Міністерство освіти і науки України
Харківський національний університет радіоелектроніки

Факультет Комп'ютерних наук
(повна назва)
Кафедра Штучного інтелекту
(повна назва)

КВАЛІФІКАЦІЙНА РОБОТА
Пояснювальна записка

рівень вищої освіти перший (бакалаврський)

Розробка вебдодатку для генерації персоналізованих навчальних завдань з програмування на основі штучного інтелекту
(тема)

Виконав:
здобувач четвертого року навчання,
групи ІТШІ-21-3

Олександр Гладкий
(власне ім'я, прізвище)

Спеціальність 122 Комп'ютерні науки

(код і повна назва спеціальності)
Тип програми освітньо-професійна
Освітня програма Штучний інтелект

(повна назва освітньої програми)
Керівник доц. Олег Золотухін
(посада, власне ім'я, прізвище)

Допускається до захисту

Завідувач кафедри ШІ _____ Олег ЗОЛОТУХІН
(підпис) (власне ім'я, прізвище)

2025 р.

Харківський національний університет радіоелектроніки

Факультет Комп'ютерних наук

Кафедра Штучного інтелекту

Рівень вищої освіти перший (бакалаврський)

Спеціальність 122 Комп'ютерні науки
(код і повна назва)

Тип програми освітньо-професійна

Освітня програма Штучний інтелект
(повна назва)

ЗАТВЕРДЖУЮ:
Зав. кафедри _____
(підпис)
«_____» _____ 20 ___ р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

здобувачеві Гладкому Олександру Вадимовичу
(прізвище, ім'я, по батькові)

1. Тема роботи Розробка вебдодатку для генерації персоналізованих навчальних завдань з програмування на основі штучного інтелекту

затверджена наказом університету від 19 травня 20 25 р. № 378Ст

2. Термін подання студентом роботи до екзаменаційної комісії 19 червня 20 25 р.

3. Вихідні дані до роботи офіційні документації до React.js, TypeScript, Mantine UI, NestJS, Firebase, OpenAI API, Auth0 та Monaco Editor.

4. Перелік питань, що потрібно опрацювати в роботі

1) Аналіз предметної галузі та постановка задачі

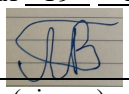
2) Проектування системи

3) Програмна реалізація

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів роботи	Строк / терміни виконання етапів роботи	Примітка
1	Отримання завдання на кваліфікаційну роботу	19.05.2025	виконано
2	Визначення проблеми	19.05.2025	виконано
3	Аналіз існуючих аналогів	20.05.2025	виконано
4	Розробка компонентів системи	21.05.2025	виконано
5	Реалізація вебдодатку	22.05.2025	виконано
6	Оформлення пояснювальної записки	01.06.2025	виконано
7	Захист	19.06.2025	виконано

Дата видачі завдання 19 травня 2025 р.

Здобувач  _____
(підпис)

Керівник роботи _____ доцент Олег Золотухін
(підпис) (посада, власне ім’я, прізвище)

РЕФЕРАТ

Пояснювальна записка: 90 с., 24 рис., 1 дод., 20 джерел.

ГЕНЕРАЦІЯ ЗАДАЧ, ОСВІТНІ ВЕБДОДАТКИ, ПРОГРАМУВАННЯ, ПЕРЕВІРКА РОЗВ'ЯЗКІВ, GPT-4, INTELLIGENT TUTORING SYSTEMS, NESTJS, OPENAI API, REACT.

Об'єктом дослідження є інтелектуальні вебдодатки, що генерують та перевіряють задачі з програмування.

Предметом дослідження виступають алгоритмічні та програмні засоби побудови системи автоматичної генерації задач і перевірки рішень із використанням моделей штучного інтелекту.

Метою роботи є розробка вебдодатку, який забезпечує персоналізовану генерацію задач з програмування та оцінювання розв'язків на основі моделі GPT.

У процесі дослідження були застосовані методи аналізу предметної області, проектування клієнт-серверної архітектури, використання API OpenAI, Firebase та Auth0, а також методи розробки з використанням стеку React та NestJS.

У результаті створено функціональний вебдодаток, здатний формувати задачі, приймати розв'язки, проводити перевірку з поясненням помилок, що забезпечує новизну через інтеграцію API генеративної моделі GPT-4 у навчальний процес.

Результати можуть бути використані для підтримки викладання програмування у форматі онлайн-платформи.

ABSTRACT

Bachelor’s thesis contains: 90 pp., 24 fig., 1 ann., 20 references.

EDUCATIONAL WEB APPLICATIONS, GPT-4, INTELLIGENT TUTORING SYSTEMS, NESTJS, OPENAI API, PROGRAMMING, REACT, SOLUTION VALIDATION, TASK GENERATION.

The object of the research is intelligent web applications that generate and evaluate programming problems.

The subject of the research includes algorithmic and software tools for building a system of automatic problem generation and solution verification using artificial intelligence models.

The purpose of the work is to develop a web application that provides personalized programming problem generation and solution assessment based on the GPT model.

During the research, methods of domain analysis, client-server architecture design, use of OpenAI, Firebase, and Auth0 APIs, as well as development techniques using the React and NestJS stack, were applied.

As a result, a functional web application was created that can generate tasks, accept user-submitted solutions, and evaluate them with detailed error explanations, which introduces novelty through the integration of the GPT-4 generative model API into the educational process.

The results can be applied to support programming education in the format of an online learning platform.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

ШІ – штучний інтелект;

API – Application Programming Interface – прикладний програмний інтерфейс;

GPT – Generative Pre-trained Transformer – генеративний попередньо натренований трансформер;

HTTP – Hypertext Transfer Protocol – протокол передачі гіпертексту;

IT – Information Technology – інформаційні технології;

JWT – JSON Web Token – вебтокен у форматі JSON;

MVP – Minimum Viable Product – мінімально життєздатний продукт;

SPA – Single Page Application – односторінковий застосунок;

SQL – Structured Query Language – мова структурованих запитів;

UI – User Interface – інтерфейс користувача;

UX – User Experience – користувацький досвід.

ВСТУП

У сучасних умовах цифрової трансформації освіти зростає потреба у створенні інноваційних інструментів для самостійного опанування програмування. Швидкий розвиток ІТ-галузі зумовлює постійне збільшення попиту на фахівців, які не лише володіють теоретичними знаннями, а й здатні оперативно розв’язувати практичні задачі. У відповідь на ці виклики активно розвивається сегмент онлайн-платформ, що пропонують навчання через задачі, інтерактивні середовища для написання коду, аналіз помилок і побудову персоналізованої траєкторії розвитку. Проте більшість таких сервісів мають обмежену адаптивність і використовують статичні бази завдань, що не враховують індивідуальні особливості користувачів.

Популярні платформи пропонують значну кількість готових задач, систему рейтингів, автоматичну перевірку рішень та базові елементи гейміфікації. Проте ці рішення здебільшого орієнтовані на широке коло користувачів із фіксованими освітніми маршрутами. Вони не забезпечують гнучкого підходу до побудови контенту в реальному часі та не надають повноцінного інтелектуального зворотного зв’язку. Завдання формуються вручну, мають стандартні формулювання й не адаптуються під потреби конкретного користувача. Унаслідок цього втрачається ефективність навчального процесу, знижується мотивація та уповільнюється прогрес.

Одним із перспективних напрямів подолання цих обмежень є використання штучного інтелекту, зокрема великих мовних моделей, для генерації навчального контенту та аналізу відповідей. Інтеграція ШІ у структуру освітнього продукту дає змогу створювати персоналізовані завдання залежно від рівня складності, тематики, мови програмування, а також забезпечувати гнучкий зворотний зв’язок у режимі реального часу. Такий підхід відкриває нові можливості в побудові динамічних освітніх платформ, які здатні адаптуватися до потреб і темпу кожного окремого користувача. Генерація задач за запитом, пояснення помилок, рекомендації

наступних тем для вивчення – усе це формує якісно нову парадигму самостійного навчання, яка не обмежується статичними модулями або фіксованим порядком матеріалу.

У межах цієї кваліфікаційної роботи розглядається проект створення вебдодатку, який дозволить користувачам самостійно вивчати програмування через персоналізовані завдання. Основною ідеєю є розробка системи, що поєднує гнучкий інтерфейс із можливістю генерації задач та перевірки рішень за допомогою ШІ. Очікується, що така система буде особливо корисною для початківців, студентів, а також усіх, хто хоче покращити практичні навички у зручному форматі.

Актуальність цієї розробки визначається як загальною тенденцією до персоналізації освіти, так і високою динамікою розвитку інструментів штучного інтелекту. Більшість існуючих рішень надають набір готових задач без можливості масштабування контенту під кожного користувача, тоді як мовні моделі відкривають потенціал до генерації не просто текстів, а повноцінних освітніх елементів, з урахуванням контексту й рівня користувача. Реалізація подібної системи дозволить зробити навчання гнучким, інклюзивним і дійсно адаптивним.

Ціллю роботи є розробка прототипу вебдодатку для генерації та перевірки навчальних завдань з програмування, який дозволить автоматизувати створення індивідуального контенту для кожного користувача. Запропоноване рішення має охоплювати як логіку формування завдань із використанням ШІ, так і інтерактивне середовище для введення, надсилання та аналізу рішень. Очікується, що така система сприятиме підвищенню ефективності самостійного навчання, зменшенню порогу входу для новачків та оптимізації освітнього процесу в онлайн-середовищі. Потенційними сферами застосування є самостійна підготовка до співбесід, супровід університетських курсів програмування, створення адаптивних онлайн-курсів і внутрішнє навчання в IT-компаніях.

1 АНАЛІЗ ПРЕДМЕТНОЇ ГАЛУЗІ ТА ПОСТАНОВКА ЗАДАЧІ

1.1 Огляд стану предметної області

Сучасне навчання програмуванню стрімко змінюється під впливом цифрових технологій, що дозволяють створювати ефективні, адаптивні та масштабовані освітні рішення. Традиційні методики, орієнтовані на пасивне засвоєння теоретичного матеріалу, поступаються місцем інтерактивним платформам, які поєднують практичні завдання, автоматизований зворотний зв’язок і персоналізовані траєкторії навчання. В умовах зростаючого попиту на фахівців із прикладними навичками дедалі важливішими стають інструменти, що дозволяють вивчати програмування в дії – через задачі, симуляції, тестування та аналіз помилок.

Широке використання інтернету, відкритих API та хмарних обчислень сприяє розвитку інтелектуальних систем, здатних генерувати навчальний контент у реальному часі. Водночас зростає роль штучного інтелекту як інструмента автоматизації створення та перевірки завдань, що відкриває нові горизонти в персоналізованому навчанні. Середовище, в якому поєднуються гнучкість користувача, адаптивність контенту та технологічна ефективність, формує нову парадигму вивчення програмування, яка буде докладно розглянута в наступних підрозділах.

1.1.1 Сучасні підходи до навчання програмування

Упродовж останнього десятиліття підходи до навчання програмуванню зазнали істотних змін. Якщо раніше переважали академічні курси з акцентом на теоретичні знання, то сьогодні в центрі уваги перебуває прикладна складова. Це зумовлено зростанням вимог ринку праці до практичних навичок та здатності вирішувати реальні задачі. Формальний

підхід, побудований на лекціях і читанні, все більше доповнюється або замінюється активними методами навчання.

Сучасні навчальні стратегії орієнтовані на принцип «навчання через дію». Такий підхід підтверджується численними дослідженнями в галузі когнітивних наук, де показано, що засвоєння матеріалу істотно покращується, коли студенти активно взаємодіють із навчальним середовищем. В контексті програмування це означає регулярне написання коду, роботу над задачами, отримання зворотного зв’язку та поступове ускладнення контенту.

Особливу популярність отримали так звані задачні платформи, які дають змогу застосовувати знання в практичному контексті відразу після їх отримання. Такий підхід не лише сприяє глибшому розумінню алгоритмів, структур даних і принципів програмування, але й розвиває навички самостійного мислення та пошуку рішень. Крім того, регулярне вирішення задач допомагає закріпити синтаксис мови програмування та сформувати інтуїцію щодо її особливостей.

Водночас зростає інтерес до гібридних методик, які поєднують відеолекції, інтерактивні вправи, симуляції та елементи гейміфікації. Така багаторівнева структура навчання дозволяє адаптувати процес до потреб конкретного користувача, враховуючи його стиль мислення, рівень підготовки та темп роботи. Усе це свідчить про перехід від лінійного пасивного навчання до гнучких, динамічних систем, що спрямовані на глибоке засвоєння матеріалу через активну практику.

1.1.2 Завдання як основа практичного оволодіння програмуванням

У процесі вивчення програмування вирішення практичних задач є не лише доповненням до теоретичного матеріалу, а й одним із ключових інструментів формування реальних навичок. Практика дозволяє не просто повторювати матеріал, а активно засвоювати його через створення робочого

коду, пошук помилок та оптимізацію рішень. Такий підхід формує глибше розуміння механіки роботи мови програмування та її застосування в різних контекстах.

Емпірично підтверджено, що найбільш ефективні методи навчання пов’язані саме з активною діяльністю. Далі можна побачити узагальнене представлення ефективності різних освітніх форматів (рисунок 1.1), де зазначено, що «практика через дію» сприяє засвоєнню до 75% матеріалу. Це значно перевищує ефективність пасивних методів, таких як читання (10%) чи лекції (5%).

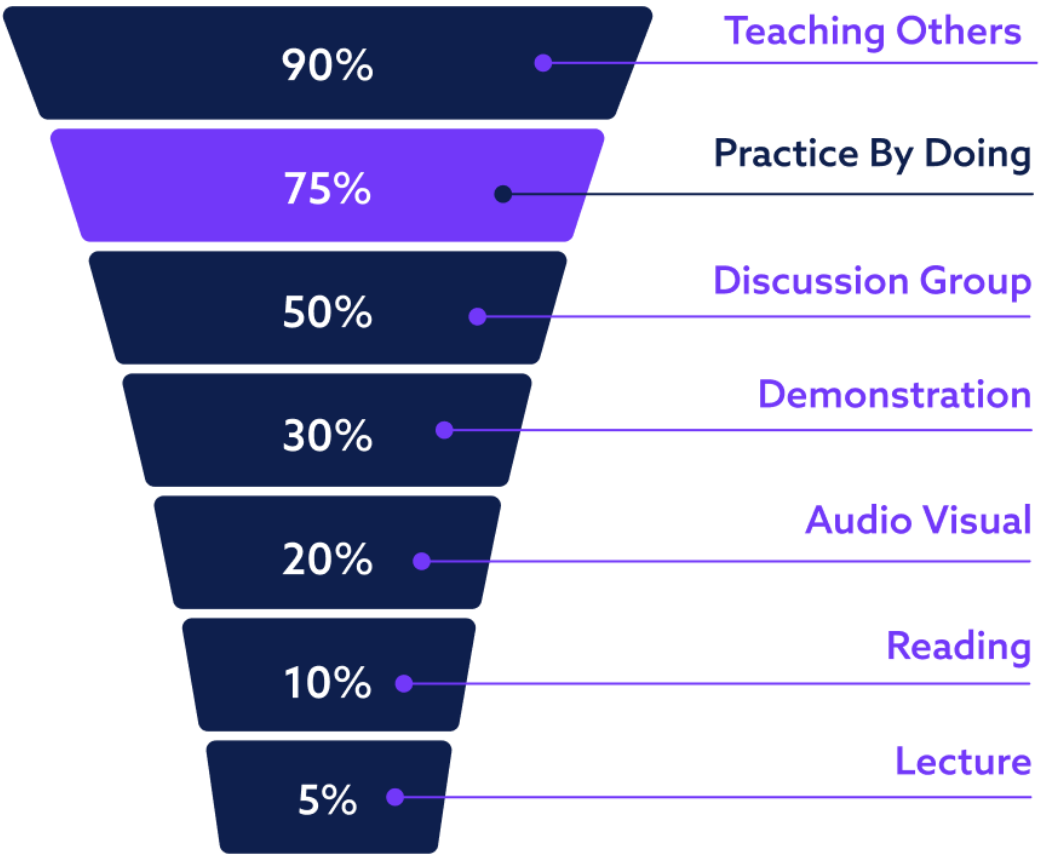


Рисунок 1.1 – Інверсна піраміда засвоєння знань у навчанні програмуванню

Рішення задач є універсальним методом, що охоплює широкий спектр когнітивних навичок: від запам’ятовування і розпізнавання структур до аналізу й синтезу рішень. Така активність змушує мислити, будувати алгоритми, приймати рішення в умовах обмежень. Це не лише прискорює процес навчання, але й сприяє збереженню знань у довготривалій пам’яті.

Крім того, завдання мають ще одну важливу властивість – їх можна адаптувати за складністю, темою та форматом. Це дозволяє реалізувати принципи індивідуалізованого навчання навіть у масштабованих онлайн-системах. Саме тому рішення задач стало невіддільною складовою більшості сучасних освітніх платформ з програмування.

1.1.3 Інтерактивні платформи практики кодування

Інтерактивні платформи стали важливою складовою сучасної IT-освіти, оскільки забезпечують користувачам можливість не просто споживати інформацію, а безпосередньо взаємодіяти з нею. Завдяки поєднанню технічної інфраструктури, гейміфікації та елементів соціального навчання такі платформи формують динамічне середовище, в якому процес оволодіння програмуванням стає більш захоплюючим і ефективним.

Сильними сторонами таких систем є чітке формулювання цілей, автоматизована оцінка прогресу, обмін знаннями між користувачами та підтримка командної взаємодії. Далі можна побачити приклади основних принципів, які реалізуються в подібних середовищах (рисунок 1.2), включаючи постановку цілей, нагородження, гейміфікацію та здорову конкуренцію. Ці елементи стимулюють користувача залишатися активним у процесі навчання та рухатися вперед. Інтерактивність платформ проявляється не лише у формі візуального або структурного оформлення завдань, а й у реалізації логіки їх адаптації до результатів користувача.

Рисунок 1.2 – Елементи інтерактивності у навчання

Зазвичай платформи враховують точність рішень, середній час виконання задач, складність обраного рівня і можуть на цій основі пропонувати індивідуальні траєкторії навчання. Така адаптивність є ключем до високої мотивації й ефективності.

Ще однією важливою особливістю є підтримка соціальних механізмів: рейтинги, змагання, обговорення, спільне вирішення задач і обмін досвідом. Це дозволяє формувати навчальні спільноти, у межах яких відбувається не лише засвоєння технічних знань, але й розвиток комунікаційних і командних навичок. Усе це сприяє перетворенню індивідуального досвіду програмування на багатовимірну освітню діяльність.

1.1.4 Поточні підходи до створення завдань для самостійної практики кодування

Сучасні підходи до створення завдань для самостійного навчання програмуванню базуються на принципах поступового ускладнення, актуальності контенту та різноманітності тематик. Більшість платформ використовують велику базу попередньо підготовлених задач, які класифікуються за рівнем складності, мовою програмування та

тематичними категоріями. Такий підхід дозволяє користувачу поступово переходити від простих прикладів до складніших завдань, водночас формуючи стабільне розуміння базових концепцій.

Частина рішень реалізує напів автоматизовану генерацію варіацій задач на основі шаблонів, змінюючи вихідні умови, імена змінних або варіанти вхідних даних. Проте ці завдання, як правило, мають фіксовану логіку та не адаптуються під реальні помилки або стиль мислення конкретного користувача. Бракує гнучких механізмів, що дозволяють створювати задачі в режимі реального часу з урахуванням попередніх спроб, частоти помилок або тематичних уподобань, що обмежує можливості персоналізованого навчання.

1.1.5 Штучний інтелект як новий рушій в навчальних середовищах

Штучний інтелект поступово перетворюється на ключовий елемент освітніх систем нового покоління. Його застосування дозволяє автоматизувати процес створення навчального контенту, адаптувати завдання до рівня підготовки конкретного користувача та надавати персоналізований зворотний зв’язок. Завдяки інтеграції мовних моделей, таких як GPT, стало можливим генерувати унікальні формулювання задач, пояснення рішень та підказки в режимі реального часу.

Використання ШІ також відкриває перспективи щодо побудови динамічних траєкторій навчання, де зміст завдань формується на основі поведінки користувача, його прогресу та типових помилок. Це дозволяє не просто навчати, а й активно підтримувати процес пізнання, враховуючи індивідуальні особливості студента. У результаті формується нова парадигма навчання, у центрі якої – інтелектуальний супровід користувача впродовж усього освітнього шляху.

1.2 Аналіз існуючих платформ для навчання програмуванню

У сфері онлайн-навчання програмуванню вже сформувалося стійке конкурентне середовище, представлене великою кількістю платформ, що пропонують різні формати взаємодії з навчальним контентом. Ці сервіси орієнтовані як на початківців, так і на досвідчених розробників, пропонуючи завдання різного рівня складності, тематичні модулі, систему оцінювання та прогресу. Рішення задач є основним механізмом навчання на таких платформах, що підтверджує актуальність задачного підходу як провідної освітньої моделі.

