КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

ІМЕНІ ТАРАСА ШЕВЧЕНКА

ФАКУЛЬТЕТ РАДІОФІЗИКИ, ЕЛЕКТРОНІКИ ТА КОМП'ЮТЕРНИХ СИСТЕМ

Кафедра комп'ютерної інженерії

До захисту допущено: «На правах рукопису»
Завідувач кафедри _____ Юрій Бойко
« _ » _____ 2023 р.

КВАЛІФІКАЦІЙНА РОБОТА БАКАЛАВРА

на тему:

«РОЗРОБКА ПОРТАЛУ ДЛЯ АВТОМАТИЧНОЇ ПЕРЕВІРКИ ЛАБОРАТОРНИХ РОБІТ»

Виконав:
студент 4-го курсу бакалаврату
денної форми навчання
спеціальності 123 Комп'ютерна інженерія
ОНП «_____»
Реплянчук Вадим Олегович _____

Науковий керівник:
кандидат фізико-математичних наук, доцент
Іваненко Дмитро Олександрович _____

Рецензент:
_____

Засвідчую, що у цій бакалаврській роботі
немає запозичень з праць інших авторів без
відповідних посилань
Студент _____

Робота допущена до захисту в ЕК рішенням кафедри _____
від «__»_____ 2023 р., протокол № __.

Завідувач кафедри _____,
кандидат фізико-математичних наук, доцент
Бойко Юрій Володимирович

(підпис)

Київ – 2023

РЕФЕРАТ

Обсяг роботи 59 сторінок, 35 ілюстрацій, 7 джерел посилань.

Метою дипломної роботи було розроблення порталу для автоматичної перевірки лабораторних робіт з використанням фреймворку Django. У ході роботи було проведено аналіз існуючих порталів для автоматичної перевірки, розглянуто їхні можливості та обмеження. Було розроблено власний портал, який надає можливість авторизуватися студентам та викладачам, створювати лабораторні роботи та завантажувати відповіді студентів. Основний функціонал порталу включає автоматичну перевірку коду, формування звітів та зберігання результатів.

Під час розробки порталу було використано фреймворк Django для швидкої та зручної розробки веб-додатків. Була створена база даних для зберігання інформації про студентів, викладачів, лабораторні роботи та їх результати.

Ключові слова: автоматизація, фреймворк, портал, лабораторна робота, база даних.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ СКОРОЧЕНЬ І ПОЗНАЧЕНЬ

API (Application Programming Interface) – інтерфейс програмування додатків

СУБД - Система Управління Базами Даних

LMS (Learning Management System) - система управління навчанням

FRONTEND - частина розробки, яка займається побудовою користувацького інтерфейсу

BACKEND - програмно-апаратна частина розробки

ВСТУП

Автоматизований процес володіє більш стабільними характеристиками, ніж процес, що виконується в ручному режимі. У багатьох випадках автоматизація процесів дозволяє підвищити продуктивність, скоротити час виконання процесу, знизити вартість, підвищити точність і стабільність виконуваних операцій. За допомогою спеціального рішення можливо автоматизувати завдання, що вимагають аналізу великих обсягів необроблених даних та документів.

Основні переваги автоматизації:

- **Підвищення продуктивності та ефективності роботи.** Система автоматизації зменшує виникнення можливих помилок. Менше помилок — краща результативність.
- **Скорочення часу виконання та витрат.** Завдяки цьому внутрішні процеси виконуються швидше.
- **Легке управління даними та документами.** Передача інформації або розрахунків між різними джерелами даних підвищує ризик помилок, спричинених людським фактором, і потребує багато часу. Автоматизація процесів дозволяє централізувати роботу з усіма документами та даними. Це спрощує впорядкування та пошук потрібної інформації, а також керування нею.
- **Стандартизація процесів та забезпечення відповідності вимогам.** Завдяки цьому можна відстежувати використання даних, легко реєструючи імена, дати та подробиці призначення.

Задля покращення та додавання всіх цих переваг до системи перевірки лабораторних робіт постала необхідність у розробці порталу з можливістю автоматизованої перевірки лабораторних робіт в мережі Університету, чому і буде присвячена дана робота.

АКТУАЛЬНІСТЬ ТЕМИ ДИПЛОМНОЇ РОБОТИ

Автоматична перевірка лабораторних робіт є важливою та актуальною для студентів та викладачів з кількох причин:

- Ефективність процесу оцінювання: ручна перевірка лабораторних робіт займає значну кількість часу та зусиль викладачів, що може призвести до затримки у видачі результатів та оцінок. Автоматична перевірка дозволяє значно зменшити час, необхідний для оцінювання, та збільшити точність результатів.
- Покращення якості навчання: автоматична перевірка дозволяє студентам отримувати зворотній зв'язок щодо їх робіт швидше та більш часто, що може допомогти їм вдосконалювати свої навички та підвищувати якість своєї роботи.
- Збереження даних: автоматична перевірка забезпечує збереження даних про результати перевірки, що може бути корисним для моніторингу навчального процесу та для аналізу ефективності навчання.

Отже, автоматична перевірка лабораторних робіт може значно спростити та покращити процес оцінювання, забезпечити більш об'єктивну оцінку та збереження даних про результати перевірки, що в свою чергу може позитивно вплинути на якість навчання студентів та ефективність роботи викладачів

МЕТА ТА ЗАВДАННЯ ДОСЛІДЖЕННЯ

Мета дослідження:

Головною метою дослідження є розробка порталу для автоматичної перевірки лабораторних робіт. Цей портал спрямований на полегшення процесу перевірки лабораторних робіт.

Завдання дослідження:

Аналіз існуючих підходів та систем для автоматичної перевірки лабораторних робіт. Дослідження різних методів, алгоритмів та інструментів, які використовуються для автоматичної перевірки програмного коду, оцінки правильності результатів та надання зворотного зв'язку студентам.

Визначення вимог до функціональності та інтерфейсу порталу. Розробка чіткого переліку вимог, які повинен задовольняти портал, з урахуванням потреб студентів, викладачів та адміністраторів.

Розробка архітектури та проектування порталу. Визначення структури системи, компонентів та їх взаємодії, що забезпечують автоматичну перевірку лабораторних робіт.

Реалізація та тестування порталу. Розробка програмного забезпечення, яке відповідає визначеним вимогам та архітектурі. Проведення тестування для перевірки функціональності, надійності та продуктивності порталу.

Об’єкт та предмет дослідження

Об'єктом дослідження є процес перевірки лабораторних робіт у вищих навчальних закладах. Це включає в себе всі етапи, методи та інструменти, які використовуються для оцінки правильності та якості виконання лабораторних завдань студентами.

Предметом дослідження є розробка порталу для автоматичної перевірки лабораторних робіт. Цей портал є інструментом, створеним з метою автоматизації процесу перевірки та оцінки лабораторних робіт студентів. Предметом дослідження є розробка архітектури порталу, розробка алгоритмів для автоматичної перевірки програмного коду, визначення критеріїв оцінювання правильності та якості виконання робіт, а також інтерфейсу та функціональних можливостей порталу для студентів та викладачів.

1. ТЕОРЕТИЧНІ АСПЕКТИ РОЗРОБКИ ПОРТАЛУ ДЛЯ АВТОМАТИЧНОЇ ПЕРЕВІРКИ ЛАБОРАТОРНИХ РОБІТ.

Теоретична частина даної роботи передбачає дослідження і опис різних теоретичних аспектів, які становлять основу для розробки та функціонування такого порталу. Такі як:

- **Методи та алгоритми автоматичної перевірки**: розгляд різних методів та алгоритмів, що використовуються для автоматичної перевірки лабораторних робіт. Це може включати тестування програмного коду, аналіз виходів, порівняння з очікуваними результатами та виявлення помилок.
- **Системи управління лабораторними роботами**: вивчення існуючих систем управління лабораторними роботами, які використовуються в освітніх закладах. Аналіз їх функціональності, можливостей та обмежень для побудови ефективного та зручного порталу.
- **Безпека та захист даних**: врахування аспектів безпеки та захисту даних під час розробки порталу. Розгляд важливих аспектів, таких як автентифікація користувачів, контроль доступу до ресурсів та ін.
- **Оцінювання якості та ефективності:** розробка методів та критеріїв оцінювання якості та ефективності порталу. Визначення метрик, які використовуються для вимірювання точності перевірки, часу відповіді, продуктивності системи та задоволення користувачів.

1.1 Огляд наявних методів та підходів до автоматичної перевірки лабораторних робіт

Методи перевірки лабораторних робіт:

- Статичний аналіз коду: це процес виявлення помилок та проблем у коді без його виконання. Статичний аналіз включає перевірку синтаксису, стилю коду, потенційних помилок у пам'яті, і т.д. Інструменти статичного аналізу коду включають такі як SonarQube, PMD, Checkstyle, FindBugs та ін.
- Динамічний аналіз коду: це процес перевірки програмного коду під час його виконання. Це може включати перевірку вхідних та вихідних даних, перевірку виконання та використання ресурсів.
- Юніт-тестування: це підхід, при якому окремі частини програмного коду перевіряються для визначення того, чи вони правильно працюють. Це дозволяє автоматизувати процес перевірки коду та забезпечити, що він відповідає певним вимогам.
- Системи управління навчанням (LMS): системи, такі як Moodle або Canvas, мають вбудовані можливості автоматичної перевірки робіт, включаючи перевірку на плагіат.
- Онлайн платформи для перевірки коду: існує багато онлайн-платформ, таких як HackerRank, LeetCode, Codewars, Sphere Engine які надають можливість автоматичної перевірки коду.
- Системи перевірки плагіату: системи, такі як Turnitin або Plagscan, дозволяють виявляти плагіат в коді, перевіряючи його на співпадіння з іншими джерелами.

В ході дослідження існуючих порталів для автоматичної перевірки лаб. робіт було розглянуто портали SonarQube, Moodle, HackerRank, Turnitin та зроблено такі висновки щодо їх функціоналу:

SonarQube[2]

SonarQube є відкритою платформою для статичного аналізу коду, призначеною для виявлення помилок програмування, вразливостей безпеки, відсутності стандартів кодування та інших проблем в програмному забезпеченні. Цей інструмент надає зручний спосіб перевірки якості коду і поліпшити загальну стабільність та надійність програмного забезпечення. SonarQube пропонує набір правил, які перевіряють код на дотримання кращих практик програмування. Він може аналізувати код, написаний на різних мовах програмування, таких як Java, C#, C/C++, Python, JavaScript і багатьох інших. Статичний аналіз виконується шляхом перевірки коду на основі цих правил і надання звіту з результатами, включаючи виявлені проблеми та рекомендації щодо їх виправлення.

Переваги:

- Автоматизована перевірка якості коду.
- Підтримка багатьох мов програмування.
- Інтеграція з CI/CD системами.
- Візуалізація результатів аналізу.

Недоліки:

- Може вимагати значного часу на налаштування для специфічних проектів.
- Деякі можливості доступні тільки в платній версії.

Moodle[3]

Moodle - це відкрите програмне забезпечення для управління навчанням (Learning Management System, LMS), призначене для підтримки навчального процесу та сприяння ефективної взаємодії між викладачами та студентами в електронному середовищі. Воно дозволяє створювати та організовувати онлайн-курси, надавати навчальний матеріал, спілкуватися зі студентами, а також відстежувати їх академічні досягнення.

Moodle забезпечує широкий спектр функціональних можливостей, включаючи:

1. Створення курсів: викладачі можуть створювати структуровані курси з різними розділами та модулями, такими як лекції, завдання, тести, форуми для обговорень та інші.
2. Представлення навчального матеріалу: Moodle дозволяє завантажувати та розміщувати різні типи навчальних матеріалів, таких як текстові документи, презентації, відео, аудіо та інші ресурси, які можуть бути доступні для самостійного вивчення студентами.
3. Здійснення комунікації: Moodle надає інструменти для спілкування між викладачами та студентами, включаючи можливість обговорювати

питання та завдання у форумах, обмінюватися повідомленнями та використовувати чат для миттєвої комунікації.

4. Оцінювання та звітність: Moodle дозволяє викладачам створювати тести, опитування та завдання для оцінювання знань студентів. Він також надає інструменти для автоматичного оцінювання та створення звітів про академічні досягнення студентів.

Переваги:

- Широкий спектр інструментів для управління курсами.
- Інтеграція з системами перевірки плагіату.
- Можливість створювати автоматичні тести.

Недоліки:

- Інтерфейс може виглядати застарілим та складним для нових користувачів.
- Може вимагати спеціального налаштування для певних потреб.

HackerRank[4]

HackerRank - це онлайн-платформа для оцінювання та розвитку навичок програмування, яка пропонує використання задач програмування та викликів для тестування та розвитку вмінь розробників.

HackerRank надає бібліотеку завдань програмування з різних областей, таких як алгоритми, структури даних, штучний інтелект, бази даних, мережі та

багато інших. Ці завдання часто засновані на реальних викликах, з якими можна зіткнутися у своїй роботі. Вони можуть бути вирішені різними мовами програмування, такими як C++, Java, Python, JavaScript тощо. Кожне завдання на HackerRank має певну кількість тестових випадків, на яких перевіряється правильність розв'язку. Після виконання завдання, розв'язок автоматично перевіряється на цих тестах, а результати показуються користувачу. Також можуть бути надані додаткові коментарі та рекомендації щодо оптимізації та покращення розв'язку.

Крім викликів програмування, HackerRank також пропонує інші типи завдань, такі як математичні задачі, графічні завдання та практичні вправи, що допомагають розвивати ширший спектр навичок.

Переваги:

- Багато представлених вправ та задач.
- Інтегрована система ранжування.
- Автоматизована перевірка коду на правильність.

Недоліки:

- Може бути складно налаштувати для специфічних вимог курсу.
- Зосереджена на конкурсному програмуванні, що може бути не ідеальним для навчального середовища.

Turnitin[5]

Переваги:

- Високоефективна система перевірки плагіату.

- Інтеграція з багатьма LMS.

Недоліки:

- Не підходить для перевірки коду.
- Може давати не точні співпадіння.
- Висока вартість для установи.

