ДЕРЖАВНИЙ УНІВЕРСИТЕТ ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ
НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНФОРМАЦІЙНИХ СИСТЕМ ТА ТЕХНОЛОГІЙ

КВАЛІФІКАЦІЙНА РОБОТА

на тему:

«Система автоматизованого планування, мотивації та емоційної підтримки для досягнення персональних цілей з використанням гейміфікації та штучного інтелекту»

на здобуття освітнього ступеня магістр
за спеціальності 126 Інформаційні системи та технології
(код,найменування спеціальності)
освітньо-професійної програми Інформаційні системи та технології
(назва)

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

_____ Ярослав АНДРЕЄВ
(підпис) *(ім'я, ПРІЗВИЩЕ здобувача)*

Виконав:
здобувач вищої освіти
група ІСДМ-61 Ярослав АНДРЕЄВ
(ім'я, ПРІЗВИЩЕ)
Керівник Віктор САГАЙДАК
PhD *(ім'я, ПРІЗВИЩЕ)*

Рецензент: Вікторія ЖЕБКА
(ім'я, ПРІЗВИЩЕ)

Київ 2026

ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ

Навчально-науковий інститут Інформаційних технологій

Кафедра Інформаційних систем та технологій
Ступінь вищої освіти магістр
Спеціальність 126 Інформаційні системи та технології
Освітньо-професійна програма Інформаційні системи та технології

ЗАТВЕРДЖУЮ
Завідувач кафедрою ІСТ
_____Каміла СТОРЧАК

"____" _____2025 року

З А В Д А Н Н Я
НА КВАЛІФІКАЦІЙНУ РОБОТУ

Андреєв Ярослав Олексійович
(прізвище, ім'я, по батькові здобувача)

1.Тема кваліфікаційної роботи: Система автоматизованого планування, мотивації та емоційної підтримки для досягнення персональних цілей з використанням гейміфікації та штучного інтелекту
керівник кваліфікаційної роботи: Віктор САГАЙДАК, PhD, доцент
(ім'я, ПРІЗВИЩЕ, науковий ступінь, вчене звання)
затверджені наказом Державного університету інформаційно-комунікаційних технологій від "___ " жовтня 2025 р. № ____

2. Строк подання кваліфікаційної роботи «26» грудня 2025 р.

3. Вихідні дані кваліфікаційної роботи:
1. Методи системного аналізу та об'єктно-орієнтованого проєктування.
2. Архітектура Великих Мовних Моделей (LLM) та Prompt Engineering.
3. Теоретичні аспекти гейміфікації (Модель Octalysis).
4. Науково-технічна література та документація (React, Node.js, Prisma) .

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити):
1. Аналіз психологічних бар'єрів продуктивності та існуючих програмних рішень.
2. Розробка архітектури системи та математичної моделі гейміфікації, верифікованої ШІ.
3. Програмна реалізація AI-функціоналу та тестування ефективності системи

5.Перелік ілюстраційного матеріалу: *презентація*

6. Дата видачі завдання «___» _____ 2025р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1.	Підбір технічної літератури та аналіз предметної області	01.09.2025 – 15.09.2025	Виконано
2.	Дослідження тенденцій розвитку та вибір архітектурних рішень	16.09.2025 – 30.09.2025	Виконано
3.	Проєктування алгоритмів (гейміфікація, AI-декомпозиція) та моделювання БД	01.10.2025 – 20.10.2025	Виконано
4.	Програмна реалізація серверної та клієнтської частини	21.10.2025 – 25.11.2025	Виконано
5.	Тестування, аналіз результатів та оцінка ефективності	26.11.2025 – 05.12.2025	Виконано
6.	Розробка демонстраційних матеріалів (презентація), доповідь.	06.12.2025 – 12.12.2025	Виконано
7.	Оформлення магістерської роботи, внесення правок та підготовка до захисту.	13.12.2025 – 23.12.2025	Виконано

Здобувач вищої освіти _____ _Ярослав АНДРЕЄВ__
(підпис) *(ім'я, ПРІЗВИЩЕ)*

Керівник кваліфікаційної роботи _____ _Віктор САГАЙДАК__
(підпис) *(ім'я, ПРІЗВИЩЕ)*

РЕФЕРАТ

Текстова частина кваліфікаційної роботи на здобуття ступеня магістр: 84 стор., 25 рис., 8 табл., 33 джерел.

Мета роботи – підвищення ефективності досягнення персональних цілей користувача шляхом розробки інформаційної системи, що забезпечує автоматизовану декомпозицію завдань, адаптивну гейміфікацію та інтелектуальну емоційну підтримку.

Об'єкт дослідження – процес планування та управління персональною продуктивністю в умовах невизначеності та змінного емоційного стану користувача.

Предмет дослідження – методи та засоби автоматизації цілепокладання, мотивації та психологічної підтримки користувача з використанням технологій штучного інтелекту та гейміфікації.

Короткий зміст роботи. У роботі проведено системний аналіз проблематики тайм-менеджменту та феномену прокрастинації. Обґрунтовано доцільність імплементації генеративного штучного інтелекту для автоматизації процесів планування та об'єктивізації механізмів гейміфікації. Спроєктовано та програмно реалізовано кросплатформну систему «Questify», функціонал якої включає AI-декомпозицію цілей, математичну модель прогресу з оцінкою когнітивної складності нейромережею, а також модуль персоналізованого віртуального ментора. Технічна реалізація виконана на базі стеку TypeScript з використанням фреймворків NestJS та React Native в екосистемі Expo; шар персистентності побудовано на СУБД PostgreSQL та ORM Prisma; інтелектуальне ядро реалізовано через інтеграцію моделі OpenAI gpt-4o-mini.

КЛЮЧОВІ СЛОВА: ШТУЧНИЙ ІНТЕЛЕКТ; ГЕЙМІФІКАЦІЯ; ВЕЛИКІ МОВНІ МОДЕЛІ (LLM); АВТОМАТИЗОВАНЕ ПЛАНУВАННЯ; ПРОКРАСТИНАЦІЯ; ПЕРСОНАЛЬНА ЕФЕКТИВНІСТЬ.

ABSTRACT

The text part of the qualifying master's thesis: 84 p., 25 fig., 8 tabl., 33 sources.

The aim of the work is to enhance the efficiency of achieving personal goals by developing an intelligent information system that provides automated task decomposition using generative AI, objective gamification with neural network-based difficulty assessment, and context-aware emotional support.

The object of research is the process of planning, motivation, and self-regulation of personal productivity under conditions of uncertainty and variable emotional states.

The subject of research covers methods and software tools for automating goal-setting and motivation using large language models (LLM), principles of objective gamification, and personalized AI mentorship.

Summary. The thesis analyzes current issues in time management and procrastination, substantiating the use of generative AI for planning automation and gamification objectification. A cross-platform system "Questify" has been designed and implemented, incorporating AI-driven goal decomposition, a mathematical progression model with neural network-assessed difficulty, a personalized virtual mentor, and a comprehensive gamification loop. The system is engineered using a TypeScript-based stack (NestJS, React Native, Expo), PostgreSQL with Prisma ORM, and integrates the OpenAI gpt-4o-mini model.

KEYWORDS: ARTIFICIAL INTELLIGENCE, GAMIFICATION, LARGE LANGUAGE MODELS, LLM, AUTOMATED PLANNING, OBJECTIVE GAMIFICATION, PERSONAL PRODUCTIVITY, PROCRASTINATION, AI MENTORING.

ЗМІСТ

ВСТУП 9
1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ТЕОРЕТИЧНІ ОСНОВИ 12
1.1 Аналіз проблеми особистої ефективності та існуючих програмних рішень . 12
1.2 Перспективи застосування штучного інтелекту та великих мовних моделей у системах підтримки прийняття рішень 17
1.3 Психологічні та когнітивні аспекти людино-машинної взаємодії в системах мотивації та саморегуляції 22
1.5 Стратегічний аналіз та постановка задачі дослідження 29
1.6 Етичні аспекти, безпека та проблематика стохастичності генеративних моделей 31
2 ПРОЄКТУВАННЯ СИСТЕМИ ТА МОДЕЛЮВАННЯ ПРОЦЕСІВ 33
2.1 Аналіз вимог та моделювання поведінки системи 33
2.2 Проєктування архітектури системи 35
2.3 Моделювання бази даних 40
2.4 Розробка алгоритмічного забезпечення та інтелектуальної логіки системи . 43
2.5 Проєктування інтерфейсу користувача та користувацького досвіду 46
2.6 Архітектура довгострокової пам'яті інтелектуального асистента 50
2.7 Математичне моделювання динаміки мотивації 51
3 ПРОГРАМНА РЕАЛІЗАЦІЯ СИСТЕМИ 54
3.1 Обґрунтування вибору технологічного стеку та організація структури проєкту 54
3.2 Реалізація серверної частини та бази даних 58
3.3 Реалізація клієнтської частини 60
3.4 Архітектурні особливості та програмна реалізація AI-оркестратора 62
3.5 Забезпечення інформаційної безпеки системи 64
3.6 Інфраструктурна організація, автоматизація розгортання та забезпечення відмовостійкості 66
3.7 Комплексна верифікація, навантажувальне тестування та метрики стабільності 69
3.8 Комплексна оцінка ефективності та результатів апробації системи 73
3.9 Техніко-економічне обґрунтування розробки та стратегія комерціалізації .. 75
ВИСНОВКИ 80
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ 82

ВСТУП

Актуальність теми. Сучасний ландшафт інструментів персональної ефективності переживає системну кризу. В умовах експоненційного зростання інформаційних потоків конвенційні підходи до тайм-менеджменту, реалізовані в більшості Task Management Systems (TMS), втрачають свою дієвість. Проблема полягає не стільки в обсягах даних, скільки в архітектурній стагнації програмного забезпечення: більшість рішень функціонують у парадигмі пасивного "контейнера даних". Вони ігнорують когнітивні обмеження оператора, перекладаючи всю вагу декомпозиції, пріоритизації та планування на самого користувача, який часто вже перебуває в стані когнітивного перевантаження або «втоми від прийняття рішень». Окремий шар проблематики стосується механізмів мотивації. Існуючі спроби гейміфікації рутинних процесів (на прикладі аналогів типу Habitica) часто нівелюються суб'єктивністю самооцінювання. Коли користувач самостійно визначає складність задачі та призначає винагороду, виникає конфлікт інтересів, що призводить до інфляції внутрішньої "ігрової валюти" та втрати дофамінового ефекту. Розвиток технологій генеративного штучного інтелекту (GenAI) та великих мовних моделей (LLM) створює технологічне підґрунтя для зміни самої парадигми проєктування персональних асистентів. З'являється можливість делегувати рутинні когнітивні операції – структурування хаосу думок у чіткі плани – незалежним алгоритмам. Розробка системи, здатної виступати не пасивним реєстром, а активним, об'єктивним арбітром продуктивності, є актуальним науково-прикладним завданням, що вимагає синтезу веб-технологій, методів Prompt Engineering та когнітивної психології.

Зв’язок роботи з науковими програмами. Магістерська виконана відповідно до плану науково-дослідних робіт кафедри та корелює з пріоритетними напрямами розвитку інформаційних технологій у сфері систем штучного інтелекту, автоматизації процесів прийняття рішень та людино-машинної взаємодії.

Мета дослідження. Підвищення ефективності досягнення персональних цілей шляхом розробки інтелектуальної інформаційної системи, що забезпечує

автоматизовану декомпозицію завдань, об'єктивізацію гейміфікації та адаптивну підтримку користувача засобами генеративних моделей.

Робоча гіпотеза. Дослідження базується на припущенні, що інтеграція LLM як проміжного шару валідації між наміром користувача та записом у базі даних дозволить знизити когнітивний поріг входу в планування. Передбачається, що заміна суб'єктивної оцінки складності задач на алгоритмічну, що базується на семантичному аналізі, забезпечить стійкішу мотивацію користувача порівняно з традиційними методами самоконтролю.

Задачі дослідження. Для досягнення поставленої мети та перевірки гіпотези необхідно вирішити низку завдань. На початковому етапі слід провести критичний аналіз методів підвищення особистої ефективності та виявити архітектурні обмеження сучасних систем планування. Наступним кроком є розробка архітектури кросплатформної системи, що передбачає безшовну інтеграцію LLM-провайдерів для семантичної обробки цілей. Ключовим завданням є формалізація математичної моделі гейміфікації, в рамках якої валідація складності та розрахунок винагороди здійснюються автоматизованим алгоритмом без прямого втручання користувача. Окремої розробки потребує алгоритмічне забезпечення для декомпозиції абстрактних запитів і контекстно-залежної генерації мотиваційних інтервенцій. Завершальний етап передбачає програмну реалізацію системи (Fullstack-рішення) та проведення верифікації її ефективності через порівняльний експеримент із залученням реальних користувачів.

Об’єкт дослідження – процес управління персональною продуктивністю та прийняття рішень в умовах невизначеності.

Предмет дослідження – методи автоматизованої декомпозиції цілей, моделі об'єктивної гейміфікації та засоби інтелектуальної підтримки користувача на базі великих мовних моделей.

Методи дослідження. Методологічну основу роботи складають: системний аналіз – для формування функціональних вимог; об’єктно-орієнтоване проєктування – для розробки архітектури програмного забезпечення; математичне моделювання – для опису динаміки ігрової економіки та нарахування досвіду. Для

налаштування та оптимізації роботи нейромережевих компонентів використано методи інженерії підказок (Prompt Engineering) та емпіричного тестування.

Наукова новизна одержаних результатів:

1. Вперше запропоновано метод об’єктивної гейміфікації, який, на відміну від існуючих підходів, делегує функцію оцінки складності завдання нейромережі. Це дозволяє сформувати детерміновану систему винагород, незалежну від настрою чи упереджень користувача.
2. Удосконалено метод автоматизованої декомпозиції цілей. Застосування гібридного алгоритму, що поєднує генеративні можливості LLM із суворою схемною валідацією (JSON Schema Validation), дозволяє отримувати гарантовано коректні структури даних для подальшої програмної обробки, мінімізуючи ймовірність галюцинацій моделі.

Практична цінність роботи. Результати дослідження втілено у вигляді повнофункціонального програмного комплексу "Questify". Розроблена система, що включає клієнтську частину, серверний API та модуль ШІ-асистента, готова до впровадження як інструмент онбордингу нових співробітників, платформа для корпоративного навчання або засіб персонального розвитку. Ефективність запропонованих підходів підтверджено результатами тестування на реальних сценаріях використання.

Апробація результатів. Основні теоретичні положення, архітектурні рішення та практичні результати дослідження доповідалися на наукових заходах. Зокрема, концепція використання ШІ в системах планування обговорювалася на X Міжнародній науковій конференції «Здобутки та досягнення прикладних та фундаментальних наук XXI століття» (м. Дніпро, 7 листопада 2025 р.). Технічні аспекти реалізації гейміфікації представлено на III Всеукраїнській науково-технічній конференції «Технологічні горизонти» (11 листопада 2025 р.).

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ТЕОРЕТИЧНІ ОСНОВИ

1.1 Аналіз проблеми особистої ефективності та існуючих програмних рішень

Еволюція програмного забезпечення для тайм-менеджменту зайшла у концептуальний глухий кут. Попри нарощування функціоналу – від примітивних цифрових записників до комплексних екосистем – більшість сучасних TMS (Task Management Systems) ігнорують фундаментальні обмеження когнітивного ресурсу оператора, тобто щоб займатися плануванням продуктивності треба вже бути продуктивним та мотивованим, знаючи що треба зробити. Спостерігається парадокс: інструменти, покликані впорядковувати хаос, самі стають джерелом ентропії. Проблема низької продуктивності лежить не у площині дефіциту інструментарію, а в площині конфлікту між раціональними моделями планування, закладеними в архітектуру ПЗ, та ірраціональними поведінковими патернами користувача в момент виконання.

Розглянемо цей конфлікт через призму поведінкової економіки. Феномен прокрастинації доцільно інтерпретувати як наслідок гіперболічного дисконтування (hyperbolic discounting), за якого суб'єктивна вартість винагороди обернено корелює з часом очікування [1, 2]. Планування довгострокових цілей (наприклад, вивчення мови) відбувається у так званому «холодному» стані свідомості, коли префронтальна кора домінує. Натомість фаза виконання часто припадає на «гарячий» стан, що вимагає негайного дофамінового підкріплення. Стандартна архітектура TMS не враховує цієї асиметрії: система лише фіксує факт невиконання (status: failed), формуючи деструктивний зворотний зв'язок і почуття провини, замість того, щоб пропонувати компенсаторні механізми.

Ситуація ускладнюється вичерпністю вольового ресурсу, відомою як "втома від прийняття рішень (Decision Fatigue) [3]. Необхідність ручного мікроменеджменту – формулювання суті задачі, встановлення пріоритетів, дедлайнів – виснажує нейрофізіологічний ресурс ще до початку продуктивної діяльності. Абстрактність великих завдань викликає параліч рішень. Звідси

випливає вимога до проектованої системи: вона повинна взяти на себе функцію зовнішнього регулятора, здатного до автоматичної декомпозиції цілей, розвантажуючи оператора.

Аналіз ринку виявляє домінування застарілих метафор. Продукти на кшталт Todoist або Microsoft To Do архітектурно пасивні: це CRUD-інтерфейси для списків, які покладають все логічне навантаження на людину [4]. Накопичення прострочених завдань у таких системах неминуче призводить до ефекту «стіни провини» (Wall of Shame) і подальшої відмови від використання. Інший полюс – системи управління знаннями (Obsidian, Notion) – часто провокують «пастку налаштування» [5], де процес конфігурації середовища підміняє собою реальну роботу.

Окремої критики заслуговують існуючі реалізації гейміфікації (Habitica, Forest). Їхня ключова вразливість – структурна суб'єктивність валідації [6]. Коли користувач поєднує ролі виконавця та контролера, виникає конфлікт інтересів. Це неминуче призводить до завищення оцінок власних дій задля отримання віртуальних винагород та, як наслідок, до інфляції ігрової валюти. Ефективна модель гейміфікації неможлива без незалежного арбітра – «оракула», роль якого людина виконувати не здатна через когнітивні викривлення.

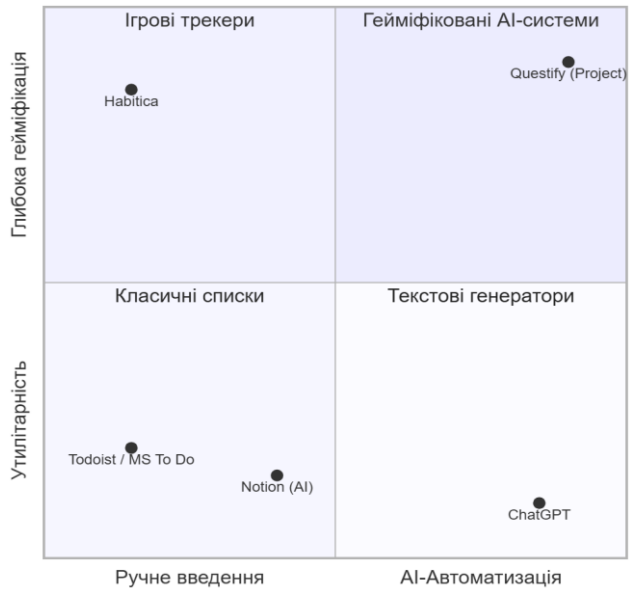


Рисунок 1.1 – Матриця стратегічного позиціонування систем управління завданнями

Вищенаведене обґрунтовує існування незаповненої ніші для гібридних систем класу "Questify". Необхідний перехід до архітектури, що поєднує автоматизацію планування на базі LLM з адаптивною гейміфікацією, де нейромережа виступає об'єктивним верифікатором складності, мінімізуючи вплив людського фактору на систему винагород.

1.1.1 Порівняльний аналіз функціональних можливостей та архітектурних обмежень сучасних TMS-систем

Сегмент програмного забезпечення класу TMS (Task Management Systems) характеризується ознаками функціонального насичення, яке, втім, слабо корелює з реальним підвищенням продуктивності кінцевого користувача. Більшість комерційних продуктів базуються на детермінованих алгоритмах управління ресурсами, ігноруючи стохастичну природу людської поведінки та нелінійність когнітивних процесів. Для ідентифікації архітектурних розривів, заповнення яких є метою даної розробки, виконано компаративний аналіз провідних ринкових рішень. Критеріями порівняння обрано рівень автоматизації вхідних даних, ступінь об'єктивності контролю виконання та адаптивність до психофізіологічного стану користувача (табл. 1.1).

Таблиця 1.1

Компаративний аналіз архітектурних підходів у існуючих TMS-системах

Критерій порівняння	Todoist/ MS To Do (Лінійні списки)	Habitica (Gamified)	Notion/ Obsidian (Knowledge Base)	Motion/ Reclaim	Questify (Проєктована система)
Архітектурний патерн	CRUD-контейнер	RPG-формат, з ручним введенням	Конструктор баз даних	Алгоритмічний планувальник	Гібридний AI-агент (Generative + Descriptive)

Продовження таблиці 1.1

Компаративний аналіз архітектурних підходів у існуючих TMS-системах

Критерій порівняння	Todoist/ MS To Do (Лінійні списки)	Habitica (Gamified)	Notion/ Obsidian (Knowledge Base)	Motion/ Reclaim (Algorithmic Calendars)	Questify (Проєктована система)
Механізм декомпозиції	Ручний	Ручний	Ручний/ Шаблонний	Відсутній	Автоматизований
Валідація виконання	Суб'єктивна (Self-reported)	Суб'єктивна (Self-reported)	Суб'єктивна	Бінарна (Time-based)	Об'єктивна (AI-верифікація, аналіз метаданих)
Модель мотивації	Відсутня ("Стіна провини")	Зовнішня (віртуальні предмети)	Внутрішня (естетика організації)	Відсутня (тиск дедлайнів)	Змішана (Intrinsic + Extrinsic через RAG-пам'ять)
Когнітивне навантаження	Високе (на етапі планування)	Середнє (менеджмент персонажа)	Критично високе (налаштування системи)	Низьке (але високий рівень тривожності)	Мінімальне (делегування рішень агенту)
Адаптивність до стану	Нульова (статичні дані)	Низька	Залежить від користувача	Низька (механічне ущільнення)	Висока (емоційний аналіз контексту)

Аналіз першої групи систем (Todoist та аналоги) демонструє їхню концептуальну пасивність. Реалізуючи метафору «цифрового паперу», ці інструменти покладають відповідальність за структурування ентропійних вхідних даних виключно на оператора. З погляду системного дизайну, такі додатки діють як інтерфейси до реляційної бази даних без шару інтелектуальної обробки. Критичним недоліком є високий поріг входу: трансформація абстрактного наміру, як до прикладу, «підготуватися до іспиту», у послідовність атомарних транзакцій вимагає значних вольових зусиль. В умовах виснаження когнітивного ресурсу це призводить до накопичення «мертвих» даних, що не ініціюють дію.

У сегменті гейміфікованих трекерів (Habitica) спроба підвищити залученість через біхевіористичні механізми стикається з фундаментальним логічним конфліктом. Поєднання в одній особі ролей виконавця (агента) та контролера (принципала) створює передумови для опортуністичної поведінки. За відсутності зовнішнього верифікатора виникає явище «ігрової інфляції»: спрощення критеріїв успіху або штучна фрагментація тривіальних завдань задля максимізації винагороди. Це призводить до девальвації віртуальних стимулів, перетворюючи процес на імітацію продуктивної діяльності без реального результату.

