

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ імені ІГОРЯ СІКОРСЬКОГО»
Факультет інформатики та обчислювальної техніки
(повна назва інституту/факультету)

Кафедра інформатики та програмної інженерії
(повна назва кафедри)

«До захисту допущено»

Завідувач кафедри

_____ Едуард ЖАРІКОВ
(підпис) (ім'я прізвище)

“ _____ ” _____ 2023 р.

Дипломний проєкт
на здобуття ступеня бакалавра
за освітньо-професійною програмою «Інженерія програмного забезпечення
інформаційних систем»
спеціальності «121 Інженерія програмного забезпечення»

на тему: Програмне забезпечення для перевірки якості написання
SQL-запитів

Виконав студент IV курсу, групи ІП-92
(шифр групи)

Фрадинська Дарія Віталіївна
(прізвище, ім'я, по батькові)

_____ (підпис)

Керівник доцент, к.т.н., доц., Олійник Ю.О
(посада, науковий ступінь, вчене звання, прізвище та ініціали)

_____ (підпис)

Консультант
з графічної
документації доцент, к.т.н., доц., Лішук К. І.
(посада, науковий ступінь, вчене звання, прізвище та ініціали)

_____ (підпис)

Рецензент Доцент кафедри ІСТ, к.т.н., доц. Рибачук Л.В.
(посада, науковий ступінь, вчене звання, прізвище та ініціали)

_____ (підпис)

Засвідчую, що у цій дипломній роботі
немає запозичень з праць інших
авторів без відповідних посилань.

Студент _____
(підпис)

Київ –2023

Національний технічний університет України

«Київський політехнічний інститут імені Ігоря Сікорського»

Факультет інформатики та обчислювальної техніки

Кафедра інформатики та програмної інженерії

Рівень вищої освіти – перший (бакалаврський)

Спеціальність – 121 Інженерія програмного забезпечення

Освітньо-професійна програма – Інженерія програмного забезпечення інформаційних систем

ЗАТВЕРДЖУЮ

Завідувач кафедри

_____ Едуард ЖАРІКОВ
(підпис) (ім'я прізвище)

“ ” _____ 2023 р.

ЗАВДАННЯ

на дипломний проєкт студенту

Фрадинській Дарії Віталіївній

(прізвище, ім'я, по батькові)

1. Тема проєкту Програмне забезпечення для перевірки якості написання SQL-запитів

керівник проєкту Олійник Юрій Олександрович, к.т.н., доцент
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджені наказом по університету від «31» травня 2023 р. №2101-с

2. Термін подання студентом проєкту « 17 » червня 2023 року

3. Вихідні дані до проєкту: технічне завдання

4. Зміст пояснювальної записки

1) Загальні положення: основні визначення та терміни, аналіз вимог до програмного забезпечення, огляд програмних та технологічних аналогів, визначення функціональних та нефункціональних вимог, постановка задачі.

2) Моделювання та конструювання програмного забезпечення: розробка контекстної діаграми, діаграми діяльності, макету інтерфейсу користувача, вибір мови та програмного середовища, вибір архітектури для програмного забезпечення, реалізація проєкту системи, аналіз безпеки даних

3) Аналіз якості та тестування програмного забезпечення: опис процесів тестування, опис контрольного приладу

4) Впровадження програмного забезпечення: розгортання програмного забезпечення, супровід програмного забезпечення

5. Перелік графічного матеріалу

1) Схема структурна варіантів використання

2) Схема структурна станів системи

3) Схема структурна класів програмного забезпечення

- 4) Схема структурна діяльності
- 5) Схема структурна розгортання
- 6) Креслення вигляду екранних форм

6. Консультанти розділів проєкту

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв

7. Дата видачі завдання «10» березня 2023 року

Календарний план

№ з/п	Назва етапів виконання дипломного проєкту	Термін виконання етапів проєкту	Примітка
1	Вивчення рекомендованої літератури	10.03.2023	
2	Аналіз існуючих методів розв'язання задачі	03.03.2023	
3	Постановка та формалізація задачі	10.04.2023	
4	Розробка інформаційного забезпечення	15.04.2023	
5	Алгоритмізація задачі	25.04.2023	
6	Обґрунтування вибору використаних технічних засобів	30.04.2023	
7	Розробка програмного забезпечення	05.05.2023	
8	Налагодження програми	10.05.2023	
9	Виконання графічних документів	20.05.2023	
10	Оформлення пояснювальної записки	29.05.2023	
11	Подання ДП на попередній захист	02.06.2023	
12	Подання ДП рецензенту	12.06.2023	
13	Подання ДП на основний захист	17.06.2023	

Студент

(підпис)

Дарія ФРАДИНСЬКА

(ініціали, прізвище)

Керівник

(підпис)

Юрій ОЛІЙНИК

(ініціали, прізвище)

АНОТАЦІЯ

Пояснювальна записка дипломного проєкту складається з чотирьох розділів, містить 18 таблиць, 14 рисунків та 18 джерел – загалом 68 сторінок.

Дипломний проєкт присвячений розробці програмного забезпечення для перевірки якості написання SQL-запитів

Мета полегшення процесу розробки та підтримки SQL запитів та зменшення кількості помилок при реалізації запитів

Об'єкт дослідження: програмне забезпечення для перевірки якості написання SQL-запитів

Предмет дослідження: перевірка якості написання SQL-запитів

В першому розділі роботи проведено аналіз предметної області, а саме вивчено функціонал та призначення програмного забезпечення, що має спеціалізовану назву лінтер. Досліджено та проаналізовано робочий функціонал існуючих програмних рішень, на базі чого виділено риси, що відсутні у існуючих програмних рішень. Розроблено функціональні та нефункціональні вимоги, поставлено задачу дипломного проєкту.

Другий розділ роботи присвячено моделюванню та конструюванню програмного забезпечення. Визначено мову програмування, що буде використовуватись для розробки та шаблони за якими буде створюватись архітектура програмного забезпечення. Розглянуто середовища та фреймворки, що будуть використовуватись у розробці. Проведено аналіз безпеки програмного забезпечення.

В третьому розділі проведено аналіз якості програмного забезпечення та тестування створеного програмного забезпечення, розглядається контрольний приклад. Для аналізу якості програмного забезпечення, що створено, введено метрику Ассурасу, Проаналізовано різні способи тестування та вибрано найбільш оптимальний для проєкту. Описано тестові сценарії, що покривають основний функціонал програмного забезпечення. Проведено аналіз коду за допомогою модульних тестів. Здійснено опис контрольного прикладу з ілюстраціями web-інтерфейсу.

В четвертому розділі розглянуто питання розгортання програмного забезпечення та його оновлення. Запропоновано використання відповідних практик для оновлення забезпечення.

Також наведено: опис програми, технічне завдання, програму та методику тестування, керівництво користувача та графічний матеріал.

КЛЮЧОВІ СЛОВА: SQL, DOCKER, БАЗИ ДАНИХ, СИНТАКСИЧНИЙ АНАЛІЗ, МІКРОСЕРВІСНА АРХІТЕКТУРА

ABSTRACT

The explanatory note of the diploma project consists of four sections, contains 12 tables, 14 figures and 18 sources – in total 68 pages.

The purpose of the diploma project is facilitating the process of developing and supporting SQL queries and reducing the number of errors in the implementation of queries

Research subject: software for checking the quality of writing SQL queries

The purpose of the study: checking the quality of writing SQL queries

In the first section of the work, an analysis of the subject area was carried out, namely, the functional and purpose of the software, which has a specialized name linter, was studied. The basic working functionality of the existing software solutions has been studied and analyzed, and the features that are absent in the existing software solutions have been identified. Functional and non-functional requirements were developed, and the task of the diploma project was set.

The second part of the work is devoted to software modeling and design. The programming language that will be used for development and the templates that will be used to create the software architecture are defined. The environment and frameworks that will be used in the development are considered. Software security analysis was conducted.

In the third section, an analysis of software quality and testing of the created software is carried out, a control example is considered. To analyze the quality of the created software, the Accuracy metric was introduced, various testing methods were analyzed and the most optimal one was selected for the project. Described test scenarios covering basic software functionality. The code was analyzed using unit tests. The description of the control example with the results of the web interface was carried out.

The fourth chapter deals with software deployment and updates. It is suggested to use appropriate practices for updating the provisioning.

KEYWORDS: SQL, DOCKER, DATABASES, SYNTAX ANALYSIS, MICROSERVICE ARCHITECTURE

№ з/п	Формат	Позначення	Найменування	Кількість листів	Примітка
1	A4		Завдання на дипломний проєкт	2	
2	A4	КПІ.ІП- 9228.045420.01.91	Технічне завдання	13	
3	A4	КПІ.ІП- 9228.045420.02.81	Пояснювальна записка	68	
4	A4	КПІ.ІП- 9228.045420.03.12	Текст програми	21	
5	A4	КПІ.ІП- 9228.045420.04.51	Програма та методика тестування	7	
6	A4	КПІ.ІП- 9228.045420.05.34	Керівництво користувача	7	
7	A3	КПІ.ІП- 9228.045420.06.99 ССВ	Схема структурна варіантів	1	
			використань		
8	A3	КПІ.ІП- 9228.045420.07.99 ССС	Схема структурна станів системи	1	
9	A3	КПІ.ІП- 9228.045420.08.99 ССД	Схема діяльності	1	
10	A3	КПІ.ІП- 9228.045420.09.99 ССК	Схема структурна класів	1	
			програмного забезпечення		
11	A3	КПІ.ІП- 9228.045420.10.99 КЕ	Креслення вигляду екранних форм	1	
12	A3	КПІ.ІП- 9228.045420.11.99 ССР	Схема структурна розгортання	1	
13	A3	КПІ.ІП- 9228.045420.12.99 ССМ	Схема структурна компонентів	1	

					КПІ.ІП-9228.045420.00.90								
Змін.	Арк.	№ докум.	Підп.	Дата	Відомість дипломного проєкту					Літ.	Аркуш	Аркушів	
Розроб.	Фрадинська Д. В.											1	
Перевір.	Олійник Ю. О.												
Н.контр.	Ліщук К. І.												
Затв.	Жаріков Е.В.												
					КПІ ім. Ігоря Сікорського ФІОТ каф. ІПІ гр. ІП-92								

					КПІ.ІП-9228.045420.00.90	Арк.
Змін.	Арк.	№ докум.	Підп.	Дата.		8

Факультет інформатики та обчислювальної техніки

Кафедра інформатики та програмної інженерії

“ЗАТВЕРДЖЕНО”

Завідувач кафедри

_____ Едуард ЖАРІКОВ

“ ____ ” _____ 2023 р.

**Програмне забезпечення для перевірки якості написання SQL-
запитів**

Технічне завдання

КП.ІП-9228.045420.01.91

“ПОГОДЖЕНО”

Керівник проєкту:

_____ Юрій ОЛІЙНИК

Нормоконтроль:

_____ Катерина ЛІЩУК

Виконавець:

_____ Дарія ФРАДИНСЬКА

Київ – 2023

ЗМІСТ

1	НАЙМЕНУВАННЯ ТА ГАЛУЗЬ ЗАСТОСУВАННЯ.....	4
2	ПІДСТАВА ДЛЯ РОЗРОБКИ	5
3	ПРИЗНАЧЕННЯ РОЗРОБКИ.....	6
4	ВИМОГИ ДО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ.....	7
4.1	Вимоги до функціональних характеристик.....	7
4.1.1	Користувацького інтерфейсу.....	7
4.1.2	Для користувача:.....	7
4.1.3	Додаткові вимоги:.....	8
4.2	Вимоги до надійності	8
4.3	Умови експлуатації.....	8
4.3.1	Вид обслуговування	8
4.3.2	Обслуговуючий персонал	8
4.4	Вимоги до складу і параметрів технічних засобів.....	8
4.5	Вимоги до інформаційної та програмної сумісності	9
4.5.1	Вимоги до вхідних даних.....	9
4.5.2	Вимоги до вихідних даних.....	9
4.5.3	Вимоги до мови розробки	9
4.5.4	Вимоги до середовища розробки	9
4.5.5	Вимоги до представленню вихідних кодів	9
4.6	Вимоги до маркування та пакування	9
4.7	Вимоги до транспортування та зберігання.....	9
4.8	Спеціальні вимоги.....	9
5	ВИМОГИ ДО ПРОГРАМНОЇ ДОКУМЕНТАЦІЇ	10

5.1	Попередній склад програмної документації	10
5.2	Спеціальні вимоги до програмної документації.....	10
6	СТАДІЇ І ЕТАПИ РОЗРОБКИ.....	11
7	ПОРЯДОК КОНТРОЛЮ ТА ПРИЙМАННЯ.....	12

					КПІ.ІП-9228.045420.01.91	Арк.
Змін.	Арк.	№ докум.	Підп.	Дата.		3

1 НАЙМЕНУВАННЯ ТА ГАЛУЗЬ ЗАСТОСУВАННЯ

Назва розробки: Програмне забезпечення для перевірки якості написання SQL-запитів.

Галузь застосування:

Наведене технічне завдання поширюється на розробку програмного забезпечення sqlinter КП.ІП-9228.045420, котра використовується для перевірки якості написання SQL-запитів та призначена для спрощення розробки та підвищення якості написання SQL-запитів для роботи з базами даних.

					КП.ІП-9228.045420.01.91	Арк.
Змін.	Арк.	№ докум.	Підп.	Дата.		4

2 ПІДСТАВА ДЛЯ РОЗРОБКИ

Підставою для розробки програмного забезпечення для перевірки якості написання SQL-запитів є завдання на дипломне проєктування, затверджене кафедрою інформатики та програмної інженерії Національного технічного університету України «Київський політехнічний інститут імені Ігоря Сікорського».

					КПІ.ІП-9228.045420.01.91	Арк.
Змін.	Арк.	№ докум.	Підп.	Дата.		5

3 ПРИЗНАЧЕННЯ РОЗРОБКИ

Розробка призначена для спрощення розробки та підвищення якості написання SQL-запитів для роботи з базами даних.

Метою розробки є полегшення процесу розробки та підтримки SQL запитів та зменшення кількості помилок при їх експлуатації

					КП.ІП-9228.045420.01.91	Арк.
Змін.	Арк.	№ докум.	Підп.	Дата.		6

4 ВИМОГИ ДО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

4.1 Вимоги до функціональних характеристик

Програмне забезпечення повинно забезпечувати виконання наступних основних функцій:

4.1.1 Користувацького інтерфейсу

- наявність поля для введення запиту для аналізу
- наявність поля для отримання результатів аналізу
- наявність кнопки для очищення полів
- наявність кнопки для відправки відгуку на сервер
- наявність кнопки для формування звіту з результатами аналізу

Прототип екранної форми, що складається з двох основних секцій. Верхня секція містить велике текстове поле з підписом "Поле введення запиту для аналізу". Нижня секція містить три кнопки: "Форматувати запит" (з синім квадратом), "Відправити" та "Очистити". Під кнопками знаходиться ще одне велике текстове поле з підписом "Поле отримання синтаксичних та стилістичних підказок".

