інноваційних рішеннях, здатних допомогти користувачам ставити цілі, контролювати свій прогрес та досягати бажаних результатів.

Темою кваліфікаційної роботи є створення програмної системи для відстеження своїх досягнень або задач. Система має слугувати персональним цифровим помічником, що сприяє саморозвитку, підвищенню продуктивності та підтримці щоденної мотивації. Основне завдання полягає у впровадженні функціоналу для постановки цілей, фіксації результатів, візуалізації прогресу, а також наданні персоналізованих порад.

Метою роботи є проведення аналізу галузі, формування вимог та планування перевірки якості й надійності цієї програмної системи шляхом комплексного тестування. У межах роботи потрібно протестувати серверну, клієнтську частини, та мобільний застосунок, що забезпечують взаємодію користувача з системою.

Тестування охоплює функціональні та нефункціональні аспекти, включаючи навантаження, безпеку, стабільність та зручність використання. Особлива потрібно перевірити цілісність даних, коректність роботи API та адаптивність інтерфейсів.

Цільовою аудиторією є користувачі віком від 18 років, які прагнуть покращити особисту ефективність, розвинути нові звички й краще керувати своїм часом.

1 АНАЛІЗ ПРЕДМЕТНОЇ ГАЛУЗІ

1.1 Аналіз предметної галузі

У сучасному світі цифрові технології все глибше проникають у повсякденне життя, формуючи основу цифрової економіки. Однією з ключових тенденцій цієї трансформації є зростання зацікавленості до сервісів, що сприяють особистісному розвитку та самостійному плануванню. Зокрема, активно розвивається сегмент застосунків для підвищення особистої продуктивності, які використовують інструменти контролю звичок, мотиваційної підтримки та соціальної взаємодії. Такий напрямок знаходиться на стику психології саморегуляції, гейміфікації та технологічних рішень для здоров'я та навчання.

Доведено, що використання застосунків для формування звичок знижує мотиваційні бар'єри у навчальному процесі, а також підвищує автоматизм поведінки, що робить початок та виконання дій менш залежним від внутрішньої мотивації. У роботі також зазначено, що складні та змінні звички, можуть бути автоматизовані завдяки повторенню у схожому контексті, що забезпечує стабільність і гнучкість в реалізації дій, враховуючи різноманітність змісту. Розглядаючи концепцію мотиваційних конфліктів, бачимо, що суперечливі думки та альтернативні варіанти дій заважають зосередженню на основному завданні. В той час, як автоматизовані звички, які запускаються певним контекстом або сигналом, допомагають зменшити частоту таких конфліктів та забезпечують безперервність дій та ментальне розвантаження [1].

Швидкий розвиток цифрових технологій в освіті спричинив трансформацію традиційних підходів до викладання, сприяючи переходу до активного, орієнтованого на навчання здобувачів. У контексті концепції Education 4.0 ключовим є впровадження технологій, які відповідають потребам сучасного суспільства та цифрових поколінь. Застарілі методи дедалі більше поступаються інтерактивним та гнучким підходам, заснованим на цифрових інструментах [2].

Одним із таких рішень є технології XR, які охоплюють VR, AR і MR. Вони створюють захопливе, інтерактивне середовище, де здобувачі не лише краще залучені до навчання, а й мають змогу розвивати навички через занурення у

змодельовані ситуації. Особливо ефективними XR стають у поєднанні з гейміфікацією, яка інтегрує ігрові механіки – такі як змагання, віртуальні нагороди та рівні – у навчальні процеси, стимулюючи мотивацію та інтерес до навчання.

Досвід використання гейміфікаційних підходів у навчанні та технологіях для фізичної активності свідчить, що такі елементи, як система балів, віртуальні відзнаки та змагальні механіки, суттєво впливають на рівень залучення та утримання користувачів. Наприклад, у фітнес-застосунку «START» реалізовано нарахування балів за активність, віртуальні бейджі (наприклад, за проходження 5 км на день) та підрахунок днів безперервної активності. Щоденний лічильник кроків і нагадування сприяли зростанню середньої кількості кроків на тиждень, формуючи у користувачів відчуття прогресу та компетентності [3].

Тим часом, результати одного з експериментальних досліджень показали, що візуальне оформлення гейміфікаційних елементів не є ключовим чинником підвищення мотивації. Учасники, які виконували однакове завдання з трьома варіантами зворотного зв’язку (без гейміфікації, із текстовим та з розважальним оформленням), продемонстрували схожі рівні мотивації. Це вказує на важливість самої механіки надання зворотного зв’язку, а не її візуального подання [4].

Ще одним критично важливим аспектом ефективної гейміфікації є соціальна взаємодія та наставництво. Можливість підтримки, спілкування з наставниками або однолітками, а також отримання індивідуальної відповіді в межах мікрозавдань позитивно впливає на залученість та довготривале використання.

Сучасні цифрові системи персоналізації дедалі частіше застосовують алгоритми штучного інтелекту (ШІ) для адаптації вмісту відповідно до індивідуальних характеристик користувачів.

Такі системи можуть покращувати досвід користувача, знижуючи інформаційне перевантаження та допомагаючи досягати цілей, особливо для тих, хто має контрольовану мотивацію.

Розглядаючи користувачів із автономною мотивацією, бачимо, що дня них надзвичайно важливою є прозорість алгоритмів і можливість впливати на процес

персоналізації. Лише за цих умов персоналізоване середовище може сприяти не лише задоволенню потреб, а й довготривалому добробуту [5].

1.2 Аналіз конкурентів

Розглянемо продукти, що демонструють приклади адаптивного підходу до мотивації користувача, персоналізації досвіду та впровадження поведінкової аналітики. Розгляд таких систем дозволяє виокремити ефективні практики та виявити наявні обмеження.

«Beeminder» – це застосунок для відстеження цілей, який поєднує самооцінку та контрактні зобов'язання (див. рис. 1.1–1.3). Користувачі встановлюють графік досягнення цілей і повинні дотримуватись прогресу, інакше «Beeminder» стягує штраф у вигляді грошових зобов'язань.

Платформа дозволяє автоматично відстежувати дані з таких сервісів, як «Fitbit» або «duolingo», і має яскраво виражену «червону лінію», що показує, коли користувач відхиляється від мети [6].

Переваги:

– мотивація через фінансові штрафи за відхилення від мети;
– підтримка численних інтеграцій з іншими застосунками;
– чітке візуальне відображення прогресу через графік.

Недоліки:

– високий рівень стресу через можливі фінансові санкції;
– відсутність деяких функцій соціалізації, що обмежує взаємодію з іншими користувачами.

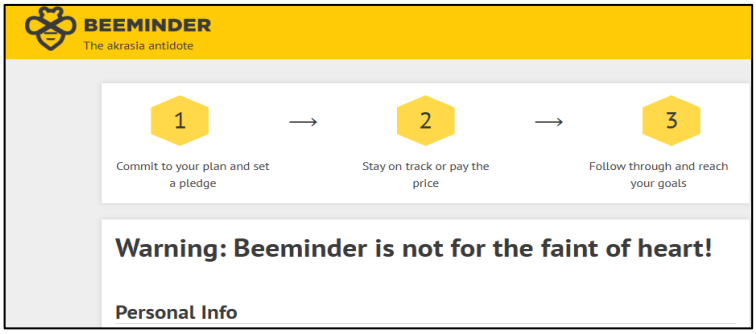

Рисунок 1.1 – Сторінка «Реєстрація»

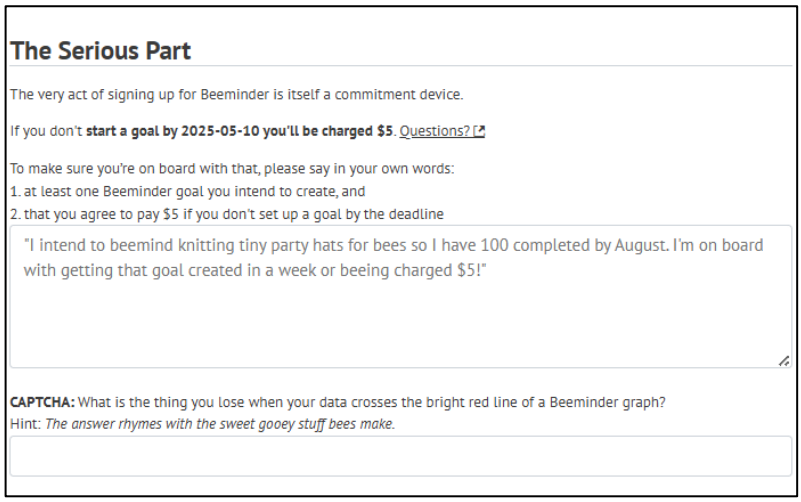

Рисунок 1.2 – Важлива інформація зі сторінки «Реєстрація»

Рисунок 1.3 – Графік досягнень

«HabitBull» – це мобільний застосунок, який допомагає формувати й підтримувати корисні звички, зосереджуючись на досягненні особистих цілей (див. рис. 1.4). Простий, але функціональний інтерфейс дозволяє швидко переглядати прогрес у зручному календарному форматі, що охоплює цілий місяць. Основна увага приділяється візуалізації, що надає користувачам можливість аналізувати свою динаміку за допомогою графіків, діаграм та аналітичних інструментів. Застосунок також дає змогу створювати індивідуальні плани, наприклад, тренування кілька разів на тиждень або щоденне вивчення іноземної мови [7].

Переваги:

– інтуїтивний інтерфейс дозволяє легко орієнтуватися в додатку та швидко переглядати загальну картину прогресу;
– розширені інструменти аналітики надають доступ до графіків, які відображають динаміку досягнення цілей у зручному форматі;
– можливість експорту даних у форматі CSV дозволяє проводити детальний аналіз або використовувати зовнішні аналітичні засоби;
– гнучке налаштування цілей і повторень дає змогу створювати як прості щоденні звички, так і складніші довготривалі цілі;
– підтримка хмарного зберігання та синхронізації забезпечує збереження прогресу незалежно від пристрою;
– активна спільнота сприяє мотивації та обміну досвідом серед користувачів.

Недоліки:

– обмежені функції у безкоштовній версії;
– застосунок не має ігрових елементів, таких як бали, бейджі чи нагороди, які б додатково стимулювали користувачів;
– хоча існують зовнішні форуми, безпосередньої інтеграції соціальних функцій (наприклад, спільні цілі або виклики) поки що відсутні.

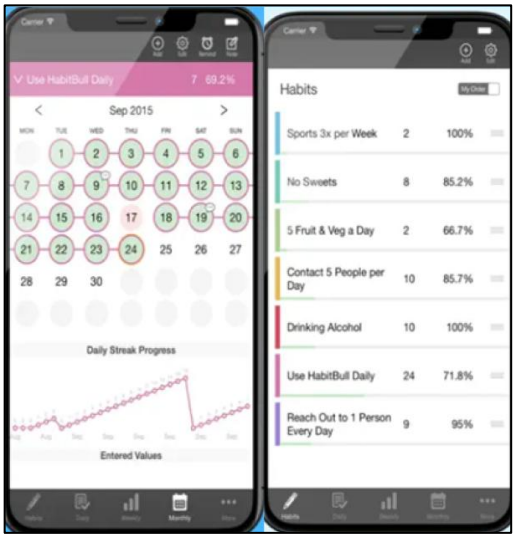

Рисунок 1.4 – Знімок з екрана мобільного застосунку для відстеження прогресу

«Way of Life» – це мобільний застосунок, розроблений для формування й моніторингу звичок. Його основна мета – допомогти користувачам розвивати корисні звички, ставити перед собою досяжні цілі та регулярно відстежувати свій прогрес (див. рис. 1.5). Користувачі мають змогу створювати індивідуальні звички, встановлювати періодичність виконання, отримувати нагадування та вести особисту статистику.

Застосунок надає простий і інтуїтивно зрозумілий інтерфейс, де можна легко відзначати виконання або пропуск звичок. «Way of Life» має зручну функцію створення графіків і статистики для візуалізації прогресу, що допомагає підтримувати мотивацію [8].

Переваги:

- інтерфейс мінімалістичний і зрозумілий, що дозволяє швидко почати відстежувати звички;
- наявність візуалізації результатів через графіки та статистики дозволяє чітко бачити прогрес у досягненні цілей;
- регулярні нагадування допомагають не забути про виконання завдання;
- користувачі можуть налаштовувати різні параметри для звичок (частота, час виконання, тощо);
- безкоштовна версія дозволяє відстежувати до трьох звичок, що для багатьох є достатнім.

Недоліки:

- відсутність системи балів, нагород або інших елементів гейміфікації;
- немає можливості взаємодії з іншими користувачами або підтримки через соціальні функції;
- для розширених можливостей (наприклад, більший вибір звичок або додаткові звіти) необхідно оновити до преміуму версії.

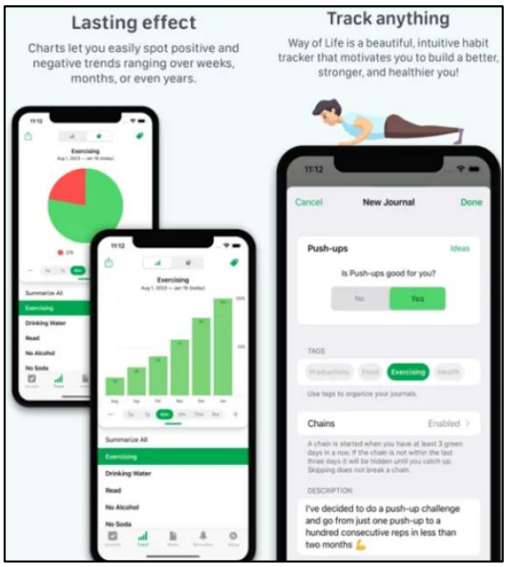

Рисунок 1.5 – Знімок з екрана застосунку «Way of Life»

«Productive» – це мобільний застосунок, створений для формування, організації та підтримки звичок, який дозволяє користувачам структурувати свій день, досягати цілей і розвивати корисні щоденні практики (див. рис. 1.6–1.7). Програма призначена для користувачів, які прагнуть поліпшити власну продуктивність, встановити чіткий розпорядок дня та закріпити нові звички в повсякденному житті [9].

Переваги:

– простий і зручний дизайн, що дозволяє легко створювати та відстежувати звички;
– можливість налаштовувати частоту виконання звичок;
– графіки, що показують виконання звичок і загальний прогрес;
– регулярні нагадування;
– можливість відстежувати серії виконаних днів, що підтримує мотивацію.

Недоліки:

– базовий функціонал доступний безкоштовно, але для доступу до розширених функцій потрібна платна підписка;
– застосунок орієнтований на індивідуальне використання і не підтримує взаємодію між користувачами.

