МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ЧЕРКАСЬКИЙ ДЕРЖАВНИЙ ФАХОВИЙ БІЗНЕС-КОЛЕДЖ
Циклова комісія комп’ютерної інженерії та інформаційних технологій

КВАЛІФІКАЦІЙНА РОБОТА

на тему

ПЛАТФОРМА ДЛЯ ПЕРЕВІРКИ КОДУ З АВТОМАТИЧНИМ ТЕСТУВАННЯМ АЛГОРИТМІЧНИХ ЗАДАЧ

Виконав:
студент групи 1П-22
спеціальності
121 Інженерія програмного забезпечення
Андрій ВАЛОВИЙ
Керівник:
Наталя ФАЛЬЧЕНКО

Черкаси 2026

ЗМІСТ

ВСТУП

Автоматизація контролю знань є одним з пріоритетних напрямків розвитку сучасної системи освіти у галузі інформаційних технологій. Процес підготовки фахівців за спеціальністю 121 Інженерія програмного забезпечення вимагає виконання значної кількості практичних та лабораторних робіт, пов’язаних із розробкою алгоритмів та написанням вихідного коду. Основним етапом перевірки таких завдань є верифікація коректності роботи створених студентами програм. Традиційні підходи до оцінювання, що базуються на усному захисті або надсиланні файлів через загальні канали зв’язку, демонструють низьку ефективність при роботі з великими групами здобувачів освіти.[7] Викладач змушений витрачати значний обсяг робочого часу на рутинне розгортання, компіляцію або інтерпретацію кожного окремого скрипту, що суттєво уповільнює надання зворотного зв’язку та вносить елемент суб’єктивності в процес виставлення оцінок.

Існуючі глобальні хмарні платформи для автоматичної перевірки коду мають закриту архітектуру, жорстко обмежений набір вбудованих завдань та орієнтовані на комерційний рекрутинг або підготовку до співбесід. Вони повністю залежать від стабільного підключення до мережі Інтернет та не надають безкоштовних інструментів для локального адміністрування, розширення бази тестів та адаптації контенту під поточні робочі програми конкретного навчального закладу.

Актуальність даної кваліфікаційної роботи обумовлена технічною необхідністю створення автономного, гнучкого інструменту для автоматизованої перевірки алгоритмічних задач, який можна безперешкодно інтегрувати в локальну інфраструктуру навчального закладу, масштабувати та модифікувати під будь-які дидактичні вимоги без прив’язки до зовнішніх комерційних сервісів.

Метою кваліфікаційної роботи є проєктування та реалізація локальної вебплатформи, яка забезпечує автоматичне тестування правильності

програмного коду на мові Python, здійснює його безпечну валідацію в ізольованому середовищі та фіксує точні метрики часу виконання завдань.

Для досягнення поставленої мети необхідно виконати такі завдання:

1. Провести системний аналіз предметної області та дослідити архітектурні особливості сучасних систем автоматичного оцінювання вихідного коду.
2. Сформулювати повний перелік функціональних та нефункціональних вимог до розроблюваного програмного забезпечення.
3. Розробити реляційну структуру бази даних для гнучкого керування категоріями, завданнями та фіксації результатів тестування.
4. Реалізувати серверну частину застосунку для безпечного виконання користувацьких скриптів через механізм ізольованих підпроцесів.
5. Розробити адаптивний клієнтський інтерфейс із вбудованим текстовим редактором та модулем динамічного відображення результатів перевірки.
6. Провести тестування створеної платформи

Об’єктом дослідження є система автоматизованої перевірки правильності програмного коду та оцінювання алгоритмічних завдань у вебсередовищі.

Предметом дослідження є методи проєктування та технології реалізації локальної вебплатформи для перевірки коду на мові Python.

Практична значущість отриманих результатів полягає у створенні повністю автономного програмного комплексу, який може бути використаний у навчальному закладі для оптимізації проведення практичних занять, зрізів знань та лабораторних робіт з програмування. Відкрита архітектура рішення дозволяє викладачам самостійно розширювати базу даних, додавати нові типи завдань будь-якої складності та повністю контролювати процес навчання.

Кваліфікаційна робота пройшли апробацію на XVIII Всеукраїнській науково-практичній конференції студентів, аспірантів та молодих вчених (23–24 квітня 2026 року, місто Черкаси) з доповіддю на тему: «Управління з’єднаннями з базою даних у Flask-застосунку для запобігання блокуванню SQLite».

РОЗДІЛ 1 ОГЛЯД ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Характеристика предметної області та аналіз процесів оцінювання коду

Сучасна парадигма освіти у галузі комп’ютерних наук та інженерії програмного забезпечення базується на безперервному отриманні практичних навичок. Процес формування професійних компетентностей майбутніх розробників вимагає систематичного розв’язання алгоритмічних задач різного рівня складності. Традиційні підходи до організації навчального процесу, які передбачають ручну перевірку вихідних текстів програм викладачем, наразі демонструють критичну неефективність. Головним деструктивним фактором є висока трудомісткість аналізу. Викладач змушений самостійно копіювати код кожного студента, розгортати відповідне середовище виконання, вручну задавати вхідні параметри та звіряти отримані результати з еталонними значеннями. Це призводить до суттєвих затримок у наданні зворотного зв’язку, через що втрачається дидактичний ефект від виконання лабораторної чи практичної роботи. Крім того, ручна перевірка в умовах великих академічних груп неминуче вносить елемент суб’єктивності, оскільки критерії оцінювання можуть несвідомо змінюватися під впливом людського фактора або втоми.

Для вирішення зазначених проблем в освітню практику впроваджуються системи автоматизованого тестування та оцінювання програмного коду.[8] Функціонування таких систем базується на методології динамічного тестування «чорної скриньки». Користувач надсилає вихідний текст програми через веб-інтерфейс, після чого система запускає цей код на серверній стороні з наперед визначеними наборами вхідних даних (тестами) та здійснює посимвольне або покрокове порівняння стандартного потоку виведення програми з очікуваним еталонним результатом. Автоматизація цього процесу дозволяє студенту миттєво дізнатися про помилки у логіці алгоритму та самостійно виправити їх до моменту остаточного захисту роботи.

Проте проектування та реалізація автоматичних суддів супроводжується серйозними технічними викликами, пов’язаними із безпекою інфраструктури та контролем ресурсів операційної системи. Пряме виконання стороннього програмного коду, написаного користувачами, створює пряму загрозу для серверної машини. Студентський скрипт може містити нескінченні цикли, які повністю блокують обчислювальні потужності процесора, або команди, що призводять до вичерпання доступної оперативної пам’яті. Більш небезпечним є навмисне або випадкове використання деструктивних системних викликів, спрямованих на видалення файлів, модифікацію конфігурації сервера або відкриття несанкціонованих мережевих з’єднань. Специфіка предметної області вимагає від розробника створення захищеного середовища, спроможного ізолювати процес виконання коду, жорстко обмежити час його роботи за допомогою тайм-аутів та безпечно перехоплювати всі системні винятки інтерпретатора.