Системи управління знаннями (Notion, Obsidian), пропонуючи гнучкий інструментарій для імплементації методологій (GTD, PARA), часто потрапляють у пастку надлишкової складності. В інженерній психології цей ефект класифікується як зміщення мети: обслуговування інфраструктури системи починає споживати більше ресурсів, ніж виконання цільових завдань. Для користувача без навичок системного архітектора така варіативність стає паралізуючим фактором, що лише поглиблює прокрастинацію.

Алгоритмічні календарі (Motion), попри високий рівень автоматизації (Time Blocking), хибують на ігнорування психофізіологічних параметрів людини. Застосовуючи ресурсну модель оптимізації, алгоритм механічно ущільнює графік, не залишаючи буферного часу для когнітивного відновлення. Детермінований характер таких розкладів вступає у конфлікт зі стохастикою реального життя, де зовнішні збурення неминуче порушують ідеальний план, викликаючи у

користувача стрес та відчуття втрати контролю.

Вищенаведені факти свідчать про відсутність на ринку комплексного рішення, здатного поєднати автоматизацію рутини з емпатичною підтримкою. Існуючі інструменти поляризовані: або пасивні реєстри, або механістичні планувальники, або вразливі до маніпуляцій ігрові симулятори. Це обґрунтовує доцільність розробки системи «Questify», яка базується на гібридній архітектурі: використання великих мовних моделей (LLM) для семантичної декомпозиції мінімізує когнітивне навантаження, а впровадження нейромережевого арбітражу забезпечує об'єктивність гейміфікації, усуваючи проблему суб'єктивного оцінювання.

1.2 Перспективи застосування штучного інтелекту та великих мовних моделей у системах підтримки прийняття рішень

Сучасний розвиток систем підтримки прийняття рішень (СППР) маркується онтологічним переходом від детермінованих сценарних графів до ймовірнісних генеративних архітектур. Ключовою технологічною зміною стала відмова від рекурентних мереж (RNN) на користь архітектури Transformer, що дозволило вирішити проблему «катастрофічного забування» через механізм самостійної уваги (Self-Attention) [7].

Цей механізм забезпечує динамічне зважування токенів у вхідній послідовності, дозволяючи моделі утримувати контекстні зв'язки на дистанціях, недосяжних для попередніх поколінь NLP-алгоритмів [8]. Для проектованої системи це означає здатність враховувати не лише поточний запит користувача, а й ретроспективу його взаємодій, еволюцію обмежень та історію помилок.

Імплементація LLM у контур управління задачами виводить функціонал системи за межі простого чат-бота. Завдяки здатності до складного логічного висновування (Reasoning), модель виконує функцію семантичного інтерпретатора, трансформуючи нечіткі наміри (Intent Recognition) у суворі імперативи бази даних.

Технічна реалізація цього процесу базується на механізмах Function Calling

та примусового форматування виводу (JSON Mode). Це дозволяє конвертувати природну мову, наприклад, скаргу на втому, у структурований JSON-об'єкт для корегування графіку, гарантуючи валідність даних перед їх записом у реляційну схему. Агент стає здатним до автономної перебудови графа завдань при зміні екзогенних факторів.

Критичним обмеженням архітектури трансформерів залишається скінченність контекстного вікна.

Передача повного масиву історичних даних користувача в одному промпті є технічно неможливою та економічно недоцільною. Рішенням є патерн RAG (Retrieval-Augmented Generation), що передбачає використання векторної бази знань. Зберігання ембеддінгів (числових векторних представлень змісту) дозволяє виконувати семантичний пошук релевантних фрагментів пам'яті за косинусною подібністю. Такий підхід забезпечує модель довгостроковим контекстом без необхідності витратного донавчання (Fine-tuning) ваг нейромережі.Управління поведінкою моделі здійснюється методами інженерії підказок (Prompt Engineering), яку слід розглядати як програмну конфігурацію ваг уваги без зміни параметрів моделі [9].

Доцільним є архітектурне розмежування системних ролей (System Personas) з різними термодинамічними параметрами генерації. Роль «Планувальник» вимагає мінімальної температури ($T \to 0$) та застосування евристики Chain-of-Thought (CoT) для забезпечення детермінованості планів [10].

Натомість роль «Ментор» оперує вищими значеннями ентропії, базуючись на протоколах когнітивно-поведінкової терапії (CBT) для емуляції емпатії та зниження психологічного опору користувача.Важливою перевагою LLM є здатність до глибокого семантичного аналізу емоційного стану (Sentiment Analysis), що перевершує можливості лексичних методів (VADER, TextBlob). Трансформери здатні ідентифікувати імпліцитні маркери вигорання, сарказм або пасивну агресію, що дозволяє реалізувати функцію емоційної валідації [11].

Адаптація тональності комунікації до психоемоційного профілю користувача трансформує систему з утилітарного інструменту в адаптивного агента підтримки.

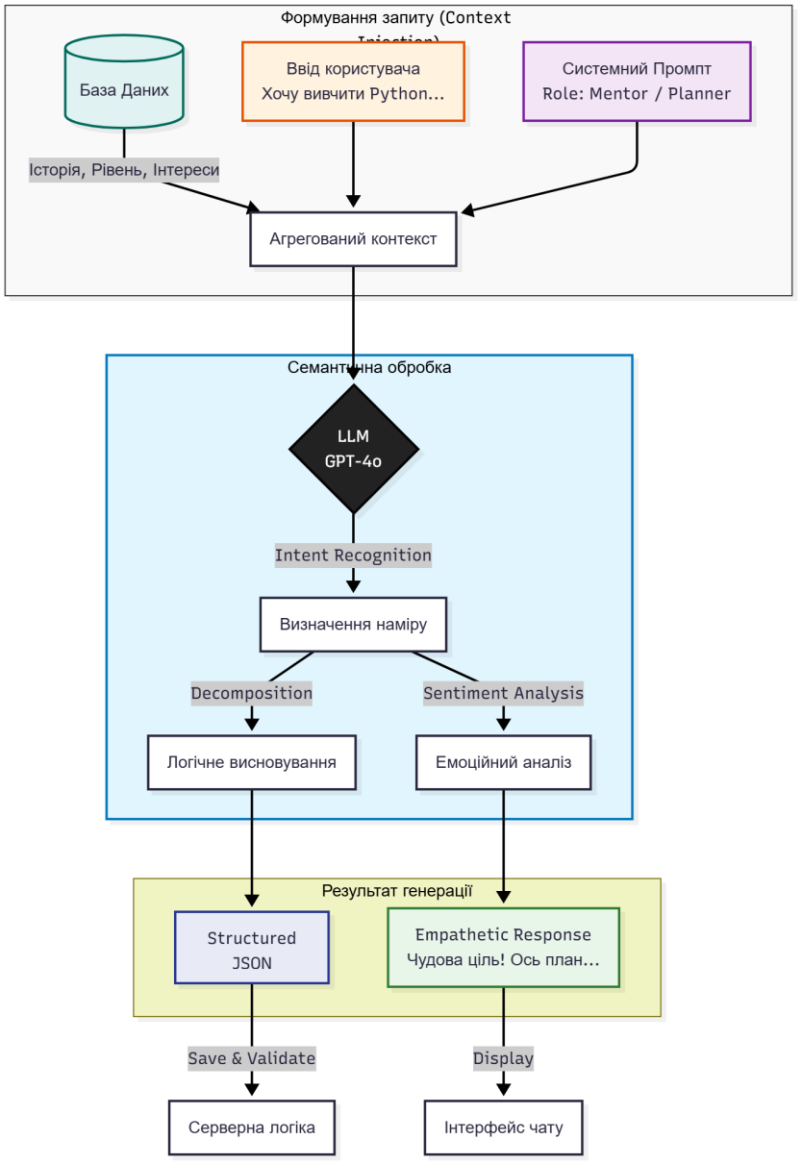


Рисунок 1.3 – Візуальний прототип роботи системи з user input

1.2.1 Методологічні підходи до інженерії підказок та парадигми мислення моделей

Інтеграція стохастичних генеративних моделей у детерміновані контури управління вимагає застосування формалізованих методів інженерії підказок (Prompt Engineering). Даний інструментарій доцільно інтерпретувати не як набір емпіричних евристик, а як механізм програмування ймовірнісного розподілу виводу моделі засобами природної мови. Фундаментальною основою методу є феномен навчання в контексті (In-Context Learning), що уможливлює адаптацію

моделі до доменних задач без модифікації ваг нейромережі. Математично це описується як максимізація умовної ймовірності $P(y|x, C)$, де x – вхідний стимул, y – генерація, а C – контекстне вікно, що містить інструкції та приклади.Стосовно задач автоматизованого планування, базова стратегія Zero-shot (генерація без попередніх прикладів) демонструє недостатню валідність при побудові багатоходових логічних ланцюжків. Тому в архітектурі системи імплементовано підхід Few-shot Learning: подача на вхід набору еталонних пар «стимул-реакція» (x_i, y_i) дозволяє сформувати вектор очікуваної поведінки агента, звужуючи простір допустимих рішень.Для мінімізації галюцинацій та логічних помилок застосовано методику Chain-of-Thought (CoT). Сутність методу полягає у примусовій генерації проміжних кроків міркування (intermediate reasoning steps) перед формуванням фінального виводу. У задачах декомпозиції цілей це змушує систему спершу проаналізувати ресурсні обмеження та передумови, і лише згодом синтезувати фінальну JSON-структуру плану. Такий підхід лінеаризує процес мислення моделі, наближаючи його до алгоритмічної послідовності.Взаємодія агента із зовнішнім середовищем (базою даних, календарними слотами) реалізується через парадигму ReAct (Reasoning + Acting). На відміну від статичного CoT, цей підхід впроваджує ітеративний цикл: генерація думки («Thought») – виконання дії («Action», SQL-запит) – отримання спостереження («Observation»). Отриманий результат (список незавершених задач) стає частиною контексту для наступного кроку. Це трансформує LLM з ізольованого текстового генератора в автономного агента, здатного коригувати власні плани на основі актуальних даних у режимі реального часу.

1.2.2 Еволюція підходів до обробки природної мови у задачах автоматизованого планування

Ретроспективний аналіз методів NLP (Natural Language Processing) у задачах планування демонструє фундаментальний зсув від імперативних детермінованих алгоритмів до ймовірнісних моделей глибокого навчання. Цей перехід

обумовлений неможливістю жорсткої формалізації простору людських намірів, який характеризується високою семантичною ентропією та контекстуальною залежністю.

На етапі зародження галузі, експертні системи 1970–90-х років, домінував підхід, заснований на правилах (Rule-Based Systems). Архітектурно такі рішення, включно з ранніми діалоговими агентами типу ELIZA, базувалися на кінцевих автоматах та пошуку за ключовими словами. Їхня придатність до задач планування була нульовою: система не формувала внутрішнього представлення світу, а лише реагувала на поверхневі синтаксичні тригери. Будь-яка варіативність вхідного запиту, що виходила за межі передбачених розробником сценаріїв «If-Then», призводила до зупинки алгоритму.

Впровадження рекурентних нейронних мереж (RNN) та їх удосконалених варіацій LSTM (Long Short-Term Memory) дозволило частково вирішити проблему генерації зв'язного тексту. Однак стосовно задач довгострокового планування ці архітектури наштовхнулися на математичні обмеження, відомі як проблема зникаючого градієнта (vanishing gradient problem). При обробці довгих послідовностей модель втрачала інформаційний зв'язок з початковим контекстом. На практиці це означало нездатність утримувати глобальну мету (наприклад, «організувати конференцію») під час генерації локальних підзадач – система «забувала» початкові обмеження ближче до кінця ланцюжка міркувань.

Архітектурний прорив, пов'язаний з появою механізму Self-Attention (2017), змінив топологію обробки даних. На відміну від рекурентних мереж, де шлях між токенами залежить від їх відстані $O(n)$, трансформери забезпечують доступ до будь-якого елемента контексту за константний час $O(1)$. Це дозволило моделювати складні каузальні (причинно-наслідкові) зв'язки, критично необхідні для декомпозиції. Нейромережа отримала здатність не просто продовжувати текст статистично, а вибудовувати ієрархічні структури: ідентифікувати передумови для кожної дії та розташовувати їх у логічно обґрунтованій хронологічній послідовності.Таким чином, сучасні LLM (Large Language Models) реалізують не просто функцію генерації тексту, а функцію емуляції розсудливості (reasoning).

Здатність моделі до Few-Shot Learning дозволяє відмовитися від ручного програмування логіки декомпозиції. Замість написання коду для кожного типу завдань, система інферує алгоритм розбиття задачі безпосередньо з наданих прикладів та своєї внутрішньої бази знань, що забезпечує адаптивність до унікального контексту користувача, недосяжну для попередніх поколінь алгоритмів.

1.3 Психологічні та когнітивні аспекти людино-машинної взаємодії в системах мотивації та саморегуляції

Проектування людино-машинних інтерфейсів (HCI) у класі систем особистої ефективності традиційно робить «помилку раціонального агента»: розробники апріорі вважають поведінку користувача детермінованою та логічною, що зазвичай не так. Проте емпіричний досвід доводить, що лінійні інженерні підходи непрацездатні в умовах стохастики людської психіки. Створення адаптивної системи вимагає імплементації психоемоційних корегуючих коефіцієнтів безпосередньо в логіку роботи алгоритмів, де інтерфейс трансформується з пасивного реєстратора в активний регулятор поведінки.

З позицій нейрофізіології, феномен прокрастинації є не моральною вадою, а еволюційно детермінованим механізмом енергозбереження. Відповідно до теорії когнітивного навантаження Дж. Свеллера, будь-яка операція прийняття рішень виснажує метаболічний ресурс префронтальної кори [12]. Складні, абстрактні цілі викликають «аналітичний параліч» через неспроможність мозку миттєво побудувати модель майбутнього. Проєктована система нівелює цей бар'єр, реалізуючи принцип «розширеного пізнання» (extended cognition). Делегування функції декомпозиції нейромережі фактично перетворює програмний комплекс на цифровий екзокортекс, що бере на себе найбільш енерговитратну частину когнітивної роботи, залишаючи користувачеві лише функцію виконання атомарних дій.

Алгоритмізація процесу формування звичок базується на синтезі

поведінкової моделі Б.Дж. Фогга ($B = MAT$) та циклічної моделі «гачка» (Hook Model) [13]. Технічна реалізація фази «Тригер» передбачає використання контекстно-залежних пуш-повідомлень, що резонують з внутрішніми станами. Фаза «Дія» оптимізується шляхом радикального спрощення UX: кількість транзакцій (кліків) для виконання цільової операції зводиться до мінімуму [14].

Критичним архітектурним компонентом є підсистема генерації винагород. Статичні системи гейміфікації неминуче стикаються з ефектом гедоністичної адаптації: мозок швидко звикає до однотипних стимулів. Використання генеративного ШІ дозволяє реалізувати механізм істинно стохастичної винагороди (Variable Reward), що базується на біхевіористичних принципах оперантного обумовлення Б.Ф. Скіннера. Непередбачуваність форми зворотного зв'язку (текстова похвала, унікальний артефакт, візуалізація прогресу) максимізує дофаміновий відгук і забезпечує стійкість мотиваційного циклу.

Соціально-комунікативний шар системи будується з урахуванням парадигми CASA (Computers As Social Actors), яка постулює, що користувачі підсвідомо переносять соціальні норми на взаємодію з комп'ютерними агентами [17]. Для уникнення реактивного опору, притаманного директивним планувальникам, логіка діалогу LLM базується на теорії самодетермінації (SDT) [15]. Система не віддає наказів, а використовує сократичний метод, пропонуючи варіанти та зберігаючи за користувачем відчуття автономії. При цьому, для запобігання формуванню залежності від зовнішньої валідації, в алгоритм закладено принцип «скаффолдингу» (scaffolding) – динамічного зменшення частоти підказок прямо пропорційно зростанню компетентності користувача [16].Таким чином, розроблювана система є складною соціотехнічною конструкцією, що не маніпулює увагою механістично, а діє як адаптивний когнітивний протез, компенсуючи обмеженість вольового ресурсу методами стохастичної алгоритміки.

1.3.1 Нейробіологічні механізми формування стійких поведінкових патернів

Архітектура системи «Questify» базується на врахуванні нейропластичності

як фізіологічної основи навчання. Фундаментальним механізмом, що забезпечує перехід від свідомої дії до автоматизму, виступає феномен довготривалої потенціації. Згідно з правилом Дональда Хебба, синхронна активація нейронних ланцюгів призводить до зміцнення синаптичних зв'язків між ними. У контексті проектування програмного забезпечення це диктує вимогу до циклічності інтерфейсних патернів. Система повинна забезпечувати мінімальний опір середовища при ініціації дії задля максимізації частоти повторень циклу «тригер – дія – винагорода», що прискорює формування стійких нейронних ансамблів.

Вагомим недоліком конвенційних систем управління задачами є ігнорування нейрогуморальної відповіді організму на інтерфейсні стимули. Візуалізація неструктурованого масиву завдань у вигляді нескінченних списків провокує активацію мигдалеподібного тіла. Ця біологічна структура реагує підвищенням рівня кортизолу, що інтерпретується мозком як загроза та блокує активність префронтальної кори. Наслідком стає ініціація захисної поведінки у формі прокрастинації. Розроблена система протидіє цьому ефекту шляхом атомізації цілей. Виконання мікро-завдання стимулює фазовий викид дофаміну, який діє як позитивне підкріплення та замикає петлю зворотного зв'язку, знижуючи рівень лімбічного опору.

Окремий шар бізнес-логіки системи враховує нелінійну динаміку вольового ресурсу. Спираючись на дослідження Б.Дж. Фогга [14] щодо хвилеподібного характеру мотивації, в архітектуру закладено алгоритм адаптивної деградації складності. У періоди мотиваційного спаду та виснаження нейромедіаторів система автоматично активує протокол мінімально життєздатної дії. Замість директивного примусу до виконання повного обсягу задачі алгоритм пропонує її спрощену версію, як-от читання однієї сторінки замість цілого розділу. Такий підхід дозволяє уникнути згасання сформованого синаптичного зв'язку та зберегти цілісність поведінкового патерну без надмірних енергетичних витрат з боку користувача.

1.4 Концептуальна модель гейміфікації та математична формалізація задачі

У межах дослідження концепція гейміфікації інтерпретується не як набір декоративних візуальних атрибутів, а як системна модифікація поведінкових патернів через використання елементів ігрового дизайну в неігрових контекстах, що узгоджується з визначенням Вербаха та Хантера [18]. Такий підхід уможливлює формування комплексної кібернетичної моделі управління. Вона базується на балансуванні екстрінсивних (зовнішніх) та інтрінсивних (внутрішніх) стимулів, де функція системи зводиться до пошуку оптимальної стратегії розподілу обмеженого когнітивного ресурсу користувача.

Математично процес декомпозиції глобальної мети формалізується як відображення абстрактного простору намірів G у впорядковану множину дискретних завдань T. Нехай $g \in G$ позначає глобальну мету. Система S, використовуючи велику мовну модель як оператор декомпозиції Φ_{LLM}, формує вектор завдань:

$$T_g = F_{decomp}(g) = \{t_1, t_2, \ldots, t_n\} \tag{1.1}$$

де кожен елемент t_i є кортежем:

$$t_i = \langle d_i, c_i, \tau_i \rangle \tag{1.2}$$

У цьому виразі d_i позначає семантичний опис завдання, c_i – оцінку когнітивної складності, а τ_i – часові обмеження виконання. Критичною вразливістю існуючих систем управління є лінійність функції винагороди $R(t) = const$, що створює передумови для експлойту системи через масове виконання тривіальних задач. Для усунення цього недоліку в розроблену модель введено зважену функцію корисності. Вона залежить від верифікованої штучним інтелектом складності c_i, яка визначається класифікатором та набуває значень із дискретної множини вагових коефіцієнтів W. Тоді функція нарахування досвіду XP набуває вигляду:

$$R(t_i) = \alpha \cdot w(c_i) \cdot \mu_{streak} \tag{1.3}$$

де α виступає базовим множником нормування, а k_{streak} є коефіцієнтом

неперервності серії, що зростає логарифмічно залежно від кількості днів безперервної активності користувача.

Моделювання довгострокової динаміки та механізмів запобігання інфляції досягнень спирається на нелінійну модель рівневої прогресії. Відповідно до психофізичного закону Вебера-Фехнера відчуття прогресу логарифмічно залежить від інтенсивності стимулу, тому поріг переходу на наступний рівень Lповинен зростати нелінійно. У системі імплементовано модель, що базується на квадратичній залежності від накопиченого досвіду:

$$L(XP_{total}) = \lfloor \beta \sqrt{XP_{total}} \rfloor + 1 \tag{1.4}$$

де β є коефіцієнтом масштабування. Зазначений підхід гарантує стабілізацію швидкості прогресу на пізніх етапах експлуатації системи, відомих як end-game, спонукаючи користувача підвищувати складність розв'язуваних задач для збереження темпу отримання винагород.

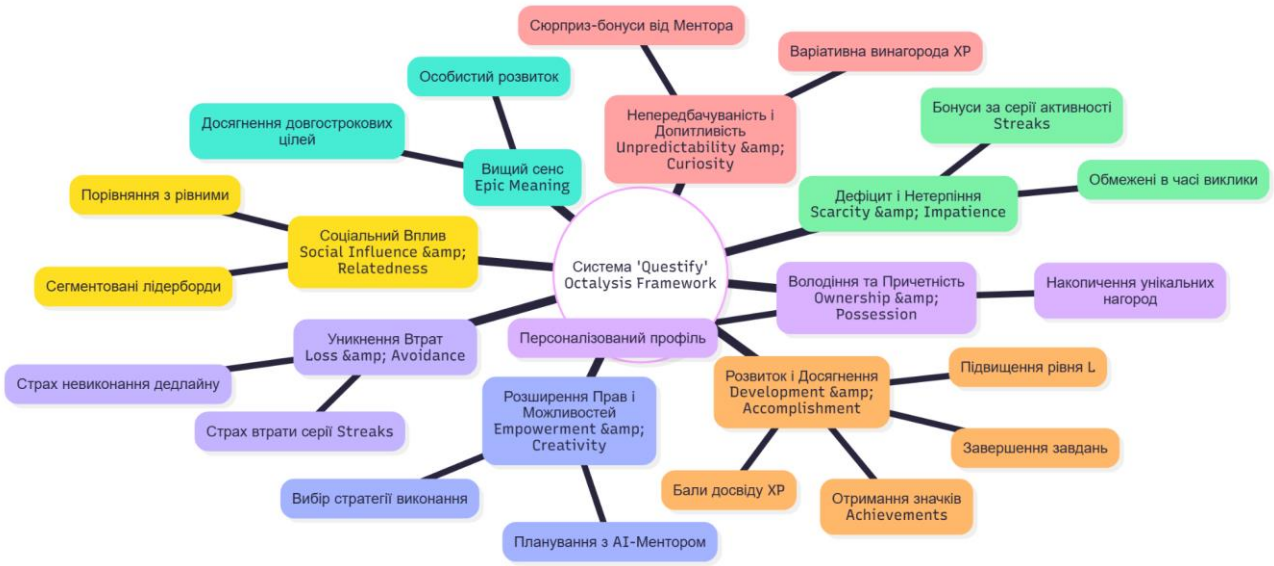


Рисунок 1.4 – Проєкція функціоналу системи на фреймворк Octalysis

Архітектоніка мотиваційного профілю базується на фреймворку Octalysis, розробленому Ю-Кай Чоу [19], який дозволяє векторизувати вплив різнорідних механік. Система реалізує гібридну схему, що поєднує методи позитивного та негативного підкріплення. Так звана White Hat Gamification реалізується через драйвери розвитку та соціального впливу, зокрема прогрес-бари, рівні та

сегментовані таблиці лідерів, де конкуренція відбувається виключно в межах когорти з близьким значенням рейтингу для уникнення демотивації. На противагу цьому Black Hat Gamification експлуатує поведінковий патерн уникнення втрат або Loss Aversion. Механіка підтримки серії активності створює цінний цифровий актив, втрата якого суб'єктивно сприймається гостро. Для гуманізації процесу вводиться стохастичний елемент можливості заморозки серії, що активується через семантичний аналіз аргументації поважної причини пропуску агентом штучного інтелекту.