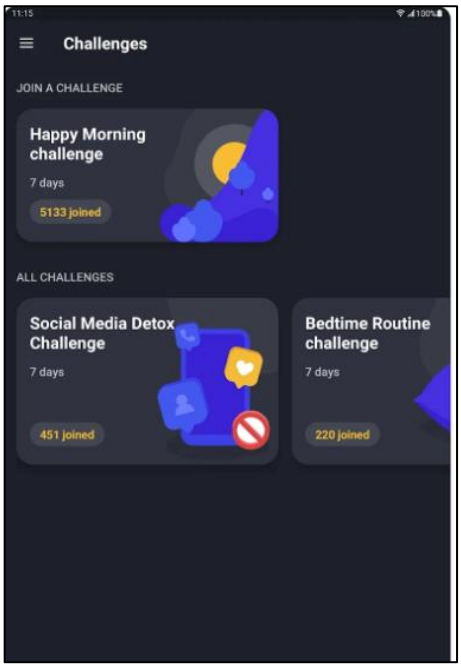

Рисунок 1.6 – Знімок з екрана застосунку «Productive» сторінка «Challenges»

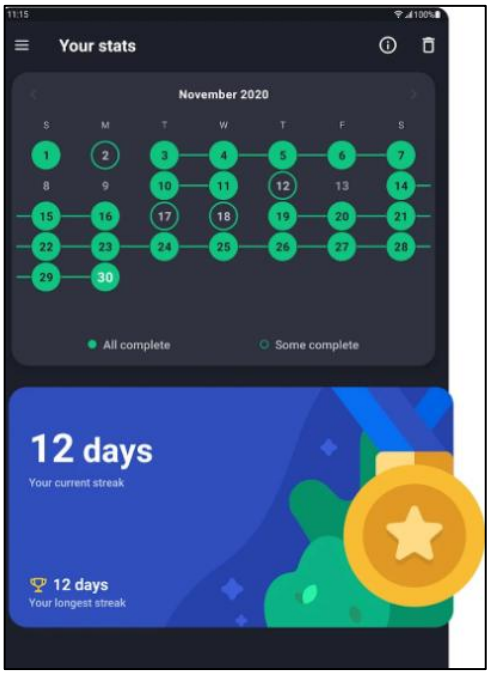

Рисунок 1.7 – Знімок з екрана застосунку «Productive»

«Fabulous» – це мобільний застосунок для формування звичок, турботи про психічне здоров’я та загального саморозвитку, який допомагає користувачам поступово покращувати якість життя, знижувати рівень стресу та досягати особистих і професійних цілей (див. рис. 1.8–1.9). Застосунок вирізняється на ринку своєю орієнтацією на поведінкову психологію та науково обґрунтовані

методики формування звичок. Він використовує індивідуальний підхід до створення щоденних рутин, включаючи фізичні, емоційні та когнітивні аспекти здоров’я [10].

Переваги:

- застосунок пропонує персоналізовані програми на основі аналізу звичок і цілей користувача;
- користувачі «працюють» над своїм «життєвим проєктом», отримуючи нагороди за виконання завдань і завершення етапів;
- фокус на психічне здоров'я – «Fabulous» включає практики медитації, вправи для зменшення тривожності, рекомендації для боротьби з прокрастинацією;
- інтуїтивний дизайн робить застосунок легким у використанні, навіть для тих, хто вперше займається відстеженням звичок;
- рекомендації базуються на психології звичок і дослідженнях поведінкової науки.
- «Thrive with ADHD» – спеціалізовані програми, які допомагають користувачам із СДУГ покращити концентрацію, зменшити прокрастинацію та структурувати своє життя. Це включає вправи на уважність, методи управління часом та інструменти для боротьби зі стресом.

Недоліки:

- багато функцій доступні лише в преміум версії, яка коштує близько $39,99 на рік;
- «Fabulous» не включає функцій взаємодії між користувачами, таких як спільноти чи групи підтримки;
- програми переважно орієнтовані на встановлення рутин, що може обмежити його для користувачів, які потребують ширших функцій, таких як контроль навичок або складних цілей.

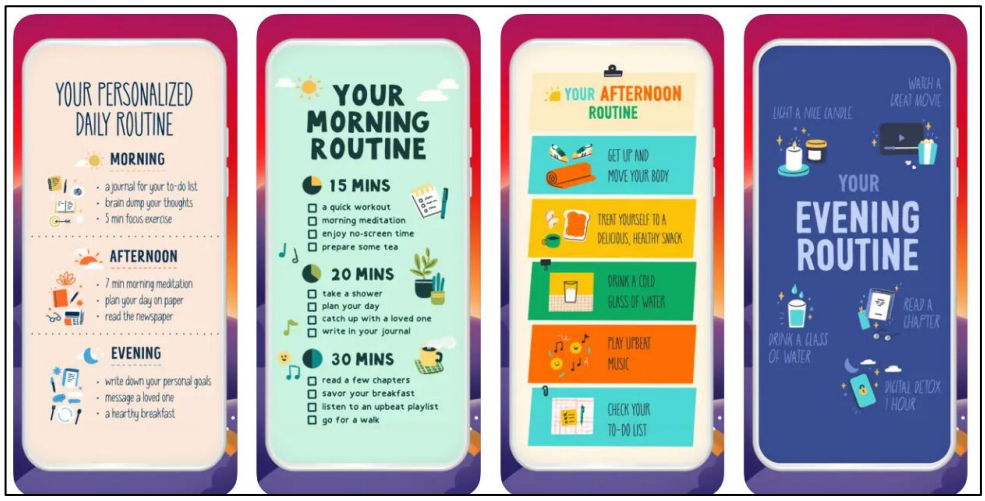

Рисунок 1.8 – Знімок з екрана застосунку «Fabulous»

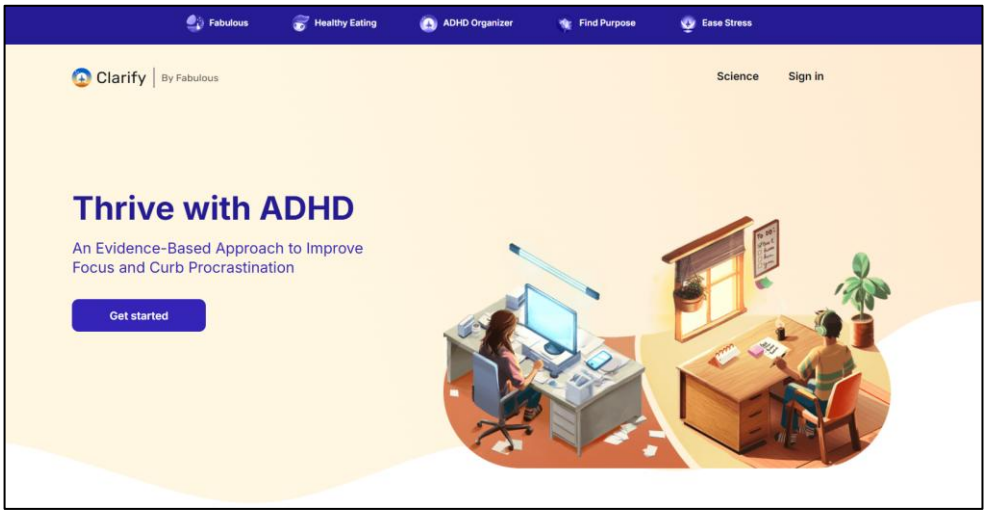

Рисунок 1.9 – Сторінка «Thrive with ADHD»

На основі аналізу рішень, таких як «Beeminder», «HabitBull», «Way of Life», «Productive» та «Fabulous», можна зробити висновок, що сучасні застосунки поєднують різні механіки: від фінансової мотивації та візуалізації прогресу до використання науково обґрунтованих методів та гейміфікації. Проте в більшості з них спостерігається обмежена соціальна взаємодія, що може знижувати рівень залученості. Найбільш ефективними вважаються ті платформи, які дозволяють персоналізувати досвід користувача та надають не лише інструменти відстежування, а й підтримку у вигляді нагадувань, рекомендацій та елементів позитивного підкріплення.

1.3 Виявлення та вирішення проблем

На основі аналізу платформ для контролю цілей і звичок, можна виявити низку спільних проблем, які потребують вирішення або вдосконалення. Однією з головних є обмежений функціонал безкоштовних версій, що спостерігається у більшості застосунків, таких як «HabitBull», «Way of Life», «Productive» та «Fabulous». Це створює бар'єр для нових користувачів, які не готові відразу вкладати кошти, і значно зменшує залучення аудиторії на початкових етапах. Водночас деякі продукти, як «Beeminder», використовують фінансову мотивацію у формі штрафів, що дійсно може стимулювати досягнення цілей, але водночас провокує підвищений рівень стресу і тривожності. Такий підхід є суперечливим і не підходить користувачам, схильним до перфекціонізму або зниженого психологічного ресурсу.

Ще однією спільною проблемою є відсутність або обмеженість соціальної взаємодії між користувачами. Більшість платформ, зокрема «Way of Life», «HabitBull» та «Productive», орієнтовані на індивідуальне користування й не пропонують можливостей підтримки з боку спільноти чи друзів. Це знижує потенціал взаємного підбадьорення, обміну досвідом або змагального елементу, який міг би сприяти довгостроковому дотриманню звичок. Навіть у таких просунутих продуктах, як «Fabulous», попри наявність персоналізованих програм та наукового підґрунтя, відсутні інструменти для створення відчуття спільноти, що могло б суттєво посилити ефект підтримки у користувачів із ментальними труднощами або соціальними потребами.

Також спостерігається недостатня реалізація ігрових елементів. У деяких застосунках гейміфікація або зовсім відсутня, або зведена до мінімуму, що зменшує привабливість процесу формування звичок, особливо серед молоді. Наприклад, у «Fabulous» вона добре поєднана з мотиваційними цілями, однак інші продукти часто взагалі не мають системи нагород чи віртуальних досягнень. Це знижує привабливість ігрового підходу до саморозвитку, особливо серед молоді та користувачів, які шукають не лише функціональність, а й емоційне залучення.

Водночас візуалізація даних і зручність інтерфейсу є сильною стороною більшості розглянутих застосунків, зокрема «HabitBull» та «Productive», однак неможливість підлаштовуватись під індивідуальні потреби або надмірна простота можуть не задовольнити потреби просунутих користувачів, що, у свою чергу, зумовлює підвищену потребу в застосуванні штучного інтелекту для поглиблення індивідуалізації досвіду.

Таким чином, виявлені проблеми вказують на потребу в розробці платформи нового покоління, яка б поєднувала індивідуальний підхід, продуману гейміфікацію, гнучкий і доступний базовий функціонал, а також елементи соціалізації. Додатково, інтеграція інтелектуальних алгоритмів для формування персоналізованих рекомендацій дозволила б не лише підтримувати щоденну мотивацію, а й створювати умови для формування стійкої звички шляхом поєднання зовнішнього заохочення та внутрішньої цілеспрямованості.

1.4 Постановка задачі

Для подолання виявлених проблем і кращого задоволення потреб користувачів у сфері саморозвитку та формування звичок, постає необхідність у розробці сучасної програмної системи, здатної об’єднати ефективні механізми мотивації, самоконтролю та соціальної взаємодії.

Передбачається створення рішення, яке дає змогу ставити особисті цілі, відстежувати прогрес, отримувати персоналізовані рекомендації та взаємодіяти з іншими користувачами. Елементи гейміфікації, наприклад, накопичення балів, здобуття нагород, просування по рівнях і система лідерства – сприятимуть підвищенню залученості користувачів і підтримці внутрішньої мотивації. Додатково, інтегрований чат з помічником, а саме ШІ, надаватиме персоналізовані відповіді на запити, пов’язані з цілями, звичками та особистим прогресом.

Такий підхід дасть змогу користувачам формувати й підтримувати позитивні звички, отримувати підтримку від спільноти, а також зберігати високу залученість завдяки системі рівнів, нагород і лідерства.

Розробка має охоплювати як вебінтерфейс, так і мобільний застосунок, адаптовані до потреб цільової аудиторії. З боку тестувальника ключовим завданням є всебічна перевірка стабільності, зручності й відповідності функціоналу заявленим вимогам, а також забезпечення безперебійної та інтуїтивно зрозумілої взаємодії користувача з платформою.

1.5 Цільова аудиторія

Програмна система повинна бути орієнтована на повнолітніх користувачів віком від 18 років незалежно від статі, професії чи рівня технічної підготовки. Цільовою аудиторією є особи, які прагнуть покращити свою особисту ефективність, планувати й досягати цілей, формувати корисні звички або розвивати нові навички.

Серед користувачів можуть бути як новачки, які вперше користуються подібними застосунками, так і досвідчені користувачі, що вже мають сформовані підходи до самоорганізації, але шукають інструмент з гнучким функціоналом та індивідуальним підходом.

2 ФОРМУВАННЯ ВИМОГ ДО ПРОГРАМНОЇ СИСТЕМИ

2.1 Постановка мети

Метою даної кваліфікаційної роботи є тестування програмної системи для управління персональними цілями, яка сприятиме підвищенню мотивації, самоорганізації та продуктивності користувачів.

Система повинна надавати можливість постановки, редагування та відстеження цілей, візуалізації прогресу, використання гейміфікаційних елементів та отримання інтелектуальної підтримки у форматі діалогу з ШІ.

У зв’язку з цим головним завданням є не лише перевірка функціональності цих можливостей, але й забезпечення стабільності, безпеки та коректної роботи системи на різних платформах.

Особливу увагу слід звернути на зручність користувацького інтерфейсу, щоб взаємодія із системою була інтуїтивною та приємною для широкої аудиторії користувачів.

Тестування повинно охоплювати ключові компоненти системи: взаємодію клієнта і сервера, функціонал вебверсії в популярних браузерах, а також адаптивність і стабільність мобільного застосунку на Android. Крім того, перевіряються нефункціональні вимоги – час відгуку, безпека даних і відновлення після збоїв. Ретельне тестування є необхідним для забезпечення високої якості продукту.

2.2 Загальний опис

Повинні бути реалізовані такі функціональні вимоги:

- цілей;
- має бути передбачене візуальне представлення прогресу за допомогою графіків і визначення прогресу;
- система повинна включати гейміфікаційні елементи: бали, рівні, досягнення, таблицю лідерів;

- користувач має отримувати сповіщення про дедлайни, мотиваційні цитати, підтримка від друзів та інше;
- має бути реалізована можливість додавання друзів, перегляду профілів та взаємодії з іншими користувачами;
- у системі має бути реалізований чат з ШІ, який надає персоналізовані поради та відповіді у форматі діалогу в реальному часі.

Нефункціональні вимоги:

- система повинна бути масштабованою і стабільною при великій кількості користувачів;
- час відгуку при взаємодії з інтерфейсом не повинен перевищувати 2 секунд;
- повинні бути реалізовані механізми безпеки даних, шифрування та контроль доступу;
- застосунок має працювати як у вебверсії, у поширених браузерах (наприклад, Chrome, Edge), так і на мобільних пристроях з операційною системою Android, зі схожим функціоналом;
- інтерфейс повинен бути інтуїтивно зрозумілим та логічно структурованим;
- дані користувача мають зберігатися з урахуванням резервного копіювання і можливістю відновлення після збоїв;
- оновлення системи не повинні впливати на збережені дані або порушувати роботу активних сесій;
- текстові повідомлення, сповіщення та елементи інтерфейсу повинні бути локалізовані українською мовою, з можливістю подальшого розширення мовної підтримки.

2.3 Припущення та залежності

Планування, підготовка та виконання тестування базуються на низці припущень і зовнішніх залежностей, які є критичними для забезпечення коректності перевірки функціоналу системи «Naviria». Зокрема, передбачається:

- вихідний код системи є структурованим, підтримуваним і доступним у репозиторії на GitHub, що дозволяє тестувальнику виконувати локальні збірки, створювати окремі гілки для тестів та перевіряти останні зміни;
- після впровадження нової функції (або кількох функцій) розробниками, розпочинається відповідне тестування;
- вимоги до реалізації функціоналу встановлюються розробниками в межах заданої архітектури та технічних обмежень, а отже, тестування орієнтується на ці рамки як на основу для створення тест-кейсів;
- тестувальник має доступ до усіх необхідних залежностей, бібліотек і конфігурацій, що дозволяє виконувати збірку проєкту локально та запускати юніт-, інтеграційні та інтерфейсні тести без технічних обмежень;
- тестове середовище стабільне, ізольоване від продуктивного та містить штучні дані, що виключає витік або пошкодження реальних даних;
- тестування через Swagger і інтерфейс відбувається напряму, без емуляції, з активним сервером, від якого залежать веб- і мобільні клієнти.

Припускається, що тестувальник має доступ до:

- тестового середовища з підключеною базою даних;
- API-документації (Swagger);
- вебзастосунку та мобільної версії;
- прав для створення/редагування користувачів і даних;
- журналів подій (логів) для виявлення технічних проблем.

Припущення та залежності обов’язкові для ефективного тестування. Їх порушення може викликати затримки, дефекти або зниження якості.

3 АРХІТЕКТУРА ТА ПРОЄКТУВАННЯ ТЕСТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

3.1 Моделювання прецедентів і користувацьких сценаріїв

Система повинна бути орієнтована як на неавторизованих користувачів, які ще не мають облікового запису, так і на тих, хто вже створив профіль. Розглянемо інформаційні потреби користувачів через діаграму прецедентів (див. рис. 3.1).

