МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
ІМЕНІ ТАРАСА ШЕВЧЕНКА
Факультет інформаційних технологій
Кафедра інтелектуальних технологій

ВИПУСКНА КВАЛІФІКАЦІЙНА РОБОТА БАКАЛАВРА

НА ТЕМУ

Веб-застосунок “тренувальний майданчик” для вирішення задач з програмування

Галузь знань **12 «Інформаційні технології»**

Спеціальність **122 «Комп’ютерні науки»**

Освітня програма **«Комп’ютерні науки»**

Освітній рівень: бакалавр

Виконав: студент 4 курсу, групи КН- 42

Бойчук В.С. 

Керівник к.т.н., доцент Доманецька І.М.

Випускна кваліфікаційна робота бакалавра допущена до захисту рішенням кафедри інтелектуальних технологій

Протокол № 13 від 05.06.2023 р.

зав. кафедри _____ доц. Іларіонов О.Є.

Київ – 2023

КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ІМЕНІ ТАРАСА ШЕВЧЕНКА

Факультет інформаційних технологій

Кафедра інтелектуальних технологій

Спеціальність 122 «Комп'ютерні науки»

ЗАТВЕРДЖУЮ
Завідувач кафедри
інтелектуальних технологій
Іларіонов О.Є.

_____
«___» _____ 2023 р.

ЗАВДАННЯ

НА ВИПУСКНУ КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТОВІ

Бойчуку Віталію Сергійовичу

1. Тема проекту (роботи)

«Веб-застосунок "тренувальний майданчик" для вирішення задач з програмування»

затверджена протоколом засідання кафедри від « 11 » листопада 2022 р. №4

2. Термін здачі студентом закінченого проекту (роботи) 29 травня 2023 року.
3. Вихідні дані до проекту (роботи)

   Розробити застосунок для вирішення задач з програмування.

4. Зміст розрахунково-пояснювальної записки (перелік питань, що їх належить розробити)

   1. Аналіз особливостей задачі 2. Проектування програмного забезпечення 3. Розробка програмного забезпечення застосунку

5. Перелік презентаційного матеріалу (з точним зазначенням обов'язкових презентацій)

   1. Об'єкт та предмет дослідження, мета роботи (1 слайд)
   2. Актуальність розробки застосунку та аналіз наявних систем (1 слайд).
   3. Проектні рішення з реалізації веб-застосунку (9 слайдів)
   4. Програмна реалізація веб-застосунку (8 слайдів).
   5. Висновки (1 слайд).

6. Консультанти з випускної кваліфікаційної роботи із зазначенням розділів випускної кваліфікаційної роботи, що їх стосуються

Розділ	Консультант	Підпис, дата	
		Завдання видав	Завдання прийняв

Дата видачі завдання 15 лютого 2023 року

Керівник / Доманецька І.М. /

Завдання прийняв до виконання / Бойчук В.С. /

КАЛЕНДАРНИЙ ПЛАН

Пор. №	Назва етапів випускної кваліфікаційної роботи	Термін виконання етапів випускної кваліфікаційної роботи	Примітка
1	Опрацювання літератури	15.02 – 27.02	
2	Робота над розділом 1. Аналіз особливостей функціонування застосунків - навчальних майданчиків	27.02 – 08.03	
3	Робота над розділом 2. Проектування програмного забезпечення	08.03 – 08.04	
4	Робота над розділом 3. Розробка застосунку	08.04 – 13.05	
5	Робота над оформленням пояснювальної записк	13.05 – 25.05	
6	Робота над презентацією	25.05 – 29.05	

Студент-дипломник / Бойчук В.С. /

Керівник випускної кваліфікаційної роботи / Доманецька І.М. /

АНОТАЦІЯ

Дипломна робота викладена на 67 сторінках, містить 3 розділи, 43 ілюстрації, 12 таблиць, 13 джерел в переліку посилань.

Об’єктом дослідження роботи є процес створення навчального веб-застосунку, а саме тренувального майданчику для вирішення задач з програмування.

Предметом дослідження є методи, технології та програмні рішення зі створення веб-застосунку для вирішення задач з програмування.

Метою роботи є розробка веб-застосунку тренувального майданчика для вирішення задач з програмування, що забезпечує підвищення ефективності процесу засвоєння знань.

У першому розділі проведено аналіз предметної області навчальних платформ, проаналізовано існуючі рішення на ринку, обґрунтовано актуальність вирішуваної задачі. Визначені функціональні та нефункціональні вимоги до застосунку. У другому розділі спроектовано функціональну структуру застосунку, визначені архітектурні рішення, розроблено структуру бази даних для системи та запропоновані інтерфейсні рішення застосунку. У третьому розділі наведено інформацію про використані програмні засоби у дипломному проекті, розроблено план тестування та перевірено основні функції готового застосунку.

Ключові слова: програмування, навчання, веб-застосунок, навчальні платформи, база даних.

ANNOTATION

The thesis is presented on 67 pages, contains 3 chapters, 43 illustrations, 12 tables, 13 sources in the list of references.

The object of research is the process of creating an educational web application, namely a “training area” for solving programming problems.

The subject of research is methods, technologies and software solutions for creating a web application for solving programming problems.

The aim of the work is to develop a web application “training area” for solving programming problems, which provides an increase in the efficiency of the process of learning.

In the first section, an analysis of the subject area of the educational platform was carried out, existing solutions on the market were analyzed, and the relevance of the problem to be solved was substantiated. Functional and non-functional application requirements are defined. In the second section, the functional structure of the application is designed, the architectural solutions are defined, the database structure for the system is developed, and the interface solutions of the application are proposed. The third section provides information about the software tools used in the diploma project, developed a test plan, and tested the main functions of the ready-made application.

Keywords: programming, learning, web application, educational platforms, database.

ЗМІСТ

ВСТУП

Сфера інформаційних технологій почала свій розвиток у 1960-х роках, паралельно з появою перших інформаційних систем. З того часу, вона стрімко розвинулася і почала проникати у все більше й більше галузей діяльності людини. Попит на фахівців у сфері інформаційних технологій з кожним роком значно зростає, що породжує появу різних інструментів, які допомагали б навчатися програмуванню новачкам, а також покращувати свої вміння вже і досвідченим спеціалістам. Звичайно, наявність гарних знань у певній мові програмування та вміння швидко й ефективно вирішувати задачі є чи не однією з найбільш важливих рис затребуваного на ринку праці спеціаліста. Тому, створення зручного застосунку, який міг би допомагати людям у даній сфері розвивати на практиці вищезгадані вміння є доволі актуальною потребою.

Метою роботи є розробка веб-застосунку тренувального майданчика для вирішення задач з програмування, що забезпечує підвищення ефективності процесу засвоєння знань.

Об’єктом дослідження в даній дипломній роботі виступає процес створення навчального веб-застосунку, а саме тренувального майданчику для вирішення задач з програмування. Предметом дослідження виступають методи, технології та програмні рішення зі створення веб-застосунку для вирішення задач з програмування.

Завдання дослідження: проаналізувати проблему, що буде вирішуватися в рамках дипломної роботи; визначити бізнес архітектуру предметної області; порівняти існуючі підходи, щодо вирішення задачі; провести функціональний аналіз розроблюваної системи; обгрунтувати вибір інструментів для реалізації застосунку; описати архітектуру системи; описати та проаналізувати тестові приклади роботи застосунку.

РОЗДІЛ 1. АНАЛІЗ ОСОБЛИВОСТЕЙ ФУНКЦІОНУВАННЯ ЗАСТОСУНКІВ - НАВЧАЛЬНИХ МАЙДАНЧИКІВ

1.1 Застосунки - тренувальні майданчики, їх призначення та затребуваність

Стрімкий ріст популярності інформаційних технологій призвів до росту потреби у кваліфікованих спеціалістах у даній галузі. Кожен рік у світ програмування занурюється все більше і більше людей, їхня кількість обраховується сотнями тисяч, ця тенденція провокує появу різноманітних навчальних платформ, де ці люди зможуть розвинути потрібні їм навички у програмуванні та бути якісними фахівцями, не забуваємо й про ті мільйони програмістів, які прагнуть удосконалювати наявні у них вміння. Саме через це, актуальним є поява все нових та кращих застосунків, які сприятимуть, як і новачкам так і досвідченим спеціалістам у досягненні їхніх цілей. Також, як ми знаємо, не завжди люди мають зручний та повноцінний доступ до інструментів програмування на власному комп’ютері, тому важливо, щоб додатки, які покликані допомагати програмістам у їхньому розвитку, володіли хоча б задовільною частиною цих інструментів та надавали їх користувачам прямо із коробки. І останнє, що варто зазначити, так це те, що користувачам різних навчальних платформ повинно бути зручно знаходити та підбирати під себе задачі, які вони хочуть вирішувати, а також отримувати зворотній зв’язок від системи про правильність їхнього рішення і, звичайно, в сучасних реаліях цей процес мусить бути автоматизований, аби допомогти якомога більшій кількості людей.

Сутність задачі, яка розв’язується у даній дипломній роботі, полягає в тому, аби створити веб-застосунок тренувальний майданчик, який буде надавати користувачам різноманітні задачі та інструменти для їх вирішення. Ці задачі будуть поділені по рівню складності та категоріям, загалом існуватиме три рівня складності: легкий, середній та високий. Дане розбиття за рівнями складності дозволить людям краще розвиватися, починаючи свій шлях від вирішення легких задач і поступово переходячи до все більш складних. Задачі за

категоріями в свою чергу будуть поділені за наступними темами: сортування, динамічне програмування, масиви, рядки, графи, жадібні алгоритми, знаходження найкоротшого шляху. Для вирішення даних задач, існуватиме можливість вибору однієї з декількох існуючих, найбільш популярних мов програмування, для того, щоб користувач мав змогу вирішити завдання бажаною мовою чи декількома.

Без сумнівів дана робота є не аби як актуальною та затребуваною в наш час, коли так багато людей планують вивчати програмування або ставати кращими в ньому, тому вони постійно знаходяться в пошуках різноманітних онлайн додатків, які зможуть їм в цьому допомогти якнайкраще.

1.2 Аналіз існуючих програмних систем

На сьогодні існує декілька навчальних платформ, які призначені для розв’язку задачі, що вирішується в даній дипломній роботі. Вони надають змогу користувачам розвиватися у вирішенні задач з програмування, приймати участь в змаганнях з нього, обмінюватися одне з одним власними рішеннями задач та обговорювати їх, слідкувати за своїм прогресом, а також допомагають організаціям знайти талановитих програмістів. Окрім того, ці платформи часто пропонують змагання та конкурси, що стимулюють користувачів до досягнення нових вершин у програмуванні. Участь в таких змаганнях дає змогу не лише продемонструвати свої навички, але й отримати визнання від спільноти програмістів та потенційних роботодавців. Вони збирають широкий спектр задач з різних галузей програмування, що дозволяє користувачам поглиблювати свої знання із різних областей та експериментувати з новими концепціями.

Рисунок 1.1 Існуючі навчальні платформи

Ми розглянемо та проведемо порівняння двох найбільш популярних серед користувачів платформ, а саме “LeetCode” та “Codewars”.

“LeetCode” - це платформа, чия спільнота налічує сотні тисяч людей зі всього світу, вона допомагає покращити навички написання коду та підготуватися до інтерв’ю у багатьох технічних компаніях, налічуючи більше двох тисяч запитань, котрі можна опрацювати для цього. Також, на цій платформі часто проходять змагання, змагаючись в яких та досягаючи результатів можна отримати різні нагороди. Щоб отримати доступ до повноцінного контенту даного ресурсу, вам потрібно зареєструватися, після чого ви зможете вивчати різні матеріали на сайті, вирішувати задачі та спілкуватися з іншими користувачами. Вирішення задач, а також їх відладка та перевірка відбуваються у спеціально відведеному майданчику.

“Codewars” - платформа, що багато в чому схожа на “Leetcode”. Вона пропонує вирішити різні програмні головоломки, відомі як “ката”. Кожна з цих головоломок допомагає користувачу покращувати певне вміння, а також вони різняться за рівнями складності, тобто знайдуться завдання як для новачків, так

і для досвідчених людей, щоб приступити до їх вирішення вам також доведеться зареєструватися на сайті.

Основні відмінності між цими двома платформами полягають у тому, що “LeetCode” окрім вирішення прикладних задач, також робить ставку на допомогу користувачам у проходженні технічних співбесід у компанії, надаючи для цього різні питання, які часто зустрічаються на даних співбесідах, “Codewars” в свою чергу більше орієнтований на вирішення задач, змагання з іншими користувачами. Також на даній платформі ви можете бачити свій прогрес по відношенню до інших користувачів, в той час коли на “LeetCode” ви бачите лише свої результати. Окрім цього, “Codewars” містить блок, у якому ви можете переглянути наявні вакансії від певних компаній.

Політика оплати цих двох платформ схожа між собою, вони є базово безкоштовними, але мають додаткові платні версії, що надають додаткові можливості користувачам. Наприклад, преміальна версія “LeetCode” має такі функції, як відео-рішення, вибрані запитання від компаній та преміальні рішення. “Codewars” має преміальну версію, відому як “Codewars Red”, яка надає користувачам розширену статистику, щоб поділитися своїми навчальними здобутками, оцінкою рішень та детальною статистикою з іншими користувачами. Також ви зможете порівняти себе з іншими користувачами і побачити, як інші вирішили такі ж завдання. Преміум-версія дозволяє використовувати “Codewars” без реклами, мати ранній доступ до бета-функцій і отримувати трансляцію перевірки рішення завдання у режимі реального часу. Крім того, ви отримаєте значок «Codewars Red».

Приклад вигляду платформи “LeetCode” проілюстровано нижче (рис. 1.2, 1.3).

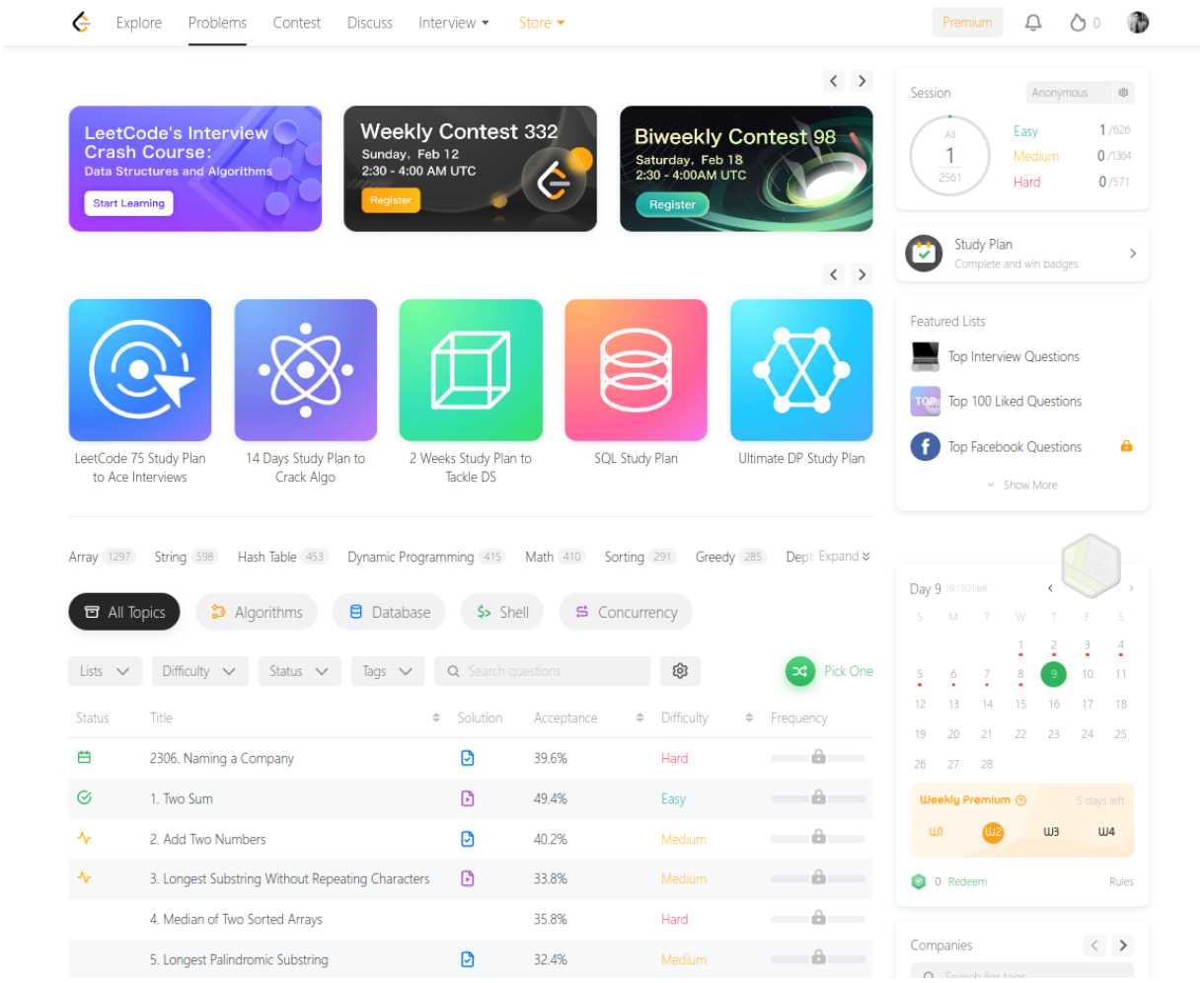

Рисунок 1.2 Сторінка з задачами платформи “LeetCode”

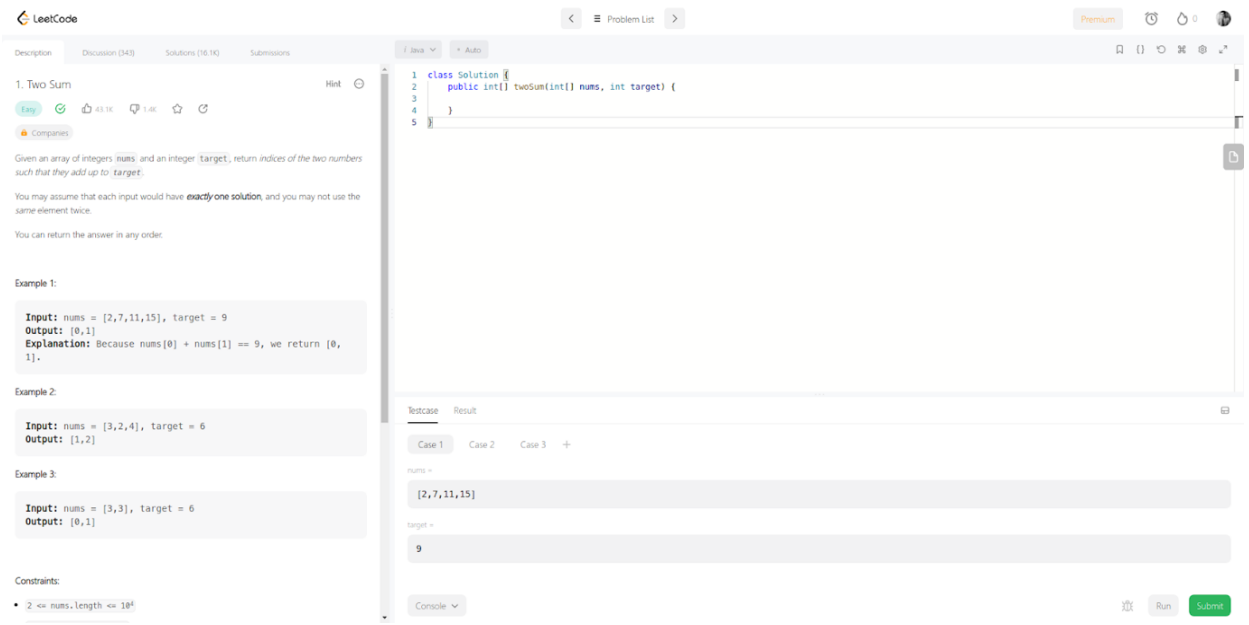

Рисунок 1.3 Сторінка вирішення обраної задачі на платформі “LeetCode”

Приклад вигляду платформи “Codewars” проілюстровано далі (рис. 1.4, 1.5).

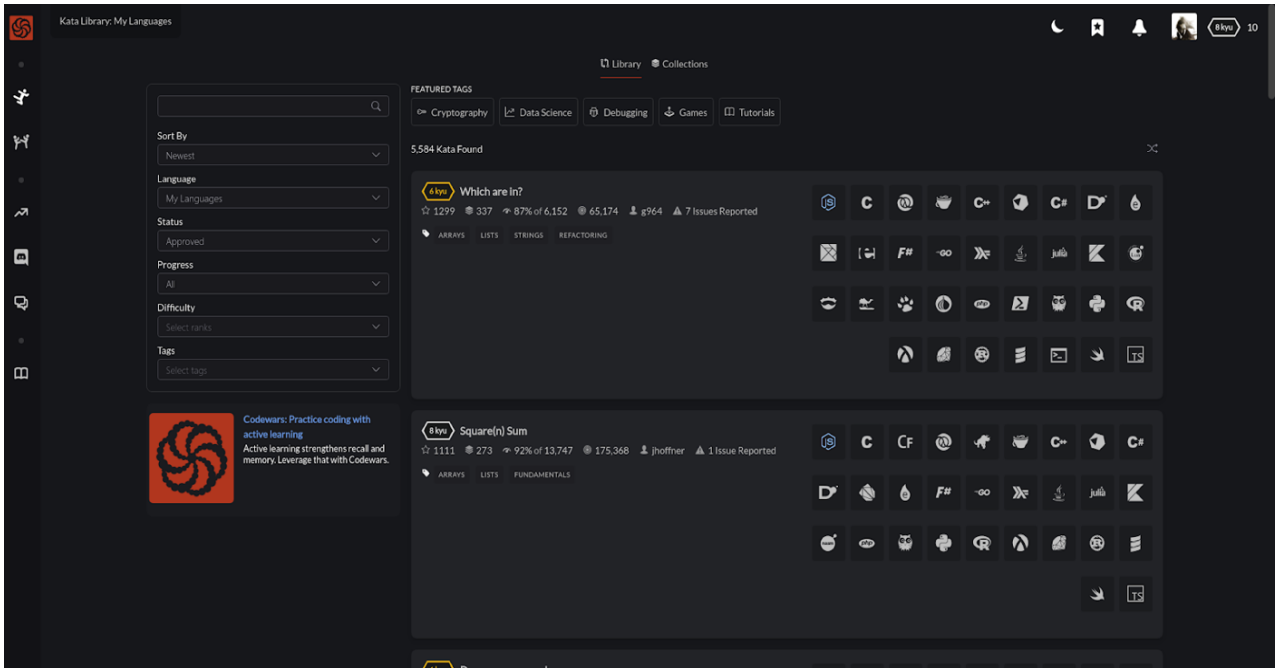

Рисунок 1.4 Сторінка з задачами платформи “Codewars”

З огляду даних ілюстрацій одразу видно, що дані платформи мають як багато схожостей так і відмінностей, які будуть детально проаналізовано далі у роботі.

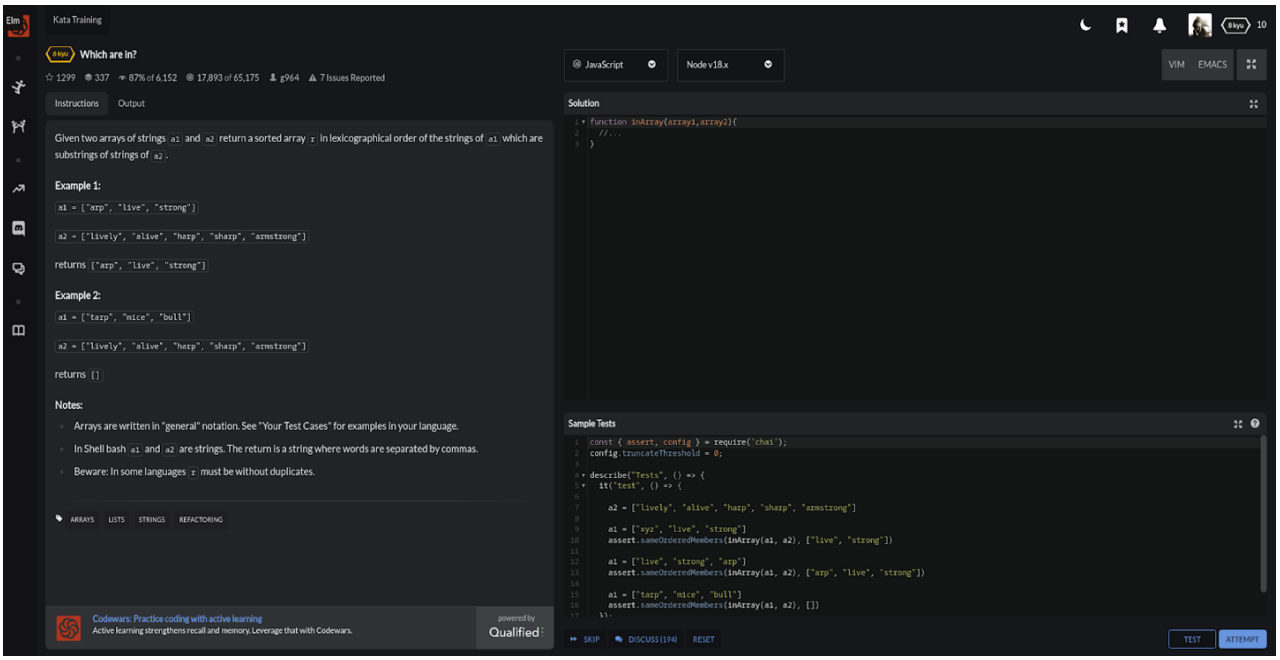

Рисунок 1.5 Сторінка вирішення обраної задачі на платформі “Codewars”

Дані програмні системи ми будемо оцінювати за наступними критеріями:

- наявність потрібного функціоналу;
- зручність та зрозумілість у використанні;

За відповідність до заданих критеріїв будемо оцінювати вибрані платформи за п’ятибальною шкалою. За наявність потрібного функціоналу ставитимемо оцінку 5, а зручність та зрозумілість у використанні даного функціоналу будемо оцінювати з власної точки зору і давати оцінку від 1 до 5. Якщо якийсь функціонал не є присутнім у тій чи іншій системі, оцінку зручності не вказуватимемо.