Більшість платформ реалізують однакові базові функції: поділ задач за рівнями, підтримку кількох мов програмування, інтерактивне середовище написання коду, автоматичну перевірку та зберігання статистики. Водночас існують відмінності у підходах до персоналізації, механіці мотивації користувача, можливості спільної роботи та використанні додаткових інструментів аналізу. Кожна система має свої сильні та слабкі сторони, які визначають її цільову аудиторію та ефективність у навчальному процесі.

Одним із ключових викликів, що постають перед більшістю існуючих платформ, є обмежена адаптивність задач і нездатність гнучко реагувати на індивідуальні потреби користувача. Контент зазвичай створений вручну, тому його масштабування вимагає значних ресурсів. Це обмежує можливість створення унікальних завдань у реальному часі та звужує потенціал персоналізованого навчання.

У зв’язку з цим зростає інтерес до нових рішень, які поєднують класичні підходи з технологіями штучного інтелекту. Такі системи можуть не лише імітувати функціональність існуючих платформ, а й виходити за їх межі, забезпечуючи індивідуалізований досвід навчання. Подальші підрозділи розглянуть найбільш популярні сервіси, їхні особливості, а також виявлять наявні обмеження, що мотивують розробку нових підходів.

1.2.1 Огляд провідних платформ-конкурентів

Ринок освітніх платформ для вивчення програмування представлений великою кількістю сервісів, що конкурують за увагу користувача за рахунок зручності, якості задач, системи мотивації та додаткових можливостей. Кожна з них має свою цільову аудиторію та реалізує власну освітню філософію. У цьому підрозділі розглядаються чотири найпопулярніші рішення, які стали еталоном у сфері задачного навчання: LeetCode, Codewars, HackerRank та Exercism. Вони не лише охоплюють величезну кількість задач, але й формують спільноти навколо програмування.

LeetCode (рисунок 1.3) орієнтований передусім на підготовку до технічних співбесід та алгоритмічного мислення. Користувачам доступні сотні задач із категоризацією за темами, складністю та мовами програмування. Окрім самих завдань, система пропонує рейтинги, аналітику активності та історію рішень.

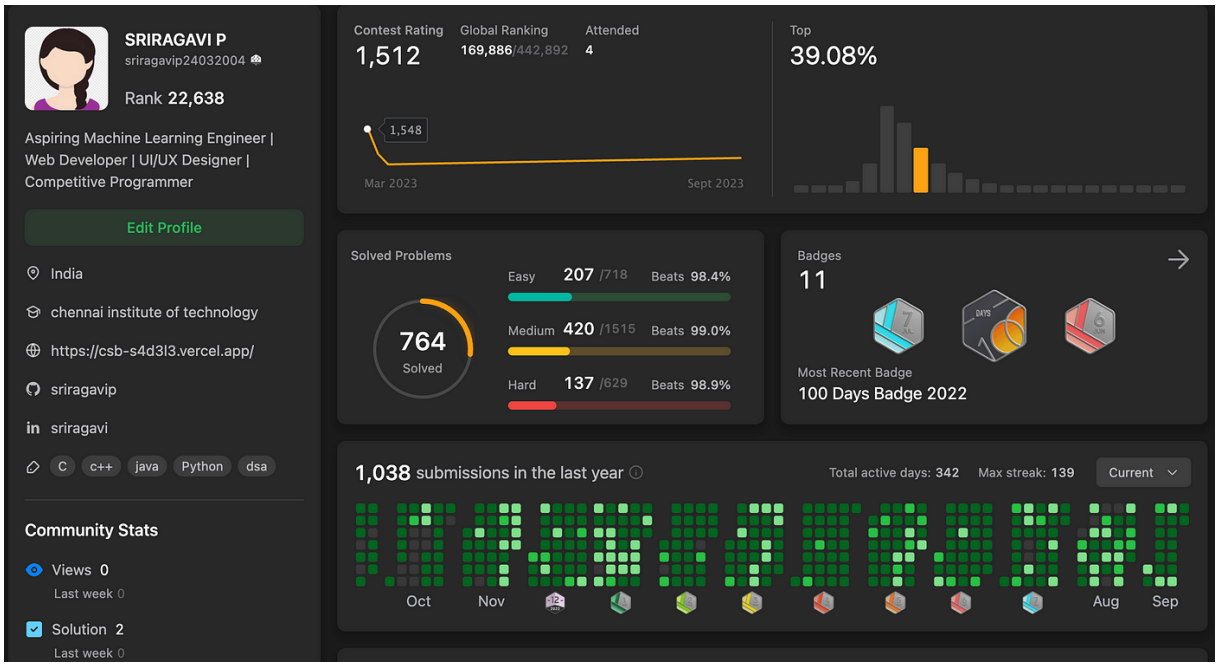


Рисунок 1.3 – Інтерфейс LeetCode

Далі можна побачити приклад профілю користувача на LeetCode, який демонструє прогрес, статистику розв'язків, розподіл складності та активність у часі. Такий інтерфейс дозволяє відстежувати динаміку власного розвитку та встановлювати цілі, що робить навчання більш структурованим.

Codewars, на відміну від LeetCode, більше зосереджується на гейміфікації процесу навчання. Кожне завдання подається як «ката» з бойових мистецтв, що підвищує мотивацію і втягнутість користувача. Платформа підтримує багато мов програмування, а також надає можливість створювати власні завдання для інших учасників.

Далі можна побачити типову структуру задачі на Codewars: формулювання, приклади, поле для написання рішення та модульні тести, які автоматично перевіряють код (рисунок 1.4). Такий підхід стимулює користувача не тільки вирішувати задачі, а й розуміти логіку перевірки рішень.

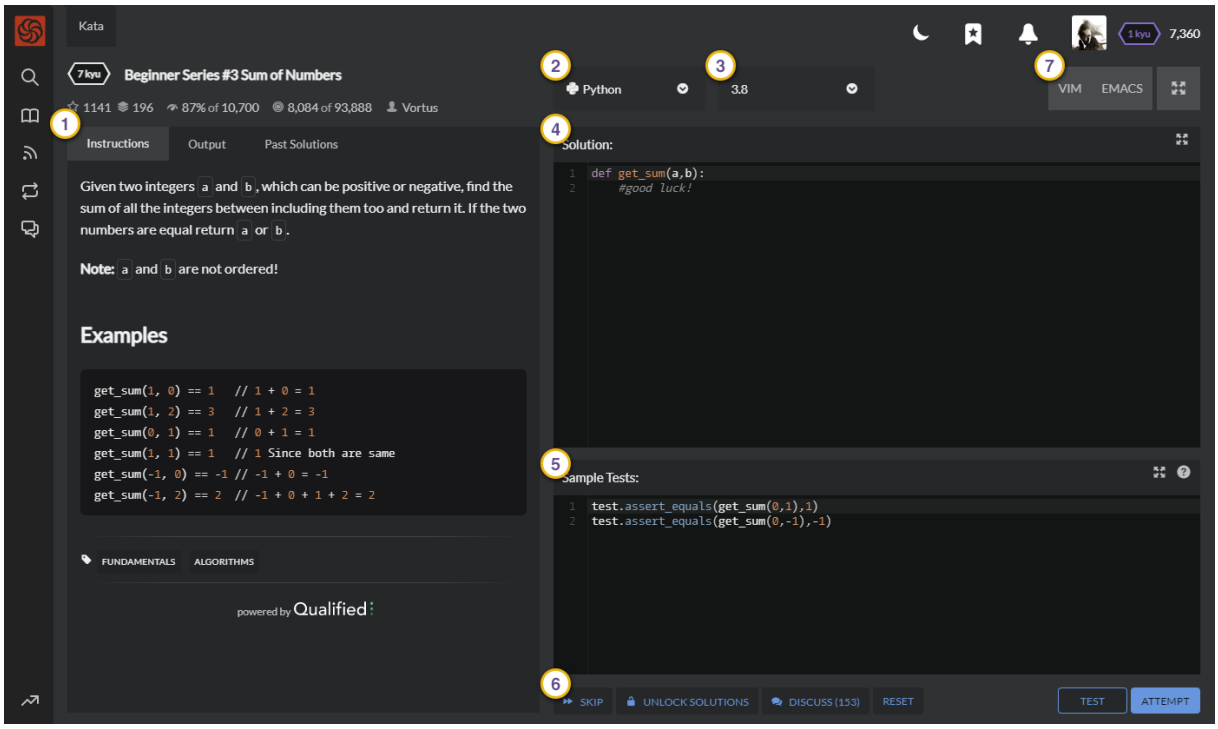


Рисунок 1.4 – Інтерфейс Codewars

HackerRank поєднує задачі на кодування з симуляцією реального технічного інтерв'ю. Окрім класичних алгоритмічних завдань, тут доступні блоки з SQL, безпекою, DevOps, штучним інтелектом та навіть підготовкою до сертифікацій. Особливістю HackerRank є орієнтація на співпрацю з компаніями: платформа пропонує фахівцям проходити випробування, які потім можуть бути використані для працевлаштування. Це робить її привабливою не лише для навчання, а й для побудови кар'єри.

Exercism вирізняється своїм підходом до навчання через менторство. Платформа підтримує понад 50 мов програмування, але замість автоматичної перевірки рішень пропонує спілкування з реальними менторами. Завдяки цьому користувач отримує більш глибокий фідбек та пояснення помилок. Exercism орієнтований на поступове навчання, де кожен розв'язок коментується, що дозволяє краще зрозуміти концепції, а не лише знайти правильний результат.

Кожна з розглянутих платформ демонструє власне бачення навчального процесу, різною мірою реалізуючи адаптивність, гейміфікацію, соціальну взаємодію та підтримку з боку спільноти. Такий різноманітний підхід свідчить про високий інтерес до інструментів, що дозволяють ефективно вивчати програмування через практику. Однак усі ці сервіси мають і спільне обмеження – обмежену гнучкість у персоналізації контенту, що відкриває простір для нових рішень із залученням штучного інтелекту.

1.2.2 Порівняльний аналіз функціональності конкурентів

Ключовою характеристикою, яка відрізняє сучасні задачні платформи одна від одної, є функціональність, що охоплює як технічні можливості системи, так і методику організації навчання. LeetCode зосереджений на алгоритмічних задачах з акцентом на підготовку до співбесід, тому надає велику кількість задач, ранжованих за темами та складністю. Крім того, платформа підтримує інтерактивне середовище кодування, детальну

аналітику, систему досягнень і рейтинги, що дозволяє користувачеві бачити прогрес у реальному часі. Проте навчання тут здебільшого є індивідуальним і не передбачає активної взаємодії зі спільнотою.

Codewars, навпаки, робить ставку на спільноту та гейміфікацію. Завдання тут мають форму ката, які проходять через перевірку іншими користувачами. Після успішного розв’язання задачі користувач отримує доступ до рішень інших учасників, що дозволяє аналізувати альтернативні підходи. Водночас Codewars обмежено реалізує структуроване навчання: немає чіткого поділу за темами або навчальних треків. Інтерфейс сприяє самостійному дослідженню контенту, але може бути складним для новачків без супроводу.

HackerRank вирізняється гнучкістю та широтою охоплення. Окрім задач із програмування, платформа включає модулі з баз даних, системного адміністрування, логіки, машинного навчання. Також вона підтримує автоматичне тестування, завантаження файлів, таймери та інтерфейси, які імітують умови технічного інтерв’ю. Важливою функцією є можливість компаній створювати власні тести для кандидатів, що робить HackerRank платформою з елементами рекрутингу, а не лише навчання.

Exercism має інший підхід до функціональності: вона обмежена в автоматизації, але компенсує це через систему менторства. Кожне рішення переглядається людиною, що дозволяє уникнути поверхневого засвоєння матеріалу. Програма навчання структурована у вигляді треків для кожної мови, але темп просування залежить від активності користувача та наявності менторів. Така модель підходить для глибокого, але повільного навчання, і є менш придатною для масштабування в системах.

1.2.3 Виявлення недоліків у підходах до персоналізації завдань

Попри високу якість задач і багатий функціонал, більшість платформ не забезпечують повноцінної персоналізації навчального процесу.

Структура подачі матеріалу зазвичай фіксована, а набір доступних задач однаковий для всіх користувачів. Навіть за наявності фільтрів за темами та складністю система не враховує рівень підготовки конкретного студента, його типові помилки або індивідуальні прогалини в знаннях. Це призводить до ситуацій, коли користувач змушений самостійно шукати відповідний рівень складності, що може знижувати ефективність навчання.

Іншою проблемою є статичність контенту. Завдання зазвичай створені вручну, з фіксованими умовами та прикладами. Це обмежує кількість варіацій задач і не дозволяє повторно використовувати їх без ризику втрати мотивації або ефекту запам’ятовування. Автоматичне генерування варіантів задач реалізоване лише частково, без глибокої адаптації до конкретного користувача. Водночас більшість платформ не мають системи інтелектуального аналізу результатів, яка могла б формувати рекомендації або адаптувати навчальний маршрут.

Також обмежено реалізовані можливості зворотного зв’язку. У більшості випадків платформи просто повідомляють про помилку або неправильний результат, не пояснюючи причину. Користувач залишається наодинці з невдалою спробою, що ускладнює процес навчання, особливо для початківців. Відсутність пояснень, варіативних підказок або коментарів від системи зменшує ефективність задачного підходу в тих випадках, коли користувач не має достатньої підтримки або досвіду.

1.3 Огляд потреб користувачів

Успішність навчального програмного продукту значною мірою залежить від того, наскільки точно враховано потреби його користувачів. У сфері задачного навчання програмуванню користувачі очікують не лише якісного контенту, а й зручного інтерфейсу, зрозумілої логіки подачі матеріалу та можливості гнучко керувати власною траєкторією навчання.

Зростання популярності самостійного онлайн-навчання формує високі вимоги до персоналізації, адаптивності та інтерактивності платформи.

Користувачі потребують систем, які допомагають формулювати цілі, відслідковувати прогрес і надають миттєвий зворотний зв’язок. Зокрема, початківці шукають підказки та пояснення, а досвідчені користувачі очікують складніших задач і детальної аналітики власної діяльності. Таким чином, платформа має адаптуватися до рівня знань, стилю навчання і темпу роботи кожного користувача, підтримуючи водночас мотивацію до регулярної практики.

Крім того, важливим чинником є простота взаємодії з системою. Складні інтерфейси або перевантажені елементи можуть відштовхнути частину користувачів, тоді як інтуїтивно зрозуміле середовище сприяє кращому засвоєнню матеріалу. Усе це вимагає ретельного аналізу очікувань цільової аудиторії та подальшого врахування їх у дизайні архітектури, функціоналу та користувацького інтерфейсу платформи.

1.3.1 Цільова аудиторія

Основною цільовою аудиторією задачних навчальних платформ є студенти технічних спеціальностей, самоучки, а також молоді спеціалісти, які готуються до технічних співбесід. Ці користувачі зазвичай мають базові навички програмування або перебувають у процесі їх формування, тому особливо потребують доступного, структурованого і поступового подання навчального матеріалу. Важливим для них є наявність різнорівневих завдань, можливість обрати мову програмування та отримати фідбек щодо виконаних рішень.

До цієї ж аудиторії належать користувачі, які прагнуть підтримувати рівень навичок або готуються до конкретних сертифікацій, хакатонів або вступу на позиції в IT-компанії. Їхня зацікавленість спрямована не лише на самі задачі, а й на систему відстеження прогресу, аналітику, рейтинги та

конкуренцію з іншими учасниками. Такі користувачі шукають платформу, яка дозволяє практикуватися регулярно, мати виклики відповідного рівня складності та отримувати підтвердження своїх навичок.

Окрему групу складають ті, хто повертається до програмування після тривалої перерви або переходить з іншої сфери. Для них критично важливим є зручність інтерфейсу, наявність підказок, мотиваційних механізмів і поступовий вступ до навчального процесу. Така аудиторія потребує підтримки в адаптації, тому особливу роль відіграють пояснення, приклади, зворотний зв’язок і відсутність високого порогу входу в систему.

1.3.2 Очікування від навчального вебдодатку

Батьки та викладачі, які супроводжують процес навчання, мають свої окремі очікування щодо функціональності освітнього продукту. Для них важливо, щоб платформа була не лише корисною для самостійного навчання, а й прозорою у плані відстеження результатів. Наявність аналітики, оцінок, рівнів складності та історії виконаних завдань дозволяє контролювати прогрес учня, виявляти слабкі місця та формувати індивідуальні рекомендації.

Також цю категорію користувачів цікавить доступність контенту та його якість. Бажано, щоб завдання не тільки перевіряли знання синтаксису, а й сприяли розвитку логіки, алгоритмічного мислення та креативності. Для викладача важливим є наявність чітко структурованої програми, яку можна інтегрувати в навчальний процес, а також можливість налаштування рівня складності відповідно до аудиторії. Для батьків – важливо, щоб інтерфейс був зрозумілим і мотивував дитину до самостійного навчання.

Ще одним очікуванням є безпечність і доступність системи з різних пристроїв, а також відсутність зайвих відволікаючих елементів. Платформа має бути зосередженою на навчальному процесі, підтримувати зосередженість учня та пропонувати прогресивне навчання без

перевантаження. Усі ці фактори роблять участь батьків і викладачів важливим джерелом зворотного зв’язку при розробці ефективного освітнього середовища.

1.3.3 Проблеми при традиційному навчанні програмуванню

Однією з поширених проблем при традиційному навчанні програмуванню є обмежена практична складова. Часто курси або підручники акцентують увагу на теорії, не забезпечуючи достатньої кількості завдань для відпрацювання навичок. У результаті користувачі знають синтаксис, але не можуть застосовувати знання для вирішення реальних задач. Такий підхід призводить до поверхневого засвоєння матеріалу та втрати мотивації при спробах самостійно розв’язувати практичні проблеми.

Ще однією проблемою є відсутність гнучкості у темпах навчання. Традиційні форми не враховують індивідуальних особливостей користувача, таких як рівень підготовки, швидкість засвоєння або інтерес до певних тем. Це створює відчуття надмірного навантаження або, навпаки, нудьги, що знижує ефективність навчання. Крім того, брак миттєвого зворотного зв’язку ускладнює самостійну роботу над помилками та формування глибокого розуміння матеріалу.

1.3.4 Необхідність аналітики та простоти використання

У сучасному навчанні важливу роль відіграє аналітика, яка дозволяє користувачеві розуміти свій прогрес, виявляти сильні й слабкі сторони та планувати подальший маршрут. Відсутність візуалізованої статистики або хоча б базової інформації про кількість вирішених задач, рівень складності, час виконання та типові помилки знижує ефективність навчального

процесу. Користувач потребує прозорої системи оцінювання, яка не лише фіксує результат, а й пояснює його.

Простота інтерфейсу також є критично важливою, особливо для новачків. Ускладнений або перевантажений функціоналом інтерфейс може відштовхнути користувача ще на початковому етапі. Інтуїтивна взаємодія з платформою, зручна навігація, логічне розміщення елементів та швидкий доступ до потрібних функцій формують позитивний досвід і сприяють більшій залученості. Це створює передумови для формування сталої навчальної звички та підвищення рівня мотивації.

1.4 Визначення ключових функцій та можливостей

Функціональність сучасного вебдодатку для навчання програмуванню має відповідати не лише базовим очікуванням користувачів, але й вимогам до гнучкості, адаптивності та інтерактивності. У центрі архітектури такого продукту мають бути механізми генерації задач, обробки рішень у реальному часі, збору статистики та зручної авторизації. Усі ці компоненти повинні працювати як єдине цілісне середовище, що підтримує навчальний процес і забезпечує позитивний користувацький досвід.

Важливою перевагою платформи є її здатність адаптувати контент до потреб конкретного користувача, включаючи рівень складності, тему задачі та мову програмування. Також необхідно враховувати технологічний стек, який забезпечить стабільну роботу, масштабованість та інтеграцію із зовнішніми сервісами, зокрема API для генерації задач та перевірки рішень.

1.4.1 Технологічний стек

Під час проектування вебдодатку для генерації персоналізованих навчальних завдань з програмування планується використання перевірених

сучасних технологій, які забезпечують гнучкість розробки, масштабованість системи та можливість інтеграції з зовнішніми сервісами. Вибір кожного інструменту обґрунтовується його відповідністю до задач проекту, надійністю в комерційному середовищі та активною підтримкою спільноти.