Локальна веб-платформа повинна забезпечувати повну автономність роботи, мінімальні вимоги до апаратних ресурсів навчальної лабораторії та надавати простий механізм адміністрування. Головною особливістю проектування є реалізація концепції динамічного конструювання контексту виконання, коли надісланий студентом код не просто запускається як окремий файл, а невидимо для користувача склеюється із заздалегідь підготовленою структурою даних або набором змінних. Це дозволяє перевіряти чисті алгоритмічні навички без необхідності написання студентом громіздких модулів введення та виведення інформації.

1.2 Порівняльний аналіз наявних систем автоматичної перевірки коду

На світовому ринку інформаційних технологій існує декілька потужних хмарних платформ, які реалізують функціонал автоматичної перевірки програмного коду. Найбільш відомими серед них є LeetCode, HackerRank та Codeforces. Проте детальний аналіз їхньої архітектури та ліцензійної політики

вказує на неможливість їх ефективної та безкоштовної адаптації під потреби локального освітнього процесу в окремому навчальному закладі.

Платформа LeetCode є визнаним стандартом для підготовки інженерів до співбесід у провідні технологічні компанії. Вона володіє величезною базою завдань та підтримує десятки мов програмування. Проте LeetCode є комерційним хмарним сервісом із закритою архітектурою. Вона не надає інструментів для безкоштовного створення ізольованих навчальних кабінетів для викладачів навчального закладу, не дозволяє гнучко додавати власні специфічні задачі із внутрішніх методичних вказівок та повністю залежить від зовнішнього підключення до мережі Інтернет. Інтерфейс платформи є надлишковим для студентів початкових курсів, а система оцінювання орієнтована на глобальні рейтинги, а не на фіксацію конкретної академічної групи.

Платформа HackerRank позиціонується як інструмент для технічного рекрутингу та проведення змагань. Вона має функціонал для створення корпоративних тестів, проте використання цих можливостей є платним та фінансово недоступним для державних закладів освіти. Більш того, створення та валідація власних завдань на HackerRank вимагає написання складних конфігураційних скриптів та unit-тестів, що створює високий поріг входження для викладачів, які не мають часу на вивчення специфічного внутрішнього інструментарію платформи. Система також функціонує виключно у хмарі, що робить її вразливою до перебоїв із мережевим живленням або відсутністю доступу до глобальної мережі в навчальній аудиторії.

Платформа Codeforces орієнтована виключно на спортивне програмування та проведення масштабних олімпіад. Вона базується на жорсткому порівнянні стандартних потоків введення та виведення даних, де студент повинен самостійно реалізовувати парсинг складних рядків. Такий підхід є методологічно неправильним для навчання базовим алгоритмам, оскільки фокусує увагу студента на рутинних операціях введення-виведення, а не на побудові логіки самого алгоритму. Крім того, інфраструктура Codeforces є

повністю монолітною та не передбачає розгортання локальних копій системи всередині приватної мережі навчального закладу.

Для систематизації результатів аналізу та наочного відображення переваг розроблюваного програмного продукту було сформовано порівняльну таблицю наявних аналогів за ключовими критеріями, що мають вирішальне значення для впровадження системи в освітній простір навчального закладу, табл. 1.1.

Таблиця 1.1 — Порівняльний аналіз систем автоматичної перевірки коду

Критерій порівняння	**LeetCode**	**HackerRank**	**Codeforces**	**Розроблювана платформа**
Тип розгортання	Виключно хмарне	Виключно хмарне	Виключно хмарне	Локальне автономне
Вартість використання	Платна підписка	Платна для інституцій	Безкоштовна	Повністю безкоштовна
Контроль бази завдань	Закрита база платформи	Складна конфігурація	Складна система Polygon	Пряме керування через БД
Специфіка використання	Підготовка до інтерв'ю	Корпоративний рекрутинг	Спортивне програмування	Освітній процес навчального закладу
Залежність від Інтернету	Повна залежність	Повна залежність	Повна залежність	Абсолютно незалежна
Модифікація інтерфейсу	Неможлива	Неможлива	Неможлива	Повна свобода дій

Аналіз таблиці 1.1 доводить, що жодне з існуючих масштабних рішень не задовольняє специфічні вимоги навчального закладу. Головним недоліком комерційних систем є їхня монолітність та неможливість кастомізації. Розроблювана локальна платформа, яка запускається за допомогою простого командного файлу та функціонує на базі легко вагомих технологій, усуває всі зазначені обмеження. Вона гарантує абсолютну незалежність від зовнішніх

серверів, дозволяє викладачеві миттєво змінювати логіку перевірки, додавати необмежену кількість нових задач через прямі SQL-запити або інтерфейс і повністю адаптувати візуальну частину під фірмовий стиль закладу освіти.

1.3 Обґрунтування вибору технологічного стеку розробки

Реалізація локальної системи автоматичного тестування алгоритмічних задач вимагає використання надійних, швидких та простих у підтримці інструментів розробки програмного забезпечення. Зважаючи на архітектурні вимоги до системи, яка має функціонувати в межах локальної комп'ютерної мережі лабораторій, було обрано стек технологій, що поєднує мову програмування Python із мікрофреймворком Flask на серверній стороні, реляційну базу даних SQLite для збереження інформації та стандартні технології веб-інтерфейсу на стороні клієнта.

Вибір мови Python як основного інструменту реалізації серверної логіки обумовлений декількома факторами. По-перше, Python є офіційною мовою навчання на технічних спеціальностях навчального закладу, що спрощує подальший супровід та модернізацію проекту іншими студентами або викладачами. По-друге, мова має потужну вбудовану підтримку низькорівневої взаємодії з операційною системою через модуль стандартної бібліотеки, що є критично важливим для створення ізольованих підпроцесів виконання стороннього коду та перехоплення системних потоків даних.

Як базовий серверний каркас було обрано мікрофреймворк Flask.[1] На відміну від громіздких та монолітних рішень, таких як Django, фреймворк Flask надає виключно необхідний мінімальний інструментарій для маршрутизації HTTP-запитів та обробки сесій.[2] Це дозволяє уникнути надлишковості коду, забезпечує максимальну швидкість обробки мережевих запитів та дозволяє розробникові самостійно проектувати архітектуру застосунку. Flask є ідеальним рішенням для локальних автономних сервісів, оскільки він легко запускається на машинах із будь-якою конфігурацією та не потребує складного попереднього адміністрування веб-сервера.

Для збереження інформації про категорії, завдання та користувацькі спроби було обрано реляційну систему керування базами даних SQLite. Головна перевага SQLite полягає в її безсерверній архітектурі. База даних зберігається в одному бінарному файлі всередині директорії проекту, що повністю нівелює потребу у встановленні, налаштуванні та підтримці окремого великого сервера баз даних, такого як PostgreSQL або MySQL. Це забезпечує максимальну портативність системи: для перенесення всієї платформи на інший комп’ютер викладачеві достатньо скопіювати папку з проектом. При цьому SQLite повністю підтримує стандарт мови SQL, забезпечує цілісність даних через зовнішні ключі та демонструє високу швидкість роботи при одночасній роботі в межах однієї навчальної аудиторії.