Ці платформи виконують різні функції і важко їх порівняти безпосередньо. Однак, вони можуть бути інтегровані для створення комплексної системи автоматичної перевірки лабораторних робіт. Наприклад, Moodle може бути використано як основна платформа для подання робіт, яка інтегрована з SonarQube для перевірки якості коду та Turnitin для перевірки на плагіат. Hacker Rank може бути використаний як додатковий ресурс для практичного навчання та вдосконалення навичок програмування студентів.

1.2 Вимоги до архітектури порталу

Модульність: архітектура порталу повинна бути гнучкою і модульною, щоб дозволити легку інтеграцію нових компонентів та функціональності.

Безпека: Портал має враховувати основні принципи безпеки, такі як автентифікація, авторизація, захист даних та ін.

Швидкість та масштабованість: Архітектура порталу повинна бути здатна витримувати велику кількість користувачів та запитів, забезпечуючи стабільну роботу та відповідний час відгуку.

Інтеграція з іншими системами: портал повинен мати можливість легкої інтеграції з існуючими системами управління навчанням (LMS), системами перевірки коду та іншими сервісами.

Клієнторієнтованість: Архітектура порталу повинна дозволяти налаштування для певних вимог та потреб користувачів та установ.

Інтуїтивний інтерфейс: Архітектура порталу повинна передбачити створення користувацького інтерфейсу, який буде зручним для користувачів та відповідатиме сучасним стандартам дизайну.

Сумісність: Архітектура порталу повинна бути сумісна з різними операційними системами та браузерами.

1.3 Вибір інструментів для розробки та оцінки працездатності порталу

Основні інструменти, що використовувались в ході розробки - мова програмування Python, веб-фреймворк Django, база даних PostgreSQl.

Python: Вибір Python як основної мови програмування обумовлений його простотою, гнучкістю та великою кількістю доступних бібліотек. Python має сильну підтримку від спільноти, а також підходить для розробки веб-додатків, обробки даних та автоматизації.

Django[1]: Django є потужним веб-фреймворком для Python, який надає швидку та просту розробку веб-додатків. Django містить вбудовані інструменти для роботи з базами даних, аутентифікації, авторизації та інших типових завдань. Використання Django сприятиме прискоренню процесу розробки порталу.

PostgreSQL[6]: PostgreSQL є відкритою об'єктно-реляційною системою управління базами даних (СУБД), яка відома своєю надійністю, масштабованістю та продуктивністю. Використання PostgreSQL дозволить забезпечити ефективну роботу з даними, гарантувати безпеку інформації та інтегруватися з Django.

2. ПРОЕКТУВАННЯ ТА РОЗРОБКА ПОРТАЛУ ДЛЯ АВТОМАТИЧНОЇ ПЕРЕВІРКИ ЛАБОРАТОРНИХ РОБІТ

До етапів розробки порталу можна віднести:

- Аналіз вимог: визначення ключових функціональних та нефункціональних вимог до порталу. Це може включати збір вимог від основних зацікавлених сторін, таких як викладачі, студенти, адміністратори та інші.
- Проектування інтерфейсу: створення прототипу інтерфейсу користувача, щоб показати, як буде виглядати портал та як користувачі будуть взаємодіяти з ним. *В даній роботі було розроблено функціонал для вже існуючого інтерфейсу порталу.*
- Проектування архітектури: визначення структури системи, включаючи основні компоненти та їх взаємозв'язки.
- Розробка: розробка порталу відповідно до проекту, використовуючи вибрані мови програмування та фреймворки. Розробка зазвичай включає написання коду, інтеграцію з іншими системами, налаштування бази даних та інше.
- Тестування: проведення різних типів тестування, включаючи модульне тестування, інтеграційне тестування, тестування продуктивності, навантажувальне тестування, тестування безпеки та тестування користувачів, щоб переконатися, що портал працює правильно та відповідає всім вимогам.

2.1 Аналіз вимог користувачів до порталу

Можливі вимоги до порталу від студентів:

Можливість подавати лабораторні роботи в різних форматах (наприклад, .doc, .pdf, .zip для коду тощо).

Отримання автоматичного зворотнього зв'язку про статус перевірки роботи.

Отримання звіту про результати перевірки.

Можливість перегляду вимог до лабораторних робіт та критеріїв оцінювання.

Можливі вимоги до порталу від викладачів:

Автоматична перевірка поданих робіт на плагіат.

Автоматична перевірка якості коду в програмних роботах.

Можливість задавати вимоги до лабораторних робіт та критерії оцінювання.

Можливість бачити прогрес кожного студента, включаючи подані роботи, оцінки та відгуки.

Можливість взаємодії зі студентами, включаючи надання зворотнього зв'язку та відповідей на питання.

Можливі вимоги до порталу від адміністраторів порталу:

Можливість управління користувачами, включаючи додавання, видалення та редагування профілів студентів та викладачів.

Можливість налаштовувати портал, включаючи параметри перевірки, параметри безпеки та ін.

Можливість моніторингу активності на порталі та виявлення можливих проблем.

2.2 Опис архітектури порталу та вибір технологій

Розроблений портал базується на фреймворку Django і включає два основні додатки - "Users" та "Courses", які відповідають за реєстрацію, авторизацію користувачів та управління курсами відповідно.

Додаток "Users" має наступні функціональні можливості: реєстрацію та авторизацію користувачів на порталі. Він також надає можливість налаштування профілю користувача, де він може змінити певні настройки свого аккаунту, включаючи пароль.

Додаток "Courses" забезпечує викладачам можливість перегляду та управління курсами. Він дозволяє викладачам створювати нові курси та додавати завдання до них. Зі сторони студентів, додаток "Courses" відображає список доступних курсів та завдань, які студенти можуть виконати. Окрім цього, додаток забезпечує можливість автоматичної перевірки відповідей, які студенти надсилають для завдань.

Застосування фреймворку Django дозволяє зручно та ефективно реалізувати функціональність порталу. Він надає потужні інструменти для розробки, такі як вбудована система аутентифікації та управління користувачами, зручний шаблон для відображення сторінок, інструменти для роботи з базою даних та інші.

Загалом, розроблений портал на основі фреймворку Django забезпечує зручну реєстрацію та авторизацію користувачів, управління курсами, відображення завдань та автоматичну перевірку відповідей студентів. Він створений з метою спростити та поліпшити процеси навчання та перевірки лабораторних робіт.

Демонстрація роботи описаних вище додатків.

USERS

Сторінка авторизації (рис.2.2.1):

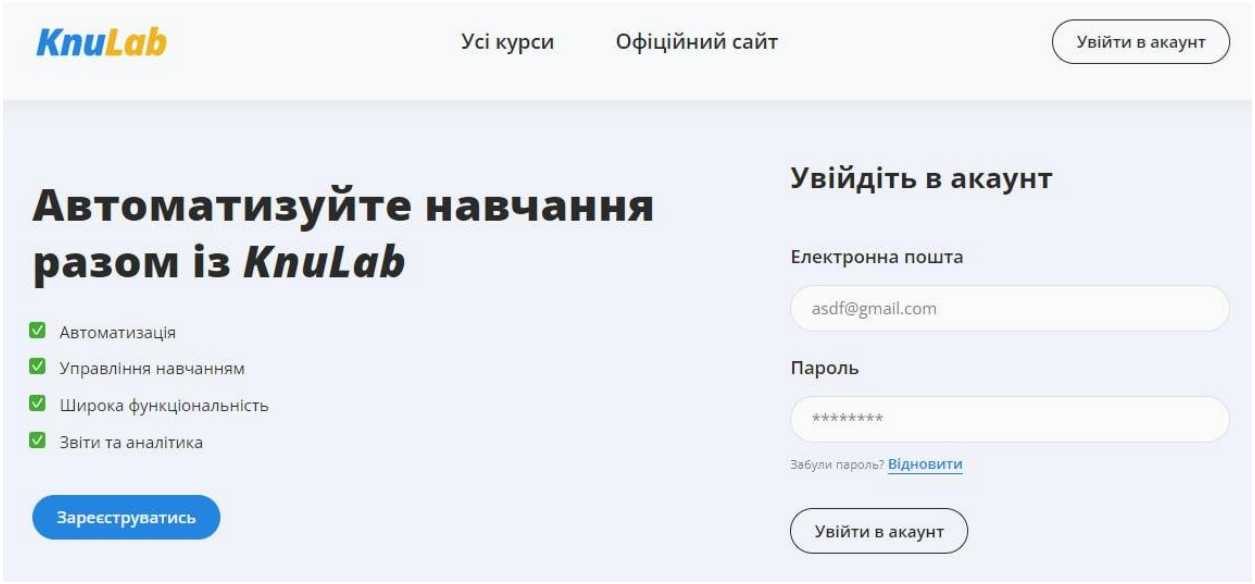


Рис. 2.2.1

Сторінка реєстрації (рис.2.2.2):

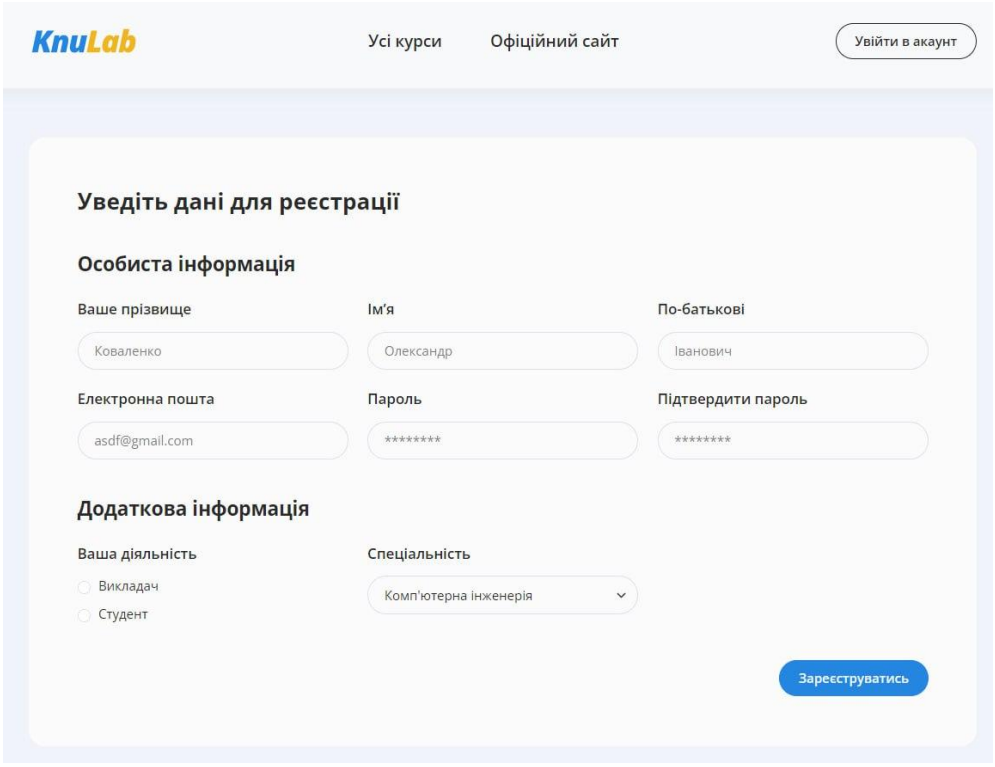


Рис. 2.2.2

COURSES

Головна сторінка викладача (рис.2.2.3):

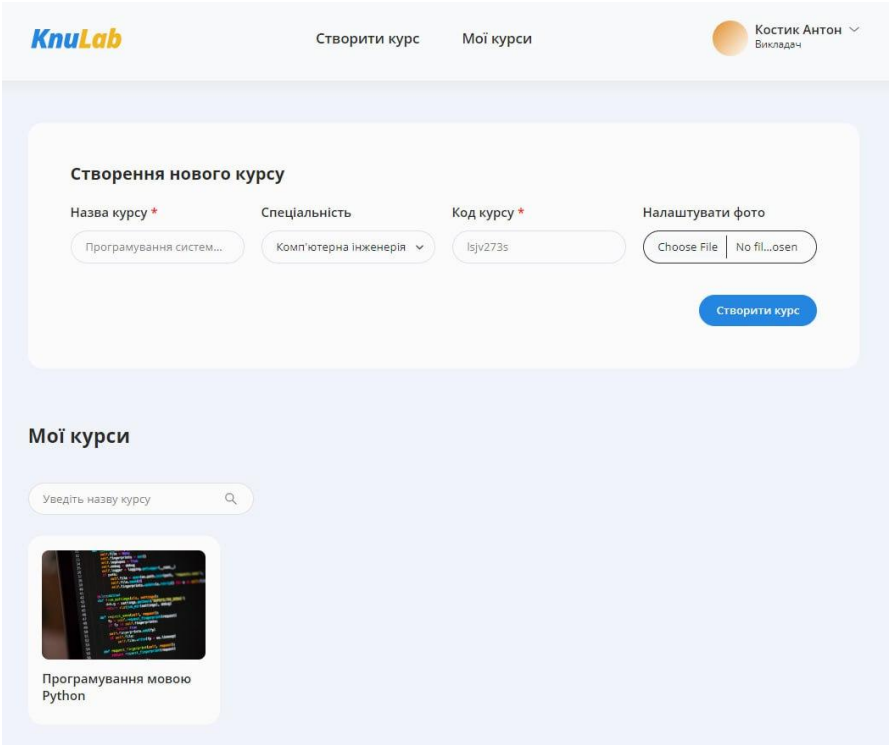


Рис. 2.2.3

Сторінка курсу зі сторони Викладача (рис.2.2.4)