На стороні клієнта (front-end) планується використання React у поєднанні з мовою TypeScript. React є однією з найпоширеніших бібліотек для побудови інтерфейсів, яка дозволяє створювати компоненти з високою швидкістю взаємодії. Використання TypeScript забезпечує статичну типізацію, що допомагає уникати помилок на ранніх етапах розробки та спрощує підтримку коду. Додатково для стилізації інтерфейсу буде використано бібліотеку Mantine, яка має набір адаптивних і функціональних UI-компонентів, що відповідають сучасним вимогам до дизайну.

Для організації бекенду (back-end) обрано фреймворк Nest.js, який також побудований на основі TypeScript. Його модульна архітектура дозволяє структурувати логіку додатку, полегшує інтеграцію зовнішніх сервісів та підтримує шаблони, знайомі з корпоративної розробки. Такий вибір забезпечує узгодженість між фронтендом і бекендом, що прискорює розробку та зменшує кількість помилок на стику між частинами системи.

Нарешті, ключовим компонентом додатку є інтеграція з OpenAI API, яка забезпечить генерацію завдань на основі вхідних параметрів. Завдяки використанню потужної мовної моделі можна створювати унікальні формулювання задач, адаптовані до заданої тематики, рівня складності та мови програмування. Такий підхід дозволяє вийти за межі статичних баз задач і забезпечити справжню персоналізацію навчального процесу.

Загалом, обраний стек технологій є надійним, гнучким і добре задокументованим, що створює сприятливі умови для реалізації інноваційного освітнього продукту з елементами штучного інтелекту.

1.4.2 Аутентифікація користувача та захист персональних даних

Аутентифікація користувачів буде реалізована через сервіс Auth0, який забезпечує просту інтеграцію з сучасними вебдодатками та відповідає вимогам безпеки. Auth0 підтримує багато стратегій авторизації, включаючи Google, GitHub, email-пароль та інші, що дозволяє охопити ширшу аудиторію без потреби у складній реалізації з нуля.

1.4.3 Автоматизоване створення задач

Автоматизоване створення завдань є однією з ключових функцій планованого додатку, що дозволяє формувати унікальні задачі в режимі реального часу на основі заданих параметрів. На відміну від традиційних платформ, де задачі створюються вручну і мають обмежений набір варіацій, інтеграція зі штучним інтелектом відкриває можливість генерувати нові формулювання для кожного користувача. Такий підхід забезпечує високу варіативність контенту, знижує ймовірність повторів і підвищує мотивацію до вирішення нових завдань.

Планується використання OpenAI API для генерації тексту задачі на основі введених параметрів: теми, мови програмування, рівня складності та додаткових побажань. Користувач зможе обрати ці параметри через інтерфейс, після чого система сформує задачу із чітким формулюванням, прикладами вхідних і вихідних даних, а також обмеженнями. Такий підхід дозволяє адаптувати зміст до індивідуального рівня користувача та формувати персоналізований освітній маршрут.

1.4.4 Перевірка рішень в реальному часі

У додатку перевірка рішень в реальному часі здійснюватиметься з використанням OpenAI API, що дозволяє не лише визначити правильність

відповіді, а й проаналізувати логіку виконаного коду. Після надсилання рішення користувача система формує запит до мовної моделі, яка виконує роль віртуального асистента-рев'ювера: вона перевіряє відповідність розв’язку умові задачі, порівнює очікувані та фактичні результати, а також може надати пояснення помилок або поради щодо покращення. Такий підхід забезпечує більш гнучкий і інтелектуальний зворотний зв’язок у порівнянні з класичними системами автотестування.

1.5 Постановка задачі

Метою розробки є створення інтелектуального вебдодатку, що дозволяє генерувати персоналізовані завдання з програмування на основі вибраних параметрів користувача та здійснювати їх перевірку в реальному часі. Об'єктом дослідження виступає процес адаптивного навчання програмуванню в онлайн-середовищі, а предметом – технологічні рішення для автоматизованого створення й перевірки задач із використанням штучного інтелекту. Для досягнення поставленої мети необхідно реалізувати низку задач: проаналізувати сучасні освітні платформи задачного типу та потреби користувачів; визначити архітектуру вебдодатку та вибрати оптимальний технологічний стек; розробити інтерфейс для генерації задач та подання рішень; забезпечити інтеграцію з OpenAI API для формування умов задач і перевірки відповідей; створити базову систему збереження прогресу й статистики.

Очікуваним результатом є створення функціонального прототипу вебдодатку, що дозволяє користувачеві формувати індивідуальні завдання за темою, мовою та рівнем складності, розв’язувати їх у вбудованому редакторі коду та отримувати миттєвий інтелектуальний зворотний зв’язок. Реалізація системи має перспективу масштабування та подальшого розвитку з урахуванням розширення функціоналу, впровадження навчальних треків, рейтингових систем і підтримки менторства.

2 ПРОЕКТУВАННЯ СИСТЕМИ

2.1 Огляд технічних і функціональних вимог

Успішна реалізація вебдодатку для генерації персоналізованих навчальних завдань з програмування потребує ретельного визначення технічних і функціональних вимог, що відображають як загальну логіку роботи системи, так і специфіку її окремих модулів. На цьому етапі здійснюється формалізація цілей, які має реалізовувати система, а також окреслення вимог до технологій, масштабованості, інтеграції, безпеки та зручності для кінцевого користувача. Формування цих вимог ґрунтується на результатах попереднього аналізу предметної галузі, виявлених потребах користувачів, а також огляді існуючих рішень і технологічних можливостей.

Серед основних функціональних можливостей, які повинні бути реалізовані у системі, можна виділити: можливість авторизації та реєстрації користувачів, інтерфейс для вибору параметрів задачі (тема, рівень складності, мова програмування), генерацію унікальної умови задачі за заданими параметрами з використанням OpenAI API, надання інтерактивного середовища для написання та запуску коду, а також механізм перевірки рішень з наданням зворотного зв’язку. Кожна з перелічених функцій є необхідною для реалізації повноцінного навчального процесу, що поєднує практику програмування з елементами адаптивного навчання.

З точки зору користувацької взаємодії важливим є забезпечення простоти й інтуїтивності інтерфейсу. Користувач повинен мати змогу легко орієнтуватися у додатку, швидко знаходити потрібні функції та здійснювати основні дії без додаткових пояснень або інструкцій. Зокрема, процес створення задачі має включати послідовний вибір параметрів, зручну кнопку для запуску генерації, автоматичне відображення згенерованого

завдання, а також доступ до редактора коду. Результати перевірки мають відображатися у зручному форматі, з можливістю повторного надсилання рішення або зміни вхідних параметрів задачі.

Функціональні вимоги також передбачають реалізацію модулю перевірки рішень у реальному часі. Цей модуль має отримувати код користувача, надсилати його у сформованому вигляді на сервер (або безпосередньо через OpenAI API), отримувати відповідь у форматі тексту з аналізом коду та відображати її в інтерфейсі. Бажано, щоб результат включав не лише інформацію про правильність розв’язку, а й пояснення типових помилок, коментарі або підказки щодо поліпшення коду. Це дозволить реалізувати зворотний зв’язок, наближений до реального менторства.

Крім функціональних, система повинна задовольняти низку нефункціональних вимог, які визначають її якість, надійність і здатність до подальшого масштабування. Насамперед додаток має бути доступним через браузер, не вимагати встановлення додаткового програмного забезпечення та підтримувати адаптивність до різних розмірів екранів (десктоп, планшет, смартфон). Важливо також забезпечити високу швидкодію – час відгуку після запиту користувача на генерацію задачі або перевірку коду має бути мінімальним. Система має працювати стабільно незалежно від кількості одночасних користувачів, що вимагає продуманої серверної архітектури та оптимізації запитів до зовнішніх API.

Надійність і безпека – ще один критичний аспект проекту. Для автентифікації користувачів планується використання сервісу Auth0, який дозволяє швидко інтегрувати механізми входу з різних джерел (електронна пошта, Google, GitHub) і забезпечити відповідний рівень захисту. Дані користувача, включно з історією виконаних задач, обраними параметрами та отриманими результатами, повинні зберігатися у хмарному сховищі Firestore, яке підтримує контроль доступу, масштабування та

відмовостійкість. Під час передачі даних між клієнтом і сервером мають використовуватись шифровані канали зв’язку (HTTPS).

Варто також зазначити важливість логування подій у системі. Для повноцінної підтримки, виявлення помилок та подальшого розвитку додатку необхідно фіксувати основні дії користувача, запити до зовнішніх сервісів, помилки генерації задач і результати перевірки рішень. Це дозволить здійснювати моніторинг стану додатку в реальному часі, виявляти типові проблеми користувачів і вдосконалювати механізми генерації та зворотного зв’язку.

Важливою вимогою є масштабованість системи. Оскільки додаток потенційно може використовуватись великою кількістю користувачів одночасно, його архітектура має бути готова до горизонтального масштабування. Для цього передбачається використання контейнеризації через Docker та можливість деплою у середовище Google Cloud Platform, що забезпечує автоматичне масштабування ресурсів відповідно до навантаження. Такий підхід дозволяє уникнути деградації продуктивності при зростанні кількості запитів.

Окремо варто вказати вимогу щодо розширюваності системи. Структура додатку повинна бути модульною, із чітким поділом на компоненти (інтерфейс, бізнес-логіка, API, збереження даних), що дозволить у майбутньому безболісно додавати нові функції – наприклад, створення повноцінних навчальних треків, реалізацію менторських підказок, аналіз стилю коду або рейтинг користувачів. Модульна архітектура також спрощує тестування та підтримку проекту.

У підсумку, визначення технічних і функціональних вимог на цьому етапі проектування дає змогу сформувати чітке бачення майбутньої системи, її структури, логіки роботи та способів взаємодії з користувачем і зовнішніми сервісами. Такий підхід створює основу для подальшої розробки та знижує ризики виникнення помилок на етапі впровадження.

2.2 Поведінкова модель користувача в системі

Поведінкова модель користувача в системі визначає основні шляхи взаємодії людини з інтерфейсом вебдодатку, відображаючи послідовність дій, очікувану логіку переходів між станами та реакцію системи на введення. Створення такої моделі є необхідним етапом проектування, оскільки дозволяє формалізувати ключові сценарії використання системи, виявити потенційні бар’єри у взаємодії та забезпечити інтуїтивно зрозумілий досвід для кінцевого користувача. З практичної точки зору, вона слугує основою для розробки як клієнтської, так і серверної частини додатку, дозволяючи узгодити функціональність, дизайн та логіку обробки запитів.

Поведінкове проектування базується на гіпотетичних або емпірично встановлених моделях користувацької поведінки, враховуючи типові цілі, потреби та очікування користувача. У межах цього проекту така модель була побудована на основі аналізу аналогічних систем, потреб цільової аудиторії та передбачуваної структури інтерфейсу. Було визначено, що користувач взаємодіє із системою через декілька ключових сценаріїв, що охоплюють повний цикл роботи з додатком – від аутентифікації до перегляду результатів виконаних завдань.

Задля візуалізації обраних сценаріїв доцільно використати діаграми послідовностей, які відображають хронологію повідомлень між актором, користувачем, та компонентами системи. Такий підхід дозволяє чітко показати логіку обробки запитів і залежності між модулями. У цьому підрозділі будуть детально описані три основні сценарії: генерація нового завдання, виконання згенерованого завдання та перегляд статистики користувача. Крім того, розглянуто узагальнену сесію, яка демонструє типову поведінку користувача під час повноцінного використання вебдодатку.

Сценарій генерації нового завдання (рисунок 2.1) є першим і найважливішим кроком у процесі взаємодії користувача з навчальним вебдодатком. Він починається з ініціативи користувача, який обирає бажані параметри для майбутнього завдання, зокрема тему, мову програмування та рівень складності. Ці дії здійснюються через інтерфейс користувача, який забезпечує зручний механізм введення параметрів. Далі можна побачити послідовність повідомлень, які ілюструють логіку цього процесу.

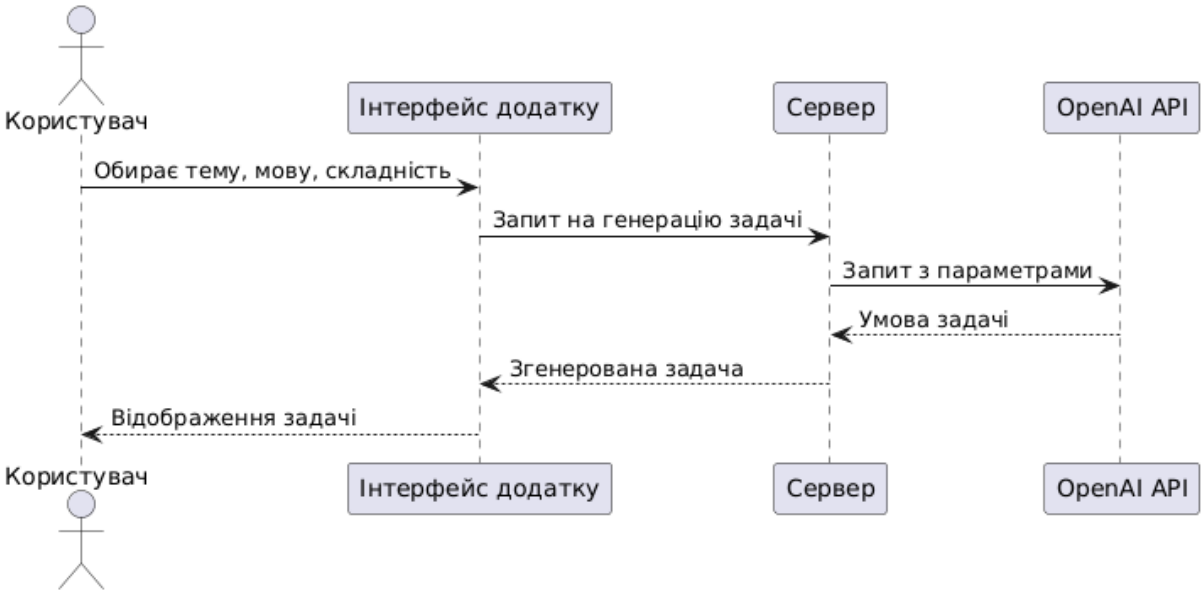


Рисунок 2.1 – Генерація нового завдання

Після задання параметрів інтерфейс надсилає запит до серверної частини системи. Сервер, обробивши отримані дані, формує запит до зовнішнього сервісу OpenAI API, який відповідає за генерацію умов задач. Сформований запит містить обрані параметри користувача у відповідному форматі. У відповідь OpenAI API надсилає сформульовану задачу у вигляді текстової інструкції, яку сервер передає назад до інтерфейсу користувача.

Отримана умова задачі відображається у веб-інтерфейсі у зручному для користувача вигляді. На цьому етапі система завершує сценарій генерації й переходить у стан очікування взаємодії з користувачем, який

може розпочати розв’язання задачі або змінити параметри генерації. Така модульна структура сценарію дозволяє легко масштабувати логіку створення задач, а також інтегрувати додаткові етапи – наприклад, фільтрацію тем або обмеження за рівнем складності.

Сценарій виконання згенерованого завдання передбачає активну взаємодію користувача з інтерактивним середовищем вебдодатку, де користувач має змогу ввести програмний код, що відповідає умовам раніше отриманої задачі. Цей процес розпочинається після перегляду згенерованого завдання й реалізується у межах вбудованого редактора коду. Далі можна побачити послідовність взаємодій, що відбуваються під час цього сценарію (рисунок 2.2).

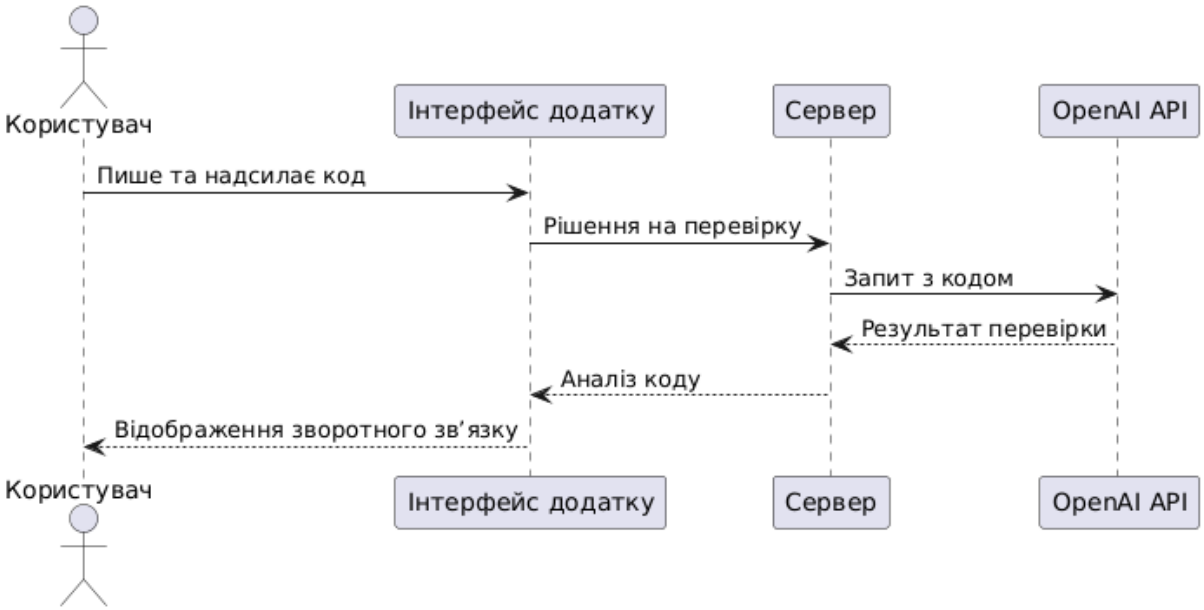


Рисунок 2.2 – Виконання згенерованого завдання

Користувач надсилає свій код на перевірку, після чого інтерфейс додатку передає відповідний запит до серверної частини системи. Сервер обробляє отриманий код, формує запит до OpenAI API, який виконує аналіз рішення. У запит включається не лише сам код, але й умова задачі та додаткові інструкції, які дозволяють моделі оцінити правильність

виконаного завдання. OpenAI API повертає текстову відповідь, яка містить результати перевірки, виявлені помилки, а також можливі поради щодо покращення рішення.

Сервер приймає відповідь від моделі та передає її назад у фронтенд. Інтерфейс відображає результат користувачу у зручному форматі, наприклад, виділяючи помилки, пояснення або зауваження щодо стилю написання коду. Такий механізм створює ефект діалогу між користувачем та цифровим наставником, що сприяє глибшому розумінню матеріалу та розвитку навичок самостійного виправлення помилок. Уся інформація про результат сесії також може зберігатися для подальшого аналізу й побудови статистики.

Сценарій перегляду статистики дозволяє користувачу отримати зведену інформацію про результати власної навчальної діяльності в системі. Цей процес починається з відкриття головної сторінки додатку, яка містить відповідний модуль аналітики. Далі можна побачити логіку взаємодії, що реалізується при зверненні до статистичних даних (рисунок 2.3).

Рисунок 2.3 – Перегляд головної сторінки зі статистикою

Інтерфейс надсилає запит до серверної частини на отримання статистики, пов’язаної з виконаними задачами. Сервер, у свою чергу, звертається до бази даних Firestore, де зберігається історія рішень, результати перевірок, кількість спроб, дати виконання та інші релевантні метадані. Отримані дані адресуються і структуруються у формат, зручний для візуалізації, після чого передаються назад до інтерфейсу додатку.

На основі наданої інформації інтерфейс формує узагальнений огляд, який може включати графіки активності, відсоток правильних рішень, розподіл складності задач, рейтинг серед користувачів тощо. Такий підхід не лише дозволяє користувачу аналізувати власний прогрес, а й стимулює до подальшої практики завдяки елементам саморефлексії та внутрішньої мотивації. Сценарій перегляду статистики також є основою для реалізації адаптивного навчання, оскільки дозволяє системі враховувати індивідуальні особливості користувача.

Звичайна сесія користувача у системі охоплює послідовність ключових дій, які охоплюють основний функціонал вебдодатку – від входу до отримання зворотного зв’язку та перегляду статистики. Цей сценарій починається з відкриття додатку, що ініціює процес аутентифікації через Auth0. Успішна авторизація призводить до отримання токена доступу, на основі якого сервер звертається до зовнішньої бази даних для отримання профілю користувача. Система повертає інформацію, яка включає початкові налаштування користувача, і інтерфейс готовий до подальшої взаємодії.

Після завантаження головної сторінки користувач має змогу сформувати нове завдання, обравши параметри, такі як тема, мова програмування та складність. Інтерфейс надсилає запит до серверної частини, яка у свою чергу звертається до OpenAI API для генерації умови задачі. Згенерована задача повертається до клієнта й відображається користувачу, після чого той може приступити до її виконання. Написаний код надсилається на сервер, де формується запит до OpenAI API з метою перевірки рішення.