Клієнтська частина веб-платформи реалізована без використання сторонніх JavaScript-фреймворків, таких як React, Vue або Angular, та базується виключно на стандартних технологіях конструювання інтерфейсів. Використання мови мови сценаріїв JavaScript у її базовому вигляді дозволяє уникнути стадії компіляції та збирання проекту, суттєво зменшує об’єм файлів, що передаються клієнту, та гарантує миттєве завантаження сторінок у браузері навіть на застарілих комп’ютерах навчальних лабораторій. Асинхронна взаємодія із сервером реалізована за допомогою сучасного вбудованого API, що дозволяє відправляти вихідний код на перевірку та отримувати результати тестування у форматі JSON без перезавантаження веб-сторінки, забезпечуючи високий рівень комфорту для користувача.

У першому розділі кваліфікаційної роботи було проведено детальний аналіз предметної області та процесів оцінювання програмного коду. Розглянуто архітектурні особливості наявних систем автоматизованої перевірки, виявлено їхні переваги та ключові недоліки, зокрема складність розгортання та залежність від сторонніх хмарних серверів. На основі проведеного порівняльного аналізу було обґрунтовано вибір технологічного стеку для розробки локальної вебплатформи, що забезпечує оптимальний баланс між швидкодією, безпекою та автономністю функціонування системи.

РОЗДІЛ 2 ПРОЄКТУВАННЯ ТА ПРОГРАМНА РЕАЛІЗАЦІЯ ПЛАТФОРМИ АВТОМАТИЗОВАНОГО ТЕСТУВАННЯ

2.1 Проєктування реляційної структури бази даних

Функціонування сучасних засобів автоматизації контролю знань безпосередньо залежить від ефективності та надійності архітектури збереження даних. На етапі проєктування платформи автоматизованого тестування алгоритмічних задач для навчального закладу виникла потреба у створенні такої моделі даних, яка б поєднувала низькі вимоги до системних ресурсів навчальних комп’ютерів із повною підтримкою принципів реляційної цілісності. Враховуючи необхідність забезпечення абсолютної автономності та простоти копіювання програмного комплексу між робочими станціями лабораторії, як базову систему керування базами даних було обрано SQLite.[2] Логічна структура сховища розроблялася з урахуванням нормалізації даних до третьої нормальної форми, що дозволило уникнути аномалій вставки, оновлення та видалення інформації, а також мінімізувати надлишковість при фіксації великої кількості студентських спроб.

Архітектура бази даних, яка фізично зберігається у файлі проекту під назвою tasks.db, базується на трьох основних взаємопов’язаних таблицях, кожна з яких виконує чітко визначену системну функцію. Графічне відображення структури розробленої бази даних, що ілюструє зв'язки між таблицями категорій, задач та рішень, наведено на рис. 2.1. Першою структурною одиницею є таблиця категорій, яка у системному коді має назву categories. Ця сутність призначена для тематичного розподілу навчального матеріалу і містить унікальний числовий ідентифікатор із властивістю автоматичного інкременту, текстове поле для збереження офіційної назви категорії, а також унікальне індексне поле slug для формування людино-зрозумілих ідентифікаторів у веб-середовищі. Наявність обмеження UNIQUE для поля slug гарантує відсутність дублювання логічних розділів та дозволяє серверній частині

застосунку коректно взаємодіяти з клієнтськими запитами при майбутньому масштабуванні системи.

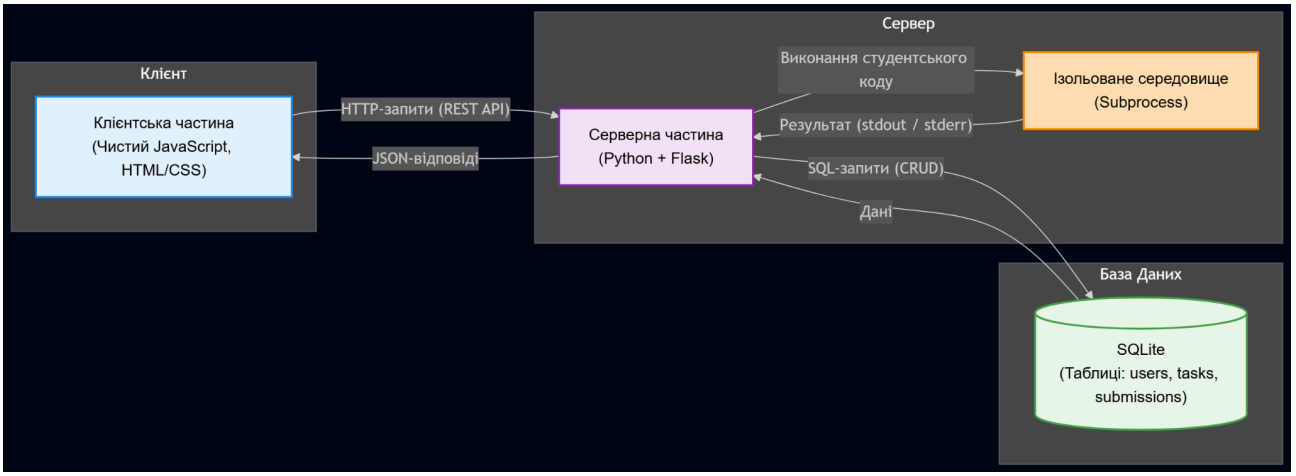

Рисунок 2.1 — ER-діаграма логічної структури бази даних

Центральне місце у структурі займає таблиця алгоритмічних задач, яка іменована як tasks. Ця сутність знаходиться у відношенні «багато до одного» із таблицею категорій, що реалізовано через явне визначення зовнішнього ключа category_id, який посилається на первинний ключ таблиці categories. Для забезпечення високої деталізації кожне завдання описується унікальним набором полів, серед яких заголовок задачі, розгорнутий текстовий опис умов виконання, обмеження максимального часу роботи підпроцесу у форматі дійсного числа, а також два критично важливі елементи для проведення автоматичної перевірки. Першим елементом є поле prelude, що зберігає вихідний код ініціалізації вхідних параметрів алгоритму, який невидимо для студента інтегрується в його програму. Другим елементом є поле expected_output, яке містить точний еталонний рядок виведення, з яким система порівнює фактичний результат роботи скрипту.

Останнім елементом архітектури бази даних є таблиця submissions, призначена для ведення детального аудиту та логування всіх дій користувачів. Вона пов’язана з таблицею задач відношенням зовнішнього ключа task_id, що дозволяє відстежувати історію розв’язання кожного конкретного завдання. Ця

таблиця адаптована під фіксацію динамічних метрик і містить поля для збереження повного тексту надісланого коду, текстового статусу перевірки, фактичного результату виведення програми із потоку корисних даних, точного часу виконання підпроцесу операційної системи з точністю до мілісекунд, а також мітки часу створення запису, яка генерується автоматично за допомогою системної функції CURRENT_TIMESTAMP операційної системи. Такий підхід забезпечує повне логування освітнього процесу, надаючи викладачеві можливість детального аналізу помилок.