Рисунок 4.1 - Прототип екранної форми

4.1.2 Для користувача:

- проведення синтаксичного аналізу запиту;
- проведення стилістичного аналізу запиту;
- проведення оптимізаційного аналізу запиту;

- формування результатів аналізу запиту;
- форматування запиту;
- використання програмного забезпечення через веб-клієнт;

4.1.3 Додаткові вимоги:

- робота з програмним забезпеченням через PyCharm;
- зрозумілий та простий у використанні інтерфейс;
- мова інтерфейсу та пояснень - українська;
- сумісність з різними пристроями та операційними системами;
- мікросервісна архітектура.

Розробку виконати на платформі Windows, Linux чи MacOS

4.2 Вимоги до надійності

Передбачити контроль введення інформації та захист від некоректних дій користувача.

4.3 Умови експлуатації

Умови експлуатації згідно СанПін 2.2.2.542 – 96.

4.3.1 Вид обслуговування

Вимоги щодо обслуговування не надаються

4.3.2 Обслуговуючий персонал

Вимоги до обслуговуючого персоналу не висуваються

4.4 Вимоги до складу і параметрів технічних засобів

Мінімальна конфігурація технічних засобів:

- тип процесору: Pentium;
- об'єм ОЗП: 32 Мб;
- підключення до мережі Інтернет зі швидкістю від 20 мегабіт;

Рекомендована конфігурація технічних засобів :

- тип процесору: Intel Core i7;
- об'єм ОЗП: 16 Гб;
- підключення до мережі Інтернет зі швидкістю від 100 мегабіт;

					КП.ІП-9228.045420.01.91	Арк.
Змін.	Арк.	№ докум.	Підп.	Дата.		8

4.5 Вимоги до інформаційної та програмної сумісності

Програмне забезпечення повинно працювати під управлінням операційних систем сімейства WIN32 (Windows'XP, Windows NT і т.д.) або Unix.

4.5.1 Вимоги до вхідних даних

Вхідні дані повинні бути представлені в наступному форматі: SQL запит поданий текстом

4.5.2 Вимоги до вихідних даних

Результати повинні бути представлені в наступному форматі: результати аналізу подані текстом або файл розширення *.html.

4.5.3 Вимоги до мови розробки

Розробку виконати на мові програмування Python

4.5.4 Вимоги до середовища розробки

Розробку виконати на за допомогою середовища PyCharm

4.5.5 Вимоги до представлення вихідних кодів

Вихідний код програми має бути представлений у вигляді контейнерів Docker.

4.6 Вимоги до маркування та пакування

Вимоги до маркування та пакування не висуваються.

4.7 Вимоги до транспортування та зберігання

Вимоги до транспортування та зберігання не висуваються.

4.8 Спеціальні вимоги

Керівництво користувача з прикладами та описом запитів та позначеннями, що допустимі для використання.

					КПІ.ІП-9228.045420.01.91	Арк.
Змін.	Арк.	№ докум.	Підп.	Дата.		9

5 ВИМОГИ ДО ПРОГРАМНОЇ ДОКУМЕНТАЦІЇ

5.1 Попередній склад програмної документації

У склад супроводжувальної документації повинні входити наступні документи на аркушах формату А4:

- пояснювальна записка;
- технічне завдання;
- текст програми;
- програма та методика тестування;
- керівництво користувача;

Графічна частина повинна бути виконана на аркушах формату А3 та містити наступні документи:

- схема структурна варіантів використання;
- схема структурна діяльності;
- схема структурна станів програмного забезпечення;
- схема структурна компонентів програмного забезпечення;
- схема структурна розгортання програмного забезпечення;
- схема структурна класів програмного забезпечення;
- креслення вигляду екранних форм;

5.2 Спеціальні вимоги до програмної документації

Програмні модулі, котрі розробляються, повинні бути задокументовані, тобто тексти програм повинні містити всі необхідні коментарі.

6 СТАДІЇ І ЕТАПИ РОЗРОБКИ

№	Назва етапу	Строк	Звітність
1.	Вивчення літератури за тематикою проекту	21.02	
2.	Розробка технічного завдання	03.03	Технічне завдання
3.	Аналіз вимог та уточнення специфікацій	19.04	Специфікації програмного забезпечення
4.	Проектування структури програмного забезпечення, проектування компонентів	30.04	Схема структурна програмного забезпечення та специфікація компонентів (діаграма класів, схема алгоритму)
5.	Програмна реалізація програмного забезпечення	05.05	Тексти програмного забезпечення
6.	Тестування програмного забезпечення	10.05	Тести, результати тестування
7.	Розробка матеріалів текстової частини проекту	14.05	Пояснювальна записка
8.	Розробка матеріалів графічної частини проекту	20.05	Графічний матеріал проекту
9.	Оформлення технічної документації проекту	29.05	Технічна документація

7 ПОРЯДОК КОНТРОЛЮ ТА ПРИЙМАННЯ

Тестування розробленого програмного продукту виконується відповідно до “Програми та методики тестування”.

					КП.ІП-9228.045420.01.91	Арк.
Змін.	Арк.	№ докум.	Підп.	Дата.		12

**Пояснювальна записка
до дипломного проєкту**

на тему: Програмне забезпечення для перевірки якості написання SQL-
запитів

КПІ.ПІ-9228.045420.02.81

Київ – 2023

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ.....	4
ВСТУП	6
1 АНАЛІЗ ВИМОГ ДО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ	7
1.1 Загальні положення	7
1.2 Змістовний опис і аналіз предметної області	8
1.3 Аналіз успішних ІТ-проектів.....	9
1.3.1 Аналіз відомих алгоритмічних та технічних рішень.....	9
1.3.2 Аналіз відомих програмних продуктів.....	12
1.4 Аналіз вимог до програмного забезпечення	15
1.4.1 Розроблення функціональних вимог	15
1.4.2 Розроблення нефункціональних вимог	20
1.5 Постановка задачі	20
Висновки до розділу	21
2 МОДЕЛЮВАННЯ ТА КОНСТРУЮВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ	22
2.1 Моделювання та аналіз програмного забезпечення	22
2.1.1 Розробка контекстної діаграми	22
2.1.2 Діаграма діяльності	24
2.1.3 Графічний інтерфейс користувача.....	25
2.2 Архітектура програмного забезпечення.....	26
2.2.1 Статична модель програмного забезпечення	27
2.2.2 Динамічна модель програмного забезпечення	36
2.3 Конструювання програмного забезпечення.....	37
2.3.1 Вибір мови та середовища розробки	37
2.3.2 Діаграма компонентів	42
2.4 Реалізація проекту системи	43
2.4.1 Реалізація інтерфейсу користувача.....	43
2.4.2 Реалізація REST API.....	45
2.5 Аналіз безпеки даних	46

Висновки до розділу	47
3 АНАЛІЗ ЯКОСТІ ТА ТЕСТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ	
49	
3.1 Аналіз якості ПЗ.....	49
3.2 Опис процесів тестування	51
3.3 Опис контрольного прикладу	56
Висновки до розділу	58
4 ВПРОВАДЖЕННЯ ТА СУПРОВІД ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ.	60
4.1 Розгортання програмного забезпечення.....	60
4.2 Підтримка програмного забезпечення.....	61
Висновки до розділу	62
ВИСНОВКИ.....	64
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ.....	66
ДОДАТОК А ЗВІТ ПОДІБНОСТІ.....	68

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

API - Application Programming Interface - Інтерфейс прикладного програмування

ASGI - Asynchronous Server Gateway Interface - Інтерфейс шлюзу асинхронного сервера

CI/CD - continuous integration / continuous delivery - безперервна інтеграція / безперервна доставка

CRUD - create read update delete - створення, читання, оновлення і видалення (4 основні функції управління даними)

CSS - Cascading Style Sheets - Каскадні таблиці стилів

FQDN - Fully Qualified Domain Name - Повне доменне ім'я

HTML - HyperText Markup Language - Мова розмітки гіпертексту

HTTP - HyperText Transfer Protocol - Протокол передачі гіпертексту

HTTPD - Hyper Text Transfer Protocol Daemon - Серверне програмне забезпечення для роботи з протоколом HTTP у режимі демона

JSON - JavaScript Object Notation - Нотація об'єктів JavaScript

QA - Quality Assurance - Гарантія якості

REST - Representational State Transfer - Передача репрезентативного стану

SQL -Structured Query Language – Структурована мова запитів

STD – State transfer diagram – Діаграма переходу станів

UI – User interface - інтерфейс користувача

UML - Unified Modeling Language - Уніфікована мова моделювання

URL - Uniform Resource Locator – Єдиний вказівник на ресурс

WSGI - Web Server Gateway Interface - Інтерфейс шлюзу веб-сервера

					КПІ.ІП-9228.045420.02.81	Арк.
Змін.	Арк.	№ докум.	Підп.	Дата.		4

БД – База даних

ПЗ – Програмне забезпечення

					КПІ.ІП-9228.045420.02.81	Арк.
						5
Змін.	Арк.	№ докум.	Підп.	Дата.		

ВСТУП

Інформаційні технології і процеси цифрової трансформації проникають у всі сфери життя сучасної людини, з цим росте попит на кваліфікованих розробників, що здатні ефективно працювати з базами даних. Використання баз даних та мови SQL (Structured Query Language) є необхідними компонентами для розробки програмного забезпечення.

Написання правильних SQL-запитів вимагає від програмістів не тільки знання синтаксису SQL, але й розуміння принципів роботи баз даних та оптимальних методів обробки даних. Неправильно структуровані або неоптимальні SQL-запити можуть призвести до недостовірних результатів, втрати даних, або низької продуктивності системи.

Застосування програмного забезпечення, яке автоматично перевіряє якість написання SQL-запитів, може значно полегшити роботу розробників і покращити загальну якість роботи з базами даних. Таке програмне забезпечення може надавати рекомендації щодо оптимізації запитів, виявляти потенційні помилки або недоліки в структурі запитів, а також попереджати про можливі проблеми, що допомагає уникнути непередбачуваних наслідків та покращити продуктивність баз даних.

Для початківців, які тільки вчаться програмуванню і роботі з базами даних, таке забезпечення може стати незамінним інструментом навчання. Воно може надати детальні рекомендації та пояснення щодо найкращих практик написання SQL-запитів. Новачки зможуть вчасно виявляти та виправляти свої помилки, отримувати рекомендації щодо оптимізації запитів і навчитися писати ефективні SQL-запити з самого початку свого професійного розвитку.

					КПІ.ІП-9228.045420.02.81	Арк.
Змін.	Арк.	№ докум.	Підп.	Дата.		6

1 АНАЛІЗ ВИМОГ ДО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

1.1 Загальні положення

Перш ніж перейти до аналізу предметної області та аналогічних ІТ-проектів, формулюванню вимог до програмного забезпечення визначимо основні поняття важливі для подальшого аналізу.

SQL (Structured Query Language) - SQL є мовою програмування, яка використовується для управління та маніпулювання базами даних. Для ефективного розуміння проблеми перевірки якості SQL запитів, необхідно мати глибоке розуміння мови SQL та її основних конструкцій [1].

Якість SQL запитів - поняття якості SQL запитів охоплює кілька аспектів, включаючи синтаксичну правильність, ефективність, оптимізацію, безпеку та зручність читання. Основна мета програмного забезпечення з перевірки якості SQL запитів полягає у виявленні потенційних проблем та наданні рекомендацій щодо поліпшення якості запитів.

Алгоритми перевірки SQL запитів - для реалізації програмного забезпечення з перевірки якості SQL запитів необхідно вивчати та аналізувати різні алгоритми, які використовуються для перевірки правильності та оптимізації запитів. Це можуть бути алгоритми лексичного та синтаксичного аналізу, алгоритми оптимізації запитів, алгоритми виявлення потенційних проблем тощо.

Параметри, екранування та підготовка запитів - Особливу увагу слід приділити питанням безпеки SQL запитів. Важливо вивчити та розуміти концепцію параметрів, що дозволяє уникнути SQL-ін'єкцій, а також принципи екранування та підготовки запитів для запобігання зловживанням та атакам на безпеку [2].

					КПІ.ІП-9228.045420.02.81	Арк.
Змін.	Арк.	№ докум.	Підп.	Дата.		7

1.2 Змістовний опис і аналіз предметної області

Предметною областю даної дипломної роботи є програмне забезпечення з перевірки якості написання SQL запитів.

ПЗ для перевірки якості написання SQL запитів є інструментом, призначеним для оцінки та аналізу правильності та ефективності SQL запитів, які використовуються для доступу до баз даних.

Написання ефективних та коректних SQL запитів може бути складною задачею, особливо для початківців, і невірні або неефективні запити можуть призвести до неправильних результатів або погіршення продуктивності системи.

Програмне забезпечення з перевірки якості написання SQL запитів буде допомагати користувачам виявляти та виправляти проблеми з їхніми запитами. Розробка буде мати набір правил, які аналізують синтаксичну правильність, ефективність та безпеку запитів.

Основні функції програмного забезпечення з перевірки якості написання SQL запитів включають [3]:

- синтаксична перевірка: ПЗ перевіряє, чи відповідають запити синтаксичним правилам мови SQL. Воно виявляє помилки, такі як неправильно розміщені оператори, неправильне використання ключових слів тощо;
- аналіз оптимізації: ПЗ аналізує структуру запитів та рекомендує оптимальні методи доступу до даних. Воно може виявляти запити, які можна покращити шляхом використання індексів, оптимізації JOIN-операцій та інших методів;

Ціллю дипломного проєкту є полегшення процесу розробки та підтримки SQL запитів, покращення продуктивності та зменшення кількості помилок та витрат на їх виправлення.