Далі можна побачити логіку усіх етапів у єдиній інтегрованій послідовності (рисунок 2.4). Такий сценарій ілюструє узгоджену взаємодію між усіма модулями системи та підкреслює важливість кожного етапу для забезпечення повноцінного освітнього процесу.

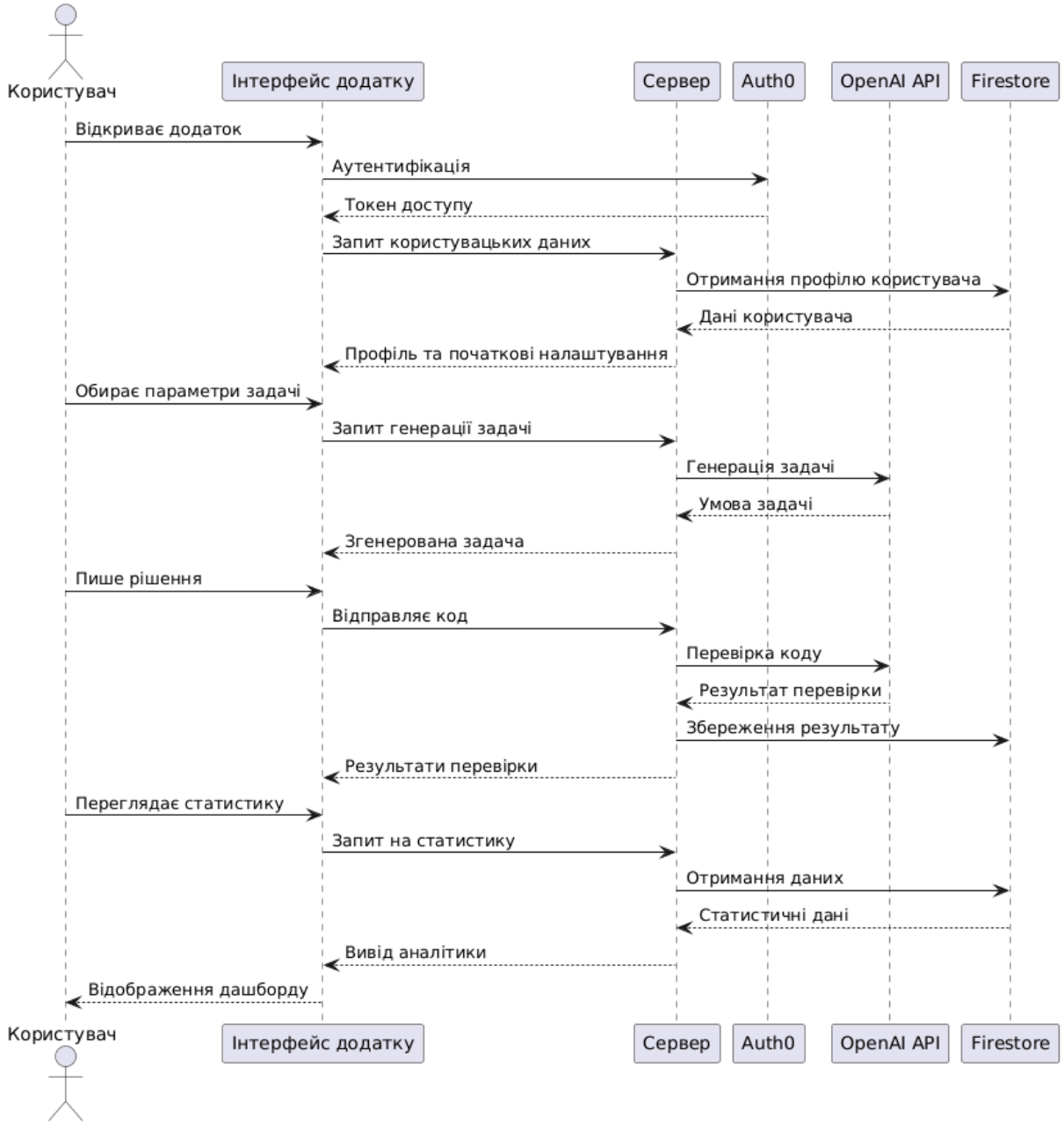


Рисунок 2.4 – Типова сесія користувача у вебдодатку

На основі результатів аналізу коду, модель повертає детальний висновок, який включає як правильність розв’язку, так і додаткові коментарі

або підказки. Сервер обробляє отриману інформацію, передає її клієнту для відображення, а також зберігає відповідні дані до Firestore. Таким чином забезпечується як миттєвий зворотний зв’язок, так і накопичення аналітичної інформації для подальшого використання.

Завершальним етапом типової сесії є перегляд статистики. Користувач надсилає відповідний запит, сервер витягує релевантні дані з Firestore, агрегує їх і повертає на клієнтську частину.

2.3 Обґрунтування обраного технологічного стеку

Під час проектування вебдодатку для генерації персоналізованих навчальних завдань з програмування особливу увагу було приділено вибору відповідного технологічного стеку, що має забезпечити надійність, масштабованість, безпеку, гнучкість і зручність розробки. Система має включати інтерактивний фронтенд з підтримкою динамічного інтерфейсу, потужну серверну частину для обробки запитів і логіки взаємодії з зовнішніми API, а також сучасне хмарне середовище для зберігання даних. Враховуючи ці вимоги, було обрано комбінацію технологій, які зарекомендували себе в освітніх та комерційних системах із подібним функціональним навантаженням.

Для реалізації клієнтської частини системи було обрано фреймворк React, що базується на мові програмування JavaScript з підтримкою розширення TypeScript. React дозволяє створювати масштабовані односторінкові додатки з компонентною архітектурою, що ідеально підходить для динамічних інтерфейсів з інтерактивними елементами – такими як редактор коду, форми налаштувань, повідомлення зворотного зв’язку. Застосування TypeScript дозволяє підвищити стабільність та передбачуваність коду завдяки статичній типізації. У рамках фронтенду також використовуються бібліотеки для побудови інтерфейсів, зокрема

shadcn/ui – набір стилізованих компонентів, що пришвидшує розробку і гарантує відповідність сучасним вимогам до UX.

Для маршрутизації між сторінками застосовується React Router, що забезпечує швидку зміну контенту без повного перезавантаження сторінки. Для роботи з формами обрано react-hook-form – легку та продуктивну бібліотеку, яка дозволяє валідовувати введення та керувати станом форм. Усе це дозволяє реалізувати зручну структуру взаємодії користувача з системою: від реєстрації та входу до роботи із задачами та перегляду аналітики. Крім того, враховуючи використання авторизації через сторонній сервіс, передбачено інтеграцію з Auth0 через відповідний SDK, що дозволяє швидко реалізувати вхід через Google або GitHub.

Серверна частина реалізована на основі фреймворку NestJS – прогресивного бекенд-фреймворку для Node.js, що побудований за принципами модульності та інверсії залежностей. NestJS дозволяє застосовувати архітектуру на основі контролерів, сервісів і модулів, чим значно полегшує масштабування та підтримку коду. Він також чудово інтегрується з TypeScript, що забезпечує єдиний стек між фронтендом і бекендом, спрощуючи передачу типів та моделювання даних. У проекті NestJS використовується для організації API, обробки авторизації, комунікації з OpenAI API, збереження даних у Firestore, а також логування та моніторингу.

Для генерації та перевірки умов задач обрано OpenAI API з доступом до моделей GPT-4 або GPT-3.5. Це рішення дозволяє динамічно створювати унікальні текстові умови завдань на основі заданих параметрів, а також аналізувати код користувача та повертати текстові пояснення з результатами перевірки. Однією з головних переваг застосування OpenAI є адаптивність генерації та здатність моделі до формулювання задач у стилістиці, близькій до реального викладача. Завдяки великій мовній моделі можливе покриття широкого спектру мов програмування, включно з Python, JavaScript, C++, Java та іншими.

Для зберігання даних у системі використовується хмарна база даних Firestore, яка є частиною платформи Firebase від Google. Firestore забезпечує зберігання структурованих документів у колекціях, підтримує режим реального часу, а також гарантує масштабованість і відмовостійкість. Обраний формат ідеально підходить для зберігання даних користувача, історії рішень, аналітики та профілів. Використання Firestore також дозволяє реалізувати швидкий пошук, фільтрацію, обробку статистичних даних і просту інтеграцію з іншими сервісами Google Cloud.

Додаток розгортається у хмарному середовищі Google Cloud Platform, що дозволяє налаштувати автоматичне масштабування ресурсів, контейнеризацію за допомогою Docker і забезпечити високу доступність. Такий підхід дозволяє зменшити витрати на інфраструктуру на ранніх етапах розробки, зберігаючи можливість легко масштабувати додаток у разі зростання кількості користувачів. Крім того, інтеграція з Firebase Hosting дозволяє швидко деплоїти фронтенд, а Cloud Functions – реалізовувати серверну логіку на безсерверній архітектурі, що може бути використано для подій, які не потребують постійно активного серверу.

Ще одним важливим аспектом є забезпечення безпеки додатку. Для цього використовується JWT-автентифікація, що дозволяє перевіряти запити до приватних ендпоінтів на основі токенів. Усі API-запити захищені через HTTPS, а дані в Firestore зберігаються згідно з ролями доступу, які можна конфігурувати на рівні правил безпеки Firebase. Крім того, вбудоване логування NestJS дозволяє контролювати події в системі, виявляти аномалії та проводити аудит.

Таким чином, обраний технологічний стек є збалансованим і оптимальним для поставленої задачі. Він поєднує передові інструменти для створення сучасного веб-інтерфейсу, надійний бекенд для обробки логіки, гнучкі засоби генерації та перевірки навчального контенту, а також хмарне середовище з безпечним і масштабованим зберіганням. Таке поєднання

дозволяє не лише створити ефективний MVP для проекту, а й розвивати систему у повноцінний навчальний сервіс у майбутньому.

2.4 Архітектурна концепція вебдодатку

Архітектурна концепція вебдодатку визначає логічну структуру основних компонентів системи, їхню взаємодію та розподіл обов'язків між клієнтською, серверною і зовнішніми частинами. В основі рішення лежить принцип розділення відповідальності, що дозволяє досягти високої гнучкості, масштабованості та зручності обслуговування проекту. Кожен модуль архітектури виконує конкретну роль, забезпечуючи узгоджену роботу в межах цілісної інфраструктури. Далі можна побачити візуальну схему взаємозв'язків між компонентами системи (рисунок 2.5).

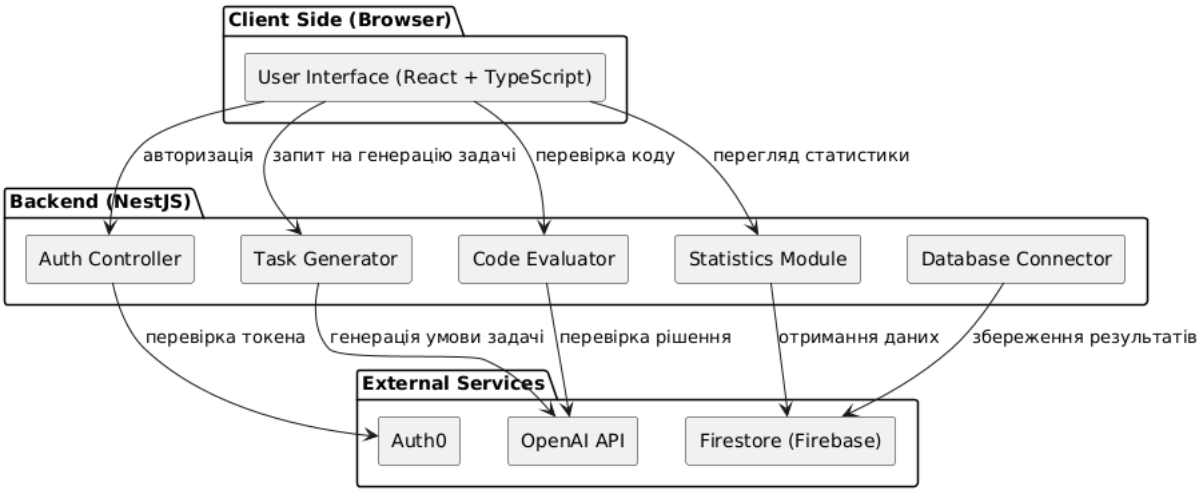


Рисунок 2.5 – Архітектура вебдодатку

Клієнтська частина реалізована у вигляді односторінкового додатку, побудованого на базі React з використанням TypeScript. Основне завдання фронтенду – забезпечити зручний інтерфейс для користувача, передавати запити до бекенду і відображати отримані результати. Тут відбувається авторизація користувача через інтеграцію з Auth0, формування параметрів

для генерації задач, написання та відправка коду, а також перегляд статистики. Завдяки чіткому поділу функцій, інтерфейс легко адаптується до потреб користувача і масштабування функціоналу.

Серверна частина реалізована за допомогою NestJS – модульного фреймворку, що дозволяє створювати ізольовані функціональні блоки. У наведеній архітектурі ці блоки включають Auth Controller для керування автентифікацією, Task Generator для взаємодії з OpenAI API при формуванні задач, Code Evaluator для перевірки рішень, Statistics Module для формування статистики користувача та Database Connector для зв’язку з базою даних Firestore. Кожен з цих модулів спілкується з іншими через чітко визначені API, що дозволяє ізолювати помилки і спрощує тестування окремих частин системи.

Зовнішні сервіси відіграють критично важливу роль у загальній архітектурі. Auth0 відповідає за перевірку токена та управління сесією користувача. OpenAI API є джерелом генерації умов задач і перевірки рішень, використовуючи великі мовні моделі. Firestore використовується як база даних для зберігання результатів, профілів і аналітичної інформації. Така інтеграція дозволяє відмовитися від розробки складної логіки генерації завдань на власному сервері, натомість використовуючи перевірені, надійні сервіси, що забезпечують високу якість відповіді та масштабованість. Обрана архітектура дозволяє досягти балансу між продуктивністю, функціональністю та гнучкістю розвитку системи.

2.5 Організація структури даних у хмарному середовищі

Для забезпечення коректної роботи вебдодатку та надійного зберігання користувацьких даних, завдань, результатів перевірки та аналітики необхідно спроектувати відповідну структуру даних у хмарному середовищі. Враховуючи вибраний стек технологій, оптимальним рішенням стало використання Firestore – гнучкої документоорієнтованої бази даних,

що належить до екосистеми Firebase від Google. Цей сервіс забезпечує не лише зручне зберігання документів у колекціях, а й синхронізацію в режимі реального часу, масштабованість і вбудовані механізми безпеки на рівні доступу до документів. Firestore особливо добре підходить для вебдодатків з динамічно змінюваним контентом і складною структурою взаємопов'язаних даних.

На концептуальному рівні модель зберігання даних базується на кількох основних сутностях: користувач, завдання, рішення та аналітика. Кожен користувач має унікальний ідентифікатор, який формується після автентифікації через Auth0 і використовується як ключ у Firestore. Документ користувача зберігає основну інформацію: ім'я, email, обрану мову програмування, дату реєстрації, рівень складності задач, прогрес у виконанні завдань. Ці дані дозволяють системі адаптувати генерацію задач та виводити персоналізовану аналітику. Кожен користувач також може мати вкладені колекції, наприклад розділ з розв'язаними задачами або накопичуваними досягненнями.

Завдання, що генеруються за допомогою OpenAI API, мають зберігатися як окремі документи в колекції tasks. Кожне завдання містить інформацію про тему, мову програмування, складність, дату створення, унікальний текст умови та параметри, які були використані при його генерації. Для того щоб уникнути дублювання, завдання можуть кешуватись – тобто зберігатись і використовуватись повторно, якщо параметри генерації повністю співпадають. Крім того, до кожного завдання можуть бути прикріплені зразкові рішення, які формуються автоматично і слугують для перевірки коректності майбутніх відповідей.

Рішення користувачів зберігаються у вкладеній колекції userSolutions всередині кожного профілю користувача або у глобальній колекції solutions з вказівкою на userId. Кожен документ такого типу містить посилання на завдання, відправлений код, часові мітки, результати перевірки, кількість спроб, а також відповідь, отриману від OpenAI API. Завдяки цьому можна

реалізувати зручну систему зворотного зв’язку, відновлення сесій користувача, а також аналітику помилок. Якщо користувач виконує кілька спроб розв’язання однієї задачі, ці записи агрегуються або зберігаються як окремі версії, що дозволяє аналізувати динаміку навчання.

Окрема увага приділяється зберіганню статистичних даних. Для цього використовується колекція statistics або агреговані поля в межах профілю користувача. Вона може містити дані про кількість успішних і неуспішних розв’язків, середній рівень складності виконаних завдань, активність по днях або тижнях, обрану мову програмування. Частина цієї інформації формується у реальному часі, а частина – підсумовується у фоновому режимі або при кожній дії користувача. Така гібридна модель дозволяє досягти балансу між оперативністю доступу до даних та ефективністю зберігання великих обсягів інформації.

Firestore також надає механізми для організації доступу до даних за допомогою правил безпеки, які дозволяють обмежити доступ до певних документів лише авторизованим користувачам. У межах даного проекту це дає змогу реалізувати модель із розмежуванням прав: користувач може переглядати лише власну статистику та історію рішень, а доступ до загальних задач або навчального контенту може бути відкритий для всіх. Крім того, при бажанні система дозволяє реалізувати адміністрування та модерування даних – наприклад, перевірку якості згенерованих задач або ручне додавання викладачем кастомного контенту.

Таким чином, структура зберігання даних у Firestore забезпечує не лише ефективне збереження навчального контенту та взаємодії користувача з системою, але й гнучкість для подальшого розвитку системи. У разі необхідності зберігання додаткових типів контенту або інтеграції з новими модулями, як-от рейтинг користувачів, система легко масштабуватиметься завдяки гнучкій документоорієнтованій природі Firestore.

2.6 Формування логіки генерації навчального контенту

Процес генерації навчального контенту є центральною складовою роботи системи, оскільки саме від його якості та адаптивності залежить успішність навчального досвіду користувача. Згенеровані задачі мають бути не лише релевантними до обраної теми й мови програмування, але й враховувати рівень складності, який відповідає поточному прогресу користувача. Використання великої мовної моделі через OpenAI API дозволяє створювати задачі, що варіюються за формулюванням, контекстом, структурою та навіть стилем викладу. Це мінімізує повторюваність і підвищує мотивацію до самостійної практики.

На початковому етапі користувач задає параметри задачі через інтерфейс. Ці параметри включають тему програмування, мову, рівень складності, а також, за потреби, тип задачі (наприклад, алгоритмічна, логічна, робота з масивами). Обрані дані передаються на сервер, де відбувається їх валідація і формування відповідного запиту до OpenAI API. Важливо, що структура запиту містить чітко сформульовану інструкцію для моделі – шаблон prompt, що дозволяє забезпечити сталість стилю задач і відповідність загальній методиці побудови навчального контенту.

Модель OpenAI повертає відповідь, яка містить умову задачі у вигляді тексту. У деяких випадках до задачі може додаватись зразковий варіант розв’язку, що дозволяє пізніше використати його для перевірки рішення користувача або побудови підказок. Система перевіряє структуру отриманої задачі на коректність, після чого зберігає її у Firestore разом із параметрами генерації. Таким чином, у випадку повторного запиту з ідентичними параметрами, задача може бути отримана з кешу, що економить ресурси й зменшує час очікування.

Генерація навчального контенту також підтримує можливість побудови задач різного формату – з відкритою відповіддю, тестуванням окремих функцій, обробкою вхідних та вихідних даних. Це дозволяє

охопити більший спектр практичних ситуацій і готувати користувача до різних форматів перевірки знань. У перспективі можливо реалізувати багаторівневу генерацію, коли декілька задач об’єднуються в сесію або проект із кількох етапів, що моделює реальні задачі в IT-компаніях або на технічних співбесідах.

Таким чином, побудова логіки генерації контенту ґрунтується на адаптивності, варіативності та актуальності навчального матеріалу. Застосування штучного інтелекту у цій частині системи дозволяє забезпечити індивідуалізований підхід до кожного користувача, підтримуючи принципи персоналізованого навчання і водночас знижуючи навантаження на розробників у частині створення статичного контенту. Це робить систему гнучкою, масштабованою і придатною до використання в умовах масового навчання.

2.7 Побудова процесу перевірки та зворотного зв’язку

Процес перевірки рішень є критично важливим компонентом навчальної системи, оскільки він забезпечує зворотний зв’язок і формує основу для оцінювання успішності засвоєння матеріалу. У традиційних навчальних середовищах перевірка здійснюється або через вбудовані тести, або вручну викладачем, що обмежує масштабованість. У межах проектованої системи реалізовано підхід, за якого для перевірки рішень користувачів застосовується OpenAI API. Це дозволяє автоматизувати аналіз коду та формувати зрозуміле текстове пояснення результату, орієнтоване на навчальні цілі.