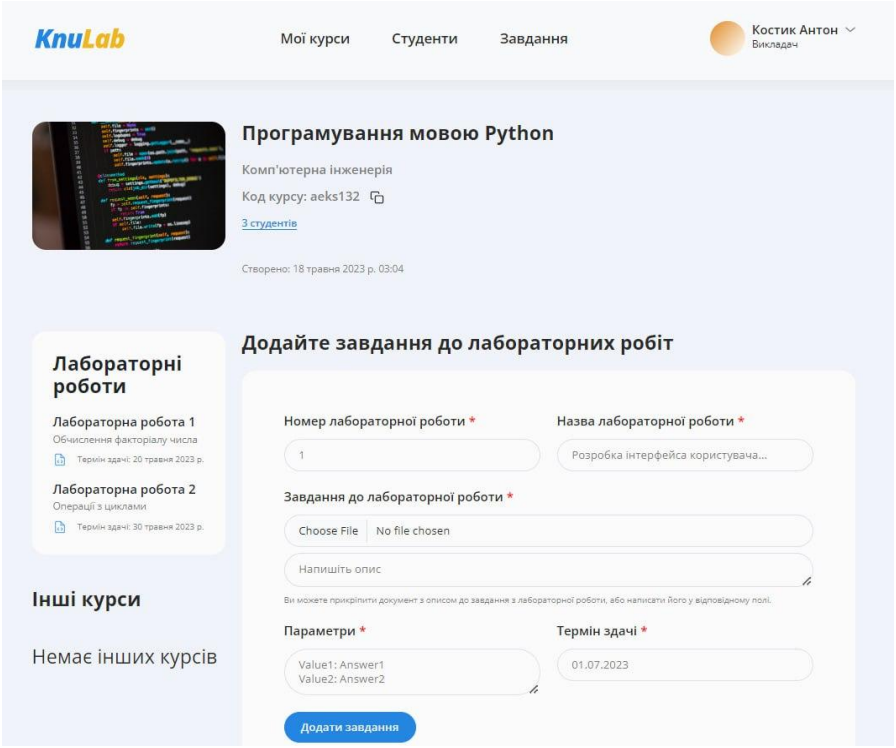


Рис. 2.2.4

Список студентів на поточному курсі (рис.2.2.5):

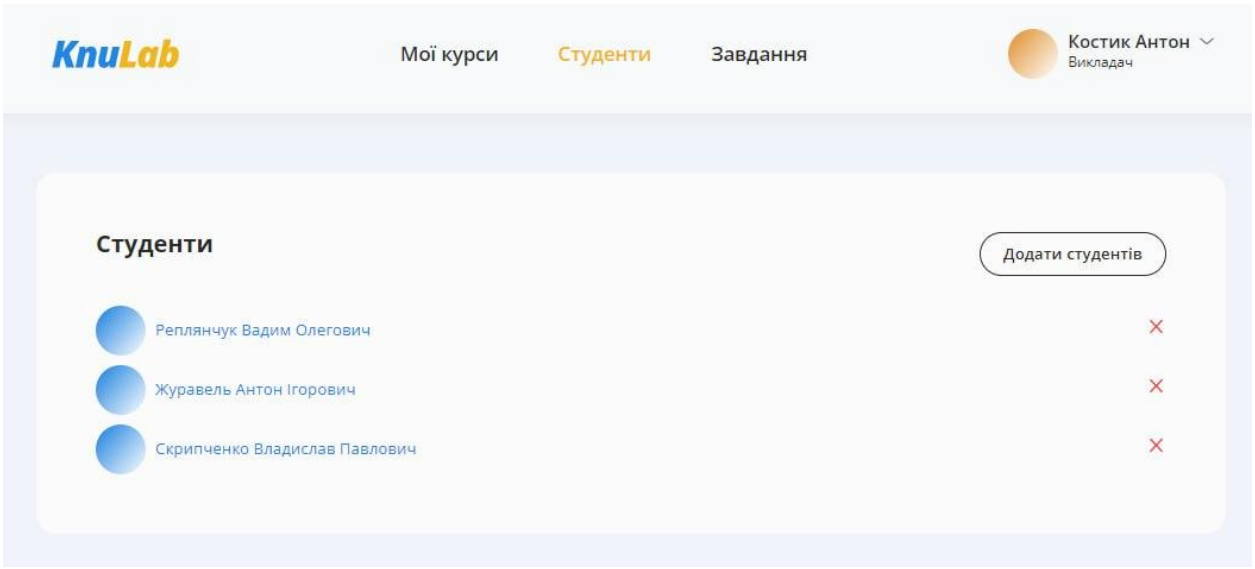


Рис. 2.2.5

Сторінка результатів перевірки для відповідного студента (зі сторони Викладача) – рис.2.2.6

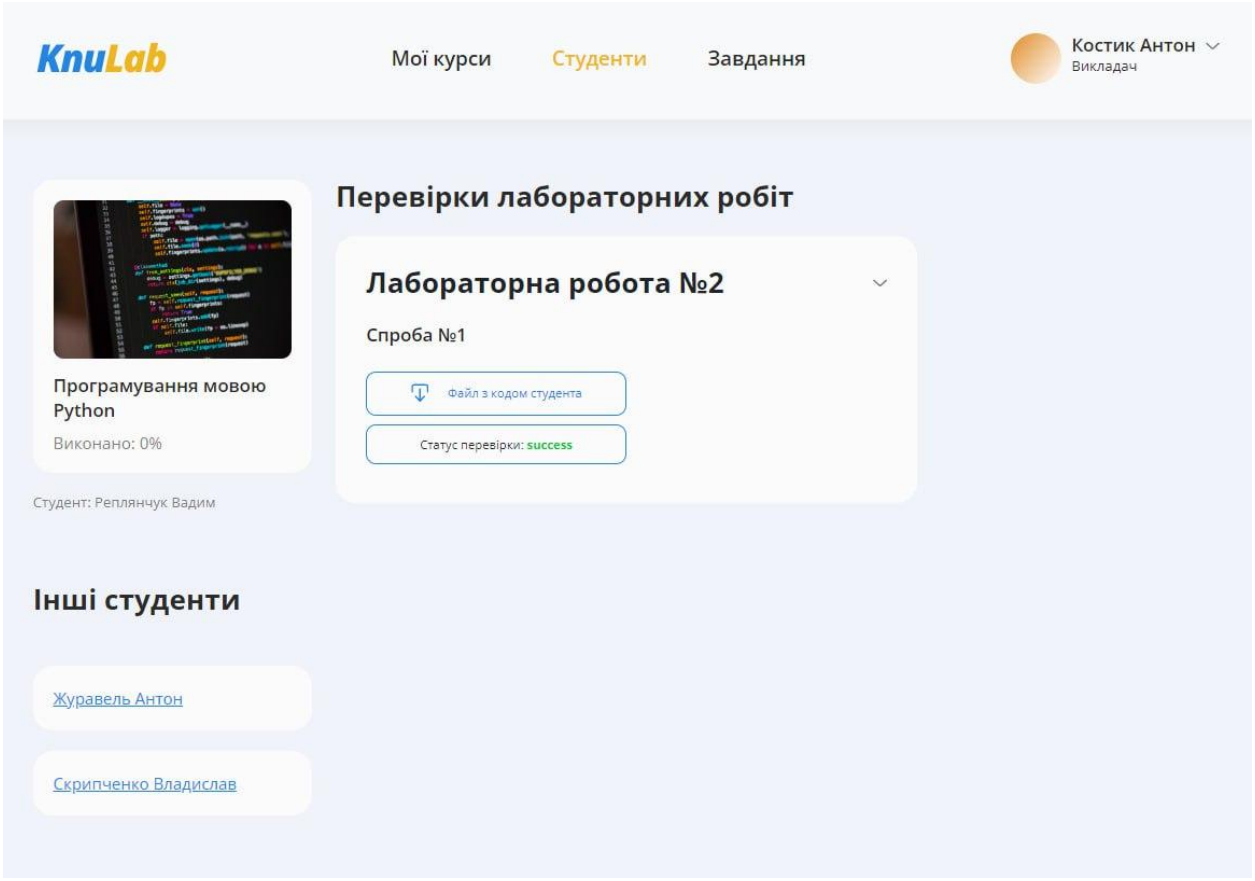


Рис. 2.2.6

Головна сторінка Студента (рис.2.2.7)

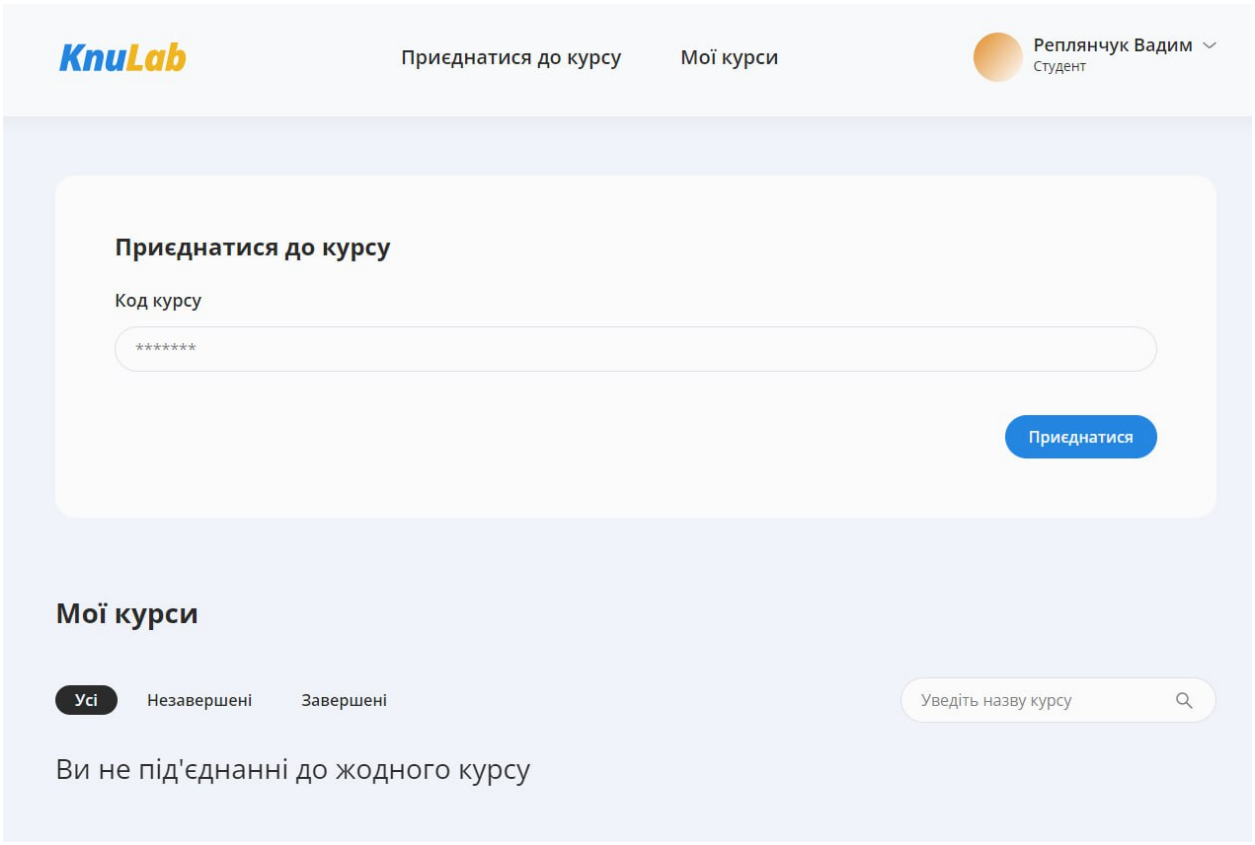


Рис. 2.2.7

Приєднання до нового курсу, який створив Викладач через Код курсу (рис.2.2.8):

Рис. 2.2.8

Сторінка курсу зі сторони Студента (рис.2.2.9):

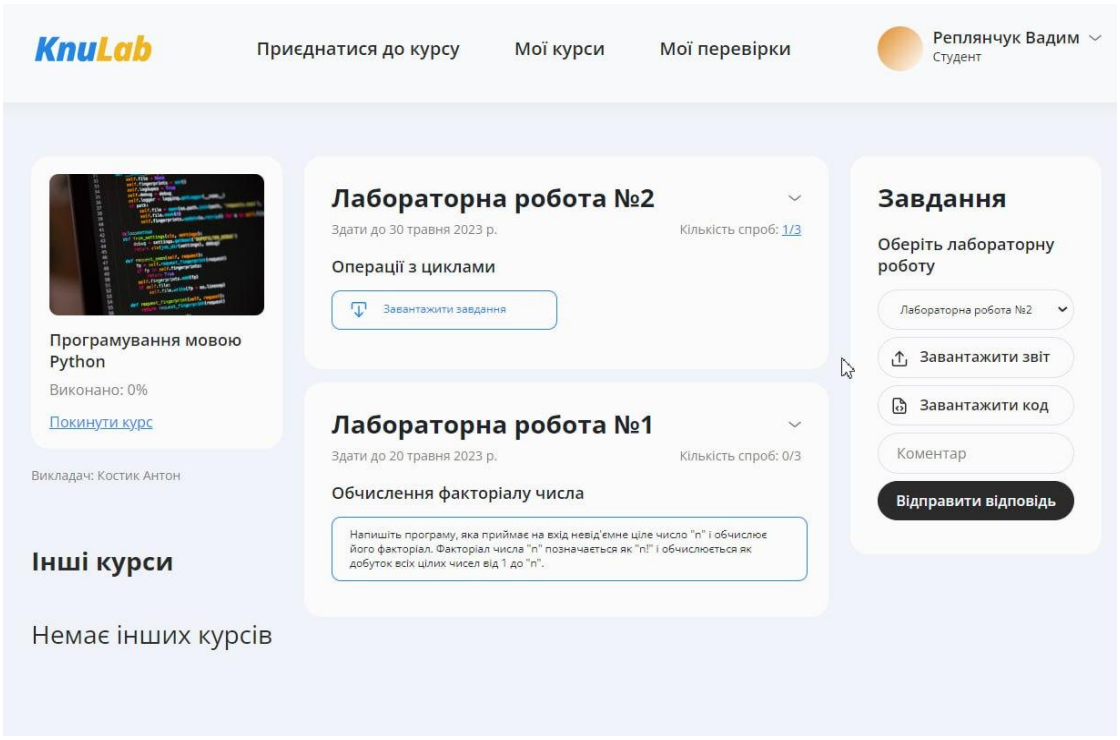


Рис. 2.2.9

Сторінка результатів перевірки лабораторних робіт зі сторони Студента (рис.2.2.10):