Загалом, програмне забезпечення з перевірки якості написання SQL запитів є корисним інструментом для розробників та аналітиків, які працюють

з базами даних і бажають забезпечити якість, ефективність та безпеку своїх SQL запитів

1.3 Аналіз успішних ІТ-проектів

1.3.1 Аналіз відомих алгоритмічних та технічних рішень

Існує декілька видів аналізу, для перевірки якості написання SQL запитів. Їх можна класифікувати наступним чином:

- синтаксичний аналіз: Використовується для перевірки правильності синтаксису SQL запитів. Алгоритми цього типу перевіряють, чи відповідає запит синтаксичним правилам мови SQL;
- семантичний аналіз: Забезпечує перевірку коректності використання об'єктів бази даних, таблиць, стовпців та інших елементів в SQL запитах. Цей тип алгоритмів перевіряє наявність таких об'єктів і їхню відповідність з правилами бази даних;
- оптимізація запитів: Алгоритми оптимізації SQL запитів стараються знайти оптимальний план виконання запиту, щоб забезпечити його ефективність. Вони враховують такі фактори, як використання індексів, розподіл обробки даних та використання кешування для покращення продуктивності запиту;
- аналіз статистики: Враховується статистика про розподіл даних в базі даних для прийняття рішень щодо виконання запиту. Алгоритми аналізу статистики визначають оптимальний план запиту, враховуючи обсяги даних та їх розподіл;
- виявлення SQL-ін'єкцій: Алгоритми виявлення SQL-ін'єкцій перевіряють SQL запити на наявність вразливостей, які можуть бути використані для несанкціонованого доступу до бази даних або зміни даних. Вони враховують потенційні загрози безпеки;
- перевірка форматування: Алгоритми перевірки форматування SQL запитів забезпечують виконання стандартів форматування, таких як

					КПІ.ІП-9228.045420.02.81	Арк.
Змін.	Арк.	№ докум.	Підп.	Дата.		9

розташування відступів, використання пробілів та роздільників. Вони полегшують читання та розуміння коду SQL;

– використання кращих практик: Ці алгоритми перевіряють, чи використовуються кращі практики при написанні SQL запитів, такі як використання оптимальних ключів для з'єднань, використання індексів та оптимізація запитів;

В даному дипломному проєкті було прийнято рішення сконцентруватись на розробці і впровадженні синтаксичного, стилістичного та оптимізаційного аналізів.

Синтаксичний аналіз (парсинг) - це процес аналізу вхідної послідовності символів з метою розбору граматичної структури відповідно до заданої формальної граматики. Під час синтаксичного аналізу текст перетворюється в структуру даних, зазвичай в дерево, яке відображає синтаксичну структуру вхідної послідовності і дозволяє подальшу обробку і аналіз. Розглянемо декілька видів алгоритмів для синтаксичного аналізу.

Метод рекурсивного спуску (Recursive Descent)[17] - є одним з популярних підходів до синтаксичного аналізу. Він базується на рекурсивних викликах функцій, які відповідають нелистковим символам граматики.

Основна ідея методу рекурсивного спуску полягає в тому, що граматика розбивається на набір правил, кожне з яких відповідає одному нелистковому символу. Для кожного нелисткового символу створюється окрема функція-парсер, яка виконує розбір для цього символу. Під час розбору функції-парсера можуть викликатися інші функції-парсери для обробки підрозділів граматики.

Процес розбору починається з кореневого символу граматики. Функція-парсер для цього символу виконує розбір і викликає функції-парсери для нелисткових символів, що з'являються в правилах граматики. Цей процес продовжується рекурсивно, доки не досягнуто листкових символів, які відповідають терміналам (лексемам).

При реалізації методу рекурсивного спуску, кожна функція-парсер відповідає правилу граматики і має завдання розпізнати послідовність символів, що відповідає цьому правилу. Функція-парсер може виконувати обробку та аналіз токенів, викликати інші функції-парсери для обробки підрозділів та здійснювати взаємодію з іншими компонентами парсера.

Метод рекурсивного спуску має деякі переваги, такі як простота реалізації, читабельність коду та можливість використання додаткових функцій для обробки семантики. Однак, його використання може бути обмежене граматиками, що не мають лівої рекурсії або вкладених викликів.

Для ситуацій, в яких не підходить метод рекурсивного спуску може застосовуватись аналіз LL (Left-to-right, Leftmost derivation) або LR (Left-to-right, Rightmost derivation). Це дві поширені техніки парсингу, що використовуються в комп'ютерній науці для аналізу синтаксичної структури вхідного рядка на основі формальної граматики.

У методі LL використовується предиктивний аналізатор[18], який базується на проглядуванні певного числа символів вхідного рядка (lookahead), щоб визначити правило граматики для вибору. Для кожного нетермінального символу граматики створюється окрема функція-парсер, яка виконує відповідний аналіз для даного символу.

Один з найпоширеніших варіантів методу LL - LL(1), де 1 вказує на один символ проглядування вхідного рядка. Для успішного розбору граматики методом LL(1) граматика повинна бути LL(1)-граматикою, тобто має властивість однозначності. Це означає, що для кожного нетермінального символу та проглядаючого символу існує єдине правило граматики, яке можна застосувати.

Метод LR працює аналогічно, але при його використанні правила для аналізу застосовуються у зворотньому порядку. Синтаксичні аналізатори LR використовують стек для відстеження часткового аналізу та таблицю аналізу для визначення відповідних дій. Найпоширенішим варіантом синтаксичного

аналізу LR є LR(1), який використовує один символ попереднього перегляду для визначення дій аналізатора.

Ще одним алгоритмом синтаксичного аналізу є LALR (Look-Ahead LR) [18] це варіант синтаксичного аналізу LR, який поєднує ефективність аналізу LR(1) із більш компактною таблицею синтаксичного аналізу. Парсери LALR часто використовуються на практиці через їх баланс між потужністю та ефективністю. LALR аналіз дозволяє ефективно та швидко аналізувати послідовності токенів з великою кількістю можливих правил граматики. Він знаходить широке застосування в компіляторних системах та інших областях, де потрібен синтаксичний аналіз мов програмування.

1.3.2 Аналіз відомих програмних продуктів

Наразі існує кілька відомих програмних рішень для перевірки якості написання SQL запитів. Ось декілька з них:

- SQLLint є простим інструментом, який перевіряє синтаксичну правильність SQL-запитів згідно з встановленими стандартами. Він допомагає виявляти основні помилки і недоліки в SQL-коді, такі як неправильний синтаксис і використання недійсних ключових слів. SQLLint не має великого набору функціональності, але він добре підходить для базових перевірок. SQLLint може бути інтегрованим з текстовими редакторами або використовуватися в якості окремого інструменту командного рядка. Основними недоліками SQLLint є його обмежена підтримка SQL діалектів, відносно невеликі ресурси для аналізу, що призводить до затримок, особливо, якщо обробляються великі обсяги коду та відсутність повноцінних перевірок безпеки;

- SQLFluff є потужною програмою для перевірки якості SQL-коду. Вона пропонує більш широкий спектр перевірок, включаючи перевірку стилевих правил, оформлення, неймінгу та структури SQL-запитів. SQLFluff підтримує різні діалекти SQL і має гнучкі налаштування. Основними

недоліками цієї програми є доволі високій вхідний поріг: інтерпретація результатів виводу може бути складною для новачків, як і розуміння правил перевірки та параметрів налаштування. Також у забезпеченні відсутня можливість автоматичної корекції коду, програма призначена в першу чергу для надання зворотного зв'язку;

- SQLCop є бібліотекою, яка перевіряє виконання набору правил та настанов щодо якості SQL-коду. Вона дозволяє налаштовувати правила перевірки, включаючи оптимізацію запитів, використання індексів, уникнення потенційних проблем безпеки та інші аспекти. SQLCop надає звіти про помилки та порушення правил і може бути інтегрованою у робочий процес розробки. Як і з SQLFluff, SQLCop має високий вхідний рівень і також може бути складною у налаштуванні і користуванні для новачків, до того ж, він вимагає попередньої конфігурації, навідміну від інших представлених продуктів;

- SonarLint є розширенням для інтегрованих середовищ розробки, таких як IntelliJ IDEA, Visual Studio, Eclipse і т.д. Він надає можливість перевіряти SQL-код на основі правил, встановлених в SonarQube. SonarLint допомагає виявляти потенційні проблеми з безпекою, ефективністю та стилістикою SQL-коду, надаючи рекомендації щодо покращення. З недоліків SonarLint можна виділити його залежність від інтегрованих середовищ розробки, складність попередньої конфігурації та великі вимоги до ресурсів користувача (пам'яті, завантаженості процесора);

Ці програмні рішення надають різноманітні функції для перевірки якості написання SQL запитів. Вибір залежить від ваших потреб, вимог до функціональності та інтеграції з вашим робочим процесом.

Проведемо порівняння дипломного проєкту з програмними аналогами у таблиці 1.1

					КПІ.ІП-9228.045420.02.81	Арк.
Змін.	Арк.	№ докум.	Підп.	Дата.		13

Таблиця 1.1 – Порівняння з аналогом

Функціонал	Дипломний проєкт	SQLFluff	SQLCop	SonarLint	SQLLint
Синтаксичний аналіз	+	+	+	+	+
Перевірка форматування	+	+	+	+	+
Стандартизація	+	+	+	+	+
Наявність незалежного веб-застосунку	+	-	-	-	+
Отримання результатів аналізу користувачем у вигляді підказок	+	-	+	+	-
Оптимізація запитів	+	+	+	+	-

Отже, враховуючи аналіз таблиці 1.1 є необхідність розробки окремого програмного забезпечення з перевірки якості написання SQL, що буде містити такі особливості як потужний синтаксичний аналізатор, перевірку стилю та стандартів, окрему можливість форматування запиту та отримання результатів аналізу користувачем у вигляді підказок та звіту.

1.4 Аналіз вимог до програмного забезпечення

1.4.1 Розроблення функціональних вимог

Перед формулюванням функціональних вимог побудуємо діаграму варіантів використання та необхідні допоміжні таблиці до неї.

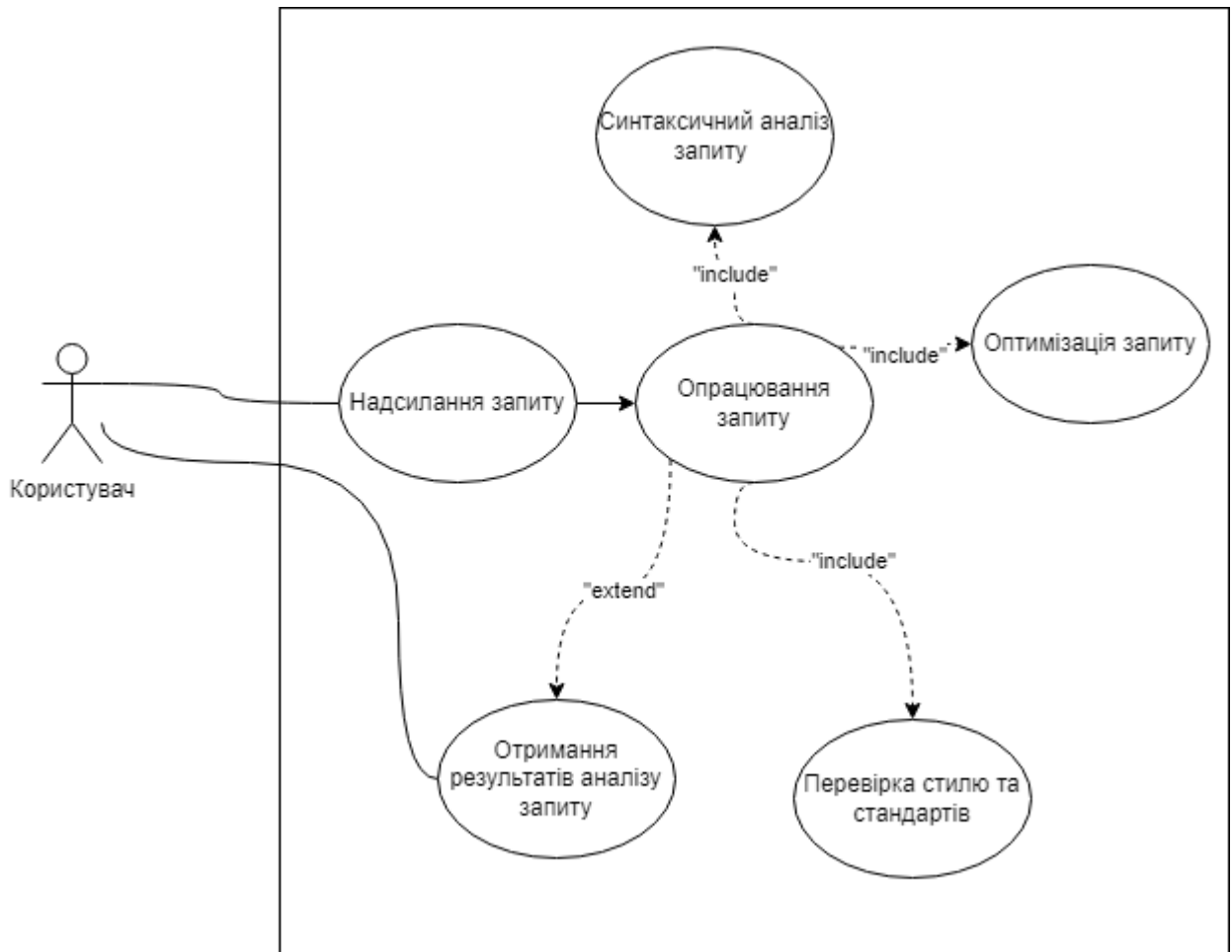


Рисунок 1.1 – Діаграма варіантів використання

Розглянемо детальніше кожен із виділених варіантів використання

Таблиця 1.2 – Варіант використання UC-O1

Use Case ID	UC-O1
Use Case Name	Надсилання запиту
Goals	Надіслати SQL запит у веб-застосунок
Actors	Користувач
Trigger	Користувач створив новий SQL запит і його потрібно перевірити на якість написання
Pre-conditions	Користувач заходить до веб-застосунку

Продовження таблиці 1.2

Flow of events	- На головній сторінці веб-застосунку Актор вводить SQL запит; - Актор надсилає введений запит
Extension	-
Post-condition	Актор надіслав запит у систему

Таблиця 1.3 – Варіант використання UC-O2

Use Case ID	UC-O2
Use Case Name	Опрацювання запиту
Goals	Провести аналіз якості написання SQL запиту
Actors	Користувач
Trigger	Користувач надіслав SQL запит у систему
Pre-conditions	Користувач заходить до веб-застосунку і надсилає запит
Flow of events	- Актор надсилає запит у систему - Система проводить аналіз якості запиту
Extension	-
Post-condition	SQL запит було перевірено, результати були надіслані користувачу

Таблиця 1.4 – Варіант використання UC-O3

Use Case ID	UC-O3
Use Case Name	Синтаксичний аналіз запиту
Goals	Перевірити синтаксичну правильність написання SQL запиту
Actors	Користувач
Trigger	Користувач надіслав SQL запит у систему
Pre-conditions	Користувач заходить до веб-застосунку і надсилає запит
Flow of events	- Актор надсилає запит у систему - Система проводить синтаксичний аналіз запиту
Extension	-
Post-condition	SQL запит було перевірено на синтаксичні помилки

Таблиця 1.5 – Варіант використання UC-O4

Use Case ID	UC-O5
Use Case Name	Оптимізація запиту
Goals	Оптимізувати надісланий SQL запит
Actors	Користувач
Trigger	Користувач надіслав SQL запит у систему
Pre-conditions	Користувач заходить до веб-застосунку і надсилає запит
Flow of events	- Актор надсилає запит у систему - Система проводить оптимізацію запиту
Extension	
Post-condition	SQL запит було оптимізовано

Таблиця 1.6 – Варіант використання UC-O5

Use Case ID	UC-O7
Use Case Name	Перевірка стилю та стандартів
Goals	Перевірити надісланий SQL запит на помилки форматування
Actors	Користувач
Trigger	Користувач надіслав SQL запит у систему
Pre-conditions	Користувач заходить до веб-застосунку і надсилає запит
Flow of events	- Актор надсилає запит у систему - Система проводить перевірку запиту на помилки форматування
Extension	Отримання результатів аналізу запиту
Post-condition	SQL запит було перевірено на наявність помилок у форматуванні

Таблиця 1.7 – Варіант використання UC-O6

Use Case ID	UC-O8
Use Case Name	Отримання результатів аналізу запиту
Goals	Отримати підказки щодо помилок або покращень у надісланому SQL запиті
Actors	Користувач
Trigger	Запит було проаналізовано системою
Pre-conditions	Користувач заходить до веб-застосунку і надсилає запит

Продовження таблиці 1.7

Flow of events	- Користувач надсилає запит - Запит було проаналізовано системою - Система виводить користувачу результати аналізу запиту
Extension	-
Post-condition	Користувач отримує результати аналізу SQL запиту

Згідно з описаними варіантами використання можна сформулювати наступні функціональні вимоги.