Визначимо функціонал, яким мають володіти дані програмні системи:

1. наявність широкого списку задач, з яких користувач обирає для вирішення собі потрібну;
2. можливість для користувача відфільтрувати задачі, щоб бачити тільки ті, які підходять під його критерії;
3. можливість пошуку конкретної задачі за іменем;
4. можливість вибору різних мов програмування, для вирішення обраного завдання;
5. наявність вбудованого онлайн редактора для написання коду, під задачу, яка вирішується користувачем;
6. наявність панелі для відображення результатів роботи, написаної користувачем програми;
7. наявність підказок у вирішенні задачі для користувача;
8. наявність готових рішень, якими користувач може скористатися в разі виникнення необхідності;
9. наявність панелі, для спілкування з іншими користувачами даної платформи;
10. можливість відслідковування власного прогресу у відповідному розділі додатку.

Таблиця 1.1 - Порівняльний аналіз існуючих програмних систем “LeetCode” та “Codewars”

№	Опис	LeetCode		Codewars	
		Н	З	Н	З
1	Наявний зрозумілий та широкий список задач з додатковими даними.	5	5	5	5
2	Можливо відфільтрувати завдання за багатьма критеріями.	5	5	5	5
3	Можливо знайти задачу за іменем.	5	5	5	3
4	Великий вибір мов програмування.	5	5	5	4
5	Наявний редактор коду з потрібним функціоналом.	5	5	5	4
6	Наявна панель для відображення результатів.	5	5	5	5
7	Наявна можливість скористатися підказкою.	5	-	0	-
8	Є можливість подивитися готове рішення задачі.	5	-	0	-

9	Наявна можливість спілкування з спільнотою платформи.	5	5	5	4
10	Наявна можливість слідкування за власним прогресом.	5	5	5	4

Позначення, що використані у таблиці 1.1:

- Н - наявність функціоналу у даній програмній системі;
- З - зручність та зрозумілість у використанні даного функціоналу.

Підбиваючи підсумки порівняння існуючих програмних систем за заданими критеріями, було визначено, що “Leetcode” отримав оцінку якості в 90 бали, а “Codewars” в 69 балів. Також, варто зазначити, що обидві системи, з суб’єктивної точки зору, працюють з однаковою та прийнятною швидкістю відклику. В цілому, обидві платформи доволі добре відповідають тим вимогам, котрим і повинні відповідати системи подібного роду.

1.3 Постановка задачі на створення веб-застосунку тренувального майданчика для вирішення задач з програмування

В рамках даної дипломної роботи, буде вирішуватися задача зі створення веб-додатку тренувального майданчика по вирішенню задач з програмування, з клієнтською та серверною частинами. Даний додаток даватиме змогу користувачам обирати різні задачі з програмування та вирішувати їх прямо на сайті, у вбудованому редакторі, також додаток буде перевіряти правильність їхнього рішення.

Метою випускної роботи є розробка веб-застосунку тренувального майданчика для вирішення задач з програмування задля підвищення ефективності навчання.

Об’єктом дослідження в даній дипломній роботі виступає процес створення навчального веб-застосунку, а саме тренувального майданчику для вирішення задач з програмування.

Предметом дослідження виступають методи, технології та програмні рішення по створенню веб-застосунку для вирішення задач з програмування.

Алгоритм розв’язку даної задачі:

1. Створити базу даних для зберігання необхідних даних про задачі, користувачів та все необхідне, що з ними пов’язане.
2. Реалізувати модуль по взаємодії з базою даних та потрібними сутностями, що в ній зберігаються.
3. Реалізувати модуль по авторизації та аутентифікації користувача у даній системі.
4. Реалізувати модуль, який буде використаний для запуску та перевірки коду користувача у ізольованому контексті операційної системи.
5. Створити модуль для роботи з рішенням задачі користувача.
6. Реалізувати клієнтську частину, з якою буде взаємодіяти користувач та яка буде взаємодіяти з серверною частиною додатку.

Наведемо узагальнений алгоритм роботи з платформою, через його представлення у вигляді схеми. (див. рис. 1.6)

Дана схема ілюструє узагальнений процес використання користувачем розроблюваного нами додатку, що складається з наступних кроків: спочатку користувач виконує реєстрацію на платформі, після чого в нього з’являється можливість переглядати наявні у додатку задачі, додатково при необхідності фільтруючи їх за певною категорією та рівнем складності або ж пошуком за іменем, вибравши потрібну задачу, користувач перенаправляється у спеціальний майданчик, де ознайомлюється в детальності із умовою завдання та приступає до його вирішення, написавши код-рішення до вибраної головоломки, він запускає процес перевірки свого завдання і чекає на його результат, через певний час система повідомляє користувача, чи перевірка пройшла успішно, якщо так, то завдання зараховується і користувач може переходити до

розв’язання наступних, в разі якщо ні, це може трапитися з двох причин, завдання було розв’язане не вірно або виникла помилка під час запуску наданого коду, то користувач продовжує вирішення даної задачі, допоки воно не пройде успішно перевірку системою та не буде зараховане.

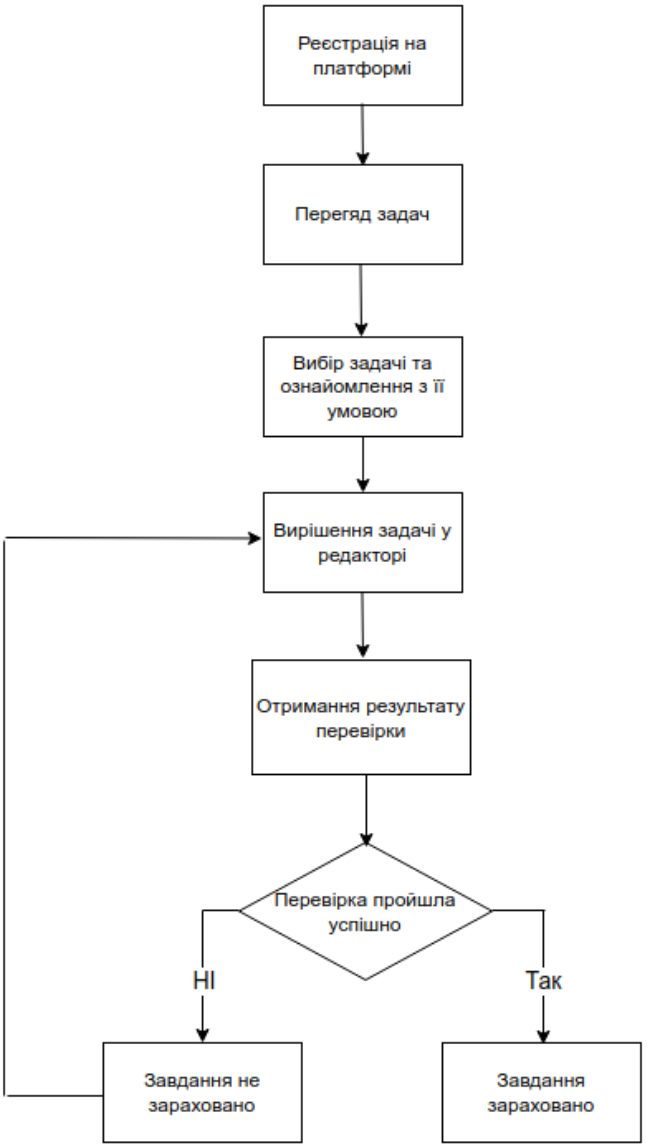

Рисунок 1.6 Узагальнений алгоритм роботи з платформою

Наведемо функціональні та нефункціональні вимоги до програмного забезпечення, з метою кращого розуміння усіх аспектів задачі, яку плануємо вирішувати.

Функціональні вимоги:

- Наявність механізму аутентифікації та авторизації для користувачів.

- Наявність панелі адміністратора, доступ до котрої матимуть лише адміністратори застосунку.
- Можливість перегляду наявних задач.
- Можливість вибору задачі зі списку для її вирішення.
- Наявність детального опису задачі.
- Наявність опису рішення задачі.
- Можливість написання коду по вирішенню задачі у вбудованому редакторі.
- Можливість для авторизованих користувачів залишати коментарі до задач, редагувати їх та видаляти.
- Можливість для авторизованих користувачів завантажувати картинки для свого аватару.
- Наявність функції по збереженню історії розв’язаних задач користувачем.
- У панелі адміністратора мусить бути функціонал по додаванню, видаленню та редагуванню задач.
- У панелі адміністратора мусить бути можливість переглядати інформацію про користувачів.
- Наявність процесу запуску та перевірки правильності рішення задачі.
- Наявність функціоналу для відображення користувачу результатів перевірки рішення задачі, що міститиме також і час виконання коду користувача.
- Можливість вирішення задач такими мовами програмування як: JavaScript, Python.

Нефункціональні вимоги:

- Зручне масштабування панелей на сторінці, що призначена для вирішення проблеми.
- Запуск коду користувача в ізольованому контексті, для забезпечення надійності та безпеки системи від потенційно шкідливого коду.

- Незалежність логіки по роботі з кодом-рішенням користувача, від мови програмування на якій це рішення було написане, тобто можливість зручно додавати підтримку рішень задач на різних мовах програмування до системи.
- Зберігання даних у реляційній базі даних.
- Написання серверної частини застосунку на Node.js.
- Написання клієнтської частини застосунку використовуючи інструменти, що підтримуватимуть рендеринг сторінок додатку на сервері.
- При перезавантаженні сторінки, з рішенням задачі, код, що написаний користувачем у вбудованому редакторі, повинен зберігатися.
- Коментарі користувачів до задачі повинні бути доступні на сторінці задачі в окремій панелі.
- Гама кольорів інтерфейсу додатка мусить бути не яскрава та приємна для очей.

Висновки до першого розділу

В результаті виконання першого розділу було проведено дослідження особливостей вирішуваної задачі, визначено мету та завдання даної роботи.

В рамках першого розділу був проведений аналіз проблеми, що вирішується та її актуальність в наш час, переглянуто та порівняно існуючі конкурентні рішення на ринку, виділені їх сильні та слабкі сторони, була здійснена постановка задачі та описано алгоритм її розв’язку, наведено узагальнений алгоритм роботи з платформою, а також проведено функціональний аналіз розроблюваної системи, виокремлені відповідні функціональні та нефункціональні вимоги до неї, що являють собою закладений в систему функціонал та визначають подальші критерії прийняття системи як завершеної по закінченню розробки на даному етапі.

РОЗДІЛ 2. ПРОЕКТНІ РІШЕННЯ З РЕАЛІЗАЦІЇ ВЕБ-ЗАСТОСУНКУ ТРЕНУВАЛЬНОГО МАЙДАНЧИКА ДЛЯ ВИРІШЕННЯ ЗАДАЧ З ПРОГРАМУВАННЯ

2.1 Аналіз бізнес процесів та бізнес архітектури предметної області

В ході проектування було виокремимо основні бізнес процеси у роботі системи навчального веб-застосунку та бізнес ролі, тих хто їх виконує:

- Управління задачами та користувачами:
    - робота з користувачами - адміністратор;
    - робота з задачами (видалення, додавання, перегляд, редагування).
- Використання застосунку:
    - аутентифікація та авторизація у системі - користувач;
    - редагування власного профілю - користувач;
    - перегляд профілю користувачів- користувач;
    - вирішення задач - користувач;
    - написання коментарів - користувач;
    - відновлення доступу до акаунту - користувач.

Як бачимо з опису бізнес процесів, що наведений вище, їх виконання покладене на дві ключові фігури, а саме на адміністратора даного застосунку, котрим може бути будь яка довірена особа та на користувача. На адміністратора покладені такі обов’язки як робота з користувачами, а саме перегляд основної інформації про них, а також управління задачами через наданий йому UI, він буде взаємодіяти з базою даних, де зберігаються дані задачі. Користувач у свою чергу проходить процес аутентифікації та авторизації у системі, має змогу редагувати власний обліковий запис, переглядати інформацію інших користувачів, вирішувати задачі на платформі, спілкуватися з іншими користувачами та у разі необхідності відновлювати доступ до акаунту.

Для кращого розуміння даних процесів, побудуємо бізнес архітектуру системи навчального веб-застосунку (рис. 2.1).

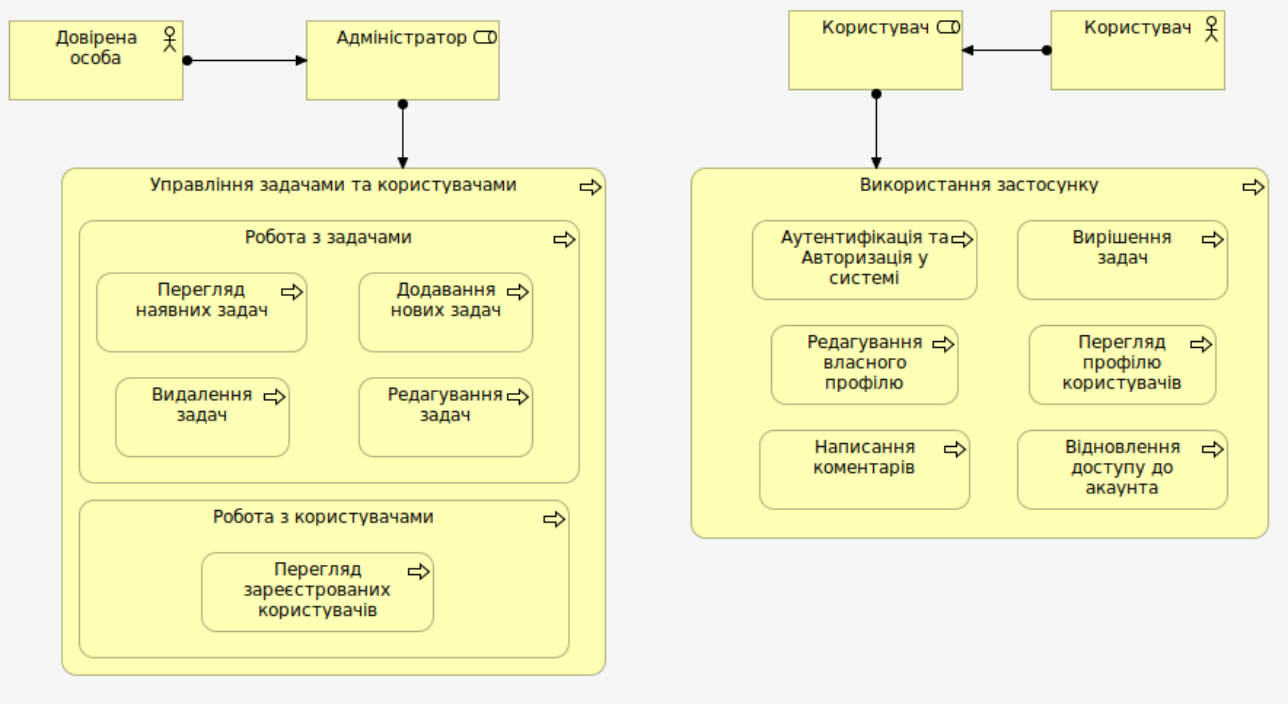

Рисунок 2.1 Бізнес архітектура системи додатку

2.2 Функціональне моделювання роботи застосунку

Використаємо нотацію IDEF0 для формалізації та опису наших бізнес-процесів. На найвищому рівні зображення процесів матимемо основний бізнес-процес - “Використання платформи” (рис. 2.2).

Вхідні стрілки - “Формулювання задач”, “Рішення задач”. Вони є вхідними даними, що потрібні для повноцінної роботи з системою. “Формулювання задач” - дані з яких складається задача, вони вносяться адміністратором застосунку, а саме: назву задачі, опис задачі, пояснення розв’язання задачі, обмеження до задачі, вхідні дані для подання на тестування, вихідні дані для перевірки рішення користувача, шляхом порівняння з даними отриманими від рішення користувача. “Рішення задач” - дані (мова програмування, код-рішення), які надають користувачі вирішуючи обрані задачі.

“Контроль” (стрілки зверху) є механізмом управління нашою системою та включає в себе такі обмеження як: “Формат формулювання задач”, що задає потрібну структуру даних задачі, а саме подання вхідних та вихідних тестових

даних у форматі JSON, а обмежень до задачі через рядки розділені комою; “Алгоритм розв’язання задач”, що передбачає використання користувачами необхідних алгоритмів для вирішення обраних задач, з метою досягнення найкращого результату.

“Механізми” (стрілки знизу) необхідні для проведення роботи по використанню платформи, у нашому випадку - адміністратор та користувач. Адміністратор відповідає за роботу з задачами, їхнє коректне формулювання, а користувач за рішення даних задач.

На виході з даного процесу матимемо інформацію, щодо виконання задач.

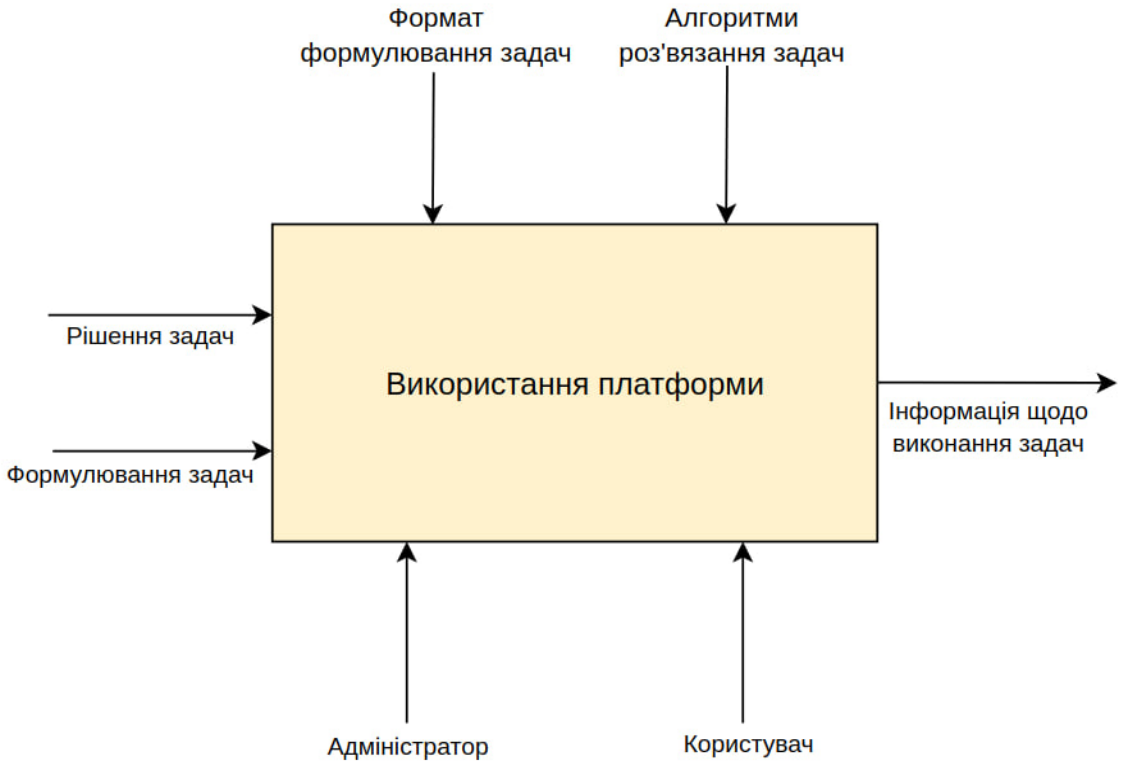

Рисунок 2.2 Подання процесу “Використання платформи” у вигляді контекстної IDEF0 діаграми

Вище була описана загальна схема по використанню платформи, далі деталізуємо даний бізнес-процес (рис. 2.3). Для цього декомпозуємо блок “Використання платформи” на пов’язані між собою етапи, а саме “Адміністрування” та “Розв’язання задач”. На даній схемі добре видно, на якому етапі, які елементи та механізми задіюються. Так адміністратор виконує

роботу по адмініструванню застосунку, результатом якої є набір сформульованих задач, які в свою чергу далі потрапляють на розв’язання до користувача, після чого вже отримуються вихідні дані щодо виконання задач.

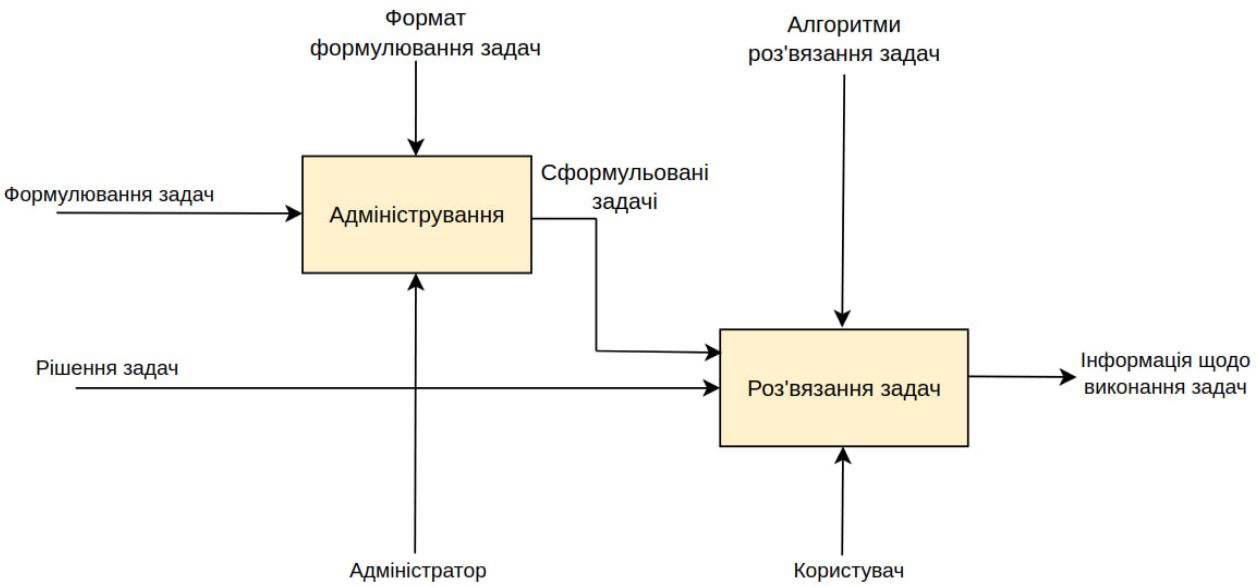

Рисунок 2.3 Декомпозиція процесу “Використання платформи” у вигляді IDEF0 діаграми 1-го рівня

2.3 Опис варіантів використання застосунку-тренувального майданчика для вирішення задач з програмування

Створення моделі варіантів використання передбачає визначення:

Дійові особи:

- Користувач – користувач програмного продукту.
- Адміністратор - людина, що виконує обов’язки адміністратора та має доступ до панелі адміністратора на сайті.

Для кращого розуміння загального представлення функціональних вимог до системи, яку ми намагаємося реалізувати, побудуємо діаграму варіантів використання даної програми (рис. 2.4).