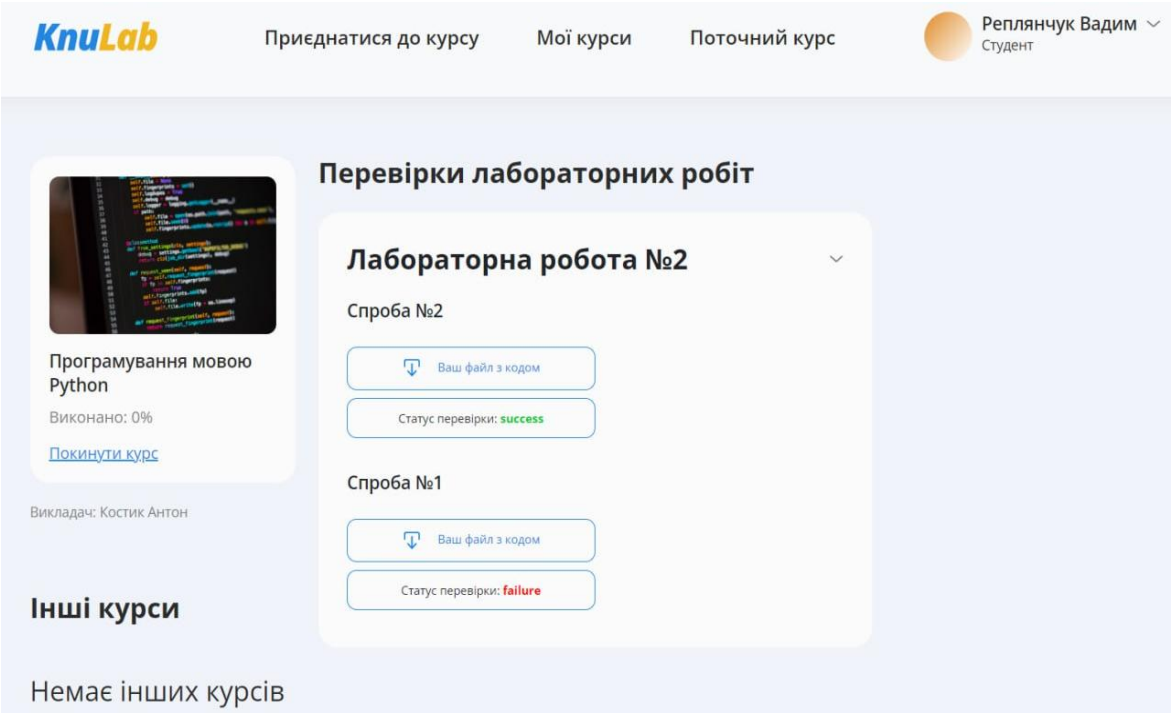


Рис. 2.2.10

2.3 Розробка бази даних для зберігання результатів перевірки лабораторних робіт

Основні задачі розробки Бази даних:

- Визначення структури даних: необхідно визначити, які дані необхідно зберігати для кожної лабораторної роботи та її результатів. Наприклад, це можуть бути ідентифікатори студента та викладача, назва роботи, оцінка, дата та інші відповідні атрибути.
- Вибір типу бази даних: необхідно обрати тип бази даних для зберігання результатів перевірки лабораторних робіт. Наприклад, можна використати реляційну базу даних, таку як MySQL або PostgreSQL, яка забезпечує структуроване зберігання даних.
- Розробка схеми бази даних: необхідно створити схему бази даних, яка відображає структуру даних та зв'язки між ними. Наприклад, можна створити таблиці для зберігання інформації про студентів, викладачів, лабораторні роботи та результати перевірки.
- Розробка запитів та процедур: необхідно розробити запити та процедури для ефективного взаємодії з базою даних. Наприклад, можна реалізувати запити для отримання результатів перевірки конкретного студента, викладача або лабораторної роботи.
- Забезпечення безпеки даних: необхідно врахувати питання безпеки при розробці бази даних. Наприклад, можна використати механізми аутентифікації та авторизації для обмеження доступу до даних тільки авторизованим користувачам.
- Резервне копіювання та відновлення даних: необхідно розробити стратегію резервного копіювання бази даних, щоб уникнути втрати даних у разі

непередбачуваних ситуацій. Також важливо розробити процедуру відновлення даних в разі потреби.

- Оптимізація продуктивності: необхідно провести оптимізацію бази даних для покращення продуктивності та швидкості доступу до даних. Це може включати індексацію таблиць, використання кешування, оптимізацію запитів та інші методи.
- Після розробки бази даних для зберігання результатів перевірки лабораторних робіт, вона буде інтегрована з розробленим порталом, що дозволить зберігати, оновлювати та отримувати доступ до результатів перевірки лабораторних робіт зручним та ефективним способом.
- Демонстрація розробленої бази даних.

Для розробки бази даних було обрано базу даних **PostgreSQL**.

PostgreSQL є високорозвиненою реляційною базою даних, спрямованою на повну відповідність стандартам і можливість розширення. Ця система баз даних, також відома як Postgres, стежить за виконанням стандартів SQL ANSI/ISO.

Однією з відмінних рис PostgreSQL в порівнянні з іншими системами керування баз даних полягає у його об'єктно-орієнтованому функціоналі, включаючи повну підтримку концепції ACID (атомарність, послідовність, ізоляція, стійкість).

Завдяки потужній технології, на якій базується PostgreSQL, вона ефективно впорається з одночасною обробкою багатьох завдань. Підтримка конкурентності реалізована з використанням методу контролю конкурентності мультиверсійного керування (MVCC), що також забезпечує відповідність принципам ACID.

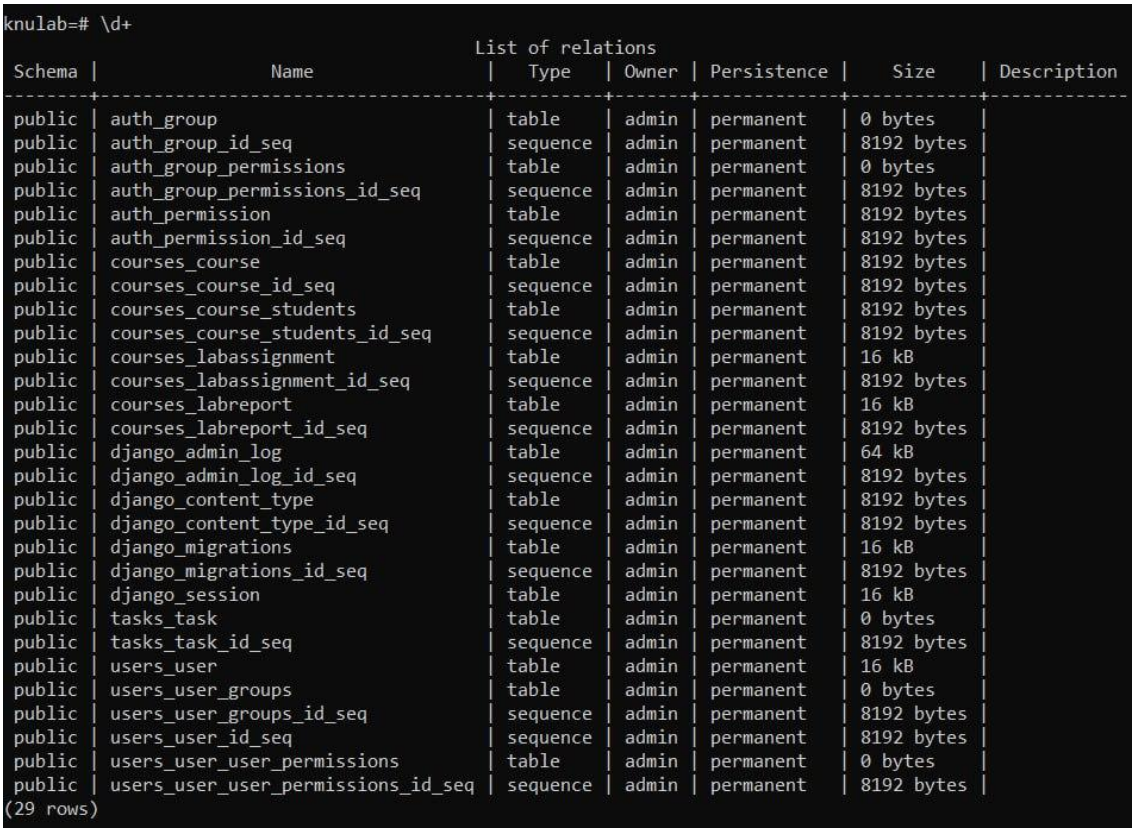
Незважаючи на те, що PostgreSQL не така популярна, як MySQL, існує багато сторонніх інструментів і бібліотек, які полегшують роботу з цією системою баз даних.

Для забезпечення ефективного зберігання та управління результатами перевірки лабораторних робіт на розробленому порталі для автоматичної перевірки, необхідно створити відповідну базу даних, використовуючи PostgreSQL.

Переваги

- Повна сумісність з SQL стандартами.
- Активна та досвідчена спільнота: PostgreSQL підтримується відомою спільнотою 24/7.
- Підтримка сторонніми організаціями: незважаючи на дуже сучасні функції, PostgreSQL використовується в багатьох інструментах, пов'язаних з РСКБД.
- Розширюваність: PostgreSQL може бути програмно розширена за допомогою збережених процедур.
- Об'єктно-орієнтованість: PostgreSQL є не просто реляційною системою управління базами даних, але й об'єктно-орієнтованою СУБД.

Основна база даних, з якою взаємодіє портал (рис 2.3.0):



```
knulab=# \d+
                                    List of relations
 Schema |                Name                |   Type   | Owner | Persistence |    Size    | Description
--------+------------------------------------+----------+-------+-------------+------------+-------------
 public | auth_group                         | table    | admin | permanent   | 0 bytes    |
 public | auth_group_id_seq                  | sequence | admin | permanent   | 8192 bytes |
 public | auth_group_permissions             | table    | admin | permanent   | 0 bytes    |
 public | auth_group_permissions_id_seq      | sequence | admin | permanent   | 8192 bytes |
 public | auth_permission                    | table    | admin | permanent   | 8192 bytes |
 public | auth_permission_id_seq             | sequence | admin | permanent   | 8192 bytes |
 public | courses_course                     | table    | admin | permanent   | 8192 bytes |
 public | courses_course_id_seq              | sequence | admin | permanent   | 8192 bytes |
 public | courses_course_students            | table    | admin | permanent   | 8192 bytes |
 public | courses_course_students_id_seq     | sequence | admin | permanent   | 8192 bytes |
 public | courses_labassignment              | table    | admin | permanent   | 16 kB      |
 public | courses_labassignment_id_seq       | sequence | admin | permanent   | 8192 bytes |
 public | courses_labreport                  | table    | admin | permanent   | 16 kB      |
 public | courses_labreport_id_seq           | sequence | admin | permanent   | 8192 bytes |
 public | django_admin_log                   | table    | admin | permanent   | 64 kB      |
 public | django_admin_log_id_seq            | sequence | admin | permanent   | 8192 bytes |
 public | django_content_type                | table    | admin | permanent   | 8192 bytes |
 public | django_content_type_id_seq         | sequence | admin | permanent   | 8192 bytes |
 public | django_migrations                  | table    | admin | permanent   | 16 kB      |
 public | django_migrations_id_seq           | sequence | admin | permanent   | 8192 bytes |
 public | django_session                     | table    | admin | permanent   | 16 kB      |
 public | tasks_task                         | table    | admin | permanent   | 0 bytes    |
 public | tasks_task_id_seq                  | sequence | admin | permanent   | 8192 bytes |
 public | users_user                         | table    | admin | permanent   | 16 kB      |
 public | users_user_groups                  | table    | admin | permanent   | 0 bytes    |
 public | users_user_groups_id_seq           | sequence | admin | permanent   | 8192 bytes |
 public | users_user_id_seq                  | sequence | admin | permanent   | 8192 bytes |
 public | users_user_user_permissions        | table    | admin | permanent   | 0 bytes    |
 public | users_user_user_permissions_id_seq | sequence | admin | permanent   | 8192 bytes |
(29 rows)
```

Рис. 2.3.0

Вона містить 4 основних таблиці які містять дані про зареєстрованих користувачів, про існуючі курси, про створені лабораторні роботи в курсах та про звіти лабораторних робіт відповідно.