Таблиця 1.8 – Функціональна вимога FR-1

Назва	Введення запиту у програму
Опис	Система повинна надати користувачеві можливість ввести запит для аналізу

Таблиця 1.9 – Функціональна вимога FR-2

Назва	Синтаксичний аналіз запитів
Опис	Система повинна провести синтаксичний аналіз отриманого запиту

Таблиця 1.10 – Функціональна вимога FR-3

Назва	Оптимізація запитів
Опис	Система повинна провести оптимізаційний аналіз запиту

Таблиця 1.11 – Функціональна вимога FR-4

Назва	Перевірка стилю запитів
Опис	Система повинна провести стилістичний аналіз запиту

Таблиця 1.12 – Функціональна вимога FR-5

Назва	Перевірка стандартів запитів
Опис	Система повинна провести аналіз запиту на відповідність стандартам

Таблиця 1.13 – Функціональна вимога FR-6

Назва	Отримання результатів
Опис	Система повинна дати змогу користувачу отримати результати аналізу запиту

Представимо відповідність варіантів використання до функціональних вимог у вигляді матриці трасування.

Таблиця. 1.8 – Матриця трасування

	UC-01	UC-02	UC-03	UC-04	UC-05	UC-06
R 1						
R 2						
R 3						
R 4						
R 5						
R 6						

1.4.2 Розроблення нефункціональних вимог

Нефункціональні вимоги програмного забезпечення для перевірки якості написання SQL запитів включають:

- ефективність, програмне забезпечення повинно забезпечувати швидку перевірку SQL запитів без зайвих затримок;
- програмне забезпечення повинно бути стійким до помилок і непередбачуваних ситуацій. Воно повинно правильно обробляти помилки, які можуть виникнути при неправильному написанні або виконанні SQL запитів;
- стандартизованість, перевірка дотримання правил та стандартів повинна будуватись на основі Mozilla SQL Style Guide [5];
- зрозумілість і простота використання: Програмне забезпечення повинно мати зрозумілий та легкий у використанні інтерфейс. Воно повинно надавати зрозумілі повідомлення про помилки, а також можливість надання рекомендацій для виправлення проблем в SQL запитах;
- основний програмний модуль повинен бути реалізований у вигляді мікро - сервісу з певним API. Інтерфейс користувача – web-інтерфейс, що взаємодіє з розробленим мікросервісом літера. Така архітектура дозволяє збільшувати кількість інтерфейсів (мобільний додаток, десктопний додаток, тощо) без зміни основного мікросервісу;

Ці нефункціональні вимоги допоможуть забезпечити якість перевірки SQL запитів та забезпечити ефективну та безпечну роботу з базами даних.

1.5 Постановка задачі

Метою даного дипломного проекту є полегшення процесу розробки та підтримки SQL запитів, покращення продуктивності та безпеки розроблюваних систем, а також зменшення кількості помилок та витрат на їх виправлення. Поставлені наступні задачі:

- вибрати та обґрунтувати вибір алгоритмів для перевірки якості написання SQL запитів;

					КПІ.ІП-9228.045420.02.81	Арк.
						20
Змін.	Арк.	№ докум.	Підп.	Дата.		

– програмне забезпечення повинно мати зрозумілий інтерфейс, ефективно обробляти дані та проводити перевірку згідно стандартам Mozilla SQL Style Guide;

– програмне забезпечення повинно проводити синтаксичний аналіз, аналіз форматування та оптимізаційний аналіз;

Висновки до розділу

У першому розділі дипломного проекту було розглянуто загальні положення, наведено змістовний опис та аналіз предметної області. Було визначено, що написання якісних SQL-запитів є необхідним вмінням як для новачків так і для професійних розробників, адже запити поганої якості можуть призводити до помилок та неправильних результатів роботи під час розробки.

Було проаналізовано відомі технічні та алгоритмічні рішення. Визначено можливі види аналізу для забезпечення якості написання SQL-запитів. Досліджено алгоритми для синтаксичного аналізу, їх відмінності та специфіку застосування.

Був проведений аналіз декількох відомих програмних продуктів для перевірки якості написання SQL-запитів. Основними їх недоліками є складне налаштування, залежність від стороннього програмного забезпечення та середовищ розробки та відсутність зрозумілих результатів аналізу.

Було створено діаграму варіантів використання, сформульовано функціональні вимоги і складено матрицю трасування вимог. Крім того, були визначені нефункціональні вимоги та сформульована задача для дипломної роботи.

2 МОДЕЛЮВАННЯ ТА КОНСТРУЮВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

2.1 Моделювання та аналіз програмного забезпечення

2.1.1 Розробка контекстної діаграми

У першу чергу, контекстна діаграма [4, 6] забезпечує можливість швидко, зрозуміло та стисло описати призначення і обмеження системи, виявити та вирішити загальні непорозуміння щодо цих аспектів, а також продемонструвати та домовитися про її масштаб.

Друге її призначення - служити джерелом для швидкої генерації первинного набору системних функціональних вимог при необхідності проектування системи не «зверху-вниз», від бізнес-моделі, бізнес-вимог, моделі діяльності організації, вимог зацікавлених осіб, моделі використання, як пропонує нам системна інженерія, а «з середини».

Контекстна діаграма належить до категорії діаграм, що описують систему лише на рівні «чорного ящика» — лише зовнішні властивості (потоки даних).

Контекстна діаграма містить 3 основні компоненти:

- проєктований об'єкт (наприклад, система);
- взаємодіючі з проєктованим об'єктом елементи оточення (групи користувачів, суміжні системи);
- потоки даних (вихідні та вхідні);
- розроблена контекстна діаграма подана на рисунку 2.1;



Рисунок 2.1 – Контекстна діаграма

В зображеному випадку в якості додатку розглядається мікросервіс (лінтер), а web-сторінка, що є інтерфейсом користувача, лише формує та відправляє запит на з використанням API мікросервісу. Користувач направляє текст запиту в початковому вигляді (raw data) й отримує перелік підказок (hints) по покращенню запиту. Кожна підказка містить посилання на конкретний рядок запиту.

Приклад запиту до системи.

```

SELECT *
      FROM
        TableNames AS a      -- Name = 'Lee, Barry'
INNER JOIN
        TableFullNames AS b -- Name = 'Lee.Barry'
      on /* */ a.id = b.id
WHERE
      LEFT(a.name, 1) = 'n' AND
      LEFT(a.name, 1) = 'z';
  
```

Приклад відгуку системи.

Рядок 1 -> SELECT <- Кореневе слово не на окремому рядку.
 Рядок 2 -> FROM <- Кореневе слово не вирівняно за лівим краєм.
 Рядок 6 -> on <- Зарезервоване слово у нижньому регістрі.
 Рядок 8 -> AND <- Логічний оператор AND не на початку рядка.

При реалізації системи запиту та відгуки до API обгорнуті у JSON – формат.

2.1.2 Діаграма діяльності

При проектуванні системи часто потрібно деталізувати алгоритмічні та логічні деталі операцій, які виконує система. Для цього зазвичай використовуються блок-схеми.

У мові UML для моделювання процесу виконання операцій використовуються діаграми діяльності (activity diagrams) [4,6]. Ці діаграми схожі на діаграми станів, але вони представляють дії системи. Вершини діаграми діяльності відображають стани дій, а дуги - переходи між ними.

Кожен етап, зображений на діаграмі діяльності, відповідає виконанню окремої простої операції. Діаграми діяльності в мові UML дозволяють втілити процедурну та синхронну логіку керування, і вони мають відповідні терміни і семантику в метамоделі UML.

Основне використання діаграм діяльності - це візуалізація реалізації операцій класів, коли потрібно показати алгоритми їх виконання. Кожен етап на діаграмі може представляти виконання операції класу або її фрагменту, що дозволяє використовувати діаграми діяльності для опису реакцій на внутрішні події системи.

Стан дії (action state) є спеціальним видом стану, який має певну вхідну дію і принаймні один перехід, що виходить з нього. Стан дії є елементарним і, отже, не має внутрішніх переходів. Дія може бути представлена природною мовою, псевдокодом або мовою програмування.

Оскільки відбувається перевірка SQL – запиту відповідності усьому набору правил Mozilla, діаграма діяльності (Рисунок 2.2) є лінійною з певними розгалуженнями. По-перше, розгалуження відтворюють паралельну роботу алгоритму з різними правилами, що дозволяє зменшити загальний час формування відгуку API. По друге, розгалуження вказує на застосування чи не застосування опції форматування запиту згідно розглянутих синтаксичних стандартів.

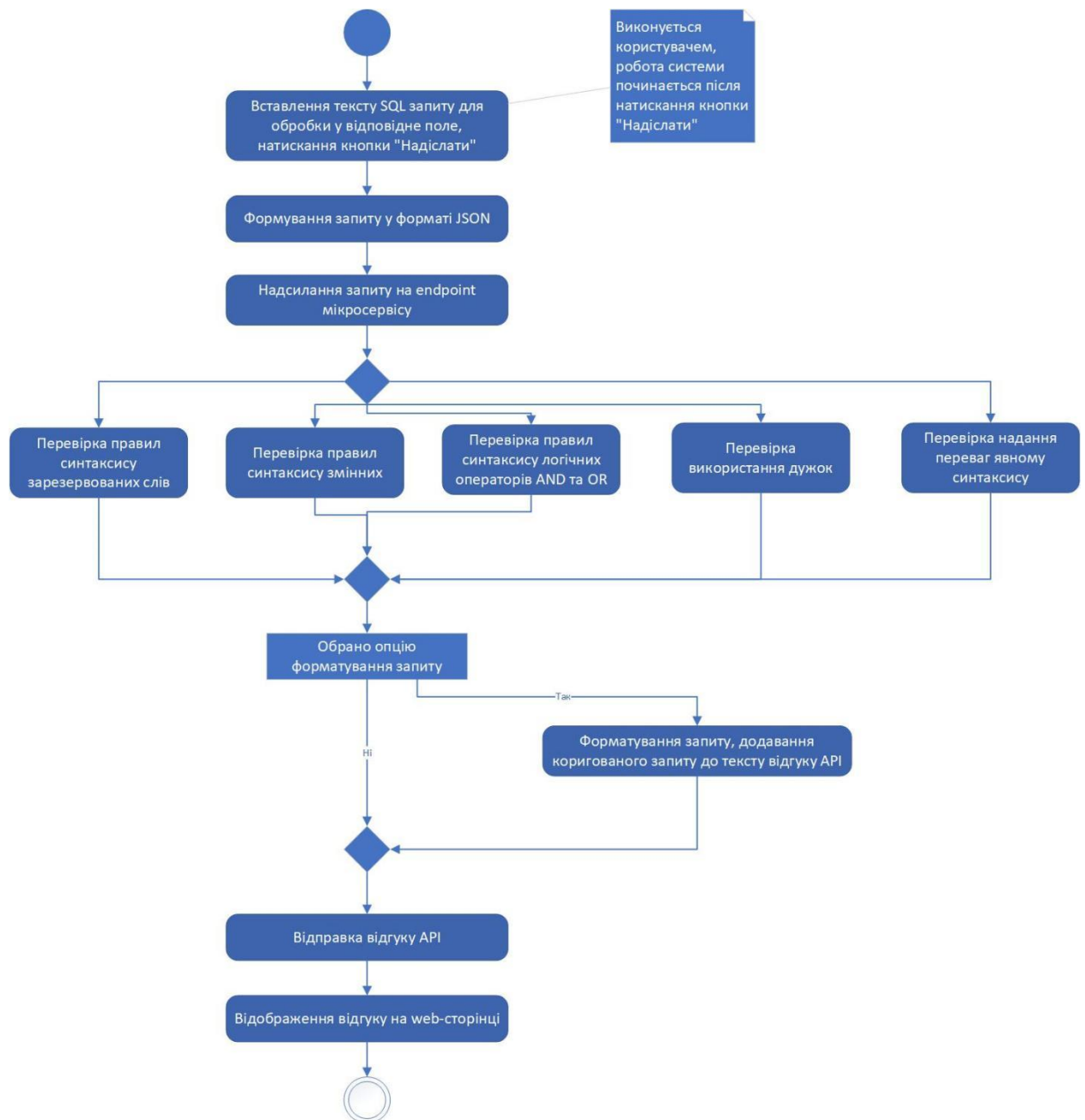


Рисунок 2.2 – Діаграма діяльності синтаксичного аналізу

2.1.3 Графічний інтерфейс користувача

Інтерфейс користувача (UI) — це «спосіб, яким ви виконуєте будь-яке завдання за допомогою будь-якого продукту, а саме дії, що вами вчиняються і те, що ви отримуєте у відповідь».