Після написання коду користувачем у відповідному вікні інтерфейсу рішення передається на сервер разом із унікальним ідентифікатором задачі. Сервер формує запит до моделі OpenAI, який містить умову задачі, розв’язок користувача та інструкцію до перевірки. Ця інструкція є частиною спеціально сформульованого prompt, що має на меті спрямувати модель на

аналіз правильності логіки, виявлення помилок, а також формулювання освітнього зворотного зв’язку. Це значно відрізняється від класичної компіляції або тестування, оскільки модель здатна розпізнавати не лише технічні помилки, але й логічні невідповідності.

У відповіді модель надає вердикт щодо правильності розв’язку, описує виявлені проблеми та може запропонувати покрокове виправлення. Така форма перевірки особливо ефективна для новачків, які ще не мають усталеної практики тестування власного коду. Отримана відповідь аналізується сервером і повертається користувачу через інтерфейс. Користувач бачить текстовий висновок моделі, що може містити коментарі до стилю написання, неочевидні помилки, поради щодо оптимізації або уточнення логіки виконання функції.

Окрім освітнього аспекту, перевірка через OpenAI дозволяє створити гнучку систему оцінювання. У майбутньому можна реалізувати шкалу оцінювання, де модель, на основі рівня складності задачі та якості коду, формує числову або текстову оцінку. Також можлива реалізація адаптивного навчання: якщо модель виявила серію однотипних помилок, система може запропонувати додаткові задачі з фокусом на слабкі місця користувача. Це наближує систему до інтелектуального тьютора, що працює в реальному часі.

Таким чином, перевірка за допомогою OpenAI API дозволяє вийти за межі статичного тестування й наблизити навчальний процес до моделі індивідуального наставництва. Завдяки цьому користувач не лише отримує відповідь, чи правильне його рішення, а й розуміє, чому саме воно таке, як його можна покращити і що варто вивчити додатково.

2.8 Забезпечення безпеки, автентифікації та збереження даних

Одним з ключових аспектів при проектуванні будь-якої веб-системи, що передбачає обробку персональних даних, є забезпечення безпеки,

автентифікації та конфіденційності інформації. У запропонованому додатку ці функції реалізуються через інтеграцію з сервісом Auth0, що надає механізми авторизації та автентифікації на базі стандарту OAuth 2.0. Завдяки цьому система не зберігає паролі користувачів, а працює з перевіреними токенами доступу, що значно знижує ризики витоку даних. Такий підхід дозволяє також легко реалізувати вхід через облікові записи Google, GitHub та інших популярних платформ.

Усі запити до серверної частини додатку супроводжуються перевіркою JWT-токена, що містить інформацію про користувача та його права доступу. Верифікація токена відбувається на стороні сервера перед виконанням будь-якої дії з приватними даними, зокрема створенням нових задач, переглядом історії рішень або доступом до аналітики. Крім того, у Firestore застосовуються окремі правила доступу до колекцій, які обмежують можливість перегляду або редагування документів лише авторизованими користувачами, причому лише до тих документів, які їм належать.

Особливу увагу приділено захисту даних під час передачі. Усі API-запити від клієнта до сервера і назад, а також звернення до зовнішніх сервісів, таких як OpenAI або Firestore, відбуваються через захищене з’єднання HTTPS. Серверна частина реалізована з урахуванням принципів безпечного програмування – з фільтрацією вхідних даних, обмеженням прав доступу до маршрутів і логуванням спроб несанкціонованого доступу. Усе це забезпечує захист користувачів від несанкціонованого доступу, зловживання правами та порушення цілісності навчального процесу.

Таким чином, реалізація системи автентифікації, шифрування переданих даних і контроль доступу до ресурсів дозволяє гарантувати безпечне та надійне функціонування додатку, що є обов’язковою умовою для сучасних освітніх платформ з персоналізованим підходом.

3 ПРОГРАМНА РЕАЛІЗАЦІЯ

3.1 Опис застосованих інструментів реалізації

Реалізація вебдодатку для генерації персоналізованих завдань з програмування базується на використанні сучасного технологічного стеку, що дозволяє забезпечити ефективну взаємодію між клієнтською, серверною та зовнішніми складовими системи. Правильний вибір інструментів визначає стабільність, масштабованість, безпеку та гнучкість платформи, а також зручність її подальшого супроводу і розвитку. При виборі технологій враховувалися як функціональні вимоги системи, так і практичний досвід використання відповідних рішень у схожих проектах.

Клієнтська частина була реалізована з орієнтацією на швидку та інтерактивну взаємодію з користувачем, тому використовуються інструменти для створення динамічного інтерфейсу, що адаптується під різні пристрої. Важливою вимогою до інтерфейсу є зручність формування параметрів задачі, редагування коду та відображення результатів перевірки. З огляду на це, було обрано перевірені рішення, які дозволяють швидко будувати компонентну структуру додатку та масштабувати її за потреби.

Серверна частина виконує ключову роль у формуванні логіки генерації задач, перевірки коду, авторизації користувачів та збереження даних. Усі ці функції реалізовані за допомогою модульного фреймворку, що дозволяє підтримувати чітку структуру застосунку. У наступних підрозділах буде детально розглянуто кожен із застосованих інструментів.

3.1.1 Обрана мова програмування

У межах реалізації даного вебдодатку основною мовою програмування було обрано TypeScript. Це сучасне розширення JavaScript, яке підтримує статичну типізацію, дозволяє виявляти помилки ще на етапі

компіляції та забезпечує кращу підтримку інтегрованого середовища розробки. Застосування TypeScript значно підвищує масштабованість проекту, спрощує підтримку коду та полегшує спільну роботу над проектом завдяки чітко визначеним типам, інтерфейсам і структурі. Це особливо важливо для застосунків, що включають як клієнтську, так і серверну частину.

TypeScript обрано не лише з міркувань безпеки коду, а й тому, що він є природним вибором при використанні фреймворку React на фронтенді та NestJS на бекенді. Завдяки цьому розробник може використовувати єдину мову програмування на обох сторонах вебдодатку, що мінімізує контекстні перемикання та підвищує ефективність роботи. Усі модулі, типи запитів, інтерфейси для API та моделі об'єктів описані типізовано, що дає змогу значно скоротити кількість помилок, пов’язаних із неправильною структурою даних.

Ще однією перевагою є активна підтримка TypeScript у сучасних редакторах коду, таких як Visual Studio Code, який був використаний під час реалізації проекту. Автодоповнення, перевірка типів у реальному часі, виявлення помилок без запуску застосунку – усе це прискорює процес розробки та знижує імовірність неочевидних помилок у логіці. Крім того, велика спільнота розробників TypeScript забезпечує доступ до численних бібліотек, документації та прикладів, що також стало перевагою при реалізації системи.

Окремо варто відзначити, що TypeScript добре інтегрується з бібліотеками для роботи з API, маршрутизації, форм і стилізації, що робить його універсальним інструментом для реалізації повноцінного користувацького інтерфейсу. Його використання сприяє створенню структурованого, логічно організованого коду, який легко підтримувати, тестувати та розширювати. Таким чином, вибір TypeScript як основної мови програмування повністю відповідає вимогам до розробки вебдодатку.

3.1.2 Інструменти для реалізації клієнтської частини

У реалізації клієнтської частини вебдодатку ключову роль відіграє бібліотека React, яка забезпечує компонентний підхід до побудови інтерфейсу. Використання React дозволяє створювати гнучку архітектуру з незалежними UI-компонентами, що спрощує розробку, тестування й подальше масштабування інтерфейсу. Компоненти, які відповідають за генерацію задач, редагування коду, перегляд статистики та авторизацію, мають чітке розмежування відповідальностей і взаємодіють між собою через керування станом та передавання параметрів.

Для створення естетичного, адаптивного та доступного інтерфейсу у вебдодатку використовується бібліотека Mantine. Вона надає велику кількість стилізованих елементів управління, таких як кнопки, форми, вкладки, таблиці, модальні вікна. Це дозволяє будувати інтерфейс швидко й уніфіковано, при цьому зберігаючи контроль над кастомізацією. Завдяки адаптивним властивостям бібліотеки компоненти коректно відображаються на різних типах пристроїв, що є важливим для користувачів, які працюють з мобільних чи планшетних екранів.

У якості редактора коду було використано monaco-editor – той самий інструмент, що лежить в основі Visual Studio Code. Його інтеграція у вебдодаток дозволяє надати користувачу середовище з підсвіткою синтаксису, автодоповненням, підтримкою кількох мов програмування та зручним інтерфейсом для написання рішень. Monaco-editor працює всередині компонента React і має хорошу документацію, що дозволило інтегрувати його без втрати продуктивності застосунку.

Для роботи з формами у застосунку використано бібліотеку react-hook-form, яка надає легкий API для керування станом, валідації та обробки подій форм. Це зменшує кількість шаблонного коду та забезпечує реактивність при зміні значень. А для маршрутизації між сторінками, як-от головна, задачі, статистика чи профіль, використовується React Router, що

дозволяє створювати SPA-застосунок із динамічною зміною маршруту без перезавантаження сторінки. Увесь клієнтський код зібрано за допомогою Vite – сучасного високошвидкісного білдера, який забезпечує миттєвий старт, гаряче оновлення модулів та швидку збірку у фінальний продакшн-бандл.

3.1.3 Фреймворк NestJS для побудови серверної логіки

Фреймворк NestJS було обрано як основу для реалізації серверної частини вебдодатку завдяки його модульній архітектурі, підтримці TypeScript та сумісності з сучасними практиками побудови API. NestJS базується на концепціях об'єктно-орієнтованого програмування, інверсії залежностей та декораторів, що дозволяє будувати масштабовані та логічно структуровані серверні застосунки. У межах цього проекту NestJS використовується для обробки HTTP-запитів, організації маршрутизації, реалізації логіки генерації та перевірки задач, авторизації користувачів та взаємодії з хмарними сервісами.

Серверна частина побудована у вигляді набору окремих модулів: модуль авторизації, модуль генерації задач, модуль перевірки коду, модуль статистики та модуль взаємодії з базою даних. Кожен модуль складається з контролера, сервісу та, за потреби, окремих утиліт або DTO для типізації даних. Такий підхід дозволяє легко масштабувати систему, додаючи нові функціональні блоки без порушення логіки існуючих. Важливою перевагою є можливість централізованої обробки помилок, логування, а також впровадження механізмів кешування або обмеження запитів до зовнішніх API.

Усі запити від клієнтської частини до серверної обробляються за допомогою REST API, яке реалізовано через HTTP контролери NestJS. При отриманні запиту сервер виконує логіку відповідного сервісу: наприклад, у випадку генерації задачі формується запит до OpenAI API, у випадку

перевірки – формуються аргументи для виводу результату. Отримані дані, включно з результатами генерації чи перевірки, можуть бути збережені у Firestore через окремий модуль доступу до бази даних, що також реалізований в межах NestJS.

Завдяки використанню NestJS розробка серверної частини стала чіткою й структурованою, із можливістю повторного використання коду, автоматичного тестування та зручною інтеграцією сторонніх бібліотек. Фреймворк добре документований, підтримує сучасні засоби безпеки, middleware, guard-и та інші механізми, необхідні для захисту маршрутів і даних. У результаті NestJS дозволив реалізувати повноцінну серверну інфраструктуру для вебдодатку, що підтримує взаємодію з клієнтом, зовнішніми сервісами та хмарним середовищем, залишаючись при цьому гнучкою та масштабованою.

3.1.4 OpenAI API як механізм генерації й перевірки

OpenAI API є ключовим компонентом інтелектуальної частини вебдодатку, оскільки виконує функції генерації умов задач та перевірки розв’язків користувачів. Завдяки великій мовній моделі GPT, система здатна динамічно створювати завдання з програмування на основі заданих параметрів, а також аналізувати код, наданий користувачем, і повертати текстовий відгук із поясненнями. Таке застосування API дозволяє досягти високого рівня персоналізації навчального контенту та імітує роль живого наставника або викладача, який не лише оцінює відповідь, але й надає коментарі.

Інтеграція з OpenAI API реалізована на серверній частині вебдодатку, написаній на NestJS. Коли користувач створює запит на генерацію задачі, система формує відповідний prompt, у якому вказано тему, мову програмування, рівень складності та стиль задачі. Цей prompt надсилається через REST-запит до OpenAI, який повертає згенеровану текстову

інструкцію – умову задачі. Після цього ця умова передається на клієнтську частину, де користувач має змогу ознайомитись із нею та розпочати написання розв’язку.

Аналогічно, при надсиланні рішення користувача до сервера формується новий prompt, який включає умову задачі, розв’язок і чітку інструкцію для моделі, що просить перевірити правильність коду та виявити помилки. Модель повертає текстову відповідь, яка може містити коментарі до логіки, виявлені баги, стилістичні зауваження та поради щодо покращення. Така форма зворотного зв’язку значно перевершує традиційні підходи на основі жорстких тестів і дозволяє користувачу краще зрозуміти помилку, а не просто побачити, що розв’язок невірний.

Для оптимізації роботи з API були враховані обмеження на кількість запитів та обсяг переданих даних. Усі виклики до OpenAI супроводжуються контролем таймаутів, логуванням і базовим кешуванням відповідей при повторному зверненні з ідентичними параметрами. Це дозволяє знизити навантаження на API та зменшити час очікування відповіді. У перспективі система може бути доповнена логікою вибору моделі залежно від типу задачі – GPT-3.5 для загального генеративного контенту та GPT-4 для глибшої аналітики коду. Таким чином, OpenAI API забезпечує інтелектуальну основу системи, яка перетворює її з генератора задач у справжній персоналізований навчальний інструмент.

3.2 Побудова бекенд-функціоналу системи

Сервіс автентифікації відповідає за створення нового користувача після входу через Auth0, використовуючи ідентифікатор та електронну пошту. У коді, поданому у лістингу 3.1, перевіряється унікальність auth0Id` у колекції Firestore, і якщо такий користувач вже існує – створення припиняється. Далі аватар користувача завантажується з вказаного URL і зберігається у Firebase Storage під іменем, закодованим на основі

електронної пошти. Після цього у Firestore створюється новий запис з усіма необхідними даними користувача, включаючи посилання на збережений аватар.

Лістинг 3.1 – Програмний код, що створює нового користувача на основі даних із Auth0

```
@Injectable()
export class AuthService {
  constructor(private readonly firebaseService: FirebaseService) {}
  async createAuth0User({ auth0Id, email, name, avatar }: CreateAuth0User): Promise<void> {
    const db = this.firebaseService.getFirestore();
    const users = db.collection('users');
    const existing = await users.where('auth0Id', '==', auth0Id).get();
    if (!existing.empty) throw new Error('User already exists');
    const res = await fetch(avatar);
    if (!res.ok) throw new Error('Failed to fetch avatar');
    const buffer = Buffer.from(await res.arrayBuffer());
    const path = `avatars/${Buffer.from(email).toString('base64')}.jpg`;
    await getStorage().bucket().file(path).save(buffer, {
  contentType: res.headers.get('content-type')});
    await users.doc().set({
      auth0Id,
      email,
      name,
  avatar:`https://storage.googleapis.com/${getStorage().bucket().name}/${path}`,
      problems: [],
      solutions: [],});}
```

Сервіс Firebase виконує ініціалізацію з’єднання з хмарною інфраструктурою Google на основі переданих через змінні середовища облікових даних. У лістингу 3.2 показано, як шляхом виклику initializeApp створюється об’єкт із доступом до Firestore і хмарного сховища Storage. Отримані інстанси зберігаються у властивостях класу та повертаються методами getFirestore і getBucket для подальшого використання в інших частинах застосунку. Така інкапсуляція спрощує доступ до Firebase з боку сервісів, що працюють із базою даних або файлами.

Лістинг 3.2 – Програмний код, що ініціалізує Firebase у застосунку, надаючи доступ до Firestore та хмарного сховища

```
@Injectable()
export class FirebaseService {
  private readonly firestore;
  private readonly bucket;
  constructor(config: ConfigService) {
    const credentials = {
      projectId: config.get('FIREBASE_PROJECT_ID'),
      privateKey:
config.get('FIREBASE_PRIVATE_KEY')?.replace(/\\n/g, '\n'),
      clientEmail: config.get('FIREBASE_CLIENT_EMAIL'),};
    admin.initializeApp({
      credential: admin.credential.cert(credentials),
      databaseURL:
`https://${credentials.projectId}.firebaseio.com`,storageBucke
t: `${credentials.projectId}.firebasestorage.app`,});
    this.firestore = admin.firestore();
    this.bucket = getStorage().bucket();}
  getBucket() {return this.bucket;}
  getFirestore() {return this.firestore;}}
```

У лістингу 3.3 подано сервіс, який виконує пошук користувача за унікальним значенням auth0Id у колекції users бази Firestore. Якщо

користувача знайдено, його дані об'єднуються з ідентифікатором документа і повертаються як об'єкт типу User. У випадку, якщо результат запиту порожній, функція повертає null. Така логіка забезпечує базову перевірку наявності користувача в системі перед виконанням подальших дій.

Лістинг 3.3 – Програмний код, що отримує користувача з бази Firestore за auth0Id і повертає його дані або null

```
@Injectable()
export class UserService {
  constructor(
    private readonly firebase: FirebaseService,
    private readonly config: RemoteConfigService,) {}
  async getUserByAuth0Id(auth0Id: string): Promise<User | null> { const snapshot = await this.firebase.getFirestore().collection('users').where('auth0Id', '==', auth0Id).limit(1).get();
    return snapshot.empty ? null : { id: snapshot.docs[0].id, ...snapshot.docs[0].data() } as User;}}
```

У лістингу 3.4 представлено контролер, який реалізує захищені маршрути для роботи з розв’язками задач. Перший маршрут дозволяє отримати один конкретний розв’язок за його ідентифікатором, інший дозволяє повертати всі розв’язки, що належать поточному користувачу. Третій маршрут приймає новий розв’язок, який передається у вигляді problemId та code, і передає його на обробку відповідному сервісу. Усі маршрути захищені за допомогою jwt-автентифікації.

Лістинг 3.4 – Програмний код, що обробляє маршрути для отримання і створення розв’язків користувача

```
@Controller('/solutions')
@UseGuards(AuthGuard('jwt'))
export class SolutionsController {
```

Продовження лістингу 3.4

```
    constructor(private readonly solutionsService:
SolutionsService) {}
    @Get('/:id')
   getOne(@CurrentUser() { id }, @Param('id') solutionId:
string) {
      return this.solutionsService.getSolution(id,
solutionId);}
    @Get()
    getAll(@CurrentUser() { id }) {
      return this.solutionsService.getSolutions(id);}
    @Post()
    create(
      @CurrentUser() { id },
      @Body() { problemId, code }: CreateSolutionInput,) {
      return this.solutionsService.uploadSolution(id,
problemId, code);}}
```

У лістингу 3.5 наведено контролер, що відповідає за обробку запитів до задач користувача. Перший маршрут дозволяє створити нову задачу на основі вхідних параметрів та auth0Id користувача. Другий маршрут повертає всі задачі, які були згенеровані даним користувачем. Третій маршрут надає доступ до конкретної задачі за її ідентифікатором.

Лістинг 3.5 – Програмний код, що обробляє захищені маршрути для створення, отримання всіх і окремих задач користувача

```
  @Controller('/problems')
  @UseGuards(AuthGuard('jwt'))
  export class ProblemController {
    constructor(private readonly problemService:
ProblemService) {}
    @Post()
    generate(@Body() body: CreateProblemInput,
@CurrentUser() { id }) {
```

Продовження лістингу 3.5

```
    return this.problemService.generateProblem(id, body);}
  @Get()
  getAll(@CurrentUser() { id }) {
    return this.problemService.getProblems(id);}
  @Get('/:id')
  getOne(@CurrentUser() { id }, @Param('id') problemId:
string) {
   return this.problemService.getProblem(id, problemId);}}
```

Усі запити захищені механізмом jwt-автентифікації, що гарантує доступ лише авторизованим користувачам.

3.3 Генерація навчального контенту за допомогою OpenAI API

У лістингу 3.6 показано сервіс, який ініціалізує клієнт openai за допомогою api-ключа, отриманого з конфігурації середовища. Після створення екземпляра клієнта він зберігається як приватне поле класу. Метод getOpenAI дозволяє іншим модулям системи отримати доступ до цього клієнта для подальших запитів.