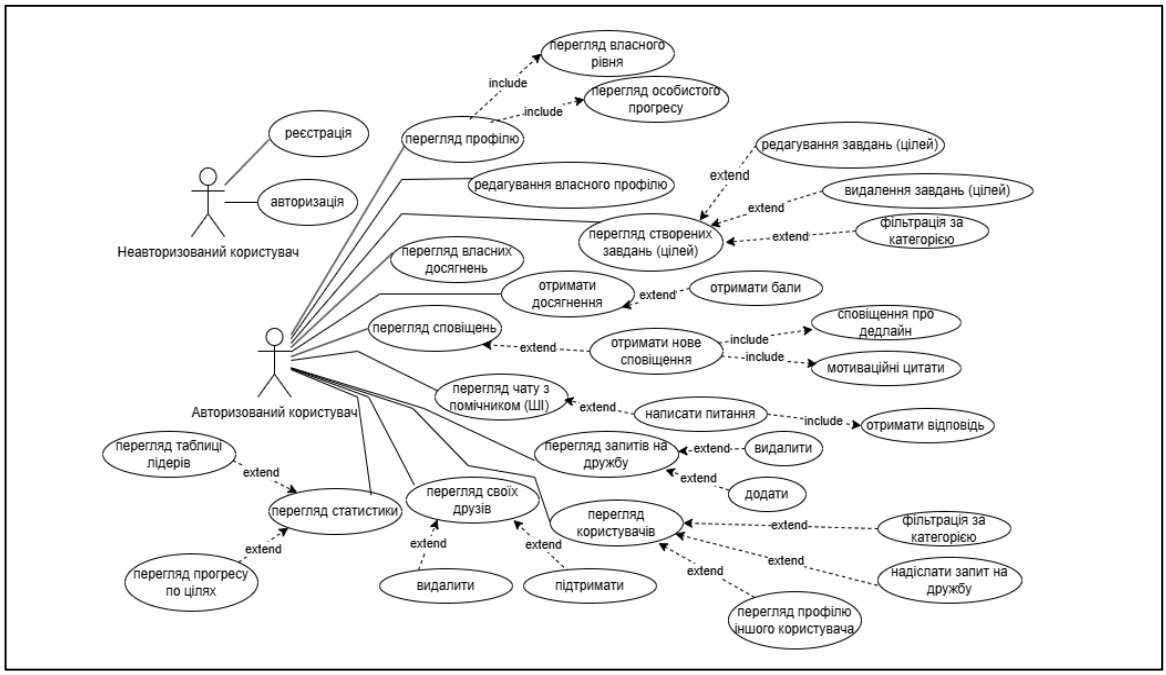

Рисунок 3.1 – Діаграма прецедентів з актором користувач

До ключових функцій системи потрібно віднести процеси реєстрації та входу до облікового запису, можливість редагування особистої інформації, створення, редагування, видалення завдань (цілей) із функціоналом фільтрації за категоріями, а також візуалізація досягнутого прогресу у вигляді графіків із підрахунком витраченого часу.

Користувачі повинні мати змогу переглядати власну статистику активності, накопичені досягнення, рівні та бали, що доповнюється наявністю таблиці лідерів як елементу гейміфікації. Механізм сповіщень потрібен для повідомлення про наближення дедлайнів або надсилання мотиваційних цитат.

Крім того, визначаємо соціальну складову – користувачі можуть переглядати профілі інших, надсилати та приймати запити на дружбу, а також організовувати

список друзів за обраними категоріями. Чат із віртуальним помічником на базі штучного інтелекту забезпечує інтерактивну взаємодію у форматі реального часу, відповідаючи на запити та надаючи рекомендації для досягнення цілей.

Визначаємо назву програмної системи для відстеження своїх досягнень або задач – «Naviria».

3.2 Візуалізація процесу тестування

Узагальнену діаграму життєвого циклу тестування наведено на рисунку 3.2, яка ілюструє етапи від аналізу вимог до завершення тестування. Вона демонструє як послідовність основних етапів, так і паралельні напрями перевірки (наприклад, ручне й автоматизоване тестування), що відповідає гібридному підходу в проєкті «Naviria».

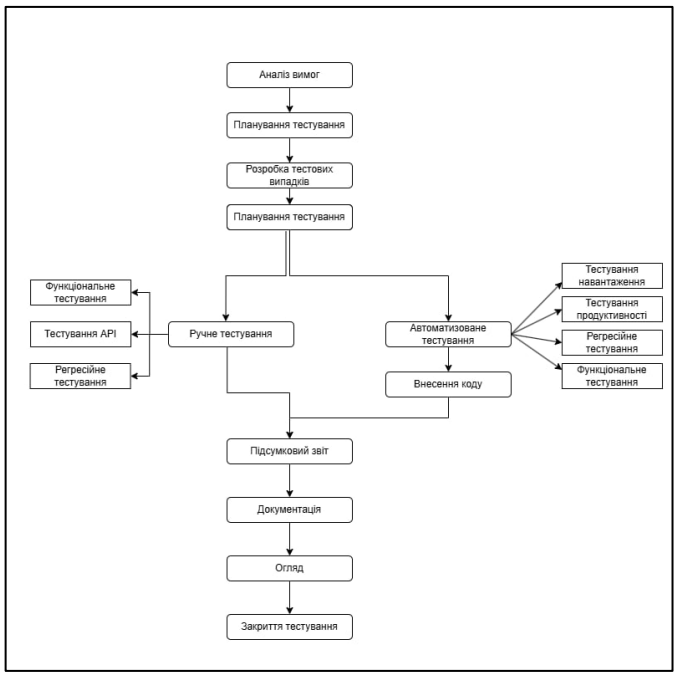

Рисунок 3.2 – Діаграма життєвого циклу тестування

Процес запланованого тестування програмного забезпечення у межах реалізації даної системи охоплює чітко визначену послідовність дій, що забезпечують всебічну перевірку функціональності, продуктивності та стабільності ПЗ. Робота над тестуванням розпочинається з аналізу функціональних і нефункціональних вимог, що надаються командою розробників або аналітиків. У

цьому етапі детально вивчаються специфікації, на основі яких визначається очікувана поведінка системи та формуються критерії приймання. Після цього здійснюється планування тестування, в межах якого визначаються обсяги перевірки, обирається відповідна стратегія, розподіляються ролі в команді тестування, встановлюються дедлайни та обираються тестові середовища, з урахуванням ресурсів і характеру функціоналу.

На основі зібраних вимог формується набір тестових випадків. Вони включають як ручні сценарії, орієнтовані на користувацьку поведінку, так і автоматизовані скрипти для перевірки повторюваних або критичних функцій. Паралельно з підготовкою тестів команда розробників вносить зміни до коду або реалізує нові функції, які надалі підлягають перевірці. Після внесення змін розпочинається функціональне тестування, під час якого перевіряється відповідність реалізованого функціоналу очікуваним вимогам, включно з позитивними та негативними сценаріями, перевіркою граничних умов та логіки обробки даних.

Система має відкриті інтерфейси, тож наступним кроком є тестування API, під час якого перевіряється коректність відповідей, відповідність статус-кодів стандартам та правильність форматів переданих і прийнятих даних. Ручне тестування, що відбувається паралельно або після функціонального, зосереджене на UI/UX компонентах: перевіряється візуальна відповідність, поведінка елементів інтерфейсу, логіка навігації, а також взаємодія користувача із системою. Далі виконується регресійне тестування, яке має на меті впевнитися, що нововнесені зміни не порушили вже стабільний функціонал.

Крім перевірки логіки, особливу увагу приділяється навантажувальному тестуванню, яке дозволяє оцінити стабільність і продуктивність системи в умовах інтенсивної експлуатації. З цією метою застосовуються як синтетичні сценарії навантаження, так і реальні дані для оцінки граничної пропускної здатності. Автоматизоване тестування виконується регулярно в межах CI/CD-процесів для оперативного виявлення критичних дефектів, що дозволяє мінімізувати людський фактор і забезпечити стабільність релізів.

За результатами всіх етапів формується підсумковий звіт, який містить дані про виконані тести, виявлені дефекти, їхній пріоритет і статус. Цей звіт є основою для прийняття рішень щодо подальшої розробки або випуску продукту. Останнім етапом є огляд результатів тестування, підтвердження відповідності продукту встановленим вимогам, оновлення супровідної документації та формальне завершення тестової активності.

3.3 Структура системи та технології

У проєкті чітко розділено клієнтську (веб і мобільну) та серверну частини, що відповідає підходу service-oriented architecture. Сервер реалізовано на ASP.NET Core (.NET 8) і взаємодіє з хмарними сховищами MongoDB Atlas для зберігання документно-орієнтованих даних і Cloudinary для обробки зображень. Зовнішні API, зокрема OpenAI GPT для AI-помічника та Google OAuth для автентифікації, інтегровані через REST API з відповідними бібліотеками. Вебклієнт створено на React, що забезпечує динамічний та інтерактивний інтерфейс, а мобільний застосунок розроблено на Kotlin з використанням Jetpack Compose. Обидва клієнти комунікують із сервером через REST API, отримуючи доступ до даних, профілю, завдань і сповіщень. Для роботи з базою даних використовується MongoDB .NET Driver, а для зовнішніх сервісів – відповідні .NET SDK. Така архітектура забезпечує модульність, масштабованість та безпеку системи.

3.4 Створення плану тестування

План тестування було створено на основі функціональних і нефункціональних вимог до системи, а також після аналізу специфікацій, архітектури та цілей проєкту «Naviria». Детальний план тестування включено до додатку В. Процес планування охоплював наступні етапи:

- визначено основні цілі тестування;
- було обрано відповідні типи тестування (функціональне, нефункціональне, інтеграційне, навантажувальне тощо) відповідно до архітектури проєкту та цілей якості;

- визначено критерії початку та завершення тестування;
- стратегію тестування;
- визначено життєвий цикл дефекту;
- визначено конфігурації серверної частини, бази даних, мобільного емулятора та інших необхідних інструментів.

У процесі тестування потрібно забезпечити дотримання загальноприйнятих підходів і термінології відповідно до міжнародних стандартів якості програмного забезпечення [11], а також методології тестування [12].

3.5 Розробка чек-листів

Перед початком тестування, потрібно виділити модуль тестування, написати чек-листи та вже після тест-кейси. Наведемо приклад чек-листів для реєстрації нового користувача через вебінтерфейс (див. табл. 3.1).

Таблиця 3.1 – Таблиця чек-листів для реєстрації через вебінтерфейс

№	Тестова операція	Очікуваний результат	Статус	Коментарі
1	Перевірка наявності доступу до сторінки реєстрації	Сторінка реєстрації доступна за правильним URL		
2	Перевірка кнопки «Реєстрація» на головній сторінці	Кнопка «Реєстрація» відображається на головній сторінці		
3	Перевірка обов’язкових полів: Ім’я	Присутній placeholder для імені		
4	Перевірка на коректність введення ім’я	Перевірити правильність відображення повідомлення про помилку при введенні некоректного значення		
5	Перевірка обов’язкових полів: Прізвище	Присутній placeholder для прізвища		
6	Перевірка на коректність введення прізвища	Перевірити правильність відображення повідомлення про помилку при введенні некоректного значення		

Кінець таблиці 3.1

№	Тестова операція	Очікуваний результат	Статус	Коментарі
7	Перевірка обов’язкових полів: Пошта	Присутній placeholder для пошти		
8	Перевірка валідації для пошти	Помилка при неправильному форматі: без @, без домену, з пробілами тощо		
9	Перевірка обов’язкових полів: Пароль	Присутній placeholder для пароля		
10	Перевірка валідації для пароля	Помилка при: довжині < 8, відсутності великої/малої літери або цифри		
11	Перевірка валідації поля: Підтвердіть пароль	Присутній placeholder для підтвердити пароля		
12	Перевірка валідації для підтвердження пароля	Помилка при невідповідності значення з полем Пароль		
13	Перевірка обов’язкових полів: Стать	Присутні радіокнопок для вибору статі		
14	Перевірка валідації поля стать	Перевірити, чи працює валідація для обов'язковості вибору статі		
15	Перевірка обов’язкових полів: Дата народження	Присутнє поле, яке дозволяє вибір дати з календаря		
16	Перевірка валідації для дня народження	Перевірити, чи працює валідація для мінімального віку (18 років)		
17	Перевірка обов’язкових полів: Прізвисько (Nickname)	Присутній placeholder для прізвиська		
18	Перевірка валідації для Nickname	Помилка при введенні символів, окрім латинських літер і цифр; довжина < 3 або > 20		
19	Перевірка кнопки «Почати» на сторінці для реєстрації	Кнопка «Почати» відображається на сторінці для реєстрації		
20	Перевірити, реєстрацію з валідними даними, після натискання на «Почати»	Користувача було створено, перейшов на сторінку «Профіль»		

Чек-листи для ручного тестування блоку додати друга та пошук за прізвиськом навели у таблиці 3.2.

Таблиця 3.2 – Таблиця чек-листів для додавання нових друзів

№	Тестова операція	Очікуваний результат	Статус	Коментарі
1	Перейти на сторінку «Ком'юніті» через верхню панель	Відкривається сторінка «Ком'юніті»		
2	Перевірити наявність блоку «Знайти друзів»	Елемент «Знайти друзів» присутній на сторінці		
3	Перевірити, чи відображається список користувачів	Список користувачів, яких можна додати в друзі, присутній		
4	Перевірити наявність нікнейму у кожного користувача	Виводиться нікнейм кожного користувача		
5	Перевірити наявність рівня у кожного користувача	Виводиться рівень користувача		
6	Перевірити наявність опису користувача	Якщо опис присутній – виводиться; якщо відсутній – відображається «Опис відсутній»		
7	Ввести частину нікнейму в поле пошуку й натиснути кнопку «Пошук»	Виводяться користувачі, нікнейми яких відповідають введеному тексту		
8	Очистити поле пошуку та натиснути кнопку «Пошук»	Виводиться повний список користувачів		
9	Натиснути кнопку «Додати в друзі» біля одного з користувачів	Кнопка змінюється на, наприклад, «Запит надіслано» (або інше повідомлення про успішний запит)		

Сформулюємо чек-листи для тестування серверної логіки, зокрема для юніт-тестування сервісу, що відповідає за контроль взаємодії з чат-асистентом – AssistantChatService (див. табл. 3.3).

Таблиця 3.3 – Таблиця чек-листів для AssistantChatService

№	Тестова операція	Очікуваний результат	Статус	Коментарі
1	Перевірка виклику GetUserChatAsync з порожнім userId	ArgumentException з повідомленням «User ID cannot be null or empty»		Покриває перевірку на null/empty значення
2	Перевірка виклику GetUserChatAsync, коли користувача не існує	NotFoundException		Мокаємо _userService.UserExistsAsync
3	Перевірка GetUserChatAsync, коли є збережені повідомлення	Повертається список AssistantChatMessageDto		Перевірити правильність мапінгу
4	Перевірка GetUserChatAsync, коли список повідомлень порожній	Повертається порожній список		
5	Виклик SendMessageAsync з порожнім UserId	ArgumentException з повідомленням «User ID is required.»		
6	Виклик SendMessageAsync з порожнім Message	ArgumentException з повідомленням «Message is required. »		
7	Перевірка виклику SendMessageAsync, коли кількість повідомлень < 20	Нове повідомлення додається		Перевірити, що метод AddMessageAsync викликаний двічі
8	Перевірка виклику SendMessageAsync, коли кількість повідомлень >= 20	Видалення історії повідомлень		Перевірити, що DeleteAllForUserAsync викликаний
9	Перевірка коректності виклику _messageSecurityService.Validate	Метод викликається з правильними аргументами		Мокаємо Validate
10	Перевірка, що відповідь GPT-4 генерується і зберігається у базі	Нове повідомлення з роллю assistant додається		Перевірити контент, роль і userId

Кінець таблиці 3.3

№	Тестова операція	Очікуваний результат	Статус	Коментарі
11	Перевірка, що в історії GPT передається лише останні 10 пар повідомлень	GetByUserIdAsync викликається і результати обрізаються		
12	Перевірка, що історія GPT передається з коректними ролями (user, assistant)	Усі повідомлення мають валідні ролі		
13	Перевірка логування при успішному отриманні історії повідомлень	Метод _logger.LogInformation викликається		Можна протестувати через ILogger<T> mock
14	Перевірка логування при порожній історії	Лог з повідомленням про порожню історію		
15	Перевірка логування при винятках (наприклад, не валідний userId)	Виводиться warning log		

Аналогічно розробимо тест кейси й для інших частин.