Макети графічних інтерфейсів відповідають сценаріям використання додатку. Враховуючи наявність єдиного варіанту використання, маємо єдину сторінку графічного інтерфейсу, макет якої наведено на рисунку 2.3.

Рисунок 2.3 – Макет графічного інтерфейсу

2.2 Архітектура програмного забезпечення

Система складається з мікросервісу, який виконує синтаксичний та стилістичний аналіз SQL – запиту, поданого у вигляді тексту, та однієї з можливих реалізацій інтерфейсу користувача – web-інтерфейсу, що є одиночною web-сторінкою з інтерактивними елементами.

Таким чином, маємо проект, що складається з двох незалежних у реалізації частин, що поєднуються визначеним контрактом.

Основний мікросервіс з API буде реалізовано з використанням Python - фреймворку FastAPI, web – інтерфейс – HTML5 сторінка з динамічними елементами JavaScript.

2.2.1 Статична модель програмного забезпечення

Аналітичні моделі надійності ПЗ поділяють на динамічні (дискретні і неперервні) і статичні (за областю помилок, за областю даних) [7]

Система використовує дві основні сутності: текст запиту та відповідь лінера. Діаграма сутностей показана на рисунку 2.4.

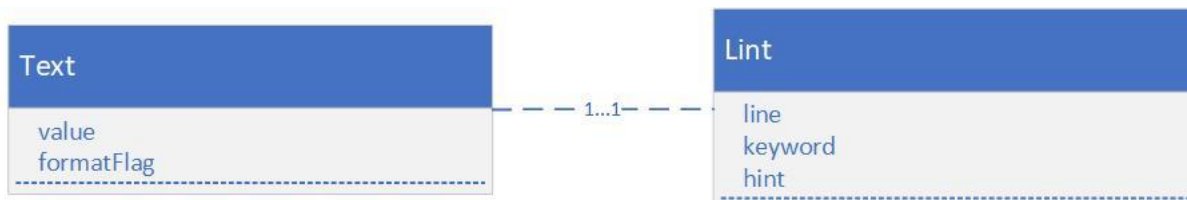


Рисунок 2.4 – ER–діаграма

Між собою згадані сутності відносяться як один до одного.

В реалізації ПЗ мікросервісу використовується об’єктно – орієнтований підхід. Розглянемо діаграму класів для модуля лінера.

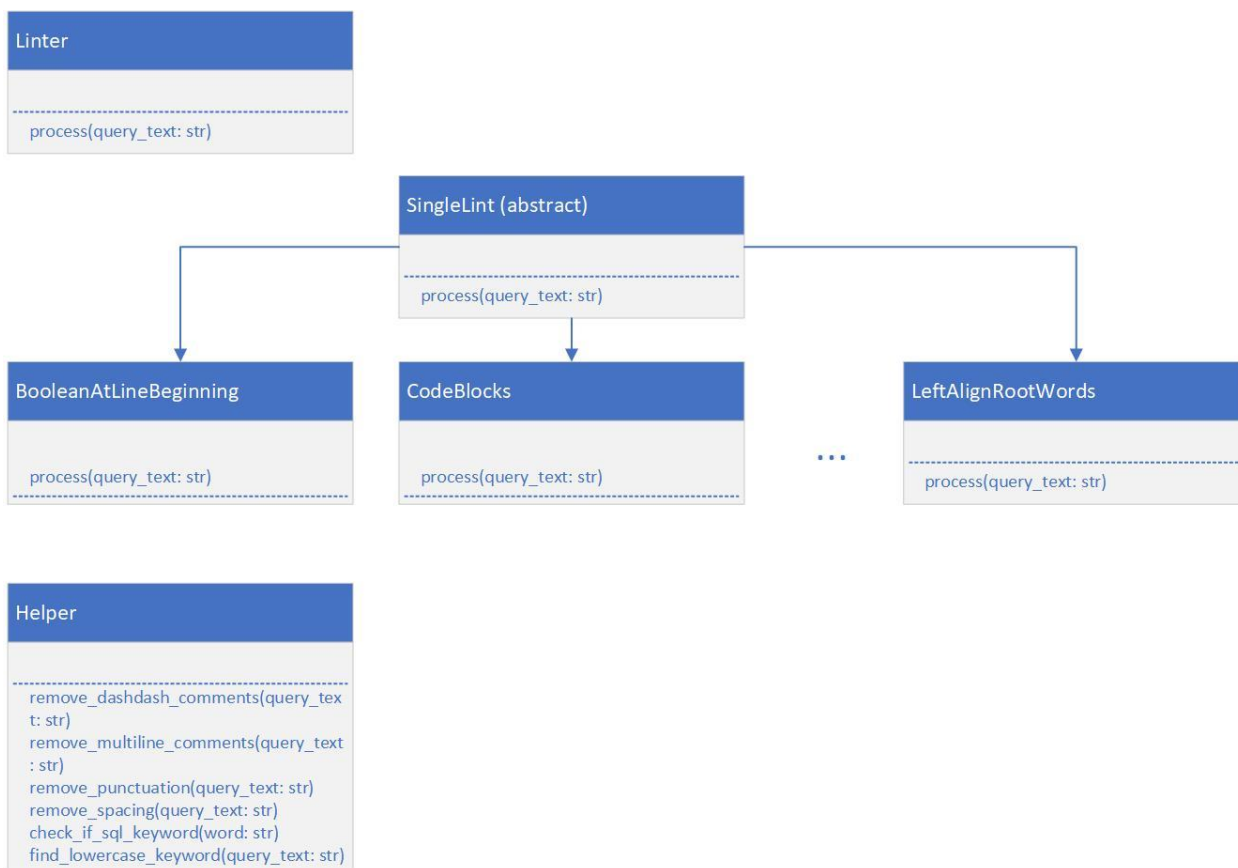


Рисунок 2.5 – Діаграма класів модуля лінера

Модуль лінтера складається з головного класу Linter, який має єдиний статичний метод process(query_text:str). Модуль отримує в якості аргументу текст запиту для аналізу та повертає перелік (список) підказок, що є результатами аналізу запиту.

Кожен окремий тип аналізу втілюється окремим класом, що успадковується від абстрактного класу SingleLint. Це дозволяє мати єдину структуру усіх аналізаторів запиту. Що в свою чергу дозволяє ітеративну обробку тексту запиту після препроцесінгу. При конструюванні класу SingleLint розглядалося два альтернативних методи реалізації виклику, обидва з яких притаманні при реалізації аналізу текстів. Перший – реалізація методу __call__(), що дозволяє викликати метод обробки безпосередньо за ім'ям класу. Другий – реалізація статичного методу process(). Було зроблено вибір на користь другого методу реалізації, оскільки клас SingleLint та його нащадки є stateless класами, отже, немає потреби в створенні екземпляру класу. Такий підхід є оптимальним з точки зору оптимізації пам'яті.

Побудова модуля server базується на рекомендаціях (best practice) роботи з FastAPI. В головному модулі fast_api_server створюється та налаштовується екземпляр класу FastAPI(), інші модулі, що входять до сервер, є допоміжними. Модуль crud відповідає за CRUD – операції з базою даних; database – є адаптером бази даних, що використовується; pass_enc_dec відповідає за шифрування/дешифрування паролів користувачів з метою уникнення зберігання паролів у відкритому вигляді.

Додатково використовується зовнішній форматер SQL запитів, який спрацьовує за умови отримання REST API відповідного флагу в запиті. Якщо флаг True – форматується та форматований запит додається до відповіді серверу, в іншому випадку в полі *formatted_request* повертається *None*.

Таблиця 2.1 – Опис функцій модулів програмного забезпечення

Назва методу	Аргументи	Значення, що повертаються	Призначення методу
Server/crud.py			
get_user()	db: Session – екземпляр об'єкту Session (SQLAlchemy), username: str – ім'я користувача	HTTPBasicCredentials – дані автентифікації	Отримання з БД даних автентифікації по імені користувача.
Server/fast_api_server.py			
def get_db():		SessionLocal	Отримання екземпляру SessionLocal для роботи з БД.
check_credentials((	db: SessionLocal , credentials: HTTPBasicCredentials	None	Перевірка коректності даних автентифікації користувача. У випадку, коли логін чи пароль не відповідають коректним даним, викидає HTTPException з кодом 401.

Продовження таблиці 2.1

health_check()	db: Session, credentials: HTTPBasic Credentials, text: Text	Dict	Перевірка стану сервісу. У випадку, коли сервіс працює, повертає { "health_check": "ok" }. Використовується для перевірки статусу сервісу під час тестування та є необхідним у випадку деплойа на певних VPS, наприклад, DigitalOcean.
get_main_respon se()	db: Session credentials: HTTPBasic Credentials text: Text	Response	Метод основного end- point сервісу. Приймає дані авторизації користувача та текст запиту (екземпляр класу Text) повертає результати аналізу переданого SQL запиту в екземплярі класу Response.
Server/pass_enc_dec.py: PasswordEncryptDecrypt			

Продовження таблиці 2.1

encrypt()	data: str		Шифрування паролю для збереження в БД. Пароль передається у відкритому вигляді – рядок.
check(	data: str, encrypted: bytes	bool	Перевірка відповідності пароля. Передається пароль у вигляді рядка, що треба перевірити, та послідовність байтів – пароль в зашифрованому вигляді, що зберігається в БД. Повертає True, якщо паролі збігаються, інакше – False.

Продовження таблиці 2.1

<code>_encrypt_with_salt</code> <code>()</code>	<code>data: str,</code> <code>salt: bytes</code>	<code>bytes</code>	Шифрування пароля. Перший аргумент – пароль, що шифрується. Другий параметр – сіль, рядок даних, який передається геш-функції разом з паролем. Головним чином використовується для захисту від перебору за словником і атак з використанням райдужних таблиць, а також приховування однакових паролів.
Linter/boolean_at_line_beginning.py: BooleanAtLineBeginning(SingleLint)			
<code>process()</code>	<code>query: str</code>	<code>List[Lint]</code>	Передається рядок запиту, повертається перелік лінтів – список екземплярів об'єкту Lint. Перевіряє розташування логічних операторів на початку нового рядка тексту запиту.
Linter/code_blocks.py: CodeBlocks(SingleLint)			

Продовження таблиці 2.1

process()	query: str	List[Lint]	Передається рядок запиту, повертається перелік лінтів – список екземплярів об’єкту Lint. Перевіряє, що кореневі ключові слова знаходяться на початку рядка.
Linter/left_align_root_words.py: LeftAlignKeywords(SingleLint)			
process()	query: str	List[Lint]	Передається рядок запиту, повертається перелік лінтів – список екземплярів об’єкту Lint. Перевіряє, що всі кореневі ключові слова знаходяться на єдиній межі.
Linter/reserved_words.py: ReservedWords(SingleLint)			
process()	query: str	List[Lint]	Передається рядок запиту, повертається перелік лінтів – список екземплярів об’єкту Lint. Перевіряє, що всі ключові слова знаходяться у верхньому регістрі.

Продовження таблиці 2.1

Linter/single_lint.py: SingleLint			
process()	query: str	List[Lint]	Передається рядок запити, повертається перелік лінтів – список екземплярів об'єкту Lint. Метод абстрактного класу, що імплементується у методах – нащадках.
Linter/common.py			
remove_dashdash_comments()	query: str	str	Передається рядок запити, повертається перетворений рядок. Видаляє коментарі стилю --
remove_multiline_comments()	query: str	str	Передається рядок запити, повертається перетворений рядок. Видаляє багаторядкові коментарі.
remove_punctuation()	query: str	str	Передається рядок запити, повертається перетворений рядок. Видаляє знаки пунктуації з тексту запити.

Продовження таблиці 2.1

remove_spacing()	query: str	str	Передається рядок запиту, повертається перетворений рядок. Видаляє пробіли з тексту запиту.
check_if_sql_key word()	word: str	bool	Передається слово запиту, повертається логічне значення. Якщо передане слово – ключове, повертається – True, інакше – False.
find_lowercase_s ql_keywords()	query: str	List[str]	Передається рядок запиту, повертається перелік ключових слів в нижньому регістрі.
conf.py			
db_path()		PosixPath	Повертає шлях до бази даних, що використовує додаток.
models.py			
get_response_fro m_values()	initial_text: str, response_list: List[Hint], formatted_sql_ request: str	Response	Створює екземпляр класу Response (модель вігуку) з даних, що передані.

2.2.2 Динамічна модель програмного забезпечення

Діаграми переходів станів (std) використовуються для моделювання поведінки системи STD (State Transition Diagrams), що залежить від часу або реакцій системи на деякі події.

STD складається з наступних об'єктів [6, 7]:

Стан – модельована система в будь-який заданий момент часу повинна бути точно в одному з кінцевої множини станів.

Початковий стан є стартовою точкою для початкового системного переходу, що відповідає стану системи після її інсталяції. STD повинна мати лише один початковий стан, а також будь-яке (кінцеве) число завершальних станів.

Перехід у визначенні вказує на переміщення моделі з одного стану в інший. Ім'я переходу ідентифікує подію, що спричиняє перехід та керує ним. Зазвичай ця подія складається з керуючого потоку або сигналу, які можуть виникати як у зовнішньому середовищі, так і всередині системи при виконанні певної умови.

Дія - це операція, яка пов'язана з переходом і виконується при здійсненні переходу.

У структурній діаграмі станів (STD) стани представлені вузлами, а переходи - дугами. Умови ідентифікуються іменем переходу і викликають виконання переходу. Дії або відгуки на події прив'язуються до переходів і записуються під певною умовою. Початковий стан на діаграмі повинен мати вхідний перехід, який зображається потоком зі стартового вузла. Застосовуються два способи побудови STD. Перший спосіб полягає в ідентифікації всіх можливих станів та подальшому дослідженні всіх зв'язків (переходів) між ними. За другим способом спочатку будується початковий стан, потім наступні його й т.д.

Система має дві частини – користувацьку (клієнтську) та серверну. Будемо розглядати діаграму станів з точки зору, що система є єдиною, незважаючи на те, що вона розподілена.