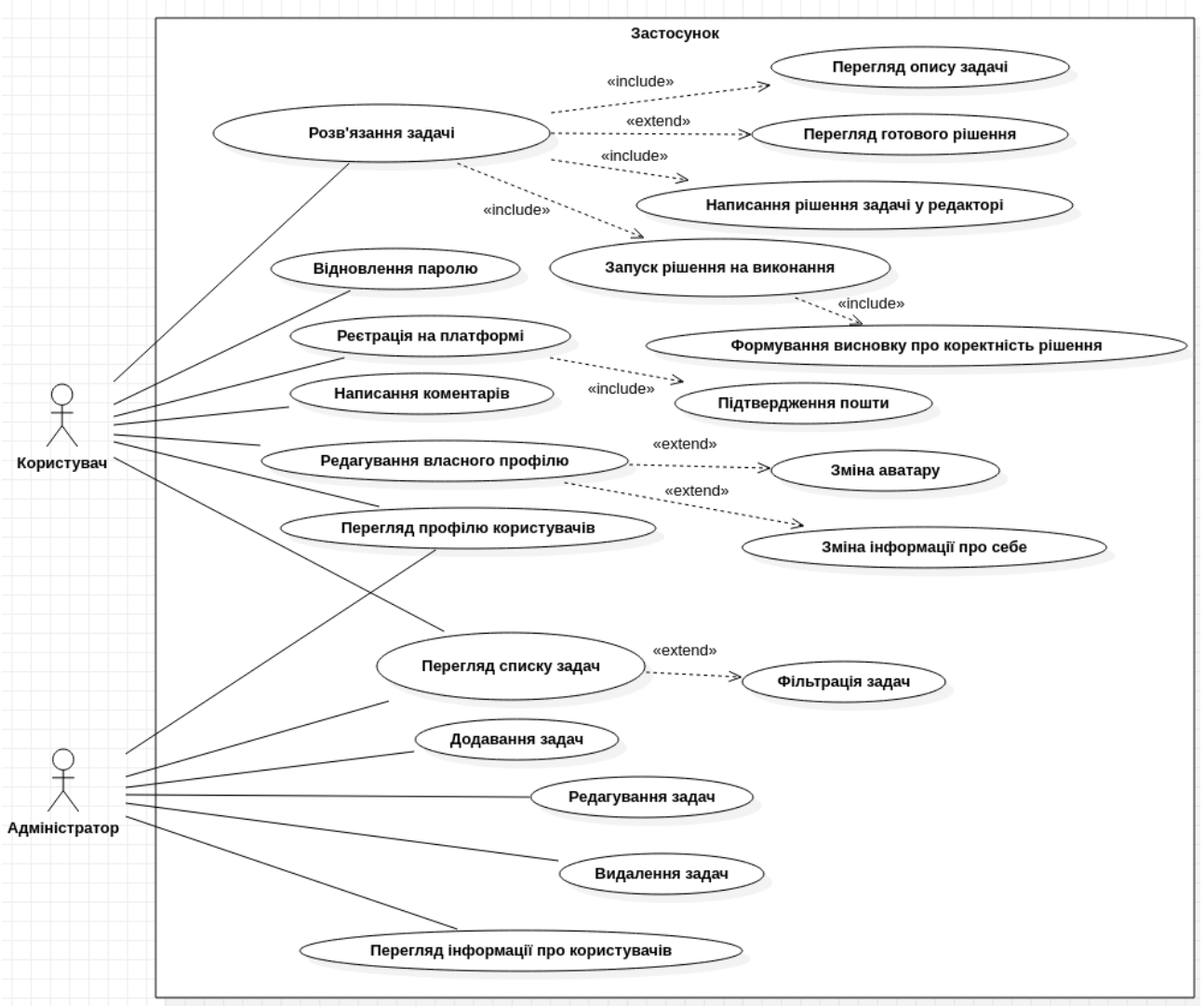

Рисунок 2.4 Діаграма варіантів використання розроблювальної програми

Варіанти використання:

- Реєстрація на платформі.
- Підтвердження пошти.
- Відновлення паролю.
- Перегляд списку задач.
- Фільтрація задач.
- Розв’язання задачі.
- Перегляд опису завдання.
- Перегляд готового рішення.
- Написання рішення задачі у редакторі.

- Запуск рішення на виконання.
- Формування висновку про коректність рішення.
- Написання коментарів.
- Перегляд профілю користувачів.
- Редагування власного профілю.
- Додавання аватару.
- Додавання інформації про себе.
- Додавання задач на сайт.
- Редагування задач.
- Видалення задач.
- Перегляд інформації про користувачів.

Оформлення сценаріїв:

Основний сценарій для користувача:

1. Користувач реєструється на платформі.
2. Користувач підтверджує реєстрацію через пошту, яку він надав.
3. Користувач авторизується на сайті.
4. Користувач переглядає список існуючих задач.
5. Користувач обирає цікаву йому задачу.
6. Користувач ознайомлюється з описом задачі.
7. Користувач вирішує задачу у вбудованому редакторі.
8. Користувач натискає кнопку запуску та перевірки його рішення.

Альтернативний сценарій для користувача:

1. Якщо при реєстрації користувач надав неіснуючу пошту або пошту до якої у нього немає доступу, він не зможе завершити успішно процес реєстрації, адже він мусить підтвердити надану ним пошту, через перехід по посиланню, яке буде відправлене йому повідомленням на вказану ним пошту. Також користувач має змогу запросити про повторне відправлення повідомлення з підтвердженням.

2. Якщо користувач забув свій пароль та не може увійти у свій профіль, він має змогу змінити пароль на новий, надавши свою електронну пошту, яку він використовував при реєстрації та перейшовши за посиланням, яке прийде у повідомленні на його адресу.
3. Якщо користувачу не вдається вирішити задачу, він має змогу переглянути готове рішення.
4. Якщо користувач допустить помилку в написаному коді для вирішення задачі, він отримає відповідне повідомлення та інформацію про саму помилку.

Основний сценарій для адміністратора:

1. Адміністратор авторизується на сайті.
2. Адміністратор переходить у панель адміністратора.
3. Адміністратор додає нову задачу для вирішення у систему або редагує чи видаляє існуючу.
4. Адміністратор переглядає інформацію про користувачів.

2.4 Узагальнена технологія роботи із застосунком-тренувальним майданчиком для вирішення задач з програмування

Користувач веб-застосунку реєструється у системі, після чого входить у неї ввівши дані, що він вказав при реєстрації, а саме електронну адресу та пароль. Далі в нього є можливість або переглянути власний профіль, де він може змінити аватар, редагувати особисту інформацію, переглянути статистику по вирішеним задачам та вийти з облікового запису, або переглянути список усіх задач, що доступні для вирішення. Дані задачі в нього є можливість відсіяти за категорією, складністю, чи здійснивши пошук за конкретною назвою задачі, після чого він вибирає задачу, яку бажає розв’язати та переходить на сторінку розв’язання, яка має опис завдання, опис готового рішення, на випадок, якщо користувачу потрібна буде допомога, секцію коментарів, де він може прочитати думки інших користувачів або ж висловити свою, вікно

редактора для написання коду. Вирішивши задачу користувач вказує мову програмування якою був написаний код-рішення до задачі та натискає кнопку запуску коду, після чого отримує результат перевірки його рішення у окремій панелі (рис. 2.5).

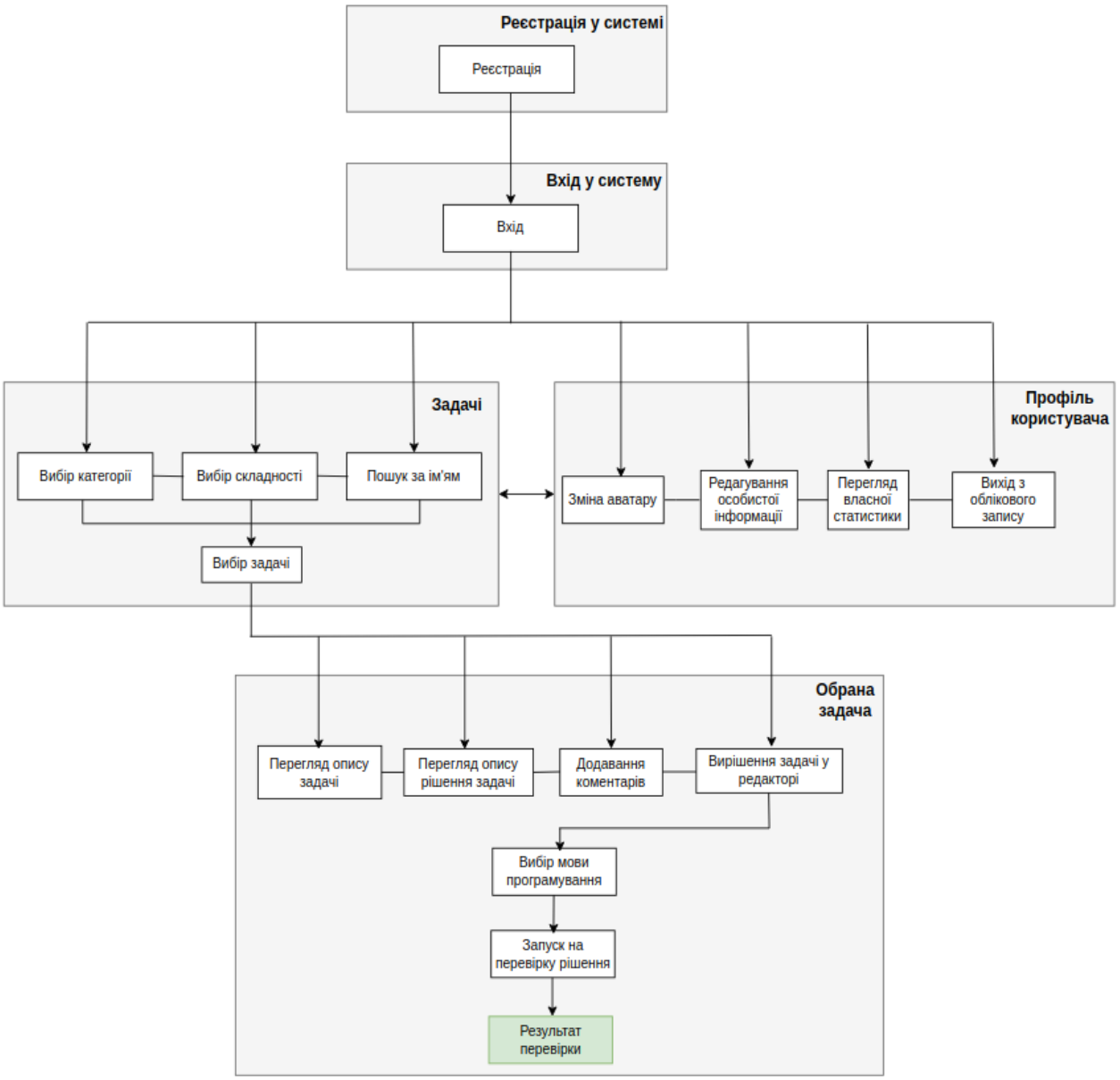

Рисунок 2.5 Технологія роботи з платформою для користувача

Адміністратор даного застосунку, після того як здійснив вхід до системи, ввівши коректні дані, має доступ до адмін-панелі, яка надає йому змогу додавати нові задачі на сайт, переглядати інформацію про користувачів та

видаляти і редагувати вже існуючі задачі, переглянувши попередньо їхню наявність (рис. 2.6).

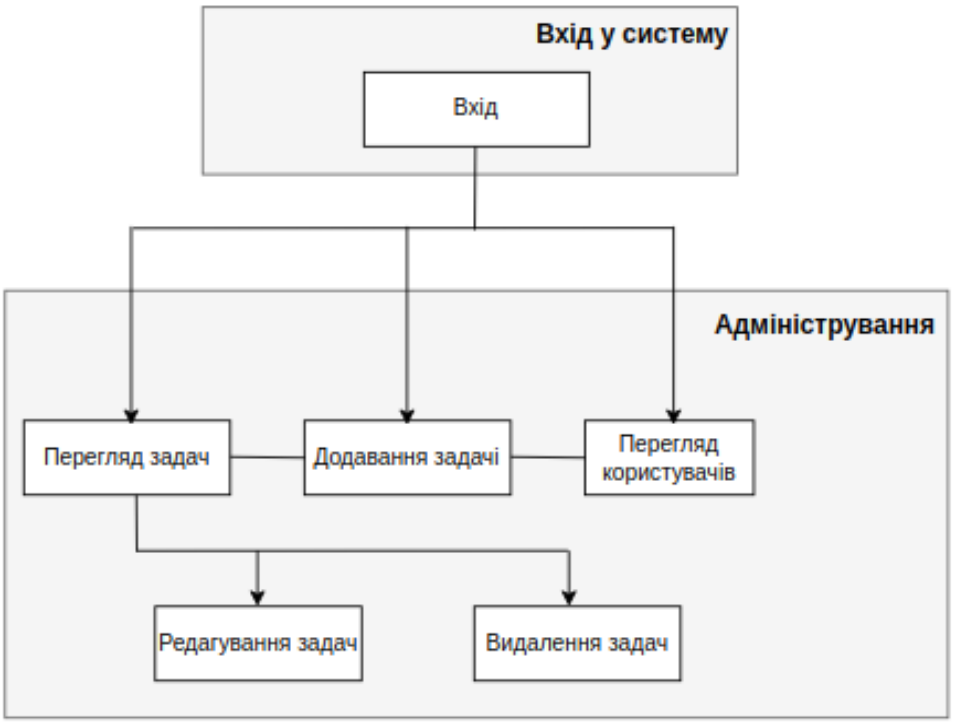

Рисунок 2.6 Технологія роботи з платформою для адміністратора

Панель адміністратора буде знаходитись на сторінці, що буде доступна тільки користувачам, які володіють роллю “Адміністратор”. Отримавши доступ до даного функціоналу, адміністратор виконуватиме покладені на нього обов’язки, стежучи за неточностями у наявних задачах та при знайдені таких виправляючи їх завдяки наявності можливості редагування та видалення задач. Також він буде знаходити нові задачі та додаватиме їх до системи, приводячи до конкретного формату. При виникненні потреби, переглядатиме основну інформацію про користувачів, що зареєстровані у системі.

2.5 Опис архітектури застосунку

Архітектурні особливості застосунку - тренувального майданчика для вирішення задач з програмування обумовлені використанням

- клієнт-серверної технології;

- контейнерів Docker;
- СУБД PostgreSQL;

Складові частини застосунку складатимуться з чотирьох компонентів - клієнтської системи та сервера, які будуть спілкуватися одне з одним використовуючи REST API по протоколу HTTP, бази даних, програми docker взаємодіяти з якою буде серверна частина через IPC (міжпроцесова взаємодія) (рис. 2.7).

Основна причина використовувати Docker для розробників — розв'язання проблеми залежності від робочого оточення.

У своєму ядрі docker дозволяє запускати практично будь-який додаток, безпечно ізольований в контейнері. Безпечна ізоляція дозволяє запускати на одному хості багато контейнерів одночасно.

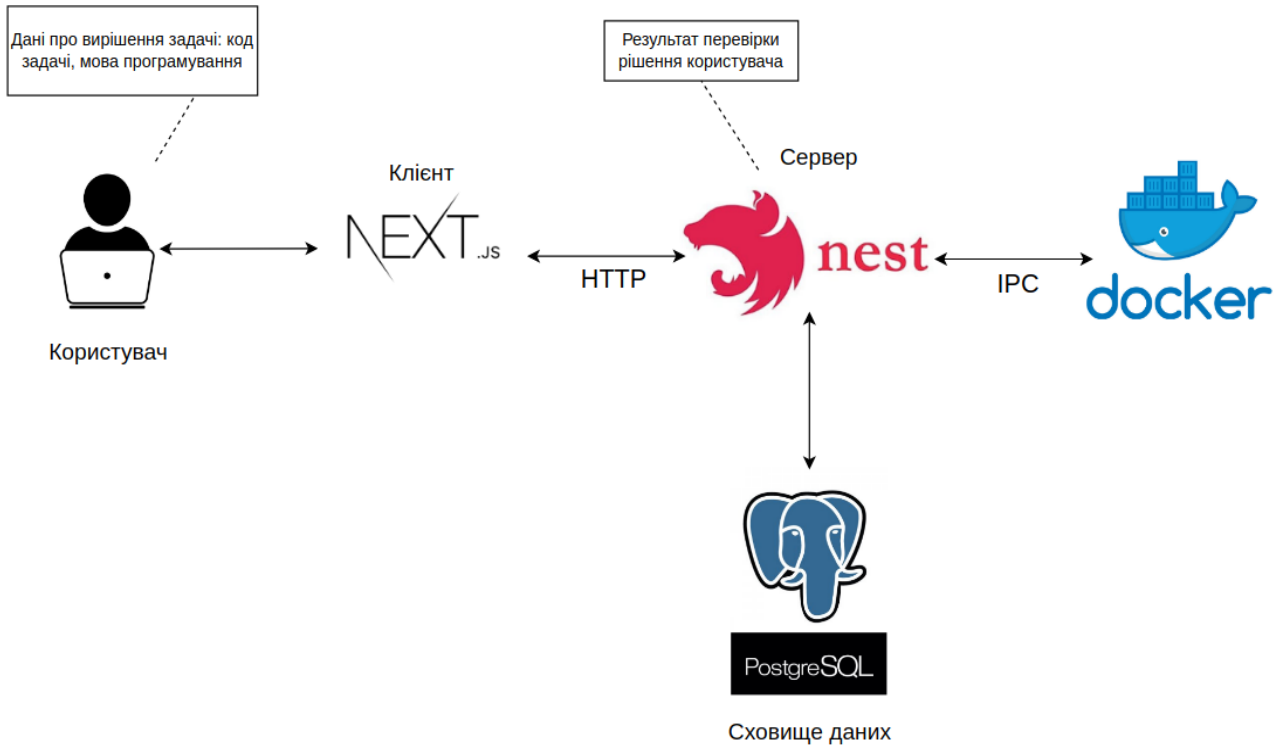

Рисунок 2.7 Архітектура розроблюваної системи

Серверна частина веб-застосунку буде в себе включати наступні елементи (рис. 2.8):

- модуль конфігурації, який відповідатиме за налаштування застосунку, ініціалізацію змінних оточення та встановлення з’єднання з базою даних, використовуючи бібліотеку TypeORM.
- модуль аутентифікації та авторизації користувача, він надає змогу реєструватися на платформі та входити у власний обліковий запис;
- модуль користувачів, що реалізовуватиме функціонал по додаванню користувачів до бази даних, їх пошуку, редагуванню інформації про них та встановленню аватару, за допомогою модулю файлів;
- модуль файлів призначений для збереження файлів у базі даних та роботою з ними;
- модуль електронної пошти надаватиме змогу сповіщати користувача про певні важливі події, відправляючи листи йому на пошту, також він використовується у модулі аутентифікації та авторизації під час реєстрації користувача, відправляючи йому повідомлення з підтвердженням пошти та під час випадків, коли користувачу потрібно відновити пароль до свого облікового запису;
- модуль задач, що призначений для роботи з задачами, які будуть вирішувати користувачі, він взаємодіє з базою даних та виконує такі функції як: збереження, редагування, пошук та видалення задач;
- модуль категорій задач, призначення котрого додавати нові категорії задач до застосунку, він реалізує ідентичний функціонал, що і модуль задач;
- модуль коментарів, призначений для надання змоги користувачам додавати коментарі до задач на сайті, редагувати власні коментарі та видаляти їх;
- модуль по роботі з рішенням задачі користувачем, він реалізує сервіс, що призначений для того, аби запускати код-рішення користувача на перевірку в docker-контейнері та в залежності від результату тестування даного коду, буде формувати відповідь для нашого клієнта.

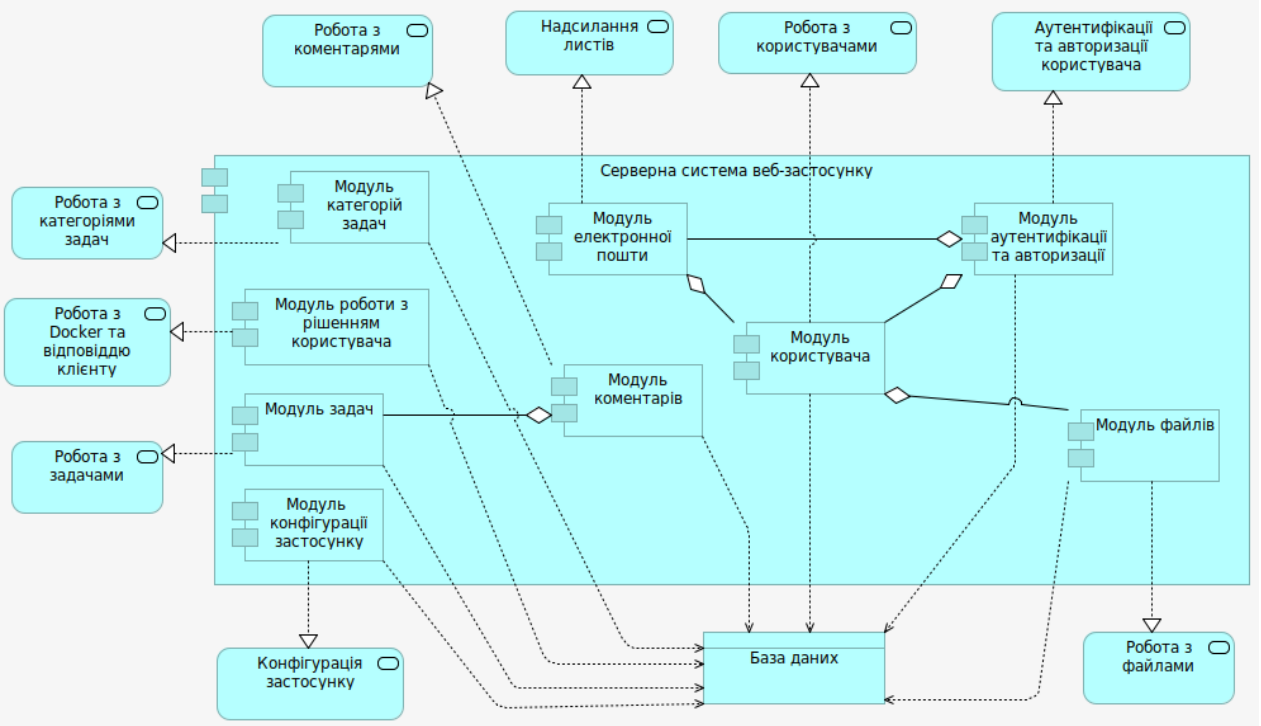

Рисунок 2.8 Діаграма представлення структури програми сервера

2.6 Проектування бази даних застосунку

База даних, з якою взаємодіє серверна частина, складається з семи таблиць, а саме таблиці “problem”, яка зберігає у собі дані про задачі для вирішення, “comment” - відповідає за збереження коментарів, “user” - відповідає за збереження користувачів, “category” - відповідає за збереження категорій до яких будуть відноситися задачі, “comment” - відповідає за збереження коментарів користувачів, “file” - відповідає за збереження файлів, “password_reset_token” - відповідає за збереження токенів для відновлення доступу до облікового запису, “user_solved_problem” - відповідає за збереження історії завдань, які були вирішені користувачем.

Для більш кращого розуміння структури бази даних та її сутностей, побудуємо модель “сутність-зв'язок” у нотації Пітера Чена (рис. 2.9) та опишемо атрибути кожної із таблиць, надавши пояснення та зазначивши їхній тип даних. На рисунку 2.10 подана фізична модель спроектованої бази даних.