2.2 Архітектура взаємодії та специфікація програмного інтерфейсу

Організація взаємодії між компонентами розробленого програмного комплексу базується на архітектурному шаблоні декомпозиції клієнт-серверної інфраструктури за допомогою розробки програмного інтерфейсу REST API.[5] Серверна частина, яка функціонує під керуванням мікрофреймворку Flask, виступає в ролі ізольованого обчислювального вузла та сховища інформації, тоді як клієнтська сторона відповідає за візуалізацію даних, взаємодію з користувачем та асинхронне відправлення вихідного тексту коду на перевірку. Обмін інформацією між браузером та сервером здійснюється виключно за допомогою протоколу HTTP, де як універсальний формат серіалізації структур даних використовуються текстові об’єкти JSON.[6] Такий підхід забезпечує мінімальний об’єм мережевого трафіку всередині локальної мережі комп’ютерного класу навчального закладу, гарантує миттєву обробку подій та дозволяє повністю відмовитися від рутинного перезавантаження веб-сторінок.

Першим базовим маршрутом серверної частини є обробник з адресою /api/categories, який підтримує виключно метод отримання даних GET та відповідає за формування повної структури навчального контенту, необхідної для побудови клієнтського інтерфейсу. При надходженні запиту до цього маршруту сервер ініціює відкриття транзакції до бази даних SQLite, зчитує всі записи з таблиці категорій, а потім для кожної знайденої категорії виконує вкладений SQL-запит для вибірки пов’язаних алгоритмічних задач із таблиці

tasks. Сервер виключає з вибірки чутливі поля, такі як очікуваний результат, і формує складний вкладений словник, де кожна категорія містить масив об’єктів задач з їхніми назвами, описами та лімітами часу. Після цього структура серіалізується за допомогою вбудованої функції jsonify і повертається клієнту у вигляді JSON-масиву, де кожен елемент верхнього рівня містить ідентифікатор, назву розділу, його унікальний текстовий індекс та внутрішній масив об’єктів завдань, що дозволяє фронтенду динамічно генерувати дерево навігації в бічному меню.

Другим інтеграційним маршрутом є обробник динамічних запитів деталей конкретного завдання, який зареєстрований за шаблоном /api/tasks/ за умови передачі цілого числового параметра ідентифікатора задачі та підтримує метод GET. Цей маршрут забезпечує ізольоване отримання атрибутів однієї конкретної алгоритмічної задачі, коли студент переходить до її розв’язання у веб-інтерфейсі. Серверний код приймає змінну ідентифікатора, виконує безпечний параметризований запит до таблиці tasks, витягує опис та технічні обмеження, а у разі відсутності запису в базі даних повертає стандартизовану JSON-відповідь із кодом помилки та HTTP-статусом 404. При знаходженні інформація трансформується у JSON-об’єкт, який містить поля назви, розгорнутої умови та ліміту часу, що дозволяє клієнтському сценарію миттєво оновити вміст робочої області та підготувати текстовий редактор до введення коду.

Третім, найбільш критичним та архітектурно складним компонентом інтерфейсу є маршрут надсилання рішень на перевірку, який функціонує за адресою /api/submit та приймає виключно POST-запити. Клієнтська сторона передає на цей маршрут JSON-тіло, яке містить два обов’язкові параметри, якими є числовий ідентифікатор задачі task_id та повний вихідний текст написаної студентом програми. Серверний обробник виконує валідацію вхідних структур, зчитує з бази даних SQLite параметри контексту ініціалізації масивів та очікуваного виведення для вказаної задачі, після чого передає дані внутрішньому модулю ізольованого тестування. Результатом роботи цього

роуту є повернення клієнту деталізованого JSON-об’єкта із фінальним вердиктом, фактичним текстовим рядком виведення програми, часом роботи підпроцесу в секундах, а також вмістом потоку помилок stderr у разі виникнення винятків інтерпретатора, що забезпечує студента вичерпною інформацією для налагодження алгоритму.

2.3 Програмна логіка безпечного виконання та верифікації коду

Найбільш відповідальним та технічно складним компонентом серверної частини розробленої вебплатформи є модуль автоматизованого тестування, який безпосередньо відповідає за верифікацію надісланого користувачами програмного коду. Основна програмна логіка модуля автоматизованого тестування, що відповідає за верифікацію користувацького коду, безпечний запуск підпроцесів та обробку винятків, реалізована у головному файлі серверного застосунку, повний лістинг якого наведено у Додатку А. Реалізація цього модуля вимагала розв’язання сукупності взаємопов’язаних інженерних завдань, серед яких на першому місці стоїть забезпечення абсолютної безпеки операційної системи сервера, точність вимірювання динамічних метрик роботи скриптів, а також коректна обробка будь-яких помилок інтерпретатора. Логіка роботи цього компонента повністю інтегрована в головний файл серверного застосунку app.py і запускається під час обробки POST-запитів на аналітичний маршрут надсилання рішень.

Процес перевірки починається з етапу динамічного конструювання повного тексту виконуваного скрипту, що дозволяє реалізувати методологію прихованого тестування. Після отримання вихідного тексту коду студента з HTTP-запиту, серверний сценарій виконує параметризований запит до бази даних SQLite для отримання технічних параметрів поточної задачі. Зчитане з таблиці tasks поле прелюдії prelude, яке містить ініціалізацію конкретних тестових структур даних, таких как несортований масив numbers_to_sort для алгоритму бульбашкового сортування або початкове число number для рекурсивних обчислень, конкатенується у єдиний монолітний рядок із кодом

користувача. Метод текстового склеювання дозволяє повністю звільнити студента від рутинного написання коду введення-виведення інформації, фокусуючи його увагу виключно на логіці самого алгоритму, тоді як сервер отримує готовий до виконання скрипт.

Для безпосереднього запуску згенерованого коду було категорично відкинуто використання небезпечних вбудованих функцій динамічного проектування, таких як exec або eval, оскільки вони виконують сторонній код у контексті поточного процесу самого вебсервера, що створює критичні вразливості та загрозу падіння всієї платформи. Замість цього в системі реалізовано механізм міжпроцесової взаємодії за допомогою модуля стандартної бібліотеки Python під назвою subprocess. Перевірка коду здійснюється шляхом ініціалізації ізольованого операційного підпроцесу за допомогою виклику функції run, яка приймає масив параметрів, що містить шлях до інтерпретатора Python, прапор виконання командного рядка та сам згенерований текст програми. Для забезпечення повного контролю над процесом функція конфігурується з обов’язковим використанням параметрів перехоплення стандартного потоку виведення stdout та потоку системних помилок stderr, а також переводиться в режим суворого текстового декодування, що усуває потребу в ручній обробці байтових масивів.