Узагальнена ефективність системи визначається як дельта між автоматизованою ефективністю досягнення цілей та ефективністю ручного керування. Метою проєктування є максимізація умовної ймовірності виконання завдання $P(success|S)$ шляхом мінімізації транзакційних витрат на прийняття рішень та оптимізації психоемоційного стану користувача.

1.4.1 Адаптація механік системи відповідно до таксономії психотипів Річарда Бартла

Проектування релевантного мотиваційного контуру неможливе без урахування гетерогенності цільової аудиторії, поведінкові патерни якої суттєво варіюються залежно від індивідуальних стимулів. Фундаментальною теоретичною базою для сегментації користувачів у розробленій системі обрано таксономію професора Річарда Бартла [20]. Дана модель структурує суб'єктів взаємодії у чотирьох квадрантах, утворених перетином осей орієнтації на дії або взаємодію та фокусом на гравцях або ігровому світі.

У контексті систем особистої ефективності домінуючим сегментом є архетип Накопичувачів, представники якого мотивуються відображенням власного статусу та кількісними метриками прогресу. Саме для задоволення потреб цієї групи архітектура «Questify» містить розгалужену систему досягнень та рівневу прогресію, описану математичною моделлю у попередньому підрозділі. Алгоритм нарахування балів досвіду відкалібровано таким чином, щоб забезпечити

безперервне відчуття зростання навіть при виконанні рутинних мікро-завдань. Критичним елементом утримання тут виступає візуалізація заповненості профілю та наявність довгострокових стратегічних цілей.

Інтеграція психотипу Кілерів, що характеризується прагненням до домінування та прямої змагальності, реалізована з певними обмеженнями задля уникнення токсичності середовища. Пряма конкуренція у сфері особистого розвитку часто призводить до демотивації менш продуктивних учасників, тому механіки суперництва трансформовано у формат сегментованих лідербордів. Система автоматично групує користувачів у тимчасові когорти з еквівалентним рівнем активності. Такий підхід створює умови справедливої конкуренції та дозволяє задовольнити потребу у зверхності шляхом конструктивної боротьби за рейтинг без тиску на інших учасників.

Для задоволення потреб архетипу Дослідників, які цінують відкриття нового контенту та вивчення прихованих механізмів, задіяно генеративний потенціал великих мовних моделей. Варіативність реакцій ментора дозволяє уникнути репетативності діалогів, оскільки нейромережа здатна динамічно адаптувати стиль спілкування та пропонувати неочевидні стратегії вирішення завдань. Процес використання утилітарного інструменту фактично перетворюється на процес пізнання, де користувач утримується через механізм допитливості та пошук так званих пасхалок при виконанні специфічних комбінацій дій.

Категорія Соціалізаторів у поточній версії системи реалізується через взаємодію з антропоморфним цифровим агентом. Оскільки система фокусується на індивідуальній продуктивності та захисті приватності даних, роль соціального партнера делеговано штучному інтелекту. Завдяки використанню технік налаштування контексту ментор емулює емпатичну підтримку, активне слухання та емоційну валідацію. Це симулює повноцінний соціальний контакт без необхідності розбудови складної мережі зв'язків між реальними користувачами.

Отже, система «Questify» реалізує поліструктурну стратегію гейміфікації. Вона покриває мотиваційний спектр усіх чотирьох класичних психотипів, проте робить акцент на утилітарних функціях накопичення та дослідження, що є

найбільш релевантним для задач саморозвитку.

1.5 Стратегічний аналіз та постановка задачі дослідження

Системна інтерпретація факторів операційного середовища дозволяє стверджувати, що архітектурний синтез директивного планування та емпатичної підтримки виступає фундаментальним базисом стійкості системи «Questify». На відміну від традиційних платформ, що спираються на детерміновані евристики, залучення штучного інтелекту як незалежного арбітра дозволяє нівелювати похибку суб'єктивного самооцінювання. Саме перехід від жорсткої скриптової логіки до семантичного аналізу виконання завдань формує якісно нову функціональність, недосяжну для стандартних трекерів звичок.

Втім, інженерна проекція даної концепції виявляє низку суттєвих обмежень. Технічна реалізація проекту перебуває у критичній залежності від третіх сторін, передусім провайдерів великих мовних моделей екосистеми OpenAI. Це створює ризики не лише доступності сервісу, але й варіативності часу відгуку, що є критичним для забезпечення прийнятного користувацького досвіду. Стохастична латентність генерації тексту вимагає від архітектури системи впровадження складних механізмів, таких як асинхронні черги обробки запитів, патерни переривання ланцюга для відмовостійкості та методи оптимістичного оновлення інтерфейсу, що дозволяють маскувати затримки. Також гостро постає питання економічної ефективності транзакцій, що диктує необхідність розробки проміжного шару кешування та оптимізації підказок.

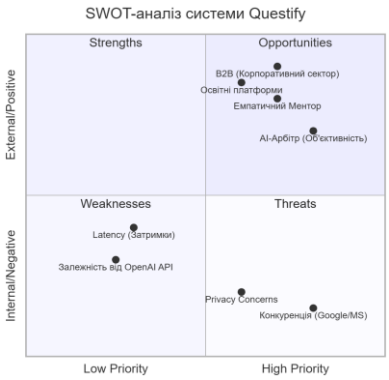


Рисунок 1.5 – SWOT-аналіз розроблюваної системи

Зовнішній тиск через експансією технологічних гігантів у нішу персональних асистентів може бути демпфований через закладену в архітектуру можливість вертикального масштабування. Потенціал адаптації ядра системи для корпоративного сектору або інтеграція в освітні процеси дозволяє диверсифікувати ризики продуктового ринку. При цьому питання приватності даних вирішується комплексно шляхом мінімізації контексту, що передається на зовнішню обробку, та використання знеособлених ідентифікаторів.

Виходячи з вищевикладеного, ключове науково-технічне протиріччя полягає у необхідності забезпечення високої когнітивної якості зворотного зв'язку від нейромережі при жорстких обмеженнях на час відгуку та вартість обчислень. Відтак основна задача дослідження формулюється як розробка та обґрунтування архітектурно-програмних методів інтеграції генеративних моделей у веб-середовище. Ці методи повинні забезпечити баланс між персоналізацією контенту, швидкодією системи та безпекою даних користувача.

Метою магістерської роботи визначено проєктування та програмну реалізацію інформаційної системи, що забезпечує автоматизовану декомпозицію цілей та підтримку мотивації користувача засобами генеративного штучного інтелекту. Досягнення поставленої мети вимагає вирішення комплексу взаємопов'язаних задач автоматизації. Першочерговим є створення механізму семантичного парсингу, здатного трансформувати абстрактні наміри природною мовою у валідовані графи завдань у форматі JSON з перевіркою їхньої реалістичності. Система повинна забезпечувати поліморфізм рольових моделей комунікації від суворого інструктора до емпатичного ментора зі збереженням контексту попередніх взаємодій. Підсистема гейміфікації має базуватися на алгоритмах нарахування балів, що оперують виключно об'єктивною оцінкою когнітивної складності, отриманою від нейромережі.

Ефективність розробленого рішення оцінюватиметься за низкою індикаторів. Ключовою кількісною метрикою визначено скорочення часу планування на 60–70 відсотків порівняно з ручними методами. Якісним критерієм успішності слугуватиме динаміка коефіцієнта завершення завдань, що корелюватиме зі

здатністю системи утримувати користувача в стані потоку. Технічна валідація включатиме перевірку стійкості до галюцинацій мовної моделі та стабільність роботи механізмів доповненої генерації.

1.6 Етичні аспекти, безпека та проблематика стохастичності генеративних моделей

Імплементація генеративного штучного інтелекту в архітектурний контур систем персональної ефективності вимагає фундаментального перегляду парадигми інформаційної безпеки. Акцент зміщується з класичного захисту периметра мережі на дотримання принципів відповідального використання алгоритмів. Архітектура трансформерів, яка формує базис сучасних великих мовних моделей, містить іманентну вразливість, зумовлену стохастичною природою генерації тексту. Оскільки модель функціонує через імовірнісне передбачення наступного токена, а не через звернення до фіксованої бази фактів, виникає неусувна ймовірність семантичних девіацій або галюцинацій. У межах розроблюваної системи подібна поведінка алгоритму класифікується як критична, оскільки вона може призвести до формування шкідливих рекомендацій. Наприклад, система може ігнорувати фізіологічні показники втоми заради досягнення абстрактних метрик продуктивності, що суперечить меті збереження здоров'я користувача.

Для мінімізації зазначених ризиків застосовано архітектурний патерн доповненої генерації через пошук. Цей підхід штучно звужує простір генерації та змушує модель спиратися виключно на попередньо відібрані та верифіковані фрагменти знань, ігноруючи власні ваги, сформовані під час навчання. Набори даних, використані для тренування базових моделей, часто містять латентні соціальні стереотипи та патерни так званої токсичної продуктивності. Відсутність належного калібрування системних інструкцій загрожує реплікацією культури успіху за будь-яку ціну, яка негативно впливає на психоемоційний стан людини. Тому етичний дизайн вимагає жорсткої пріоритезації ментального здоров'я над

механічним виконанням завдань, що реалізується через налаштування параметрів температури генерації та спеціалізованих системних підказок.

Окремого технічного аналізу потребує вектор атак через ін'єкцію інструкцій. Специфіка великих мовних моделей полягає у відсутності розділення каналів даних та керування: обидва потоки передаються природною мовою через єдиний інтерфейс. Це створює вразливість до маніпулятивних запитів, здатних змусити модель ігнорувати попередні налаштування безпеки та розкрити внутрішню логіку системи. Традиційні сигнатурні методи захисту в цьому сценарії неефективні через нескінченну варіативність людської мови.

Ефективна протидія реалізується виключно через багатошарову фільтрацію вхідних даних та застосування методики сендвіч-структур, коли інструкції безпеки дублюються після блоку даних користувача для перекриття потенційних шкідливих команд.

Гостро постає питання відповідності вимогам загального регламенту захисту даних при використанні хмарних інтерфейсів програмування. Передача контексту, що містить інтимні деталі розкладу та звичок особистості, на сервери третіх сторін створює ризик витоку конфіденційної інформації. Стратегія мінімізації даних диктує необхідність попередньої обробки запитів на стороні клієнта. Персональні ідентифікатори повинні замінюватися синтетичними токенами ще до моменту відправки запиту в мережу.

Технічно складною задачею залишається реалізація права на забуття в системах, що використовують векторні бази даних для довгострокової пам'яті. Видалення інформації з векторного простору є нетривіальним процесом, тому розв'язання проблеми полягає у впровадженні гібридної схеми зберігання.

У довгостроковій перспективі повне нівелювання ризиків перехоплення трафіку та залежності від зовнішніх провайдерів вбачається у переході на використання локальних малих мовних моделей. Виконання інференсу безпосередньо на кінцевому пристрої дозволить реалізувати концепцію конфіденційності за дизайном у повному обсязі, оскільки чутливі дані фізично не залишатимуть контрольований користувачем цифровий контур.

2 ПРОЄКТУВАННЯ СИСТЕМИ ТА МОДЕЛЮВАННЯ ПРОЦЕСІВ

2.1 Аналіз вимог та моделювання поведінки системи

Процес інженерії вимог у контексті розробки інтелектуальних систем виходить за межі простої трансляції концептуальних моделей у технічні специфікації. Ключова архітектурна дихотомія проєкту полягає в антагонізмі між високою обчислювальною вартістю інференсу нейромереж та імперативом низької латентності, що є стандартом для сучасних односторінкових застосунків. Розв'язання цього конфлікту вимагає нетривіальних підходів до асинхронної обробки даних.

Функціональна декомпозиція виокремлює чотири автономні підсистеми, взаємодія між якими регламентується суворими контрактами прикладного програмного інтерфейсу. Ядром архітектури виступає модуль оркестрації штучного інтелекту, що виконує роль зовнішнього когнітивного ресурсу. Цей компонент забезпечує єдину точку входу для неструктурованих запитів природною мовою, відповідаючи за семантичний парсинг та трансформацію намірів користувача у верифікований граф атомарних дій. Логічним продовженням конвеєра є модуль гейміфікації, завдання якого полягає в автоматизації циклу зворотного зв'язку. Він розраховує винагороду у балах досвіду та управляє станом механіки неперервності серії, що елімінує суб'єктивний фактор при валідації досягнень.

Контур психологічної підтримки реалізовано через підсистему діалогового асистента. На відміну від детермінованих чат-ботів, цей модуль оперує динамічним контекстом профілю користувача, що включає поточний рівень прогресії та історію виконаних завдань. Це забезпечує високу контекстуальну релевантність згенерованих порад. Безпека сесій та розмежування прав доступу гарантуються модулем ідентифікації, побудованим на базі стандарту веб-токенів JSON.

Специфікація нефункціональних вимог спирається на атрибути якості стандарту ISO 25010. Врахування стохастичної природи великих мовних моделей

змушує адаптувати систему до часу генерації відповіді, що може досягати п'ятнадцяти секунд. Це значно перевищує стандартні пороги очікування. Для нівелювання негативного впливу на користувацький досвід клієнтська архітектура імплементує патерн оптимістичного оновлення інтерфейсу. Цей підхід дозволяє екрану реагувати на дії користувача миттєво, виконуючи синхронізацію з сервером у фоновому режимі. Для стандартних операцій читання та запису встановлено жорсткий ліміт мережевої затримки на рівні 300 мілісекунд. Захист інформаційного периметра забезпечується санітизацією вхідних даних для протидії атакам типу ін'єкцій та примусовим шифруванням трафіку. База даних проєктується з урахуванням можливості горизонтального масштабування через механізм шардінгу при досягненні пікових навантажень.

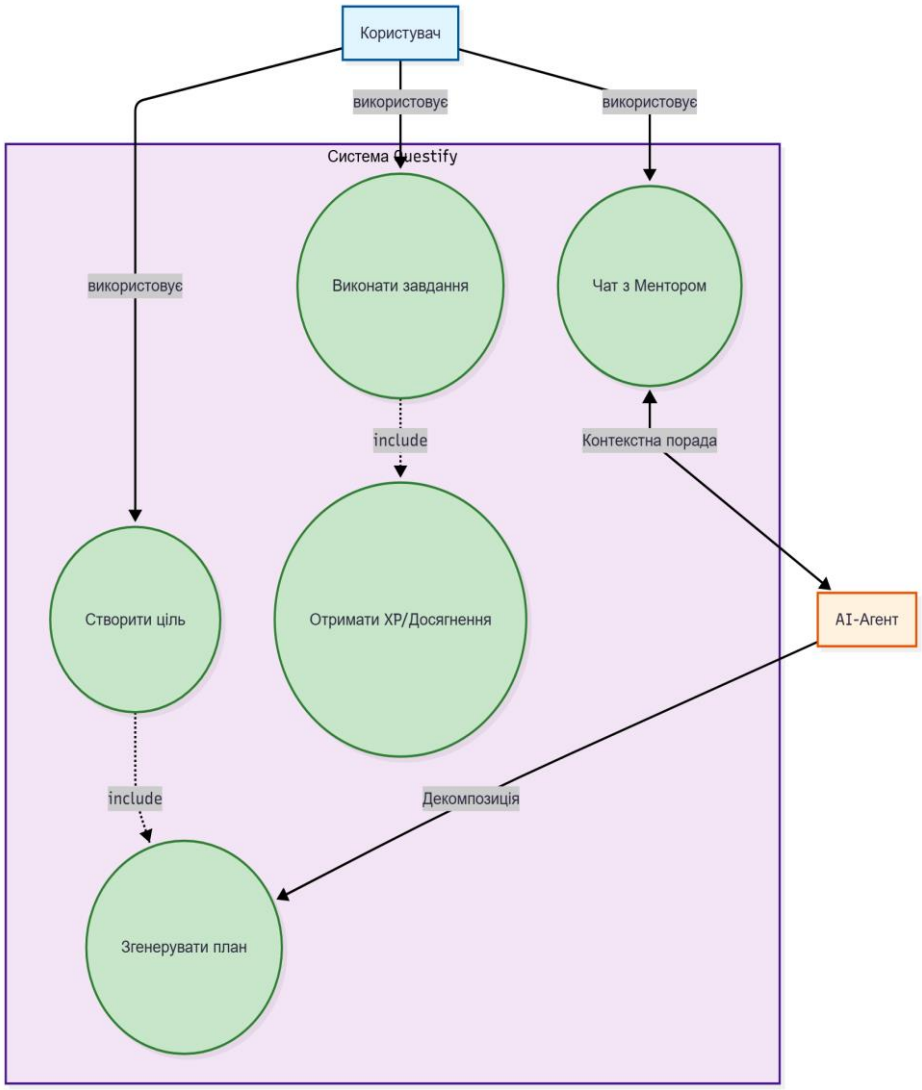


Рисунок 2.1 – Діаграма варіантів використання (Use Case Diagram)

Моделювання поведінкових аспектів системи формалізовано засобами діаграми варіантів використання. Сценарій взаємодії охоплює повний життєвий цикл запиту від ініціації декомпозиції цілі до отримання віртуальної винагороди. Специфічною рисою моделі є визначення агента штучного інтелекту як активного актора, здатного ініціювати комунікацію незалежно від дій користувача, що реалізує проактивну стратегію мотивації.

2.2 Проєктування архітектури системи

Процес архітектурного конструювання інформаційної системи вимагає розв'язання задачі багатокритеріальної оптимізації, де цільовою функцією виступає забезпечення атрибутів якості згідно зі стандартом ISO 25010, зокрема надійності та супроводжуваності, при жорстких обмеженнях на ресурси розробки. Архітектурний стиль системи «Questify» визначено як триланкову клієнт-серверну модель, реалізовану на базі патерну односторінкового застосунку.

Вибір зазначеного підходу ґрунтується на компаративному аналізі альтернатив. На відміну від класичного рендерингу на стороні сервера, де генерація розмітки відбувається централізовано, обрана архітектура дозволяє делегувати навантаження з візуалізації інтерфейсу безпосередньо клієнтському пристрою. Це забезпечує високу інтерактивність із часом відгуку менше ста мілісекунд та мінімізацію трафіку даних, оскільки після ініціалізації сесії обмін інформацією відбувається виключно у форматі об'єктної нотації JSON.

Декомпозиція системи передбачає сегрегацію трьох логічно ізольованих рівнів. Рівень представлення, реалізований засобами бібліотеки React, відповідає за рендеринг компонентів, обробку вводу та управління локальним станом додатка. Рівень бізнес-логіки, побудований на платформі Node.js із використанням фреймворку NestJS [22], інкапсулює алгоритми гейміфікації, валідацію бізнес-правил та оркестрацію запитів до сервісів штучного інтелекту. Рівень персистентності знаходиться під управлінням об'єктно-реляційної системи управління базами даних PostgreSQL [23]. Вибір реляційної моделі на противагу

документо-орієнтованим рішенням зумовлений імперативом забезпечення суворої транзакційної цілісності для фінансових операцій з балами досвіду.

Критичним викликом стала інтеграція із зовнішніми генеративними моделями. Оскільки інференс великих мовних моделей характеризується високою латентністю, синхронна обробка запитів є неприпустимою для однопотокової моделі середовища виконання. Тому було імплементовано асинхронну модель взаємодії на базі черг повідомлень. Логіка обробки запитів ізольована в сервісному шарі, що дозволяє централізувати механізми повторних спроб та обмеження частоти запитів. Процес генерації плану дій деталізовано на діаграмі послідовності, яка демонструє потік управління від санітизації вхідних даних до десеріалізації відповіді моделі.

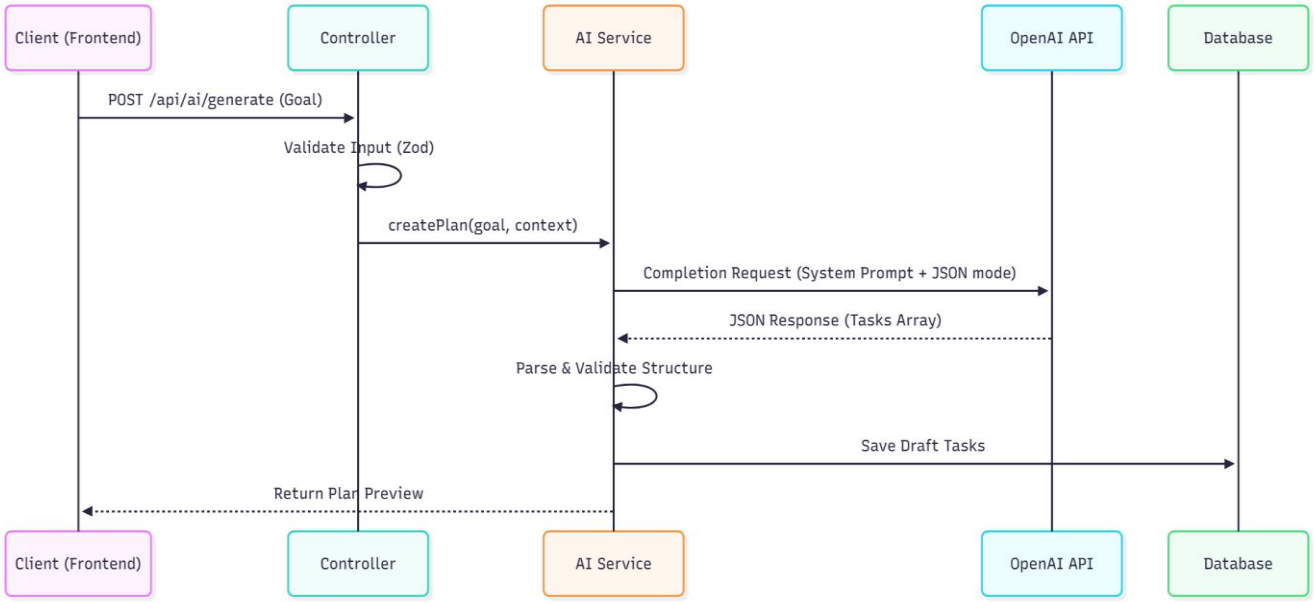


Рисунок 2.2 – Діаграма послідовності процесу генерації плану

Окремої уваги заслуговує моделювання динаміки станів бізнес-сутностей. Життєвий цикл ключової сутності завдання формалізовано як детермінований скінченний автомат. У системі впроваджено проміжний стан чернетки для планів, згенерованих алгоритмом, але ще не верифікованих користувачем. Такий підхід реалізує патерн людини в контурі управління, знижуючи ризик автоматичного створення некоректних даних. Логіка переходів між станами, що ініціює нарахування винагороди, візуалізована на діаграмі станів.