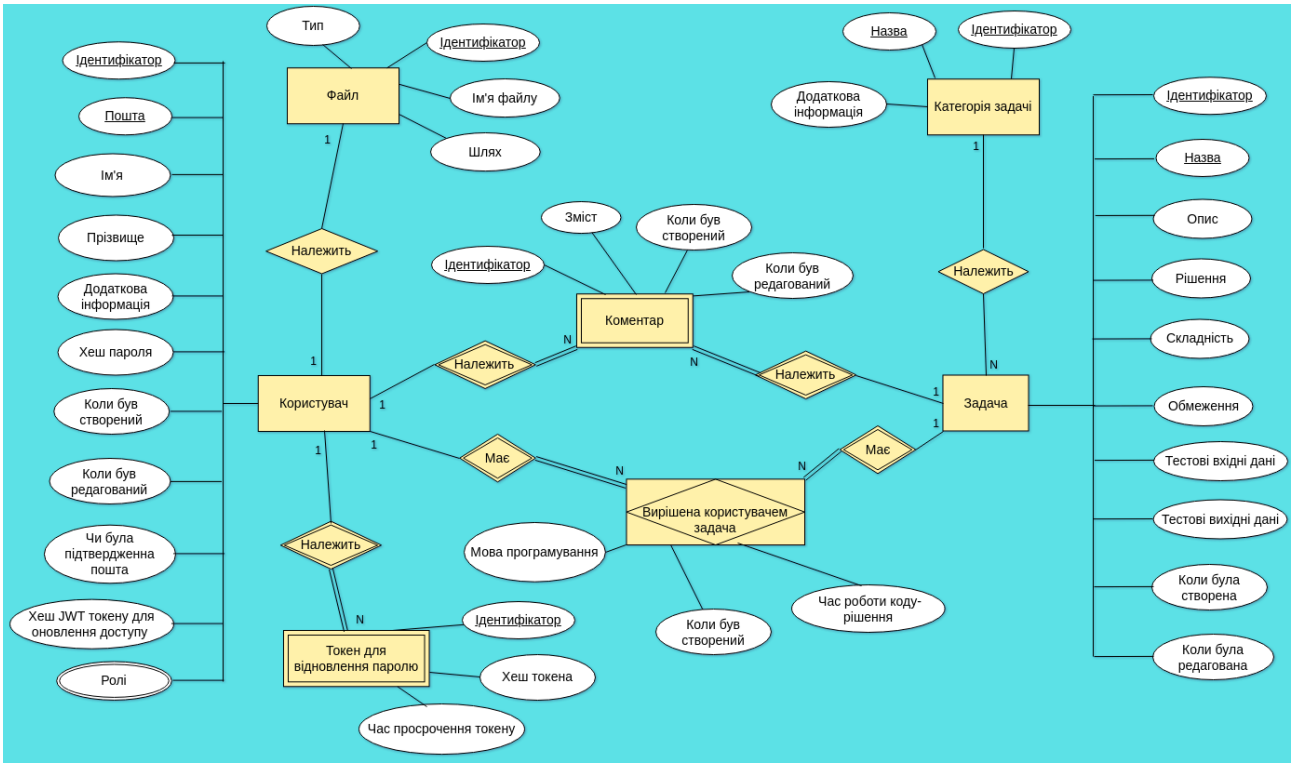

Рисунок 2.9 Схема бази даних веб-застосунку у нотації Чена

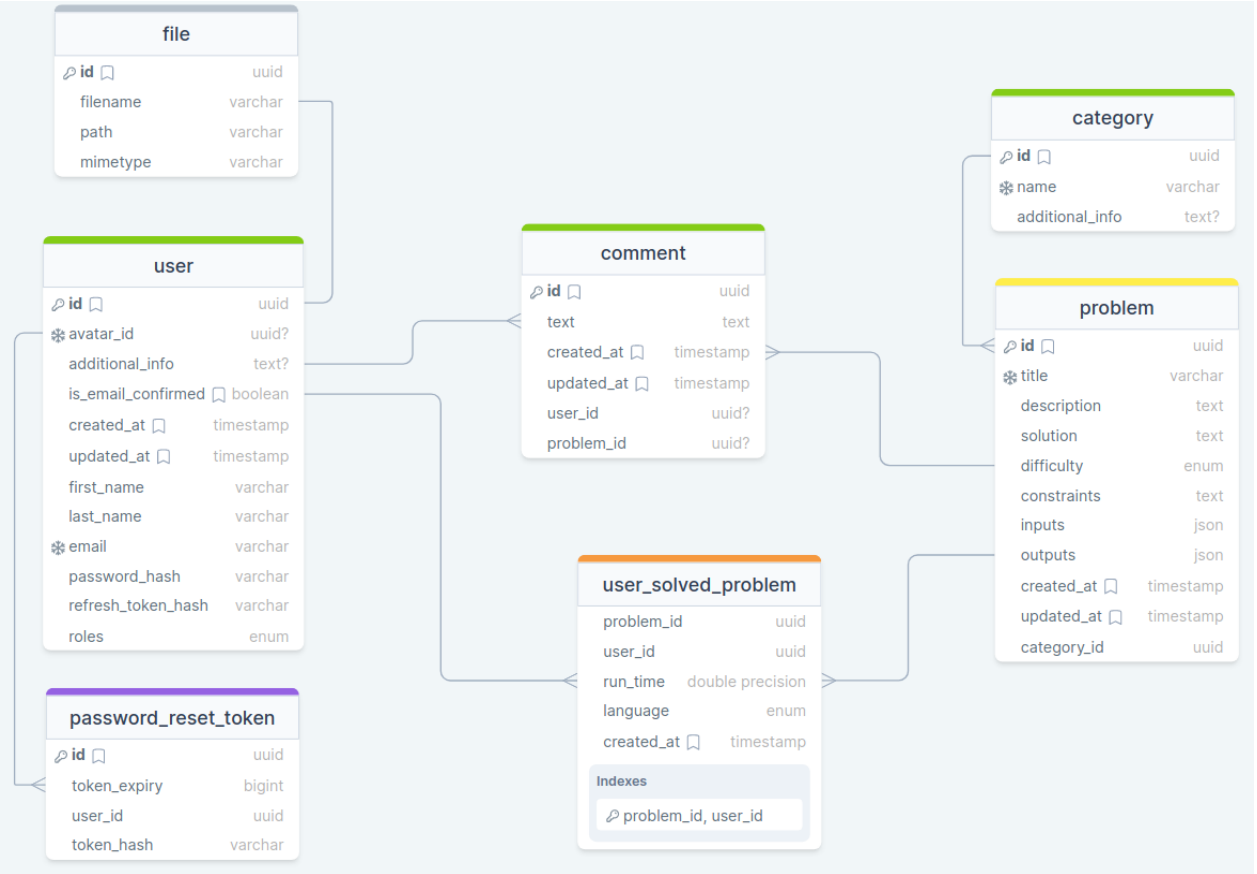

Рисунок 2.10 Фізична схема бази даних веб-застосунку

Опис атрибутів таблиць бази даних:

1. Атрибути таблиці “problem” включають:

- id (ідентифікатор) з типом даних uuid, що є первинним ключем даної таблиці;

- category_id (ідентифікатор категорії, до якої відноситься дана задача), з типом даних uuid, що є зовнішнім ключем даної таблиці;

- title (ім’я задачі) з типом даних varchar;

- description (опис задачі) з типом даних text;

- difficulty (складність задачі) з типом даних enum, що включає в себе такі значення як: “Easy”, “Medium”, “Hard”;

- solution (готове рішення) з типом даних text;

- constraints (умови обмеження задачі) з типом даних text array;

- inputs (вхідні дані для тестування) з типом даних json;

- outputs (вихідні дані для тестування) з типом даних json;

- created_at (відмітка дати створення задачі) з типом даних timestamp;

- updated_at (відмітка дати модифікації задачі) з типом даних timestamp.

2. Атрибути таблиці “category” включають:

- id (ідентифікатор) з типом даних uuid, що є первинним ключем даної таблиці;

- name (ім’я категорії) з типом даних varchar;

- additional_info (додаткова інформація) з типом даних text.

3. Атрибути таблиці “user” включають:

- id (ідентифікатор) з типом даних uuid, що є первинним ключем даної таблиці;

- avatar_id (ідентифікатор файлу, який є аватаром користувача), з типом даних uuid;

- first_name (ім'я користувача) з типом даних varchar;

- last_name (прізвище користувача) з типом даних varchar;

- email (електронна адреса користувача) з типом даних varchar;

- password_hash (захешований пароль користувача) з типом даних varchar;

- additional_info (додаткова інформація про користувача) з типом даних text;

- roles (ролі, якими володіє користувач) з типом даних enum array, що включає в себе такі значення як: “User”, “Admin”. За замовчуванням встановлюється значення “User”;

- is_email_confirmed (чи користувач підтвердив свою пошту) з типом даних boolean;

- refresh_token_hash (захешований токен для авторизації на сайті) з типом даних varchar;

- created_at (відмітка дати створення профілю) з типом даних timestamp;

- updated_at (відмітка дати модифікації профілю) з типом даних timestamp.

4. Атрибути таблиці “comment” включають:

- id (ідентифікатор) з типом даних uuid, що є первинним ключем даної таблиці;

- user_id (ідентифікатор користувача, якому належить коментар), з типом даних uuid, що є зовнішнім ключем даної таблиці;

- problem_id (ідентифікатор задачі, до якої був залишений коментар), з типом даних uuid, що є зовнішнім ключем даної таблиці;

- text (текст коментаря) з типом даних text;

- created_at (відмітка дати створення) з типом даних timestamp;

- updated_at (відмітка дати модифікації) з типом даних timestamp.

5. Атрибути таблиці “user_solved_problem” включають:

- user_id (ідентифікатор користувача), з типом даних uuid, що є первинним ключем даної таблиці;

- problem_id (ідентифікатор задачі), з типом даних uuid, що є первинним ключем даної таблиці.

- run_time (час з яким відпрацював код-рішення) з типом даних double;

- language (мова якою була вирішена задача) з типом даних enum;

- created_at (відмітка дати створення) з типом даних timestamp.

5. Атрибути таблиці “file” включають:

- id (ідентифікатор) з типом даних uuid, що є первинним ключем даної таблиці;

- filename (ім’я файлу) з типом даних varchar;

- path (шлях до файлу) з типом даних varchar;

- mimetype (тип файлу) з типом даних varchar.

6. Атрибути таблиці “password_reset_token” включають:

- user_id (ідентифікатор користувача), з типом даних uuid, що є первинним ключем даної таблиці;

- token_hash (захешований токен для відновлення паролю), з типом даних varchar, що є первинним ключем даної таблиці;

- token_expiry (закінчення терміну дії токену) з типом даних bigint;

- created_at (відмітка дати створення) з типом даних timestamp.

2.7 Розробка дизайну користувацького інтерфейсу веб-застосунку

Дизайн користувацького інтерфейсу був розроблений у веб-застосункі Figma. Представимо макети п’ятьох основних сторінок нашої платформи, сторінки з задачами, майданчик для вирішення завдань, сторінка користувача та сторінка реєстрації (рис. 2.11, рис. 2.12, рис. 2.13 , рис. 2.14, рис. 2.15).

Сторінка задач, на якій користувач може обрати для вирішення будь яке завдання складається зі списку наявних задач у системі, котрі були додані адміністратором та панелі фільтрації, що включає можливість сортування задач за складністю, категорією або ж пошук за назвою. Для переходу до майданчика вирішення завдання, користувачеві потрібно натиснути по імені задачі.

Майданчик веб-застосунку - це сторінка, на якій користувач вирішує обране ним завдання, вона включає панель для ознайомлення з детальним описом завдання, панель відповіді, котру користувач може переглянути, якщо хоче подивитися на готове рішення задачі, панель коментарів, де користувач може прочитати коментарі інших користувачів до даної задачі або ж залишити свій коментар, панелі текстового редактора, що призначення для написання

коду-рішення завдання та панелі консолі, у якій відображатиметься результат перевірки відповіді до задачі.

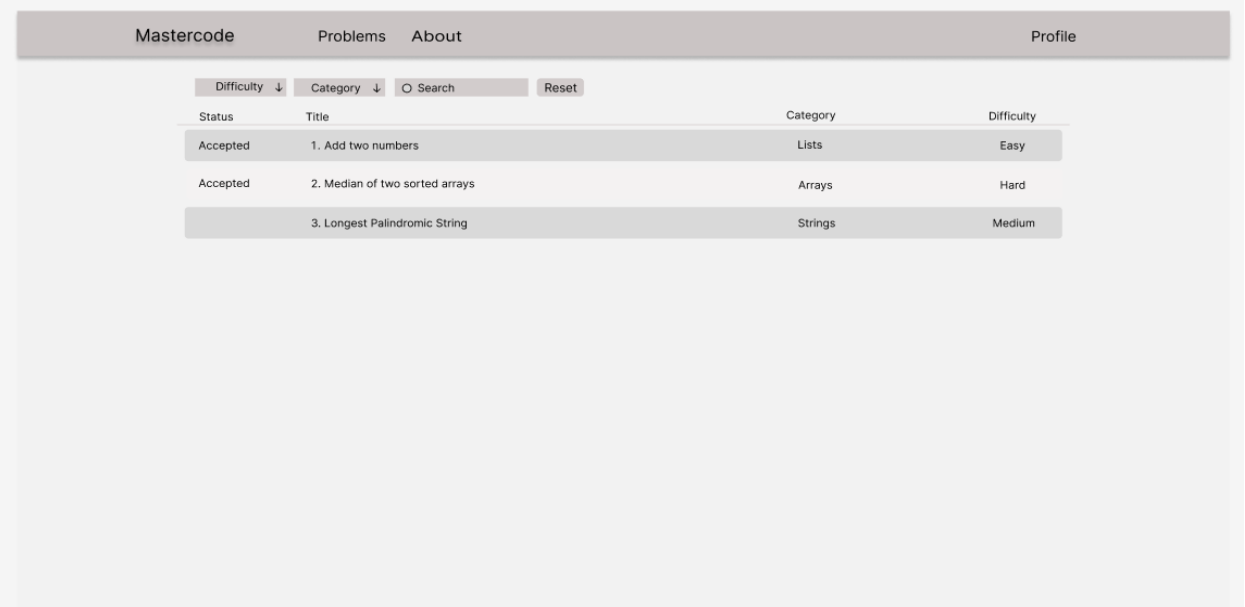

Рисунок 2.11 Сторінка зі списком задач веб-застосунку

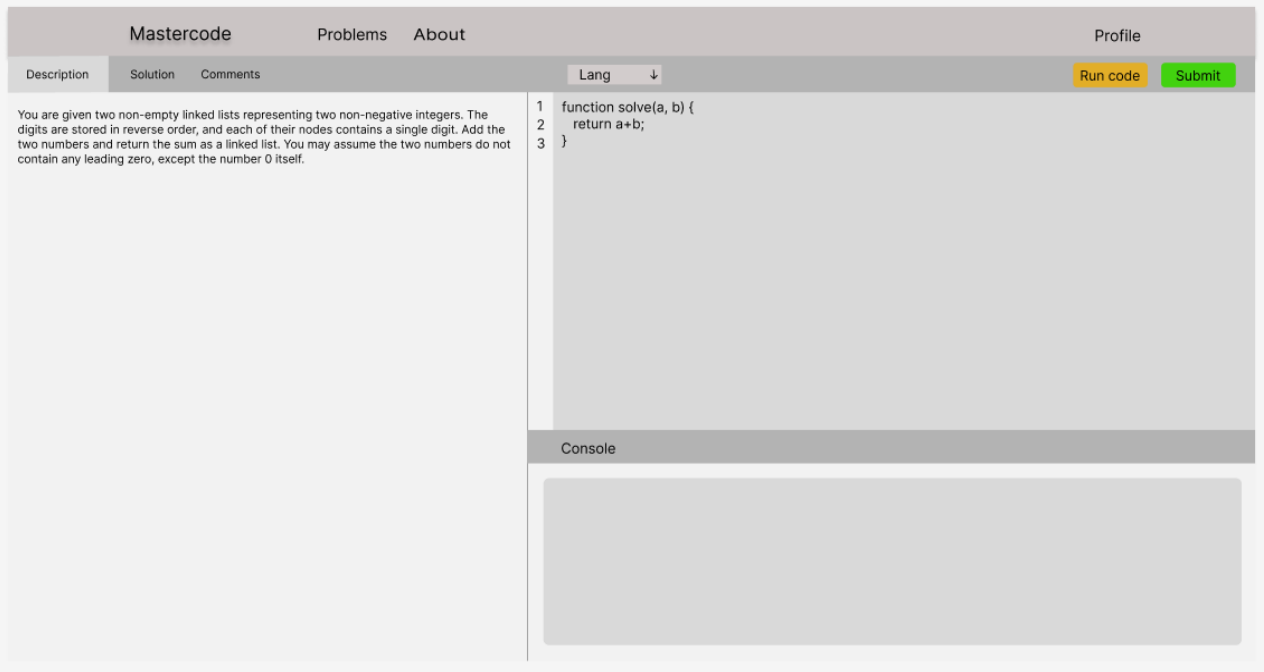

Рисунок 2.12 Майданчик веб-застосунку для вирішення завдань

На сторінці профілю користувача відображається інформація, яку він додав про себе, його аватар та статистика по вже вирішеним завданням. Наявний віджет кнопки для редагування власного профайлу.

Сторінка адміністратора призначена для роботи з задачами (додавання, видалення, редагування задач) та для перегляду інформації про користувачів, містить три кнопки, кожна з яких відображає різний контекст на сторінці. Кнопка “Problems” виводить список наявних задач, кнопка “Add problem” відображає форму для додавання задачі і кнопка “Users” виводить список зареєстрованих користувачів.

Сторінка реєстрації складається з простої форми, яку потрібно заповнити користувачеві аби мати доступ до всіх можливостей платформи. Користувач має ввести своє ім’я, прізвище, електронну адресу (на неї буде відправлено лист для підтвердження реєстрації), пароль та підтвердження паролю, після чого натиснути на кнопку з написом “Sign up”.

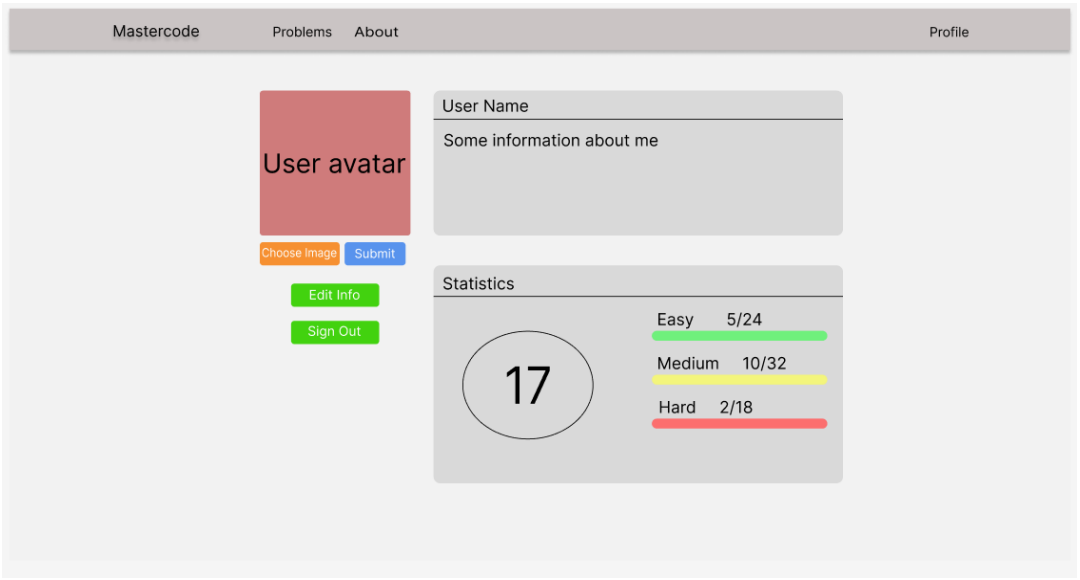

Рисунок 2.13 Сторінка профілю користувача

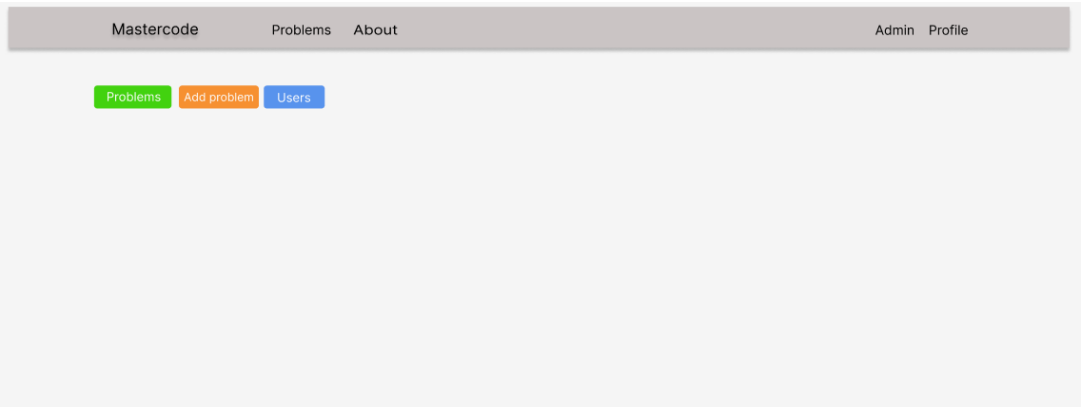

Рисунок 2.14 Сторінка реєстрації на платформі

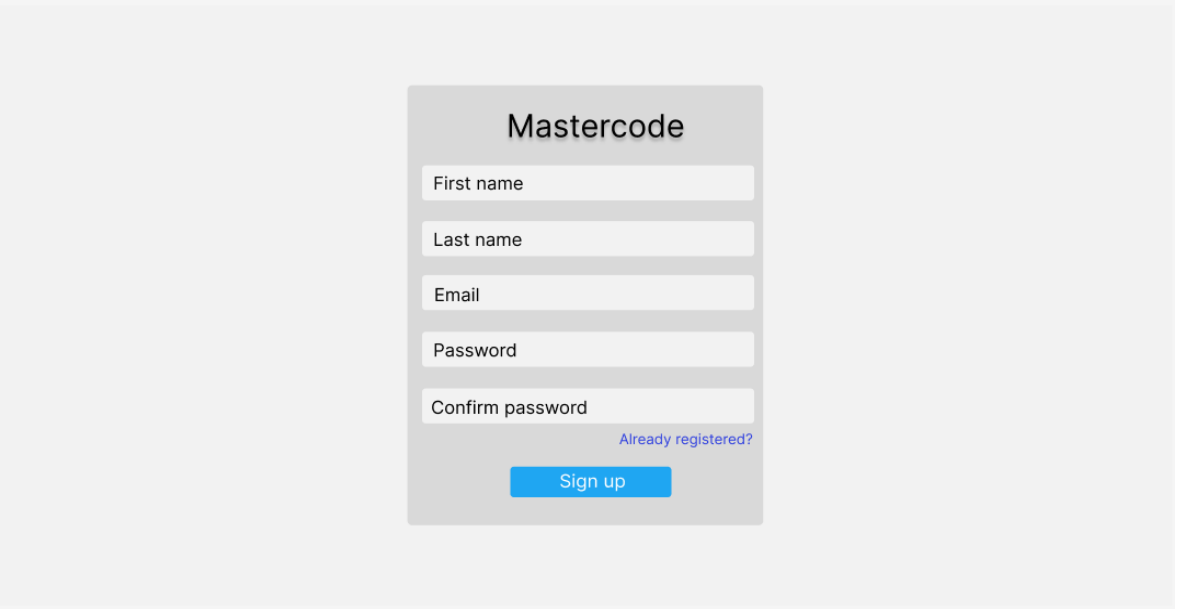

Рисунок 2.15 Сторінка реєстрації на платформі

Висновки до другого розділу

Отже, в результаті виконання другого розділу дипломної роботи, було визначено та проаналізовано основні бізнес процеси предметної області, виконано функціональне моделювання роботи застосунку та описано узагальнену технологію по роботі з ним, визначено основні сценарії використання, детально описано архітектуру додатка, взаємодію його компонентів, побудовано та описано схему бази даних, розроблено макет сторінок майбутнього інтерфейсу додатка.

РОЗДІЛ 3. ПРОГРАМНА РЕАЛІЗАЦІЯ ВЕБ-ЗАСТОСУНКУ ТРЕНУВАЛЬНОГО МАЙДАНЧИКА ДЛЯ ВИРІШЕННЯ ЗАДАЧ З ПРОГРАМУВАННЯ

3.1 Обґрунтування вибору інструментальних засобів для програмної реалізації застосунку

Розроблювальна система складається з двох основних частин, а саме з серверної та клієнтської. Виходячи з цього, для кожної з програмних систем були підібрані конкретні власні технології, які найкраще підходять для реалізації поставлених задач.

Спільні технології для клієнтської та серверної частин:

TypeScript - мова програмування, яка радше є обгорткою для JavaScript, що дозволяє додати до неї типізацію, тим самим допомагає уникнути багатьох помилок на етапі розробки або компіляції, які виникають при розробці програм, через динамічну систему типів JavaScript [2].

Клієнтська частина:

JavaScript - мова програмування, яка найчастіше використовується у сучасних браузерах, найкраще поміж інших мов підходить для створення веб-додатків, гнучка до використання різних парадигм, а також підтримує асинхронність [8].

React.js - JavaScript бібліотека, яка призначена для створення інтерактивних користувацьких інтерфейсів. React надає корисні функції для створення інтерфейсу користувача, але залишає розробнику вибір, де використовувати ці функції у своїй програмі. Частково успіх React полягає в тому, що він відносно байдужий щодо інших аспектів створення програм. Це призвело до процвітання екосистеми інструментів і рішень сторонніх розробників.

Next.js - гнучкий React фреймворк, що надає потрібні інструменти для створення сучасних веб-застосунків, а також дозволяє працювати з різними типами рендерингу HTML розмітки, одним з яких є SSR (server side rendering).

Під фреймворком мається на увазі, що Next.js бере на себе потрібні інструменти та конфігурацію, необхідні для React, і забезпечує додаткову структуру, функції та оптимізацію для програми розробника. Ви можете використовувати React для створення свого інтерфейсу користувача, а потім поступово застосовувати функції Next.js для вирішення типових вимог до додатків, таких як маршрутизація, вибірка даних, інтеграція – і все це покращує роботу розробника та кінцевого користувача [3].