3.6 Розробка тест-кейсів

Розглянемо приклад тест-кейсів для автоматизованої перевірки функціоналу реєстрації нового користувача на вебсайті (див. табл. 3.4). Реєстрація є одним із критичних етапів взаємодії користувача із системою, тому важливо переконатися, що вона працює стабільно та коректно за різних умов. Для забезпечення цього застосовуються як позитивні, так і негативні сценарії автоматизованого тестування.

Передумови для виконання тестів:

- у системі ще не існує облікового запису з тими самими даними, що плануються для реєстрації. Зокрема, зазначені нікнейм та електронна адреса мають бути унікальними;

– вебсайт завантажується без помилок, і серверна частина системми функціонує у штатному режимі, без збоїв або технічного обслуговування.

Таблиця 3.4 – Тест-кейси для реєстрації через вебінтерфейс.

№	Назва тесту	Кроки відтворення	Очікуваний результат	Пріоритет
TC001	Доступ до сторінки реєстрації	1. Перейти за URL /register	Відображається форма реєстрації	Високий
TC002	Кнопка «Реєстрація» на головній сторінці	1. Відкрити головну сторінку 2. Перевірити наявність кнопки	Кнопка «Реєстрація» відображається	Середній
TC003	Поле Ім’я – placeholder	1. Відкрити сторінку реєстрації 2. Перевірити placeholder у полі «Ім’я»	Placeholder присутній, наприклад: «Введіть ім’я»	Низький
TC004	Ім’я – валідація неправильного вводу	1. Ввести «@!$» у поле «Ім’я» 2. Натиснути «Почати»	Повідомлення: «Ім'я введено некоректно»	Середній
TC005	Поле Прізвище – placeholder	1. Перевірити поле «Прізвище»	Placeholder присутній	Низький
TC006	Прізвище – валідація неправильного вводу	1. Ввести «123» у поле 2. Подати форму	Повідомлення: «Прізвище введено некоректно»	Середній
TC007	Поле Пошта – placeholder	1. Перевірити placeholder поля Email	Placeholder присутній	Низький
TC008	Email – валідація формату	Ввести: - example.com - example@ - example@com - @example.com	Виводиться помилка: «Невірний формат пошти»	Високий
TC009	Пароль – placeholder	1. Перевірити наявність placeholder для пароля	Placeholder присутній	Низький

Продовження таблиці 3.4

№	Назва тесту	Кроки відтворення	Очікуваний результат	Пріоритет
TC010	Пароль – валідація складності	Ввести: - abc - Password - 12345678 - pass1234	Повідомлення: «Пароль має містити великі та малі літери та цифру»	Високий
TC011	Повтор пароля – placeholder	1. Перевірити наявність поля для підтвердження пароля	Placeholder присутній	Низький
TC012	Підтвердження пароля – невідповідність	Ввести у «Пароль»: Pass1234, у підтвердженні: Pass5678	Повідомлення: «Паролі не збігаються»	Високий
TC013	Поле Стать – наявність	1. Перевірити наявність радіокнопок: «Чоловік», «Жінка»	Обидві кнопки присутні	Середній
TC014	Стать – валідація неправильного значення	Вибрати нічого	Повідомлення: «Оберіть стать»	Високий
TC015	Поле Дата народження – наявність	Перевірити, чи є календарний віджет	Поле доступне для вибору	Середній
TC016	Дата народження – валідація віку	Встановити дату молодше ніж 18 років від сьогодні (наприклад, 1 рік тому)	Повідомлення: «Потрібно бути старше 18 років»	Високий
TC017	Nickname – placeholder	Перевірити, чи є placeholder	Placeholder присутній	Низький
TC018	Nickname – валідація символів	Ввести: - te (коротке) - nickname_with_space - ім’я	Повідомлення: «Нікнейм має містити лише латинські літери та цифри (3-20 символів)»	Високий
TC019		Ввести: 12345678901234567890123456789012345678901 (довге)	Вставлений текст – обрізається	Середній

Кінець таблиці 3.4

№	Назва тесту	Кроки відтворення	Очікуваний результат	Пріоритет
ТС020	Успішна реєстрація з валідними даними	Заповнити всі поля коректними значеннями та натиснути «Почати»	Перенаправлення на сторінку «Профіль», де видно інформацію користувача	Високий

Перейдемо до розробки тест-кейсів для ручного тестування сторінки «Ком'юніті», а саме додавання нових друзів.

Передумова: Користувач авторизувався в системі (див. табл. 3.5).

Таблиця 3.5 – Тест-кейси для тестування додавання друзів через вебінтерфейс

№	Назва тесту	Кроки відтворення	Очікуваний результат	Пріоритет
ТС001	Перехід на сторінку «Ком'юніті»	1. Відкрити головну сторінку сайту 2. Натиснути на кнопку «Ком'юніті» у верхній панелі	Відкривається сторінка «Ком'юніті»	Високий
ТС002	Перевірка наявності блоку «Знайти друзів»	1. Перейти на сторінку «Ком'юніті»	Блок з назвою «Знайти друзів» присутня на сторінці	Середній
ТС003	Відображення списку користувачів	1. Перейти на сторінку «Ком'юніті»	Список користувачів, доступних для додавання у друзі, відображається	Високий
ТС004	Перевірка нікнейму у користувачів	1. Перейти на сторінку «Ком'юніті» 2. Перевірити кожного користувача у списку	Біля кожного користувача відображається нікнейм	Середній

Кінець таблиці 3.5

№	Назва тесту	Кроки відтворення	Очікуваний результат	Пріоритет
TC005	Перевірка рівня користувачів	1. Перейти на сторінку «Ком'юніті» 2. Перевірити кожного користувача у списку	Біля кожного користувача відображається рівень	Середній
TC006	Перевірка наявності опису користувача	1. Перейти на сторінку «Ком'юніті» 2. Знайти користувача з описом 3. Перевірити текст опису	Якщо опис є – він відображається;	Середній
TC007	Перевірка відсутності опису користувача	1. Перейти на сторінку «Ком'юніті» 2. Знайти користувача без опису 3. Перевірити текст	Якщо немає – показується «Опис відсутній»	Середній
TC008	Пошук за частиною нікнейму	1. Ввести частину нікнейму в поле пошуку 2. Натиснути кнопку «Пошук»	Виводяться користувачі, чиї нікнейми відповідають введеному тексту	Високий
TC009	Очищування поля пошуку	1. Очистити поле пошуку 2. Натиснути кнопку «Пошук»	Відображається повний список користувачів	Середній
TC010	Відправлення запиту в друзі та зміна кнопки	1. Натиснути кнопку «Додати в друзі» біля користувача	Кнопка змінюється на «Запит надіслано» (або інше підтвердження); користувач зникає зі списку	Високий

Аналогічно розробимо тест-кейси і для інших модулей та тестів.

4 ОПИС ПРИЙНЯТИХ ПРОГРАМНИХ РІШЕНЬ ТА ЇХ ТЕСТУВАННЯ

4.1 Тестування серверної частини

Для забезпечення надійності та стабільної роботи серверної частини програмного забезпечення було проведено комплексне тестування, що охоплює як функціональні, так і нефункціональні аспекти системи. Зокрема, виконано автоматизовані тести для перевірки коректної роботи сервісів і валідації даних, інтеграційні тести для оцінки взаємодії з базою даних, ручне тестування API-ендпоінтів, а також навантажувальні тести для перевірки продуктивності системи.

4.1.1 Тестування функціональності сервісного рівня

Було реалізовано юніт-тести для перевірки логіки окремих сервісів. Тестування здійснювалось із використанням фреймворку NUnit та бібліотеки Moq для створення мок-об'єктів. Основна увага приділялася перевірці:

- правильності обробки вхідних даних;
- відповідності очікуваних результатів обробки;
- правильного реагування на виняткові ситуації.

Тестували створення, оновлення та видалення задач, додавання підзадач, а також функціонал позначення задач як виконаних. Для роботи з папками та категоріями перевірялися операції створення, отримання, оновлення та видалення, включно з каскадним видаленням пов’язаних задач. Тести також охоплювали обробку запитів у друзі, включаючи створення, оновлення, видалення запитів, отримання списків друзів, пошук потенційних друзів та пошук за нікнеймом.

У рамках тестування цитат перевірялося додавання, редагування, видалення та отримання цитат. Функціонал сповіщень охоплював отримання, створення, маркування як прочитаних та видалення сповіщень. Також проводилися перевірки механізмів валідації вхідних даних і авторизації через JWT-токени.

Тестування гейміфікації включало операції над досягненнями, нарахування балів, видачу досягнень користувачам, а також підрахунок статистики у таблиці лідерів. Зокрема, перевіряли роботу сервісу TaskRewardService, що відповідає за

нарахування балів за виконання завдань і зміну їх статусу. Основний акцент робився на методі GrantTaskCompletionRewardsAsync, який реалізує логіку нарахування балів і оновлення рівня користувача (див. рис. 4.1). Для ізоляції тестів використовували мок-об’єкти сервісів рівнів та користувачів (ILevelService, IUserService). Юніт-тести охоплювали різні сценарії, зокрема правильність нарахування балів залежно від пріоритету, кількості підзавдань, тегів і термінів виконання, а також коректну зміну статусу завдань і оновлення рівня користувача.

```
[Test]
public void TC004_CalculateTaskPoints_ShouldReturnCorrectPoints()
{
    var task = new TaskEntity
    {
        Priority = 4,
        Tags = new List<Tags> { new Tags { TagName = "a" }, new Tags { TagName = "b" }, new Tags { TagName = "a" } },
        Subtasks = new List<SubtaskBase>
        {
            new SubtaskRepeatable
            {
                CheckedInDays = new List<DateTime> { DateTime.UtcNow, DateTime.UtcNow.AddDays(-1) }
            },
            new SubtaskRepeatable
            {
                CheckedInDays = new List<DateTime> { DateTime.UtcNow }
            }
        }
    };

    int expected = 10  // base
                 + 4 * 2 // priority = 8
                 + 2 * 3 // 2 subtasks = 6
                 + 2     // 2 distinct tags
                 + 3 * 2; // 3 CheckedInDays = 6

    int result = _service.CalculateTaskPoints(task);

    Assert.That(result, Is.EqualTo(expected));
}
```

Рисунок 4.1 – Тест перевірки логіки підрахунку балів у методі CalculateTaskPoints

Для роботи з користувачами тестували створення, оновлення, часткове оновлення, видалення, а також завантаження фото профілю у хмарне сховище. У частині пошуку оцінювалася коректність пошуку користувачів за різними параметрами з урахуванням виключення поточного користувача та його друзів. Нарешті, тестувалися взаємодії з помічником, включаючи надсилання запитів, отримання відповідей та отримання історії повідомлень.

Було перевірено автоматичне резервне копіювання бази даних, що реалізовано за допомогою спеціального менеджера BackupManager, який створює стиснені архіви з періодичністю не частіше ніж раз на 7 днів. Резервні копії зберігаються локально упродовж 30 днів, що забезпечує можливість відновлення даних у разі втрати або пошкодження.

Для забезпечення цілісності даних та підтримки узгодженості бази даних на серверній частині було реалізовано механізм каскадного видалення. Ця функціональність дозволяє автоматично видаляти всі залежні записи під час видалення батьківської сутності. Зокрема, до видалення досягнень вони спочатку вилучаються з профілів усіх користувачів; при видаленні категорій одночасно знищуються й усі пов’язані з ними завдання; під час видалення папок автоматично усуваються всі завдання, які до них належать. У разі видалення користувача здійснюється повне очищення від пов’язаних даних, таких як папки, завдання, сповіщення, чати з асистентом і запити на дружбу, а також цей користувач видаляється зі списків друзів інших користувачів.

Для перевірки коректності роботи зазначеної логіки були розроблені юніт-тести. Тестування охоплювало різні сценарії, зокрема успішне видалення та обробку винятків у разі відсутності необхідних сутностей у базі даних. Крім того, було проведено ручне тестування, щоб переконатися у правильності реалізації каскадного видалення в умовах реального використання системи.

Застосування такого підходу дозволяє уникнути появи не пов’язаних записів у базі даних, запобігає виникненню логічних невідповідностей у структурі збереженої інформації.

4.1.2 Валідація вхідних даних у DTO

У процесі тестування серверної частини було перевірено коректність роботи валідаційних атрибутів, реалізованих у DTO (Data Transfer Object). Основна увага приділялася тому, щоб переконатися, що система правильно обробляє як валідні, так і не валідні вхідні дані, що надходять до API. Зокрема, проводилися позитивні та негативні тести для перевірки атрибутів [Required], [EmailAddress], [MinLength], [RegularExpression] та інших.

На рисунку 4.2 наведено приклад DTO-моделі UserLoginDto, що застосовується під час авторизації користувача.

```
public class UserLoginDto
{
    [Required(ErrorMessage = "Email is required")]
    [EmailAddress]
    [RegularExpression("^[a-zA-Z0-9._%+-]+@[a-zA-Z0-9.-]+\\.[a-zA-Z]{2,}$")]
    public string Email { get; set; } = string.Empty;

    [Required(ErrorMessage = "Password is required")]
    [DataType(DataType.Password)]
    [MinLength(8, ErrorMessage = "Password must be at least 8 characters long")]
    [RegularExpression(@"^(?=.*[a-z])(?=.*[A-Z])(?=.*\d).+$",
    ErrorMessage = "Password must contain upper, lower letters and digits")]
    public required string Password { get; set; }
}
```

Рисунок 4.2 – Модель для авторизації користувача

Щоб переконатися, що валідаційні атрибути працюють належним чином, були розроблені тести, які охоплюють як позитивні, так і негативні сценарії. Наприклад, перевірка, що система не пропускає некоректну електронну адресу або пароль, який не відповідає вимогам складності (див. рис. 4.3).

```
[Test]
public void TC03_InvalidEmail_ShouldBeInvalid_WhenEmailIsInvalidFormat()
{
    // Arrange
    var userLoginDto = new UserLoginDto
    {
        Email = "john.doe@com",
        Password = "Passw0rd123"
    };

    // Act
    var validationResults = new List<ValidationResult>();
    var validationContext = new ValidationContext(userLoginDto);
    var isValid = Validator.TryValidateObject(userLoginDto, validationContext, validationResults, true);

    // Assert
    Assert.That(isValid, Is.False);
}

  [Test]
  public void TC06_InvalidPassword_ShouldBeInvalid_WhenPasswordLacksDigit()
  {
      // Arrange
      var userLoginDto = new UserLoginDto
      {
          Email = "john.doe@example.com",
          Password = "Password"
      };

      // Act
      var validationResults = new List<ValidationResult>();
      var validationContext = new ValidationContext(userLoginDto);
      var isValid = Validator.TryValidateObject(userLoginDto, validationContext, validationResults, true);

      // Assert
      Assert.That(isValid, Is.False);
      Assert.That(validationResults.Any(r => r.ErrorMessage.Contains("Password must contain upper, lower letters and digits")), Is.True);
  }
```

Рисунок 4.3 – Тести для валідації атрибутів авторизації

Під час тестування було охоплено перевірку валідації під час авторизації, створення й оновлення користувачів, додавання та редагування досягнень (achievements), а також обробку запитів до персонального помічника і систему сповіщень.