Дані про користувачів (рис.2.3.1, рис.2.3.2)

```python
class User(AbstractBaseUser, PermissionsMixin):
    email = models.EmailField(unique=True)
    first_name = models.CharField(max_length=30, blank=True)
    last_name = models.CharField(max_length=30, blank=True)
    middle_name = models.CharField(max_length=30, blank=True)
    is_active = models.BooleanField(default=True)
    is_staff = models.BooleanField(default=False)
    is_superuser = models.BooleanField(default=False)

    is_student = models.BooleanField(default=False)
    is_teacher = models.BooleanField(default=False)
    ROLE_CHOICES = [
        ('is_student', 'Студент'),
        ('is_teacher', 'Викладач'),
    ]
    role = models.CharField(max_length=20, choices=ROLE_CHOICES, default='is_student')

    specialty = models.CharField(max_length=100, blank=True)

    USERNAME_FIELD = 'email'
    REQUIRED_FIELDS = []

    objects = UserManager()

    def get_full_name(self):
        return self.last_name + ' ' + self.first_name

    def get_short_name(self):
        return self.first_name

    def __str__(self):
        return self.email
```

Рис. 2.3.1

```
knulab=# \d users_user
                                  Table "public.users_user"
    Column    |           Type           | Collation | Nullable |              Default
--------------+--------------------------+-----------+----------+-----------------------------------
 id           | bigint                   |           | not null | generated by default as identity
 password     | character varying(128)   |           | not null |
 last_login   | timestamp with time zone |           |          |
 email        | character varying(254)   |           | not null |
 first_name   | character varying(30)    |           | not null |
 last_name    | character varying(30)    |           | not null |
 is_active    | boolean                  |           | not null |
 is_staff     | boolean                  |           | not null |
 is_superuser | boolean                  |           | not null |
 is_student   | boolean                  |           | not null |
 is_teacher   | boolean                  |           | not null |
 specialty    | character varying(100)   |           | not null |
 middle_name  | character varying(30)    |           | not null |
 role         | character varying(20)    |           | not null |
```

Рис. 2.3.2 Таблиця users в базі даних

Дані про курси (рис.2.3.3, рис.2.3.4)

```python
class LabAssignment(models.Model):
    number = models.IntegerField(unique=True)
    title = models.CharField(max_length=255, unique=True)
    description = models.TextField(blank=True, null=True)
    task_file = models.FileField(upload_to='assets/static/other/lab_tasks/', blank=True, null=True)
    deadline = models.DateField()
    parameters = models.TextField()
    course = models.ForeignKey(Course, on_delete=models.CASCADE)

    def __str__(self):
        return f"Лабораторна робота {self.number} - {self.title}"
```

Рис. 2.3.3

```
knulab=# \d courses_course
                                Table "public.courses_course"
   Column    |           Type           | Collation | Nullable |              Default
-------------+--------------------------+-----------+----------+-----------------------------------
 id          | bigint                   |           | not null | generated by default as identity
 title       | character varying(100)   |           | not null |
 specialty   | character varying(100)   |           | not null |
 code        | character varying(10)    |           | not null |
 created_at  | timestamp with time zone |           | not null |
 image       | character varying(100)   |           | not null |
 teacher_id  | bigint                   |           | not null |
```

Рис. 2.3.4 Таблиця courses в базі даних

Дані про створені лабораторні роботи на курсі (рис.2.3.5, рис.2.3.6)

```python
class LabAssignment(models.Model):
    number = models.IntegerField(unique=True)
    title = models.CharField(max_length=255, unique=True)
    description = models.TextField(blank=True, null=True)
    task_file = models.FileField(upload_to='assets/static/other/lab_tasks/', blank=True, null=True)
    deadline = models.DateField()
    parameters = models.TextField()
    course = models.ForeignKey(Course, on_delete=models.CASCADE)

    def __str__(self):
        return f"Лабораторна робота {self.number} - {self.title}"
```

Рис. 2.3.5

```
knulab=# \d courses_labassignment
                          Table "public.courses_labassignment"
   Column    |          Type          | Collation | Nullable |              Default
-------------+------------------------+-----------+----------+-----------------------------------
 id          | bigint                 |           | not null | generated by default as identity
 number      | integer                |           | not null |
 title       | character varying(255) |           | not null |
 description | text                   |           |          |
 task_file   | character varying(100) |           |          |
 deadline    | date                   |           | not null |
 parameters  | text                   |           | not null |
 course_id   | bigint                 |           | not null |
```

Рис. 2.3.6 Таблиця labassigment в базі даних

Дані про звіти до лабораторних робіт (рис.2.3.7, рис.2.3.8)

```python
class LabReport(models.Model):
    course = models.ForeignKey(Course, on_delete=models.CASCADE, verbose_name='Курс')
    number = models.IntegerField(verbose_name='Номер лабораторної роботи')
    report_file = models.FileField(upload_to='assets/static/other/lab_reports/', verbose_name='Файл з звітом', blank=True, null=True)
    code_file = models.FileField(upload_to='assets/static/other/lab_codes/',verbose_name='Файл з кодом', blank=True, null=True)
    comment = models.TextField(verbose_name='Коментар', blank=True)
    student = models.ForeignKey(User, on_delete=models.CASCADE, verbose_name='Студент')
    submitted_at = models.DateTimeField(auto_now_add=True, verbose_name='Дата завантаження відповіді')
    attempt = models.IntegerField(default=0)
    check_status = models.CharField(max_length=100, default='pending')
    max_attempts = models.IntegerField(default=3)

    def save(self, *args, **kwargs):
        lab_assignment = LabAssignment.objects.filter(number=self.number).first()
        if lab_assignment:
            self.number = lab_assignment.number
            self.course = lab_assignment.course
        super().save(*args, **kwargs)

    def __str__(self):
        return f'Відповідь до ЛР №{self.number} від {self.student.email}'
```

Рис. 2.3.7

```
knulab=# \d courses_labreport
                             Table "public.courses_labreport"
    Column    |           Type           | Collation | Nullable |              Default
--------------+--------------------------+-----------+----------+-----------------------------------
 id           | bigint                   |           | not null | generated by default as identity
 number       | integer                  |           | not null |
 report_file  | character varying(100)   |           |          |
 code_file    | character varying(100)   |           |          |
 comment      | text                     |           | not null |
 submitted_at | timestamp with time zone |           | not null |
 attempt      | integer                  |           | not null |
 check_status | character varying(100)   |           | not null |
 max_attempts | integer                  |           | not null |
 course_id    | bigint                   |           | not null |
 student_id   | bigint                   |           | not null |
```

Рис. 2.3.8 Таблиця labreport в базі даних

2.4 Розробка модулів для автоматичної перевірки коду та обробки результатів

Для забезпечення автоматичної перевірки коду студентів у лабораторних роботах та обробки результатів необхідно розробити відповідні модулі в рамках розробленого порталу

Основні задачі розробки модулів для автоматичної перевірки коду та обробки результатів:

1. Вибір інструментів для автоматичної перевірки: необхідно обрати інструменти або фреймворки, які дозволять виконувати автоматичну перевірку коду студентів. Це можуть бути засоби для компіляції, виконання тестів, аналізу коду або інші інструменти, які відповідають вимогам проекту.

2. Розробка модуля автоматичної перевірки: необхідно розробити модуль, який забезпечує автоматичну перевірку коду студентів згідно з вимогами лабораторної роботи. Цей модуль може включати компіляцію та виконання коду, порівняння результатів з очікуваними значеннями, виявлення та обробку помилок та інші функціональності.

3. Розробка модуля обробки результатів: необхідно розробити модуль, який забезпечує обробку та зберігання результатів перевірки. Цей модуль може включати збереження результатів у базі даних, створення звітів або оцінок для студентів, генерацію повідомлень про помилки або підказок та інші функціональності.

4. Тестування модулів: після розробки модулів для автоматичної перевірки коду та обробки результатів необхідно провести їх тестування. Це допоможе переконатися, що модулі працюють правильно, відповідають вимогам та не мають помилок.

Розробка цих модулів дозволить автоматизувати процес перевірки коду студентів та ефективно обробляти результати, спрощуючи роботу викладачів та забезпечуючи швидкий зворотній зв'язок для студентів.

В даній роботі було використано **Sphere Engine**[7] **- Compile API**.

Sphere Engine - Compile API (Application Programming Interface) є програмним інтерфейсом, який надає можливість компіляції та виконання програмного коду у віртуальному середовищі. Цей API розроблений компанією Sphere Engine і призначений для розробників програмного забезпечення та платформ для онлайн-перевірки коду.

Основними функціональними можливостями Sphere Engine - Compile API є:

1. **Компіляція**: API дозволяє компілювати код, написаний на різних мовах програмування, таких як C, C++, Java, Python та інші. Він підтримує використання різних компіляторів та інтерпретаторів для кожної мови, забезпечуючи гнучкість та універсальність.

2. **Виконання**: Sphere Engine - Compile API дозволяє виконувати компіляцію коду в контрольованому середовищі. Це дозволяє безпечно виконувати код користувачів безпосередньо на сервері API, забезпечуючи ізоляцію та безпеку.

3. **Керування введенням-виведенням**: API надає можливість передачі вхідних даних для програмного коду та отримання результатів виведення. Це дозволяє виконувати тестування та отримувати результати виконання коду для подальшого аналізу.

4. **Обмеження ресурсів**: Sphere Engine - Compile API дозволяє встановлювати обмеження на час виконання коду, обсяг пам'яті, кількість операцій вводу-виводу та інші ресурси. Це дозволяє контролювати продуктивність та безпеку виконання коду в межах заданих параметрів.

Sphere Engine - Compile API використовується в різних областях, включаючи онлайн-курси програмування, платформи для спільного розв'язування задач, змагання з програмування та в інших сферах, де потрібна автоматична компіляція та виконання коду. Цей API допомагає спростити та автоматизувати процес перевірки та оцінки програмного коду, забезпечуючи зручний та ефективний спосіб роботи з програмними мовами та їх виконанням у віртуальному середовищі.

Демонстрація роботи автоматизованої перевірки робіт

```
# Перевірка коду лабораторної роботи
if lab_report.code_file:

    # Отримання шляху до файлу з кодом
    code_file_path = lab_report.code_file.name

    # Визначення мови програмування, якою написаний код в файлі
    file_extension = code_file.name.split('.')[-1].lower()
    if file_extension=='py':
        compilerId = 116
    elif file_extension=='cpp':
        compilerId = 1
    elif file_extension=='cs':
        compilerId = 86
    elif file_extension=='go':
        compilerId = 114
    elif file_extension=='sh':
        compilerId = 28
    elif file_extension=='jar':
        compilerId = 10
    elif file_extension=='php':
        compilerId = 29
```

Рис.2.4.0

Даний код (рис.2.4.0) перевіряє наявність файла з кодом у звіті про лабораторну роботу (змінна ***lab_report.code_file***). Якщо файл з кодом існує, код продовжує виконуватися.

Далі код отримує шлях до файлу з кодом (змінна ***code_file_path***) та визначає мову програмування, на якій написаний цей код, на основі розширення файлу. Для цього використовується метод ***split()*** для отримання розширення файлу та переведення його до нижнього регістру (***lower()***).

Далі використовується умовна конструкція (***if-elif-else***) для встановлення відповідного значення ***compilerId***, яке визначає ідентифікатор компілятора для відповідної мови програмування. Наприклад, якщо розширення файлу є "py", то ***compilerId*** встановлюється як **116**, що відповідає компілятору для мови Python.

Отримання ***compilerId*** може бути використано в подальшому коді для використання відповідного компілятора при перевірці коду лабораторної роботи.

```
    # Запис вмісту файлу з кодом в змінну "code"
    with open(code_file_path, 'r') as file:
        code = file.read()

    # Отримання параметрів, які визначив Викладач для поточної лабораторної роботи
    lab_assignment = LabAssignment.objects.get(number=lab_report.number)
    parameters = lab_assignment.parameters
    parameter_lines = parameters.split('\n') # Розділення декількох параметрів за допомогою розділювача переносу рядка

    # Парсинг параметрів, які визначив Викладач
    parameters = {}
    parameter_values = ""
    parameter_answers = ""
    for line in parameter_lines:
        line = line.strip('\r')
        parts = line.split(': ') # Розділення параметра на parameter_value та parameter_answer через розділювач ": "
        if len(parts) == 2:
            parameter_value = parts[0].strip()
            parameter_answer = parts[1].strip()
            parameters[parameter_value] = parameter_value
            parameter_values += f"{parameter_value}\n"
            parameter_answers += f"{parameter_answer} "

    existing_lab_report = LabReport.objects.get(pk=lab_report.pk)

    # Запуск перевірки лабораторної роботи та виведення результату перевірки
    result = run_lab_check(code, compilerId, parameter_values, parameter_answers, existing_lab_report)
    if not result['success']:
        messages.error(request, result['message'])
    else:
        messages.success(request, result['message'])
    return redirect('task_check', pk=pk)

else:
    messages.success(request, "Звіт до лабораторної роботи успішно завантажено.")
return redirect('task_detail', pk=pk)
```

Рис.2.4.1

Цей код (рис. 2.4.1) виконує наступні дії:

1. Відкриває файл зі шляхом `***code_file_path***` у режимі читання (`'r'`) за допомогою конструкції `***open(code_file_path, 'r') as file***`.

2. Зчитує вміст файлу та присвоює його змінній `***code***` за допомогою `***file.read()***`.

3. Отримує параметри, які визначив викладач для поточної лабораторної роботи.

4. Розбиває рядок `***parameters***` на декілька параметрів, використовуючи символ переносу рядка як розділювальний символ (`***parameters.split('\n')`).**

5. Обробляє кожен параметр, розділенням його на значення ***(`parameter_value`)*** та відповідь ***(`parameter_answer`)*** за допомогою розділювача `**':'**`.

6. Записує значення та відповідь кожного параметра у словник `***parameters***`.

7. Формує рядки `***parameter_values***` та `***parameter_answers***` зі значеннями та відповідями параметрів.

8. Отримує існуючий звіт про лабораторну роботу за його унікальним ідентифікатором (`***lab_report.pk***`).

9. Викликає функцію `***run_lab_check***` з передачею коду, ідентифікатора компілятора, значень параметрів, відповідей та існуючого звіту про роботу.

10. Якщо результат перевірки не успішний (`***result['success']***` дорівнює `**False**`) та виводить помилку повідомлення через `***messages.error()***`.

11. Якщо результат перевірки успішний ***(`result['success']`*** дорівнює `**True**`) та виводить успішне повідомлення через `***messages.success()***`.

12. Перенаправляє користувача на відповідну сторінку залежно від результату перевірки ***(`redirect('task_check', pk=pk***)` або `***redirect('task_detail', pk=pk)`).**

Отже, цей код зчитує код з файлу, обробляє параметри, викликає функцію для перевірки лабораторної роботи та виводить відповідні повідомлення про результат перевірки.

Основна функція ***run_lab_check*** знаходиться в файлі **labcheck.py**. Повний код знаходиться в розділі Додатки.

Отже, код цього файлу представляє функцію під назвою "***run_lab_check***", яка виконує перевірку лабораторної роботи. Основна мета функції - перевірити коректність виконання програмного коду у віртуальному середовищі за допомогою Sphere Engine - Compile API.

Функція приймає наступні параметри:

- **source**: код лабораторної роботи, який потрібно перевірити

- **compilerId**: ідентифікатор компілятора, який буде використовуватися для компіляції коду

- **input**: вхідні дані, необхідні для виконання коду

- **answers**: правильні відповіді, які має повернути програма

- **existing_lab_report**: існуючий звіт про лабораторну роботу, який містить інформацію про стан перевірки

Код використовує модулі "**requests**" та "**json**" для здійснення HTTP-запитів та обробки даних у форматі JSON.

Основні кроки, які виконує функція:

1. Формує URL для виклику Sphere Engine - Compile API, використовуючи отриманий доступ до API.

2. Відправляє POST-запит на компіляцію коду, передаючи необхідні параметри.