Лістинг 3.6 – Програмний код, що ініціалізує клієнт OpenAI на основі API-ключа і надає доступ до нього іншим модулям системи

```
@Injectable()
export class OpenAIService {
  private readonly openAI;
  constructor(config: ConfigService) {
    this.openAI = new OpenAI({
      apiKey:       config.get('OPENAI_API_KEY')       ??
process.env.OPENAI_API_KEY,});}
  getOpenAI() {return this.openAI;}}
```

У лістингу 3.7 представлено сервіс, який відповідає за генерацію задач користувачу з використанням моделі OpenAI та збереження результату у Firestore. Спочатку здійснюється пошук користувача за його auth0Id, після чого формується запит до openai з параметрами задачі – складністю, мовою, темами та додатковими нотатками. Отримана у форматі json відповідь парситься та доповнюється ідентифікатором і вхідними даними. Далі згенерована задача додається до колекції задач поточного користувача у базі.

Лістинг 3.7 – Програмний код, що обробляє створення задачі через OpenAI API та зберігає її у Firestore

```
@Injectable()
export class ProblemService {
  constructor(
    private readonly openAI: OpenAIService,
    private readonly firebase: FirebaseService,) {}
  async getProblem(auth0Id: string, id: string) {
    const user = await this.findUser(auth0Id);
    return user.problems.find((p) => p.id === id) ?? null;}
  async getProblems(auth0Id: string) {
    return (await this.findUser(auth0Id)).problems;}
  async generateProblem(auth0Id: string, input: CreateProblemInput) {
    const userDoc = (await this.firebase.getFirestore().collection('users').where('auth0Id', '==', auth0Id).get()).docs[0];
    const { choices } = await this.openAI.getOpenAI().chat.completions.create({model: 'gpt-4o',messages: [
{ role:'system', content: 'You are a helpful assistant.'},{
          role: 'user',
content: `Plz generate a ${input.difficulty} ${input.language} task about ${input.topics.join(',')}. ${input.notes ? `Notes: ${input.notes}` : ''} JSON only.`,},]});
```

Продовження лістингу 3.7

```
        const message = 
choices[0].message.content.replace(/^```json|```$/g, '');
        const data = JSON.parse(message);
        const problem = { id: v4(), ...data, ...input };
        await userDoc.ref.update({
          problems: [...userDoc.data().problems, problem],});
        return problem;}
      private async findUser(auth0Id: string) {
        const snap = await 
this.firebase.getFirestore().collection('users').where('auth0I
d', '==', auth0Id).get();
        if (snap.empty) throw new Error('User not found');
        return snap.docs[0].data();}}
```

У лістингу 3.8 представлено сервіс, який реалізує повний цикл роботи з розв’язками задач користувача. Метод uploadSolution формує prompt із умовою задачі та запропонованим розв’язком, надсилає його до OpenAI API і на основі відповіді визначає правильність рішення. Якщо розв’язок є коректним, він зберігається у Firestore у масиві solutions відповідного користувача.

Крім того, сервіс надає методи для отримання конкретного розв’язку разом із пов’язаною задачею, а також для отримання списку всіх розв’язків користувача з відповідними задачами. Усі запити супроводжуються попередньою перевіркою наявності користувача та задачі у базі.

Лістинг 3.8 – Програмний код, що дозволяє створювати, отримувати та перевіряти розв’язки задач користувача за допомогою OpenAI API

```
      async getSolution(auth0Id: string, solutionId: string) {
        const user = await this.findUser(auth0Id);
        const solution = user.solutions.find((s) => s.id === 
solutionId);
        if (!solution) return null;
```

Продовження лістингу 3.8

```
        const problem = user.problems.find((p) => p.id === solution.problemId);
        if (!problem) throw new Error('Problem not found');
        return { ...solution, problem };}
      async getSolutions(auth0Id: string) {
        const user = await this.findUser(auth0Id);
        return user.solutions.map((s) => {
          const problem = user.problems.find((p) => p.id === s.problemId);
          if (!problem) throw new Error('Problem not found');
          return { ...s, problem };});}
      async uploadSolution(auth0Id: string, problemId: string, code: string) {
        const userDoc = (await this.firebase.getFirestore().collection('users').where('auth0Id', '==', auth0Id).get()).docs[0];
        const user = userDoc.data();
        const problem = user.problems.find((p) => p.id === problemId);
        if (!problem) throw new Error('Problem not found');
    this.openAI.getOpenAI().chat.completions.create({model: 'gpt-4o',messages: [
    {role: 'system', content: 'You are a helpful assistant.'},{
    role: 'user', content: `Here is the problem:\n${problem.title}\n${problem.description}\nHere is the solution:\n${code}\nReturn{isCorrect,explanation}JSON.`,},],};
        const raw = choices[0].message.content.replace();
        const result = JSON.parse(raw);
        const newSolution = { id: v4(), problemId, code };
        await userDoc.ref.update({ solutions: [...user.solutions, newSolution] });
        return { isCorrect: true, solution: { ...newSolution, problem } };}
```

3.4 Реалізація фронтенду системи

У лістингу 3.9 наведено код, який створює клієнтський API для взаємодії з бекендом за допомогою Redux Toolkit Query. У конфігурації вказано базову URL-адресу, що зчитується з середовища, а також логіку автоматичного додавання токена авторизації до заголовків запитів. Описано ендпоінти для отримання даних користувача, задач і розв'язків, а також для генерації нової задачі та надсилання розв'язку. Створені хуки експортуються і використовуються в інших компонентах клієнтської частини для забезпечення реактивної роботи з API.

Лістинг 3.9 – Програмний код, що конфігурує клієнтський API для роботи з бекендом через Redux Toolkit Query

```
export const api = createApi({
  reducerPath: "mainApi",
  tagTypes: ["CurrentUser", "Quiz", "Room"],
  baseQuery: fetchBaseQuery({
    baseUrl: env.VITE_API_URL,
    prepareHeaders: (headers, { getState }) => {
const token = (getState() as RootState).auth.accessToken;
if (token) headers.set("authorization", `Bearer ${token}`);
      return headers;},}),
  endpoints: (builder) => ({
getCurrentUser:    builder.query({    query:    ()    =>
"/users/current", providesTags: ["CurrentUser"] }),
getSolution:    builder.query({    query:    (id)    =>
`/solutions/${id}` }),
getSolutions: builder.query({ query: () => "/solutions" }),
getProblem:    builder.query({    query:    (id)    =>
`/problems/${id}` }),
getProblems: builder.query({ query: () => "/problems" }),
generateProblem: builder.mutation({
```

Продовження лістингу 3.9

```
    query: (body) => ({ url: "/problems", method: "POST", body
}),}), uploadSolution: builder.mutation({
    query: ({ code, problemId }) => ({ url: "/solutions",
method: "POST", body: { code, problemId } }),}),}),});
    export const {
      useGetCurrentUserQuery,
      useGenerateProblemMutation,
      useGetProblemQuery,
      useGetProblemsQuery,
      useGetSolutionQuery,
      useGetSolutionsQuery,
      useUploadSolutionMutation,
    } = api;
```

У лістингу 3.10 представлено компонент, що відповідає за відображення сторінки задачі з усією необхідною інформацією. Користувач бачить назву задачі, її складність, тематику, опис та приклади введення і виведення, представлені у вигляді сповіщень. У правій частині розміщено редактор коду, де можна написати розв’язок, а також кнопку для його надсилання. Натискання цієї кнопки відкриває модальне вікно підтвердження з подальшим відправленням даних. Компонент реалізує зручний інтерфейс для практичного виконання задачі без переходу на інші сторінки.

Лістинг 3.10 – Програмний код, що відображає сторінку з описом задачі, її прикладами, редактором коду та формою надсилання розв’язку

```
    export const ProblemPage: FC = () => {
      const { id } = useParams<{ id: string }>();
      const [code, setCode] = useState("");
      const [isModalOpen, setIsModalOpen] = useState(false);
      const    {    data,    isLoading,    error    }    =
useGetProblemQuery(id!);
```

Продовження лістингу 3.10

```
  if (isLoading) return <LoadingOverlay />;
  if (error || !data) return <div>Error</div>;
  const problem = data;
  return ( <>
      <Split h="100%" spacing="sm" size="sm">
        <Split.Pane initialWidth="30%">
          <Stack p="md">
            <Title order={2}>{problem.title}</Title>
            <Group>
              <Badge bg={problem.difficulty === "easy" ?
"green" : problem.difficulty === "medium" ? "orange" : "red"}>
    {problem.difficulty}</Badge>  {problem.topics.map((t)  =>
(<Badge key={t}>{t}</Badge>))}</Group>
    <Markdown text={problem.description} />
    {problem.examples.map((e, i) => (
    <Notification  key={i}  title={`Example  ${i  +  1}`}
withCloseButton={false}>
    <Text fw={700}>Input:</Text> <Text>{e.input}</Text>
    <Text fw={700}>Output:</Text> <Text>{e.output}</Text>
    <Text fw={700}>Explanation:</Text> <Text>{e.explanation}>
          </Stack>
        </Split.Pane>
        <Split.Pane grow>
          <Stack>
            <Paper p="xs">
              <Group justify="space-between">
                <Group>
                  <Title order={3}>Code</Title>
                  <Badge>{problem.language}</Badge>
                </Group>
                <Button
                  disabled={!code}
                  onClick={() => setIsModalOpen(true)}
           rightSection={<IconArrowRight size={14}/>}>Submit
```

Продовження лістингу 3.10

```
                    </Button>
                  </Group>
                </Paper>
                <Box flex={1}>
                  <CodeEditor            language="javascript"
value={code} onChange={(v) => setCode(v ?? "")} height="100%" />
                </Box>
              </Stack>
            </Split.Pane>
          </Split>
          <SubmitSolutionModal   code={code}   problemId={id!}
open={isModalOpen} onClose={() => setIsModalOpen(false)} />;};
```

У лістингу 3.11 подано реалізацію модального вікна, яке відповідає за підтвердження надсилання розв’язку задачі. Після натискання кнопки submit виконується запит до openai api для перевірки правильності коду. Якщо відповідь вказує на помилку, користувачу показується пояснення у вигляді сповіщення з підтримкою markdown. У випадку правильного розв’язку користувач автоматично перенаправляється на сторінку перегляду свого рішення. Таким чином, компонент дозволяє інтерактивно взаємодіяти з системою перевірки рішень.

Лістинг 3.11 – Програмний код, що відображає модальне вікно для надсилання розв’язку задачі, обробляє результат через OpenAI API

```
    export              const              SubmitSolutionModal:
FC<SubmitSolutionModalProps> = ({open, onClose, code, problemId,
    }) => { const navigate = useNavigate();
      const   [uploadSolution,   {   isLoading,   data   }]   =
useUploadSolutionMutation();
      const handleSubmit = async () => {
        const  res  =  await  uploadSolution({  code,  problemId
}).unwrap();
```

Продовження лістингу 3.11

```
        if (res.isCorrect) navigate(`/solutions/${res.solution.id}`);};
      return (
        <Modal title="Submit you solution" opened={open} onClose={onClose} size="md" padding="lg">
          <LoadingOverlay visible={isLoading} />
          <Stack gap="md">
            {data?.isCorrect === false && (
              <Notification color="red" title="Your solution is incorrect" withCloseButton={false}>
                <Markdown text={data.explanation} />
              </Notification>
            )}
            <Text>Are you sure you want to submit your solution?</Text>
          </Stack>
          <Group mt="xl" justify="flex-end">
            <Button onClick={handleSubmit}>Submit</Button>
          </Group>
        </Modal>
      );
    };
```

У лістингу 3.12 наведено компонент, який реалізує інтерфейс для створення нової задачі за допомогою модального вікна. Користувач заповнює поля з темами, складністю, мовою програмування та додатковими нотатками, після чого відправляє форму. В результаті формується запит до backend-API, який викликає генерацію задачі через OpenAI. У разі успішного створення задачі користувач автоматично перенаправляється на сторінку перегляду новоствореної задачі. Компонент поєднує зручну форму введення з асинхронною логікою генерації.

Лістинг 3.12 – Програмний код, що реалізує модальне вікно для генерації нової задачі і відправляє запит до API для створення задачі

```
export const CreateProblemModal: FC<CreateProblemModalProps> = ({ open, onClose }) => {
  const navigate = useNavigate();
  const [generateProblem, { isLoading }] = useGenerateProblemMutation();
  const {
    control,
    handleSubmit,
    formState: { errors },
  } = useForm<FormState>({
    resolver: zodResolver(schema),
  defaultValues: { topics: [], difficulty: null, notes: "" });
  const onSubmit = async (data: FormState) => {
    const problem = await generateProblem(data).unwrap();
    navigate(`/problems/${problem.id}`);};
  return (
    <Modal title="Create a new problem" size="md" padding="lg" opened={open} onClose={onClose}>
      <LoadingOverlay visible={isLoading} />
      <Stack gap="md">
        <Controller
          control={control}
          name="topics"
          render={({ field, fieldState }) => (
            <TagsInput label="Topics" {...field} error={fieldState.error?.message} />)}/>
        <Controller
          control={control}
          name="difficulty"
          render={({ field, fieldState }) => (
            <Combobox
              store={useCombobox()}
  onOptionSubmit={(val) => field.onChange(val)}>
```

Продовження лістингу 3.12

```
                    <Combobox.Target>
                      <InputBase                label="Difficulty"
error={fieldState.error?.message}>
                        {field.value  ||  <Input.Placeholder>Pick
difficulty</Input.Placeholder>}
                      </InputBase>
                    </Combobox.Target>
                    <Combobox.Dropdown>
                      <Combobox.Options>
                        {difficulties.map((d) => (
                          <Combobox.Option key={d.value}>
                            <Text c={d.color}>{d.label}</Text>
                          </Combobox.Option>))}
                      </Combobox.Options>
                    </Combobox.Dropdown>
                  </Combobox>)}/>
              <Textarea          label="Additional          notes"
{...control.register("notes")} error={errors.notes?.message} />
            </Stack>
            <Group mt="xl" justify="flex-end">
              <Button
onClick={handleSubmit(onSubmit)}>Create</Button>
            </Group>
          </Modal>);};
```

У лістингу 3.13 показано реалізацію сторінки, що відображає список усіх згенерованих задач користувача у вигляді таблиці. У верхній частині сторінки розміщено кнопку, яка відкриває модальне вікно для створення нової задачі. У разі завантаження даних з сервера відображається індикатор очікування, а у випадку помилки – повідомлення про помилку. Компонент поєднує перегляд, створення і взаємодію з задачами в єдиному інтерфейсі.

Лістинг 3.13 – Програмний код, що відображає сторінку зі списком згенерованих задач та кнопкою для створення нової задачі

```
export const ProblemsPage: FC = () => {
  const [open, setOpen] = useState(false);
  const { data, isLoading, error } = useGetProblemsQuery();
  if (isLoading) return <LoadingOverlay />;
  if (error || !data) return <div>Error</div>;
  return (<>
      <Container>
        <Group justify="space-between" mb="lg">
          <Title order={1}>Problems</Title>
          <Button
            leftSection={<IconAi size={20} />}
            rightSection={<IconArrowRight size={14} />}
            onClick={() => setOpen(true)}>
            Create a new one
          </Button>
        </Group>
        <Paper withBorder>
          <ProblemsTable problems={data} />
        </Paper>
      </Container>
      <CreateProblemModal open={open} onClose={() => setOpen(false)} /> </>
  );
};
```

У лістингу 3.14 представлено компонент, що відповідає за головну сторінку вебдодатку та відображає аналітичну інформацію у вигляді візуалізованої статистики. Сторінка складається з сегментів статистики, графічних показників та панелі керування параметрами. Такий інтерфейс дозволяє користувачеві отримати узагальнений огляд своєї активності та ефективності взаємодії із системою.

Лістинг 3.14 – Програмний код, що формує головну сторінку вебдодатку з елементами статистики у вигляді графіків

```
export const HomePage: FC = () => { return (
    <Container>
      <Title order={1} mb="lg">Dashboard</Title>
      <Stack gap="md">
        <Group gap="md" align="stretch">
          <StatsSegments />
          <StatsGridIcons />
        </Group>
        <StatsControls />
      </Stack>
    </Container>);};
```

У лістингу 3.15 подано компонент, що реалізує сторінку перегляду розв’язку задачі користувача. У верхній частині сторінки відображається заголовок та компонент для візуального представлення точності рішень. Далі показано інформацію про задачу: назва, складність, тематика, опис та приклади з поясненнями. Окремим блоком представлено підсвічений код рішення з вказаною мовою програмування. Така структура дозволяє користувачу зручно аналізувати виконані задачі та якість власного коду.

Лістинг 3.15 – Програмний код, що відображає сторінку з детальним переглядом розв’язку користувача

```
export const SolutionPage: FC = () => {
  const { id } = useParams<{ id: string }>();
  const { data, isLoading, error } = useGetSolutionQuery(id!);
  if (isLoading) return <LoadingOverlay />;
  if (error || !data) return <div>Error</div>;
  const { code, problem } = data;
  return (
    <Container pb="md">
```

Продовження лістингу 3.15

```
          <Group mb="lg" justify="space-between">
            <Title order={1}>Solution</Title>
            <Box w="300px">
            <Accuracy positiveValue={90} negativeValue={10} />
            </Box>
          </Group>
          <Stack gap="lg">
            <Paper withBorder p="md">
              <Title order={2}>{problem.title}</Title>
              <Group gap="md">
                <Badge  bg={problem.difficulty  ===  "easy"  ?
"green" : problem.difficulty === "medium" ? "orange" : "red"}>
                  {problem.difficulty}
                </Badge>
                {problem.topics.map((t) => (
                  <Badge key={t} bg="gray">{t}</Badge>))}
              </Group>
              <Markdown text={problem.description} />
              {problem.examples.map((e, i) => (
                <Notification  key={i}  title={`Example  ${i  +
1}`} withCloseButton={false}>
            <Paper withBorder p="md">
              <Group gap="md" align="center" mb="md">
                <Title order={2}>Code</Title>
                <Badge bg={problem.language === "javascript" ?
"yellow" : problem.language === "java" ? "purple" : "blue"}>
                  {problem.language}
                </Badge>
              </Group>
     <CodeHighlight code={code} language={problem.language} />
            </Paper>
          </Stack>
        </Container>);};
```

3.5 Інтерфейс системи як складова навчального досвіду

Головна сторінка вебдодатку відіграє роль аналітичного інструменту для користувача, дозволяючи йому відстежувати прогрес у виконанні задач, аналізувати ефективність рішень та оцінювати загальну продуктивність. Цей інтерфейс має бути інтуїтивно зрозумілим, візуально структурованим та забезпечувати швидкий доступ до ключових показників. Застосування чітких візуальних маркерів, таких як кольорові індикатори рівнів складності або тенденції зміни продуктивності, сприяє кращому сприйняттю інформації.

Далі можна побачити інформаційну панель користувача з ключовими метриками взаємодії з системою (рисунок 3.1).

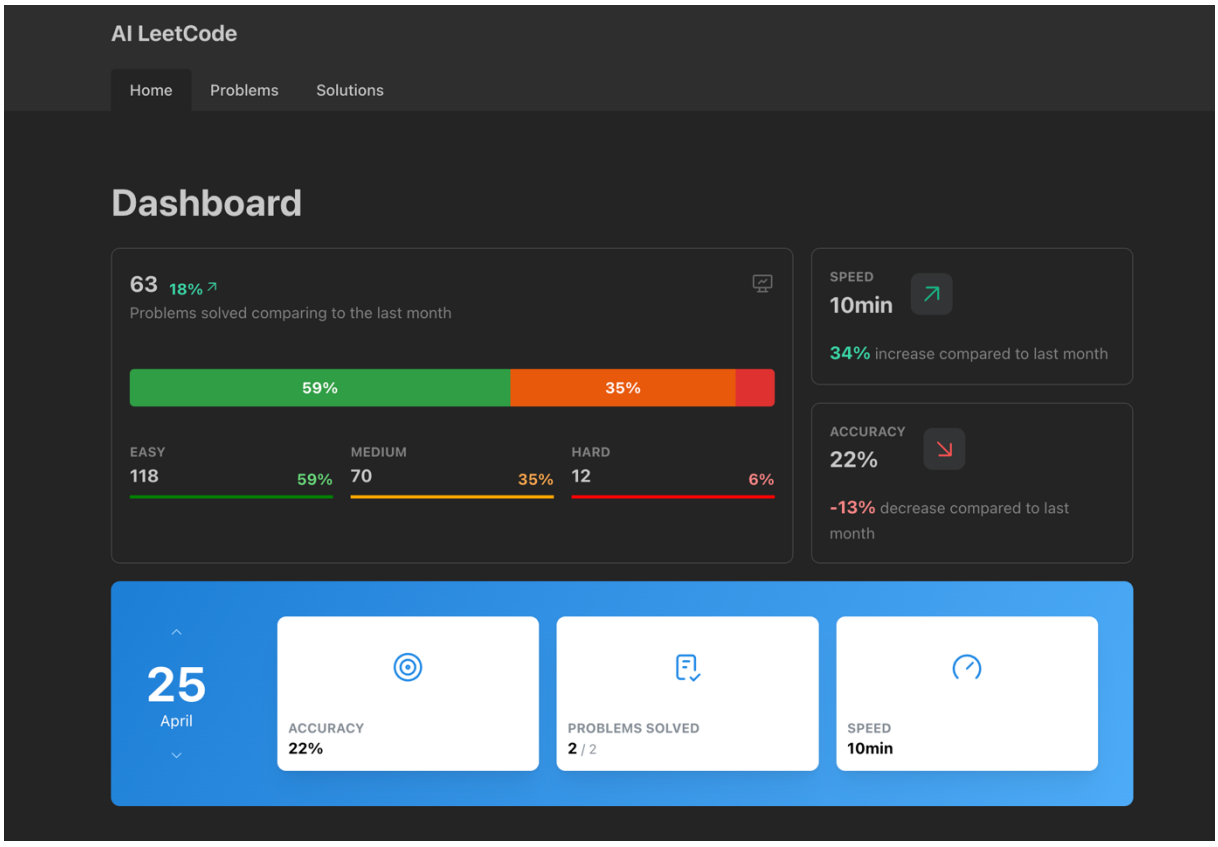


Рисунок 3.1 – Головна сторінка з інформаційною панеллю з основними показниками ефективності розв’язування задач

У верхній частині представлено загальну кількість вирішених задач, відсоткове співвідношення складності (easy, medium, hard), а також зміни за останній період. Окремо винесено блоки з даними про середній час розв’язання задачі та точність рішень. Нижній сегмент дозволяє переглянути аналітику за окремі дні, включаючи відсоток правильних відповідей, кількість задач та час виконання. Такий підхід сприяє формуванню звичок щоденної практики і дає змогу контролювати прогрес.