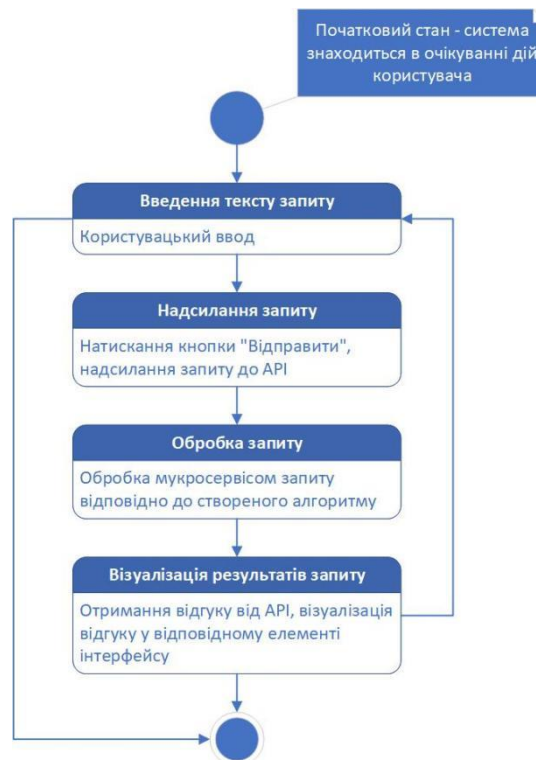


Рисунок 2.6 - Діаграма переходу станів

Як можна бачити з рисунку 2.6, перехід станів системи відбувається лінійно за винятком випадку відмови користувача від використання сервісу або переходу до здійснення нового запиту до системи. Цю лінійність можна було також бачити з діаграми сценаріїв використання.

2.3 Конструювання програмного забезпечення

2.3.1 Вибір мови та середовища розробки

Розроблена розподілена система, як було зазначено в попередніх розділах, має складатися з користувацького інтерфейсу, що відкривається у вигляді сторінки в інтернет – браузері користувача та мікросервісу з API, який безпосередньо виконує необхідні дії та повертає результат для візуалізації.

В ході попереднього аналізу було прийнято рішення web-сторінку інтерфейсу користувача побудувати з використанням загально прийнятої зв'язки HTML5 + CSS3, відправку запиту та візуалізацію результатів реалізувати з використанням Java Script.

HTML (Hypertext Markup Language) - це мова розмітки, використовувана для структурування і відображення веб-сторінок та їх контенту [8]. Наприклад, контент може бути організований у параграфи, списки або представлений за допомогою зображень та таблиць даних.

HTML не є мовою програмування, але мовою розмітки. Вона використовується для комунікації з браузером та вказівки йому, як відображати веб-сторінки для користувача. Складність або простота веб-сторінки залежить від вимог веб-дизайнера та технічного завдання для створення користувацького інтерфейсу. HTML складається з різних елементів, які вкладаються один в одного або накладаються один на одного, надаючи специфічне відображення або функціональність. Він може використовувати теги, щоб зробити слова або зображення посиланням, відформатувати текст курсивом, змінити розмір шрифту та багато іншого.

CSS (Cascading Style Sheets) - це мова таблиць стилів, яка використовується для стилізації веб-сторінок [8]. Подібно до HTML, CSS є мовою програмування. Вона використовується для визначення стилів, які застосовуються до елементів на веб-сторінці, щоб змінити їх вигляд і розташування.

JavaScript (або JS) - це повноцінна динамічна мова програмування, яка використовується для надання динамічної та інтерактивної функціональності на веб-сайтах [9]. Її розробив Брендан Ейх, співзасновник проекту Mozilla, Mozilla Foundation та Mozilla Corporation.

JavaScript є компактною, але дуже гнучкою мовою. Розробники створили багато інструментів на базі JavaScript, які надають широкі можливості з невеликими зусиллями. Це включає програмні інтерфейси

					КПІ.ІП-9228.045420.02.81	Арк.
Змін.	Арк.	№ докум.	Підп.	Дата.		38

браузера, які надають різноманітні функції, такі як динамічне створення HTML та застосування CSS стилів, робота з мультимедіа, взаємодія з веб-камерою та генерація 3D графіки та аудіо. JavaScript також дозволяє розробникам використовувати сторонні API, такі як Twitter або Facebook, для інтеграції функціональності їхніх сервісів на веб-сторінках. Крім того, існують сторонні фреймворки та бібліотеки JavaScript, які допомагають прискорити розробку веб-сайтів та програм.

Мова реалізації основного мікросервісу – Python 3.10 з фреймворком FastAPI, як швидким асинхроним фреймворком, що має необхідну з одного боку й мінімальну з іншого функціональність з коробки. Асинхроність та сбалансована функціональність – переваги фреймворку FastAPI у порівнянні з Django та Flask.

Python — це мова програмування, яка широко використовується в інтернет-додатках, розробці програмного забезпечення, науці даних і машинному навчанні (ML) [10]. Розробники використовують Python, тому що він ефективний, простий у вивченні та працює на різних платформах. Програми мовою Python можна завантажити безкоштовно, вони сумісні з усіма типами систем та підвищують швидкість розробки.

Мова Python має такі переваги [10]:

- розробники можуть легко читати та розуміти програми на Python, оскільки мова має базовий синтаксис, схожий на синтаксис англійської;
- Python допомагає розробникам бути більш продуктивними, оскільки вони можуть писати програми на Python, використовуючи менше рядків коду, ніж іншими мовами;
- Python має велику стандартну бібліотеку, що містить багаторазові коди практично для будь-якого завдання. В результаті розробникам не потрібно писати код із нуля;
- розробники можуть легко поєднувати Python з іншими популярними мовами програмування: Java, C та C++;

- активна спільнота Python складається з мільйонів розробників з усього світу. У разі виникнення проблем співтовариство допоможе в їх вирішенні;
- в мережі є безліч корисних ресурсів для вивчення Python;
- Python можна переносити на різні операційні системи: Windows, MacOS, Linux і Unix.

Python корисний при написанні серверного коду, оскільки він пропонує безліч бібліотек, що складаються із попередньо написаного коду для складних серверних функцій. Також розробники використовують широкий спектр платформ Python, які надають всі необхідні інструменти для більш швидкого та простого створення інтернет-додатків. Наприклад, розробники можуть створити «скелет» інтернет-програми за лічені секунди, тому що їм не потрібно писати код з нуля. Потім можна протестувати за допомогою інструментів тестування платформи незалежно від зовнішніх інструментів тестування.

FastAPI – асинхронний фреймворк, що працює з допомогою ASGI – серверу [12]. Фреймворк побудовано на базі Starlette, що працює з ASGI – сервером, та Pydantic, який дозволяє проводити автоматичну валідацію вхідних даних. FastAPI дозволяє просто та оперативно написати невеликий REST API, не витративши на це багато зусиль. Саме це є основною причиною, з якою велика кількість розробників віддає перевагу саме цьому фреймворку.

ASGI є наступником WSGI, давнього стандарту Python для сумісності між веб-серверами, фреймворками та програмами. WSGI вдалося надати набагато більше свободи та інновацій у веб-просторі Python, і мета ASGI полягає в тому, щоб продовжити це з урахуванням асинхронності Python.

Для використання в продакшн мікросервіс та web-сервер, що обслуговує web-сторінку користувацького інтерфейсу буде запаковано у Docker – контейнер.

Також для розгортання проекту необхідні Nginx та Uvicorn.

					КПІ.ІП-9228.045420.02.81	Арк.
Змін.	Арк.	№ докум.	Підп.	Дата.		40

Docker - це платформа, яка дозволяє розробляти, доставляти та запускати контейнерні програми. Використовуючи Docker, можна створювати контейнери, автоматизувати їх запуск і розгортання, а також керувати їх життєвим циклом. Docker дозволяє запускати багато контейнерів на одній фізичній машині.

Контейнеризація схожа на віртуалізацію, але це дві різні концепції. У віртуалізації запускається повноцінний гостьовий операційна система зі своїм віртуальним обладнанням, що працює на гіпервізорі. В контейнеризації процес запуску відбувається безпосередньо з ядра головної операційної системи без віртуалізації обладнання. Це означає, що контейнеризований додаток працює тільки в межах основної операційної системи. Контейнери не віртуалізують обладнання, тому вони споживають менше ресурсів.

Контейнери є засобом стандартизації розгортання програм та їх відокремлення від загальної інфраструктури. Кожен екземпляр програми запускається в ізольованому середовищі, що не впливає на основну операційну систему.

HTTPD - Apache2 є найпоширенішим веб-сервером у лінукс-системах. Веб-сервери використовуються для надання веб-сторінок за запитами клієнтських комп'ютерів. Зазвичай клієнти використовують веб-браузери, такі як Firefox, Opera, Chromium або Mozilla, для запиту і перегляду веб-сторінок.

Користувачі вводять URL-адресу (Uniform Resource Locator), щоб вказати веб-сервер за його повністю кваліфікованим доменним ім'ям (FQDN) та шляхом до необхідного ресурсу. Наприклад, для перегляду домашньої сторінки веб-сайту Ubuntu користувачу потрібно ввести лише FQDN: www.ubuntu.com.

Uvicorn — це реалізація веб-сервера ASGI для Python.

2.3.2 Діаграма компонентів

Діаграма компонентів дозволяє визначити структуру програмних компонентів і відображає ролі, які виконують вихідний, бінарний і виконуваний код, а також встановлює залежності між ними.

Задачі при розробці діаграми компонентів:

- специфікація можливого варіанта системи;
- специфікація загальної структури вихідного коду системи;

Ця діаграма забезпечує послідовний перехід від логічного до фізичного подання системи як програмних компонентів. Деякі компоненти можуть бути присутніми тільки на етапі компіляції програмного коду, тоді як інші можуть існувати тільки на етапі виконання. Основними елементами діаграми є компоненти, інтерфейси та залежності між ними. Крім цього, на діаграмі можуть бути відображені ключові класи, які належать до компонентів.

Компонент (англ. component) представляє фізичну частину системи. Вони можуть бути представлені як файли з вихідним кодом класів, бібліотеки, модулі тощо, і мають узгоджений набір інтерфейсів. Для графічного представлення компонентів використовуються стандартизовані графічні символи. Діаграму компонентів додатку представлено на рисунку 2.7.

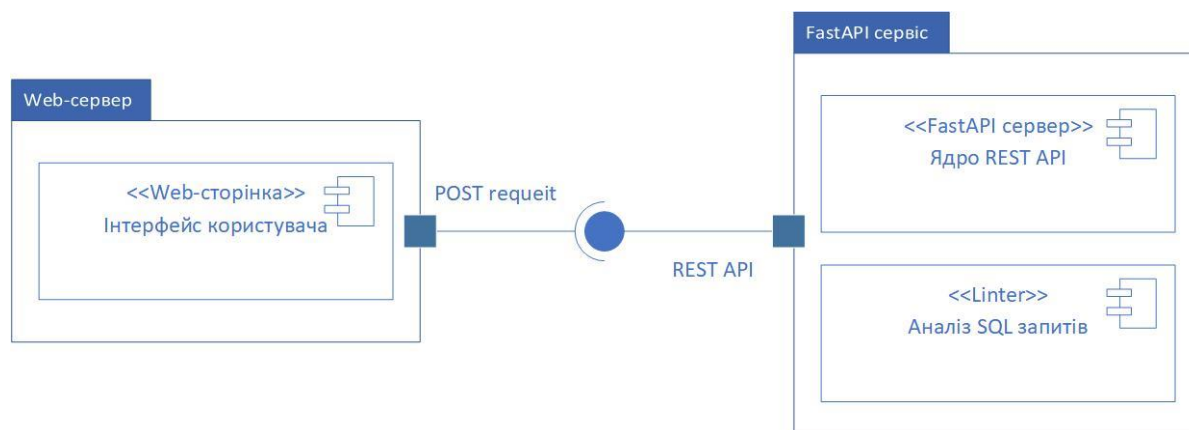


Рисунок 2.7 – Діаграма компонентів

Система складається з двох компонентів. Перший з компонентів – web – сервер, що відповідає за відтворення web-сторінки, яка є інтерфейсом користувача. Другий компонент – REST API, що поділена на два модулі – ядро

API та предметний модуль, що відповідає за синтаксичний та стилістичний контроль SQL – запитів.

Контрактом між цими компонентами є POST – запит, що містить текст запиту SQL, має бути оцінений та відповідь API формату JSON [11].

2.4 Реалізація проекту системи

2.4.1 Реалізація інтерфейсу користувача

Реалізований інтерфейс користувача показано на рисунку 2.8. Інтерфейс користувача являє собою web – сторінку, створену з використанням HTML5 та CSS3. Основне місце в HTML документі займає форма, що використовується для надсилання запиту до API. Форма є поєднанням екземпляру textarea та двох кнопок, однієї з яких призначено функціонал submit -надсилання даних, іншій – reset – очищення форми.

Безпосередньо надсилання запиту до REST API реалізовано з використанням Fetch API JavaScript. Для виконання js – скрипту при натисканні кнопки «Відправити» додано прослуховувач події натискання кнопки.

SQLint

Синтаксичний та стилістичний аналіз SQL запитів

```
SELECT *
      FROM
      TableA AS a      -- Name = 'Doe, John'
INNER JOIN
      TableB AS b -- Name = 'Doe.John'
      on /* */ a.id = b.id
WHERE
      LEFT(a.name, 1) = 'n' AND
      LEFT(a.name, 1) = 'z';
```

☐ Форматувати запит

Відправити

Очистити

Тут буде результат аналізу SQL запиту

Рисунок 2.8 – Web – сторінка інтерфейсу користувача

Fetch API дозволяє надсилати асинхронний запит при отриманні відгуку на який відбувається заповнення відповідного textarea на HTML сторінці. Беручи до уваги використанням базової авторизації на створеному API до заголовків запиту додано “Authorization”, тип контенту – “application/json”.

Надсилення запиту з використанням Fetch API загорнуто в асинхронний метод getHints(text), де в якості аргументу передається текст SQL запиту, що має бути проаналізований.

Після отримання результату запиту отримуємо JSON - тіло відгуку та формуємо з отриманого JSON (Map) текст з результатами аналізу запиту, що показаний на рисунку 2.8 в нижньому текстовому багаторядковому полі.

Якщо обрано поле «Форматувати запит», в запиті до REST API проставляється true у прапорці форматування запиту й форматований запит

надходить також і відповіді та відображається у тому ж полі, що й результати аналізу.

2.4.2 Реалізація REST API

REST API, відповідно попередньому розділу, реалізовано з використанням Fast API. Увесь необхідний функціонал міститься в модулі `app.server`.

Створюємо екземпляр класу `FastAPI()` додаємо `CORSMiddleware` для задоволення вимог протоколу безпеки та створюємо два енд – поінти, один з яких використовується під час налаштування системи, а інший – безпосередньо для роботи з текстом SQL запитів.

```
@app.get("/")
@app.post("/get_hints/", response_model=Response)
```

FastAPI, так само як інші фреймворки, реалізує модель MVC. Отже, для функціонування REST API створюється певна кількість моделей, що успадковані від `Base` або `BaseModel` (`pydantic` - перевірка даних і керування налаштуваннями за допомогою анотацій типу Python) [15. 16].

`Credentials` – модель, що є відображення даних автентифікації (пара логін-пароль).