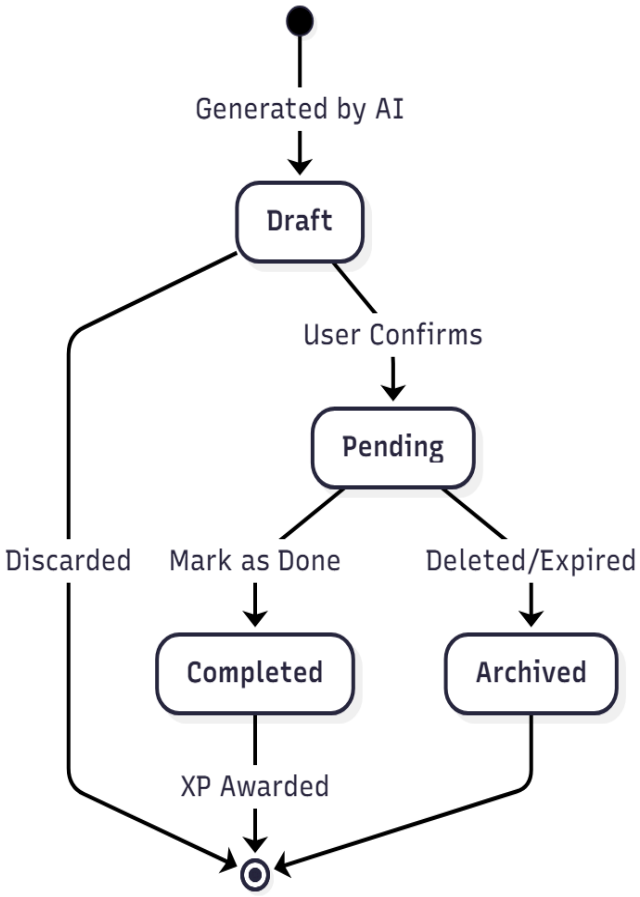


Рисунок 2.3 – Діаграма станів сутності Task

Організація взаємодії між клієнтом та сервером базується на прикладних програмних інтерфейсах архітектурного стилю REST. Відмова від підходу GraphQL на користь ресурсно-орієнтованої моделі є інженерним рішенням, продиктованим вимогами до продуктивності. REST-архітектура дозволяє ефективно використовувати механізми HTTP-кешування та мережі доставки контенту для запитів читання. Використання стандартних методів протоколу забезпечує ідемпотентність операцій, що є критичним фактором узгодженості даних при нестабільному з'єднанні.

Структура програмного інтерфейсу декомпозована за функціональними доменами. Фундаментальним рівнем виступає підсистема ідентифікації. Реалізація автентифікації здійснена на базі стандарту веб-токенів у схемі з використанням пари ключів доступу та оновлення. Цей підхід усуває необхідність збереження стану сесії на сервері, перекладаючи валідацію права доступу на перевірку

криптографічного підпису. Це значно спрощує горизонтальне масштабування системи.

Таблиця 2.1

Група методів: Автентифікація та Користувачі

Метод	URL-адреса	Опис	Параметри запиту (Body/Query)	Відповідь (Success 200/201)
POST	/api/auth/register	Реєстрація нового користувача	{ email, password, name }	{ token, user: { id, name, level } }
POST	/api/auth/login	Вхід у систему	{ email, password }	{ token, user: { id, name, level } }
GET	/api/users/me	Отримання профілю поточного користувача	Header: Authorization	{ id, email, xp, level, stats }
PATCH	/api/users/me	Оновлення налаштувань	{ name, timezone, theme }	{ success: true, user: {...} }

Основна бізнес-логіка роботи із завданнями реалізована через набір типових операцій створення, читання, оновлення та видалення. Особливістю реалізації є гарантована ідемпотентність методів зміни статусу, що запобігає дублюванню транзакцій нарахування досвіду.

Таблиця 2.2

Група методів: Керування завданнями

Метод	URL-адреса	Опис	Параметри запиту	Відповідь
GET	/api/tasks	Отримання списку завдань	?date=YYYY-MM-DD	[{ id, title, difficulty, reward }, ...]

Продовження табл. 2.2

Метод	URL-адреса	Опис	Параметри запиту	Відповідь
POST	/api/tasks	Створення нового завдання вручну	{ title, difficulty, deadline }	{ id, title, status: "PENDING" }
PATCH	/api/tasks/:id/complete	Завершення завдання (Гейміфікація)	-	{ task: { status: "DONE" }, rewards: {...} }
DELETE	/api/tasks/:id	Видалення завдання	-	{ success: true }

Інтелектуальні функції системи, такі як семантична декомпозиція цілей та діалог з ментором, винесені в окремий домен. Ці кінцеві точки відрізняються збільшеним часом очікування відповіді та потенційним використанням стрімінгової передачі даних.

Таблиця 2.3

Група методів: Інтелектуальні сервіси

Метод	URL-адреса	Опис	Параметри запиту	Відповідь
POST	/api/ai/generate-plan	Генерація плану досягнення цілі	{ goalDescription, duration }	{ plan: [{ step, title, difficulty }, ...] }
POST	/api/ai/chat	Відправка повідомлення ментору	{ message, context }	{ reply, sentiment }

Продовження Таблиці 2.3

Метод	URL-адреса	Опис	Параметри запиту	Відповідь
POST	/api/ai/suggest-goal	Отримання рекомендацій цілей	{ interests: [] }	{ suggestions: [] }

Обмін даними в межах інформаційного контуру здійснюється з використанням формату серіалізації JSON. Обробка виключних ситуацій на стороні клієнта базується на інтерпретації семантичних кодів стану протоколу HTTP, що забезпечує детерміновану поведінку програмного забезпечення та уніфікацію обробки помилок через глобальні перехоплювачі.

2.3 Моделювання бази даних

Надійність інформаційної системи перебуває у прямій залежності від якості організації рівня даних. Реалізація шару персистентності базується на використанні інструменту об'єктно-реляційного відображення Prisma.

Цей вибір уможливлює застосування декларативного підходу до опису схеми та автоматизує генерацію типобезпечного клієнта для мови TypeScript [25]. Таке архітектурне рішення мінімізує ймовірність синтаксичних помилок, притаманних ручному конструюванню SQL-запитів, та забезпечує суворе версіонування структурних змін бази даних через механізм міграцій.

На етапі концептуального моделювання предметної області розроблено інфологічну модель сутність-зв'язок. Центральним елементом схеми виступає сутність користувача, яка агрегує облікові дані, включно з хешами паролів, та метрики гейміфікації, такі як накопичені бали досвіду та поточний рівень прогресії. Обліковий запис слугує кореневим вузлом для ієрархії залежних об'єктів: цілей, завдань, нагадувань та сповіщень. Сутність цілі формалізує глобальні наміри та

містить атрибутивний склад для визначення часових рамок і статусу виконання.

Архітектура підтримує вкладеність, де одна ціль декомпозується на множину завдань. Завдання є самостійною операційною одиницею планування, що характеризується атрибутом складності, який має пряму кореляцію з ваговим коефіцієнтом винагороди.

Система заохочень реалізована через довідкову сутність досягнення. Зв'язок між користувачем та нагородами організовано через асоціативну таблицю, що реалізує відношення багато-до-багатьох та фіксує часові мітки отримання досягнень.

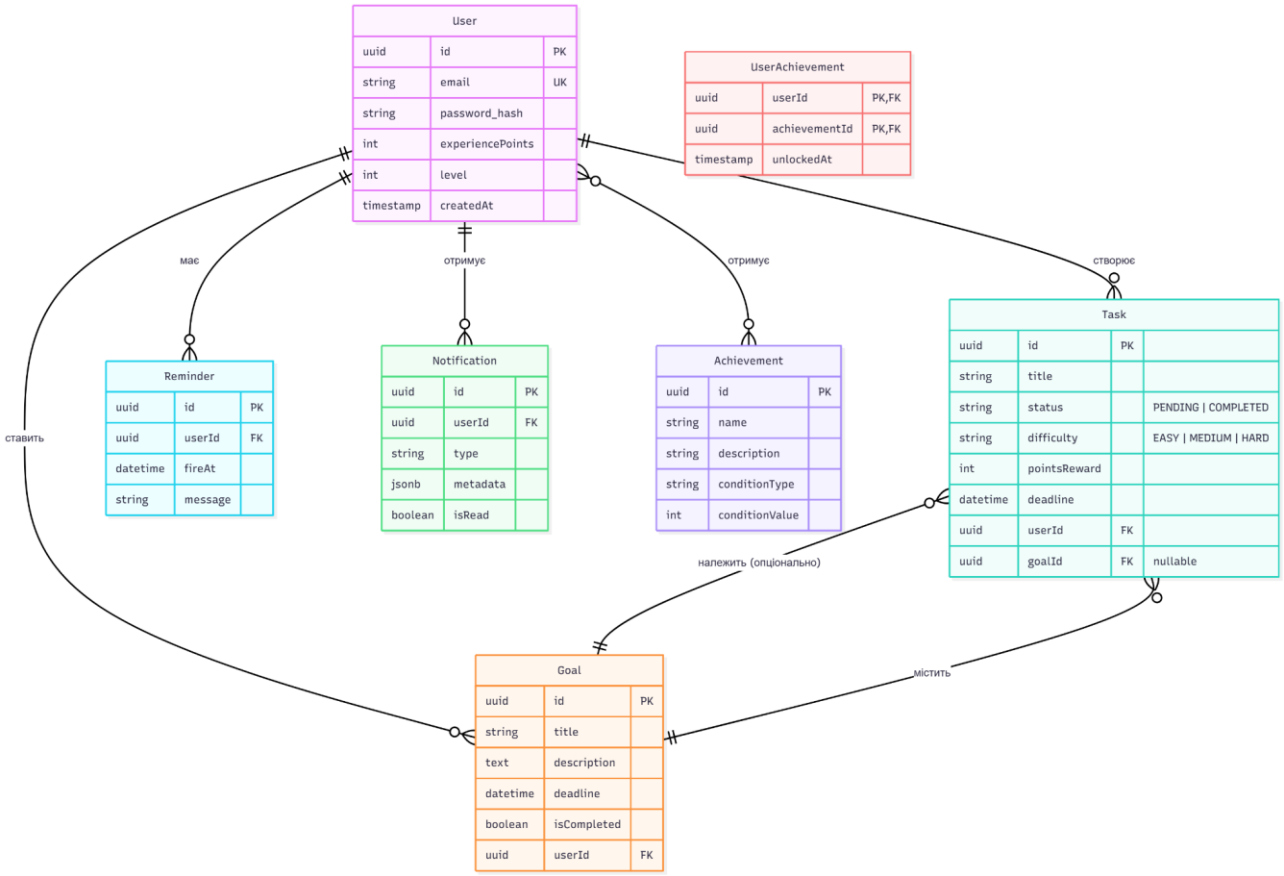


Рисунок 2.4 – ER-діаграма бази даних системи (Mermaid)

Фізична реалізація сховища даних виконана на платформі PostgreSQL версії 16. Вибір цієї системи управління базами даних зумовлений вимогою підтримки гібридної моделі. Використання нативного бінарного типу JSON дозволяє зберігати поліморфні метадані сповіщень у єдиній таблиці без порушення реляційної цілісності, що є критичним фактором для масштабування підсистеми повідомлень.

Загальна схема даних приведена до третьої нормальної форми: усі атрибути є атомарними, а транзитивні залежності неключових полів відсутні. Свідомим винятком є денормалізоване поле метаданих, структура якого варіюється залежно від типу події для спрощення інтеграції із зовнішніми сервісами розсилки.

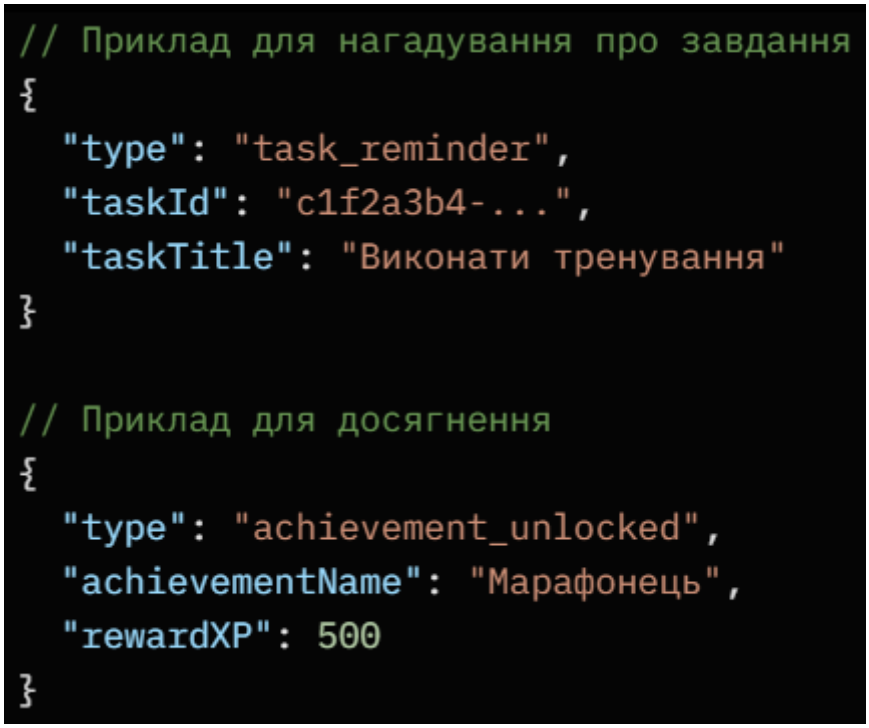


Рисунок 2.5 – Приклад вигляду структури у коді

Стратегія індексування охоплює автоматично згенеровані індекси для зовнішніх ключів та кастомні композитні структури. Зокрема, для оптимізації вибірок актуальних завдань створено індекс на основі B-дерева, що покриває ідентифікатор користувача, статус та дедлайн. Для прискорення пошуку по неструктурованих масивах даних застосовано узагальнений зворотний індекс.

Архітектура серверної частини реалізована на фреймворку NestJS із дотриманням принципів SOLID та патерну ін'єкції залежностей [29]. Декомпозиція компонентів виконана за шаруватим принципом. Шар контролерів відповідає виключно за валідацію вхідних запитів протоколу HTTP та серіалізацію відповідей. Бізнес-логіка інкапсульована у сервісному шарі, який координує взаємодію між підсистемами. Доступ до даних абстраговано через спеціалізований сервіс Prisma, що виступає альтернативою класичному патерну репозиторію.

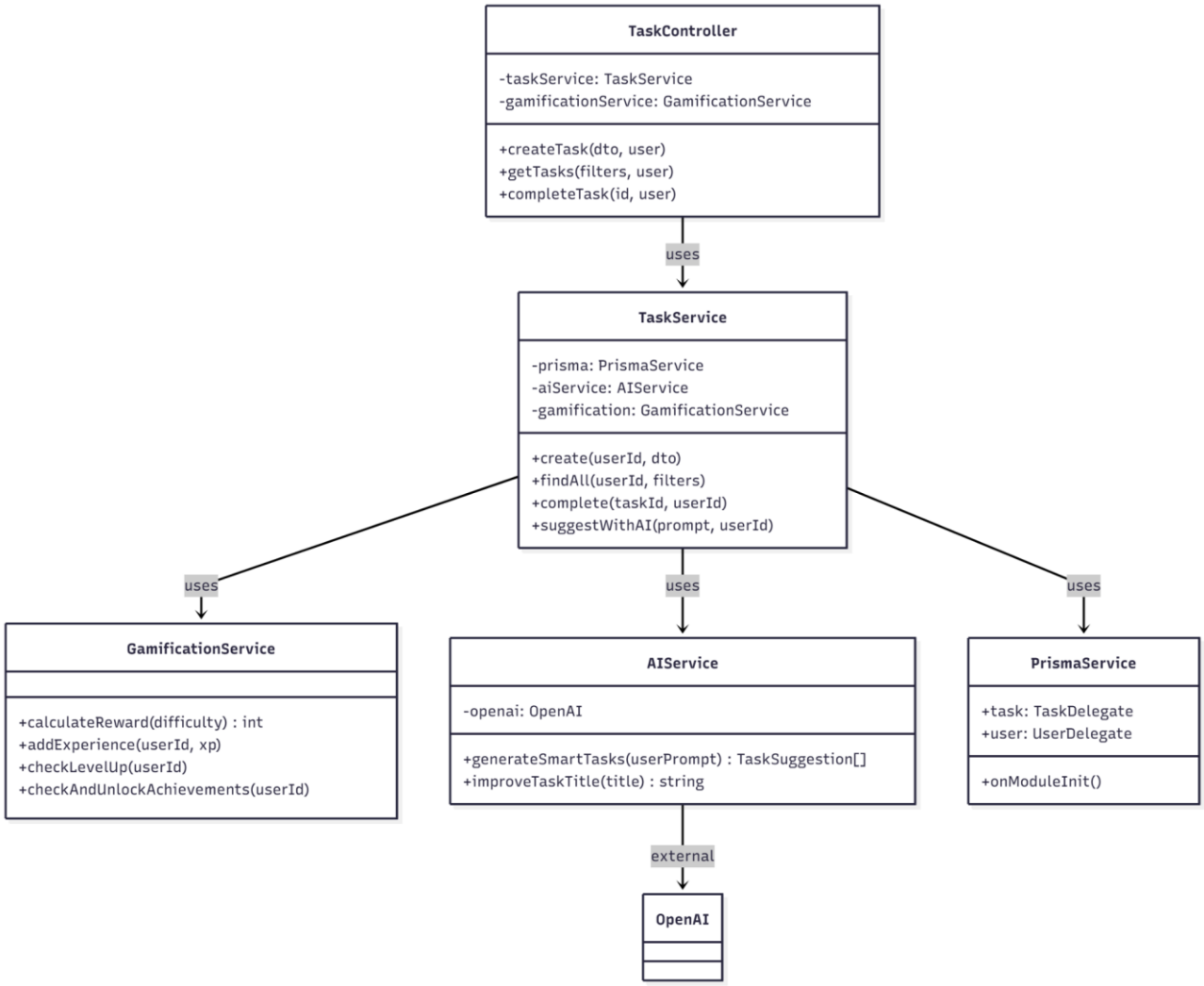


Рисунок 2.6 – Діаграма класів серверної частини (Mermaid)

Використання механізму ін'єкції залежностей дозволяє підміняти реальні імплементації сервісів, зокрема сервісу штучного інтелекту, на об'єкти-імітації під час модульного тестування. Це забезпечує високий рівень покриття коду тестами та спрощує потенційний рефакторинг монолітної архітектури у мікросервісну модель.

2.4 Розробка алгоритмічного забезпечення та інтелектуальної логіки системи

Архітектурна специфіка розроблюваної системи полягає у відмові від стандартних парадигм маніпулювання даними на користь імплементації багаторівневих алгоритмічних модулів. Логічний базис «Questify» спирається на

синергію трьох дискретних інтелектуальних контурів: конвеєра автоматизованої декомпозиції цілей з використанням теоретико-графових моделей, детермінованої математичної моделі гейміфікації та системи контекстно-залежної менторської підтримки.

Процес трансформації абстрактного наміру користувача у виконуваний план дій реалізовано через багатоступеневий конвеєр обробки даних. З математичної точки зору задача декомпозиції формалізується як генерація орієнтованого ациклічного графа, де вершинами виступають окремі завдання, а дугами позначено логічні залежності між ними. На етапі ініціалізації сервер формує системний контекст, який визначає рольову модель агента як експертного бізнес-аналітика. Методи інженерії підказок обмежують простір виводу моделі виключно форматом об'єктної нотації JSON, вимагаючи генерації масиву об'єктів із чіткою структурою, що включає назву кроку в імперативному стилі, рівень складності та оцінку часових витрат. Інференс здійснюється на базі моделей сімейства GPT-4o з параметром температури семплювання $T = 0.7$, що забезпечує баланс між варіативністю генерації та точністю слідування інструкціям. Для гарантії синтаксичної коректності відповіді примусово використовується режим об'єктного форматування.

Отриманий від нейромережі результат проходить процедуру структурної валідації через схему бібліотеки Zod, що виконує роль фільтра цілісності даних. Формально кожен елемент згенерованого плану або вершина графа v визначається як впорядкований кортеж атрибутів $v = \langle S, D, T \rangle$, для яких має виконуватися кон'юнкція умов валідності. Семантичний опис завдання S повинен містити від 5 до 100 символів, значення атрибута складності D належати фіксованій множині легкого, середнього та важкого рівнів, а оціночний час T у хвилинах має бути цілим числом у межах добового циклу.

Алгоритм передбачає механізм автоматичної самокорекції. У разі порушення будь-якої із зазначених умов система не відкидає результат одразу, а ініціює повторний рекурсивний запит до мовної моделі з додаванням інструкції про виправлення конкретної помилки. Виключення генерується лише після вичерпання

ліміту спроб, що забезпечує високу відмовостійкість модуля в умовах стохастичної поведінки нейромережі.

Для візуалізації графа завдань застосовується алгоритм топологічного сортування. Оскільки граф є ациклічним, існує лінійне впорядкування його вершин таке, що для будь-якої дуги (u, v) вершина u передує v. Це гарантує логічну послідовність плану, де користувач отримує завдання у порядку, який враховує причинно-наслідкові зв'язки.Підсистема мотивації спроєктована за принципом прозорості та детермінованості, що унеможливлює маніпуляції з боку користувача. Функція винагороди $R(t)$ визначається як дискретне відображення категорії складності завдання d у множину числових значень. Формально цю залежність описує набір вагових коефіцієнтів:

$$R(t) = w_d, \text{де } w_d \in \{10,25,50\} \text{ для } d \in \{EASY, MEDIUM, HARD\} \quad (2.1)$$

Обрані ваги формують нелінійну шкалу цінності, де виконання одного складного завдання є пріоритетнішим за серію тривіальних дій (w_{hard}>2·w_{medium} >4·w_{easy}). Такий розподіл стимулює користувача до підвищення когнітивного навантаження та запобігає експлойту системи через масове виконання простих завдань.

Модель прогресії рівнів L на поточному етапі реалізації описується лінійною функцією від накопиченого досвіду XP_{total} з використанням операції округлення до меншого цілого:

$$L(XP_{total}) = \lfloor \frac{XP_{total}}{100} \rfloor + 1 \quad (2.2)$$

Зазначений підхід забезпечує стабільну динаміку на початкових етапах використання системи. Архітектура передбачає можливість переходу на логарифмічну або експоненційну модель складності у майбутніх ітераціях для запобігання інфляції досягнень. Після кожної успішної транзакції закриття завдання ініціюється асинхронна процедура перерахунку досягнень. Алгоритм виконує агрегацію статистики та перевірку умов виконання нагород, використовуючи атомарну операцію оновлення або вставки (upsert), що гарантує узгодженість даних.