Інтерфейс інформаційної панелі поєднує у собі як загальні оглядові блоки, так і більш деталізовані показники. Це забезпечує баланс між швидким переглядом результатів і можливістю занурення у конкретику. Важливо, що всі елементи розміщені логічно послідовно, що відповідає принципам зручності користування та дозволяє оптимізувати навчальний процес у системі.

Розділ задач у вебдодатку є ключовим елементом, що забезпечує користувачеві доступ до попередньо згенерованих завдань. Цей інтерфейс дозволяє не лише переглядати наявні задачі, але й швидко аналізувати їх характеристики, зокрема тему, мову програмування та рівень складності. Інформативне представлення даних у табличній формі підвищує зручність користування, дозволяючи миттєво орієнтуватись у переліку завдань.

Далі можна побачити реалізований інтерфейс сторінки задач, де відображено таблицю з назвами завдань, їх тематикою, мовою реалізації та складністю (рисунок 3.2). Кожен рядок таблиці містить чітко структуровану інформацію, а колірне маркування тегів допомагає миттєво розпізнати рівень складності та тип мови програмування. Кнопка створення нової задачі винесена у праву частину екрану і позначена іконкою, що символізує застосування штучного інтелекту для генерації завдань. Такий підхід до організації інтерфейсу дозволяє швидко відфільтровувати задачі за потрібними критеріями і підтримує ефективну навігацію навіть за значної кількості завдань.

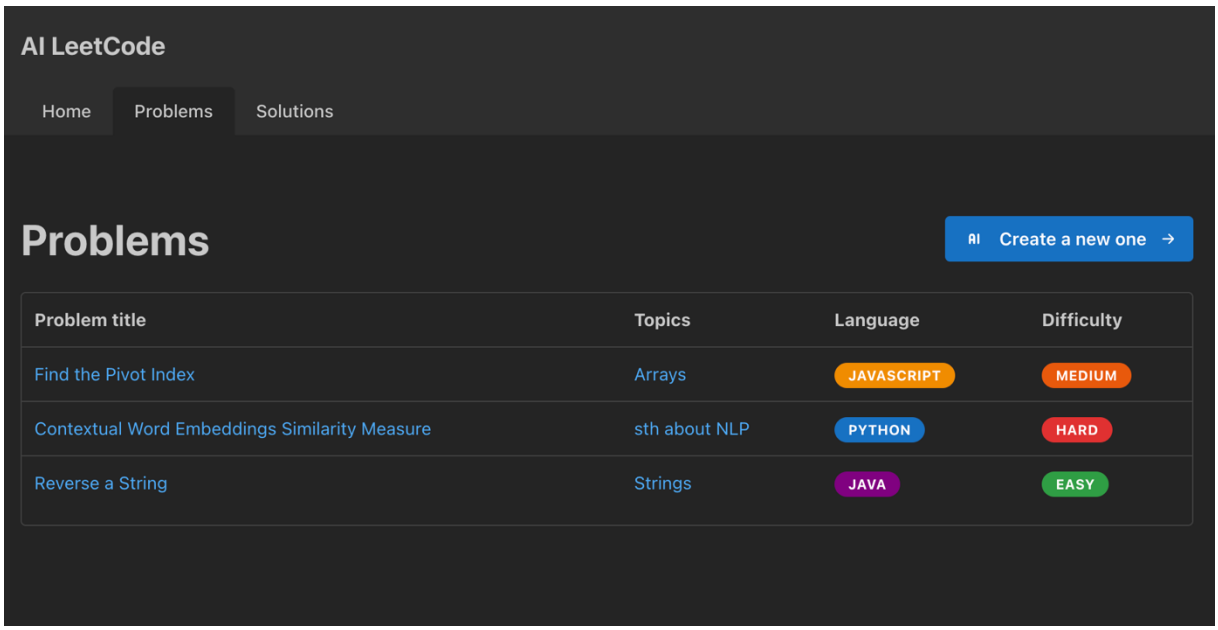


Рисунок 3.2 – Перелік задач з можливістю створення нової задачі

Інтерфейс створення нової задачі є важливим елементом, що забезпечує користувачеві можливість гнучко формулювати параметри бажаної задачі перед її генерацією за допомогою штучного інтелекту. Основна мета цього компонента – забезпечити інтуїтивне введення тематики, рівня складності, мови програмування та додаткових зауважень, що будуть передані на сервер для створення нової задачі.

Далі можна побачити вікно створення задачі, яке з’являється після натискання відповідної кнопки на сторінці задач (рисунок 3.3). У формі передбачено поля для введення теми, вибору складності та мови, а також поле для необов’язкових нотаток, що дозволяє користувачу уточнити бажані аспекти задачі. Колірна індикація вибраних значень (наприклад, червоний для складності hard, синій для мови Python) допомагає краще сприймати вибрані параметри. Таке рішення дозволяє не лише структурувати побажання користувача, але й забезпечує ефективну інтеграцію з системою генерації контенту.

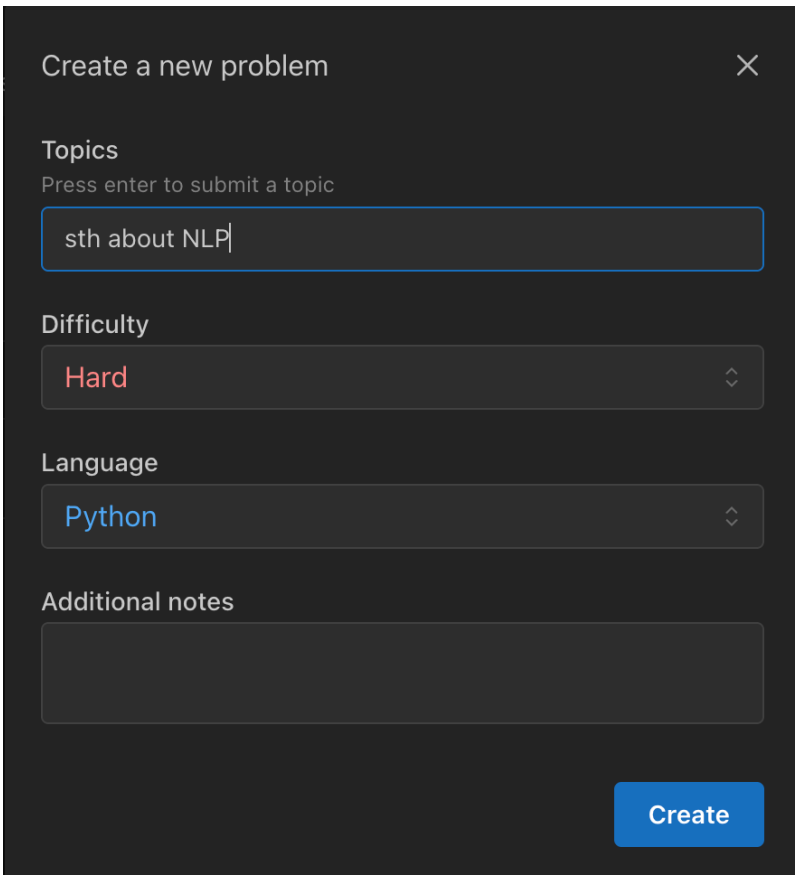


Рисунок 3.3 – Модальне вікно створення нової задачі

Формування чіткої та зрозумілої умови задачі є ключовим аспектом навчального процесу у будь-якій системі програмування. У контексті запропонованого вебдодатку формування умови задачі відбувається автоматично на основі параметрів, заданих користувачем, з використанням генеративної моделі штучного інтелекту. Зміст задачі має бути лаконічним, містити чіткі обмеження та супроводжуватися прикладами для кращого розуміння.

Далі можна побачити приклад згенерованої задачі на тему пошуку індексу повороту у масиві цілих чисел (рисунок 3.4). Умова задачі сформульована англійською мовою у стилі платформ типу LeetCode, що дозволяє користувачам одразу перейти до написання розв’язку. Приклади, наведені нижче умови, містять вхідні дані, очікуваний вихід та пояснення, які дають змогу краще зрозуміти логіку задачі навіть новачкові.

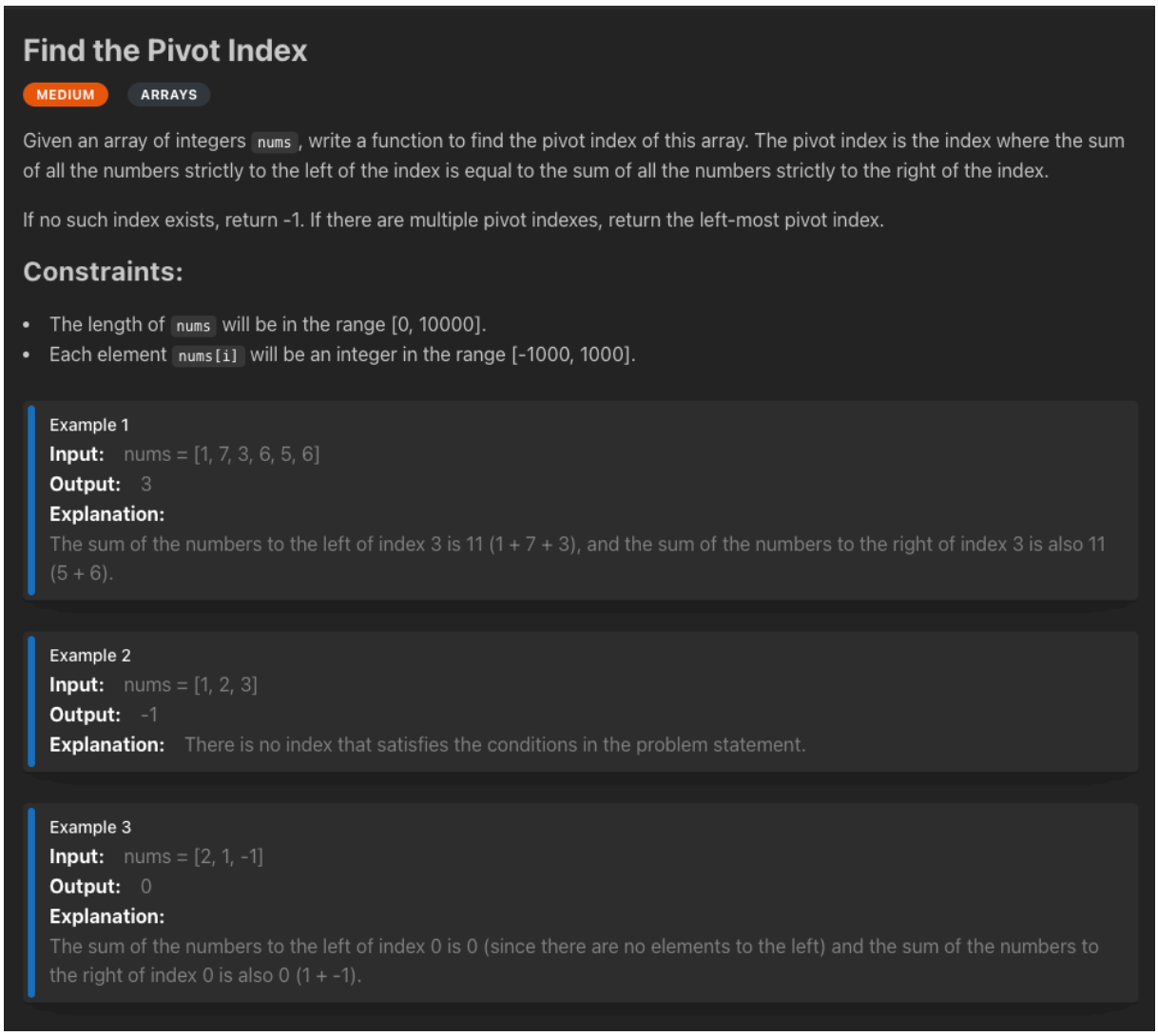


Рисунок 3.4 – Умова задачі з прикладами вхідних та вихідних даних

Обмеження, наведені в умові, зменшують ймовірність неправильного трактування задачі та дозволяють системі ефективно перевіряти коректність розв’язку. Такий підхід до формування задач значно пришвидшує процес створення нових навчальних матеріалів, зменшує залежність від викладачів або адміністраторів контенту та забезпечує персоналізацію досвіду користувача. Крім того, наявність реальних прикладів і чітко окреслених умов формує довіру до задачі, зменшуючи потребу в зовнішніх роз’ясненнях.

Написання розв’язку до задачі є одним із найважливіших етапів взаємодії користувача з вебдодатком. Саме на цьому етапі проявляється

здатність користувача застосовувати теоретичні знання на практиці. Система повинна не лише надати зручний редактор коду, але й підтримувати можливість миттєвого запуску перевірки рішення, його збереження та надання зворотного зв'язку. Таким чином, користувач отримує повний цикл роботи з задачами: від генерації до самоперевірки.

Далі можна побачити приклад реалізації функції пошуку індексу повороту у масиві, написаної мовою JavaScript (рисунок 3.5).

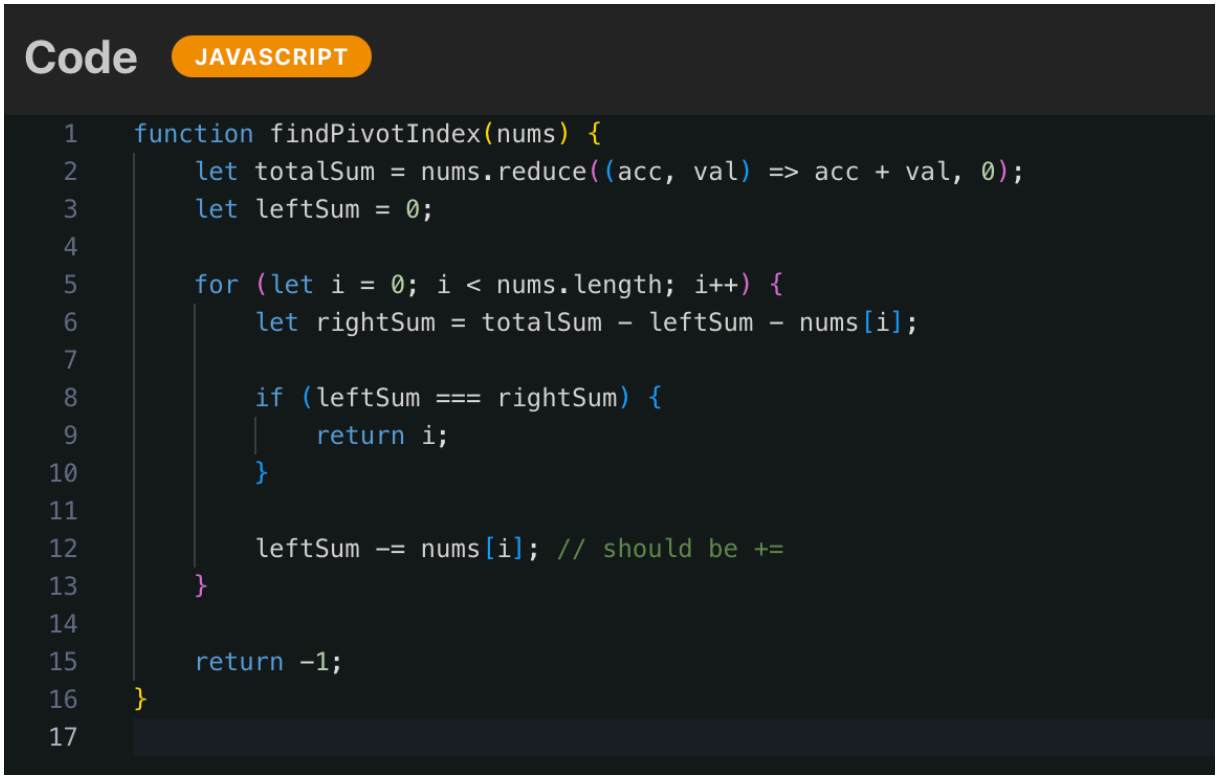


Рисунок 3.5 – Вікно написання коду для розв'язання задачі

Функція містить логіку обчислення сум лівої та правої частин масиву, а також умовну перевірку на рівність цих сум. Проте у прикладі присутня помилка – зміна змінної leftSum відбувається через віднімання замість додавання, що порушує логіку обчислення. Такий тип помилок часто виникає у початківців, і тому важливо, щоб система вміла їх виявляти й пояснювати (рисунок 3.6).

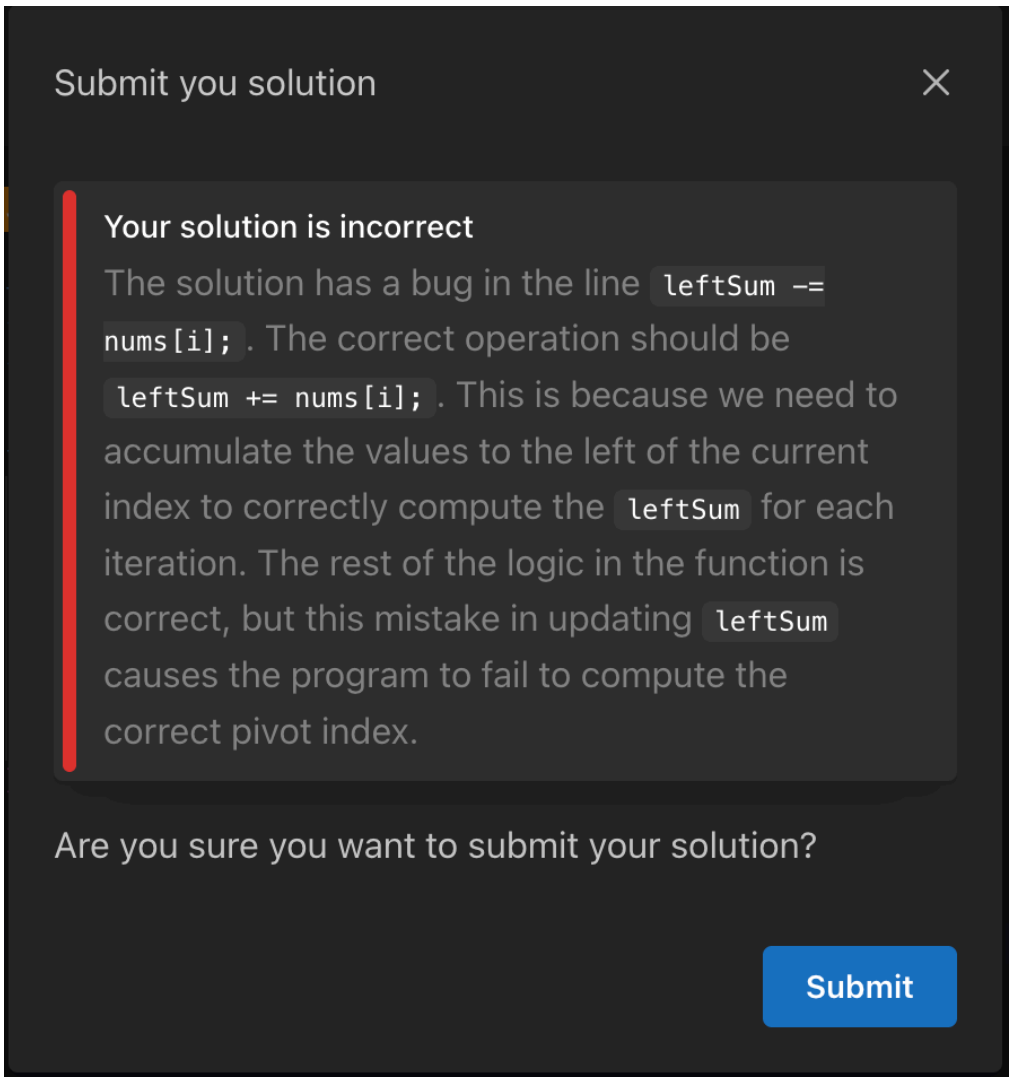


Рисунок 3.6 – Модальне вікно з результатом перевірки розв’язку

Для зручності користувача інтерфейс вебдодатку забезпечує інтерактивне середовище для ознайомлення із формулюванням задачі та реалізації її розв’язку безпосередньо в межах одного екрана. Завдяки цьому користувач може швидко зрозуміти умови задачі, розглянути приклади та відразу перейти до написання коду, не перемикаючись між окремими вкладками або сторінками.