4.1.3 Інтеграційні тести для репозиторіїв

Для тестування репозиторіїв провели інтеграційне тестування з використанням бази даних MongoDB, з якою налаштовувалося підключення через клас MongoDbContext.

Щоб забезпечити ізольованість тестів та уникнути задвоєння або конфліктів даних, перед кожним тестом виконувалося повне очищення відповідної колекції методом DeleteMany. Це очищення реалізовано в базовому класі RepositoryTestBase<TEntity>, від якого успадковуються всі тести для репозиторіїв. Таким чином, кожен тест виконувався у «чистому середовищі», що дозволяло зберігати стабільність результатів і виключити вплив попередніх запусків.

Наприклад, у тестах для оновлення папок (FolderEntity) перевірялася як успішна модифікація запису (TC005_UpdateAsync_ShouldModifyFolder), так і негативний сценарій із не валідним ідентифікатором (TC006_UpdateAsync_InvalidId_ShouldReturnFalse) (див. рис. 4.4).

```
[Test]
public async Task TC005_UpdateAsync_ShouldModifyFolder()
{
    var folder = new FolderEntity
    {
        Name = "Old Name",
        UserId = ObjectId.GenerateNewId().ToString()
    };

    await Collection.InsertOneAsync(folder);

    folder.Name = "Updated Name";
    var updated = await _folderRepository.UpdateAsync(folder);

    Assert.That(updated, Is.True);

    var fetched = await Collection.Find(f => f.Id == folder.Id).FirstOrDefaultAsync();
    Assert.That(fetched.Name, Is.EqualTo("Updated Name"));
}

[Test]
public async Task TC006_UpdateAsync_InvalidId_ShouldReturnFalse()
{
    var folder = new FolderEntity
    {
        Id = ObjectId.GenerateNewId().ToString(),
        Name = "Does Not Exist",
        UserId = ObjectId.GenerateNewId().ToString()
    };

    var result = await _folderRepository.UpdateAsync(folder);
    Assert.That(result, Is.False);
}
```

Рисунок 4.4 – Тести для перевірки оновлення папок

Такий підхід дозволяє оцінити поведінку системи в різних умовах і виявити потенційні проблеми з обробкою запитів.

4.1.4 Використання автоматизованих тестів у регресійному тестуванні

Після впровадження змін у кодовій базі, оновлення функціоналу або виправлення помилок проводилось регресійне тестування (див. рис. 4.5). З метою пришвидшення перевірки та зменшення ймовірності пропуску помилок, було активно використано раніше написані автоматизовані юніт- та інтеграційні тести, що дозволяло:

– перевірити, що функціонал не зазнав небажаних змін;

– забезпечити стабільну роботу критичних компонентів сервісу;

– швидко виявляти побічні ефекти змін.

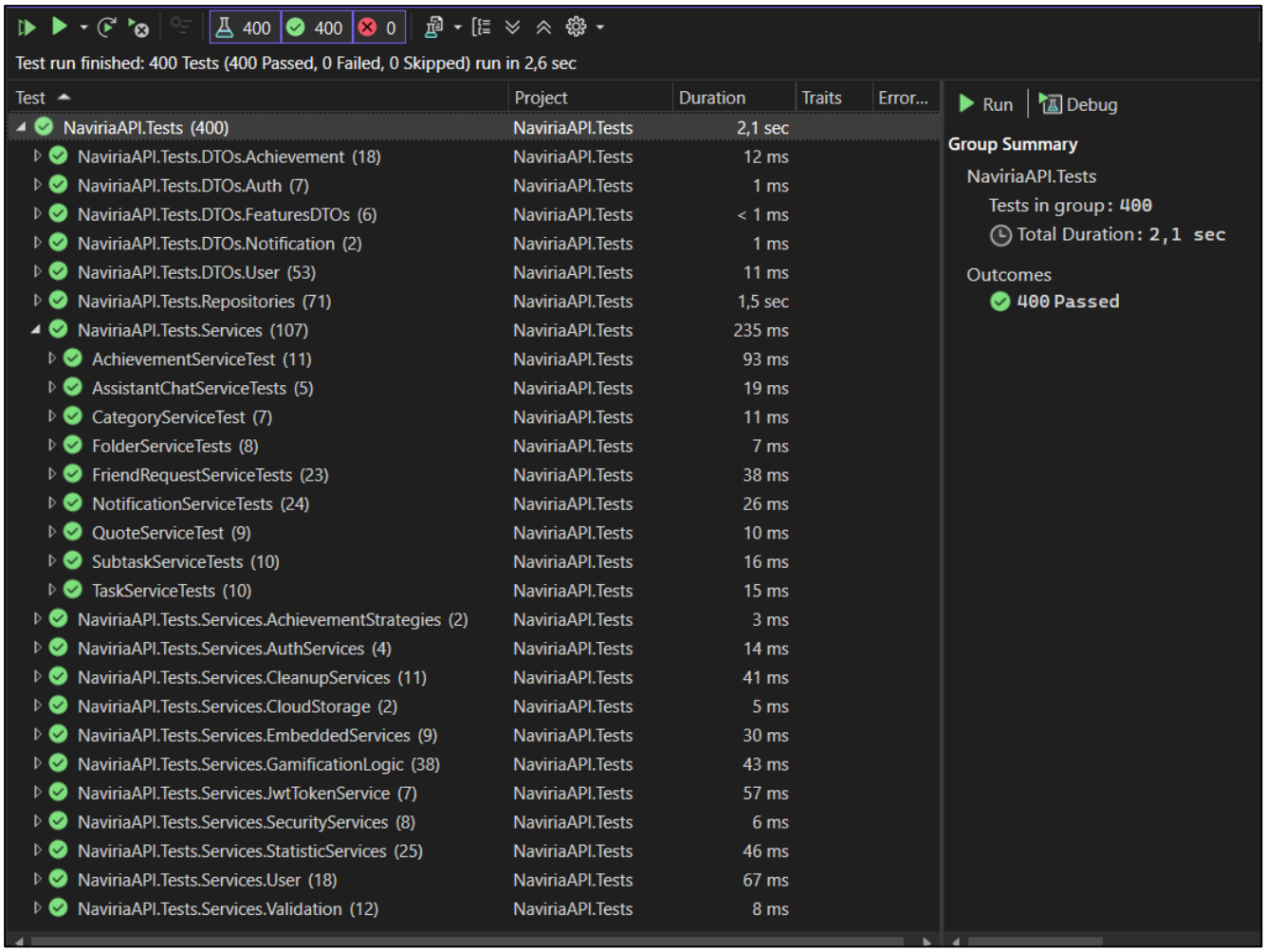

Рисунок 4.5 – «Test Explorer» тестування сервера

Таким чином, автоматизовані тести стали невід'ємною частиною процесу контролю якості на всіх етапах розробки.

4.1.4. Ручне тестування API

Функціональність API перевірялась вручну через інтерфейс Swagger UI. Після виконання кожного запиту здійснювалась ретельна перевірка відповідності змін у базі даних, що включала перевірку коректності створення, оновлення, видалення та отримання даних.

Тестування охоплювало різні типи запитів (GET, POST, PUT, DELETE) для всіх основних контролерів системи. Зокрема, перевірялись контролери для роботи з досягненнями користувачів, персональним чатом з помічником (див. рис. 4.6), авторизацією користувачів (див. рис. 4.7), завантаженням зображень у хмарне сховище, керуванням категоріями, папками (див. рис. 4.8), запитами у друзі та списками друзів. Також були протестовані контролери для таблиці лідерів, сповіщень, цитат, різних статистичних даних, підзадач, задач, користувачів, а також пошуку користувачів. В процесі тестування зверталась увага на коректність повернення статусів HTTP, відповідність структури відповіді документації, а також дотримання бізнес-логіки.

ID	Опис	Умови	Кроки виконання тестів	Очікуваний результат	Фактичний результат	Пріоритет	Результат тесту	Оточення	Дата створення
TC-01	Надсилання повідомлення з валідним UserId і Message	У базі існує користувач з UserId = 680a80f8133ea0477ae99b0b; Працюючий OpenAI API ключ	1. Відправити POST запит на /AssistantChat/ask через Swagger з валідними даними2. Перевірити, чи відповідь містить згенерований текст	Статус відповіді 200 OK; Тіло містить відповідь: "Привіт! Столиця України – Київ."	Статус 200; Тіло містить "Привіт! Столиця України – Київ."	Високий	Pass	Swagger (AssistantChat) + OpenAI API ключ	24.04.2025
TC-02	Надсилання повідомлення без UserId	-	1. Відправити POST запит на /AssistantChat/ask з порожнім UserId ("")	Статус 400 Bad Request, повідомлення: "A non-empty request body is required."	Статус 400, повідомлення "A non-empty request body is required."	Середній	Pass	Swagger (AssistantChat)	24.04.2025
TC-03	Надсилання запиту без тексту повідомлення	Користувач існує в базі	1. Відправити POST запит на /AssistantChat/ask без параметра Message	Статус 400 Bad Request, повідомлення про відсутність тексту (наприклад: "Message is required")	Статус 500, повідомлення "Message is required"	Високий	Pass	Swagger (AssistantChat)	24.04.2025
TC-04	Надсилання повідомлення з неіснуючим UserId	UserId 000000000000000000000000 не існує в базі; база не містить цього UserId	1. Відправити POST запит на /AssistantChat/ask із UserId, якого немає в базі ("000000000000000000000000")	Статус 404 та відсутність користувача у базі	відсутність користувача у базі	Високий	Pass	Swagger (AssistantChat)	24.04.2025
TC-05	Автоматичне створення задачі (навіриї) на основі запиту користувача	Користувач з userId = 680cc5ad9c376cfd07332f06 існує в базі.	1. Надіслати POST-запит на /AssistantChat/ask з { "userId": "680cc5ad9c376cfd07332f06", "message": "Хочу вивчити данську мову з рівня B2 до C1", "isTaskRequest": true }	Статус 200. Відповідь: ✅ Задача ... створена. У БД створено новий документ задачі з правильним заголовком та статусом InProgress.	Статус 200. надана відповідь Задача створена в БД з правильним заголовком і статусом. task id 682f64a723769376a175aeed	Високий	Pass	Swagger (AssistantChat), MongoDB, OpenAI	22.05.2025
TC-06	Запит на створення задачі з неіснуючим користувачем	Користувача з таким userId немає в базі.	1. Надіслати POST-запит на /AssistantChat/ask з isTaskRequest: true і валідним текстом, але з неіснуючим userId.	Відповідь 200 з повідомленням про помилку. У логах сервера має бути запис про те, що користувача не знайдено.	Статус 200. Відповідь: An error occurred.... У консолі повідомлення про відсутність користувача.	Високий	Pass	Swagger (AssistantChat), MongoDB	22.05.2025
TC-07	Запит без поля message	Користувач з userId = 680cc5ad9c376cfd07332f06 існує в базі.	1. Надіслати POST-запит на /AssistantChat/ask з відсутнім полем message. { "userId": "680cc5ad9c376cfd07332f06", "isTaskRequest": true }	Статус 400. Повідомлення про валідаційні помилки: The Message field is required., Message is too short.	Статус 400. Повернуто валідне повідомлення з помилками в полі Message.	Високий	Pass	Swagger (AssistantChat)	22.05.2025

Рисунок 4.6 – Тест-кейси для перевірки роботи з помічником через Swagger UI

ID	Опис	Умови	Кроки виконання тестів	Очікуваний результат	Фактичний результат	Пріорітет	Результат тесту	Оточення	Дата створення
TC-01	Вхід з правильним email і паролем	Користувач з email john@example.com і паролем Password123 існує	1. Відправити POST на /login з {email: john@example.com, password: Password123}2. Перевірити наявність JWT токена	1. Відповідь з токеном, статус 200 OK2. Тіло відповіді містить ключ token	1. Токен отримано, статус 2002. Токен міститься в відповіді	Високий	Pass	Тестування API через Swagger (http://localhost:5186/swagger/index.html) Auth	10.04.2025
TC-02	Вхід з email, який не існує в системі	Користувача з email jon@example.com не існує	1. Відправити POST на /login з {email: jon@example.com, password: Password123}	Відповідь має статус 401, повідомлення: "User with such email does not exist"	Помилка з повідомленням, статус 401	Високий	Pass	Тестування API через Swagger (http://localhost:5186/swagger/index.html) Auth	10.04.2025
TC-03	Вхід з паролем, який не задовольняє мінімальні вимоги безпеки	Пароль містить менше 8 символів	1. Відправити POST на /login з {email: john@example.com, password: short}	Валідаційна помилка: "The field Password must be a string or array type with a minimum length of '8'." Статус 400	Повідомлення про короткий пароль, статус 400	Високий	Pass	Тестування API через Swagger (http://localhost:5186/swagger/index.html) Auth	10.04.2025
TC-04	Вхід з неправильним email та паролем	Відсутні	1. Відправити POST на /login з {email: johnexam, password: Pass}	Помилки валідації для email і пароля, статус 400	Повідомлення про неправильний формат email та короткий пароль, статус 400	Високий	Pass	Тестування API через Swagger (http://localhost:5186/swagger/index.html) Auth	10.04.2025
TC-05	Вхід з правильним email, але неправильним паролем	Користувач john@example.com існує	1. Відправити POST на /login з {email: john@example.com, password: Password12}	Помилка: "Invalid password", статус 401 Unauthorized	Повідомлення про неправильний email або пароль, статус 401	Високий	Pass	Тестування API через Swagger (http://localhost:5186/swagger/index.html) Auth	10.04.2025

Рисунок 4.7 – Тест-кейси для перевірки авторизації через Swagger UI

ID	Опис	Умови	Кроки виконання тестів	Очікуваний результат	Фактичний результат	Пріоритет	Результат тесту	Оточення	Дата створення
TC-01	Отримання всіх папок користувача за існуючим ID	у користувача з id 680902753b4cf31ae537e9c6 є дві папки	Надіслати GET-запит на /api/folder/user/680902753b4cf31ae537e9c6	Список із 2 об'єктів папок, статус 200 OK	Список із 2 об'єктів папок, статус 200 OK	Високий	Pass	Тестування API через Swagger (http://localhost:5186/swagger/index.html) Folder	02.05.2025
TC-02	Отримання папок за неіснуючим UserId	користувач з таким id не існує	Надіслати GET-запит на /api/folder/user/000000000000000000000000	Порожній список, статус 200 OK	Порожній список, статус 200 OK	Високий	Pass	Тестування API через Swagger (http://localhost:5186/swagger/index.html) Folder	02.05.2025
TC-03	Отримання папки за існуючим ID	Папка з id існує 6813de9b91126382a3a6b291	Надіслати GET-запит на /api/folder/6813de9b91126382a3a6b291	Об'єкт з name = "Folderr1", статус 200 OK	json { "id": "6813de9b91126382a3a6b291", "userId": "680902753b4cf31ae537e9c6", "name": "Folderr1", "createdAt": "..." }	Високий	Pass	Тестування API через Swagger (http://localhost:5186/swagger/index.html) Folder	02.05.2025
TC-04	Отримання папки за неіснуючим ID	папка з таким id не існує	Надіслати GET-запит на /api/folder/000000000000000000000000	Статус 404 Not Found	Статус 404 Not Found	Високий	Pass	Тестування API через Swagger (http://localhost:5186/swagger/index.html) Folder	02.05.2025
TC-05	Створення папки з валідними даними	існує користувач з id 680902753b4cf31ae537e9c6	Надіслати POST на /api/folder з тілом: {"name": "New Folder", "userId": "680902753b4cf31ae537e9c6"}	Статус 200 OK, відповідь містить token	Статус 200 OK, об'єкт token з параметрами	Високий	Pass	Тестування API через Swagger (http://localhost:5186/swagger/index.html) Folder	02.05.2025
TC-06	Оновлення папки з валідним ID	Папка з id існує 6813de9b91126382a3a6b291	Надіслати PUT на /api/folder/6813de9b91126382a3a6b291 з тілом: { "name": "Updated" }	Статус 204 No Content, зміни видно в базі	Статус 204 No Content, зміни видно в базі	Високий	Pass	Тестування API через Swagger (http://localhost:5186/swagger/index.html) Folder	02.05.2025
TC-07	Видалення папки з неіснуючим ID	папка з таким id не існує	Надіслати DELETE на /api/folder/000000000000000000000000	Статус 404 Not Found	Статус 404 Not Found	Високий	Pass	Тестування API через Swagger (http://localhost:5186/swagger/index.html) Folder	02.05.2025

Рисунок 4.8 – Тест-кейси для перевірки роботи з папками через Swagger UI

Аналогічно провели тестування й для інших контролерів на сервері.