Перед безпосереднім викликом підпроцесу сервер фіксує початкову мітку часу за допомогою високоточного системного таймера time.time. Після завершення роботи програми фіксується кінцева мітка, а різниця між ними визначає чистий час виконання алгоритму з точністю до мілісекунд. Для запобігання критичного перевантаження обчислювальних модулів сервера через надсилання студентами нескінченних циклів або заблокованих алгоритмів, виклик ізольованого підпроцесу обертається в блок обробки винятків з жорстким обмеженням часу тайм-ауту timeout, значення якого динамічно передається з конфігурації конкретної задачі.

У разі перевищення встановленого ліміту операційна система примусово генерує системне виключення TimeoutExpired. Серверний обробник миттєво

перехоплює цей стан, примусово знищує завислий підпроцес, фіксує в базі даних спроб унікальний статус timeout та повертає клієнту відповідний вердикт, сигналізуючи про перевищення часового обмеження. Якщо програма завершується в межах встановленого ліміту, сервер аналізує код повернення підпроцесу returncode. Якщо код повернення не дорівнює нулю, це свідчить про виникнення критичної помилки під час виконання алгоритму, такої як ділення на нуль або вихід за межі масиву. В такому випадку сервер фіксує статус error і транслює вміст перехопленого потоку stderr назад користувачеві для полегшення налагодження.

При нульовому коді повернення сервер зчитує дані з потоку stdout, виконує очищення рядка від службових символів переносу за допомогою методу strip і порівнює результат з еталонним значенням expected_output з бази даних. При повному посимвольному збігу рядків спробі присвоюється фінальний статус виконання success, що відповідає індустріальному вердикту Accepted. Якщо логіка студента повернула невірний результат, система маркує запис статусом wrong, що відповідає вердикту Wrong Answer. Всі отримані метрики, включаючи фактичний текст виведення та точний час роботи процесу, записуються в таблицю submissions за допомогою параметризованого SQL-запиту, після чого серіалізуються в JSON-об’єкт і надсилаються на фронтенд для візуалізації результату.

2.4 Розробка клієнтського інтерфейсу платформи та асинхронна взаємодія

Візуальна частина та логіка поведінки користувача на стороні клієнта розроблені на базі сучасних стандартів конструювання веб-інтерфейсів без залучення важковагових сторонніх фреймворків. Для забезпечення зручної взаємодії користувача з платформою було розроблено мінімалістичний вебінтерфейс. Основна сторінка містить бічну панель навігації зі списком категорій та завдань, а також робочу область із вбудованим редактором коду. Загальний вигляд головного вікна розробленого інтерфейсу користувача

представлено на рис. 2.4. Клієнтський сценарій, реалізований у файлі script.js, повністю керує життєвим циклом сторінки, здійснює динамічний рендеринг елементів інтерфейсу на основі отриманих від сервера структур даних та забезпечує асинхронний обмін інформацією за допомогою вбудованого інтерфейсу Fetch API.[4] Логіка побудови фронтенду орієнтована на концепцію єдиного вікна, де перехід між алгоритмічними задачами, написання програмного коду та отримання результатів тестування відбуваються динамічно, що виключає необхідність рутинного перезавантаження сторінок та мінімізує затримки сприйняття інтерфейсу.

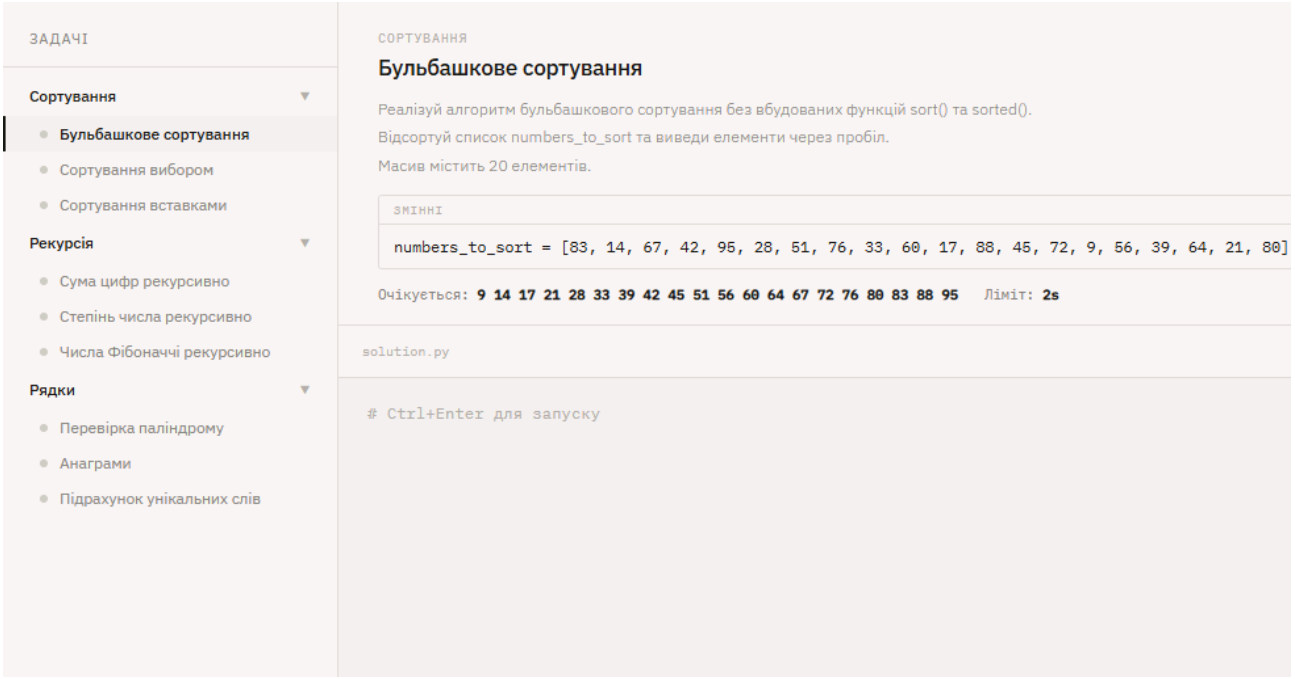

Рисунок 2.4 — Головне вікно інтерфейсу користувача локальної вебплатформи

Головним механізмом ініціалізації клієнтської частини є асинхронна функція load, яка автоматично викликається при завантаженні веб-документа. Ця функція виконує GET-запит до серверного маршруту /api/categories, отримує повний структурований масив доступних категорій та вкладених завдань, після чого зберігає його в оперативній пам'яті браузера у вигляді глобального масиву об'єктів. Для покращення користувацького досвіду ідентифікатор першої

знайденої категорії автоматично додається до глобальної структури збереження стану openCats, реалізованої на базі колекції Set, що дозволяє автоматично тримати перший тематичний розділ відкритим при первинному відвідуванні платформи студентом. Після парсингу JSON-відповіді керування передається функції циклічного рендерингу бічного меню.