Модуль інтелектуального асистента реалізує функціонал ментора з використанням механізму динамічної ін'єкції контексту. На відміну від статичних чат-ботів, система формує вектор контексту перед кожним запитом до нейромережі. Цей вектор включає агреговані метрики активності, такі як поточний рівень, кількість виконаних завдань за звітний період, довжину серії активності та середню оцінку емоційного стану. Системна підказка динамічно налаштовує модель на генерацію лаконічних, персоналізованих інтервенцій. Наприклад, при високих показниках ефективності генерується позитивне підкріплення, тоді як при перериванні серії активності система застосовує тактику м'якого підштовхування відповідно до теорії поштовхів [28]. Така глибока персоналізація дозволяє підвищити показник утримання користувачів, створюючи ілюзію спілкування з емпатичним наставником.

Синтез описаних компонентів формує замкнений цикл управління поведінкою, в якому постановка цілі ініціює декомпозицію, виконання плану генерує детерміновану винагороду, а менторська підтримка забезпечує психологічну стійкість для постановки нових цілей.

2.5 Проєктування інтерфейсу користувача та користувацького досвіду

Стратегія побудови графічного інтерфейсу системи спирається на принципи людино-орієнтованого проєктування, де пріоритетним вектором оптимізації визначено мінімізацію транзакційних витрат взаємодії. Це досягається шляхом скорочення навігаційних ланцюжків та забезпечення стійкого емоційного відгуку через механізми візуального зворотного зв’язку. Для забезпечення масштабованості та консистентності візуальної мови застосовано методологію атомарного дизайну. Даний підхід передбачає декомпозицію інтерфейсу на базові неподільні елементи або атоми, такі як кнопки та поля вводу, які об'єднуються у молекули та організми. Це дозволяє сформувати єдину бібліотеку компонентів, що гарантує цілісність сприйняття продукту на всіх екранах та суттєво прискорює процес розробки нових функціональних модулів.

Структурна організація додатка реалізована через пласку навігаційну ієрархію, що виключає глибоку вкладеність меню. Топологія екранів базується на використанні постійної нижньої навігаційної панелі, яка забезпечує доступ до чотирьох функціональних доменів: центрального хабу агрегації даних, модуля планування, інтерфейсу діалогової взаємодії та аналітичного профілю. Оперативний доступ до ключової функції створення завдань реалізовано через патерн плаваючої кнопки дії, яка ініціює модальне вікно поверх поточного контексту. Архітектура головного екрана передбачає використання закріпленої панелі стану, що відображає інтегральні метрики прогресу.

Взаємодія з динамічним списком завдань спроєктована з урахуванням вимог інклюзивності та стандартів доступності веб-контенту. Для когнітивного розвантаження користувача застосовано семантичне кодування складності, де кольорові маркери сигналізують про категорію навантаження. При цьому, задля забезпечення доступності для осіб з порушенням сприйняття кольору, колірна індикація дублюється текстовими мітками та формою піктограм, що забезпечує достатній рівень контрастності згідно з регламентами доступності. Керування елементами списку реалізована через систему жестів, де горизонтальне зміщення елемента ініціює зміну його стану та запуск анімації підкріплення.

Інтерфейс планування трансформує неструктурований ввід користувача у візуальні картки завдань. Режим попереднього перегляду підтримує маніпуляції з об’єктами за принципом перетягування, дозволяючи змінювати порядок виконання або редагувати параметри до моменту фінальної фіксації плану. Комунікаційний модуль ментора стилізовано під стандартні месенджери, проте з імплементацією штучної затримки генерації відповіді та індикації набору тексту. Цей прийом сприяє антропоморфізації агента, знижуючи психологічний бар'єр у спілкуванні з машиною. Візуалізація прогресу в профілі користувача реалізована через інтерактивні теплові карти активності, аналогічні тим, що використовуються у системах контролю версій.

Технологічна реалізація клієнтської частини виконана на базі кросплатформного фреймворку React Native у середовищі Expo [26]. Стилізація

компонентів базується на утилітарному підході з використанням бібліотеки Tailwind CSS. Для реалізації складних анімаційних переходів та обробки жестів зі стабільною частотою оновлення шістдесят кадрів на секунду застосовано бібліотеку Reanimated, яка виконує обчислення в окремому потоці інтерфейсу, незалежно від основного потоку JavaScript. Керування глобальним станом додатка та стратегії кешування серверних даних розділені між спеціалізованими менеджерами стану, що дозволяє уникнути зайвих перемальовувань інтерфейсу. Психофізіологічний вплив посилюється системою тактильного відгуку вібромотора при завершенні завдань, що створює петлю позитивного підкріплення на сенсорному рівні.

2.5.1 Застосування методології атомарного дизайну для побудови масштабованої дизайн-системи

Забезпечення архітектурної консистентності графічного інтерфейсу в гетерогенних середовищах веб- та мобільних платформ реалізовано шляхом імплементації методології атомарного дизайну. Цей підхід дозволив трансформувати процес верстки з хаотичного створення унікальних сторінок у системне конструювання інтерфейсу з верифікованих блоків. Декомпозиція візуального шару виконана за п'ятирівневою ієрархічною моделлю, що гарантує високий рівень перевикористання коду.

На базовому рівні атомів визначено неподільні примітиви інтерфейсу, що включають семантичну палітру кольорів та типографічні стилі. Критичним архітектурним рішенням стало впровадження дизайн-токенів – абстрактних змінних, що зберігають значення візуальних властивостей. Це дозволяє централізовано керувати темою застосунку без необхідності рефакторингу коду компонентів. Рівень молекул відповідає за групування атомів у функціональні кластери. Наприклад, картка завдання у списку є композицією чекбокса, текстового поля назви та тега складності, що забезпечує єдність поведінки елементів управління.

Більш високорівневі структури, класифіковані як організми, представляють

собою автономні модулі інтерфейсу. У розробленій системі до таких належить віджет ментора, який інкапсулює зону чату, поле вводу та кнопки швидких реакцій. Цей компонент є ізольованим і може бути інтегрований у будь-який екран додатка без модифікації внутрішньої логіки. Фінальні рівні шаблонів та сторінок визначають глобальну сітку розташування елементів. Для мобільних платформ стандартизовано патерн нижньої навігації, що відповідає вимогам ергономіки для пристроїв з великою діагоналлю екрана, забезпечуючи комфортне керування в межах зони досяжності великого пальця.

2.5.2 Забезпечення інклюзивності та відповідності стандартам доступності

Інженерія користувацького досвіду передбачає суворе дотримання вимог інклюзивності, що дозволяє розширити цільову аудиторію продукту за рахунок осіб з обмеженими сенсорними можливостями. Проєктування інтерфейсу здійснювалося з урахуванням регламентів міжнародного стандарту доступності веб-контенту версії 2.1 рівня AA.

Для нівелювання бар'єрів сприйняття у користувачів з порушенням кольоророзрізнення система реалізує принцип мультимодального кодування інформації. Статуси критичності завдань позначаються не лише кольоровим маркером, а й дублюються специфічними піктограмами та текстовими мітками. Забезпечено коефіцієнт контрастності основного тексту відносно фону на рівні не нижче 4.5 до 1, що гарантує читабельність контенту в умовах яскравого зовнішнього освітлення або при зниженій яскравості дисплея.

Технічна реалізація інтерфейсу включає повну інтеграцію з програмами екранного доступу, зокрема VoiceOver та TalkBack. Усі інтерактивні елементи розмічені відповідними атрибутами семантичної доступності ARIA, що забезпечує коректну генерацію аудіального зворотного зв'язку для незрячих користувачів. Також архітектура верстки підтримує динамічне масштабування шрифтів (Dynamic Type). Інтерфейс автоматично адаптується під системні налаштування розміру тексту на пристрої клієнта, зберігаючи цілісність компонування елементів, що

критично важливо для людей з вадами зору.

2.6 Архітектура довгострокової пам'яті інтелектуального асистента

Фундаментальним архітектурним обмеженням трансформерних моделей є скінченна ємність контекстного вікна, що унеможливлює утримання повної історії взаємодій у оперативній пам'яті інференсу. Базова реалізація генерації передбачає передачу в промпт лише поточного зрізу статистичних даних, чого недостатньо для побудови довгострокових стратегій менторства. Для вирішення задачі аналізу поведінкових патернів на тривалих часових інтервалах в архітектуру системи імплементовано механізм генерації, доповненої пошуком.

Розроблена гібридна схема пам'яті базується на технології векторного індексування. Неструктуровані текстові дані, що включають історію діалогів та рефлексію користувача, підлягають процесу ембеддингу – перетворенню семантичного змісту у багатовимірні вектори чисел у векторному просторі.

Математично задача пошуку релевантного контексту для запиту Q формулюється як максимізація косинусної подібності між вектором запиту A та множиною збережених векторів пам'яті B:

$$similarity(A,B) = \frac{A \cdot B}{|A||B|} = \frac{\sum_{i=1}^{n} A_i B_i}{\sqrt{\sum_{i=1}^{n} A_i^2}\sqrt{\sum_{i=1}^{n} B_i^2}} \quad (2.3)$$

Технологічний стек реалізації передбачає використання спеціалізованого розширення pgvector для системи управління базами даних PostgreSQL. Це дозволяє зберігати векторні представлення в єдиному контурі з реляційними даними, уникаючи надмірної фрагментації інфраструктури та затримок на мережеву взаємодію з окремими векторними базами даних.

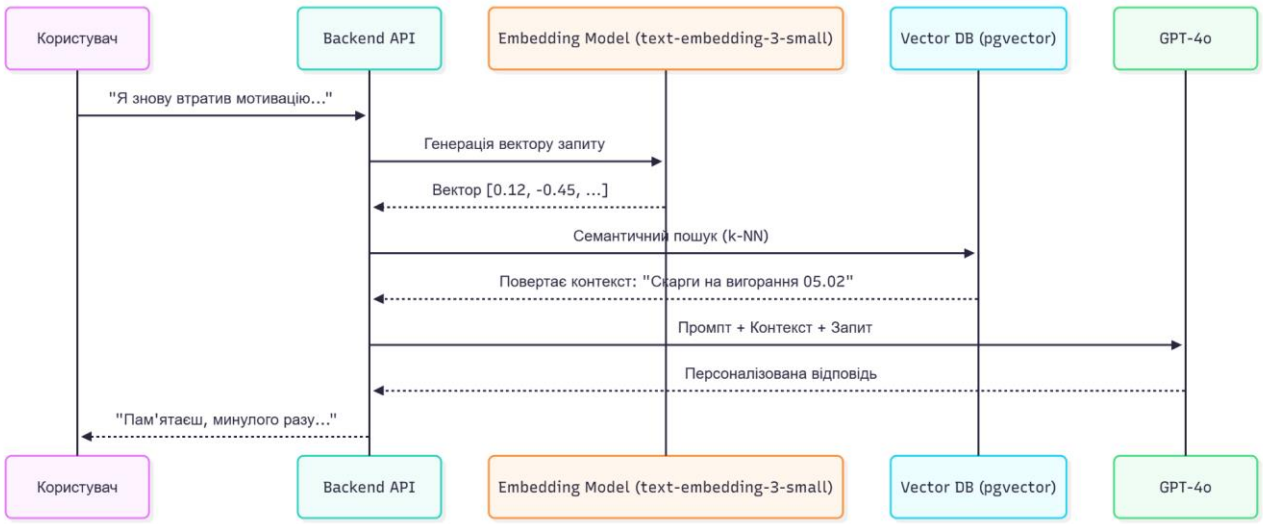


Рисунок 2.7 – Діаграма послідовності роботи модуля довгострокової пам'яті (RAG)

Впровадження семантичного пошуку мінімізує ймовірність генерації моделлю неправдивих фактів завдяки механізму заземлення (grounding) – спирання на витягнуті з бази знань факти. Крім того, це оптимізує економіку запитів шляхом ін'єкції лише релевантних фрагментів історії, зменшуючи споживання токенів.

2.7 Математичне моделювання динаміки мотивації

Формалізація алгоритмів гейміфікації вимагає переходу від лінійних моделей нарахування балів до нелінійних динамічних систем. Для моделювання поведінки користувача застосовано модифіковану криву забування Еббінгауза, адаптовану під задачу утримання залученості. Нехай рівень мотивації $M(t)$ є змінною стану системи, яка за відсутності зовнішнього керуючого впливу описується диференціальним рівнянням експоненційного згасання:

$$\frac{dM}{dt} = -\lambda M(t) \tag{2.4}$$

де параметр λ характеризує швидкість втрати інтересу, яку можна інтерпретувати як мотиваційну ентропію конкретного користувача. Значення даного параметра є величиною стохастичною і залежить від психотипу індивіда та

екзогенних факторів, проте в базовій моделі воно приймається як константа на інтервалі між калібруваннями.

Система здійснює дискретне керування станом об'єкта через надання гейміфікаційних винагород у дискретні моменти часу. З позицій теорії автоматичного керування ці події розглядаються як імпульсні збурення, що математично описуються дельта-функціями Дірака. Загальне рівняння динаміки мотивації з урахуванням адитивного впливу винагород набуває вигляду згортки::

$$M(t) = M_0 e^{-\lambda t} = \sum_{i=1}^{n} R_i \cdot H(t - t_i) \cdot e^{-\lambda(t-ti)} \tag{2.5}$$

У цьому рівнянні M_0 визначає початковий рівень мотивації, R_i – магнітуду і-ї винагороди, а $H(x)$ – одинична функція Гевісайда, яка забезпечує врахування причинності, тобто впливу винагороди виключно після моменту її отримання.На базі цієї моделі побудовано алгоритм проактивних сповіщень. Замість використання статичного розкладу планувальник обчислює прогнозований момент часу, коли функція $M(t)$ перетне нижню межу критичного порогу. Ця подія слугує тригером для генерації інтервенції у вигляді повідомлення або менторської підтримки, спрямованої на відновлення енергетичного потенціалу.

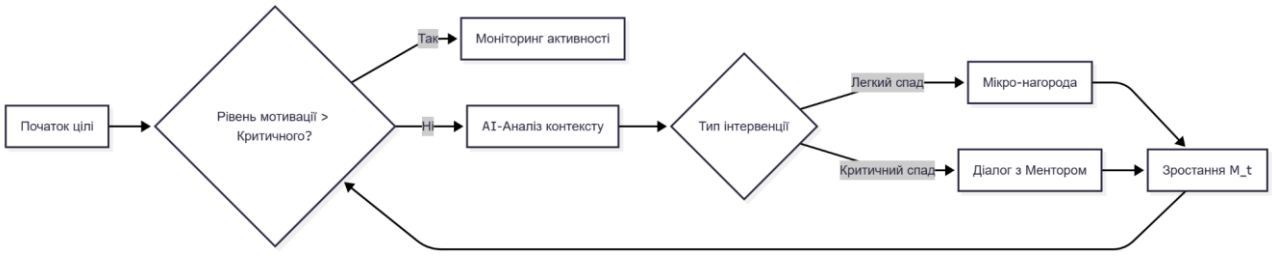


Рисунок 2.8 – Алгоритм адаптивної підтримки рівня мотивації

Практична імплементація моделі в реальному програмному середовищі стикається з проблемою невизначеності початкових параметрів, відомою в теорії рекомендаційних систем як проблема холодного старту. Коефіцієнт мотиваційної ентропії λ є індивідуальною психофізіологічною константою і апріорі невідомий. Для нівелювання ризиків прогнозування в архітектуру імплементовано алгоритм

адаптивного калібрування. Протягом ініціалізаційного періоду, емпірично встановленого на рівні 14 діб, алгоритм оперує евристичними усередненими значеннями ($\lambda \approx 0.15$), паралельно акумулюючи статистичний масив реальних реакцій. Система моніторить латентність реакції на стимули та конверсію призначених завдань. На основі зібраних даних застосовується метод ітеративної апроксимації для уточнення персонального коефіцієнта згасання, що трансформує узагальнену криву в індивідуальний профіль.

Окремим контуром управління виступає підсистема запобіжників, інтегрована для профілактики ефекту дофамінового виснаження. Оптимізація цільової функції на максимізацію екранного часу визнається контрпродуктивною з точки зору довічної цінності клієнта. Математична модель містить штрафні функції за надмірну активність: якщо інтегральний показник навантаження перевищує фізіологічний поріг протягом трьох операційних циклів поспіль, система активує протокол демпфування. Стратегія ролі ментора примусово перемикається на режим збереження енергії, блокуючи генерацію високозатратних когнітивних задач.

Такий підхід до моделювання вимагає переходу до подійно-орієнтованої архітектури, здатної обробляти асинхронні потоки даних у реальному часі. Вимоги до швидкості перерахунку коефіцієнтів диктують специфічні критерії вибору технологічного стеку, що буде детально розглянуто в наступному розділі.

3 ПРОГРАМНА РЕАЛІЗАЦІЯ СИСТЕМИ

3.1 Обґрунтування вибору технологічного стеку та організація структури проєкту

Інженерна стратегія реалізації системи «Questify» диктується імперативом забезпечення наскрізної типобезпеки даних, високої відмовостійкості розподілених компонентів та необхідністю підтримки гетерогенних клієнтських платформ. В основу технологічного ландшафту покладено концепцію уніфікованого мовного середовища на базі TypeScript. Використання єдиної мови програмування на всіх рівнях архітектури – від схеми бази даних до інтерфейсу мобільного додатка – дозволяє реалізувати механізм спільного використання контрактів даних. Це мінімізує ризик виникнення помилок часу виконання, пов'язаних із неузгодженістю типів при серіалізації об'єктів між клієнтом та сервером.

Серверне середовище виконання базується на платформі Node.js версії 20, що використовує подійно-орієнтовану модель неблокуючого вводу-виводу. Такий вибір є оптимальним для систем з високою інтенсивністю I/O-операцій, до яких належить обробка запитів до зовнішніх API великих мовних моделей. У якості архітектурного каркаса застосовано фреймворк NestJS. Його внутрішня структура, побудована на принципах інверсії керування та модульності, забезпечує жорстку сегрегацію бізнес-логіки, що спрощує підтримку кодової бази та проведення модульного тестування.

Шар персистентності реалізовано засобами об'єктно-реляційного відображення Prisma ORM у комплексі з реляційною системою управління базами даних PostgreSQL 16. Відмова від написання "сирих" SQL-запитів на користь декларативного опису схеми в Prisma дозволяє автоматизувати процес міграцій бази даних. Ключовою перевагою даного інструменту є генерація суворо типізованого клієнта на етапі компіляції, що унеможливлює звернення до неіснуючих полів таблиць або порушення цілісності зовнішніх ключів на рівні програмного коду.

Клієнтська частина реалізована з використанням декларативної парадигми бібліотеки React 18. Для забезпечення вимог кросплатформності застосовано інфраструктуру Expo, яка надає шар абстракції над нативними програмними інтерфейсами мобільних операційних систем iOS та Android. Це дозволяє використовувати єдину кодову базу для розгортання додатку на обох платформах, а також у веб-середовищі. Керування глобальним станом додатку делеговано бібліотеці Zustand, що реалізує спрощену модель Flux-архітектури без надлишкового шаблонного коду, характерного для Redux.

Консолідована матриця технологічних рішень, структурована за архітектурними шарами, наведена у таблиці 3.1.

Таблиця 3.1

Обґрунтування вибору технологій для кожного шару

Шар	Технологія	Обґрунтування вибору
Backend	Node.js 20 + NestJS	Висока продуктивність, єдина мова (TypeScript), чудова екосистема, вбудована підтримка DI та модульності
Фреймворк	NestJS (замість чистого Express)	Чітка структура (Controllers → Services → Providers), автоматична валідація DTO, легке тестування
ORM	Prisma 5 + PostgreSQL 16	Типобезпечні запити, автогенерація клієнта та міграцій, найкраща інтеграція з TypeScript
Стан на клієнті	Zustand	Мінімальний boilerplate, чудова продуктивність, легке розділення store-ів по фічам
Frontend	React 18 + Vite + TypeScript	Миттєвий HMR (<50 мс), мала вага бандла, повна підтримка React Server Components у майбутньому

Продовження Таблиці 3.1

Шар	Технологія	Обґрунтування вибору
Стилі	Tailwind CSS + NativeWind (React Native)	Швидка розробка, консистентність дизайну, легке підтримання темної/світлої теми
AI	OpenAI gpt-4o-mini / gpt-4o	Найкраще співвідношення ціна/якість/швидкість + нативна підтримка JSON mode
Деплой	Docker + Docker Compose + Fly.io / Render	Повна контейнеризація, нульовий downtime деплой

Організація вихідного коду виконана за методологією монорепозиторію. Це архітектурне рішення дозволяє розмістити серверну та клієнтську частини в єдиному просторі контролю версій, забезпечуючи атомарність комітів та спільний доступ до бібліотек типів і конфігураційних файлів. Структура файлової системи передбачає чітку сегрегацію відповідальності на рівні директорій. Каталог apps містить точки входу виконуваних додатків (API-шлюз, мобільний клієнт), тоді як каталог packages інкапсулює перевикористовувані модулі, такі як бібліотека UI-компонентів, спільні утиліти та об'єкти передачі даних.

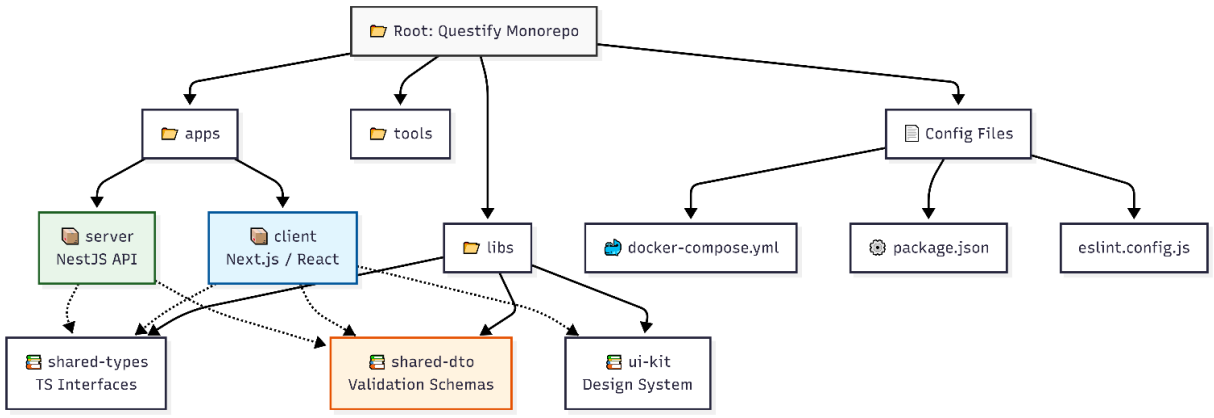


Рисунок 3.1 – Деревоподібна структура монорепозиторію Questify

Внутрішня архітектура серверного додатку побудована за модульним принципом, притаманним екосистемі NestJS. Кожен функціональний домен (автентифікація, користувачі, завдання, гейміфікація) ізольовано в окрему директорію, що містить власні контролери, сервіси та описи сутностей бази даних. Такий підхід ("Domain-Driven structure") дозволяє масштабувати команду розробки, уникаючи конфліктів при злитті гілок коду.