3. Отримує унікальний ідентифікатор підключення для перевірки стану компіляції.

4. Запитує стан компіляції до тих пір, поки він не буде завершений або виникне помилка.

5. Перевіряє результат компіляції та виконання коду, отримує вихідні дані та результати.

6. Порівнює вихідні дані з правильними відповідями та формує результат перевірки.

7. Зберігає статус перевірки в існуючий звіт про лабораторну роботу.

8. Повертає результат перевірки у форматі JSON, який містить інформацію про успішність та повідомлення.

Цей код дозволяє автоматично перевіряти код лабораторних робіт, порівнюючи його результати з правильними відповідями та зберігаючи стан перевірки до відповідного звіту про роботу.

2.5 Розробка автентифікації та налаштування акаунтів на порталі

Основні задачі розробки аутентифікації та налаштування акаунтів:

Вибір методів аутентифікації: необхідно вибрати методи автентифікації, які відповідають потребам та вимогам проекту.

Розробка системи управління акаунтами: необхідно розробити систему управління акаунтами, яка дозволить користувачам створювати акаунти на порталі, виконувати автентифікацію та керувати своїми налаштуваннями. Це може включати створення сторінки реєстрації, сторінки входу, сторінки налаштувань акаунту та інші елементи.

Захист від несанкціонованого доступу: необхідно розробити механізми та заходи безпеки, щоб запобігти несанкціонованому доступу до акаунтів користувачів. Це може включати захист паролів, обмеження доступу до конфіденційних даних та використання захисних механізмів.

Управління паролями та правами доступу: необхідно розробити систему управління ролями та правами доступу, яка дозволить адміністраторам порталу керувати доступом користувачів до різних функцій та ресурсів. Наприклад, можуть бути встановлені різні рівні доступу для студентів, викладачів та адміністраторів.

Тестування та перевірка безпеки: після розробки автентифікації та налаштування акаунтів необхідно провести тестування та перевірку безпеки системи. Це допоможе виявити можливі проблеми, помилки або уразливості, які можуть бути використані для несанкціонованого доступу.

Розробка автентифікації та налаштування акаунтів на порталі дозволить користувачам безпечно та зручно використовувати функціональні можливості порталу, адміністраторам - керувати доступом користувачів та забезпечувати безпеку системи.

Демонстрація функції аутентифікації та авторизації користувачів

```python
def login_view(request):
    if request.method == 'POST':
        form = LoginForm(request.POST)
        if form.is_valid():
            email = form.cleaned_data['email']
            password = form.cleaned_data['password']
            user = authenticate(request, email=email, password=password)
            if user is not None:
                login(request, user)
                if user.is_teacher:
                    return redirect('teachers_office')  # Редиректна сторінку викладача
                else:
                    return redirect('students_office')  # Редиректна сторінку студента
            else:
                form.add_error(None, 'Невірна електронна пошта або пароль')
        else:
            # Додати помилки до повідомлень
            for field, errors in form.errors.items():
                for error in errors:
                    messages.error(request, f"{field}: {error}")
            return render(request, 'registration/login.html', {'form': form})
    else:
        form = LoginForm()
    return render(request, 'registration/login.html', {'form': form})
```

Рис.2.5.0

Цей код (рис.2.5.0) визначає функцію `***login_view***`, яка відповідає за обробку POST-запиту на сторінку входу користувача. Код має наступний функціонал:

1. Перевіряє, чи метод запиту є POST (використовується для надсилання даних форми).

2. Створює екземпляр форми `***LoginForm***` з даними, отриманими з POST-запиту.

3. Перевіряє, чи форма є валідною (чи пройшла всі необхідні перевірки валідації).

4. Якщо форма є валідною, отримує значення електронної пошти (`***email***`) та пароля (`***password***`) з валідних даних форми.

5. Використовує функцію `***authenticate***` для перевірки введених даних та аутентифікації користувача.

6. Якщо автентифікація пройшла успішно, викликає функцію `***login***` для авторизації користувача у системі.

7. Перевіряє, чи користувач є викладачем (`***user.is_teacher***`).

- Якщо так, перенаправляє користувача на сторінку викладача (***`teachers_office`***).

- Якщо ні, перенаправляє користувача на сторінку студента (`***students_office***`).

8. Якщо автентифікація не вдалась, додає помилку до форми з повідомленням "***Невірна електронна пошта або пароль***".

9. Якщо форма не є валідною, додає помилки валідації до повідомлень.

10. Відображає сторінку входу користувача (`***login.html***`) з відповідною формою (`***form***`).

Отже, цей код виконує процес аутентифікації користувача на сторінці входу та перенаправляє користувача на відповідну сторінку в залежності від його ролі (викладач або студент).

```python
def register_view(request):
    if request.method == 'POST':
        form = UserRegistrationForm(request.POST)
        if form.is_valid():
            form.save()
            return redirect('login')
        else:
            # Додати помилки до повідомлень
            for field, errors in form.errors.items():
                for error in errors:
                    messages.error(request, f"{field}: {error}")
            return render(request, 'registration/register.html', {'form': form})
    else:
        form = UserRegistrationForm()
    return render(request, 'registration/register.html', {'form': form})
```

Рис.2.5.1

Цей код (рис. 2.5.1) визначає функцію `***register_view***`, яка відповідає за обробку POST-запиту на сторінці реєстрації нового користувача. Код має наступний функціонал:

1. Перевіряє, чи метод запиту є POST (використовується для надсилання даних форми).

2. Створює екземпляр форми `***UserRegistrationForm***` з даними, отриманими з POST-запиту.

3. Перевіряє, чи форма є валідною (чи пройшла всі необхідні перевірки валідації).

4. Якщо форма є валідною, зберігає дані форми та створює нового користувача.

5. Перенаправляє користувача на сторінку входу (`***login***`).

6. Якщо форма не є валідною, додає помилки валідації до повідомлень.

7. Відображає сторінку реєстрації (`***register.html***`) з відповідною формою (`***form***`).

Отже, цей код дозволяє користувачам реєструватися у системі шляхом заповнення форми реєстрації. Після успішної реєстрації користувача, він буде переправлений на сторінку входу (`***login***`). Якщо форма реєстрації містить помилки, вони будуть відображені на сторінці реєстрації.

Функція виходу з порталу

```
def logout_view(request):
    logout(request)
    return redirect('login')
```

Рис.2.4.4

Цей код (рис. 2.4.4) визначає функцію '***logout_view***, яка відповідає за вихід користувача з системи. У реалізації функції виконується наступне:

1. **logout(request)**: цей виклик виконує вихід користувача шляхом очищення всіх збережених даних сесії.
2. **return redirect('login')**: після виконання виходу з системи, користувач буде перенаправлений на сторінку входу **login**.

Загалом, цей код дозволяє користувачу вийти зі свого облікового запису та перенаправляє його на сторінку входу, де він зможе знову увійти в систему.

3. ТЕСТУВАННЯ ТА ОЦІНКА ЕФЕКТИВНОСТІ ПОРТАЛУ ДЛЯ АВТОМАТИЧНОЇ ПЕРЕВІРКИ ЛАБОРАТОРНИХ РОБІТ

3.1 Перевірка функції авторизації.

1. Перевіримо правильність роботи функції авторизації

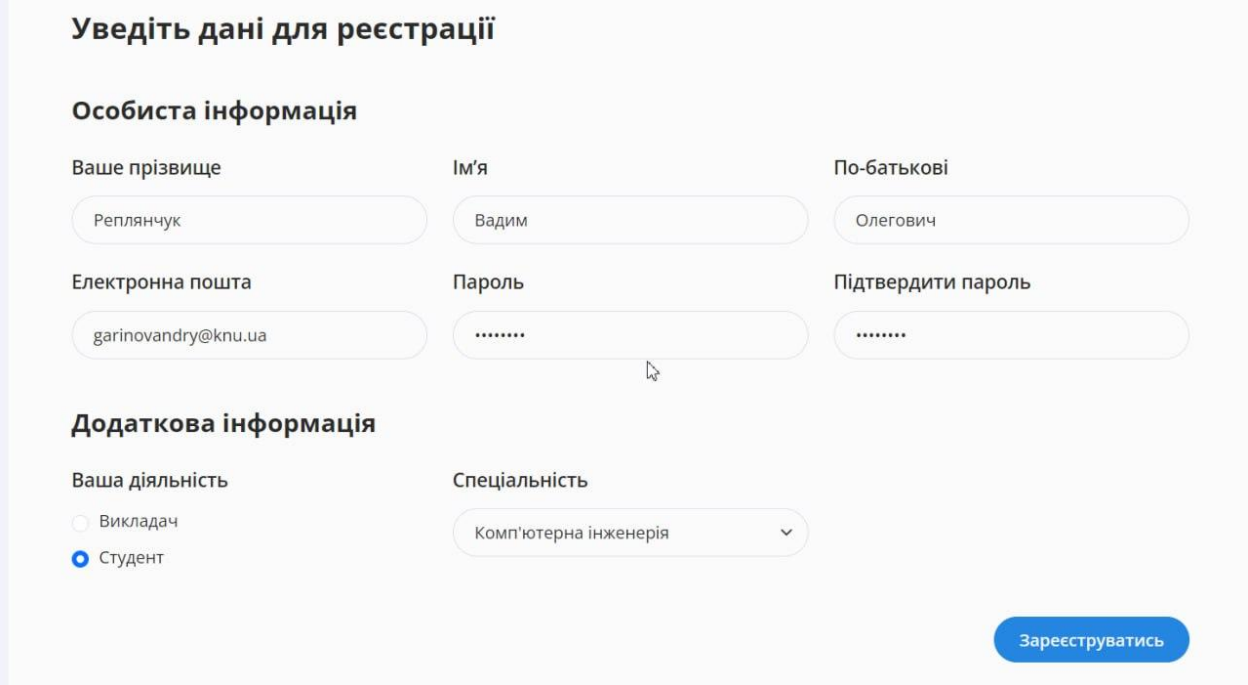


Рис.3.1.0

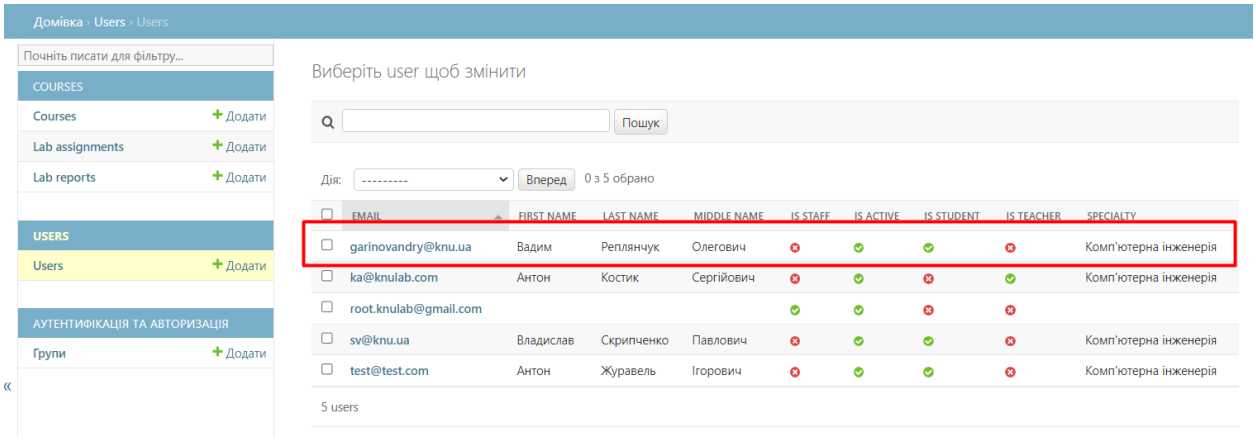


Рис.3.1.1

На рис. 3.1.0 та рис.3.1.1 показані дані про користувачів, що авторизувалися зі сторінки інтерфейсу та на адмін. панелі відповідно.

Також є попередження при вводі надто простого паролю (рис.3.1.3) або паролю який не підходить під критерії (рис.3.1.2)

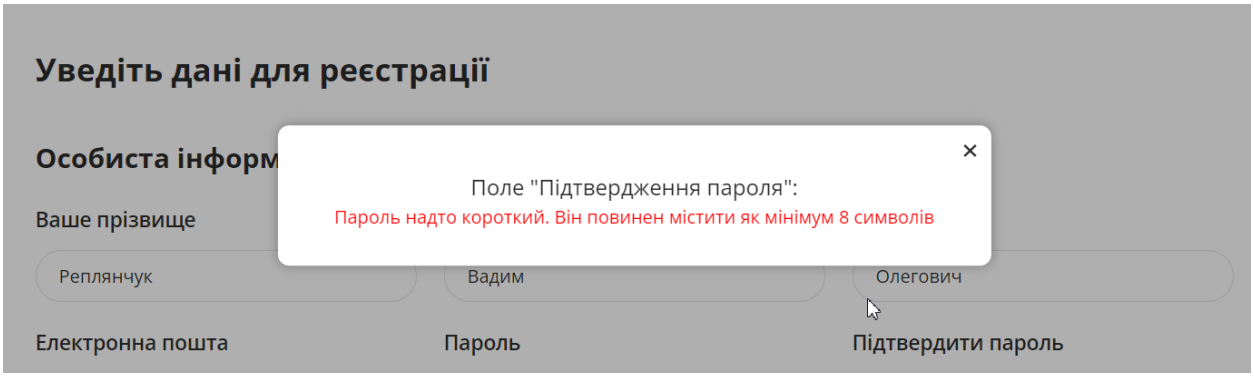


Рис.3.1.2

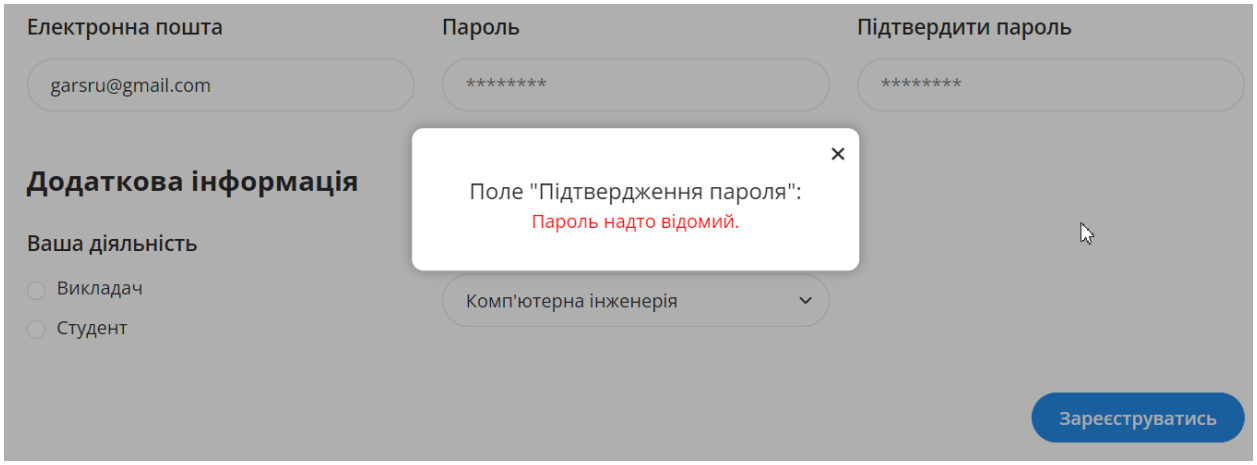


Рис. 3.1.3

При введенні невірних даних при авторизації виводиться помилка (рис.3.1.4)