Функція побудови навігаційного дерева сайту динамічно очищує вміст контейнера бічної панелі та ітеративно створює блоки елементів документа для кожної категорії. Завдяки використанню об'єкта Set система миттєво визначає поточний стан розгортання конкретного розділу та додає відповідні стильові класи для візуалізації стрілок індикаторів. При натисканні на заголовок категорії спрацьовує анонімний обробник подій, який інвертує стан наявності ідентифікатора в колекції openCats та рекурсивно викликає перемальовування меню, забезпечуючи плавний інтерфейс користувача. Внутрішній цикл функції обходить усі задачі поточної категорії, перевіряє наявність збереженого статусу виконання у глобальному словнику оцінок та додає кольоровий маркер (зелений для збігу, червоний для помилок чи невірного результату, жовтий для таймауту), що дозволяє студенту наочно бачити прогрес виконання поточних лабораторних робіт.

Найбільш унікальним та технічно важливим елементом клієнтського сценарію є модуль обробки подій введення тексту в область редактора коду, який імітує поведінку професійного середовища розробки. Традиційні текстові поля браузера при натисканні клавіші Tab переміщують фокус на наступний елемент веб-сторінки, що робить неможливим написання програмного коду на мові Python, де відступи є синтаксичною основою структури блоків. Для вирішення цієї проблеми у функції renderTask реалізовано жорсткий перехоплювач подій keydown для елемента textarea.[10] При фіксації натискання клавіші табуляції сценарій примусово скасовує стандартну поведінку браузера за допомогою викликуpreventDefault. Після цього код програмно обчислює поточні координати курсора користувача, розщеплює текстовий рядок у точці фокусу, вставляє два символи пробілу, які є стандартом оформлення коду

Python, та повертає покажчик курсора в коректну позицію, забезпечуючи повну зручність написання алгоритмів безпосередньо у вікні браузера.

Кінцева фаза взаємодії реалізована в асинхронній функції submitCode, яка активується при натисканні кнопки тестування. Перед відправкою даних інтерфейс блокує елементи керування та додає стильовий індикатор завантаження, захищаючи систему від повторних некоректних запитів під час виконання коду на сервері. Функція формує POST-запит, передає JSON-об'єкт із текстом програми та ідентифікатором задачі, а після отримання відповіді динамічно розбирає вердикт сервера. Залежно від статусу, текстовий блок результатів забарвлюється у відповідний колір, а за допомогою конструювання шаблонних рядків на сторінку виводяться блоки порівняння очікуваного та фактичного результатів, вміст системного потоку помилок stderr та точний час виконання підпроцесу, забезпечуючи миттєвий та інформативний зворотний зв'язок.

Другий розділ роботи присвячено етапам проєктування та безпосередньої програмної реалізації платформи автоматизованого тестування. У процесі виконання було розроблено реляційну структуру локальної бази даних SQLite та спроєктовано логіку ізольованого виконання користувацького коду за допомогою підпроцесів із жорстким обмеженням часу (timeout). Також було створено клієнтський веб-інтерфейс застосунку з інтегрованим редактором коду, що підтримує коректну обробку табуляції, та налаштовано асинхронну взаємодію із сервером за допомогою технології Fetch API.

РОЗДІЛ 3 ТЕСТУВАННЯ ПРОГРАМНОГО ПРОДУКТУ

3.1 Наповнення платформи алгоритмічними задачами та структура тестових даних

Важливим етапом впровадження розробленої вебплатформи в освітній процес навчального закладу є її наповнення дидактичними матеріалами та формування бази еталонних тестів. На відміну от комерційних закритих систем, де додавання нових завдань вимагає вивчення складних внутрішніх специфікацій, у створеному програмному комплексі реалізовано гнучкий підхід до керування контентом. Наповнення платформи здійснюється безпосередньо через реляційну базу даних за допомогою структурування завдань за трьома фундаментальними технічними категоріями, якими є сортування, рекурсія та обробка рядків. Загальна кількість інтегрованих на етапі розгортання базових задач складає дев'ять одиниць, що повністю покриває первинні потреби дисциплін, пов’язаних із вивченням мови програмування Python та дослідженням базових структур даних.

Організація тестових даних у таблиці tasks базується на концепції жорсткого розділення логіки користувача та контексту ініціалізації вхідних параметрів. Перший тематичний розділ сортування містить комплекс задач, спрямованих на ручну реалізацію класичних інженерних алгоритмів без використання вбудованих функцій мови. Наприклад, для задачі бульбашкового сортування у полі прелюдії prelude жорстко зафіксовано одновимірний масив numbers_to_sort, який складається з двадцяти п'яти випадкових цілих чисел. Очікуваним результатом expected_output є рядок, де ці ж числа розташовані у строгому зростаючому порядку та розділені символом пробілу. Студент повинен написати виключно логіку обходу та перестановки елементів, а система автоматично склеїть цей код із масивом, виконає його та звірить фінальний текстовий потік виведення з еталоном.

Другий розділ рекурсивних алгоритмів орієнтований на перевірку навичок проєктування функцій, які викликають самі себе, та аналізу глибини

рекурсії. Сюди включені завдання на обчислення суми цифр великого цілого числа number, знаходження факторіалу та піднесення числа до степеня. У полі прелюдії для цих задач фіксуються вихідні змінні, а в полі очікуваного результату зберігається єдине підсумкове число. Третій розділ обробки рядків містить завдання на валідацію текстових структур, пошук підрядків, інвертування послідовностей символів та підрахунок входжень елементів. Для кожного завдання в базі даних чітко визначено індивідуальний ліміт часу execution_time, що дозволяє викладачеві самостійно регулювати вимоги до ефективності написаних студентами алгоритмів, забезпечуючи гнучку адаптацію платформи під будь-які навчальні плани навчального закладу.

3.2 Методологія та результати функціонального тестування програми

Забезпечення високої якості та стабільності функціонування розробленої вебплатформи автоматичного оцінювання вимагало проведення комплексного функціонального тестування. Головною метою цього етапу стало експериментальне підтвердження коректності роботи модуля верифікації коду при обробці різних типів користувацьких сценаріїв. Оскільки платформа призначена для автономного використання в освітньому процесі навчального закладу, тестування проводилося за методологією «чорної скриньки» з формуванням декількох еквівалентних класів входних даних .[9] Кожен клас відповідав конкретному стану виконання програми студента та перевіряв здатність серверної частини застосунку коректно ідентифікувати результати, ізолювати помилки інтерпретатора та своєчасно реагувати на критичні ситуації .

Перший етап тестування був спрямований на валідацію еталонних рішень, які повністю відповідають технічному завданню алгоритмічних задач. Для цього в область текстового редактора клієнтської частини по черзі вводився заздалегідь написаний правильний код для кожного з дев'яти інтегрованих завдань. Серверний модуль здійснював конкатенацію тексту програми з контекстом ініціалізації вхідних даних (prelude), запускав операційний підпроцес і здійснював порівняння отриманого результату з еталонним рядком.

В усіх дев'яти випадках система стабільно зафіксувала код повернення підпроцесу рівний нулю, підтвердила повний посимвольний збіг очищених рядків виведення з полем expected_output бази даних і присвоїла спробам підсумковий статус success, що відобразилося в інтерфейсі користувача у вигляді індустріального вердикту Accepted .