4.1.5 Перевірка продуктивності сервера

Було проведено навантажувальне тестування (НТ) серверної частини системи за допомогою Apache JMeter з метою оцінки її продуктивності та стабільності під різними рівнями навантаження. Тестування здійснювалось на локальній машині з операційною системою Windows 11 Home, процесором AMD Ryzen 7 5700U with Radeon Graphics (1.80 GHz), 16 ГБ оперативної пам'яті, SSD-диском об'ємом 512 ГБ та підключенням до локальної мережі. Серверна частина запускалась у середовищі розробки Visual Studio 2022.

API-запит типу GET за адресою /api/Friends/${userId}/potential-friends, де userId підставлявся з даних, взятих з Excel-файлу є одним із складних для виконання на сервері. Всі тести проводилися на локальному сервері, що працює на порту 5186. Кількість користувачів (кор.) змінювалась під час тестів, але розподіл навантаження відбувався за схемою поступового збільшення (ramp up period) протягом 10 секунд. Кожен користувач виконував 10 циклів (loop count 10) запитів до сервера (таблиця 4.1).

Таблиця 4.1 – Результати НТ для «Потенційних друзі»

Метрика	Тест 1 (50 кор., 6 ID)	Тест 2 (300 кор., 6 ID)	Тест 3 (500 кор., 6 ID)	Тест 4 (800 кор., 6 ID)	Тест 5 (1000 кор., 6 ID)
# Запитів	500	3000	5000	8000	10 000
Error %	0.00%	0.00%	0.00%	0.00%	1.32%
Apdex	1.000	0.097	0.026	0.016	0.042
Мін. Час (ms)	8	216	216	29	14
Середній час (ms)	17.76	2505.56	4955.25	9550.12	15 621.50
Макс. час (ms)	224	6564	14 177	54 744	144 733
Медіана (ms)	14	2796	5418	9832	155 394
Throughput (req/s)	51.60	88.54	84.19	69.66	55.03

Аналіз результатів навантажувального тестування демонструє, що система здатна ефективно обробляти запити при середньому рівні навантаження, забезпечуючи високу стабільність і надійність у перших чотирьох тестах. Значення

Error% залишалося на рівні 0.00% до моменту, коли кількість одночасних користувачів досягла 800, що свідчить про впевнене утримання якості обробки запитів у широкому діапазоні навантаження (див. додаток Г, рис. Г.1–Г.2). Індекс задоволеності користувачів Apdex у Тесті 1 становив максимальне значення 1.0, що підтверджує миттєву та стабільну реакцію сервера при помірному навантаженні. Навіть при збільшенні кількості користувачів до 300 і 500, система зберігала працездатність, обробляючи тисячі запитів без збоїв, що говорить про її стійкість до високої інтенсивності використання.

Пропускна здатність залишалася стабільною на рівні понад 80 запитів за секунду при навантаженні до 500 користувачів, що дозволяє говорити про добру масштабованість системи в межах заданих параметрів. Зростання середнього та максимального часу відповіді при екстремальних навантаженнях (800 і 1000 користувачів) вказує на межу поточної конфігурації сервера, але разом із цим підкреслює, що система змогла обробити всі 10 000 запитів у найважчому сценарії. Це свідчить про потенціал до оптимізації й масштабування, що відкриває можливості для покращення продуктивності за допомогою відповідних технічних рішень.

Розглянемо ще один запит типу GET api/Category/${categoryId} та його навантажувальне тестування (див. табл. 4.2).

Таблиця 4.2 – Результати НТ для «Категорії»

Метрика	Тест 1 (500 кор., 1 ID)	Тест 2 (500 кор., 6 ID)
# Запитів	5000	5000
Error %	0.00%	0.00%
Apdex	0.218	0.180
Мін. Час (ms)	21	8
Середній час (ms)	1798.69	2371.21
Медіана (ms)	1808	2340.50
Throughput (req/s)	185.47	152.99

Аналіз результатів навантажувального тестування свідчить про стабільну роботу системи при одночасному використанні 500 користувачів, незалежно від кількості активних ідентифікаторів запитів. В обох тестах рівень помилок

залишився на нульовому рівні, що свідчить про високу надійність обробки запитів та відсутність критичних збоїв під час навантаження. Це є вагомою перевагою системи, адже навіть при зміні складності запитів (від одного до шести ID) вона не втрачає здатності коректно відповідати на всі звернення.

Мінімальний час відповіді залишається дуже низьким (8–21 мс), що демонструє потенціал системи до швидкої реакції в ідеальних умовах. Попри зростання середнього часу відповіді при збільшенні кількості ID, система продовжує ефективно обслуговувати запити, підтримуючи стабільну пропускну здатність: у Тесті 1 вона становила 185.47 запитів на секунду, а у Тесті 2 – 152.99.

Було проведено ще навантажувальні тести інших запитів, результати свідчать про загальну стабільність і надійність системи: при помірному та високому навантаженні вона впевнено обробляє велику кількість запитів без збоїв, демонструючи хорошу масштабованість та стійкість до ускладнення запитів. Навіть за умов зростання складності та кількості звернень система зберігає працездатність, що підтверджує її готовність до подальшого використання та оптимізації в реальному середовищі.

4.2 Заходи безпеки

У системі «Naviria» безпека даних є пріоритетом. Конфіденційні ключі для підключення до бази даних, чату та інших сервісів зберігаються у секретному вигляді і недоступні кінцевим користувачам, що підвищує рівень безпеки системи. Без їх введення розробник не зможе запустити сервер, що запобігає випадковому використанню неправильних або відсутніх ключів.

База даних реалізована на MongoDB. Для зберігання та трансформації зображень використовується Cloudinary. Додатково система взаємодіє з зовнішніми API, такими як OpenAI GPT для реалізації функцій AI-помічника та Google OAuth для авторизації через сторонні сервіси.

Зокрема, паролі користувачів хешуються на сервері за допомогою стандартного інтерфейсу IPasswordHasher<TUser>, який реалізує алгоритми безпечного хешування та перевірки паролів, забезпечуючи захист від

несанкціонованого доступу. Під час тестування реєстрації нового користувача було перевірено формат збереження даних у базі даних, що відповідає дійсності, пароль захешований.

Для забезпечення безпечної передачі даних між клієнтом та сервером використовується протокол HTTPS, хоча система підтримує й HTTP. Кожен запит до серверної частини вимагає наявності токена авторизації, що гарантує, що доступ мають лише автентифіковані користувачі.

Під час ручного тестування було виявлено, що вхід до застосунків можливий лише для авторизованих користувачів, тоді як неавторизовані користувачі мають обмежений доступ – вони можуть тільки зареєструвати новий акаунт або увійти до вже існуючого профілю. Таким чином, система забезпечує захист даних користувачів та безпечну взаємодію з сервером.

4.3 Тестування клієнтської частини

Для перевірки коректної роботи клієнтської частини вебзастосунку було проведено автоматизоване тестування з використанням інструментів Cypress для вебплатформи. Проєкт було відкрито локально, після чого, за допомогою команди npx cypress open, було запущено режим E2E Testing, який дав змогу виконувати автоматизовані тести у браузерах Chrome та Edge, що забезпечило перевірку коректної роботи інтерфейсу в різних середовищах виконання.

Основна увага приділялася перевірці наявності необхідних елементів на сторінках, коректності відображення змін на інтерфейсі, валідації форм, навігаційних переходів і відповідності статичних елементів очікуваним результатам.

Одним із прикладів таких тестів є набір негативних тестів для перевірки валідації форми реєстрації, розроблений за тест-кейсами. Тестування охоплює різні помилкові сценарії, зокрема введення некоректних імен, прізвищ, поштових адрес, слабких паролів, розбіжностей у паролях, відсутності вибору статі, недостатнього віку користувача та помилок у форматі нікнейму. Для кожного кейсу перевіряється наявність відповідного повідомлення про помилку або правильна поведінка

інтерфейсу (наприклад, обмеження довжини нікнейму до 20 символів). Таким чином, тести забезпечують перевірку надійності клієнтської валідації введених даних (див. рис. 4.9).

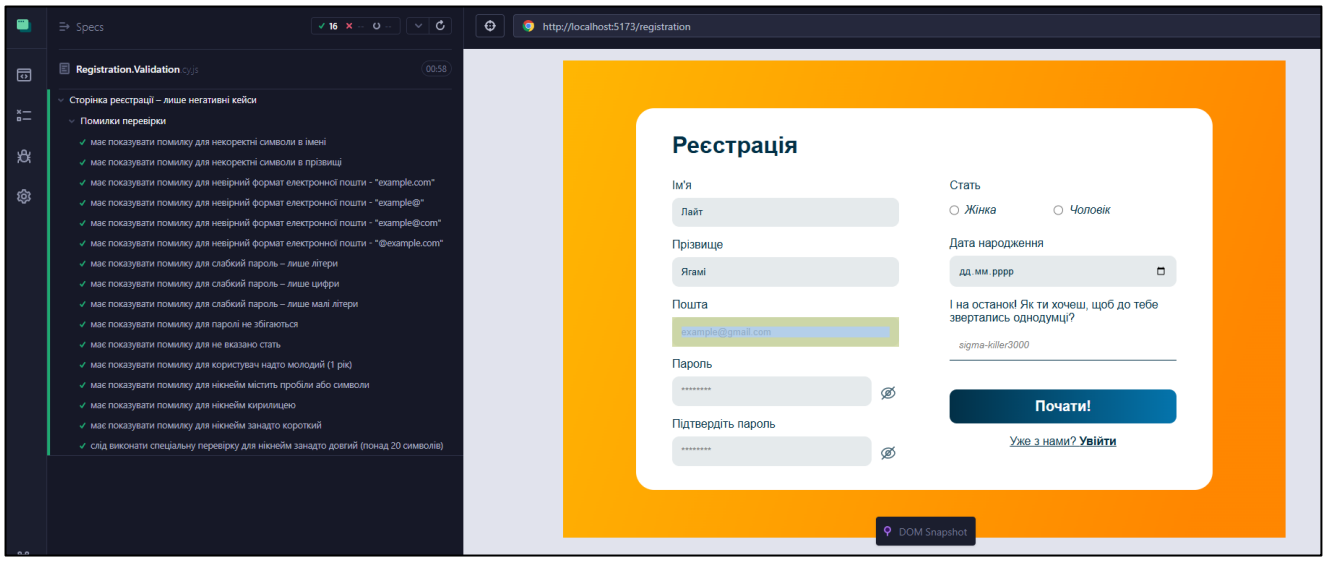

Рисунок 4.9 – Негативні кейси для сторінки «Реєстрація»

Було розроблено тести, що перевіряють коректну роботу сторінки статистики: чи відображається кругова діаграма при перемиканні між режимами статистики, чи показуються основні числові показники, такі як кількість користувачів і задач, чи правильно формується таблиця лідерів із відповідною структурою, а також чи відображається лінійний графік виконаних задач за місяцями (див. рис. 4.10).

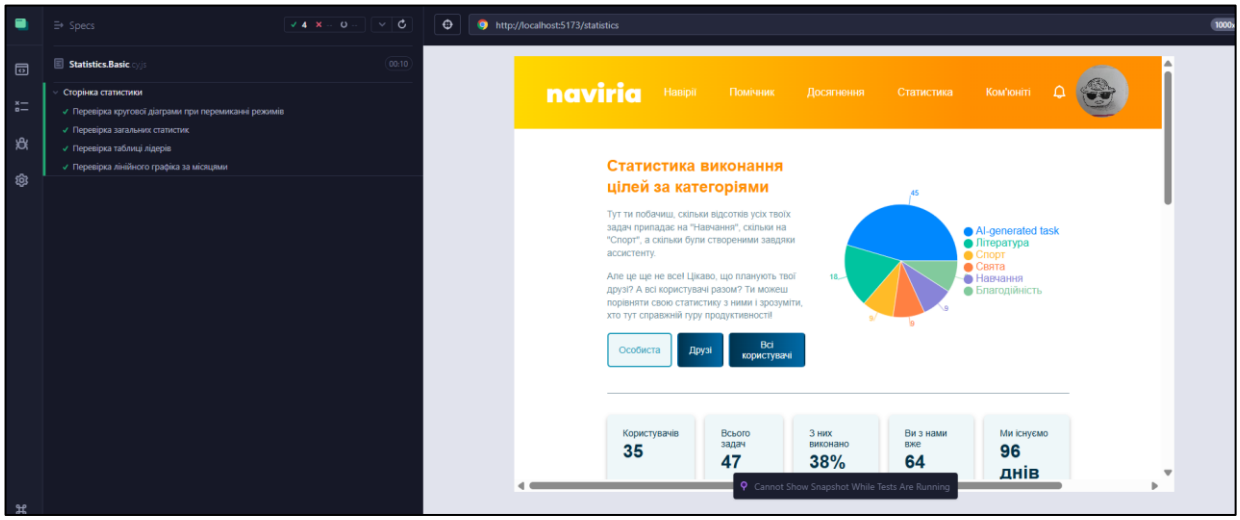

Рисунок 4.10 – Тестування сторінки «Статистика»

Провели тестування взаємодії з помічником, а саме, він створює автоматично задачу за запитом користувача. Реалізували сценарії, що при виборі опції створення задачі неможливо відправити порожнє повідомлення, а також при введенні валідного запиту помічник створює задачу (див. рис. 4.11), яка згодом з'являється в папці «Generated tasks» зі змістом, що відповідає ключовим словам із запиту (див. рис. 4.12).