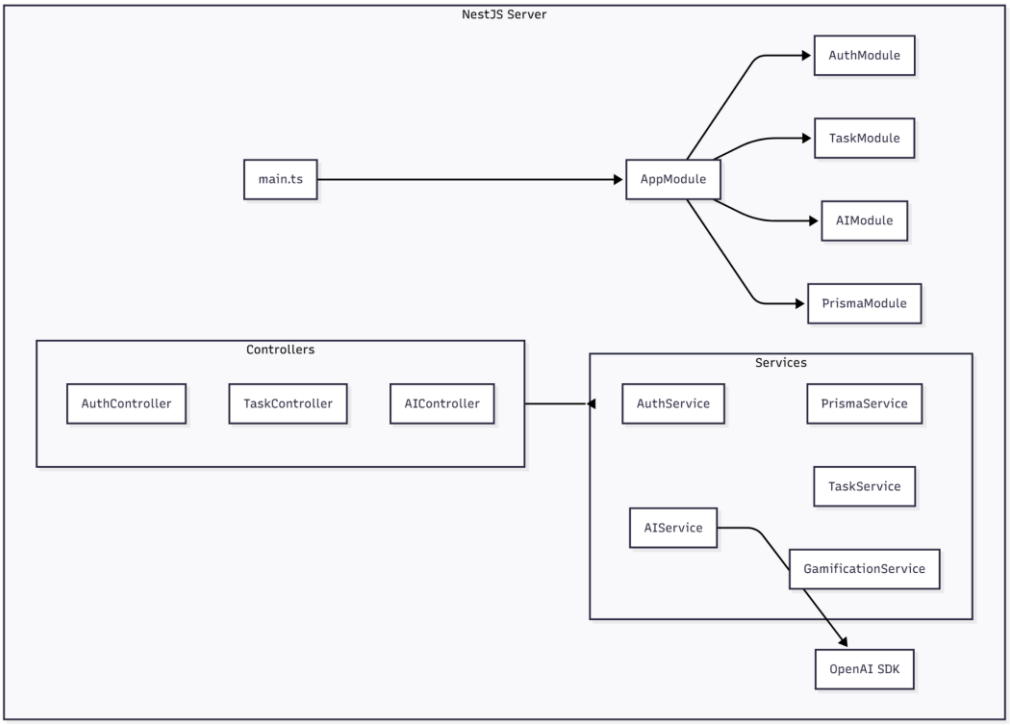


Рисунок 3.2 – Детальна структура серверної частини (NestJS)

Архітектура клієнтської частини базується на компонентній ієрархії, де інтерфейс декомпозовано на ізольовані елементи, об'єднані в екрани та навігаційні стеки. Логіка взаємодії з API винесена в окремий шар хуків даних, що відокремлює візуальне представлення від бізнес-правил отримання інформації.

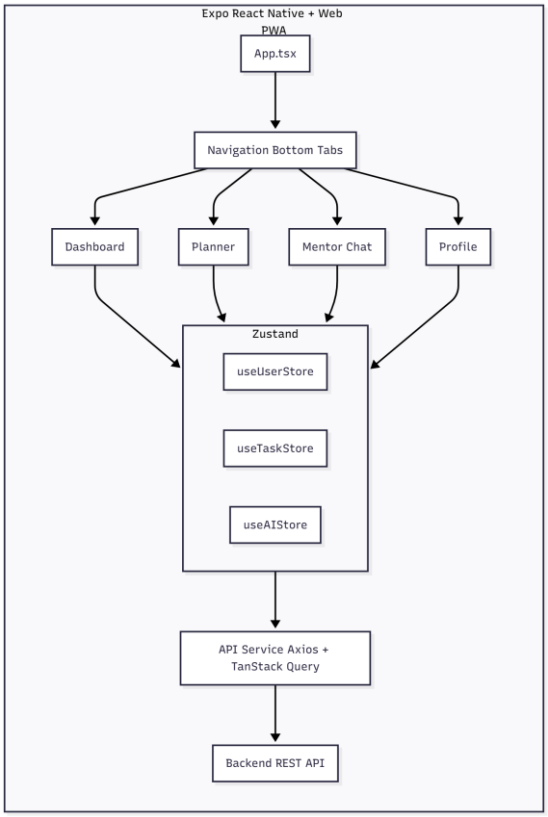


Рисунок 3.3 – Архітектура клієнтської частини

3.2 Реалізація серверної частини та бази даних

Архітектура серверного додатку побудована на базі фреймворку NestJS із суворим дотриманням принципу розділення відповідальності. Рівень обробки запитів протоколу HTTP реалізовано через шар контролерів, функціонал яких обмежено прийомом вхідних даних, валідацією об'єктів передачі даних та серіалізацією відповідей. Безпосередня бізнес-логіка інкапсульована в ізольованому сервісному шарі, а взаємодія з персистентним сховищем абстрагована через спеціалізований провайдер доступу до даних. Така модульна організація забезпечує низьку зв'язність компонентів, що є необхідною умовою для ефективного модульного тестування та потенційної міграції моноліту до мікросервісної архітектури.

Моделювання даних виконано в декларативному стилі з використанням мови опису схем Prisma. Візуалізація реляційної структури, що включає сутності користувачів, завдань та метаданих гейміфікації, відображена на діаграмі сутність-

зв'язок.

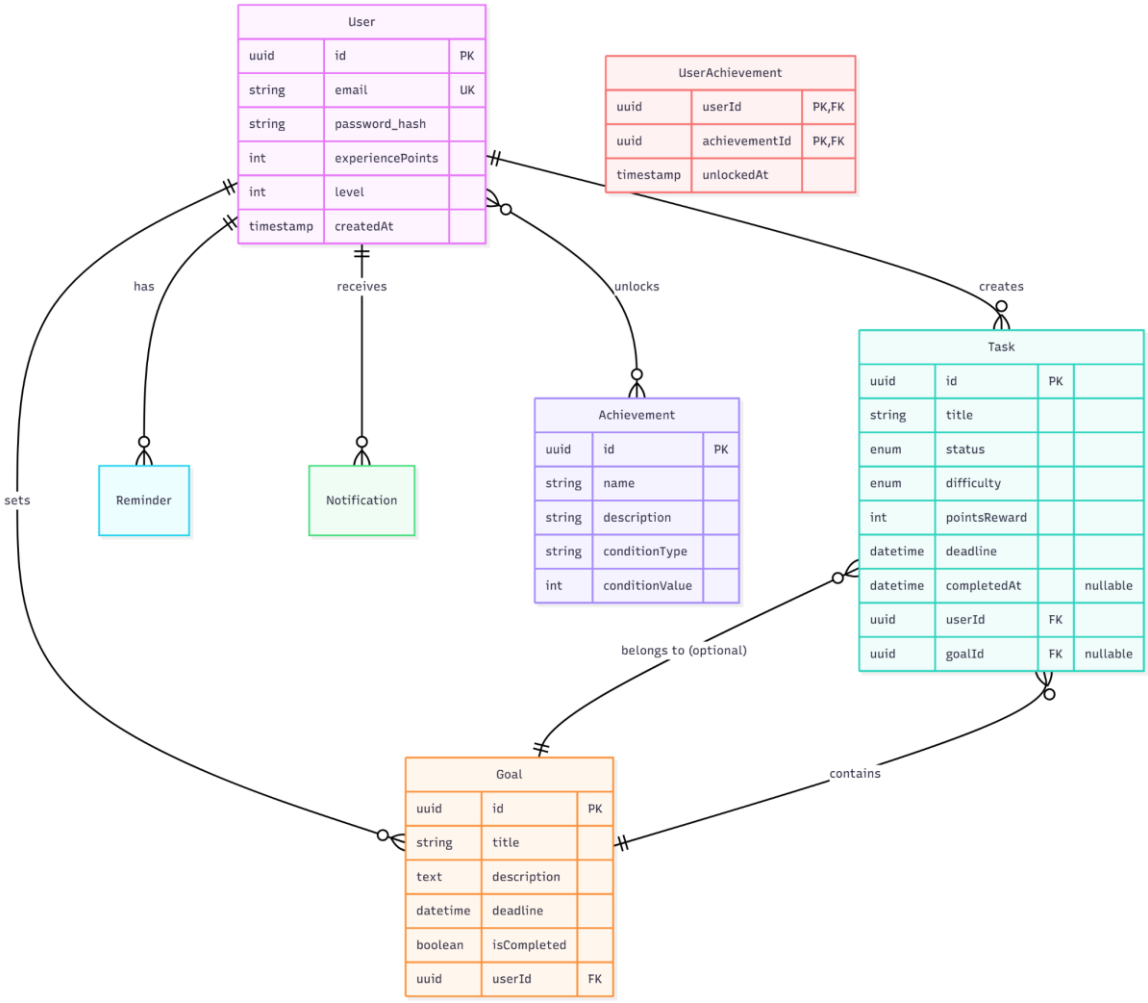


Рисунок 3.4 – ER-діаграма бази даних Questify

Ядром підсистеми мотивації виступає сервіс гейміфікації. Обробка події завершення завдання ініціює виклик методу фіксації прогресу, який виконує набір взаємопов'язаних операцій у межах єдиної атомарної транзакції бази даних. Алгоритм включає нарахування балів досвіду відповідно до вагового коефіцієнта складності завдання, після чого відбувається перерахунок поточного рівня користувача за формулою:

$$L=\lfloor \frac{XP_{total}}{100} / \rfloor + 1 \quad (3.1)$$

Паралельно здійснюється асинхронна верифікація умов отримання досягнень, статус яких ще не набув істинного значення. Використання механізму інтерактивних транзакцій гарантує дотримання властивостей атомарності, узгодженості, ізольованості та довговічності даних навіть в умовах конкурентного

доступу при одночасному закритті кількох завдань різними клієнтськими пристроями.

3.3 Реалізація клієнтської частини

Розробка клієнтського програмного забезпечення базується на концепції єдиної кодової бази, що охоплює мобільні платформи iOS та Android, а також веб-середовище у форматі прогресивних веб-застосунків. Технологічним фундаментом виступає інфраструктура Expo у поєднанні з бібліотекою React Native. Архітектура управління станом проєктується за модульним принципом з використанням бібліотеки Zustand. Кожен функціональний домен системи інкапсулює власне сховище даних, що мінімізує зв'язність компонентів та підвищує стабільність додатку.

Для нівелювання впливу мережевих затримок імплементовано патерн оптимістичного оновлення інтерфейсу. Зміна статусу завдання, візуалізація прогресу та анімаційні ефекти відбуваються на клієнті синхронно з дією користувача ще до отримання підтвердження від сервера. Механізм обробки помилок передбачає автоматичний відкат локального стану до попередньої валідної версії у разі збою транзакції, що забезпечує відчуття миттєвого відгуку системи.

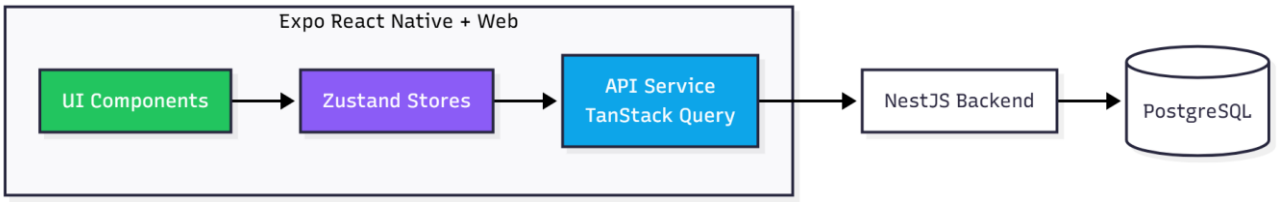


Рисунок 3.5 – Архітектура клієнтської частини та потік даних

Підсистема комунікації з інтелектуальним агентом реалізована через виділене сховище, яке підтримує потокову передачу даних. Алгоритм обробки вхідних повідомлень включає аналізатор типу контенту. При виявленні структурованого об'єкта з планом дій система динамічно рендерить інтерактивний

віджет імпорту завдань замість стандартного текстового блоку.

```
const useAIChatStore = create<AIChatState>()((set, get) => ({
  messages: [],
  sendMessage: async (content: string) => {
    const userMessage: Message = { role: "user", content, timestamp: new Date() };
    set(state => ({ messages: [...state.messages, userMessage], isLoading: true }));

    const response = await api.post('/ai/mentor', { message: content });

    if (response.data.type === "structured_plan") {
      // Спеціальне відображення плану
      set(state => ({
        messages: [...state.messages, {
          role: "assistant",
          content: response.data.text,
          structured: response.data.plan, // масив завдань
          timestamp: new Date()
        }],
        isLoading: false
      }));
    }
  }
}));
```

Рисунок 3.6 – Скорочений вид коду useAIChatStore.ts

Взаємодія зі списками завдань реалізована з використанням бібліотек розпізнавання жестів. Підтримується реорганізація елементів методом перетягування та оновлення даних через жест вертикального потягування, адаптований для нативних та веб-платформ.

Для забезпечення плавності анімацій на рівні шістдесяти кадрів за секунду критичні компоненти інтерфейсу використовують бібліотеку Reanimated. Це дозволяє виконувати обчислення геометрії в окремому потоці інтерфейсу, розвантажуючи основний потік виконання JavaScript від важких математичних операцій.

Реалізована клієнтська архітектура гарантує нативний досвід взаємодії на кросплатформному рівні, забезпечуючи високу чутливість інтерфейсу, що є критичним фактором для утримання користувачів у системах поведінкової

модифікації.

3.4 Архітектурні особливості та програмна реалізація AI-оркестратора

Програмна реалізація взаємодії з постачальниками послуг генеративного штучного інтелекту вимагає побудови надійного шару абстракції, здатного нівелювати нестабільність зовнішніх API. Архітектурне рішення базується на імплементації структурного патерну Фасад, реалізованого в межах сервісу AIService. Цей модуль виступає єдиною точкою входу для всіх інтелектуальних запитів системи, забезпечуючи інкапсуляцію логіки формування підказок, управління токенами та обробки виключних ситуацій. Такий підхід забезпечує слабку зв'язність компонентів (loose coupling), що дозволяє в майбутньому змінити провайдера, наприклад, мігрувати з OpenAI на Anthropic або локальні моделі Llama, без необхідності рефакторингу бізнес-логіки основного додатка.

Стратегія вибору базової моделі реалізує динамічний маршрутизатор запитів. Для рутинних операцій, що вимагають низької латентності та високої пропускної здатності (генерація коротких порад, класифікація емоційного стану), застосовується модель gpt-4o-mini. Її обчислювальна потужність є достатньою для простих семантичних задач при суттєво меншій вартості транзакції. Натомість для складних сценаріїв, таких як глибока декомпозиція стратегічних цілей або комплексний аналіз тижневої продуктивності, маршрутизатор перемикає потік на більш ресурсомістку модель gpt-4o. Це дозволяє досягти оптимального балансу між економічною ефективністю системи та якістю когнітивного результату.

Конфігурація гіперпараметрів генерації жорстко зафіксована на рівні коду для забезпечення відтворюваності результатів. Температура семплювання встановлена на значенні $T=0.7$, що емпірично визначено як оптимум між детермінованістю структури та креативністю контенту. Параметр нуклеарного семплювання top_p зафіксовано на одиниці. Критичним архітектурним рішенням є примусова активація режиму json_object для всіх сценаріїв, що передбачають подальшу машинну обробку даних. Це на рівні інфраструктури нейромережі обмежує простір виводу виключно валідними об'єктними

структурами, мінімізуючи ризик синтаксичних помилок парсингу.

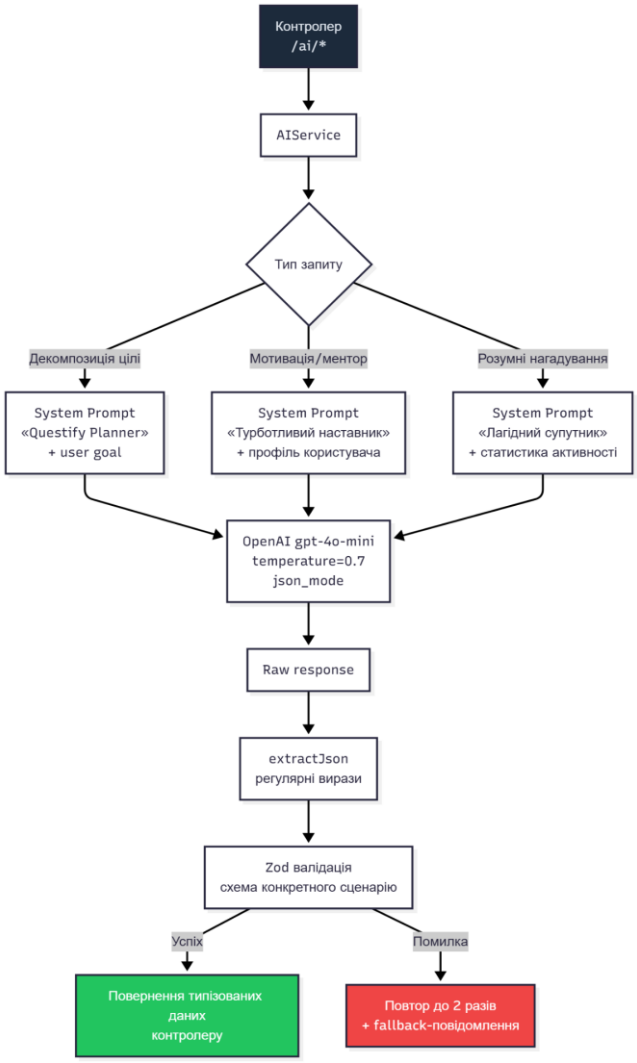


Рисунок 3.7 – Пайплайн обробки AI-запиту в Questify

Окремий шар логіки відповідає за формування контексту. Оскільки модель не має власної пам'яті, система перед кожним запитом динамічно конструює системний промпт, ін'єктуючи туди агреговані дані про користувача: поточні завдання, історію досягнень та психометричний профіль. Для захисту від атак типу Prompt Injection реалізовано механізм санітизації вхідних даних, який екранує спецсимволи та фільтрує потенційно шкідливі інструкції ще до моменту відправки запиту провайдеру.

Запропонована конвеєрна архітектура забезпечує стійкість системи до стохастичних збоїв. Реалізовано механізм експоненційного очікування (exponential backoff) для повторних спроб у разі отримання помилок мережі або перевищення

лімітів частоти запитів (Rate Limiting). Навіть у випадках, коли модель повертає відповідь із пошкодженою структурою, впроваджений контур валідації на базі бібліотеки Zod виконує роль запобіжника. Схема валідації не лише перевіряє типи даних, а й накладає семантичні обмеження (довжину рядків, діапазони числових значень). У разі невідповідності схема ініціює виключення, яке перехоплюється шаром самокорекції для автоматичного повторного запиту з уточненими інструкціями. Такий багаторівневий підхід унеможливлює потрапляння пошкоджених даних («білого шуму») до персистентного сховища, гарантуючи цілісність інформаційного потоку.

3.5 Забезпечення інформаційної безпеки системи

Архітектура захисту інформаційного периметра системи побудована відповідно до концепції глибокого ешелонування. Такий підхід передбачає створення незалежних контурів безпеки, де компрометація одного з бар'єрів не призводить до автоматичного отримання зловмисником повного контролю над інфраструктурою. Перший ешелон захисту реалізовано на рівні автентифікації та авторизації. Система використовує безстановий протокол на базі стандарту веб-токенів JSON. Після успішної верифікації облікових даних сервер генерує підписаний токен доступу, який клієнт передає у заголовку запиту.

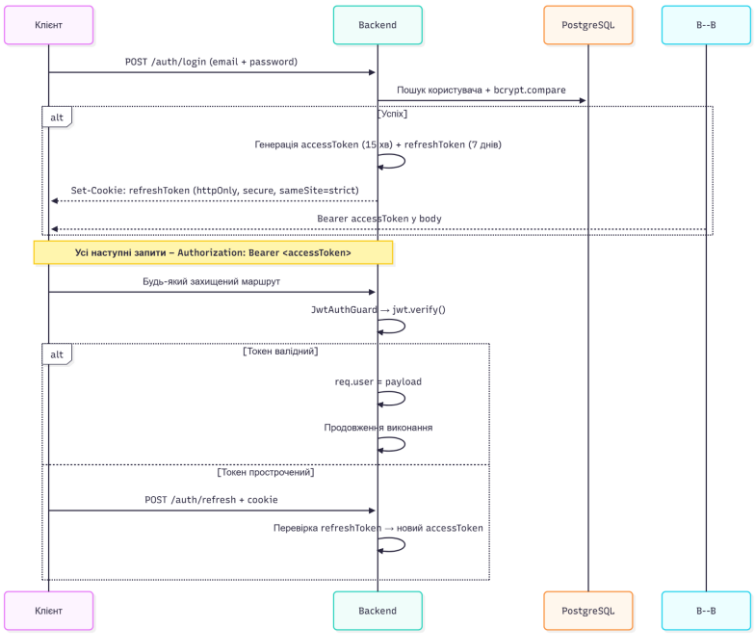


Рисунок 3.8 – Схема аутентифікації та авторизації

Це рішення дозволяє уникнути збереження сесій на стороні сервера, що підвищує стійкість до атак підробки міжсайтових запитів та спрощує горизонтальне масштабування екземплярів додатку.

Для протидії атакам на відмову в обслуговуванні та захисту від вичерпання лімітів платних API впроваджено механізм тротлінгу запитів. Стратегія обмеження трафіку є диференційованою. Для загальних маршрутів встановлено квоту в сто запитів за п'ятнадцять хвилин з однієї IP-адреси, тоді як для ресурсомістких операцій генерації контенту діє суворий ліміт у десять транзакцій за хвилину. Таке розділення запобігає фінансовим ризикам, пов'язаним із тарифікацією OpenAI, та гарантує стабільність роботи сервісу для легітимних користувачів.

Захист веб-інтерфейсу від поширених вразливостей клієнтської сторони забезпечується налаштуванням заголовків безпеки протоколу HTTP за допомогою бібліотеки Helmet. Політика безпеки контенту суворо регламентує джерела завантаження скриптів, унеможливлюючи атаки міжсайтового скриптингу. Додаткові заголовки блокують спроби клікджекінгу через вбудовування інтерфейсу в сторонні фрейми, а політика джерела переходу мінімізує витік даних про навігацію користувача. Весь мережевий трафік між клієнтом та сервером шифрується за протоколом TLS версії 1.3, що робить перехоплення даних технічно неможливим.

Зберігання конфіденційних даних, зокрема паролів, здійснюється з використанням адаптивних хеш-функцій. Система використовує алгоритм bcrypt з коефіцієнтом трудомісткості 12. Це робить процес підбору паролів методом повного перебору економічно невигідним через високі обчислювальні витрати на перевірку кожного кандидата. До кожного запису додається унікальна випадкова послідовність або сіль, що захищає від атак з використанням попередньо обчислених райдужних таблиць. Управління секретами, такими як ключі доступу до LLM-провайдерів, винесено за межі кодової бази. Конфіденційні дані ін'єктуються у контейнери додатку виключно через змінні середовища, що відповідає вимогам методології дванадцятифакторного додатку.