Tailwind CSS - бібліотека для зручного задання стилів HTML сторінки [6].

Серверна частина:

Node.js - асинхронне, неблокуюче, подійо-орієнтоване середовище для запуску JavaScript на серверній частині, що розроблене для створення масштабованих мережевих додатків.

Nest.js - фреймворк для створення ефективних, надійних та масштабованих серверних додатків. Містить у собі більшість необхідних інструментів для написання надійного коду з гнучкою логікою. Він використовує прогресивний JavaScript і повністю підтримує TypeScript, поєднує елементи ООП (об’єктно-орієнтоване програмування), ФП (функціональне програмування) і ФРП (функціональне реактивне програмування). Nest надає готову архітектуру додатків, яка дозволяє розробникам і командам створювати високоякісні, масштабовані, слабко пов’язані та легко підтримувані додатки [4].

PostgreSQL - потужна open-source система управління об’єктно-реляційною базою даних.

TypeORM - open-source ORM, для зручної роботи з базою даних на сервері. Робота з реляційними базами даних є одним з каменів спотикання у розробці додатків. Налагодження SQL-запитів або складних ORM-об’єктів часто займає години розробки. TypeORM полегшує розробникам написання своїх запитів до бази даних, надаючи чистий і безпечний API для надсилання запитів до бази даних, який повертає звичайні об’єкти JavaScript. Дана ORM може працювати на платформах NodeJS, Browser, Cordova, PhoneGap, Ionic, React Native, NativeScript, Expo та Electron і може використовуватися з

TypeScript та JavaScript. ЇЇ мета - завжди підтримувати новітні функції JavaScript і надавати додаткові функції, які допоможуть вам розробити будь-який вид додатків, що використовують бази даних - від невеликих додатків з декількома таблицями до великомасштабних корпоративних додатків з декількома базами даних. TypeORM підтримує як шаблони Active Record, так і Data Mapper, на відміну від усіх інших існуючих на даний момент ORM JavaScript, що означає, що ви можете писати високоякісні, слабо пов'язані, масштабовані, підтримувані програми найбільш продуктивним способом. TypeORM знаходиться під сильним впливом інших ORM, таких як Hibernate, Doctrine і Entity Framework. [7].

Redis - сховище даних з відкритим вихідним кодом, що використовується мільйонами розробників як база даних, кеш, потоковий двигун та брокер повідомлень. У розоблювананому додатку даний інструмент буде використовуватися для кешування відповідей на запити користувачів, що містять дані, які рідко змінюються [13].

Основні можливості:

- Структури даних в пам'яті - добре відомий як "сервер структури даних", з підтримкою рядків, хешів, списків, наборів, відсортованих наборів, потоків тощо.
- Програмованість - сценарії на стороні сервера з Lua та процедури, що зберігаються на стороні сервера, з функціями Redis.
- Розширення API модуля для створення користувацьких розширень до Redis на C, C++, Rust.
- Персистентність - зберігає набір даних у пам'яті для швидкого доступу, але також може зберігати всі записи до постійного сховища, щоб пережити перезавантаження та системні збої.
- Кластеризація - горизонтальна масштабованість з шардингом на основі хешу, масштабування до мільйонів вузлів з автоматичним повторним розбиттям при вирощуванні кластера.

- Висока доступність - реплікація з автоматичним переходом на відмову як для автономних, так і для кластерних розгортань.

Варіанти використання:

- Сховище даних у режимі реального часу - універсальні структури даних Redis в пам'яті дозволяють будувати інфраструктуру даних для додатків реального часу, які вимагають низької затримки та високої пропускної здатності.
- Кешування та зберігання сеансів - швидкість Redis робить його ідеальним для кешування запитів до бази даних, складних обчислень, викликів API та стану сеансу.
- Потокове передавання та обмін повідомленнями - тип даних потоку забезпечує високу швидкість прийому даних, обміну повідомленнями, пошуку подій і сповіщень.

Docker - інструментарій для управління ізольованими контейнерами. Docker доповнює інструментарій LXC більш високорівневим API, що дозволяє керувати контейнерами на рівні ізоляції окремих процесів. Зокрема, Docker дозволяє не переймаючись вмістом контейнера запускати довільні процеси в режимі ізоляції і потім переносити і клонувати сформовані для даних процесів контейнери на інші сервери, беручи на себе всю роботу зі створення, обслуговування і підтримки контейнерів. Даний інструмент буде використаний для запуску коду користувачів, в ізольованому контексті, що дозволить забезпечити високий рівень безпеки для нашого додатку та серверу, також він спростить задачу використання різних підходів по запуску коду для різних мов програмування у майбутньому. Також він буде використаний при викладанні нашого додатку на хмарний сервіс у майбутньому [5].

3.2 Опис тест-кейсів для перевірки працездатності застосунку

З метою досягнення більшої впевненості у працездатності програми, проведемо тестування розробленого продукту за допомогою тест-кейсів. Існує

три основних типа тестування, які визначають високорівневу класифікацію тестів: тестування чорного ящика, тестування білого ящика та тестування сірого ящика. Тестування чорного ящика – це тестування функціональної та нефункціональної складової, що не передбачає знання того, як влаштована система зсередини, ідеї для тестування здійснюються від передбачуваної поведінки користувача. Тестування білого ящика – це тестування, що основується на аналізі внутрішньої структури системи. Тестування сірого ящика – це тестування, яке включає в собі підходи як тестування чорного ящика, так і білого.

Для вибору тест-кейсів та аналізу функціональних та нефункціональних особливостей розробленої інформаційної системи, було обрано тестування чорного ящика.

Таблиця 3.1 – Тест кейс №1

Номер	1
Заголовок	Реєстрація на платформі
Первинні дані	Відкрита сторінка реєстрації веб-застосунку у браузері
Кроки	**Очікуваний результат**
Заповнити потрібні поля у формі для реєстрації та натиснути кнопку “Sign Up”.	Не було виведено повідомлення про помилки, з’явилось сповіщення про необхідність підтвердження введеної електронної пошти.
Відкрити повідомлення, що було відправлене на вказану пошту при реєстрації на платформі та перейти за наданим посиланням для підтвердження реєстрації користувача.	Було переведено на сторінку з повідомленням про успішне завершення реєстрації.

Таблиця 3.2 – Тест кейс №2

Номер	2
Заголовок	Вхід у власний обліковий запис
Первинні дані	Відкрита сторінка логіну веб-застосунку у браузері
Кроки	**Очікуваний результат**
Заповнити потрібні поля у формі для логіну та натиснути кнопку “Sign In”.	Не було виведено повідомлення про помилки, було переадресовано на головну сторінку сайту та стали доступними вкладки, що не були доступні до цього.

Таблиця 3.3 – Тест кейс №3

Номер	3
Заголовок	Перегляд власного облікового запису
Первинні дані	Відкрита головна сторінка веб-застосунку
Кроки	**Очікуваний результат**
Перейти з головної сторінки додатку на сторінку “Profile”.	На сторінці відображається інформація про користувача, статистика по вирішеним задачам.

Таблиця 3.4 – Тест кейс №4

Номер	4
Заголовок	Редагування власного облікового запису
Первинні дані	Відкрита головна сторінка веб-застосунку

Кроки	**Очікуваний результат**
Перейти з головної сторінки додатку на сторінку “Profile”.	На сторінці відображається інформація про користувача, статистика по вирішеним задачам.
Встановити власний аватар у профілю.	Встановлений аватар відображається на сторінці та не зникає при перевантажені даної сторінки.
Редагувати інформацію про себе натиснувши на кнопку “Edit Info” та вписавши дані у форму, що з’явилась.	Дані користувача змінилися на сторінці та залишаються незмінними при перевантажені сторінки.

Таблиця 3.5 – Тест кейс №5

Номер	5
Заголовок	Перегляд доступних задач для вирішення
Первинні дані	Відкритий веб-застосунок у браузері
Кроки	**Очікуваний результат**
Перейти з головної сторінки додатку на сторінку “Problems”.	На сторінці відображається список задач з інформацією про назву задачі, рівень складності задачі, категорія до якої вона відноситься. На сторінці наявна панель фільтрування задач, що дозволяє відсортувати задачі за рівнем складності, категорією або власною назвою.

Таблиця 3.6 – Тест кейс №6

Номер	6
Заголовок	Перехід на сторінку з обраною задачею
Первинні дані	Відкритий веб-застосунок у браузері
Кроки	**Очікуваний результат**
Перейти з головної сторінки додатку на сторінку “Problems”.	На сторінці відображається список задач з інформацією про назву задачі, рівень складності задачі, категорія до якої вона відноситься.
Натиснути на ім’я певної задачі.	Переведено на сторінку для вирішення обраної задачі. Сторінка містить усі потрібні панелі та в панелі “Description” відображається опис обраної задачі.

Таблиця 3.7 – Тест кейс №7

Номер	7
Заголовок	Успішне вирішення задачі
Первинні дані	Відкритий веб-застосунок на сторінці певної задачі.
Кроки	**Очікуваний результат**
Ознайомитися з умовою задачі.	Активною відображається панель “Description”, що містить умову задачі.

Написати коректне рішення обраної задачі у текстовому редакторі та натиснути кнопку “Submit”.	У панелі під редактором “Console”, з’явився текст зеленого кольору “Accepted”. Справа від нього - час виконання програми.

Таблиця 3.8 – Тест кейс №8

Номер	8
Заголовок	Невдале вирішення задачі
Первинні дані	Відкритий веб-застосунок на сторінці певної задачі.
Кроки	**Очікуваний результат**
Ознайомитися з умовою задачі.	Активною відображається панель “Description”, що містить умову задачі.
Написати некоректне рішення обраної задачі у текстовому редакторі та натиснути кнопку “Submit”.	У панелі під редактором “Console”, з’явився текст червоного кольору “Wrong answer”. Справа від нього - час з яким відпрацював код користувача.

Таблиця 3.9 – Тест кейс №9

Номер	9
Заголовок	У рішенні задачі допущена синтаксична помилка
Первинні дані	Відкритий веб-застосунок на сторінці певної задачі.
Кроки	**Очікуваний результат**

Ознайомитися з умовою задачі.	Активною відображається панель “Description”, що містить умову задачі.
Допустити у коді-рішенні задачі синтаксичну помилку та натиснути кнопку “Submit”.	У панелі під редактором “Console”, з’явився текст червоного кольору “Error”, з інформацією про допущену помилку. Справа від тексту час виконання програми відсутній.

Таблиця 3.10 – Тест кейс №10

Номер	10
Заголовок	Робота з коментарями
Первинні дані	Відкритий веб-застосунок на сторінці певної задачі.
Кроки	**Очікуваний результат**
Перейти на вкладку “Comments”.	Активною відображається панель “Comments”, що містить список коментарів користувачів до даної задачі.
Додати коментар до даної задачі, натиснувши кнопку “Write” та ввівши бажаний текст у текстове поле.	У панелі “Comments” відобразився доданий користувачем коментар до задачі, який також містить кнопки “Edit” та “Delete”, для редагування та видалення власного коментаря, відповідно, а також відображається дата коли був доданий даний коментар.

Таблиця 3.11 – Тест кейс №11

Номер	11
Заголовок	Перевірка функціоналу панелі адміністратора
Первинні дані	Користувач з правами адміністратора авторизований у системі та знаходиться на головній сторінці.
Кроки	**Очікуваний результат**
Перейти на сторінку “Admin”.	Переадресовано на сторінку адміністратора, на якій наявні три кнопки “Problems” (для відображення усіх задач, що є у системі), “Add problem” (для додавання нової задачі), “Users” (для перегляду зареєстрованих у системі користувачів).
Додамо нову задачу до системи, натиснувши кнопку “Add problem” та заповнивши потрібні поля у формі, що з’явиться, після чого натиснемо кнопку “Confirm”.	Модальне вікно з заповненою формою успішно закрилося і не було виведено повідомлення про помилку.
Перевіримо наявність у системі доданої задачі, натиснувши кнопку “Problems”.	Було виведено список наявних задач, серед яких присутня тільки, що додана задача. Біля кожної з задач відображається кнопка “Edit” та “Delete” для редагування та видалення задачі.

Відредагуємо тільки що додану задачу, натиснуваши біля даної задачі кнопку “Edit” та змінивши назву задачі, у формі, що з’явилася.	Назва задачі була успішно змінена.
Видалимо дану задачу із системи, натиснувши на кнопку “Delete”.	Дана задача більше не відображається серед списку інших задач.
Переглянемо список користувачів, що зареєстровані у системі, натиснувши на кнопку “Users”.	Був виведений список користувачів, з інформацією про електронну адресу користувача, його повне ім’я та права якими він володіє.

3.3 Опис роботи застосунку

Наведемо опис тестових прикладів для демонстрації роботи веб-застосунку тренувального майданчику для вирішення програмних задач.

Перший приклад:

Для доступу до потрібного функціоналу застосунку зареєструємося на платформі та увійдемо у створений обліковий запис. На рисунку 3.1 зображена форма для реєстрації на сайті з заповненими полями, що потрібні для успішного виконання даного процесу.

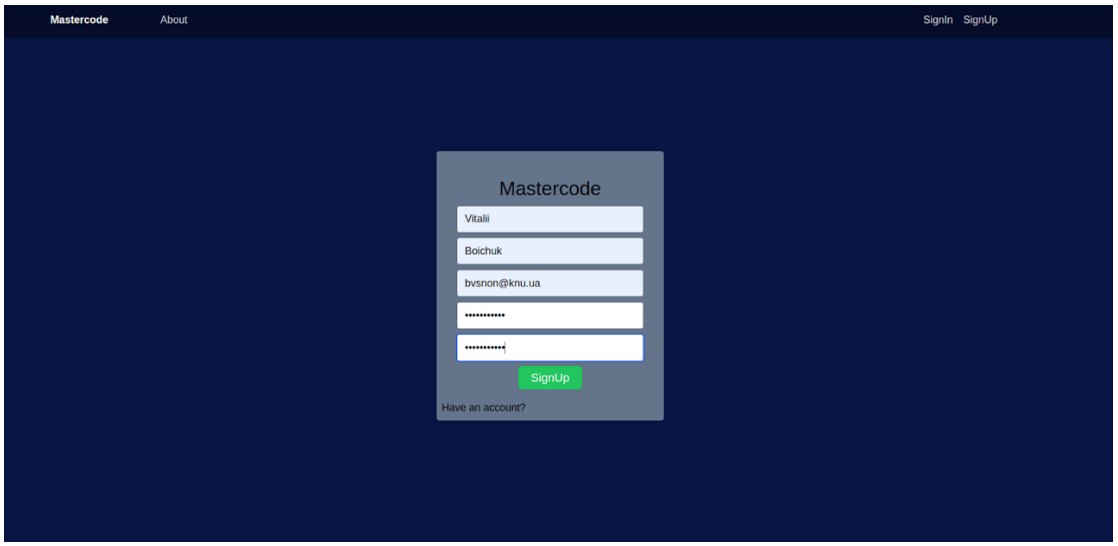

Рисунок 3.1 - Форма реєстрації

Після відправки даних на сервер, користувачу прийде повідомлення на вказану ним при реєстрації електронну адресу, з посиланням на котре потрібно перейти аби підтвердити вказану пошту та завершити процес реєстрацій у додатку (рис. 3.2).

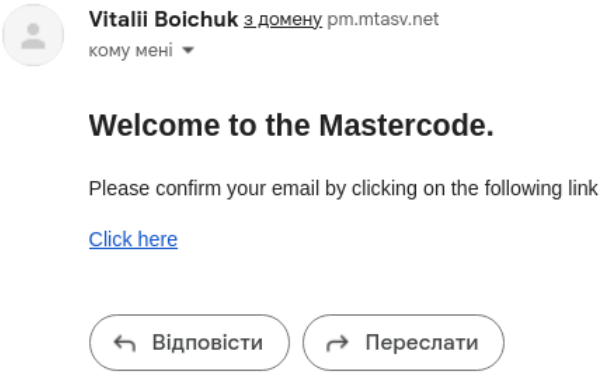

Рисунок 3.2 - Лист підтвердження

Після переходу за наданим посиланням, користувач буде переведений на спеціальну сторінку з повідомленням, що електронна адреса була успішно підтверджена і він тепер може увійти у свій обліковий запис (рис. 3.3).

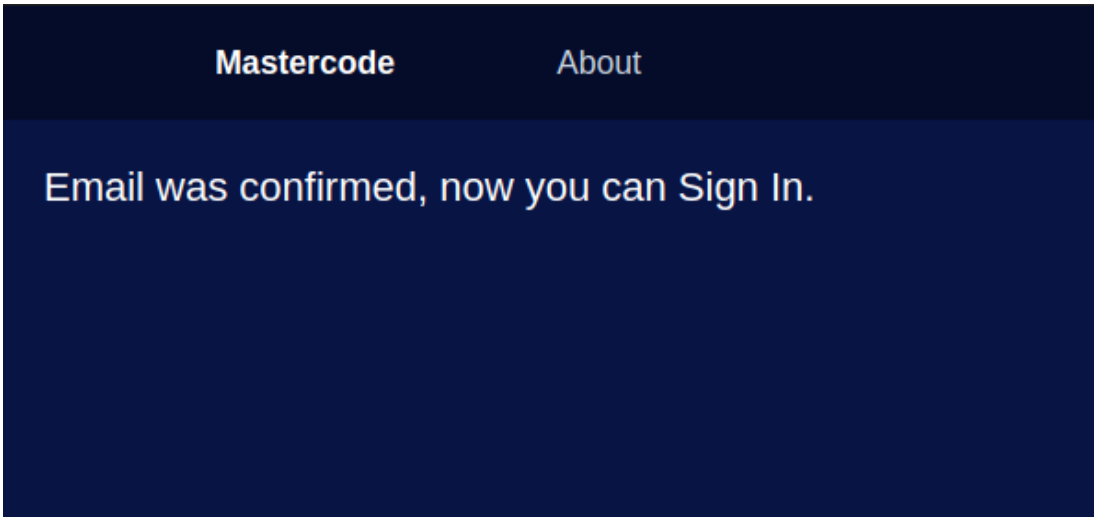

Рисунок 3.3 - Сторінка з повідомленням про підтвердженість електронної адреси

Останнім кроком залишається ввести потрібні дані, що були вказані при реєстрацій у форму для логіну (рис. 3.4).

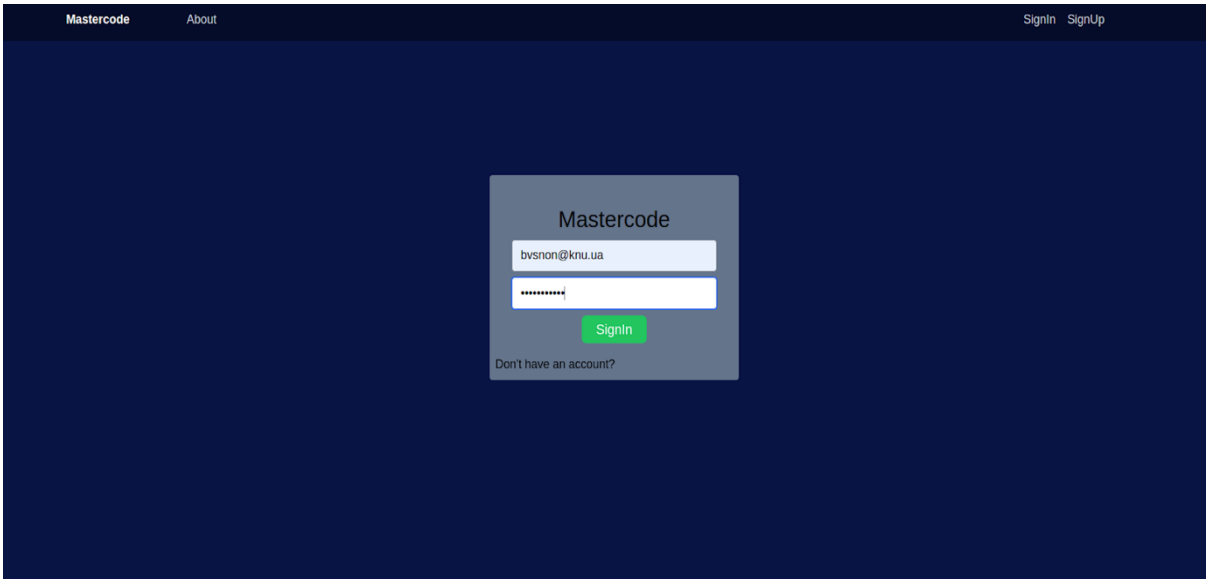

Рисунок 3.4 - Форма логіну

Якщо дані були введені коректно, користувач буде переведений на головну сторінку веб-застосунку та буде бачити вкладки у навігаційній панелі, що не відображалися до цього, в іншому випадку він не буде авторизований у системі (рис 3.5).

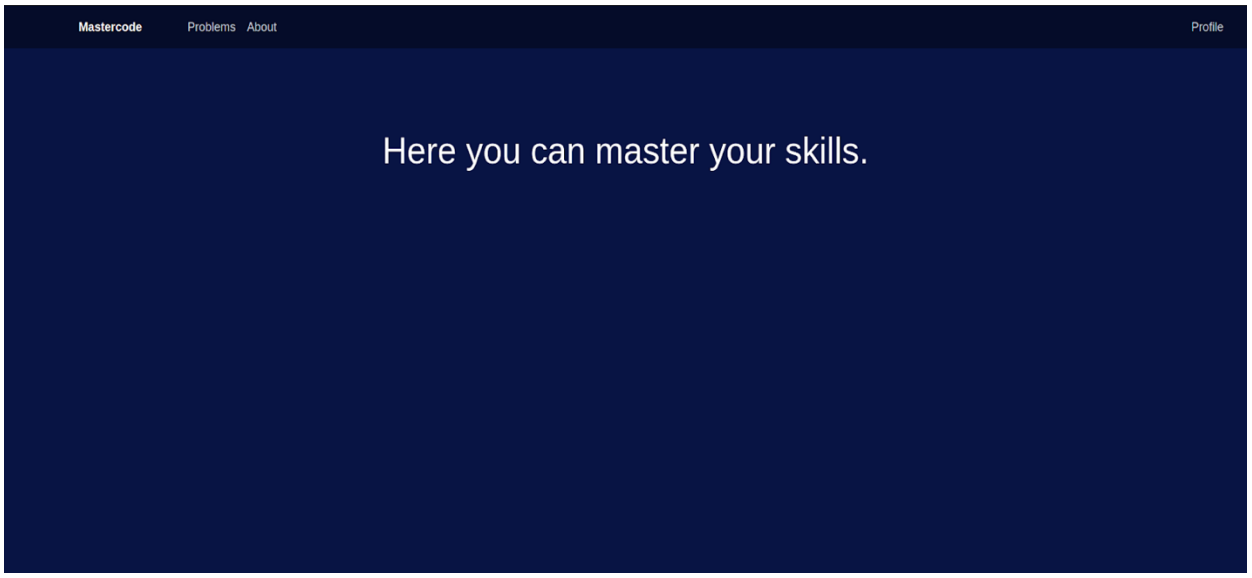

Рисунок 3.5 - Головна сторінка застосунку

Другий приклад:

Переглянемо особистий профіль користувача, натиснувши на навігаційній панелі на вкладку “Profile” (рис. 3.6) та відредагуємо його дані.

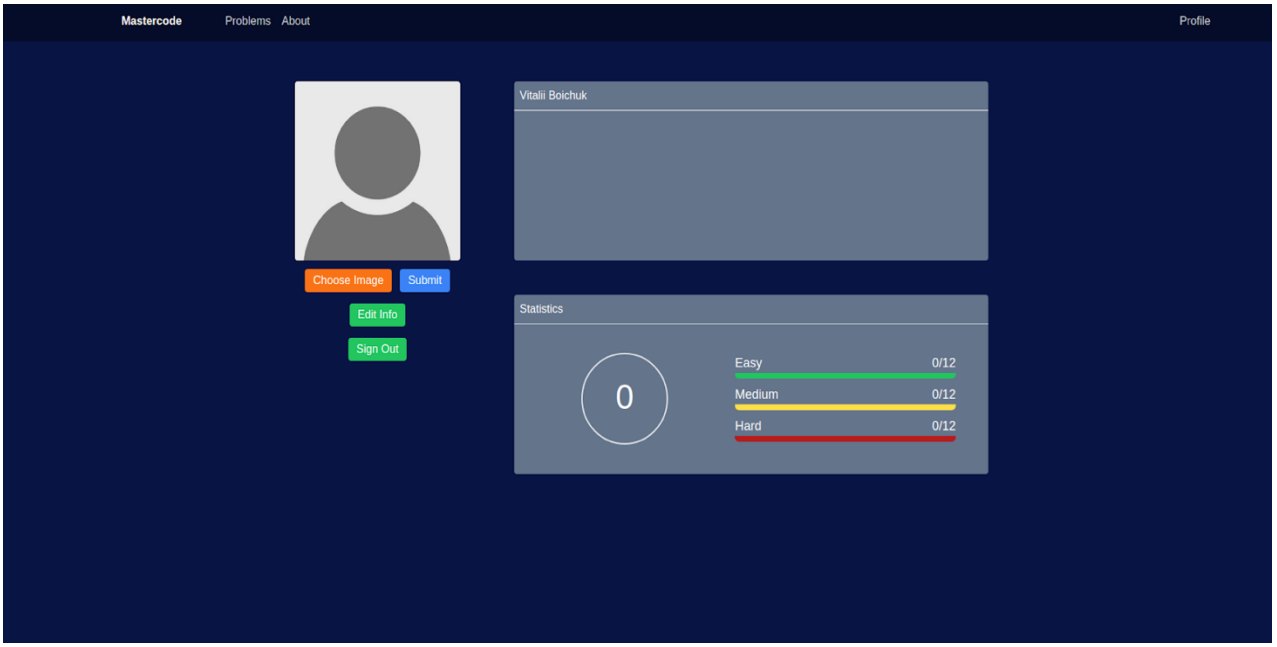

Рисунок 3.6 - Профіль користувача

Встановимо користувачу новий аватар та додамо додаткову інформацію про нього (рис. 3.7, 3.8).