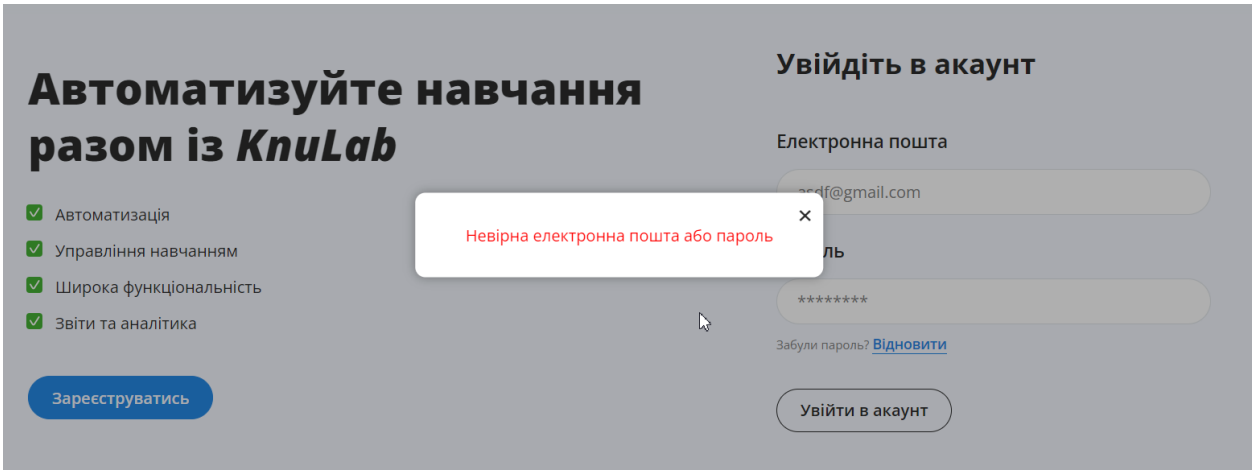


Рис.3.1.4

3.2 Тестування автоматичної перевірки лаб. роботи.

Завантаження коду на перевірку наприклад, для Лабораторної роботи №2 (рис.3.2.0)

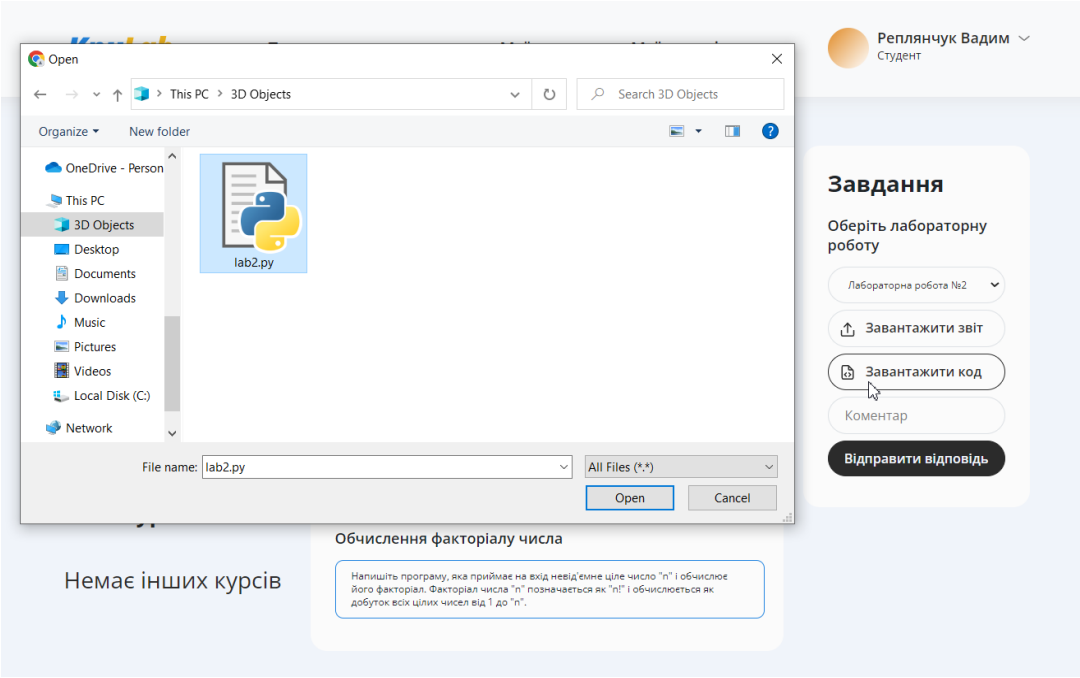


Рис.3.2.0

Тиснемо на кнопку “Відправити відповідь” (рис.3.2.1)

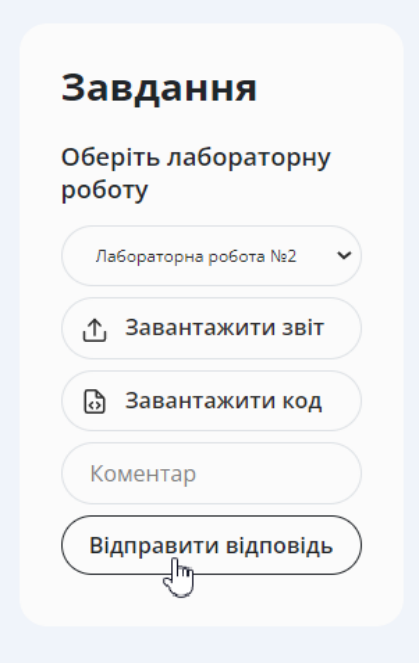


Рис.3.2.1

Звіт, який виводиться при помилках в коді (рис.3.2.2)

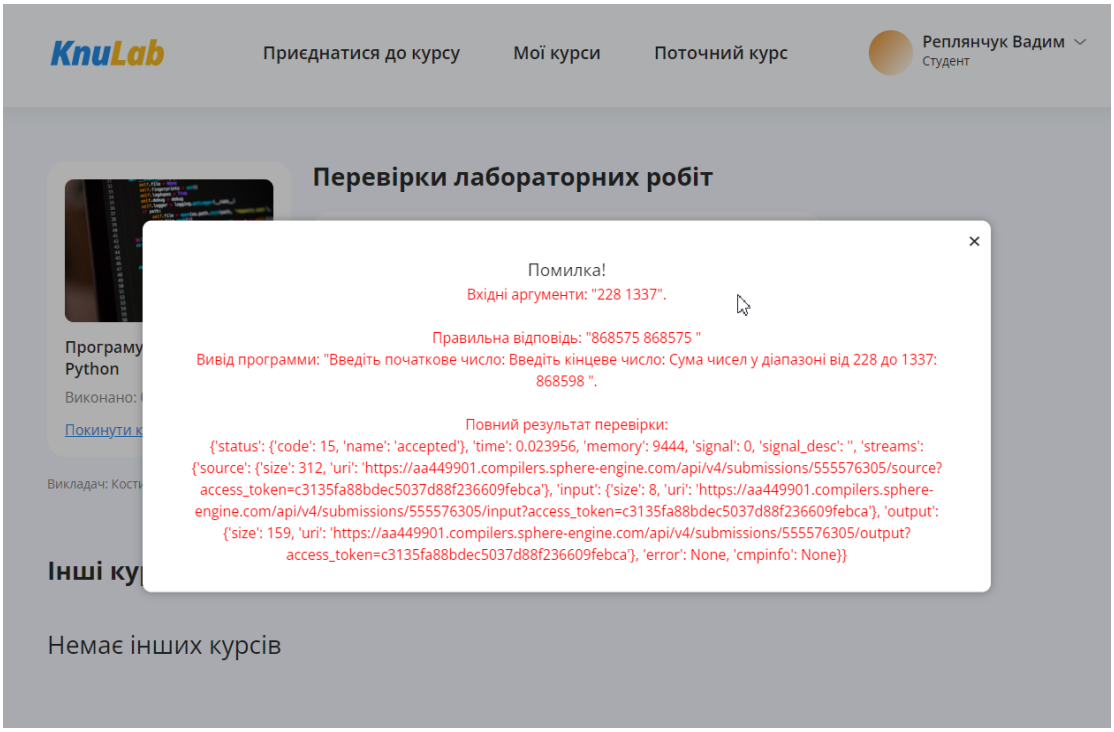


Рис.3.2.2

Звіт, який виводиться при правильній відповіді (рис.3.2.3)

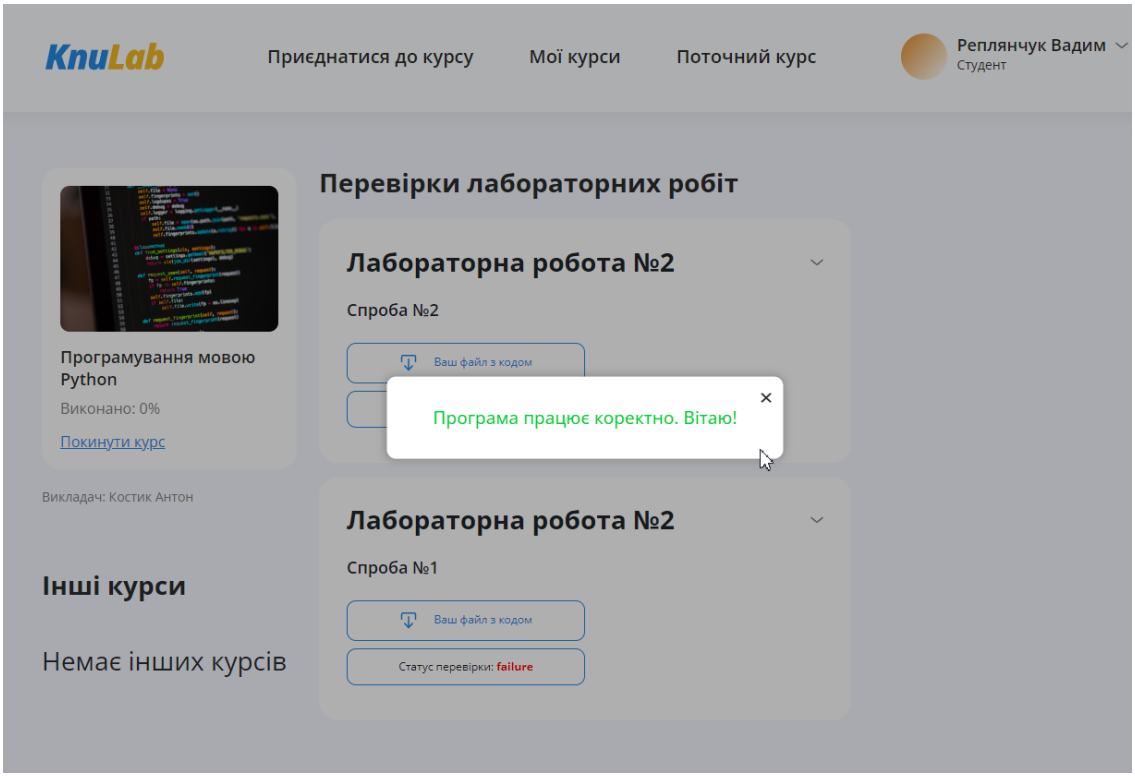


Рис.3.2.3

Сторінка перевірки лабораторних робіт після завантаження Студентом своїх відповідей (рис.3.2.5)

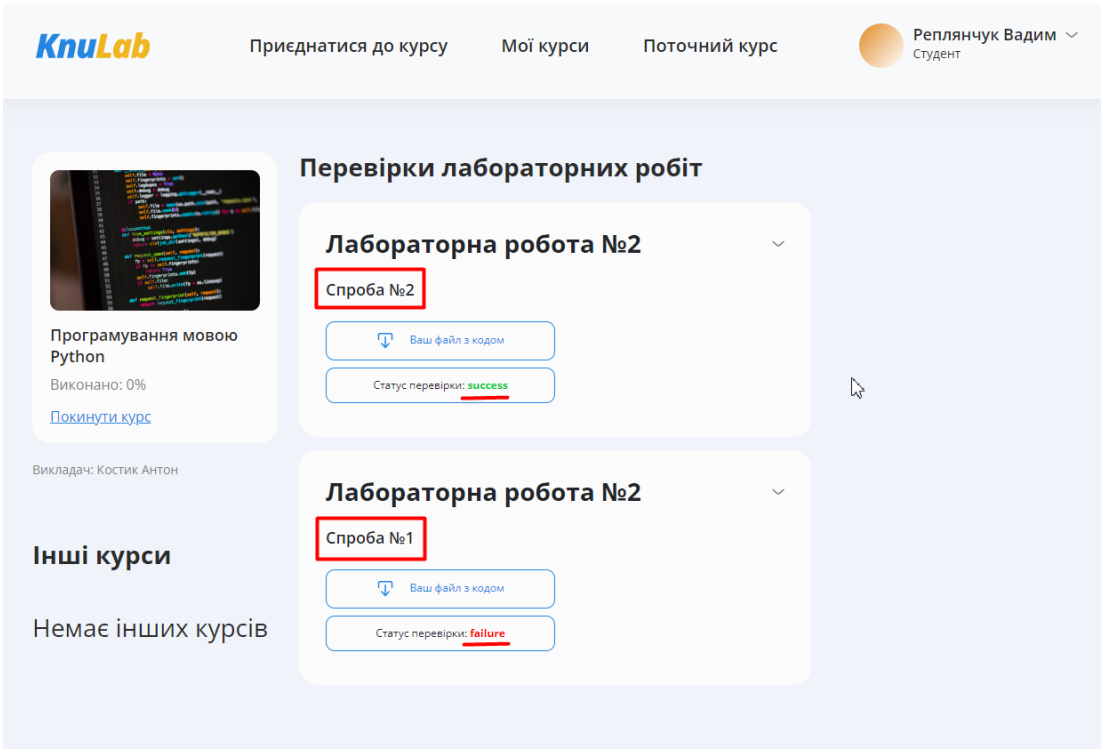


Рис.3.2.5

Відповідні записи в базі даних з приводу перевірки лабораторних робіт Студента (рис.3.2.6)