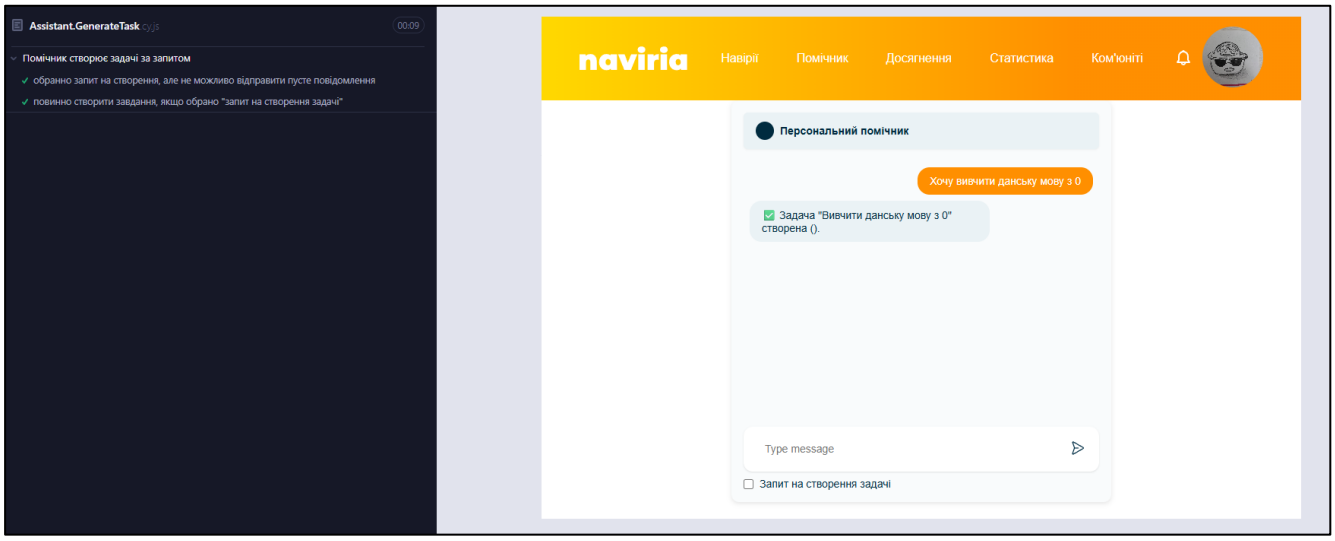

Рисунок 4.11 – Тестування чату з помічником

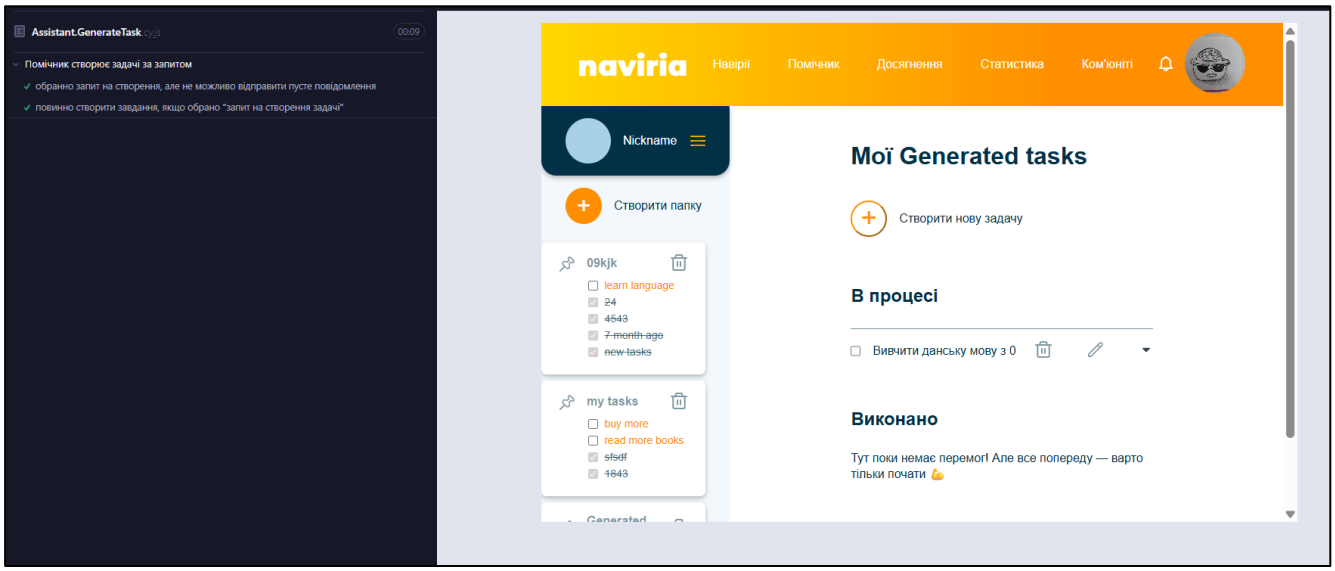

Рисунок 4.12 – Перевірка створення задачі у «Generated tasks»

Розглянемо приклади ручного тестування, яке проводилося за допомогою браузера Google Chrome (версія 136) на операційній системі Windows 11. Було

перевірено функціональність сторінки «Ком’юніті» для авторизованих користувачів, зокрема: перехід на сторінку, наявність та коректність відображення блоку «Знайти друзів», вивід нікнеймів, рівнів та описів користувачів, а також роботу пошуку й фільтрації. Тестування охоплювало 12 тест-кейсів і включало перевірку інтерфейсних елементів, логіки відображення користувачів і дій (пошук, фільтрація, запит у друзі). Усі тести були виконані вручну, результати збіглися з очікуваними, тести пройдені успішно (див. додаток Д, рис. Д.1–Д.2).

Ще один приклад тестування – перевірка функціональності блоку «Запити» на сторінці «Ком’юніті». Тестування охоплює різні аспекти відображення та взаємодії з користувацькими запитами. Зокрема, перевіряється, чи коректно відображаються категорії у випадаючому списку. Користувач, будучи авторизованим, відкриває список, і в ньому мають з’явитися відповідні категорії, такі як «AI-generated tasks», «Література», «Спорт», «Навчання», «Благодійність» та «Свята».

Окрім цього, важливо перевірити, як саме відображаються запити: вони повинні містити нікнейм користувача, рівень, опис (якщо він є), або напис «Опис відсутній». Додатково тестується наявність та зовнішній вигляд кнопок «Відхилити» та «Прийняти» – кожен запит повинен мати обидві, при цьому перша має бути червоного кольору, а друга – темно-синього. У разі натискання кнопки «Відхилити» відповідний запит має зникнути зі списку. Якщо натиснути «Прийняти», то користувач, який надіслав запит, має зникнути зі списку запитів і з’явитися у боковій панелі «Мої друзі» (див. додаток Д, рис. Д.3).

У ході тестування функціональності створення сторінки «Навірії» було перевірено ключові можливості додавання нових папок та задач із підзадачами (див. додаток Д, рис. Д.4). Спочатку протестовано сценарій створення нової папки. За умови, що користувач був авторизований і перебував на сторінці «Навірії», після натискання на значок плюсика, введення назви папки «Нова папка» та натискання кнопки «Зберегти», папка успішно з’явилася в списку, що підтверджує коректну роботу даної функції.

Після цього було виконано більш складний сценарій створення задачі з підзадачами всередині щойно створеної папки. Користувач відкрив папку, побачив кнопку для додавання задачі, після чого натиснув на неї – система вивела форму для створення задачі. У відповідні поля було введено назву задачі «Почати грати на піаніно», опис «Навчитись грати на піаніно», обрано категорію «Навчання» та додано тег «піаніно». Також було встановлено дедлайн через календар та вказано пріоритет задачі. Після вибору типу задачі «З підзадачами» з'явилася форма для додавання підзадач.

У першій підзадачі з назвою «Вивчити ноти» було обрано тип «Повторювана» та вказано дні Понеділок, Вівторок і П'ятницю. Потім була додана друга підзадача «Вивчити сонет», тип якої – «Шкала». У цій формі було заповнено поля одиниці вимірювання («сонети») та цілі (5). На завершення була додана третя підзадача «Знайти вчителя» типу «Звичайна». Після натискання кнопки «Зберегти» у колонці «В процесі» з'явилась створена задача з усіма заповненими параметрами. Усі тести пройшли успішно, очікувані результати повністю відповідали фактичним, що підтверджує стабільність та правильність роботи сторінки створення «Навірії» у зазначеному середовищі.

Аналогічно було протестовано й інші сторінки додатку та їхні елементи, зосередившись на перевірці основної функціональності, коректного відображення даних і взаємодії з користувачем. До основних охоплених сценаріїв тестування також входили: отримання досягнень, перегляд сповіщень, редагування профілю користувача, підтримка друзів, а також створення й редагування усіх типів задач і підзадач.

Під час тестування із використанням Apache JMeter сторінки «Статистика» було проведено п'ять серій тестів із поступовим збільшенням кількості одночасних користувачів від 50 до 1000. Вебсайт було запущено локально на порті 5173, що дозволило змоделювати навантаження в умовах розробки. Для всіх тестів був налаштований період поступового нарощування навантаження (ramp-up) у 5 секунд та цикл повторення (loop count) у 10, а також використовувалися актуальні токени авторизації для забезпечення доступу до захищеної сторінки (див. табл. 4.3).

Таблиця 4.3 – Результати НТ для сторінки «Статистики»

Метрика	Тест 1 (50 кор.)	Тест 2 (300 кор.)	Тест 3 (400 кор.)	Тест 4 (800 кор.)	Тест 5 (1000 кор.)
# Запитів	500	3000	4000	8000	10 000
Error %	0.00%	0.00%	0.00%	0.00%	5.40%
Apdex	1.000	1.000	0.999	0.516	0.270
Мін. Час (ms)	2	9	42	161	1
Середній час (ms)	4.37	153.09	313.86	1115.09	1276.88
Макс. час (ms)	59	240	517	2374	3708
Медіана (ms)	4.00	159.00	343.00	1201.00	1390
Throughput (req/s)	104.21	504.12	529.73	536.05	574.05

Результати показують, що вебсайт стабільно обробляв запити при збільшенні навантаження до 400 користувачів з нульовим відсотком помилок та високим рівнем задоволеності користувачів (Apdex близько 1.0). Це свідчить про дуже добре оптимізовану роботу сервера та мережевої частини системи, які забезпечують швидкий відгук (середній час відповіді не перевищував 314 мс при 400 користувачах). Медіанні та максимальні значення часу відповіді також були у прийнятних межах, що підтверджує відсутність затримок у продуктивності.

Throughput (пропускна здатність) системи демонстрував зростання до 536 запитів за секунду при 800 користувачах, що вказує на ефективну обробку одночасних запитів та стабільність сервера навіть під досить великим навантаженням.

Варто відзначити, що при найвищому навантаженні у 1000 користувачів відсоток помилок зріс до 5.4%, а Apdex впав до 0.27, що свідчить про початок перевантаження системи (див. додаток Г, рис. Г.3–Г.4). Проте середній час відповіді у 1277 мс і throughput понад 570 запитів за секунду також свідчать про те, що сервер продовжує обробляти запити з високою пропускною здатністю, незважаючи на зростання затримок.

4.4 Тестування мобільного застосунку

Функціонал мобільного застосунку загалом відповідає функціоналу вебверсії системи, що дозволило використовувати частково схожі сценарії тестування.

Під час ручного тестування мобільного застосунку особливу увагу приділяли перевірці коректного відображення списку друзів: актуалізація інформації після додавання чи видалення користувачів, відображення фотографій, та доступність дій, таких як надсилання запиту чи скасування дружби.

Окрім того, вручну тестувалася функціональність редагування профілю, зокрема можливість змінювати особисті дані, завантажувати власне фото та перевіряти їхнє коректне відображення в інтерфейсі (див. додаток Д, рис. Д.5). Було перевірено роботу із задачами (див. додаток Д, рис. Д.6). Також перевірялась робота шкали прогресу, яка дозволяє користувачу відстежувати свій рівень досягнень у застосунку, що є важливим елементом персоналізації та мотивації.

Тестування проводилось як на емуляторі Android Studio, так і на фізичному мобільному пристрої (Samsung Galaxy A52), що дало змогу забезпечити більшу повноту перевірки та виявлення помилок, пов’язаних із реальними умовами використання.

За допомогою використання Android Studio Profiler, визначили, що дані з View Live Telemetry показують низьке навантаження CPU з боку застосунку (0%) та активність 43 потоків, зокрема RenderThread і DefaultDispatch. Загальне споживання пам’яті становить 162,7 МБ, найбільше займають код, Java-об'єкти та native-компоненти (див. додаток Е, рис. Е.1).

Результати використання пам’яті (Memory Consumption демонструють, що найбільше пам’яті під час виконання додатку споживають компоненти, пов’язані з JIT-компіляцією (art::JitCompileTask::Run) та рендерингом UI (RenderThread) (див. додаток Е, рис. Е.2). Таблиця з виділеннями пам’яті підтверджує, що частина об'єктів не була звільнена, зокрема android::Bitmap::allocateHeapBitmap, що може бути потенційною причиною витоків (див. додаток Е, рис. Е.3). У той же час, значна частина пам’яті успішно звільняється, що свідчить про належну роботу збирача сміття та може вказувати на загалом стабільну роботу застосунку.

4.5 Життєвий цикл дефекту

Життєвий цикл дефекту розпочався з того, що під час тестування функціоналу входу через Google було виявлено дефект, який отримав статус «Новий». Далі цей дефект було призначено розробнику серверної частини (див. табл. 4.4).

Таблиця 4.4 – Таблиця опису дефекту BUG-004

Поле	Опис
Ідентифікатор дефекту	BUG-004
Короткий опис	Помилка авторизації через Google: redirect_url_mismatch
Кроки для відтворення	1. Запустити сервер (http)
	2. Запустити вебсайт
	3. Відкрити сторінку входу: localhost:5173/login
	4. Натиснути кнопку «Вхід через Google»
Очікуваний результат	Користувач успішно авторизується через Google-акаунт.
Фактичний результат	Помилка 400: redirect_url_mismatch.
Пріоритет	1 – Обов’язково виправити
Компонент	Авторизація / Сервер
Мітки (Labels)	auth, google_login, critical
Коментарі	Користувач вже зареєстрований з цією поштою. Помилка виникає на боці серверу при неправильній конфігурації OAuth.
Докази (Screenshots)	Дивитись рисунок 4.13
Виконавець (Assignee)	Призначено розробнику сервера (Червенко А.)
Серйозність	2 – Висока
Логи	У консолі панелі розробника: the given origin is not allowed for the given client ID
Час тестування	10:00 07.05.2025
Тестове середовище	ОС: Windows 11, Браузер: Chrome 136.0.7103.114, Сервер: ASP.NET Core (.NET 8), Frontend: React
Тестові дані	Google акаунт: mariam.almakadma.85@gmail.com
Статус	Призначено

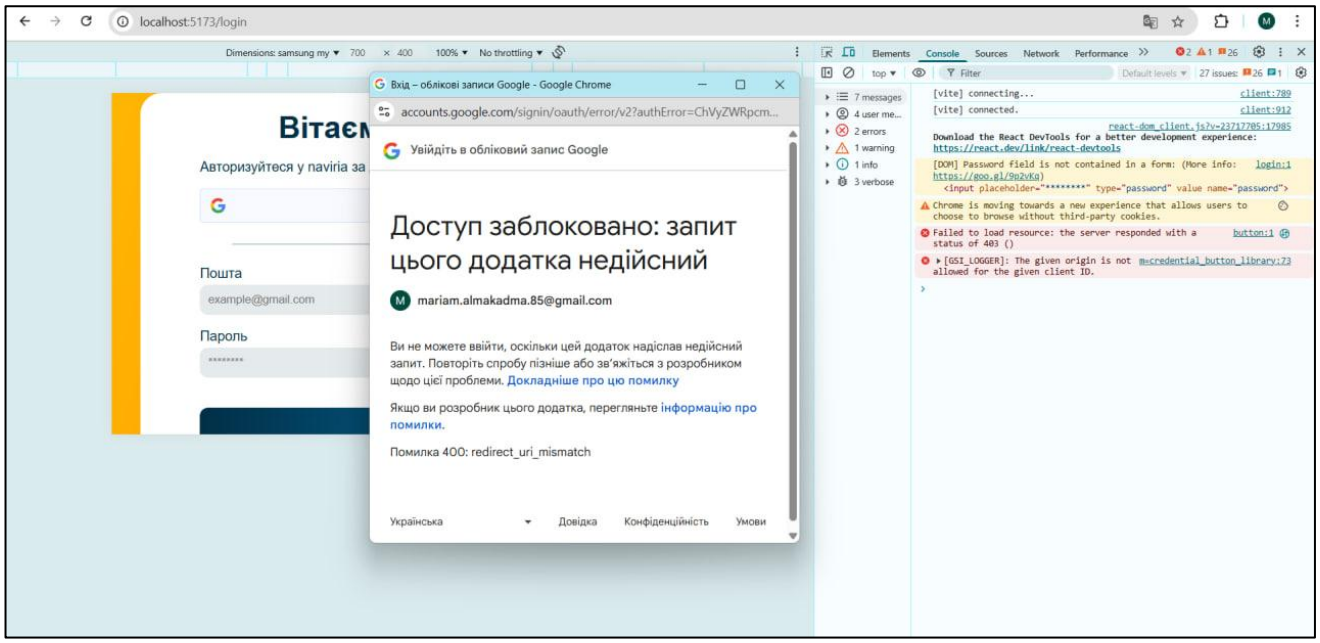

Рисунок 4.13 – Знімок екрана помилки при авторизації BUG-004

Розробник виявив причину проблеми та додав правильну адресу до списку дозволених у Google API Console у полі OAuth 2.0 Redirect URI. Після внесення змін дефект отримав статус «Вирішена». Наступним етапом стало повторне тестування, яке підтвердило, що авторизація через Google тепер працює коректно. Після цього дефект отримав статус «Перевірена». Завершальним етапом було закриття дефекту, оскільки він був остаточно усунений.