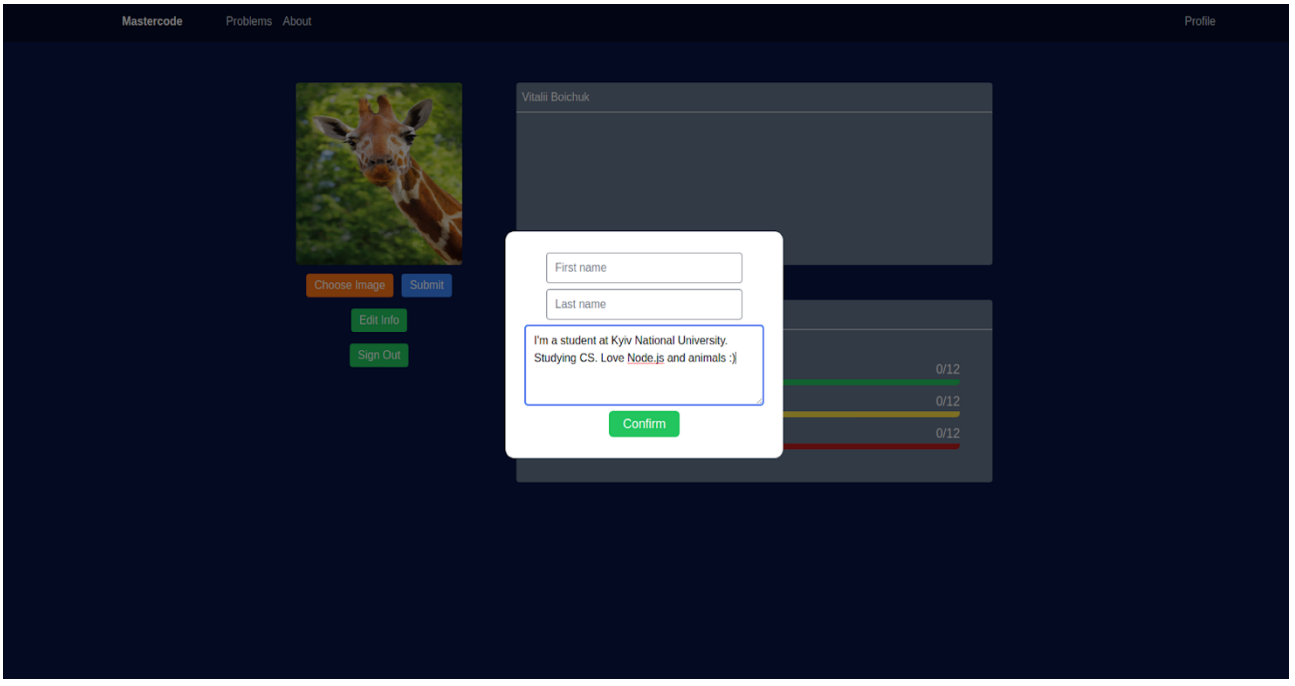

Рисунок 3.7 - Форма редагування особистих даних користувача

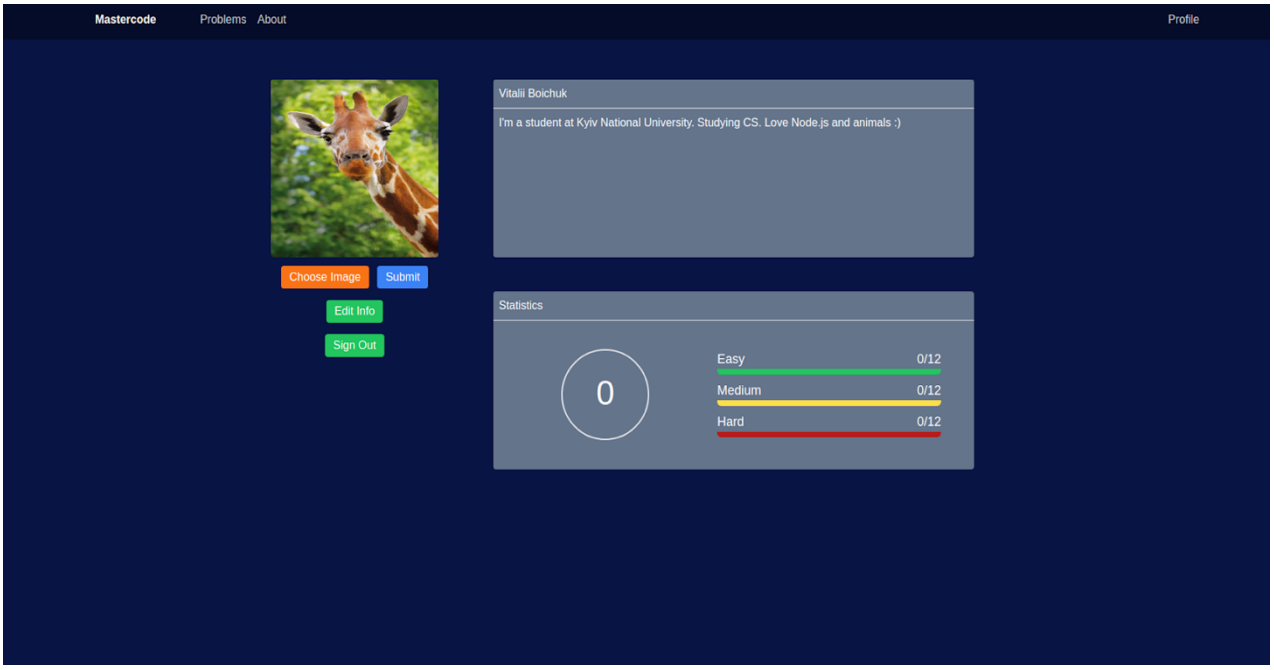

Рисунок 3.8 - Вигляд відредагованого профілю користувача

Третій приклад:

Переглянемо доступні задачі для вирішення. Перейдемо на сторінку задач, натиснувши на вкладку “Problems”, після чого перед нами з’явиться список усіх задач, які можна вирішити (рис. 3.9).

Рисунок 3.9 - Сторінка переліку задач

Четвертий приклад:

На сторінкі задач, оберемо завдання, яке бажаємо розв’язати, натиснувши на його ім’я, нехай це буде завдання під номером 4, нас переадресує на сторінку цього завдання, де буде можливо прочитати опис задачі та вирішити її (рис. 3.10).

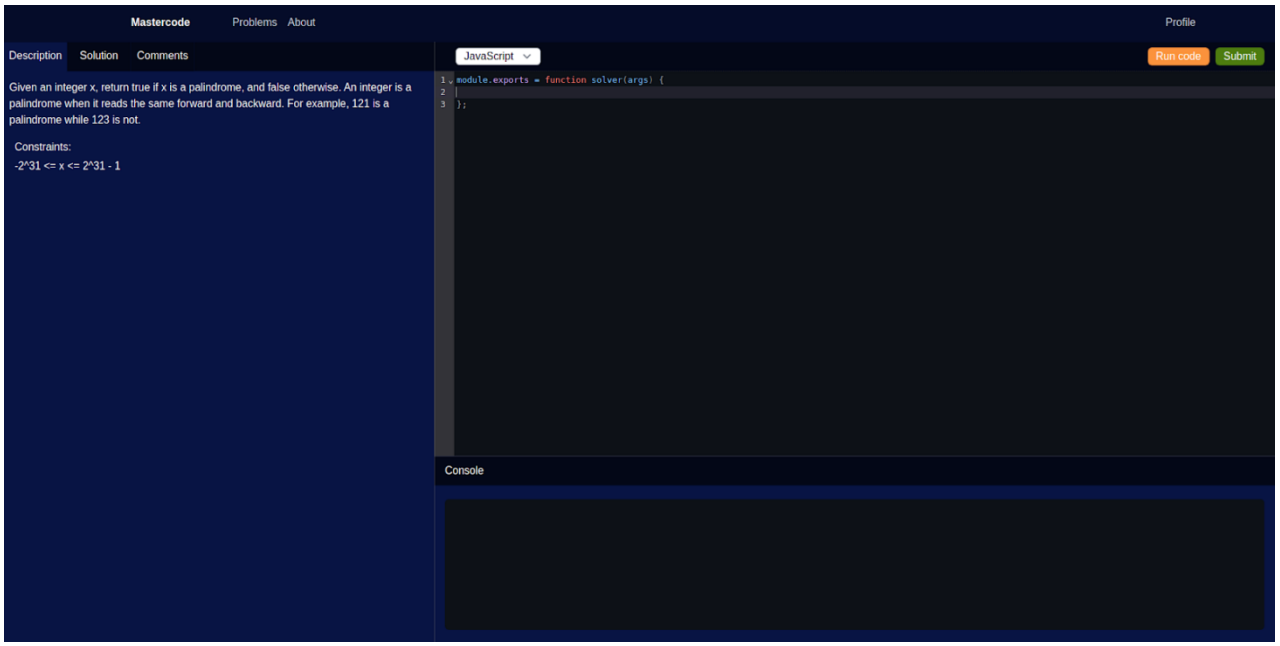

Рисунок 3.10 - Сторінка обраної задачі

Як бачимо з попереднього рисунку, дана сторінка має вбудований редактор для написання коду для рішення задачі, детальний опис даної задачі, панель, що має назву “Console”, в якій і будуть відображатися результати перевірки програми користувача, після того як він натисне кнопку “Run code” або “Submit”. Також дана сторінка має панель “Solution”, активувавши котру, можна переглянути готове рішення даної задачі.

П’ятий приклад:

Коректно вирішимо обрану задачу. Оберемо мову для вирішення даної задачі “JavaScript”, напишемо код-рішення у вбудований редактор та натиснемо кнопку “Run code” (рис. 3.11). Як бачимо з рисунку, в терміналі з’явився текст зеленого кольору “Correct” (“Правильно”), що сигналізує про коректне вирішення задачі та час з яким відпрацював алгоритм рішення користувача.

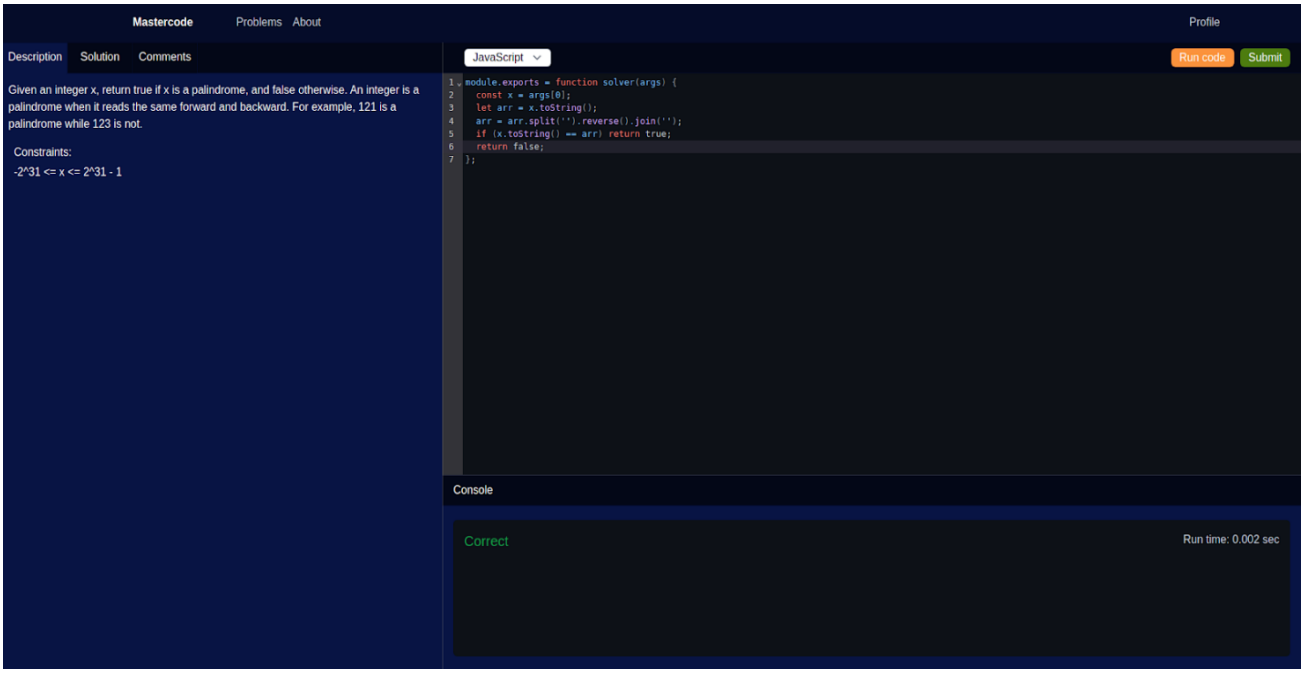

Рисунок 3.11 - Сторінка обраної задачі з коректно вирішеним завданням мовою “JavaScript”

Шостий приклад:

Неправильно вирішимо обрану задачу, написавши некоректне код-рішення у вбудований редактор та натиснувши кнопку “Run code” (рис. 3.12).

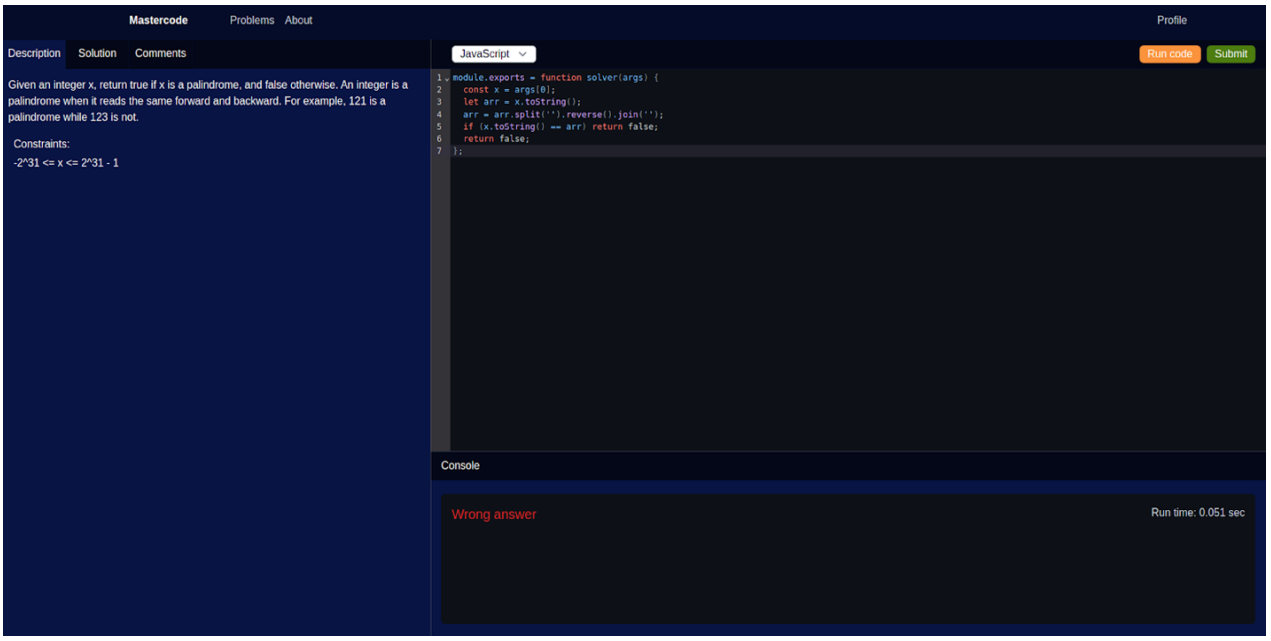

Рисунок 3.12 - Сторінка обраної задачі з некоректно вирішеним завданням

Як бачимо з рисунку, в терміналі з’явився текст червоного кольору “Wrong answer” (неправильна відповідь), що сигналізує про некоректне вирішення задачі.

Сьомий приклад:

Допустимо синтаксичну помилку в коді, при написанні рішення завдання та натиснемо кнопку “Run code” (рис. 3.13).

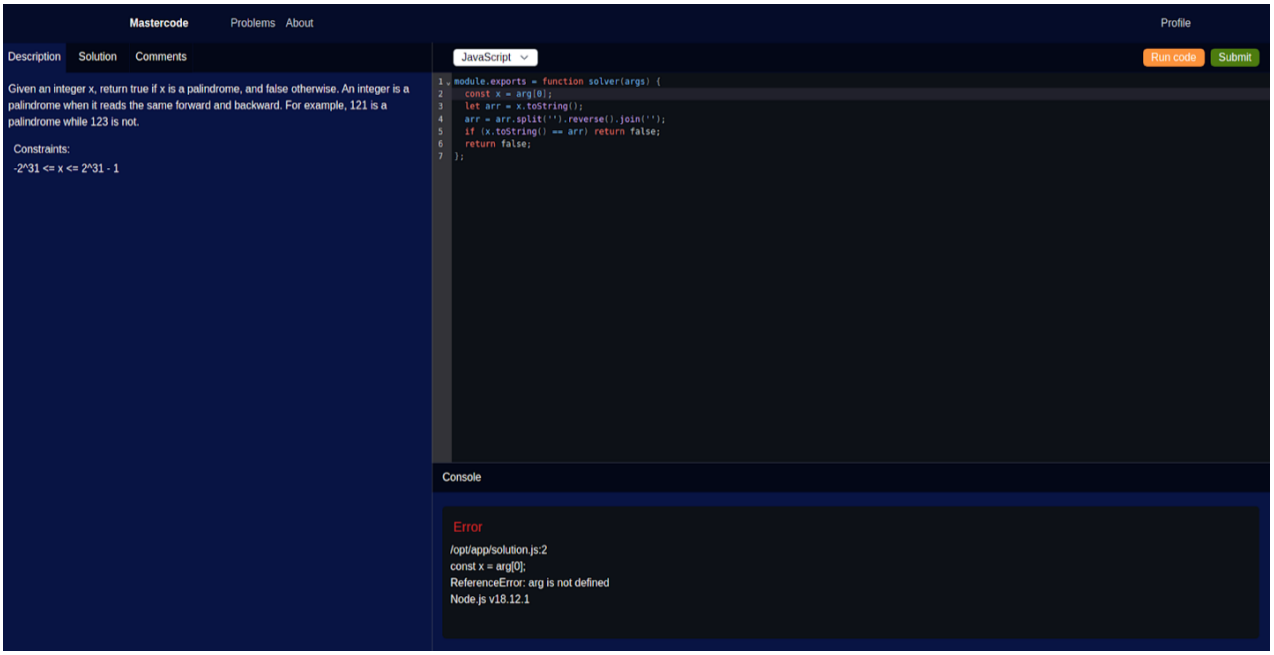

Рисунок 3.13 - Сторінка обраної задачі з синтаксичною помилкою в коді

Як бачимо з рисунку, в терміналі з’явився текст червоного кольору “Error” (помилка), що сигналізує про помилку в коді користувача, а під ним інформація про саму помилку.

Восьмий приклад:

У попередніх прикладах обраною мовою розв’язання задач була “JavaScript”, для даного прикладу оберемо мову програмування “Python” та коректно вирішимо обрану задачу (рис. 3.14).

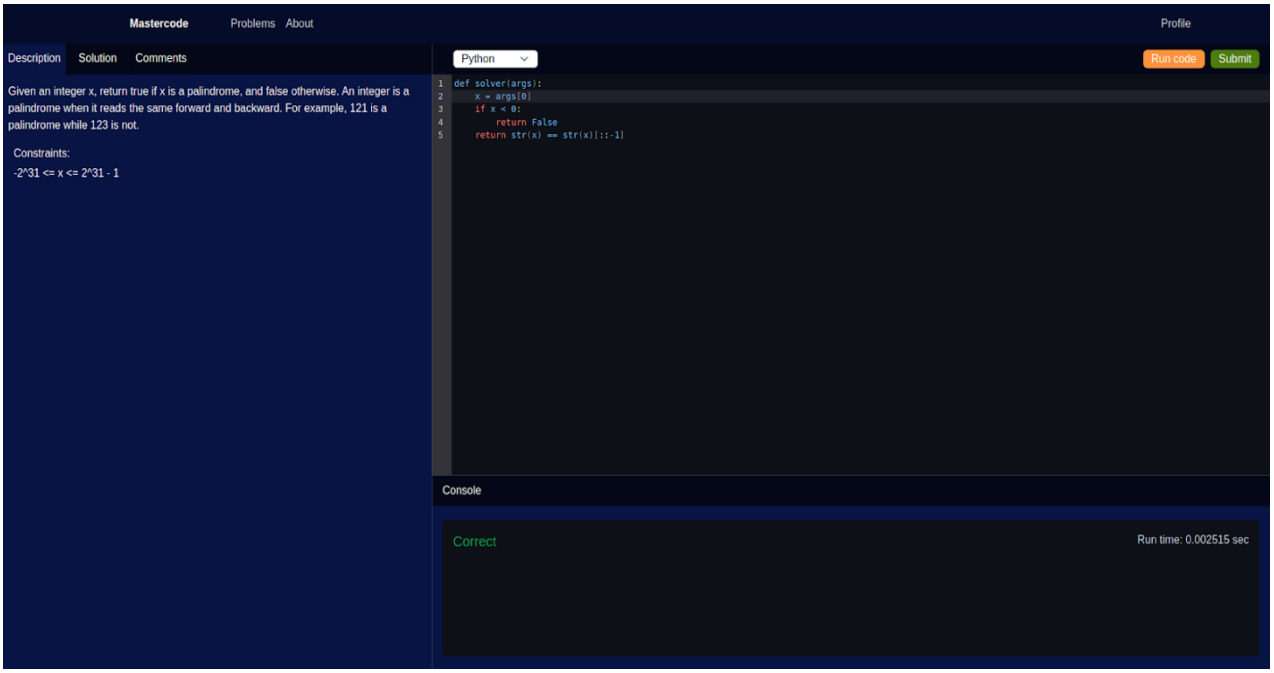

Рисунок 3.14 - Сторінка обраної задачі з коректно вирішеним завданням мовою “Python”

Дев’ятий приклад:

Додамо коментар до обраної задачі, перейшовши на вкладку “Comments” та написавши якесь повідомлення у текстовій області (рис. 3.15, 3.16).

Коментар був успішно доданий, він містить інформацію про користувача, який є автором даного коментаря та час коли він був доданий, також наявна можливість редагування та видалення коментаря.