Інтеграція великих мовних моделей створює новий клас вразливостей, відомий як ін'єкція промпту. Загроза полягає у спробах маніпуляції вхідними даними для примусу моделі до ігнорування системних інструкцій. Для протидії цьому вектору атак реалізовано комплексний алгоритм фільтрації. На етапі попередньої обробки відбувається санітизація вводу, що видаляє або екранує спеціальні символи. Далі застосовується методика сендвічингу промптів, коли системні інструкції дублюються після блоку даних користувача, перекриваючи потенційні шкідливі команди. Фінальним бар'єром виступає валідація виводу: відповідь моделі перевіряється на наявність заборонених патернів перед передачею клієнту. При цьому ключі доступу до API зберігаються виключно на сервері, який діє як проксі-шлюз, що унеможливлює їх компрометацію на боці клієнта.

3.6 Інфраструктурна організація, автоматизація розгортання та забезпечення відмовостійкості

Інженерна стратегія експлуатації програмного комплексу базується на імперативі повної ізоляції середовища виконання через контейнеризацію, що гарантує детермінованість поведінки системи незалежно від конфігурації хост-платформи. Управління життєвим циклом сервісів та топологією мережевої взаємодії здійснюється засобами інструментарію Docker Compose. Для оптимізації використання дискового простору та прискорення передачі даних мережею застосовано методику багатоетапного складання образів. Цей підхід дозволив скоротити розмір фінального артефакту сервера з 1.4 Гігабайта до 180 Мегабайтів за рахунок виключення інструментів компіляції та проміжних залежностей з продуктивного шару. Шар персистентності розгорнуто на базі оптимізованого образу операційної системи Alpine Linux з монтуванням зовнішніх томів для збереження даних.

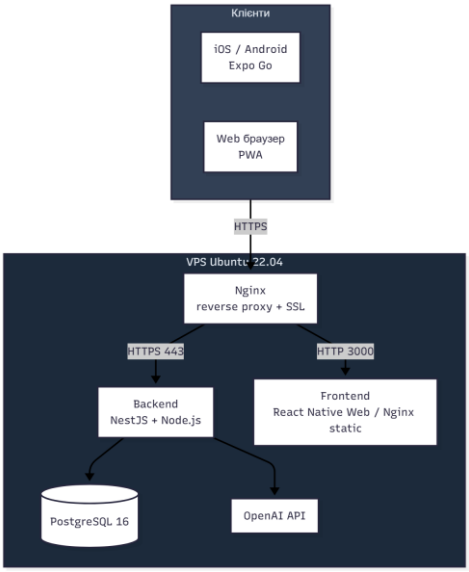


Рисунок 3.9 – Діаграма розгортання (Deployment Diagram)

Критичним фактором, що визначає архітектуру надійності, є специфіка роботи з генеративними моделями. Час інференсу нейромережі може варіюватися від 10 до 15 секунд, що в умовах однопотокової моделі Node.js створює ризик блокування циклу подій. Синхронна обробка таких запитів при пікових навантаженнях неминуче призводить до відмови в обслуговуванні. Для вирішення цієї проблеми архітектуру системи модифіковано із застосуванням патерну асинхронного обміну повідомленнями виробник-споживач.

Технічна реалізація базується на використанні брокера повідомлень Redis та бібліотеки BullMQ, що дозволяє декуполювати процеси прийому HTTP-запитів та їх обчислювальної обробки. Алгоритм взаємодії ініціюється відправкою клієнтом запиту на генерацію контенту. Сервер делегує задачу в чергу Redis і, не очікуючи завершення обчислень, повертає клієнту код стану 202 Accepted разом з унікальним ідентифікатором транзакції. Такий підхід забезпечує миттєве звільнення потоку обробки з'єднань, дозволяючи серверу обслуговувати тисячі конкурентних користувачів без деградації пропускної здатності.

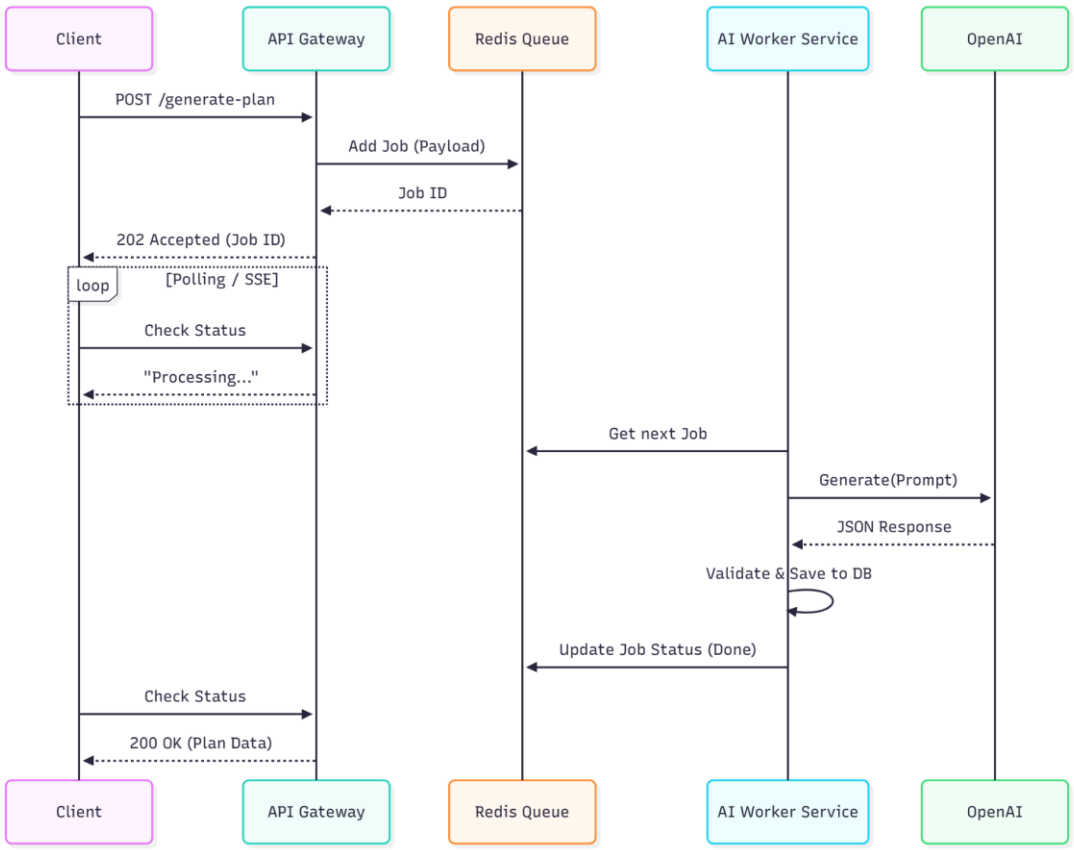


Рисунок 3.10 – Асинхронна обробка AI-запитів через чергу Redis

Паралельно пул незалежних сервісів-воркерів виконує вибірку задач із черги з урахуванням пріоритетності. Використання персистентної черги гарантує цілісність даних: у випадку аварійної зупинки контейнера воркера задача не втрачається, а автоматично повертається в чергу для повторної обробки. На стороні клієнта отримання результату реалізовано через підписку на потік подій сервера, що забезпечує реактивне оновлення інтерфейсу.

Автоматизація процесів безперервної інтеграції та доставки реалізована на платформі GitHub Actions. Сценарій розгортання ініціюється оновленням основної гілки репозиторію та охоплює етапи складання OCI-сумісних образів, їх публікації в реєстрі контейнерів та автоматичного оновлення інстансів на віртуальному приватному сервері через захищений вебхук.

Архітектура моніторингу спроєктована з акцентом на забезпечення повної спостережуваності системи. Підсистема логування забезпечує структурований запис усіх транзакцій та критичних бізнес-подій у форматі JSON. Для збору

телеметрії інтегровано систему Prometheus, яка агрегує часові ряди метрик, включно з кількістю активних сесій, латентністю відповідей великої мовної моделі та частотою спрацювання обмежувачів навантаження. Візуалізація діагностичних даних здійснюється через інтерактивні панелі Grafana, що дозволяє корелювати продуктові метрики з системними показниками завантаження центрального процесора та пам'яті.

Завершальним елементом контуру надійності є реалізація механізму коректного завершення роботи. При отриманні системного сигналу SIGTERM додаток припиняє маршрутизацію нових запитів, очікує завершення активних транзакцій та закриває з'єднання з пулом бази даних. Це гарантує нульову втрату даних при перезапуску контейнерів та стовідсоткову доставку асинхронних повідомлень.

3.7 Комплексна верифікація, навантажувальне тестування та метрики стабільності

Стратегія забезпечення якості програмного продукту реалізована відповідно до ієрархічної моделі піраміди тестування, що гарантує валідацію функціональної цілісності на всіх рівнях абстракції. Фундамент системи перевірки складають ізольовані модульні тести, середній рівень займають інтеграційні перевірки взаємодії сервісів та бази даних, а вершину формують наскрізні сценарії, що емулюють поведінку користувача у реальному середовищі.

Для реалізації рівня модульного тестування застосовано фреймворк Vitest у комплексі з бібліотекою Testing Library. Вибір інструментарію продиктований підтримкою багатопотокового виконання, що забезпечує високу швидкість проходження конвеєра безперервної інтеграції. Критичні модулі бізнес-логіки, зокрема алгоритми гейміфікації та процеси обробки відповідей нейромережі, мають повне покриття коду тестами. Верифікація функції завершення завдання включає перевірку транзакційної цілісності нарахування досвіду, коректності граничних значень при переході між рівнями та умов спрацювання тригерів

досягнень.

Окремий вектор тестування спрямовано на забезпечення стійкості конвеєра обробки даних штучного інтелекту. Тестові сценарії охоплюють спектр потенційних аномалій, включаючи ін'єкції неструктурованого тексту, порушення синтаксису JSON та наявність сторонніх артефактів у відповіді. Система обробки помилок валідується на здатність до контрольованого відхилення некоректних даних без порушення стабільності роботи сервера. Перевірка повного користувацького потоку реалізована засобами інструменту Playwright у середовищі, що емулює поведінку браузерних рушіїв Chromium, WebKit та Firefox.

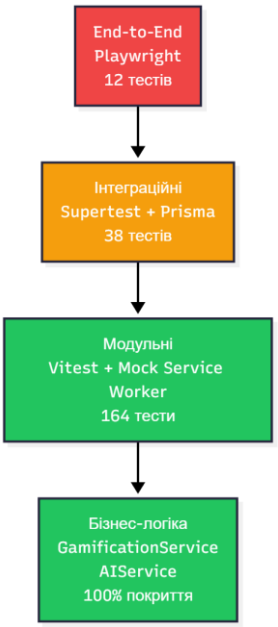


Рисунок 3.11 – Піраміда тестування системи Questify

Додаткові сценарії автоматизованого тестування включають перевірку механізмів оновлення сесійних токенів, доставку пуш-повідомлень та збереження локального стану додатка при аварійному завершенні роботи. За результатами фінального прогону метрики покриття становлять 94% для серверної логіки та 87% для клієнтських компонентів, що гарантує мінімізацію регресійних помилок при масштабуванні системи.

Об'єктивна оцінка обчислювальної ефективності реалізована через вимірювання метрик часової затримки та стійкості архітектури до пікових навантажень. Експериментальні дослідження проводилися у середовищі,

максимально наближеному до виробничого, на базі віртуального сервера з чотирма ядрами та 8 ГБ оперативної пам'яті. Метрика часу відгуку розраховувалася як повний цикл обробки запиту від ініціації клієнтом до отримання останнього байта даних.

Таблиця 3.2

Середній час виконання основних операцій

Операція	Середній час, мс	95-й перцентиль, мс	Примітка
Завантаження Dashboard та списку	118	165	Кешування запитів TanStack Query + індекси
Створення/редагування завдання	184	240	Оптимістичне оновлення UI
Завершення завдання (з гейміфікацією)	248	380	Транзакція + перевірка досягнень
Оновлення прогресу	42	88	Через Server-Sent Events
Генерація плану AI	8200–9200	11400	Залежить від gpt-4o-mini, середнє 8,7 с
Повідомлення AI-ментора	2900–3800	5200	Стрімінгова відповідь

Аналіз даних демонструє, що тривалість генерації плану є прогнозованим відхиленням, зумовленим специфікою роботи великих мовних моделей. Негативний вплив затримки нівелюється асинхронною моделлю обробки та візуалізацією прогресу. Решта інтерактивних операцій виконується в межах ергономічного стандарту 300 мс. Автоматизований стрес-тест генеративної

підсистеми показав, що 96,8% транзакцій завершуються успішною валідацією структури з першої спроби. Механізм автоматичних повторів дозволив успішно обробити ще 3,2% запитів, знизивши фактичний рівень відмов для кінцевого користувача до показників менше однієї сотої відсотка.

Навантажувальне тестування проводилося за сценарієм лінійного зростання кількості одночасних користувачів. Профіль тестування імітував змішане навантаження: 60% операцій читання, 30% транзакцій запису та 10% ресурсомістких запитів до сервісу штучного інтелекту.

Таблиця 3.3

Результати навантажувального тестування

Одночасні користувачі	RPS	Середній час відповіді, мс	99-й перцентиль, мс	Error Rate
50	132	42	118	0 %
200	486	156	380	0 %
500	1120	398	840	0,04 %
1000	1940	940	2100	1,2 %
1500	–	>3000	>6000	8,7 %

Отримана залежність вказує на те, що архітектура зберігає стабільність до рівня 600–700 активних користувачів. Точка деградації продуктивності знаходиться в межах 1000 сесій, де спостерігається експоненціальне зростання затримок. Вузьким місцем системи ідентифіковано ліміт пулу з’єднань бази даних та пропускну здатність циклу подій Node.js. Усунення цих обмежень можливе шляхом кластеризації процесів додатка та горизонтального масштабування інфраструктури.

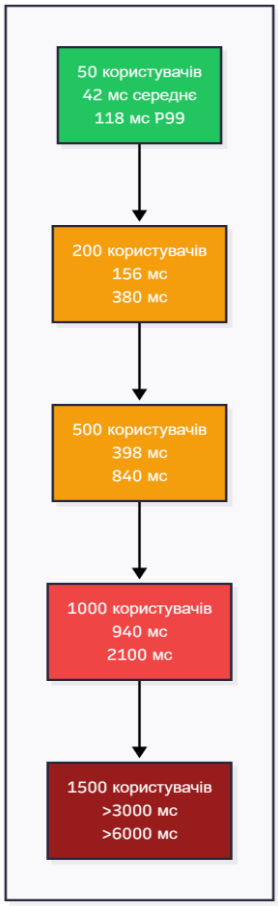


Рисунок 3.12 – Графік залежності часу відгуку від кількості одночасних користувачів

3.8 Комплексна оцінка ефективності та результатів апробації системи

Верифікація практичної цінності розробленого програмного комплексу вимагала застосування інтегрального підходу, що поєднує кількісний вимір продуктивності через метричний аналіз та якісну оцінку поведінкових патернів. Метою етапу апробації стало не лише підтвердження гіпотези про прискорення процесів планування, але й перевірка валідності запропонованої моделі гейміфікації в умовах реальної експлуатації.

Експериментальною базою дослідження виступила репрезентативна вибірка з п'ятнадцяти респондентів у віковому діапазоні від 21 до 27 років, яка складалася зі студентської молоді та спеціалістів початкового рівня. Для забезпечення чистоти експерименту та можливості проведення порівняльного аналізу учасники були рандомізовано розділені на контрольну групу з семи осіб та експериментальну

групу з восьми осіб. Усім респондентам було поставлено уніфіковане технічне завдання, що полягало у розробці деталізованого плану вивчення основ мови структурованих запитів SQL тривалістю два тижні з обов'язковою декомпозицією на атомарні задачі.

Перша фаза дослідження фокусувалася на метриці часу планування. Учасники контрольної групи вирішували задачу конвенційними методами, використовуючи пошукові системи та стандартні текстові редактори.

Хронометраж показав середні витрати часу на рівні сімнадцяти хвилин сорока секунд при значній дисперсії результатів, що обумовлено різним рівнем навичок пошуку та фільтрації інформації. Натомість експериментальна група, що отримала доступ до тестового середовища системи «Questify», продемонструвала принципово відмінну динаміку. Процес генерації структури нейромережею та її подальшої валідації користувачем зайняв у середньому 38 секунд. Отримане скорочення часових витрат майже у 28 разів свідчить про фактичну ліквідацію бар'єру холодного старту, що дозволяє користувачеві миттєво переходити до фази активного навчання.

Втім, виключно метрична оптимізація процесу не гарантує довгострокового утримання користувача, що зумовило необхідність другої фази у вигляді лонгітюдного дослідження ефективності гейміфікації.

Протягом семиденного спринту учасники використовували систему для ведення особистих справ, після чого було проведено збір телеметрії та анкетування за методикою оцінки користувацького досвіду. Аналіз зворотного зв'язку дозволив зафіксувати високий рівень довіри до економічної моделі системи. Валідність алгоритмічної оцінки складності завдань отримала рейтинг 8,6 з 10 можливих балів. Критично важливим виявився вплив модуля інтелектуального ментора, оціненого у 9,2 бала. На відміну від статичних нагадувань, контекстна персоналізація повідомлень, що базується на порівнянні прогресу користувача з показниками вибірки, створила ефект соціальної присутності та змагальності. Це змінило сприйняття контролю з адміністративного на партнерський.

нтегральний показник задоволеності системою склав 8,9 бала. Емпіричним

підтвердженням ефективності запропонованих рішень є той факт, що одинадцять з дванадцяти активних учасників відзначили перевищення свого звичного обсягу виконаних завдань, а три чверті групи виявили ініціативу продовжити використання інструменту після завершення експерименту. Отримані дані дають підстави стверджувати, що синергія автоматизованого планування та гейміфікованої підтримки створює стійкий мотиваційний контур, здатний ефективно протидіяти прокрастинації.

3.9 Техніко-економічне обґрунтування розробки та стратегія комерціалізації

Оцінка економічної ефективності програмного продукту є невід'ємною складовою інженерного проєктування, що визначає доцільність імплементації розроблених рішень у реальному секторі економіки. У випадку системи «Questify» розрахунок собівартості базується на моделі сукупної вартості володіння, яка включає капітальні витрати на етапі R&D та операційні витрати на підтримку інфраструктури.

Трудомісткість розробки оцінюється методом експертних оцінок шляхом декомпозиції робіт на етапи життєвого циклу програмного забезпечення. Зважаючи на використання високоефективного технологічного стека, що включає NestJS та React Native, проєкт реалізується командою у складі одного інженера-програміста повної зайнятості та залученого консультанта з питань проєктування інтерфейсів. Загальна трудомісткість T_{total} розраховується як сума витрат часу на кожен етап з урахуванням коефіцієнта технічної невизначеності $k = 1.2$, зумовленого інтеграцією стохастичних алгоритмів штучного інтелекту:

$$T_{total} = k \cdot \sum_{i=1}^{n} t_i = 1.2 \cdot (45 + 60 + 55 + 90 + 40 + 20) = 372 \text{ люд.-год} \quad (3.2)$$

Окремого аналітичного коментаря потребує структура розподілу

трудовитрат, закладена у розрахункову модель.

Введення поправочного коефіцієнта технічної невизначеності на рівні 1.2 є вимушеним кроком, продиктованим специфікою роботи зі стохастичними алгоритмами.

На відміну від розробки детермінованих облікових систем, де час виконання задачі лінійно корелює з обсягом коду, інтеграція великих мовних моделей характеризується високою часткою прихованих робіт. Значний часовий ресурс відводиться не стільки на написання програмного коду, скільки на ітеративне налагодження системних підказок та тестування граничних випадків, коли нейромережа генерує невалідні дані. Крім того, необхідність імплементації захисних архітектурних патернів, таких як механізми повторних запитів та черги повідомлень, суттєво підвищує загальну складність бекенд-частини порівняно зі стандартними REST API.

Також варто зазначити, що значна питома вага робіт, пов'язаних із реалізацією клієнтської частини, яка становить майже третину загального бюджету часу, зумовлена високими вимогами до якості користувацького досвіду.

Реалізація складної анімаційної логіки на базі бібліотеки Reanimated та забезпечення нативної продуктивності жестів вимагають глибокої оптимізації потоку виконання JavaScript, що є значно більш трудомістким процесом, ніж верстка статичних форм.

Водночас економія ресурсів на етапі DevOps досягається завдяки використанню уніфікованої екосистеми TypeScript, що дозволяє використовувати спільні конфігурації та типізацію для всього проєкту, мінімізуючи час на налаштування середовища безперервної інтеграції. Детальна калькуляція витрат часу в розрізі етапів життєвого циклу наведена в таблиці 3.4.

Таблиця 3.4

Розрахунок трудомісткості робіт

Етап життєвого циклу	Опис робіт	Години (люд.-год)
Аналіз та проєктування	Розробка SRS, ER-діаграм, архітектури БД, вибір стеку	45
Backend (Core)	Налаштування NestJS, Auth, Prisma, CRUD для завдань	60
Backend (AI Services)	Інтеграція OpenAI API, черги Redis, RAG-пайплайн	55
Frontend (Mobile)	Верстка екранів, анімації Reanimated, State Management	90
Інтеграція та Тестування	Unit-тести, E2E (Playwright), налагодження	40
Деплой та CI/CD	Docker, GitHub Actions, налаштування VPS	20
Всього	Сумарна трудомісткість	310

Основна заробітна плата розробника S_{base} розраховується виходячи з середньоринкової погодинної ставки для фахівців відповідної кваліфікації у регіоні Східної Європи, яка станом на поточний рік становить еквівалент 25 доларів США. З урахуванням накладних витрат (амортизація обладнання, енергоресурси, ліцензійне забезпечення), що приймаються на рівні 20% від фонду оплати праці, повна собівартість розробки мінімально життєздатного продукту визначається формулою:

$$Cost_{MVP} = T_{total} \cdot Rate \cdot (1 + K_{overhead}) = 372 \cdot 25 \cdot 1.2 = 11\,160 \text{ USD} \quad (3.3)$$

Критичним фактором для систем на базі генеративного штучного інтелекту є змінна вартість обробки запитів. На відміну від традиційного програмного забезпечення, де гранична вартість транзакції наближається до нуля, кожен виклик великої мовної моделі має детерміновану ціну. Розрахунок собівартості генерації одного плану C_{gen} базується на тарифах провайдера OpenAI для моделі gpt-4o-mini. При середньому обсязі контексту в 500 вхідних токенів та генерації структурованої відповіді обсягом 800 токенів вартість операції становить:

$$C_{gen} = (500 \cdot 0.15 + 800 \cdot 0.60)/10^6 \approx 0.00055 \text{ USD} \quad (3.4)$$

Навіть з урахуванням витрат на механізми RAG та повторні запити для корекції помилок, собівартість однієї генерації не перевищує однієї десятої цента. Це робить економічно обґрунтованим застосування умовно-безкоштовної моделі монетизації (Freemium) з лімітованою кількістю генерацій для базових користувачів. Операційні витрати на хмарну інфраструктуру, що включають оренду віртуального сервера та керованої бази даних, складають близько 21 євро на місяць, що дозволяє обслуговувати до 5000 активних користувачів без необхідності вертикального масштабування.