Другий етап функціональної перевірки передбачав моделювання ситуацій, коли студент надсилає працездатний програмний код, який не містить синтаксичних дефектів, але реалізує помилкову логіку обчислень. Для тестування цього класу в систему надсилався код алгоритму сортування, в якому було свідомо змінено знак порівняння елементів масиву, що призводило до сортування у зворотному порядку. Під час виконання інтерпретатор завершував роботу підпроцесу з нульовим кодом, проте фактичний текстовий потік даних stdout не збігався з еталонним значенням сховища SQLite. Серверна частина платформи безпомилково ідентифікувала цей стан, згенерувала внутрішній вердикт wrong (Wrong Answer) та передала клієнту об'єкт результатів, що дозволило фронтенду підсвітити блоки порівняння очікуваного та вашого виведення червоним кольором.

Третій клас тестування був орієнтований на перевірку стійкості системи до обробки синтаксичних помилок та критичних винятків часу виконання (Runtime Error). В область введення підставлявся код, що містив помилки звернення до неіснуючих змінних, ділення на нуль або виходу за межі індексів масиву. При запуску такого скрипту інтерпретатор Python аварійно завершував підпроцес, повертаючи ненульовий код помилки операційної системи. Блок аналізу в app.py зафіксував цей стан, автоматично присвоїв спробі статус error та повністю перехопив вміст системного потоку помилок stderr. Цей потік був переданий на фронтенд і виведений у спеціальному діалоговому вікні редактора, що підтвердило здатність платформи надавати студенту повну інформацію для швидкого налагодження алгоритму.

Кінцевий етап функціонального тестування мав вирішальне значення для безпеки і полягав у моделюванні критичних навантажень через надсилання

нескінченних циклів. Студентський код містив конструкцію while True без операторів примусового виходу, що зазвичай призводить до стовідсоткового зависання обчислювального потоку сервера. Завдяки реалізації жорсткого контролю ліміту часу у функції subprocess.run, операційна система примусово перервала виконання стороннього підпроцесу рівно через дві секунди, що визначено налаштуваннями задачі. Система перехопила виключення TimeoutExpired, безпечно очистила пам'ять, зробила відповідний запис у таблицю submissions зі статусом timeout та повернула керування інтерфейсу, повністю підтвердивши свою надійність і стійкість до деструктивного коду.

3.3 Інструкція з розгортання та супроводу програмного продукту

Заключним етапом інженерної розробки локальної вебплатформи автоматизованого оцінювання є проектування надійного механізму її розгортання та подальшого технічного супроводу в інфраструктурі комп’ютерних класів навчального закладу. Оскільки кінцевими користувачами системи на стороні сервера можуть виступати викладачі або лаборанти, які не мають глибоких навичок адміністрування баз даних та ручного налаштування вебсерверів, виникла гостра технічна необхідність максимізувати автоматизацію всіх стартових процесів. Для вирішення цього завдання було розроблено спеціальний командний файл інсталяції та запуску під назвою start.bat, який повністю бере на себе рутинні операції перевірки цілісності сховища та підняття локальних мережевих служб.

Логіка роботи стартового командного скрипту базується на послідовному виконанні системних директив операційної системи. При запуску файл активує режим приховування виведення службових команд за допомогою інструкції echo off, після чого виконує умовну перевірку фізичної наявності файлу реляційної бази даних tasks.db у поточному робочому каталозі застосунку. Якщо система розгортається на новій робочій станції лабораторії навчального закладу вперше і файл бази даних відсутній, командний сценарій автоматично ініціює виклик інтерпретатора для запуску допоміжного скрипту інціалізації init_db.py.

Цей скрипт у безпечному режимі зчитує інструкції створення таблиць із файлу schema.sql, відкриває транзакцію, послідовно конструює структуру категорій, задач та спроб, здійснює первинне наповнення бази дев'ятьма алгоритмічними завданнями та закриває з’єднання, повністю виключаючи потребу в ручному втручанні користувача.

Після проходження етапу валідації або створення сховища даних, командний файл start.bat переходить до фази безпосередньої організації взаємодії з користувачем. За допомогою системної інструкції start скрипт надсилає операційній системі команду на відкриття стандартного веббраузера за замовчуванням, куди автоматично передається локальна мережева адреса і порт майбутнього підняття сервера. Важливою інженерною особливістю є те, що ця команда виконується в асинхронному фоновому режимі без блокування консолі. Відразу після цього командний файл виконує фінальну директиву запуску головного серверного файлу app.py, яка піднімає локальний мікросервер Flask. Браузер, який відкрився на попередньому кроці, миттєво підхоплює активний мережевий порт сервера, надаючи викладачеві або студенту готовий до роботи інтерфейс, що робить процес запуску платформи максимально простим та швидким.

Подальший технічний супровід та масштабування розробленого програмного комплексу також мають низький поріг входження завдяки архітектурним особливостям обраного технологічного стеку. Для перенесення всієї платформи разом із накопиченою історією студентських спроб та базою завдань на інший комп’ютер навчальної аудиторіїлаборанту достатньо скопіювати єдину кореневу папку проєкту на зовнішній носій. Додавання нових завдань або редагування чинних умов може здійснюватися викладачем двома гнучкими шляхами: або через пряме написання SQL-інструкцій INSERT у файл ініціалізації schema.sql для глобального оновлення системи, або за допомогою будь-якого візуального графічного менеджера баз даних для SQLite, що забезпечує високу життєздатність та адаптивність платформи в довгостроковій освітній перспективі.

У третьому розділі кваліфікаційної роботи було розроблено методологію та успішно проведено функціональне тестування програмного продукту. Сформовано тестові сценарії для перевірки обробки системою різних типів вердиктів (успішна компіляція, помилка виконання, перевищення ліміту часу та неправильна відповідь). Окрім цього, було укладено детальну інструкцію з розгортання та супроводу платформи, а також створено командний скрипт автоматизації запуску, що підтверджує готовність локального застосунку до практичної експлуатації.

ВИСНОВКИ

У кваліфікаційній роботі розв’язано актуальне інженерно-прикладне завдання щодо проєктування, реалізації та випробування автономної вебплатформи для автоматизованої перевірки правильності програмного коду на мові Python. На основі проведеного аналізу предметної області та дослідження існуючих систем автоматичного оцінювання було виявлено, що глобальні комерційні хмарні платформи мають закриту інфраструктуру, потребують постійного підключення до мережі Інтернет та не надають безкоштовних інструментів для кастомізації під робочі програми навчального закладу. Створений програмний комплекс повністю усуває ці обмеження, забезпечуючи абсолютну незалежність від зовнішніх серверів, портативність та гнучкість конфігурації.

У процесі виконання роботи було детально формалізовано функціональні та нефункціональні вимоги до системи, що дозволило обґрунтувати вибір технологічного стеку розробки. Використання мови програмування Python та мікрофреймворку Flask на серверній стороні забезпечило високу швидкість обробки HTTP-запитів, а відмова від складних клієнтських JavaScript-фреймворків на користь стандартних вебтехнологій мінімізувала об’єм мережевого трафіку та вимоги до апаратного забезпечення комп’ютерних лабораторій. Спроєктована реляційна структура бази даних на базі безсерверної СУБД SQLite, яка складається з трьох нормалізованих таблиць, гарантує цілісність інформації, безпечне збереження навчального контенту та повний аудит усіх студентських спроб надсилання рішений.