`User` – модель, що використовується для відтворення користувача при роботі з БД.

`Hint` – модель, що є відображення окремого елементу результатів аналізу SQL – запиту.

`Response` – модель відгуку основного енд-поінту `"/get_hints"`, складається з двох полів: перше поле – текст SQL – запиту, що був проаналізований; друге поле – список об’єктів типу `Hints`. Цей перелік об’єктів є відтворенням результатів аналізу SQL – запиту й використовується на стороні web-сторінки для формування текстового поля результатів аналізу.

Text – модель, що складається з одного поля value. Використовується для обробки тіла POST запиту, що надходить на основний енд – поінт створеного REST API.

При надсиланні GET запиту на енд-поінт “/” REST API повертає JSON наступного складу {"health_check": "ok"}. Такий енд – поінт використовується для перевірки робочого стану сервісу прости GET – запитом без параметрів під час розробки та тестування системи. Також подібний енд – поінт є необхідним при деплойменті, наприклад, на DigitalOcean, ди відбувається системна перевірка стану сервісу (health – check).

При надсиланні POST запиту з текстом SQL-запиту для перевірки на енд – поінт “/get_hints” REST API повертає JSON, приклад якого наведено нижче.

```
{ 'initial_text': "SELECT *\n\nFROM\n\nTableA AS a          -- Name = 'Doe, John'\n\nINNER JOIN\n\nTableB AS b -- Name = 'Doe.John'\n\non /* */ a.id = b.id\nWHERE\n\nLEFT(a.name, 1) = 'n' AND\n\nLEFT(a.name, 1) = 'z';",\n\n'hints': {1: 'SELECT <- Кореневе слово не на власному\nрядку.',\n\n2: 'FROM <- Кореневе слово не вирівняно вліво.',\n\n6: 'on <- Зарезервоване слово в нижньому регістрі.'}}}
```

В hints числовий ключ означає номер рядка, до якого відноситься hint (зауваження, пропозиція, підказка), а текстове значення – текст цієї підказки.

2.5 Аналіз безпеки даних

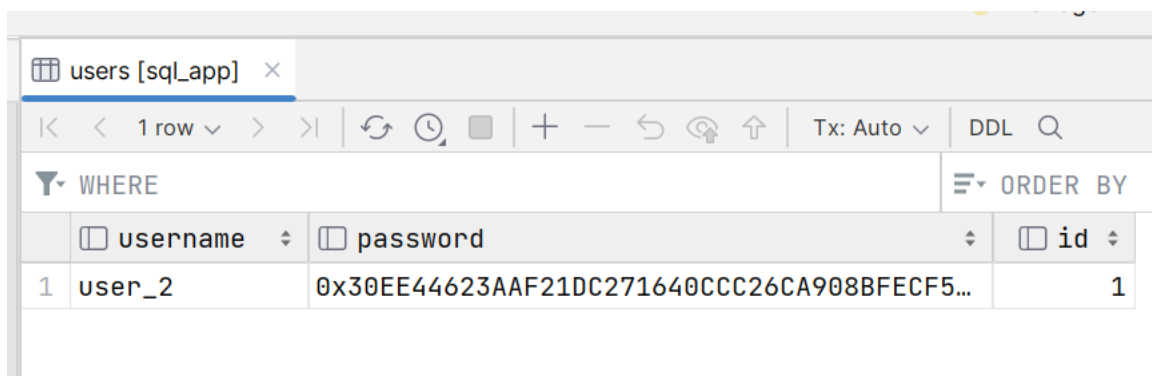
В REST API реалізовано базовий протокол автентифікації користувача за парою логін/пароль. Дані користувачів зберігаються в SQLite базі даних серверного додатку в зашифрованому виді.

Шифрування реалізовано класом PasswordEncryptDecrypt, що складається із статичних методів encrypt() та decrypt(). Для шифрування використовується алгоритм 'SHA256' [14], імплементація алгоритму – бібліотека hashlib.

Перевірка логіну/пароллю виконується методом check_credentials() основного модулю сервера (fast_api_server.py). У випадку некоректних даних автентифікація користувача повертається HTTP помилка 401. При вдалій автентифікації користувача відбувається обробка надісланого тексту SQL запиту та повертається відгук з кодом 200.

Для взаємодії з БД (читання/запис даних автентифікації користувачів) використано sqlalchemy.org, що є стандартним підходом при використанні FastAPI.

Як показано на рисунку 2.9, пароль користувача зберігається в БД в зашифрованому вигляді й розшифровується безпосередньо перед перевіркою надісланих на енд-поінт даних автентифікації.



	username	password	id
1	user_2	0x30EE44623AAF21DC271640CCC26CA908BFECF5...	1

Рисунок 2.9 – Зберігання даних користувачів

Висновки до розділу

У другому розділі було розглянуто процес моделювання, аналізу та конструювання програмного забезпечення для перевірки якості написання SQL запитів. Визначено функціональні та нефункціональні вимоги до програми, а також розроблено архітектуру та структуру програмного

забезпечення, враховуючи особливості контролю безпеки даних та різні види аналізу запитів.

В ході аналізу було прийнято рішення web-сторінку інтерфейсу користувача побудувати з використанням загально прийнятої зв'язки HTML5 + CSS3, відправку запиту та візуалізацію результатів реалізувати з використанням Java Script. Також вирішено реалізувати API з використанням Docker щоб в подальшому безпроблемно адаптувати програмне забезпечення для роботи з мобільними та іншими платформами

Додатково в REST API була реалізована можливість авторизації та аутентифікації з шифруванням даних користувача та зберіганням їх у базі даних застосунку.

З огляду на моделювання та конструювання програмного забезпечення, успішно розроблено архітектуру та реалізовано функціонал, який задовольняє вимогам, сформульованим в розділі 1. У подальших розділах будуть розглянуті питання тестування програмного забезпечення та інші кроки, до впровадження ПЗ та його технічної підтримки.

					КПІ.ІП-9228.045420.02.81	Арк.
Змін.	Арк.	№ докум.	Підп.	Дата.		48

3 АНАЛІЗ ЯКОСТІ ТА ТЕСТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

3.1 Аналіз якості ПЗ

Під терміном "якість програмного забезпечення" ми розуміємо сукупність властивостей, які визначають здатність програмного забезпечення задовольняти вимоги та запити замовника, які висловлені як вимоги до розробки.

Поняття якості програмного забезпечення є відносним і залежить від реальних умов його застосування. Тому вимоги до якості програмного забезпечення встановлюються з урахуванням конкретних умов та вимог галузі, в якій воно використовується.

В програмній інженерії аналіз якості спрямований на наступні аспекти:

- досягнення необхідного рівня якості програмного забезпечення відповідно до встановлених критеріїв;
- верифікацію та валідацію програмного забезпечення на різних етапах його життєвого циклу, а також оцінку якості розробленого продукту;
- забезпечення надійності як головної характеристики, яка гарантує якість програмного забезпечення (QA - Гарантія якості програмного забезпечення);

Аналіз та забезпечення якості програмного забезпечення враховуються на всіх етапах його життєвого циклу, включаючи планування, розробку та підтримку процесів створення програмного продукту.

В рамках цієї роботи обмежимося першим з перелічених пунктів – оцінкою ПЗ за певними критеріями (метриками).

Критерії, що можуть бути застосовані до розробленого програмного забезпечення, поділимо на три категорії:

- критерії оцінки алгоритму аналізу SQL запитів;
- критерії оцінки розробленої REST API;

– критерії оцінки розгорнутого сервісу, в тому числі швидкість та безвідмовність роботи.

Алгоритм аналізу SQL запитів оцінюємо на послідовності SQL – запитів, що аналізуються. Для тесту формуємо послідовність з 20 запитів, що будуть проаналізовані. Для кожного запиту визначаємо кількість сформованих алгоритмом рекомендацій. Вручну визначаємо для кожного тесту всі рекомендації, що повинні бути сформовані.

Таким чином, отримуємо певну адаптацію метрики Accurasy, що широко використовується, наприклад, в машинному навчанні. Зважене середнє по усім запитах, що аналізується, дозволяє знайти єдину числову метрику.

Для підрахунку кількості визначених рекомендацій для кожного SQL – запиту створюємо окремий функціональний тест, загальну кількість рекомендацій до кожного SQL – запиту також підраховуємо вручну. Результати тестування для обрахування метрики наведені в таблиці 3.1.

Таблиця 3.1 – Розрахунок метрики “Accurasy”

		1	2	3	4	5	6	7		8
Кількість рекомендацій	Визначено	4	4	6	7	3	5	7		4
	Загальна	4	5	8	8	4	6	7		8
		1.0	0.8	0.75	0.875	0.75	0.83	1.0		0.5
		9	10	11	12	13	14	15		16
Кількість рекомендацій	Визначено	4	6	5	5	6	8	8		3
	Загальна	5	7	6	7	8	8	9		4
		0.8	0.86	0.83	0.71	0.75	1.0	0.89		0.75
		17	18	19	20					
Кількість рекомендацій	Визначено	5	5	6	3					
	Загальна	6	7	8	4					
		0.83	0.71	0.75	0.75					

Шляхом розрахунку зваженого середнього отримуємо

$$Accuracy = 0.81$$

Розроблена REST API коректно проводить авторизацію та аутентифікацію користувачів, обидва end-point працюють коректно, відповідаючи JSON очікуваного формату.

Середній час відповіді віддаленого серверу (деплоймент на DigitalOcean) складає не більше 300мс, що є прийнятним часом відгуку. Час безперервної роботи віддаленого серверу (uptime) згідно логів склав більше тижня без перезавантажень чи будь-яких збоїв в роботі.

На підставі цього можна зробити висновок про нормальну роботи віддаленого серверу.

Таким чином, по кожній критеріїв розроблене ПЗ отримує позитивні оцінки:

- значення метрики $Accuracy = 0.81$ вважається хорошим показником ($0.65 < Accuracy < 0.8$ – задовільно, $Accuracy \geq 0.9$ - відмінно);
- робота REST API відповідає очікуванням;
- час відгуку та безвідмовність серверу на нормальному рівні.

3.2 Опис процесів тестування

Процес тестування було поділено на модульне та функціональне тестування.

Модульне тестування проводилось стандартними інструментами Python – unittest. В процесі розробки було використано три групи модульних тестів:

- тестування REST API;
- тестування шифрування/дешифрування пароллю користувача, що зберігається в базі даних;
- тестування безпосередньо алгоритму літера.

Тестування REST API складається з тестування моделей, тестування коректного читання та запису облікових даних користувача та тестування

					КП.ІП-9228.045420.02.81	Арк.
Змін.	Арк.	№ докум.	Підп.	Дата.		51

коректності обміну даних між модулем серверу та модулем лінтеру згідно визначеного контракту.

Основне в тестуванні моделей – коректне читання вхідного JSON та коректне формуванні відгуку API.

Тестування контракту серверу та лінтеру є аналогічним – визначає коректність даних, що передаються лінтеру та його відгуку, що потім трансформується у відгук REST API.

Тестування криптографічної складової має на увазі визначення незмінності початкового значення після послідовного шифрування та зберігання після яких відбувається зворотній процес – читання зашифрованого значення та дешифрування. Криптографічний процес було перевірено на різних послідовностях символів, що дозволені для використання в паролі: великі та маленькі англійські та кириличні літери, цифри, розділові знаки.

Нарешті тестування алгоритму лінтера відбувалося шляхом порівняння очікуваних та фактичних результатів роботи кожного з його компонентів (класів – нащадків SingleLint). Для тестування використовувався той самий набір текстів SQL запитів, що було використано в попередньому підрозділі для оцінки створеного ПЗ за метриками, в тому числі Ассурасу.

Після модульного тестування, що проводиться, як правило під час розробки та для попередньої оцінки оновлень алгоритмів ПЗ, що розробляється, переходимо до функціонального тестування системи.

Функціональне тестування розробленого програмного інтерфейсу проводимо в два способи: за допомогою спеціалізованого програмного забезпечення Postman та шляхом використання розробленого користувацького інтерфейсу (web – сторінки).

Postman - це інструмент для роботи з API, який дозволяє тестувальнику надсилати запити до сервісів та працювати з їхніми відповідями. З його допомогою можна протестувати бекенд та переконатися, що він коректно працює.

					КПІ.ІП-9228.045420.02.81	Арк.
Змін.	Арк.	№ докум.	Підп.	Дата.		52

Postman вважається найбільш популярним серед подібних інструментів та має низку переваг, серед яких:

- інтуїтивна зрозумілість й простота у використанні, не вимагає якогось складного налаштування чи знання мов програмування;
- безкоштовність;
- підтримка різних API (REST, SOAP, GraphQL);
- можливість розширення під будь-які потреби за допомогою Postman API;
- легко інтеграція в CI/CD за допомогою Newman – консольної утиліти для запуску тестів;
- запуск на будь-яких ОС;
- підтримка ручне та автоматизоване тестування;
- велике ком'юніті, де можна знайти відповіді на будь-які питання.

При тестуванні створеного програмного забезпечення основна корисність Postman у порівнянні з використанням створеного фронтенду полягає в тому, що в робочій області Postman можна сформулювати та записати певну кількість запитів з використанням яких відбувається тестування відгуків створеної REST API. Ці запити включають як заголовки (в тому числі дані авторизації), так й основну частину – текст SQL запиту, що має бути проаналізований.

Після тестування відгуків REST API з використанням Postman переходимо до повноцінного інтеграційного тестування створеного програмного забезпечення.

Оскільки створене програмне забезпечення має, фактично, єдиний сценарій використання, то застосовуємо відповідно єдиний інтеграційний тест, що імітує послідовність використання сервісу. Сценарій тесту наведено в таблиці 3.2. Другий з тестів створено для перевірки повної функціональності створеної форми на сторінці інтерфейсу користувача.

Таблиця 3.3 – Зміст (послідовність) тесту 1

					КПІ.ІП-9228.045420.02.81	Арк.
Змін.	Арк.	№ докум.	Підп.	Дата.		53

Тест	Аналіз SQL запиту.
Модуль	Програмне забезпечення в цілому.
Номер тесту	1
Початковий стан системи	Користувач знаходиться на єдиній сторінці web – інтерфейсу.
Вхідні данні	Текст SQL – запиту, що має бути проаналізований.
Опис проведення тесту	Користувач вставляє (створює) текст SQL – запиту, що має бути проаналізований у відповідне поле користувацького інтерфейсу (див. попередні розділи). Після натискання кнопки «Відправити» відбувається аналіз SQL – запиту та вивід результатів аналізу у відповідному полі форми.
Очікуваний результат	Після натискання кнопки інтерфейсу «Відправити» у відповідному полі форми відображається очікуваний результат аналізу SQL – запиту. Під «очікуваний» розуміємо співпадіння фактично отриманого тексту з тим, що має бути.
Фактичний результат	Після натискання кнопки інтерфейсу «Відправити» у відповідному полі форми відображається очікуваний результат аналізу SQL – запиту.