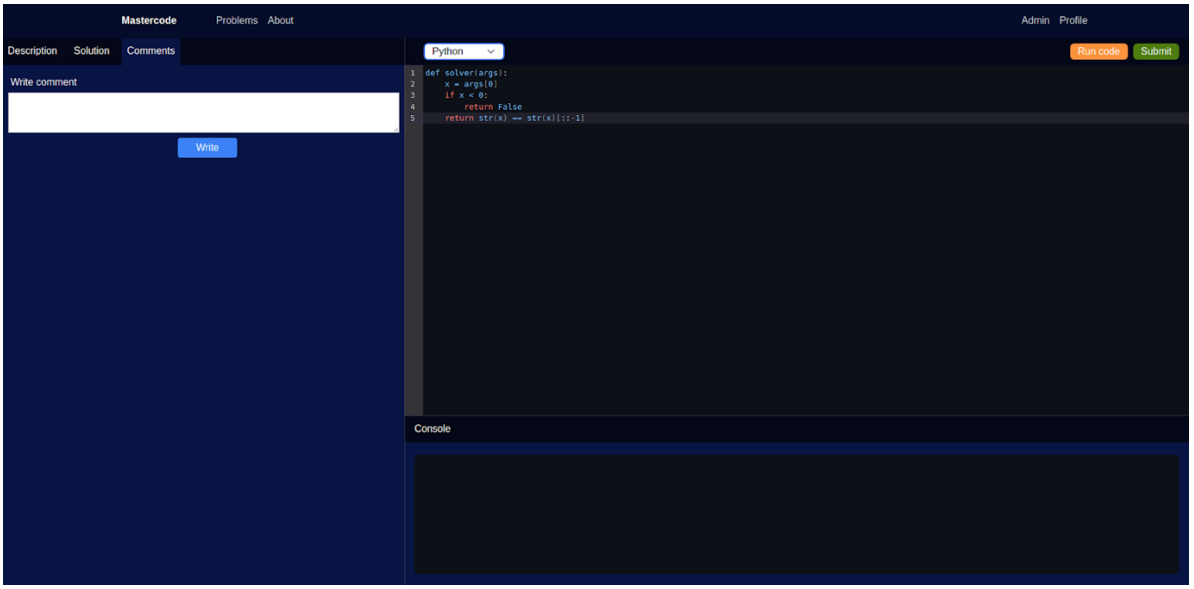

Рисунок 3.15 - Відкрита вкладка коментарів, без наявних записів

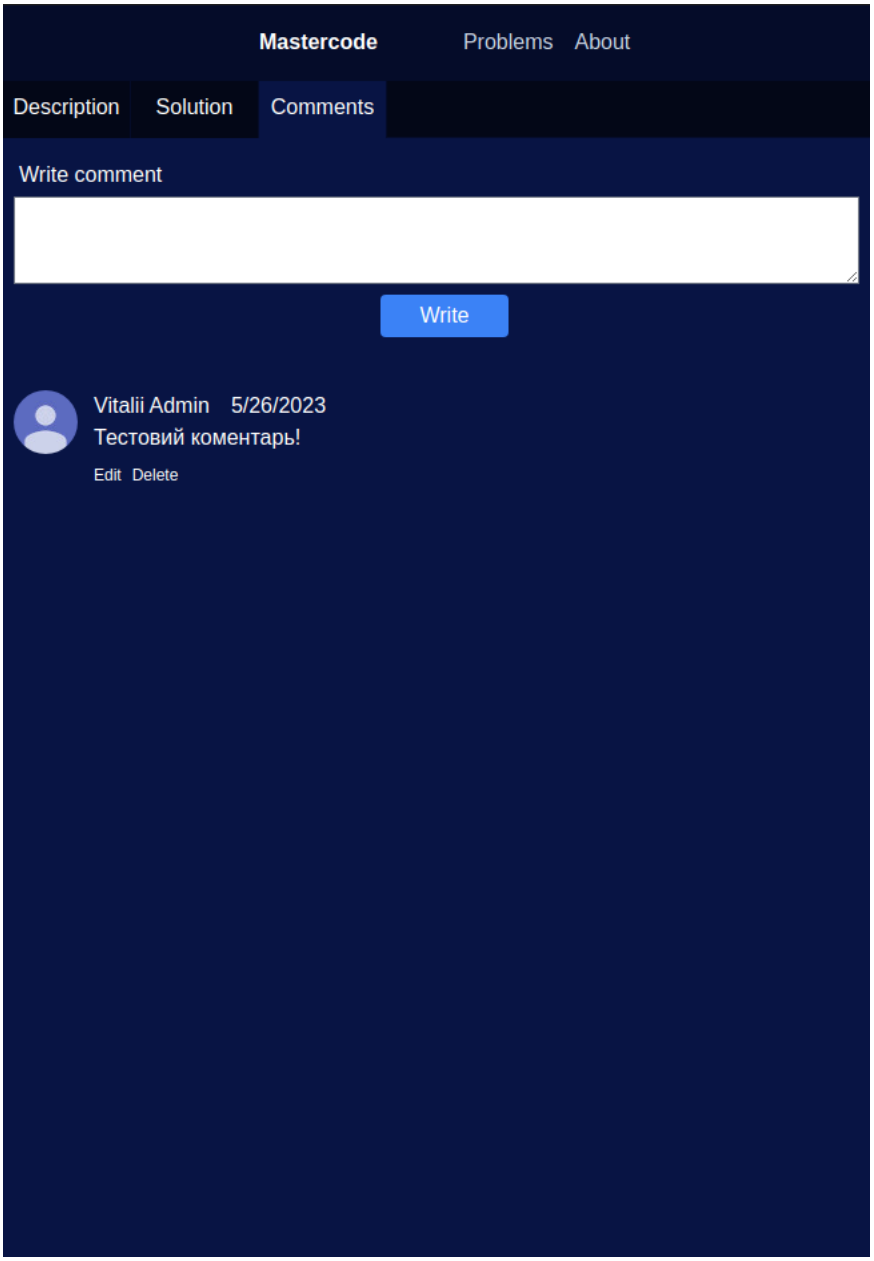

Рисунок 3.16 - Відкрита вкладка коментарів, з доданим записом

Десятий приклад:

Перевіримо функціонал, що наявний на сторінці адміністратора і є доступний користувачам, що володіють ролью “admin”. Він в себе включає перегляд наявних задач на сайті, додавання нових задач, редагування та видалення старих, також наявна можливість перегляду інформації про користувачів, що зареєстровані у системі. Спершу додамо на сайт нову задачу (рис. 3.17).

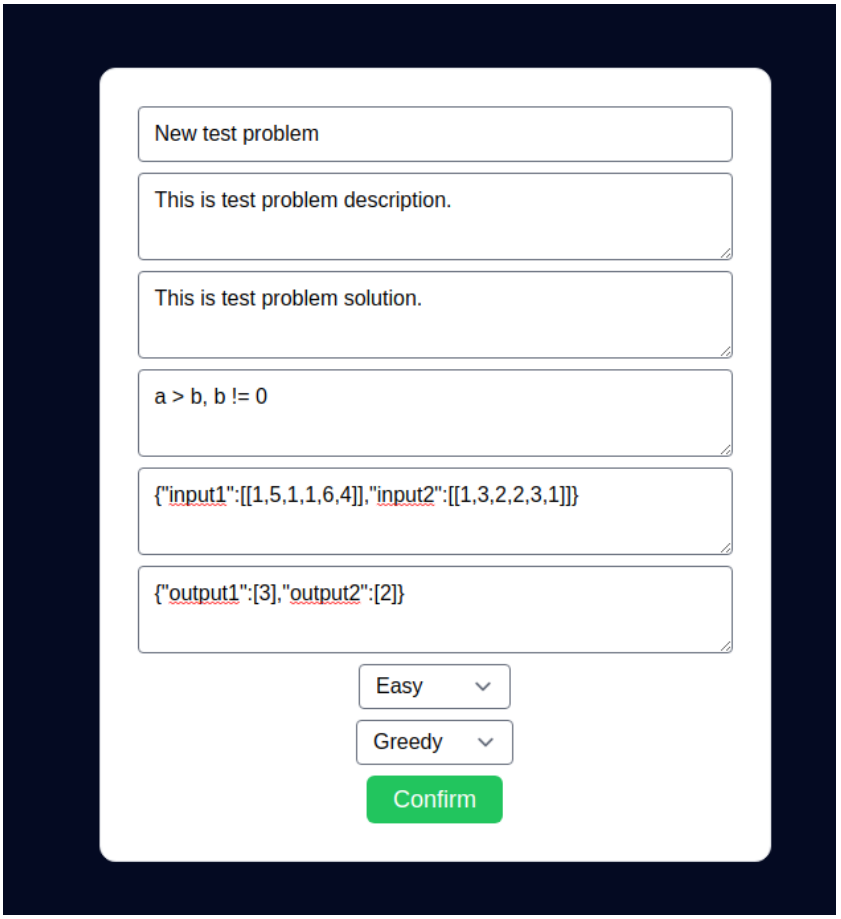

Рисунок 3.17 - Форма для додавання задачі

Переглянемо список наявних задач та переконаємося, що він містить задачу, яка була тільки що додана (рис. 3.18).

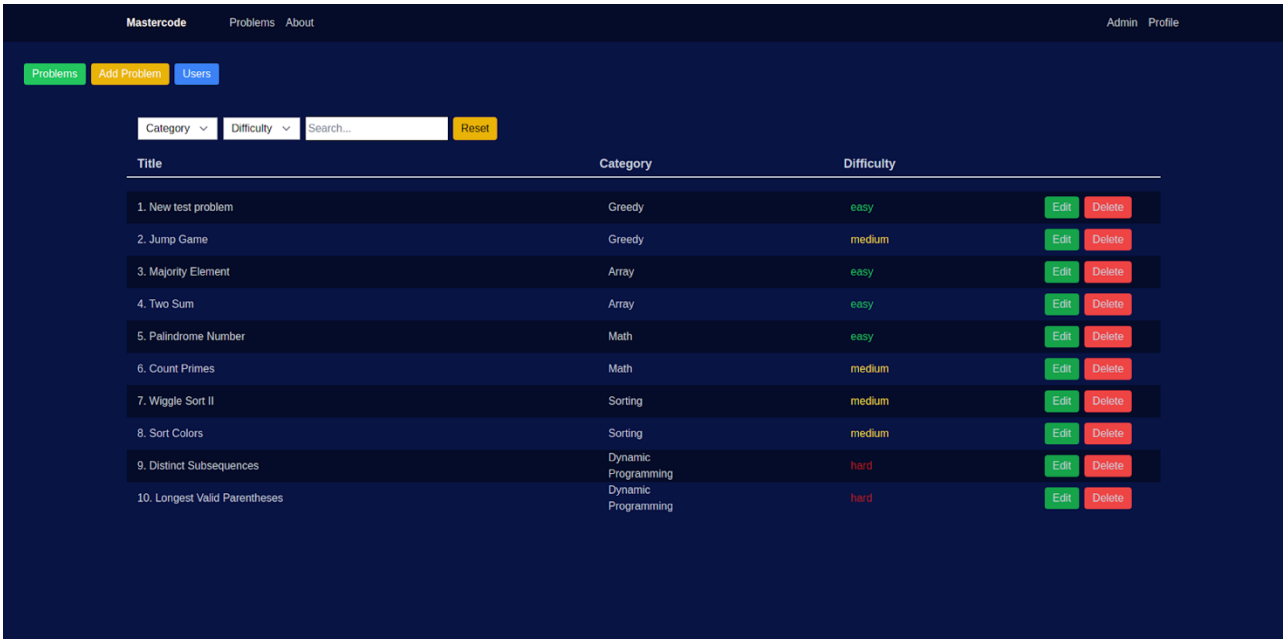

Рисунок 3.18 - Список усіх задач, серед яких наявна нова задача

Бачимо, що нова задача була успішно додана до системи та відображається на першому місці.

Відредагуємо дану задачу, натиснувши біля неї кнопку “Edit” та змінивши для неї категорію та складність (рис. 3.19).

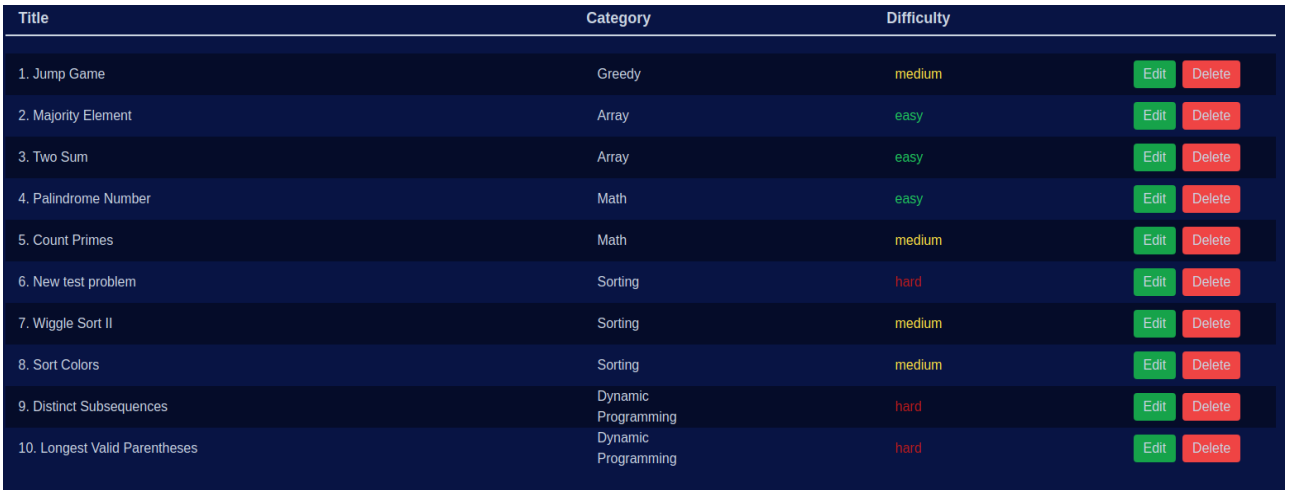

Рисунок 3.19 - Список усіх задач з редагованою задачею

Наостанок перевіримо роботу функції по видаленню задачі, натиснемо на кнопку “Delete” поряд з потрібною задачею та підтвердимо видалення у сповіщенні, що з’явиться на екрані. Наразі наявно 10 задач у системі, після видалення мусить залишитися 9 (рис. 3.20).

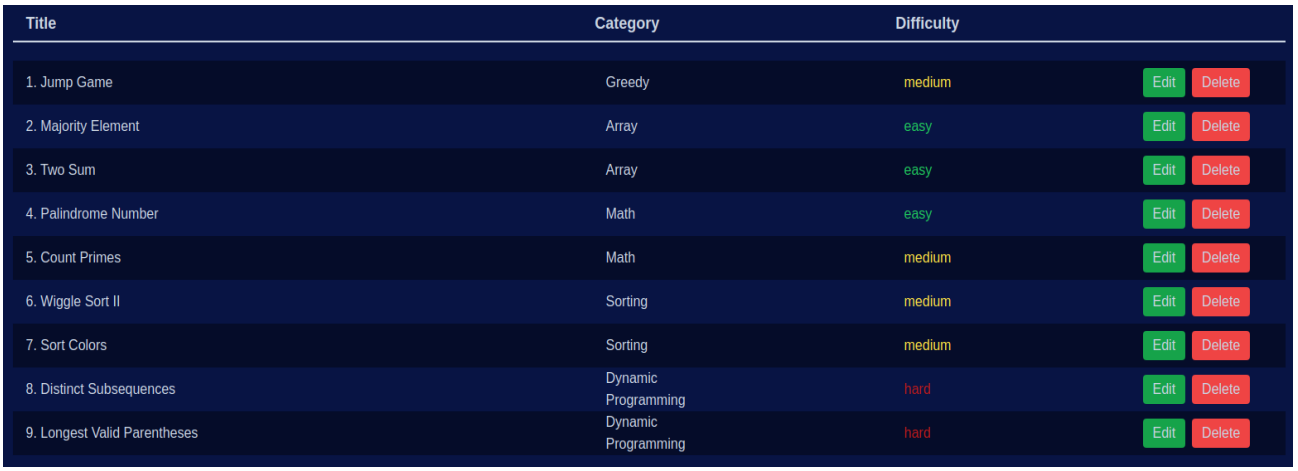

Рисунок 3.20 - Успішне видалення задачі

Остання функція, яку залишилося оглянути на сторінці адміністратора, це можливість переглядати інформацію про зареєстрованих користувачів, для її перегляду потрібно натиснути на кнопку “Users” (рис. 3.21).

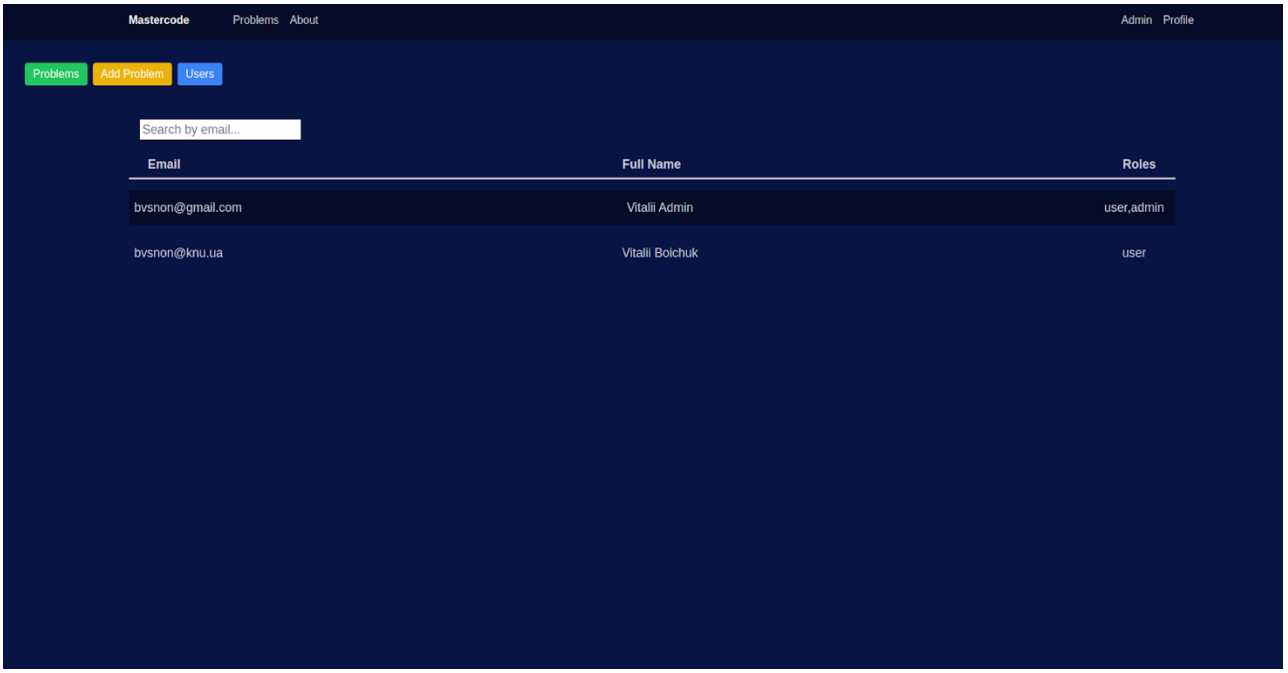

Рисунок 3.21 - Список зареєстрованих користувачів

Проаналізувавши дані тестові приклади, можемо зробити висновок, що застосунок поводить себе так, як і було заплановано перед його розробкою, ніякої непередбачуваної поведінки при даних тестах не було виявлено.

Наведемо схему роботи етапу перевірки на сервері коду-рішення завдання користувача, використавши діаграму діяльності (рис. 3.22).

Дана схема містить п’ять виконавців потрібних дій, між якими і перетікають усі потоки роботи алгоритму перевірки рішення завдання, а саме: “CompilerController”, “CompilerService”, “CompilerClass”, “Docker”, “Executor”.

Процес починається у компоненті “CompilerController”, коли приходить запит від клієнта на перевірку рішення, у даному компоненті викликається дія GetUserData, яка формує потрібні дані (ідентифікатор користувача, дані про рішення користувача) для делегування завдання у наступний компонент “CompilerService” та його метод “ServiceCompile”, який викликає дію

“FindProblem” для отримання всієї інформації про задачу, яку вирішував користувач та після цього формуються усі остаточні дані для перевірки.

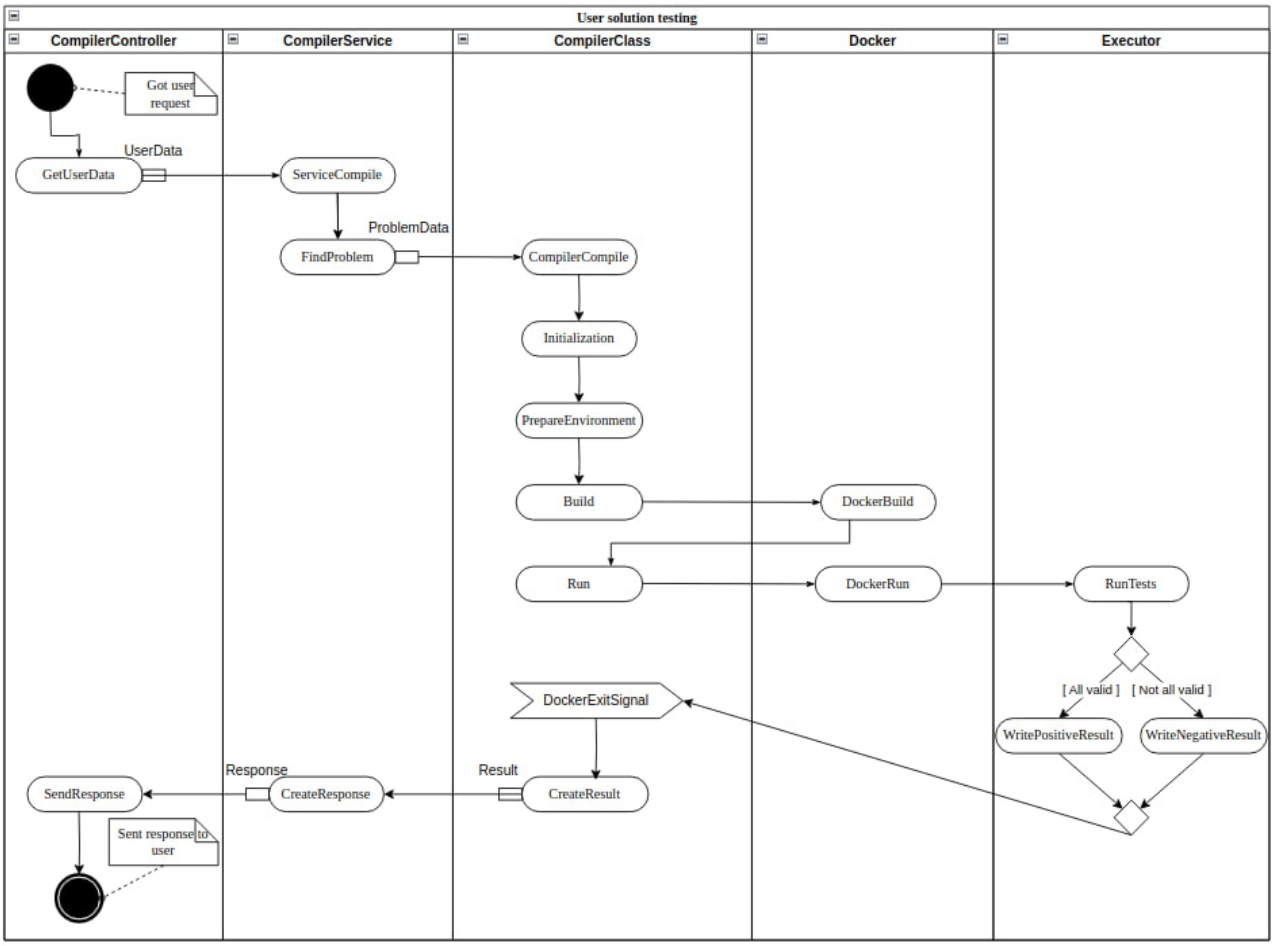

Рисунок 3.22 - Схема роботи перевірки коду-рішення користувача

Після формування остаточних даних для перевірки, процес переходить у компонент “CompilerClass”, який виконує дію “Initialization”, потрібну для ініціалізації початкових даних (наприклад прописання унікальних шляхів для потрібних файлів та папок), за цим слідує дія “PrepareEnvironment”, задача якої створити необхідні умови для запуску коду користувача та його перевірки. Далі на виконання спрацьовує команда “Build”, що викликає іншу команду “DockerBuild” у компоненті “Docker” для створення образу з файлом потрібної мови, у якому міститься рішення користувача, після її завершення потік повертається знову до компоненту “CompilerClass”, де викликається дія “Run”, яка передає управління на виконання знову компоненту “Docker” та методу

“DockerRun”, задача якої запустити контейнер з створеним раніше образом на виконання. Коли контейнер запущений, потік переходить до останнього елементу даної системи, а саме “Executor”, задача якого всередині контейнеру запустити функцію “RunTests”, яка запустить усі потрібні тести на виконання і якщо хоча б один тест провалиться або виникне помилка, то спрацює дія “WriteNegativeResult”, що сформує дані про негативний результат перевірки, інакше спрацює дія “WritePositiveResult”, яка сформує дані про успішність пройдення всіх тестів. На цьому робота “Executor” завершується, а отже завершується робота контейнера, на що спрацює дія очікування про даний сигнал завершення роботи контейнеру “DockerExitSignal” у компоненті “DockerClass”. Наступною виконається дія “CreateResult” яка сформує дані про результат перевірки тестів та передасть його команді “CreateResponse”, яка вже остаточно сформує відповідь для користувача у компоненті “CompilerService” та передасть потік у компонент де все починалося, а саме у “CompilerController”, який викличе дію “SendResponse”, яка відправить результат перевірки користувача, на чому завершиться робота системи.

Висновки до третього розділу

Отже, в результаті виконання третього розділу дипломної роботи, було описано та обґрунтовано вибір інструментальних засобів для розробки продукту, були написані тест-кейси для створеної системи, проведено тестування та детальний аналіз роботи веб-застосунку тренувального майданчика для вирішення задач з програмування.