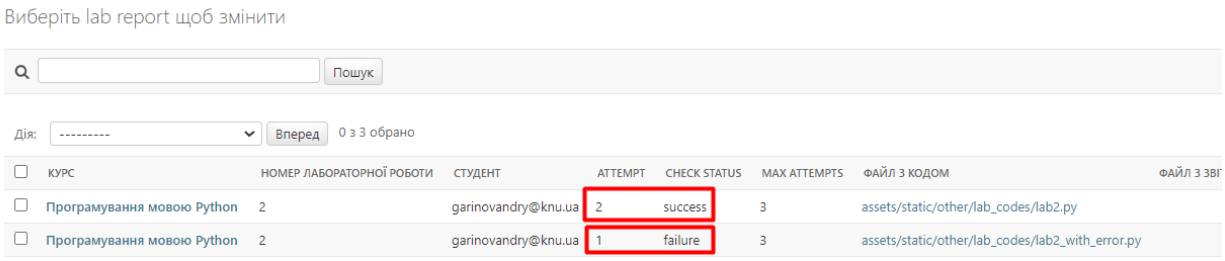


Рис.3.2.6

3.3 Рекомендації з покращення ефективності та якості роботи порталу

1. Оптимізація швидкодії: аналізувати та вдосконалювати швидкодію порталу шляхом використання кешування, оптимізації запитів до бази даних, розподілення навантаження та використання ефективних алгоритмів обробки даних.
2. Забезпечення масштабованості: розробити архітектуру та інфраструктуру, що дозволяють масштабувати портал, зокрема, шляхом використання горизонтального масштабування, розділення функціональності на мікросервіси та використання контейнеризації.
3. Забезпечення безпеки: виконати аудит безпеки порталу, перевірити наявність потенційних вразливостей та вжити заходів для їх усунення.

Застосувати криптографічні протоколи, захистити дані користувачів та запобігти несанкціонованому доступу до системи.

4. Вдосконалення інтерфейсу та користувацького досвіду: провести оцінку користувацького досвіду та внести вдосконалення у дизайн інтерфейсу порталу, забезпечивши зручну навігацію, зрозумілу структуру і чітку взаємодію з користувачами.
5. Використання аналітики та звітності: розробити механізми для збору та аналізу даних про використання порталу, що дозволяє здійснювати моніторинг продуктивності, виявляти проблемні області та приймати відповідні заходи для покращення роботи порталу.
6. Тестування та регулярні оновлення: забезпечити систематичне тестування порталу, включаючи автоматичне та ручне тестування, для виявлення помилок та проблем. Регулярно оновлювати портал, виправляючи помилки, вдосконалюючи функціональність та додавати нові можливості.
7. Постійна підтримка та комунікація: забезпечити постійну підтримку користувачів, вирішувати їх запити та проблеми, а також підтримувати відкритий канал комунікації для отримання фідбеку від користувачів та вдосконалення порталу відповідно до їх потреб.
8. Регулярні резервні копії та відновлення: розробити стратегію резервного копіювання та відновлення даних порталу для запобігання втрати даних та можливості швидкого відновлення у разі непередбачених ситуацій.

ВИСНОВКИ

У результаті дослідження та розробки дипломної роботи було проведено аналіз існуючих порталів для автоматичної перевірки лабораторних робіт. На основі цього аналізу було розроблено власний портал, який надає можливість авторизуватися студентам та викладачам, створювати лабораторні роботи з боку викладачів та завантажувати відповіді з боку студентів з подальшою автоматичною перевіркою коду та формуванням звіту з результатами.

Переваги розробленого порталу:

1. Автоматична перевірка коду: портал забезпечує автоматичну перевірку коду, що дозволяє ефективно та об'єктивно оцінювати роботи студентів. Це зменшує навантаження на викладачів та дозволяє швидко надати студентам зворотний зв'язок.

2. Автоматичне формування звіту: портал автоматично формує звіти з результатами перевірки, що дозволяє зручно та швидко ознайомитися з оцінками та коментарями. Це полегшує взаємодію між викладачами та студентами та сприяє швидкому виявленню та виправленню помилок.

3. Зручний інтерфейс користувача: портал має зручний та інтуїтивно зрозумілий інтерфейс, що дозволяє легко навігувати та використовувати його функціонал. Це забезпечує зручну та приємну взаємодію для як студентів, так і викладачів.

4. Підвищення ефективності та якості оцінювання: завдяки автоматичній перевірці та швидкому формуванню звітів, портал дозволяє підвищити ефективність та об'єктивність оцінювання лабораторних робіт. Це допомагає студентам отримувати швидкий зворотний зв'язок та вчитися на своїх помилках, а викладачам - ефективно керувати процесом навчання.

Отже, розроблений портал для автоматичної перевірки лабораторних робіт є ефективним інструментом для покращення процесу навчання та оцінювання. Його переваги включають автоматичну перевірку коду, автоматичне формування звітів, зручний інтерфейс користувача, підвищення ефективності та якості оцінювання, а також можливість зберігання та аналізу даних. Розробка такого порталу є актуальною та важливою у сучасному освітньому середовищі, сприяючи покращенню якості навчання та стимулюючи професійний розвиток студентів.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

[1] Django Documentation. [Електронний ресурс]. Режим доступу до ресурсу: https://docs.djangoproject.com/en/4.2/

[2] SonarQube Documentation. [Електронний ресурс]. Режим доступу до ресурсу: https://docs.sonarqube.org/latest/

[3] Moodle Documentation. [Електронний ресурс]. Режим доступу до ресурсу: https://docs.moodle.org/402/en/Main_page

[4] HackerRank. [Електронний ресурс]. Режим доступу до ресурсу: https://www.hackerrank.com/

[5] Turnitin Resources. [Електронний ресурс]. Режим доступу до ресурсу: https://www.turnitin.com/resources

[6] PostgreSQL Documentation. [Електронний ресурс]. Режим доступу до ресурсу: https://www.postgresql.org/docs/15/index.html

[7] Sphere Engine Documentation. [Електронний ресурс]. Режим доступу до ресурсу: https://docs.sphere-engine.com/

ДОДАТКИ

Додаток 1

```
import requests
import json

# Функція перевірки лабораторної роботи
def run_lab_check(source, compilerId, input, answers, existing_lab_report):

    compileAccessToken = "--------------------"
    endpoint = "aa449901"
    compileApiUrl = f"https://{endpoint}.compilers.sphere-
engine.com/api/v4/submissions?access_token={compileAccessToken}"

    compileData = {
        "compilerId": compilerId,
        "source": source,
        "input": input.strip()
    }

    # Створення підключення до Sphere-Engine
    response = requests.post(compileApiUrl, data=compileData)

    # Перевірка коду підключення
    if response.status_code == 200 or response.status_code == 201:
        # Отриманна "id" підключення
        response_data = json.loads(response.text)
        id = response_data["id"]
        compileResultUrl = f"https://{endpoint}.compilers.sphere-
engine.com/api/v4/submissions/{id}?access_token={compileAccessToken}"

        # Очікування виконання перевірки
        while True:
            response = requests.get(compileResultUrl)
            if response.status_code == 200 or response.status_code == 201:
                response_data = json.loads(response.text)
                result_status = response_data["result"]["status"]["name"]

                if result_status != "compiling..." and result_status !=
"running..." and result_status != "waiting...":
                    break
            else:
                existing_lab_report.check_status = "failure"
                existing_lab_report.save()
                return {"success": False, "message": f"Помилка
{response.status_code}: {response.text}"}
```

```
        print("-----------\nresponse_data: ",response_data,"\n-----------")

        # Перевірка, чи запустився код лабораторної роботи
        if response_data["result"]["signal"] == 0:
            result = response_data["result"]

            # Отримання stdout параметрів на вхід
            if response_data["result"]["streams"]["input"] is not None:
                output_url = response_data["result"]["streams"]["input"]["uri"]
            else:
                existing_lab_report.check_status = "failure"
                existing_lab_report.save()
                return {"success": False, "message": f"Помилка при отриманні вмісту
введених параметрів. <br><br>Повний результат перевірки: <br>{result}"}

            response = requests.get(output_url)
            if response.status_code == 200:
                input_content = response.content.decode('utf-8')
            else:
                existing_lab_report.check_status = "failure"
                existing_lab_report.save()
                return {"success": False, "message": f"Помилка при отриманні вмісту
введених параметрів. <br><br>Код: {response.status_code}"}

            # Отримання stdout коду
            if response_data["result"]["streams"]["output"] is not None:
                output_url = response_data["result"]["streams"]["output"]["uri"]
            else:
                existing_lab_report.check_status = "failure"
                existing_lab_report.save()
                return {"success": False, "message": f"Помилка при отриманні вмісту
виводу коду. <br><br>Повний результат перевірки: <br>{result}"}

            response = requests.get(output_url)
            if response.status_code == 200:
                file_content = response.content.decode('utf-8')
            else:
                existing_lab_report.check_status = "failure"
                existing_lab_report.save()
                return {"success": False, "message": f"Помилка при отриманні вмісту
виводу коду. <br><br>Код: {response.status_code}"}

            # Формування результату перевірки лабораторної роботи
            if result_status=="accepted":

                # Визначення лічільнику співпадання виводу програми Студента з
правильною відповіддю
                successCount = 0
```

```
                answerElements = answers.split()
                for answer in answerElements:
                    if answer.strip() in file_content.strip():
                        successCount += 1
                # Обробка результату перевірки, якщо програма запустилася та
виконала свою роботу
                if successCount == len(answerElements):
                    existing_lab_report.check_status = "success"
                    existing_lab_report.save()
                    return {"success": True, "message": f"Програма працює коректно.
Вітаю!"}
                else:
                    existing_lab_report.check_status = "failure"
                    existing_lab_report.save()
                    return {"success": False, "message": f"Вхідні аргументи:
\"{input_content}\".<br><br>Правильна відповідь: \"{answers}\"<br>Вивід программи:
\"{file_content}\".<br><br>Повний результат перевірки: <br>{result}"}
            else:
                existing_lab_report.check_status = "failure"
                existing_lab_report.save()
                return {"success": False, "message": f"Вхідні аргументи:
\"{input_content}\".<br><br>Вивід программи: \"{file_content}\".<br><br>Повний
результат перевірки: <br>{result}"}

        # Обробка результатів, якщо код лабораторної роботи не запустився
        else:

            status_name = response_data["result"]["status"]["name"]
            # Перевірка, чи є інформація про причину, чому саме код не запустився
            if response_data["result"]["streams"]["cmpinfo"] is not None:
                # Отримання stdout коду
                output_url = response_data["result"]["streams"]["cmpinfo"]["uri"]
                response = requests.get(output_url)
                if response.status_code == 200:
                    cmpinfo_content = response.content.decode('utf-8')
                else:
                    existing_lab_report.check_status = "failure"
                    existing_lab_report.save()
                    return {"success": False, "message": f"Помилка при отриманні
вмісту виводу коду. <br><br>Код: {response.status_code}"}
                existing_lab_report.check_status = "failure"
                existing_lab_report.save()
                return {"success": False, "message": f"Неможливо запустити код -
\"{status_name}\"<br><br>Повідомлення про помилку: <br><pre style=\"text-align:
left\">{cmpinfo_content}</pre>Повний результат перевірки: <br>{response_data}"}

            # Перевірка, чи є інформація про помилку
            elif response_data["result"]["streams"]["error"] is not None:
```

```
                # Отримання stdout помилки
                output_url = response_data["result"]["streams"]["error"]["uri"]
                response = requests.get(output_url)
                if response.status_code == 200:
                    error_content = response.content.decode('utf-8')
                else:
                    existing_lab_report.check_status = "failure"
                    existing_lab_report.save()
                    return {"success": False, "message": f"Помилка при отриманні
вмісту помилки. <br><br>Код: {response.status_code}"}
                existing_lab_report.check_status = "failure"
                existing_lab_report.save()
                return {"success": False, "message": f"Неможливо запустити код -
\"{status_name}\"<br><br>Повідомлення про помилку: <br><pre style=\"text-align:
left\">{error_content}</pre>Повний результат перевірки: <br>{response_data}"}

            else:
                signal_code = response_data["result"]["signal"]
                signal_desc = response_data["result"]["signal_desc"]
                existing_lab_report.check_status = "failure"
                existing_lab_report.save()
                return {"success": False, "message": f"Неможливо запустити код -
\"{status_name}\"<br><br><div style=\"text-align: left; margin: 0;\">Код сигналу:
\"{signal_code}\".<br>Повідомлення: \"{signal_desc}\"</div><br>Повний результат
перевірки: <br>{response_data}"}
    else:
        existing_lab_report.check_status = "failure"
        existing_lab_report.save()
        return {"success": False, "message": f"Помилка {response.status_code}:
{response.text}. Зверніться до адміністратора"}
```