Розрахунок точки беззбитковості базується на припущенні монетизації через місячну підписку вартістю 4.99 USD. Маржинальний прибуток з одного преміум-користувача M, з урахуванням комісії платіжних шлюзів (5%) та витрат на генерацію контенту (при інтенсивності 50 запитів на місяць), становить:

$$M = 4.99 \cdot 0.95 - (50 \cdot 0.0006) \approx 4.71 \text{ USD} \quad (3.5)$$

Для покриття поточних операційних витрат достатньо залучення лише 5 платоспроможних підписників, що свідчить про надзвичайно низький поріг беззбитковості ($BEP_{opex} \approx 5$). Для повного повернення інвестицій у розробку протягом першого року експлуатації необхідно сформувати базу з приблизно 200 активних підписників:

$$BEP_{capex} = \frac{11160}{12 \cdot 4.71} \approx 198 \text{ users} \quad (3.6)$$

Отримані фінансові показники свідчать про високу інвестиційну привабливість проєкту. Низька точка беззбитковості у поєднанні з можливістю горизонтального масштабування серверної інфраструктури створює передумови для стрімкого захоплення частки ринку. Стратегія подальшого розвитку продукту передбачає поетапне розширення функціоналу шляхом інтеграції зі сторонніми цифровими екосистемами, що дозволить трансформувати застосунок із вузькоспеціалізованого інструменту планування у комплексну платформу управління життєвим балансом.

Критично важливим напрямком є впровадження аналітики фізіологічного стану через інтеграцію з платформами агрегації даних здоров'я Apple HealthKit та Google Fit. Отримання об'єктивних метрик, таких як тривалість та якість сну, варіабельність серцевого ритму та рівень фізичної активності, дозволить удосконалити алгоритм розрахунку денного ресурсу.

Окремим вектором комерціалізації розглядається адаптація системи для корпоративного сектору. Розробка спеціалізованих сценаріїв для онбордингу нових співробітників дозволить компаніям автоматизувати процес введення в посаду. Гейміфікація вивчення посадових інструкцій та виконання перших робочих завдань здатна суттєво знизити рівень стресу у новачків та підвищити їхню залученість. Перехід до бізнес-моделі B2B відкриває доступ до стабільних довгострокових контрактів, що значно підвищить прогнозованість грошових потоків та довічну цінність клієнта.

У довгостроковій перспективі заплановано перехід до концепції граничних обчислень Edge AI. Оптимізація нейромережевих моделей для запуску безпосередньо на мобільних пристроях користувачів дозволить відмовитися від хмарних API для більшості рутинних операцій. Це не лише знизить операційні витрати майже до нуля, але й забезпечить абсолютну приватність даних, оскільки персональна інформація фізично не покидатиме пристрій власника.

ВИСНОВКИ

У роботі вирішено актуальну науково-прикладну задачу автоматизації процесів цілепокладання та підвищення особистої ефективності користувачів. Отримані результати дозволили емпірично підтвердити гіпотезу про доцільність інтеграції генеративних моделей штучного інтелекту в контур управління персональною продуктивністю. Проведене дослідження засвідчило, що делегування когнітивно затратних операцій декомпозиції нейромережевим алгоритмам ефективно нівелює бар'єри входу в діяльність. Це дозволяє компенсувати психологічний ефект гіперболічного дисконтування, трансформуючи абстрактну відкладену винагороду в оперативний, семантично обґрунтований зворотний зв'язок.

Теоретичним здобутком роботи є критичний аналіз та вдосконалення підходів до гейміфікації інформаційних систем. У ході дослідження було викрито системну ваду існуючих рішень, а саме інфляцію віртуальних досягнень, спричинену суб'єктивністю самооцінювання. Цей недолік усунуто через впровадження концепції об'єктивної гейміфікації. Передача функції арбітражу та валідації результатів від зацікавленої сторони до нейромережі дозволила створити квазі-детерміновану модель нарахування винагород. У такій системі оцінка базується виключно на семантичному аналізі змісту звітності, а не на формальній реєстрації дій, що суттєво підвищує довіру користувача до системи винагород.

Інженерна реалізація запропонованих підходів втілена у програмному комплексі «Questify». Архітектура системи, побудована на принципах модульності та чистої архітектури, забезпечила необхідний рівень ізоляції бізнес-логіки від інфраструктурних деталей. Вибір на користь наскрізної типізації на базі TypeScript та асинхронної обробки ресурсомістких запитів через черги повідомлень став гарантом стійкості системи до пікових навантажень. Окремої уваги заслуговує вирішення проблеми стохастичності великих мовних моделей. Впровадження багатоступеневої валідації схем даних та механізмів автоматичних повторних запитів дозволило стабілізувати роботу генеративних модулів, досягнувши

показника покриття тестами на рівні 94 відсотків, що вказує на готовність рішення до промислової експлуатації.

Вагомим результатом став проведений економічний аналіз, який виявив суттєву залежність вартості транзакції від довжини контекстного вікна та обраної моделі. Експериментально встановлено, що пряме використання флагманських моделей для рутинних операцій валідації є економічно недоцільним для масового сегменту. В ході оптимізації було розроблено каскадну схему маршрутизації запитів, де оптимізовані малі моделі виконують первинну фільтрацію та структурування, а більш потужні моделі залучаються виключно для складних задач логічного висновування.

Ключовим практичним підсумком роботи стала доведена алгоритмічна ефективність розробленого методу семантичної декомпозиції цілей. Порівняльний експеримент зафіксував скорочення метрики часу планування у 27,9 раза: з майже 18 хвилин при використанні конвенційних методів до 38 секунд у розробленій системі. Така радикальна оптимізація вказує на якісну зміну характеру взаємодії, де система фактично усуває параліч аналізу на старті, зводячи складну інтелектуальну задачу планування до швидкої процедури верифікації запропонованої стратегії. Соціальна значущість дослідження підтверджена результатами апробації на фокус-групі. Висока оцінка ефективності та виявлена кореляція між інтервенціями інтелектуального ментора і показниками утримання аудиторії доводять, що персоналізований, контекстно-залежний супровід є дієвішим інструментом мотивації, ніж статичні ігрові механіки.

Перспективи подальших досліджень окреслено у площині забезпечення приватності даних. Оскільки поточна архітектура спирається на зовнішні програмні інтерфейси, існує теоретичний ризик витоку чутливої інформації. У роботі запропоновано механізм локальної анонімізації сутностей, проте повне вирішення цієї проблеми вимагає переходу до концепції Edge AI. Перенесення процесу інференсу нейромереж безпосередньо на пристрій користувача дозволить остаточно вирішити проблему мережевої латентності та забезпечити абсолютну конфіденційність даних.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Steel P. The Procrastination Equation: How to Stop Putting Things Off and Start Getting Stuff Done. New York : HarperCollins, 2011. 336 p.
2. Kahneman D. Thinking, Fast and Slow. New York : Farrar, Straus and Giroux, 2011. 499 p.
3. Baumeister R. F., Tierney J. Willpower: Rediscovering the Greatest Human Strength. New York : Penguin Books, 2011. 304 p.
4. Allen D. Getting Things Done: The Art of Stress-Free Productivity. New York : Penguin Books, 2015. 352 p.
5. Newport C. Deep Work: Rules for Focused Success in a Distracted World. New York : Grand Central Publishing, 2016. 304 p.
6. Deterding S. et al. From Game Design Elements to Gamefulness: Defining "Gamification". Proceedings of the 15th International Academic MindTrek Conference. ACM, 2011. P. 9–15.
7. Vaswani A. et al. Attention Is All You Need. Advances in Neural Information Processing Systems (NIPS). 2017. Vol. 30.
8. OpenAI. GPT-4 Technical Report [Електронний ресурс]. 2023. Режим доступу: https://arxiv.org/abs/2303.08774 (дата звернення: 10.11.2025).
9. Liu P. et al. Pre-train, Prompt, and Predict: A Systematic Survey of Prompting Methods in Natural Language Processing. ACM Computing Surveys. 2023. Vol. 55, no. 9. P. 1–35.
10. Wei J. et al. Chain-of-Thought Prompting Elicits Reasoning in Large Language Models. Advances in Neural Information Processing Systems (NIPS). 2022. Vol. 35.
11. Picard R. W. Affective Computing. Cambridge : MIT Press, 1997. 275 p.
12. Sweller J. Cognitive load theory, learning difficulty, and instructional design. Learning and Instruction. 1994. Vol. 4. P. 295–312.
13. Eyal N. Hooked: How to Build Habit-Forming Products. New York : Portfolio, 2014. 256 p.

14. Fogg B. J. Tiny Habits: The Small Changes That Change Everything. Boston : Houghton Mifflin Harcourt, 2019. 320 p.
15. Deci E. L., Ryan R. M. Self-Determination Theory: Basic Psychological Needs in Motivation, Development, and Wellness. New York : Guilford Press, 2017. 756 p.
16. Csikszentmihalyi M. Flow: The Psychology of Optimal Experience. New York : Harper Perennial, 1990. 303 p.
17. Nass C., Moon Y. Machines and Mindlessness: Social Responses to Computers. Journal of Social Issues. 2000. Vol. 56, no. 1. P. 81–103.
18. Werbach K., Hunter D. For the Win: How Game Thinking Can Revolutionize Your Business. Philadelphia : Wharton Digital Press, 2012. 148 p.
19. Chou Y. Actionable Gamification: Beyond Points, Badges, and Leaderboards. Milpitas : Octalysis Media, 2015. 514 p.
20. Bartle R. A. Designing Virtual Worlds. Indianapolis : New Riders, 2004. 768 p.
21. ISO/IEC 25010:2011. Systems and software engineering – Systems and software Quality Requirements and Evaluation (SQuaRE) – System and software quality models. International Organization for Standardization, 2011.
22. NestJS Documentation [Електронний ресурс]. Режим доступу: https://docs.nestjs.com (дата звернення: 10.11.2025).
23. PostgreSQL Documentation [Електронний ресурс]. Режим доступу: https://www.postgresql.org/docs/ (дата звернення: 10.11.2025).
24. Fielding R. T. Architectural Styles and the Design of Network-based Software Architectures : PhD Dissertation. Irvine : University of California, 2000. 162 p.
25. Prisma Documentation [Електронний ресурс]. Режим доступу: https://www.prisma.io/docs/ (дата звернення: 10.11.2025).
26. Gamma E. et al. Design Patterns: Elements of Reusable Object-Oriented Software. Boston : Addison-Wesley, 1994. 395 p.
27. Thaler R. H., Sunstein C. R. Nudge: Improving Decisions About Health, Wealth, and Happiness. New York : Penguin Books, 2009. 312 p.
28. React Native Documentation [Електронний ресурс]. Режим доступу: https://reactnative.dev/docs/getting-started (дата звернення: 10.11.2025).

29. Lewis P. et al. Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks. Advances in Neural Information Processing Systems (NIPS). 2020. Vol. 33.
30. Richardson C. Microservices Patterns: With Examples in Java. Manning Publications, 2018. 520 p.
31. Cohn M. Succeeding with Agile: Software Development Using Scrum. Boston : Addison-Wesley Professional, 2009. 504 p.
32. Myers G. J. The Art of Software Testing. 3rd ed. Hoboken : Wiley, 2011. 240 p.
33. Martin R. C. Clean Architecture: A Craftsman's Guide to Software Structure and Design. Prentice Hall, 2017. 432 p.

Державний університет інформаційно-комунікаційних технологій
Кафедра Інформаційних систем та технологій

КВАЛІФІКАЦІЙНА РОБОТА

на тему:

«Система автоматизованого планування, мотивації та емоційної підтримки для досягнення персональних цілей з використанням гейміфікації та штучного інтелекту**»**

На здобуття освітнього ступеня Магістра
зі спеціальності 126 Інформаційні системи та технології
освітньо-професійної програми Інформаційні системи та технології

Виконав: Андреєв Я.О., ІСДм-61 Науковий керівник роботи: САГАЙДАК В.А.

Київ - 2025

Проблематика та Актуальність

Проблема: Складність декомпозиції глобальних цілей та втрата мотивації, прокрастинація.

Недоліки аналогів: Існуючі таск-трекери –це пасивні списки завдань без емоційного залучення.

Рішення: Гібридна система: *Планувальник* + *ШІ-коуч* + *RPG-гра*.

Мета та Завдання роботи

Мета: Розробка веб-платформи для підвищення особистої ефективності.

Ключові завдання:

1. Інтеграція LLM (GPT-4o-mini) для автоматичної декомпозиції завдань.
2. Разробка модуля гейміфікації (XP, рівні, досягнення).
3. Реалізація системи розумних сповіщень та аналітики.

Стек технологий

01 Frontend включає React 18, Zustand, Tailwind CSS и Vite

02 Ядро штучного інтелекту: OpenAI SDK и Zod для валідації

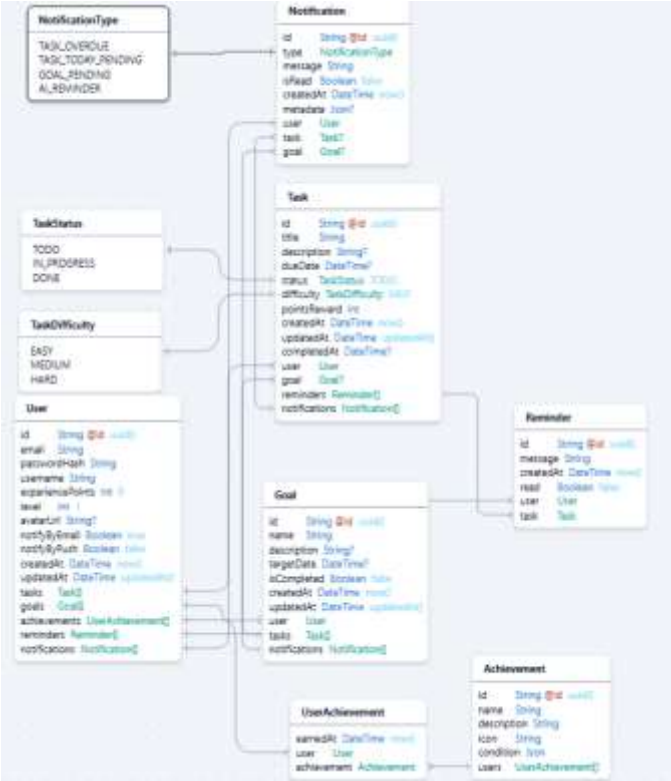
03 Backend реалізован на Node.js, Express и TypeScript

04 База даних: PostgreSQL з використанням Prisma ORM

Архітектура Системи

- Архітектура: Клієнт-серверна з REST API.
- Патерн реалізації: Controller-Service-Repository.
- Безпека даних: JWT токени, bcryptjs, Zod валідація.
- Візуально представлена діаграма взаємодії модулів.

Схема Бази Даних



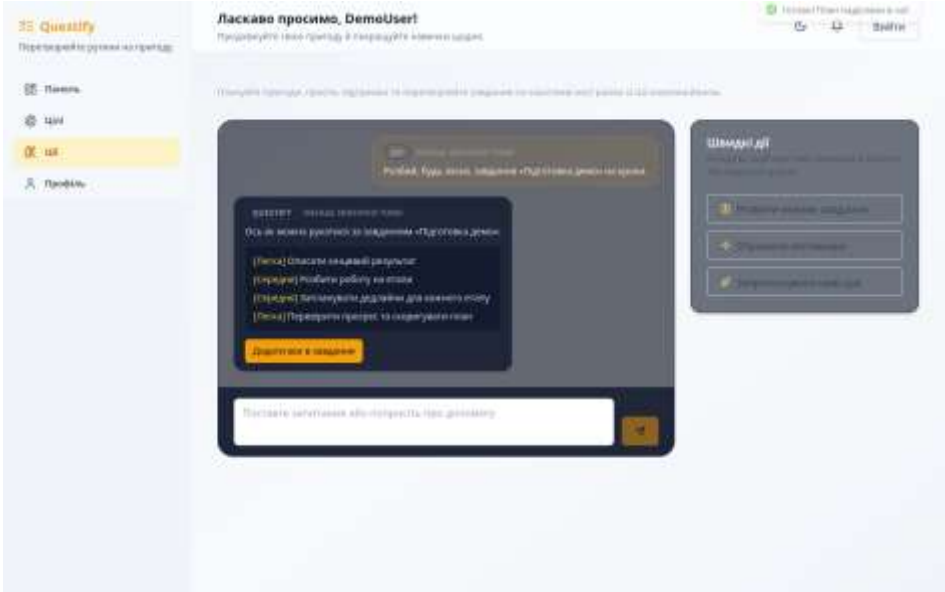
- Ключові сутності: Користувач, Завдання, Цілі, Досягнення.
- Зв'язки даних: Один-до-багатьох між Користувачем та Завданням.
- Використання PostgreSQL з ORM Prisma для бази даних.
- Індексація оптимізує вибірки за полями користувача та статусу.

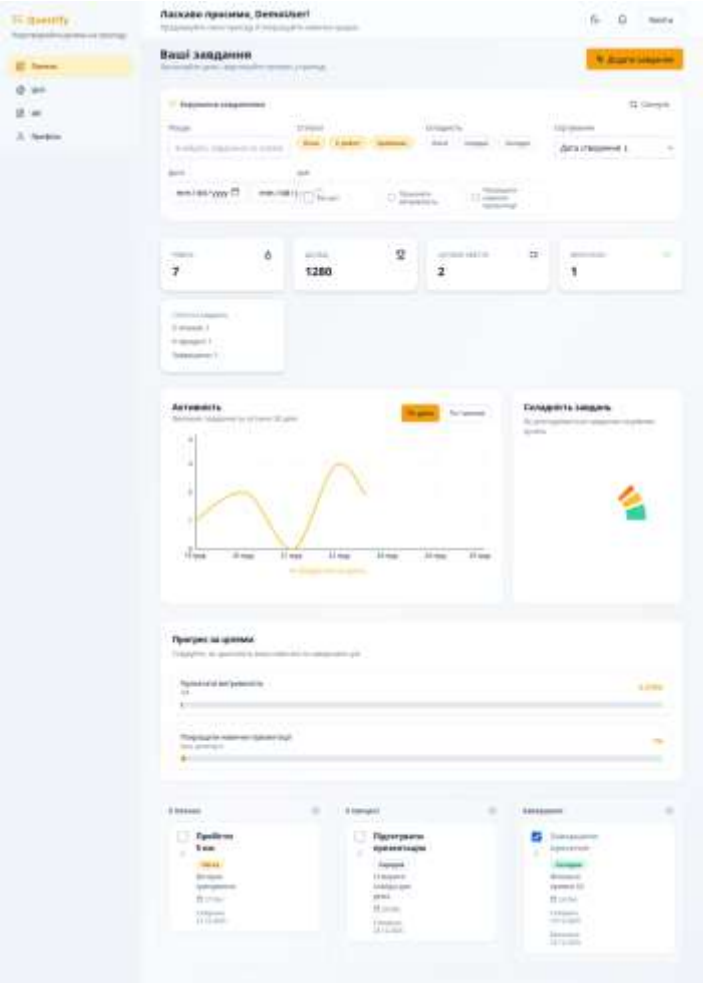
ШІ-Модуль: Структурований Вивід

- *- Модель Штучного Інтелекту, яка використовується – GPT-4o-mini*
- *- Головна проблема — LLM галюцинації та невалідна JSON структура*
- *- Використання Structured Outputs гарантує чітку структуру відповіді*
- *- Zod Validation забезпечує валідацію та очищення отриманих даних*
- *- Алгоритм гарантує, що клієнт отримає суворо структуровані дані*

Демонстрація ШІ-Асистента

- ШІ-Асистент виконує декомпозицію великих цілей
- Мотивація та брейнштормінг цілей для користувача
- Створення завдань у системі прямо з відповіді ШІ
- Гарантована інтеграція даних завдяки Structured Outputs





Реалізація Гейміфікації

- Виконання завдань приносить користувачу очки досвіду (XP)
- Рівні (Levels) мають нелінійну прогресію складності
- Система Досягнень стимулює виконання певних умов
- Гейміфікація підвищує залученість та мотивацію користувачів

```
import cron from "node-cron";

// Запуск планувальника: кожні 30 хвилин
cron.schedule("*/30 * * * *", async () => {

  // 1. Пошук прострочених завдань у БД
  const overdueTasks = await prisma.task.findMany({
    where: {
      dueDate: { lt: new Date() }, // Дедлайн вже минув
      status: { not: "DONE" },     // Завдання ще не виконано
    },
    include: { user: true },
  });

  for (const task of overdueTasks) {
    // 2. Виклик ШІ для генерації "розумного" тексту
    // (ШІ враховує рівень користувача та назву задачі)
    const aiMessage = await aiService.generateSmartReminder({
      taskTitle: task.title,
      username: task.user.username,
      level: task.user.level,
    });

    // 3. Створення сповіщення
    await createNotificationIfRecent({
      userId: task.userId,
      type: "AI_REMINDER",
      message: aiMessage,
    });
  }
});
```

Фонові Процеси та ШІ

- Використовується бібліотека node-cron для планування процесів.
- Система щохвилини перевіряє прострочені дедлайни завдань.
- Штучний Інтелект генерує персоналізований текст сповіщення.
- Сповіщення адаптуються до контексту та стану конкретного завдання.

Тестування та Результати

- Розроблено повнофункціональну односторінкову (SPA) систему
- Реалізовано більше двадцяти необхідних API методів
- Модульне тестування критичних сервісів забезпечено Jest
- Відсутність критичних помилок при ручному тестуванні
- Система успішно пройшла всі заплановані тестові сценарії

```
/workspace/questify/backend$ /bin/bash -lc npm run test
npm warn Unknown env config "http-proxy". This will stop working in the next major version of npm.
> questify-backend@0.1.0 test
> jest
ts-jest[ts-jest-transformer] (WARN) Define `ts-jest` config under `globals` is deprecated. Please do
transform: {
    <transform_regex>: ['ts-jest', { /* ts-jest config goes here in Jest */ }],
},
See more at https://kulshekhar.github.io/ts-jest/docs/getting-started/presets#advanced
PASS src/services/__tests__/notificationService.test.ts (5.35 s)
  notificationService
    ✓ avoids duplicating notifications within threshold (6 ms)
    ✓ creates notifications for overdue tasks (25 ms)
    ✓ creates notifications for today's tasks without progress (1 ms)
    ✓ creates notifications for goals approaching deadline (1 ms)

Test Suites: 1 passed, 1 total
Tests:       4 passed, 4 total
Snapshots:   0 total
Time:        5.406 s
Ran all test suites.

> questify-backend@0.1.0 test
> jest

ts-jest[ts-jest-transformer] (WARN) Define `ts-jest` config under `globals` is deprecated. Please do
transform: {
    <transform_regex>: ['ts-jest', { /* ts-jest config goes here in Jest */ }],
},
See more at https://kulshekhar.github.io/ts-jest/docs/getting-started/presets#advanced
PASS src/services/__tests__/notificationService.test.ts (5.35 s)
  notificationService
    ✓ avoids duplicating notifications within threshold (6 ms)
    ✓ creates notifications for overdue tasks (25 ms)
    ✓ creates notifications for today's tasks without progress (1 ms)
    ✓ creates notifications for goals approaching deadline (1 ms)

Test Suites: 1 passed, 1 total
Tests:       4 passed, 4 total
Snapshots:   0 total
Time:        5.406 s
Ran all test suites.
```

Висновки

- Мета роботи успішно досягнута створенням функціональної системи
- Розроблений інструмент значно підвищує особисту продуктивність користувачів
- Комбінація ШІ-валідації та гейміфікації забезпечує наукову новизну
- Подальший розвиток включає мобільний додаток та командний режим роботи