Ще один життєвий цикл дефекту розпочався з того, що під час тестування функціоналу надсилання запиту на додавання в друзі було виявлено дефект, який отримав статус «Новий».

Після цього дефект було призначено розробнику серверної частини для аналізу та усунення (див. табл. 4.5). У ході дослідження з'ясувалося, що після надсилання запиту на дружбу користувач не виключається зі списку потенційних друзів у відповідь сервера.

Таблиця 4.5 – Таблиця опису дефекту BUG-007

Поле	Опис
Ідентифікатор дефекту	BUG-007
Короткий опис	Повторна поява користувача після надсилання запиту на дружбу

Кінець таблиці 4.5

Поле	Опис
Кроки для відтворення	1. Авторизуватись
	2. Перейти до сторінки «Ком'юніті»
	3. Відкрити вкладку «Знайти друзів»
	4. Натиснути кнопку «Додати друга» біля будь-якого користувача
	5. Оновити сторінку
Очікуваний результат	Після натискання кнопки «Додати друга» користувач має зникнути зі списку, і не з’являтись знову після оновлення.
Фактичний результат	Користувач зникає зі списку, але після оновлення сторінки знову з’являється, і можна повторно надіслати запит.
Пріоритет	2 – Бажано виправити
Компонент	Список потенційних друзів / Сервер
Мітки (Labels)	friends, backend, logic_error
Коментарі	Дані про відправлений запит не враховуються при генерації списку потенційних друзів. Потрібно змінити серверну логіку фільтрації.
Докази (Screenshots)	Дивитись рисунок 4.14–4.16
Виконавець (Assignee)	Призначено розробнику сервера (Червенко А.)
Серйозність	3– Середня
Логи	Немає помилок у консолі, проблема в логіці на сервері
Час тестування	18:47 23.05.2025
Тестове середовище	ОС: Windows 11, Браузер: Chrome 136.0.7103.114, Сервер: ASP.NET Core (.NET 8), Frontend: React
Тестові дані	Акаунт: alexander.davis@example.com
Статус	Призначено

Рисунок 4.14 показує початковий стан екрана до натискання кнопки «Додати друга», після чого на рисунку 4.15 видно, що користувач зник зі списку. Рисунок 4.16 ілюструє оновлений список друзів після перезавантаження сторінки, де обраний користувач все ще у списку.

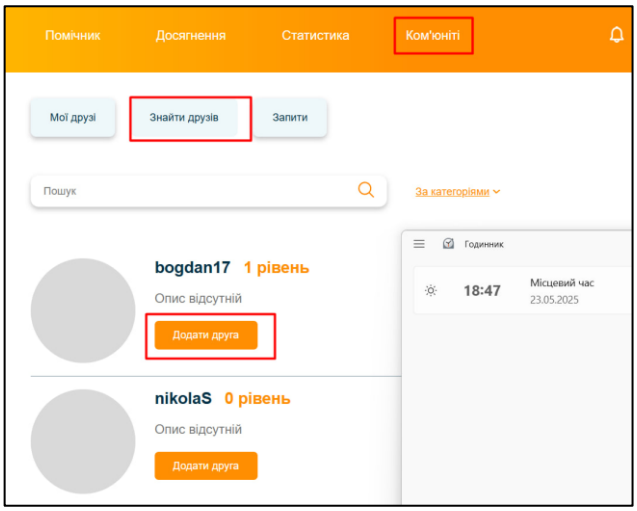

Рисунок 4.14 – Знімок екрана до натискання кнопки «Додати друга»

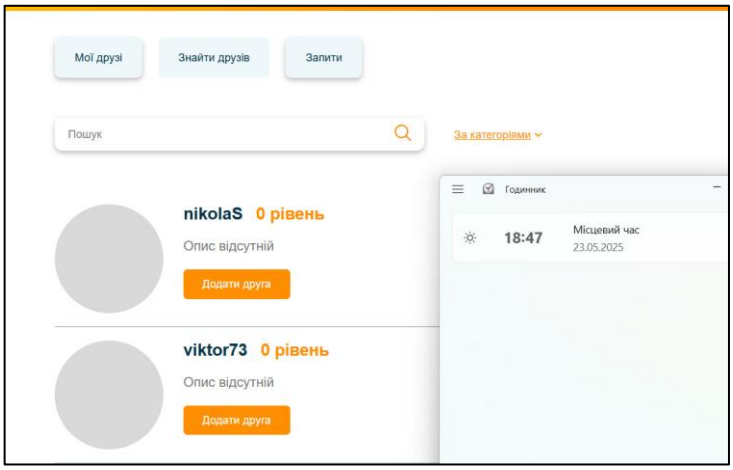

Рисунок 4.15 – Знімок екрана після натискання кнопки «Додати друга»

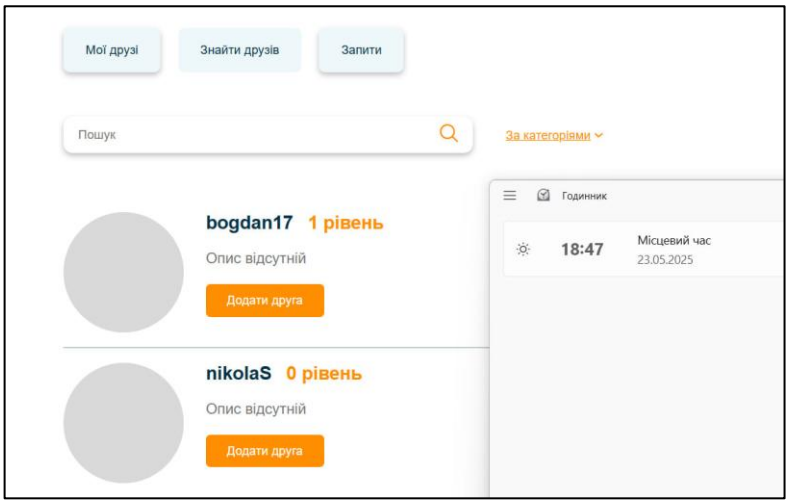

Рисунок 4.16 – Знімок екрана після оновлення сторінки

Розробник оновив логіку серверної частини, щоб обробляти та фільтрувати користувачів, яким уже надіслано запит. Після впровадження змін дефект отримав статус «Вирішено». Провели повторне тестування, яке підтвердило, що після додавання в друзі користувач більше не з’являється у списку після оновлення сторінки. Після цього дефект отримав статус «Перевірений», а згодом – «Закритий».

Життєвий цикл дефекту розпочався з того, що під час тестування персонального помічника на сторінці «Помічник» через вебсайт було виявлено помилку у відображенні відповіді: повідомлення помічника відображалося у стилі користувача, а не у вигляді окремого діалогового вікна (див. табл. 4.6). Дефект отримав статус «Новий» та був переданий фронтенд-розробнику.

Таблиця 4.6 – Таблиця опису дефекту BUG-008

Поле	Опис
Ідентифікатор дефекту	BUG-008
Короткий опис	Неправильне відображення відповіді помічника – стиль відповіді як у користувача
Кроки для відтворення	1. Авторизуватись
	2. Перейти на сторінку «Помічник»
	3. Написати повідомлення у полі для вводу
	4. Натиснути кнопку відправити
Очікуваний результат	Відповідь від помічника відображається у вигляді діалогового вікна, в іншому кольорі та зі стилем системного повідомлення
Фактичний результат	Відповідь помічника відображається як повідомлення користувача: той самий колір і стиль
Пріоритет	2 – Бажано виправити
Компонент	Інтерфейс / Помічник
Мітки (Labels)	UI, assistant, styling
Коментарі	Функціональність працює: відповіді формуються правильно, але проблема у стилізації повідомлень
Докази (Screenshots)	Дивитись рисунок 4.17
Виконавець (Assignee)	Призначено фронтенд-розробнику (Дашко Н.)

Кінець таблиці 4.6

Поле	Опис
Серйозність	4 – Низька
Логи	Відсутні
Час тестування	8:45 24.05.2025
Тестове середовище	ОС: Windows 11, Браузер: Chrome 136.0.7103.114, Сервер: ASP.NET Core (.NET 8), Frontend: React
Тестові дані	Акаунт: alexander.davis@example.com Тестовий запит: що таке поліморфізм
Статус	Призначено

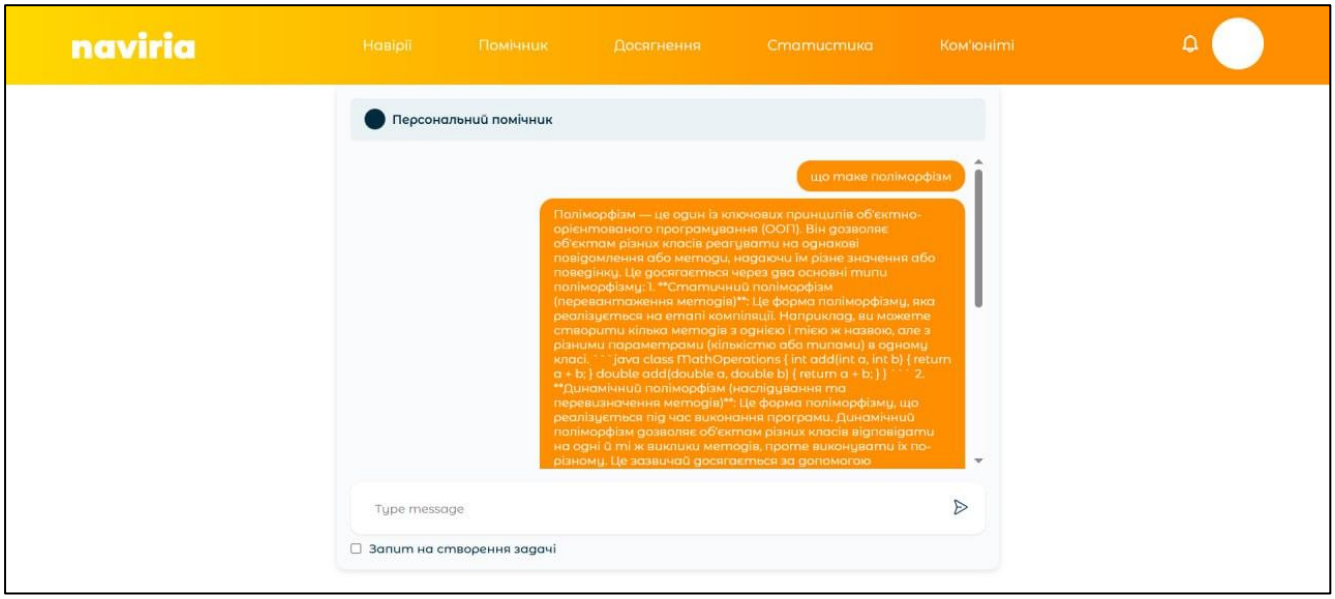

Рисунок 4.17 – Знімок екрана для візуалізації помилки BUG-008

Після внесення змін до стилів інтерфейсу та коректного розмежування повідомлень, дефект отримав статус «Вирішений». Проведене повторне тестування підтвердило, що відповідь помічника тепер відображається належним чином – в окремому стилізованому блоці з іншої сторони вікна для діалогу. Після цього дефект отримав статус «Перевірений», а згодом – «Закритий».

Усі дефекти були описані відповідно до шаблону, що гарантує зручність аналізу і визначені пріоритету.

ВИСНОВКИ

Результатом роботи є комплексне тестування багатокомпонентної програмної системи, що охоплює серверну, клієнтську частини, та мобільний застосунок. Було проведено аналіз предметної галузі, що включав вивчення сучасних рішень у сфері самоорганізації, керування часом та особистої ефективності. У результаті дослідження були виявлені ключові проблеми, наприклад, обмежений функціонал, відсутність персоналізації, слабка гейміфікація та недостатній соціальний компонент. Це підтвердило актуальність створення інноваційної системи, яка поєднує адаптивність, гейміфікацію, соціальну взаємодію та інтелектуальні підказки.

На основі аналізу було сформовано набір функціональних вимог до системи, що охоплюють всі основні модулі: реєстрацію, цілі, досягнення, візуалізацію прогресу, взаємодію з друзями та інтеграцію зі штучним інтелектом. Було розроблено тестовий план, створено чек-листи й тест-кейси, що охоплюють функціональні, нефункціональні, навантажувальні, інтеграційні та регресійні тести. У процесі перевірки використовувалися як ручні, так і автоматизовані тести з використанням сучасних фреймворків та інструментів. Зокрема, функціональні сценарії перевіряли основну логіку взаємодії користувача з системою, а нефункціональні – такі характеристики, як зручність використання, час відгуку та безпека

У результаті тестування було підтверджено, що система стабільно виконує свої функції, демонструє коректну поведінку при стандартному й піковому навантаженні, відповідає встановленим вимогам до якості, продуктивності та надійності, що засвідчує її готовність до впровадження та використання кінцевими користувачами.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Stojanovic, M., Grund, A., & Fries, S. App-Based Habit Building Reduces Motivational Impairments During Studying – An Event Sampling Study. Frontiers in Psychology. – 2020. – Vol. 11. – P. 1–15. [Електронний ресурс] – URL: https://www.frontiersin.org/journals/psychology/articles/10.3389/fpsyg.2020.00167/full (дата звернення: 08.04.2025).

2. Lampropoulos, G., Kinshuk Virtual reality and gamification in education: a systematic review. Education Tech Research Dev. – 2024. – Vol. 72. – P. 1691–1785. [Електронний ресурс] – URL: https://doi.org/10.1007/s11423-024-10351-3 (дата звернення: 08.04.2025).

3. Nicholas, J. C., Ntoumanis, N., Smith, B. J., Quested, E., Stamatakis, E., & Thøgersen-Ntoumani, C. Development and feasibility of a mobile phone application designed to support physically inactive employees to increase walking. BMC Medical Informatics and Decision Making. – 2021. – Vol. 21(1), Article number: 23. [Електронний ресурс] – URL: https://doi.org/10.1186/s12911-021-01391-3 (дата звернення: 08.04.2025).

4. Cvetkovic, P., Harbord, C., & Hlavacs, H. A Study on Gamification Effectiveness. Proceedings of the 6th International Conference on Computer-Human Interaction Research and Applications (CHIRA). – 2020. – [Електронний ресурс] – URL: DOI: https://doi.org/10.5220/0009340102360244 (дата звернення: 21.05.2025).

5. Hutmacher, F., & Appel, M. The Psychology of Personalization in Digital Environments: From Motivation to Well-Being – A Theoretical Integration. Review of General Psychology. – 2023. – Vol. 27, № 1. – P. 26–40. [Електронний ресурс] – URL: https://doi.org/10.1177/10892680221105663 (дата звернення: : 21.05.2025).

6. Beeminder. – [Електронний ресурс] – URL: https://www.beeminder.com/ (дата звернення: 09.04.2025).

7. HabitBull. – [Електронний ресурс] – URL: https://www.habitbull.com/ (дата звернення: 09.04.2025).

8. Way of Life. – [Електронний ресурс] – URL: https://wayoflifeapp.com/ (дата звернення: 09.04.2025).

9. Productive. – [Електронний ресурс] – URL: https://productive.io/ (дата звернення: 09.04.2025).

10. The Fabulous. – [Електронний ресурс] – URL: https://www.thefabulous.co/ (дата звернення: 09.04.2025).

11. ISO/IEC 25010:2023. Systems and software engineering — Systems and software Quality Requirements and Evaluation (SQuaRE) — Product quality model. – London: BSI Standards Limited, 2023. – 46 p. – ISBN 978-0-539-14084-2.

12. International Software Testing Qualifications Board (ISTQB). Certified Tester Foundation Level Syllabus. – Version 4.0. – 2023. – 74 p.

13. Github.NureAlmakadmaMariam/2025_B_PI_PZPI-21-1_Almakadma_M_I. URL: https://github.com/NureAlmakadmaMariam/2025_B_PI_PZPI-21-1_Almakadma_M_I