Найбільш вагомим практичним результатом роботи є реалізація надійної програмної логіки безпечного виконання стороннього коду за допомогою створення ізольованих операційних підпроцесів модуля subprocess з перехопленням потоків даних stdout та stderr. Впровадження механізму динамічного конструювання контексту виконання дозволило відокремити логіку алгоритму від рутинних операцій введення-виведення, а інтеграція жорсткого

контролю тайм-аутів забезпечила стійкість сервера до нескінченних циклів. Розроблений клієнтський інтерфейс із вбудованим перехоплювачем клавіші табуляції створює для студентів комфортне середовище розробки, а модуль асинхронної взаємодії Fetch API миттєво повертає деталізовані вердикти перевірки.

Комплексне функціональне тестування підтвердило працездатність системи при обробці чотирьох базових статусів відповідей, якими є виконання, логічна помилка, виняток часу виконання та перевищення ліміту часу. Створений командний файл автоматизації розгортання start.bat мінімізував поріг входження для обслуговуючого персоналу навчального закладу, забезпечуючи перевірку та автоматичне створення бази даних з файлу конфігурації. Це дозволяє рекомендувати її для впровадження у навчальний процес при проведенні лабораторних робіт та зрізів знань з алгоритмічних дисциплін.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Flask Documentation. *Pallets Projects*. 2024. URL: https://flask.palletsprojects.com/en/stable/ (дата звернення: 01.06.2026).

2. SQLite Documentation. *SQLite Consortium*. 2025. URL: https://www.sqlite.org/docs.html (дата звернення: 02.06.2026).

3. Python Software Foundation. Subprocess management module. *Python 3.12 Documentation*. 2024. URL: https://docs.python.org/3/library/subprocess.html (дата звернення: 02.06.2026).

4. MDN Web Docs. Using the Fetch API. *Mozilla Developer Network*. 2025. URL: https://developer.mozilla.org/en-US/docs/Web/API/Fetch_API/Using_Fetch (дата звернення: 02.06.2026).

5. Fielding R., Taylor R. Architectural Styles and the Design of Network-based Software Architectures. *UCI Irvine*. 2000. URL: https://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm (дата звернення: 02.06.2026).

6. Google Chrome Developers. Asynchronous JavaScript and XML (AJAX) Overview. *Web.dev*. 2024. URL: https://web.dev/articles/ajax-overview (дата звернення:02.06.2026).

7. Покутная А. В. Методи та системи автоматизованої перевірки знань з програмування у вищих навчальних закладах. *Комп'ютерні системи та мережі*. 2021. URL: http://ea.nure.ua/handle/123456789/16543 (дата звернення: 03.06.2026).

8. Глибовець М. М. Архітектура та принципи побудови систем автоматичного тестування програмного коду (Online Judges). *Наукові записки НаУКМА. Комп'ютерні науки*. 2019. URL: http://ekmair.ukma.edu.ua/handle/123456789/17021 (дата звернення: 04.06.2026).

9. ISO/IEC/IEEE 29119-1:2022. Software and systems engineering — Software testing — Part 1: General concepts. *International Organization for Standardization*. 2022. URL: https://www.iso.org/standard/81291.html (дата звернення: 06.06.2026).

10. WHATWG. HTML Living Standard: The textarea element and event handling. *Web Hypertext Application Technology Working Group*. 2026. URL: https://html.spec.whatwg.org/multipage/form-elements.html#the-textarea-element (дата звернення: 06.06.2026).

ДОДАТКИ

Додаток А - Вихідний код серверної частини застосунку.

```
import sqlite3
import subprocess
import time
from flask import Flask, request, jsonify, render_template, g

app = Flask(__name__)
DATABASE = "tasks.db"


def get_db():
    db = getattr(g, "_database", None)
    if db is None:
        db = g._database = sqlite3.connect(DATABASE)
        db.row_factory = sqlite3.Row
    return db


@app.teardown_appcontext
def close_connection(exception):
    db = getattr(g, "_database", None)
    if db is not None:
        db.close()


@app.route("/")
def index():
    return render_template("index.html")


@app.route("/api/categories", methods=["GET"])
def get_categories():
    db = get_db()
    cats = db.execute("SELECT * FROM categories").fetchall()
    result = []
    for cat in cats:
        tasks = db.execute(
            "SELECT id, title, description, prelude, expected_output, time_limit FROM
tasks WHERE category_id = ?",
            (cat["id"],)
        ).fetchall()
        result.append({**dict(cat), "tasks": [dict(t) for t in tasks]})
    return jsonify(result)


@app.route("/api/tasks/<int:task_id>", methods=["GET"])
def get_task(task_id):
    db = get_db()
```

```
    task = db.execute("SELECT * FROM tasks WHERE id = ?", (task_id,)).fetchone()
    if task is None:
        return jsonify({"error": "Not found"}), 404
    return jsonify(dict(task))


@app.route("/api/submit", methods=["POST"])
def submit_code():
    data = request.get_json()
    if not data or "task_id" not in data or "code" not in data:
        return jsonify({"error": "Missing fields"}), 400

    task_id = data["task_id"]
    user_code = data["code"]

    db = get_db()
    task = db.execute("SELECT * FROM tasks WHERE id = ?", (task_id,)).fetchone()
    if task is None:
        return jsonify({"error": "Task not found"}), 404

    time_limit = task["time_limit"]
    expected = task["expected_output"].strip()
    prelude = task["prelude"] or ""

    full_code = prelude + "\n" + user_code if prelude else user_code

    try:
        start = time.time()
        result = subprocess.run(
            ["python", "-c", full_code],
            capture_output=True,
            text=True,
            timeout=time_limit,
        )
        elapsed = time.time() - start

        actual_output = result.stdout.strip()
        stderr_output = result.stderr.strip()

        if result.returncode != 0:
            status = "error"
        elif actual_output == expected:
            status = "success"
        else:
            status = "wrong"

        db.execute(
            "INSERT INTO submissions (task_id, code, status, actual_output,
execution_time) VALUES (?, ?, ?, ?, ?)",
            (task_id, user_code, status, actual_output, elapsed),
        )
        db.commit()
```

```
        return jsonify({
            "status": status,
            "actual_output": actual_output,
            "expected_output": expected,
            "execution_time": round(elapsed, 4),
            "stderr": stderr_output if status == "error" else None,
        })

    except subprocess.TimeoutExpired:
        db.execute(
            "INSERT INTO submissions (task_id, code, status, actual_output,
execution_time) VALUES (?, ?, ?, ?, ?)",
            (task_id, user_code, "timeout", None, time_limit),
        )
        db.commit()
        return jsonify({
            "status": "timeout",
            "execution_time": time_limit,
        })

    except Exception as e:
        return jsonify({"error": str(e)}), 500


if __name__ == "__main__":
    app.run(debug=True)
```