ВИСНОВКИ

У ході виконання дипломної роботи, було розроблено веб-застосунок тренувальний майданчик, для вирішення задач з програмування, що складається з серверної та клієнтської частини.

Було проведено аналіз проблеми, яку потрібно було вирішити, обґрунтовано її актуальність в наш час. Порівняно найкращі на даний момент альтернативні платформи на ринку, котрі користуються високим попитом серед програмістів усього світу. Здійснено функціональний аналіз, визначено як функціональні так і нефункціональні вимоги до застосунку, поставлено чітку задачу та описано алгоритм її розв’язку.

Визначено основні бізнес-процеси предметної області та головні сценарії використання застосунку, проведено функціональне моделювання його роботи. Побудовано програмну архітектуру розроблювального застосунку та узагальнену технологію роботи з ним, спроектовано базу даних для зберігання задач по вирішенню, а також розроблено макети сторінок інтерфейсу.

Для практичної реалізації веб-застосунку, спочатку, були підібрані, описані та обґрунтовані інструментальні засоби, що підходили під вирішення даної задачі. Описано тест-кейси для створеної системи, проведено детальний тестовий аналіз функціонування застосунку, який був успішно пройдений.

В підсумку, зазначимо, що даний веб-застосунок може бути успішно використаний, як навчальна платформа для новачків в програмуванні, а також більш досвідчених спеціалістів, котрі прагнуть покращити свої вміння вирішувати різні задачі з програмування, адже ріст кількості людей, що починають свою кар’єру програмістами у всьому світі стрімко зростає. Також, розроблена система має всі властивості для того, аби бути покращеною та розвинутою в багатьох аспектах, як наприклад, в доданні до неї підтримки рішення задач більшою кількістю мов програмування та покращенням безпекових функцій з запуску й аналізу коду користувачів, чи імплементуванні функціоналу рейтингової системи серед користувачів.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Офіційний сайт “Node.js”. Документація “Node.js”. [Електронний ресурс] Посилання: https://nodejs.org/en/
2. Офіційний сайт “TypeScript”. Документація “TypeScript”. [Електронний ресурс] Посилання: https://www.typescriptlang.org/
3. Офіційний сайт “Next.js”. Документація “Next.js”. [Електронний ресурс] Посилання: https://nextjs.org/
4. Офіційний сайт “Nest.js”. Документація “Nest.js”. [Електронний ресурс] Посилання: https://nestjs.com/
5. Офіційний сайт “Docker”. Документація “Docker”. [Електронний ресурс] Посилання: https://www.docker.com/
6. Офіційний сайт “Tailwindcss”. Документація “Tailwindcss”. [Електронний ресурс] Посилання: https://tailwindcss.com/
7. Офіційний сайт “TypeORM”. Документація “TypeORM”. [Електронний ресурс] Посилання: https://typeorm.io/
8. JavaScript: The Definitive Guide Seventh Edition by David Flanagan, 720p. ISBN 978-5-907203-79-2.
9. "CODEKER - CONTAINERIZED COMPILER USING DOCKER", International Journal of Emerging Technologies and Innovative Research, ISSN:2349-5162, Vol.6, Issue 3, page no. pp 515-519: http://www.jetir.org/papers/JETIR1903977.pdf.
10. Офіційний сайт “Figma”. Документація “Figma”. [Електронний ресурс] Посилання: https://help.figma.com/hc/en-us
11. Офіційний сайт “Archi”. Документація “Archi”. [Електронний ресурс] Посилання: https://www.archimatetool.com/
12. Crow's Foot Notation – Relationship Symbols And How to Read Diagrams. [Електронний ресурс] Посилання: http://surl.li/hkmvt
13. Офіційний сайт Redis. [Електронний ресурс] Посилання: https://redis.io/

ДОДАТОК

Програмний код основних модулів

```
import { Problem } from 'src/problem/entities/problem.entity';
import { TLanguage } from 'src/user/entities/user-solved-problem.entity';
import { ResponseCompilerDto, Verdict } from './dto/response-compiler.dto';
import { CodeType } from './dto/compiler.dto';
import * as fsp from 'node:fs/promises';
import { randomUUID } from 'node:crypto';
import { join } from 'node:path';
import { spawn } from 'node:child_process';
export class Compiler {
private readonly code: CodeType;
private readonly lang: TLanguage;
private readonly problem: Problem;
private readonly imageName: string;
private readonly containerName: string;
private readonly dockerCompilerDir: string;
private readonly uniqueUserSolutionDockerImgDir: string;
private readonly solutionResultDirInDockerImgDir: string;
private readonly solutionResultDirInDocker: string;
private readonly solutionResultDirInHostFS: string;
private readonly solutionResultFileInHostFS: string;
private readonly solutionFileName: string;
private readonly solutionFilePath: string;
private readonly testcasesInputsJSON: string;
private readonly testcasesOutputsJSON: string;
private readonly testcasesInputsFilePath: string;
private readonly testcasesOutputsFilePath: string;
private result: ResponseCompilerDto;


constructor(code: CodeType, lang: TLanguage, problem: Problem) {
this.code = code;
this.lang = lang;
```

```
this.problem = problem;
this.imageName = randomUUID();
this.containerName = randomUUID();
this.dockerCompilerDir = join(process.cwd(), 'docker_compiler', this.lang);
this.uniqueUserSolutionDockerImgDir = join(process.cwd(), '..', 'images', randomUUID());
this.solutionResultDirInDockerImgDir = join(this.uniqueUserSolutionDockerImgDir, 'result');
this.solutionResultDirInDocker = join('/opt', 'app', 'result');
this.solutionResultDirInHostFS = join(process.cwd(), '..', 'results', this.lang, randomUUID());
this.solutionResultFileInHostFS = join(this.solutionResultDirInHostFS, 'result.txt');
this.solutionFileName = `solution.${this.lang}`;
this.solutionFilePath = join(this.uniqueUserSolutionDockerImgDir, this.solutionFileName);
this.testcasesInputsJSON = JSON.stringify(this.problem.inputs);
this.testcasesOutputsJSON = JSON.stringify(this.problem.outputs);
this.testcasesInputsFilePath = join(
this.uniqueUserSolutionDockerImgDir,
'testcasesInputs.json',
);
this.testcasesOutputsFilePath = join(
this.uniqueUserSolutionDockerImgDir,
'testcasesOutputs.json',
);
}
private async prepareEnv(): Promise<void> {
await fsp.mkdir(this.uniqueUserSolutionDockerImgDir, { recursive: true });
await fsp.cp(this.dockerCompilerDir, this.uniqueUserSolutionDockerImgDir, { recursive: true });
await fsp.mkdir(this.solutionResultDirInDockerImgDir, { recursive: true });
await fsp.mkdir(this.solutionResultDirInHostFS, { recursive: true });

await fsp.writeFile(this.testcasesInputsFilePath, this.testcasesInputsJSON);
await fsp.writeFile(this.testcasesOutputsFilePath, this.testcasesOutputsJSON);
await fsp.writeFile(this.solutionFilePath, Buffer.from(this.code.data));
}

private async clear(): Promise<void> {
await fsp.rm(this.uniqueUserSolutionDockerImgDir, { recursive: true, force: true });
```

```
await fsp.rm(this.solutionResultDirInHostFS, { recursive: true, force: true });
}

public async compile(): Promise<ResponseCompilerDto> {
await this.prepareEnv();
return new Promise((resolve, reject) => {
const logData: Buffer[] = []; // currently unused in response
const errData: Buffer[] = [];

const dockerBuild = spawn('docker', [
'build',
this.uniqueUserSolutionDockerImgDir,
'-t',
this.imageName,
]);


dockerBuild.on('exit', () => {
const dockerRun = spawn('docker', [
'run',
'--name',
this.containerName,
'-v',
`${this.solutionResultDirInHostFS}:${this.solutionResultDirInDocker}`,
this.imageName,
]);

dockerRun.stdout.on('data', (chunk) => logData.push(chunk));

dockerRun.on('exit', async () => {
if (errData.length) {
const logs = Buffer.concat(errData).toString();
const modifiedLogs = logs
.split('\n')
.filter((s) => !s.includes('at'))
.join()
```

```
.replace(/  +/g, ' ');
this.result = new ResponseCompilerDto(Verdict.Error, modifiedLogs);
} else {
const logs = Buffer.concat(logData).toString();
const [resultData, runTime] = (
await fsp.readFile(this.solutionResultFileInHostFS, 'utf-8')
).split('\n');
const verdict = resultData.includes(Verdict.Accepted)
? Verdict.Accepted
: Verdict.WrongAnswer;
this.result = new ResponseCompilerDto(verdict, logs, runTime);
}
resolve(this.result);

const rmDocContainer = spawn('docker', ['rm', this.containerName]);
rmDocContainer.on('exit', () => spawn('docker', ['rmi', this.imageName]));

await this.clear();
});

dockerRun.stderr.on('data', (errChunk) => errData.push(errChunk));
dockerRun.stderr.on('end', async () => await this.clear());

dockerRun.on('error', async (err) => {
await this.clear();
reject(err);
});
});

dockerBuild.stderr.on('data', async (errChunk) => {
await this.clear();
reject(errChunk.toString());
});

dockerBuild.on('error', async (err) => {
await this.clear();
```

```
reject(err);
});
});
}
}
import { Injectable } from '@nestjs/common';
import { InjectRepository } from '@nestjs/typeorm';
import { EntityNotFoundCustomError } from 'src/errors/custom-errors';
import { File } from 'src/file/entities/file.entity';
import { FileService } from 'src/file/file.service';
import { MFile } from 'src/file/mfile.class';
import { DataSource, DeepPartial, FindManyOptions, FindOptionsWhere, Repository } from 'typeorm';
import { User } from './entities/user.entity';
import { USER_NOT_FOUND_ERROR } from './user.constants';
import { ProblemService } from 'src/problem/problem.service';
import { PROBLEM_NOT_FOUND_ERROR } from 'src/problem/problem.constants';
import { Problem } from 'src/problem/entities/problem.entity';
import { UserSolvedProblem } from './entities/user-solved-problem.entity';
import { AddSolvedProblemDto } from './dto/add-solved-problem.dto';
import { UserQueryDto } from './dto/user-query.dto';

@Injectable()
export class UserService {
  constructor(
    @InjectRepository(User)
    private readonly userRepository: Repository<User>,
    @InjectRepository(UserSolvedProblem)
    private readonly userSolvedProblemRepository: Repository<UserSolvedProblem>,
    private readonly problemService: ProblemService,
    private readonly fileService: FileService,
    private readonly dataSource: DataSource,
  ) {}

  async create(data: DeepPartial<User>): Promise<User> {
    const user = this.userRepository.create(data);
    return this.userRepository.save(user);
  }

  async findMany(options: UserQueryDto): Promise<User[]> {
    const { skip, take, email } = options;
    return this.userRepository.find({ skip, take, where: { email } });
  }

  async findOne(where: FindOptionsWhere<User>): Promise<User | null> {
    return this.userRepository.findOneBy(where);
  }

  async findOneOrThrow(where: FindOptionsWhere<User>): Promise<User> {
    const user = await this.userRepository.findOneBy(where);
    if (!user) throw new EntityNotFoundCustomError(USER_NOT_FOUND_ERROR);
```

```
    return user;
  }

  async updateOne(where: FindOptionsWhere<User>, data: DeepPartial<User>): Promise<User> {
    const user = await this.findOneOrThrow(where);
    return this.userRepository.save({ ...user, ...data });
  }

  async updateMany(options: FindManyOptions<User>, data: DeepPartial<User>): Promise<User[]> {
    const users = await this.findMany(options);
    if (!users.length) return [];
    return this.userRepository.save(users.map((user: User) => ({ ...user, ...data })));
  }

  async remove(where: FindOptionsWhere<User>): Promise<User | null> {
    const user = await this.findOneOrThrow(where);
    return this.userRepository.remove(user);
  }

  async uploadAvatar(file: Express.Multer.File, userId: string): Promise<File> {
    const user = await this.findOneOrThrow({ id: userId });
    if (user.avatar_id) await this.removeAvatar(userId);

    const bufferWebP = await this.fileService.convertToWebP(file.buffer);
    const filename = `avatar-${userId}.webp`;
    const mimetype = 'image/webp';
    const mfile = new MFile(filename, mimetype, bufferWebP);
    const folderToSave = 'avatars';
    const avatar = (await this.fileService.saveFiles([mfile], folderToSave))[0];

    user.avatar_id = avatar.id;
    await this.userRepository.save({ ...user });
    return avatar;
  }

  async getUserAvatar(userId: string): Promise<File | null> {
    const user = await this.findOneOrThrow({ id: userId });
    if (!user.avatar_id) return null;
    return this.fileService.getFileById(user.avatar_id);
  }

  async removeAvatar(userId: string): Promise<void> {
    const queryRunner = this.dataSource.createQueryRunner();
    const user = await this.findOneOrThrow({ id: userId });
    if (!user.avatar_id) return;

    await queryRunner.connect();
    await queryRunner.startTransaction();
    try {
      await queryRunner.manager.update(User, { id: userId }, { ...user, avatar_id: null });
      await this.fileService.removeFileWithQueryRunner(user.avatar_id, queryRunner);
      await queryRunner.commitTransaction();
```

```
    } catch (e) {
      await queryRunner.rollbackTransaction();
      throw e;
    } finally {
      await queryRunner.release();
    }
  }

  async addSolvedProblem(data: AddSolvedProblemDto): Promise<void> {
    // const user = await this.findOne({ id: data.user_id });
    // if (!user) throw new EntityNotFoundCustomError(USER_NOT_FOUND_ERROR);
    // const problem = await this.problemService.findOne({ id: data.problem_id });
    // if (!problem) throw new EntityNotFoundCustomError(PROBLEM_NOT_FOUND_ERROR);

    const solvedProblem = this.userSolvedProblemRepository.create({ ...data });
    await this.userSolvedProblemRepository.save(solvedProblem);
  }

  async getSolvedProblems(userId: string): Promise<UserSolvedProblem[]> {
    return this.userSolvedProblemRepository.find({
      where: { user_id: userId },
      relations: {
        problem: true,
      },
    });
  }
}


import { Injectable } from '@nestjs/common';
import { InjectRepository } from '@nestjs/typeorm';
import { DeepPartial, FindOptionsWhere, Repository } from 'typeorm';
import { Problem } from './entities/problem.entity';
import { ProblemReaction, TReactionType } from './entities/problem-reaction.entity';
import { EntityNotFoundCustomError } from 'src/errors/custom-errors';
import { PROBLEM_NOT_FOUND_ERROR } from './problem.constants';
import { ToggleReactionResponseDto } from './dto/toggle-reaction-response.dto';
import { ProblemQueryDto } from './dto/problem-query.dto';

@Injectable()
export class ProblemService {
  constructor(
    @InjectRepository(Problem)
    private readonly problemRepository: Repository<Problem>,
    @InjectRepository(ProblemReaction)
    private readonly problemReactionRepository: Repository<ProblemReaction>,
  ) {}

  async create(data: DeepPartial<Problem>): Promise<Problem> {
    const problem = this.problemRepository.create(data);
    return this.problemRepository.save(problem);
  }
```

```
  async findMany(options: ProblemQueryDto): Promise<Problem[]> {
    const { skip, take, category, difficulty, title } = options;
    return this.problemRepository.find({
      skip: skip,
      take: take,
      where: {
        category: { name: category },
        difficulty,
        title,
      },
      relations: {
        category: true,
      },
    });
  }

  async findOne(where: FindOptionsWhere<Problem>): Promise<Problem | null> {
    return this.problemRepository.findOneBy(where);
  }

  async findOneOrThrow(where: FindOptionsWhere<Problem>): Promise<Problem> {
    const problem = await this.problemRepository.findOneBy(where);
    if (!problem) throw new EntityNotFoundCustomError(PROBLEM_NOT_FOUND_ERROR);
    return problem;
  }

  async updateOne(where: FindOptionsWhere<Problem>, data: DeepPartial<Problem>): Promise<Problem>
{
    const problem = await this.findOneOrThrow(where);
    return this.problemRepository.save({ ...problem, ...data });
  }

  async remove(where: FindOptionsWhere<Problem>): Promise<Problem> {
    const problem = await this.findOneOrThrow(where);
    return this.problemRepository.remove(problem);
  }
```

```
'use client';

import React, { FC, useEffect, useState } from 'react';
import { Allotment } from 'allotment';
import CodeEditor from './CodeEditor';
import ManagePanel from './ManagePanel';
import { LOCAL_STORAGE_CODE_KEY } from './problem.constants';
import fetcher from '../../../utils/fetcher';
import { IResponseCompiler, Verdict } from './compiler.d';
import SpinLoader from './SpinLoader';
import useAxiosAuth from '../../../lib/hooks/useAxiosAuth';
import { Problem } from '../../../models/problem/problem.type';
import ProblemService from '../../../services/problem.service';
import CompilerService, { TLanguage } from '../../../services/compiler.service';
import Comments from './Comments';
```

```
interface ProblemPageContent {
  id: string,
}

enum SelectedPanel {
  Desc,
  Sol,
  Comments,
}

const ProblemPageContent: FC<ProblemPageContent> = ({ id }) => {
  const axiosAuth = useAxiosAuth();
  ProblemService.axiosAuth = axiosAuth;
  CompilerService.axiosAuth = axiosAuth;

  const [problem, setProblem] = useState<Problem>();

  useEffect(() => {
    fetchProblem();
  }, []); // eslint-disable-line react-hooks/exhaustive-deps

  const fetchProblem = async () => {
    try {
      const res = await ProblemService.findOne(id);
      setProblem(res.data);
    } catch (e: any) {
      console.log(e);
    }
  }

  const storedCodeBuffer = Buffer.from(localStorage.getItem(LOCAL_STORAGE_CODE_KEY +
problem?.id) || []);

  const [colorDesc, setColorDesc] = useState<string>('bg-main-blue');
  const [colorSol, setColorSol] = useState<string>('bg-command-blue');
  const [colorComments, setcolorComments] = useState<string>('bg-command-blue');
  const [selectedPanel, setSelectedPanel] = useState<SelectedPanel>(SelectedPanel.Desc);
  const [code, setCode] = useState<Buffer>(storedCodeBuffer);
  const [lang, setLang] = useState<TLanguage>('js');
 const [compilerResponse, setCompilerResponse] = useState<IResponseCompiler>();
  const [disable, setDisable] = useState<boolean>(false);
  const [btnRunStyle, setBtnRunStyle] = useState<string>('hover:bg-orange-500');
  const [btnSubmitStyle, setBtnSubmitStyle] = useState<string>('hover:bg-lime-800');
  const [loading, setLoading] = useState<boolean>(false);

  const switchToDesc = (e) => {
    setSelectedPanel(SelectedPanel.Desc);
    setColorDesc('bg-main-blue');
    setColorSol('bg-command-blue');
    setcolorComments('bg-command-blue');
  };
```

```
const switchToSol = (e) => {
  setSelectedPanel(SelectedPanel.Sol);
  setColorSol('bg-main-blue');
  setColorDesc('bg-command-blue');
  setcolorComments('bg-command-blue');
};

const switchToComments = (e) => {
  setSelectedPanel(SelectedPanel.Comments);
  setcolorComments('bg-main-blue');
  setColorSol('bg-command-blue');
  setColorDesc('bg-command-blue');
};

const getCode = (usercode: string) => {
  const codeBuffer = Buffer.from(usercode);
  setCode(codeBuffer);
};

const fetchCompiledInfo = async (e) => {
  setDisable(true);
  setBtnRunStyle('opacity-60');
  setBtnSubmitStyle('opacity-60');
  setLoading(true);
  try {
    const response = await CompilerService.compile({
      lang,
      code,
      problemId: problem!.id,
      submit: e.currentTarget.id === 'submit' ? true : false,
    })
    setCompilerResponse(response.data);
    setDisable(false);
    setBtnRunStyle('hover:bg-orange-500');
    setBtnSubmitStyle('hover:bg-lime-800');
    setLoading(false);
  } catch (e) {
    setDisable(false);
    setBtnRunStyle('hover:bg-orange-500');
    setBtnSubmitStyle('hover:bg-lime-800');
    setLoading(false);
  }
};

return (
  <Allotment className='flex'>
    <Allotment.Pane snap minSize={400} preferredSize={650}>
      <ManagePanel>
        <div
          className={'h-11 border-t border-r border-nav-blue pt-2 cursor-pointer w-24 ' + colorDesc}
          onClick={switchToDesc}
```

```
            >
              <div className='text-center'>Description</div>
            </div>
            <div
              className={'h-11 border-t border-r border-nav-blue pt-2 cursor-pointer w-24 ' + colorSol}
              onClick={switchToSol}
            >
              <div className='text-center'>Solution</div>
            </div>
            <div
              className={'h-11 border-t border-r border-nav-blue pt-2 cursor-pointer w-24 ' +
colorComments}
              onClick={switchToComments}
            >
              <div className='text-center'>Comments</div>
            </div>
          </ManagePanel>
          {
            selectedPanel === SelectedPanel.Desc
              ?
              <div className='px-2 mt-3'>
                <div>{problem?.description}</div>
                <ul className='pt-4 pl-2 list-disk'>
                  <p>Constraints:</p>
                  {problem?.constraints?.map((e, i) => (
                    <li key={i} className='pt-1'><p>{e}</p></li>
                  ))}
                </ul>
              </div>
              :
             selectedPanel === SelectedPanel.Sol
              ?
              <div className='px-2 mt-3'>
                {problem?.solution}
              </div>
              :
              <div className='px-2 mt-3'>
                <Comments problemId={id} />
              </div>
          }
        </Allotment.Pane>
        <Allotment.Pane minSize={500}>
          <Allotment vertical>
            <Allotment.Pane minSize={300} className='flex flex-col'>
              <ManagePanel>
                <div className='w-full justify-between flex'>
                  <div className='ml-8 text-command-blue'>
                    <select
                      className='form-select rounded-md py-0'
                      onChange={e => setLang(e.target.value as TLanguage)} value={lang}>
                       <option value='js'>JavaScript</option>
                       <option value='py'>Python</option>
                    </select>
```

```
                </div>
                <div className='mr-4 justify-between w-56 flex'>
                  <button
                    id='run'
                    className={'bg-orange-400 rounded-md px-3 ml-12 ' + btnRunStyle}
                    onClick={fetchCompiledInfo}
                    disabled={disable}
                  >
                    Run code
                  </button>
                  <button
                    id='submit'
                    className={'bg-lime-700 rounded-md px-3 ' + btnSubmitStyle}
                    onClick={fetchCompiledInfo}
                    disabled={disable}
                  >
                    Submit
                  </button>
                </div>
              </div>
            </ManagePanel>
            <CodeEditor getCode={getCode} pageID={problem?.id} />
          </Allotment.Pane>
          <Allotment.Pane preferredSize={275} minSize={275} maxSize={275}>
            <ManagePanel>
              <span className='ml-4'>Console</span>
            </ManagePanel>
            <div className={'bg-github-dark min-w-fit h-[70%] mx-4 my-5 rounded-md pt-2 pl-2 pr-2 pb
overflow-x-hidden overflow-y-auto flex ' + (loading ? 'justify-center items-center' : '')}>
              {loading ? (
                <SpinLoader />
              ) : (
                compilerResponse?.verdict === Verdict.Accepted || compilerResponse?.verdict ===
Verdict.Correct
                  ? (
                    <div className='pl-2 pt-2 pr-2 w-full flex justify-between'>
                      <div className='text-xl text-green-600'>{compilerResponse?.verdict}</div>
                      <div className='text-slate-300'>Run time: {compilerResponse?.runTime} sec</div>
                    </div>
                  )
                  : compilerResponse?.verdict === Verdict.WrongAnswer
                    ? (
                      <div className='flex flex-col w-full'>
                        <div className='pl-2 pt-2 pr-2 flex justify-between'>
                          <div className='text-xl text-red-600'>{compilerResponse?.verdict}</div>
                          <div className='text-slate-300'>Run time: {compilerResponse?.runTime}
sec</div>
                        </div>
                        <div className='pt-2'>{compilerResponse?.logs}</div>
                      </div>
                    )
                    : compilerResponse?.verdict === Verdict.Error
                      ? (
```

```
                        <div className='flex flex-col'>
                          <div className='text-xl text-red-600 pl-2 pt-2'>{compilerResponse?.verdict}</div>
                          <div className='pt-2 pl-1 flex flex-col'>
                            {
                              compilerResponse?.logs
                                .split(',')
                                .filter((s) => !!s && s !== ' ^')
                                .map((s, i) => (
                                  <div key={i}>{s}</div>
                                ))
                            }
                          </div>
                        </div>
                      )
                      : (<></>)
                  )}
                </div>
              </Allotment.Pane>
            </Allotment>
          </Allotment.Pane >
        </Allotment >
    );
};

export default ProblemPageContent;
```