Далі можна побачити інтерфейс задачі, що включає її опис, приклади, редактор коду та кнопку для надсилання розв’язку (рисунок 3.7). Така композиція створює цілісний навчальний простір, у якому кожен елемент виконує свою роль – опис забезпечує контекст, приклади формують

уявлення про очікувану поведінку рішення, а редактор дозволяє реалізувати розв’язок у реальному часі.

Це поєднання візуальної структурованості та функціональної гнучкості сприяє підвищенню якості навчального досвіду. Завдяки миттєвій доступності всіх елементів задачі користувач зосереджується виключно на процесі розв’язання, що стимулює розвиток алгоритмічного мислення та самостійного пошуку помилок.

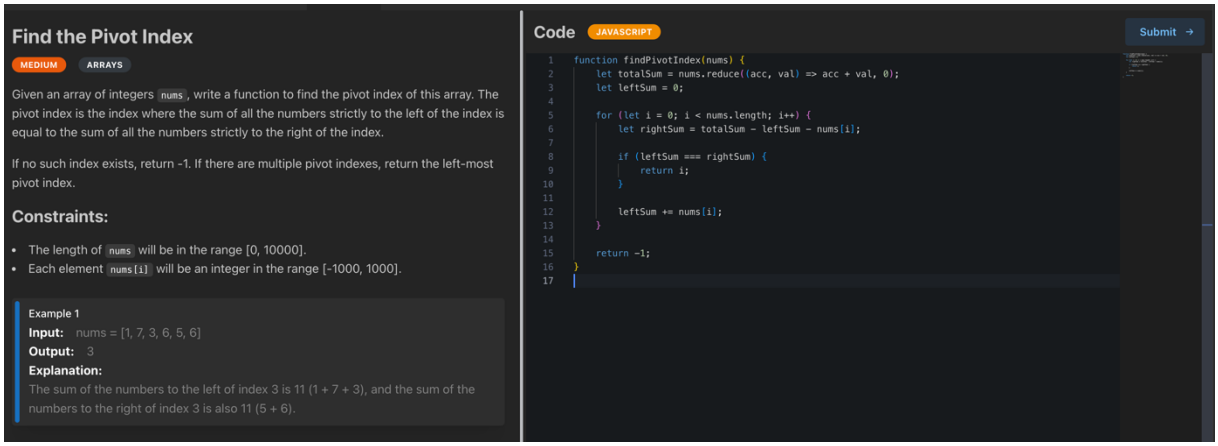


Рисунок 3.7 – Повноцінний інтерфейс сторінки задачі

Процес перевірки розв’язку передбачає обов’язкову взаємодію користувача з підтверджувальним інтерфейсом. Це дозволяє уникнути випадкового надсилання неперевіреного або незавершеного коду, особливо в умовах навчального процесу, де точність та зосередженість відіграють важливу роль.

Подібний підхід також сприяє формуванню усвідомленості дій користувача в освітньому середовищі. Далі можна побачити вікно підтвердження надсилання розв’язку (рисунок 3.8), яке викликається перед запуском механізму перевірки. У цьому вікні користувачеві пропонується переконатися у правильності власного коду та ще раз усвідомити відповідальність за результат.

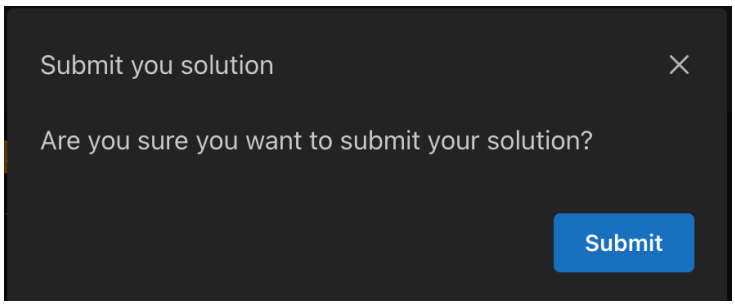


Рисунок 3.8 – Модальне вікно підтвердження здачі рішення

Далі можна побачити верхню частину сторінки, яка відображає розв'язок конкретної задачі, обраної користувачем (рисунок 3.9). У цьому інтерфейсі представлено заголовок задачі, її складність, тематику та формулювання умови, включаючи обмеження. Крім того, у правій частині розміщено графічний індикатор точності виконання, який демонструє відсоткове співвідношення правильних і неправильних відповідей.

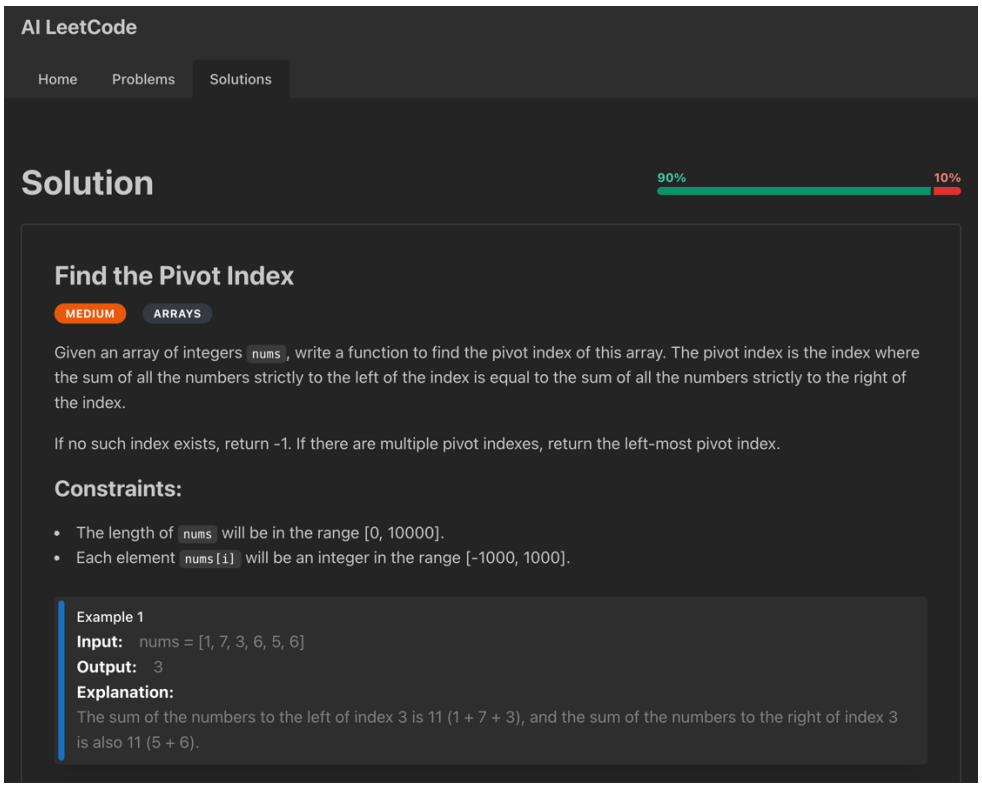


Рисунок 3.9 – Верхня частина сторінки конкретного рішення користувача

Далі можна побачити програмний код, що був поданий як розв’язок користувачем у межах відповідної задачі (рисунок 3.10). Блок коду розташований у нижній частині сторінки і дозволяє візуально оцінити правильність синтаксису, логіку реалізації та якість оформлення рішення. Використання підсвічування синтаксису значно полегшує аналіз і сприйняття коду.

Рисунок 3.10 – Нижня частина сторінки конкретного рішення користувача

Далі можна побачити перелік рішень користувача із зазначенням дати, складності задачі, мови програмування та точності виконання (рисунок 3.1). Такий формат відображення дозволяє швидко зорієнтуватися у власному прогресі, побачити, які задачі вже розв’язані, і перейти до детального перегляду будь-якого з рішень.

Окрім базової інформації, таблиця також містить кольорові індикатори точності – відсоткове співвідношення правильних і помилкових підходів. Це не тільки мотивує користувача підвищувати якість рішень, а й дозволяє аналізувати слабкі сторони за окремими аспектами.

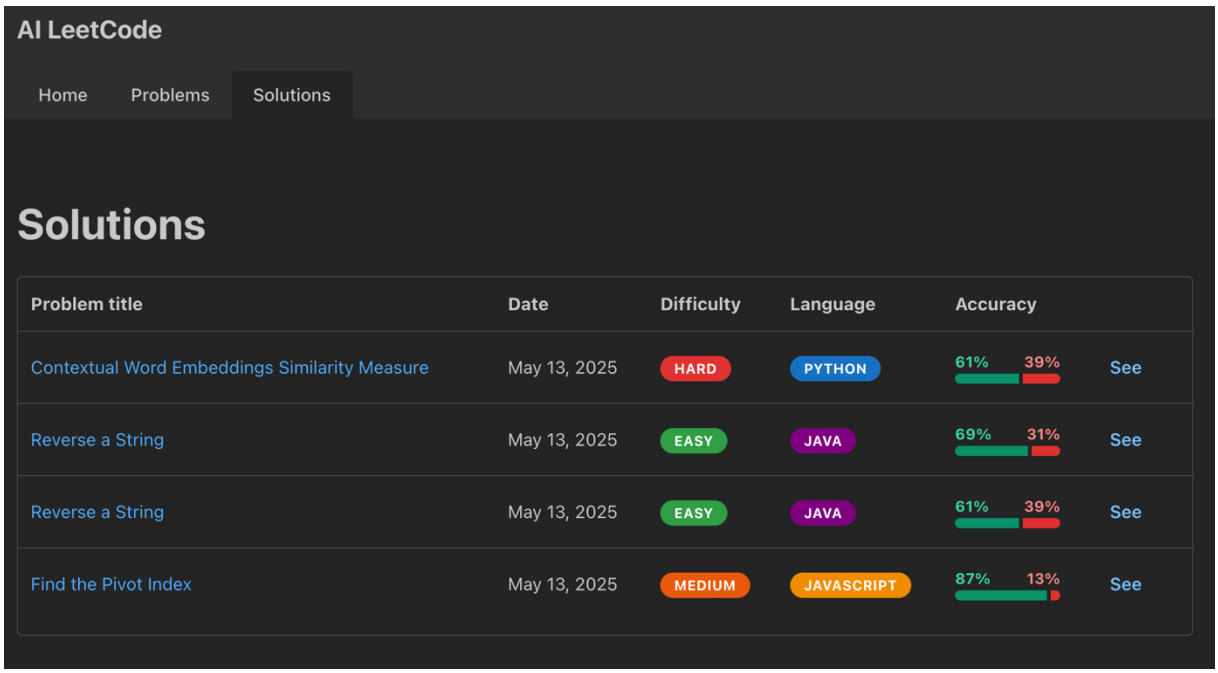


Рисунок 3.11 – Сторінка з результатами рішень користувача зі статистикою

Інтерфейс реалізованого вебдодатку виявився інтуїтивно зрозумілим, функціональним і візуально структурованим. Основна навігація здійснюється через верхнє меню, що дозволяє швидко переходити між головними сторінками – «Dashboard», «Problems» та «Solutions». Усі елементи інтерфейсу виконані в єдиному стилі з акцентом на темну тему оформлення, що підвищує зручність тривалої взаємодії з платформою. Використання кольорових позначень складності задач, мов програмування та точності рішень суттєво полегшує візуальне сприйняття інформації.

Користувацький досвід також покращується за рахунок використання модальних вікон для створення задач і надсилання рішень, що дозволяє уникнути зайвих переходів між сторінками. Форма генерації задачі містить усі необхідні поля з валідацією, а також зручний вибір складності й мови програмування. Відображення результатів виконаних рішень доповнене детальною інформацією та оцінкою точності, що забезпечує прозорість перевірки й надає користувачеві можливість вчитись на власних помилках.

ВИСНОВКИ

У процесі виконання кваліфікаційної роботи було реалізовано вебдодаток, що дозволяє користувачам створювати задачі з програмування, автоматично перевіряти запропоновані рішення за допомогою моделі штучного інтелекту OpenAI, а також вести персональну статистику результатів. Завдання кваліфікаційної роботи було виконано в повному обсязі відповідно до поставленої мети. Усі передбачені функціональні компоненти – генерація задач, надсилання та перевірка рішень, зберігання даних, а також інтерфейсна частина з аналітикою користувача – були реалізовані й інтегровані в єдину систему. У якості основи застосовано стек технологій TypeScript, React, Nest.js, Firebase Firestore та OpenAI API. Забезпечено підтримку авторизації, захисту маршрутизованих ресурсів, а також зручну взаємодію між клієнтською та серверною частинами.

З точки зору кількісних показників, система здатна обробляти десятки одночасних запитів до OpenAI API, зберігати структуровані дані користувачів, задач та рішень, а також відображати статистику у вигляді зведених індикаторів. Візуальні компоненти інтерфейсу адаптовані під потреби користувачів, реалізовано систему кольорової індикації складності задач і правильності рішень, що полегшує аналітичне сприйняття. Якісні показники полягають у високому рівні юзабіліті, мінімізації навантаження на користувача при створенні задач і відправленні рішень, а також у впровадженні зворотного зв’язку з поясненнями у випадку неправильних рішень, що підвищує навчальну ефективність.

У порівнянні з аналогічними платформами, такими як LeetCode, Codewars чи HackerRank, реалізоване рішення має важливу відмінність – автоматичне генерування задач і перевірка рішень відбуваються без участі людини, лише за допомогою великої мовної моделі. Таким чином, запропонований додаток не залежить від заздалегідь підготовленої бази задач і дозволяє динамічно адаптуватися під навчальні потреби

користувача. Це надає рішенню високу гнучкість і потенціал до масштабування. Хоча за функціональністю та глибиною тестування рішень система ще не дорівнює повноцінним платформам для змагань з програмування, у контексті індивідуального навчання вона демонструє переваги, особливо у частині адаптивності.

У подальшій роботі доцільно зосередитися на вдосконаленні механізмів оцінювання рішень, зокрема, впровадити тестування на граничних наборах вхідних даних, багатофайлову підтримку коду, а також зберігання історії змін рішень. Крім того, перспективним є розвиток системи персоналізованих рекомендацій, що базуватиметься на результатах попередніх рішень користувача, з урахуванням типових помилок і прогресу в складності задач. Важливим напрямком також є інтеграція з іншими мовами програмування, зокрема C++, Go або Rust, що дозволить залучити ширшу аудиторію користувачів. Для подальшої наукової апробації доцільним є проведення експериментального дослідження ефективності навчання користувачів на основі адаптивно сформованих задач порівняно з класичним статичним підходом. Таким чином, запропонована система має як практичну, так і дослідницьку цінність і може бути використана в освітніх цілях або в складі інтелектуальних навчальних платформ.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Ahmad A., Li F. Personalized learning paths in e-learning systems using machine learning techniques. *International Journal of Emerging Technologies in Learning*. 2022. Vol. 17, no. 5. P. 36–52. URL: https://doi.org/10.3991/ijet.v17i05.29419.

2. Aljanabi S. A., Junaid A. A. Online learning platform for programming education: A review of current features and limitations. *Education and Information Technologies*. 2021. Vol. 26. P. 7483–7508. URL: https://doi.org/10.1007/s10639-021-10591-0.

3. Auth0. Auth0 Docs: Identity for modern applications. URL: https://auth0.com/docs (date of access: 01.02.2025).

4. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding / J. Devlin et al. *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics*. 2019. P. 4171–4186. URL: https://doi.org/10.48550/arXiv.1810.04805.

5. Chen X., Xie H., Hwang G. J. A multi-perspective review of learning analytics in programming education. *Computers \& Education*. 2020. Vol. 151. P. 103861. URL: https://doi.org/10.1016/j.compedu.2020.103861.

6. Firebase Documentation. Cloud Firestore Overview. URL: https://firebase.google.com/docs/firestore (date of access: 02.02.2025).

7. Kornev I., Karpov M. Review of popular online platforms for learning programming: LeetCode, HackerRank, Codeforces and others. *CEUR Workshop Proceedings*. 2021. P. 184–189. URL: http://ceur-ws.org/Vol-2930/paper28.pdf.

8. Mantine. Mantine UI – React components library. URL: https://mantine.dev/ (date of access: 10.03.2025).

9. Monaco Editor. Monaco Editor – The code editor that powers VS Code. URL: https://microsoft.github.io/monaco-editor/ (date of access: 10.03.2025).

10. NestJS. NestJS – A progressive Node.js framework for building efficient server-side applications. URL: https://docs.nestjs.com/ (date of access: 15.03.2025).

11. OpenAI. GPT-4 Technical Report. URL: https://openai.com/research/gpt-4 (date of access: 15.03.2025).

12. ReactJS. React – A JavaScript library for building user interfaces. URL: https://reactjs.org/ (date of access: 20.03.2025).

13. Wambugu D. M., Waweru M. W. Improving computer programming education using generative AI: Potentials and pitfalls of OpenAI's Codex. *Journal of Computer Languages*. 2022. Vol. 70. P. 101103. URL: https://doi.org/10.1016/j.cola.2022.101103.

14. Yu L., Wang H., Zhao Y. Review and analysis of intelligent tutoring systems based on artificial intelligence. *Journal of Educational Computing Research*. 2020. Vol. 58, no. 2. P. 316–341. URL: https://doi.org/10.1177/0735633119845693.

15. Yusuf M. R., Halim M. F. A. Automatic code generation using GPT-3 for introductory programming problems. *Procedia Computer Science*. 2023. Vol. 217. P. 1083–1090. URL: https://doi.org/10.1016/j.procs.2023.03.155.

16. Intelligent tutoring systems and learning outcomes: A meta-analysis / W. Ma et al. *Journal of Educational Psychology*. 2014. Vol. 106, no. 4. P. 901–918. URL: https://doi.org/10.1037/a0037123.

17. Nguyen T., Gardner L., Sheridan D. A review of API design guidelines and usability studies. *Journal of Systems and Software*. 2020. Vol. 170. P. 110749. URL: https://doi.org/10.1016/j.jss.2020.110749.

18. Papamitsiou Z., Economides A. A. Learning analytics and educational data mining in practice: A systematic literature review of empirical evidence. *Educational Technology \& Society*. 2014. Vol. 17, no. 4. P. 49–64. URL: https://www.jstor.org/stable/jeductechsoci.17.4.49.

19. Sarsa J., Escudero D., Bravo C. Code quality in educational software projects: A review of the literature. *IEEE Transactions on Education*. 2023. Vol. 66, no. 1. P. 12–22. URL: https://doi.org/10.1109/TE.2022.3141972.

20. Systematic review of research on artificial intelligence applications in higher education – where are the educators? / O. Zawacki-Richter et al. *International Journal of Educational Technology in Higher Education*. 2019. Vol. 16, no. 39. URL: https://doi.org/10.1186/s41239-019-0171-0.

21. An approach to the selection of behavior patterns autonomous intelligent mobile systems / O. Zolotukhin et al. *Proceedings of the IEEE International Conference on Problems of Infocommunications Science and Technology (PIC S\&T)*. Kyiv, 2021. P. 349–352. URL: https://doi.org/10.1109/PICST54195.2021.9772110.

22. Filatov V., Semenets V., Zolotukhin O. Synthesis of semantic model of subject area at integration of relational databases. *Proceedings of the IEEE 8th International Conference on Advanced Optoelectronics and Lasers (CAOL)*. 2019. URL: https://doi.org/10.1109/CAOL46282.2019.9019532.

23. Methods of intellectual analysis of processes in medical information systems / V. Filatov et al. *Information Extraction and Processing*. 2020. Vol. 48, no. 124. P. 92–98. URL: https://doi.org/10.15407/vidbir2020.48.092.

24. The methods for the prediction of climate control indicators in the Internet of Things systems / O. Zolotukhin et al. *CEUR Workshop Proceedings*. 2021. URL: https://doi.org/10.5281/zenodo.14526027.