Другий з тестів призначений для перевірки коректності очищення форми web –інтерфейсу користувача.

Таблиця 3.4 – Зміст (послідовність) тесту 2

Тест	Очищення форми користувацького інтерфейсу.
Модуль	Користувацький інтерфейс (web – сторінка).
Номер тесту	1
Початковий стан системи	Користувач знаходиться на єдиній сторінці web – інтерфейсу.

Вхідні данні	Текст SQL – запиту, що був проаналізований в попередньому тесті. У полі відповіді знаходиться результат аналізу (текст).
Опис проведення тесту	Користувач натискає кнопку «Очистити» після виконання деякого запиту до системи й отримання відгуку – обидва поля форми web – інтерфейсу користувача непорожні.
Очікуваний результат	Після натискання кнопки інтерфейсу «Очистити» обидва поля форми повертаються до початкового стану.
Фактичний результат	Після натискання кнопки інтерфейсу «Очистити» обидва поля форми повертаються до початкового стану.

На підставі вищезазначеного можна вважати інтеграційне тестування створеного програмного забезпечення успішно виконаним.

3.3 Опис контрольного прикладу

Контрольним прикладом є використанням створеного програмного забезпечення для аналізу одного з SQL – запитів, що були використані в попередніх тестах.

Початковий вигляд сторінки web – інтерфейсу показано на рисунку 3.1.

SQLint

Синтаксичний та стилістичний аналіз SQL запитів

Введіть текст SQL запиту.

☐ Форматувати запит

Відправити

Очистити

Тут буде результат аналізу SQL запиту

Рисунок 3.1 – Початковий стан web -сторінки користувацького інтерфейсу

На першому кроці користувач вводить текст запиту, що має бути проаналізований, до відповідного поля форми. Після чого форма набуває вигляду, що показаний на рисунку 3.2.

SQLint

Синтаксичний та стилістичний аналіз SQL запитів

```
SELECT *
  FROM
    TableA AS a      -- Name = 'Doe, John'
 INNER JOIN
    TableB AS b -- Name = 'Doe.John'
  on /* */ a.id = b.id
WHERE
  LEFT(a.name, 1) = 'n' AND
  LEFT(a.name, 1) = 'z';
```

☒ Форматувати запит

Відправити

Очистити

Тут буде результат аналізу SQL запиту

Рисунок 3.2 – Форма з введеним запитом

Після введення тексту запиту, що має бути проаналізований, користувач натискає кнопку «Відправити» та отримує результати аналізу SQL – запиту, що був надісланий. Інтерфейс користувача набуває вигляду, що показаний на рисунку 3.3.

SQLint

Синтаксичний та стилістичний аналіз SQL запитів

```
SELECT *
FROM
  TableA AS a      -- Name = 'Doe, John'
INNER JOIN
  TableB AS b -- Name = 'Doe.John'
on /* */ a.id = b.id
WHERE
  LEFT(a.name, 1) = 'n' AND
  LEFT(a.name, 1) = 'z';
```

☒ Форматувати запит

Відправити

Очистити

```
Рядок 6 ->on<-Зарезервоване слово в нижньому регістрі
Рядок 8 ->AND<-Логічний оператор AND не на початку рядка.

--- --- --- Форматований SQL запит --- --- ---

SELECT
  *
FROM  tablea as a -- Name = 'Doe, John'
      INNER JOIN tableb as b -- Name = 'Doe.John'
      ON a.id = b.id /* */
```

Рисунок 3.3 – Форма з отриманим результатом аналізу

Після натискання кнопки «Очистити» форма набуває початкового вигляду, що показаний на рисунку 3.1.

Висновки до розділу

У поточному розділі було виконано визначення метрик для аналізу програмного забезпечення та аналіз програмного забезпечення з використанням цих метрик.

Після аналізу програмного забезпечення обґрунтовано процес та виконане модульне та функціональне тестування програмного забезпечення.

Процес тестування та оцінки програмного забезпечення закінчено розглядом контрольного прикладу, що демонструє роботу розробленого програмного забезпечення.

По результатах тестування програмного забезпечення можна зробити наступні висновки:

- розроблене програмне забезпечення виділяє біля 80% з загальної кількості рекомендацій, що можуть бути надані під час аналізу відповідності стилю та синтаксису SQL- запиту вимогам, сформульованим Mozilla;
- розроблений програмний інтерфейс (REST API) задовольняє загально визнаним критеріям швидкості та стабільності роботи;
- під час проведення інтеграційних тестів поведінка розробленого програмного забезпечення співпадала з очікуваною;
- розроблене програмне забезпечення відповідає функціональним та нефункціональним вимогам, сформульованим в першому розділі.

4 ВПРОВАДЖЕННЯ ТА СУПРОВІД ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

4.1 Розгортання програмного забезпечення

Діаграма розгортання, що входить до типу UML-діаграм, відображає структуру виконання системи, включаючи апаратні та програмні середовища виконання, а також засоби, що їх об'єднують.

Діаграми розгортання широко використовуються для візуалізації фізичної архітектури апаратного та програмного забезпечення системи. За допомогою цих діаграм можна зрозуміти, як система буде розгорнута фізично на різних апаратних пристроях.

Діаграму розгортання системи, що будується, показано на рисунку 4.1.

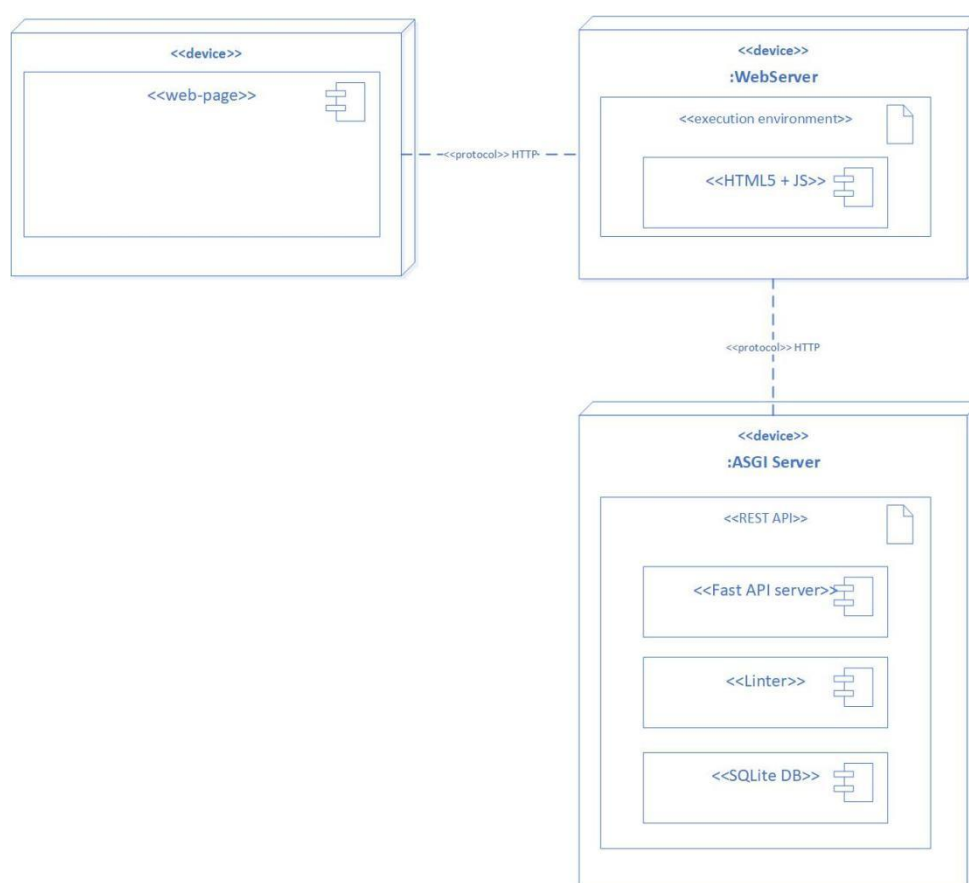


Рисунок 4.1 – Діаграма розгортання

Згідно рекомендаціям з використання Docker, REST API сервіс та HTTP сервер Apache2 розгортаються у двох незалежних контейнерах. При

тестуванні програмного забезпечення на локальному комп'ютері розробника контейнери поєднуються за допомогою Docker compose v2, між ними встановлюється мережеве з'єднання – network bridge. При деплоїменті на продуктивний сервер (production) сервіси також розгортаються незалежно, комунікуючи між собою з використанням внутрішнього простору IP адрес.

За HTTP – протоколом web – браузер користувача веде обмін даними з HTTP – сервером, що відповідає за функціонування web – сторінки інтерфейсу користувача. Між собою сервіси, розгорнуті в незалежних контейнерах, взаємодіють на підставі контракту, що був описаний вище, в рамках network bridge.

4.2 Підтримка програмного забезпечення

Обидва компоненти системи розгортаються на сервері, тому проблему оновлення, подібної до користувацьких додатків, не виникає.

Оновлення компонентів розробленого програмного забезпечення на сервері буде виконуватись з використанням DevOps практик безперервної інтеграції та безперервного постачання.

Безперервна інтеграція (Continuous Integration, CI) та безперервне постачання (Continuous Delivery, CD) є принципами та практиками, спрямованими на постійне та надійне розгортання змін у програмному забезпеченні.

CI є методологією розробки, де вносяться невеликі зміни в код з частими комітами. Мета CI полягає в послідовному та автоматизованому процесі збирання, пакування та тестування додатків. Використання CI сприяє покращенню комунікації та підвищенню якості програмного забезпечення.

CD починається там, де закінчується CI і автоматизує процес розгортання додатків у різних середовищах. Це означає, що розробники можуть працювати з продакшн-середовищем, а також з середовищами розробки та тестування. Інструменти CI/CD допомагають налаштовувати

					КПІ.ІП-9228.045420.02.81	Арк.
Змін.	Арк.	№ докум.	Підп.	Дата.		61

середовища та виконувати автоматичні запити до серверів та сервісів під час розгортання.

CI/CD потребують постійного тестування для розробки якісного програмного забезпечення. Автоматизовані тести, такі як регресійні тести та тести продуктивності, виконуються у CI/CD-конвеєрі. У зрілій практиці CI/CD може включати безперервне розгортання, коли успішні зміни автоматично розгортаються у продакшн-середовищі.

З огляду на структуру та зміст програмного забезпечення, можна стверджувати, що REST API схильне до змін та оновлень, в той час як веб-сторінки, ймовірно, матимуть меншу частоту оновлень.

Отже, для організації спрощеного CI/CD конвеєру визначаємо, що оновлення гілки main означає запуск CI/CD конвеєру з метою оновлення версії ПЗ на продакшн сервері. Окрім того використовуємо гілку dev, яка призначена для послідовних комітів у процесі розробки. Елементами CI/CD конвеєру є модульні та функціональні тести, що були розглянуті в розділі 3. Після успішного проходження тестів збирається образ REST API та відбувається його деплоймент на сервер. В цьому спрощеному конвеєрі вилучено елементи, що пов'язані з QA (Quality Assurance) через те, що розробка та підтримка ПЗ, що розглядається, ведеться одним – двома розробниками й не має на увазі наявність окремого QA – спеціаліста.

Таким чином, конвеєр спрощено до трьох елементів: оновлення гілки main на GitHub, модульного та функціонального тестування додатку, що оновлюється, та завантаження оновленого образу на продуктовий сервер.

Висновки до розділу

В поточному розділі розглянуто процеси розгортання та оновлення програмного забезпечення, що було розроблено та протестоване в попередніх розділах.

					КПІ.ІП-9228.045420.02.81	Арк.
Змін.	Арк.	№ докум.	Підп.	Дата.		62

Розгортання програмного забезпечення виконується з використанням технології контейнеризації, що дозволяє усунути можливості прив'язки до конкретного пристрою, що відповідно допомагає уникати помилок, пов'язаних з налаштуваннями оточення в окремо взятій операційній системі певного комп'ютера. Тобто, усуваються проблеми можливої відсутності певних залежностей, змінних оточення, шляхів, тощо.

Необхідне оновлення програмного забезпечення на сервері відбувається по мірі його вдосконалення розробниками з використанням CI/CD конвеєру. Це дозволяє використовувати налаштований єдиний раз конвеєр оновлення ПЗ на сервері. Дії розробника при кожному оновленні обмежуються завантаження оновленої та протестованої версії програмного забезпечення у визначену гілку репозиторія проекту на GitHub.

Таким чином, розгортання та підтримка оновлень програмного забезпечення виконуються з використанням актуальних DevOps підходів та практик.

					КПІ.ІП-9228.045420.02.81	Арк.
Змін.	Арк.	№ докум.	Підп.	Дата.		63

ВИСНОВКИ

В рамках роботи створено синтаксичний та стилістичний аналізатор SQL – запитів. Додатковою функціональністю розробленого клієнт – серверного програмного забезпечення є форматування текст SQL запитів згідно формату, що розглядається - Mozilla SQL style guide.

В першому розділі роботи проведено аналіз предметної області, а саме вивчено функціонал та призначення програмного забезпечення, що має спеціалізовану назву лінтер. Досліджено та проаналізовано робочий функціонал існуючих програмних рішень, на базі чого виділено риси, що відсутні у існуючих програмних рішень. Список рис, що відсутні у існуючого програмного забезпечення, моделювання варіантів використання програмного забезпечення, що створюється, та розуміння сучасних підходів до побудови програмного забезпечення дозволили сформулювати функціональні та нефункціональні вимоги до програмного забезпечення, що є результатом роботи, що описує перший розділ.

Другий розділ роботи присвячено моделюванню та конструюванню програмного забезпечення. З використанням універсальної мови моделювання та проектування UML побудовані необхідні для покрокового створення програмного забезпечення діаграми. В якості мови реалізації проекту обрано Python, що є типовим рішенням для розробки серверних API та роботи з текстом, що по суті виконує лінтер. В якості фреймворку побудови API обрано FastAPI, як швидкий та ненавантажений асинхронний фреймворк. Під час обрання фреймворку виконано порівняння з типовими альтернативами – Flask та Django.

Для забезпечення безпеки створений REST API працює з виконанням базової авторизації з використанням пари логін – пароль, що надсилається в заголовок POST запиту до API. Паролі користувачів зберігаються в базі даних в зашифрованому з використанням SHA256 вигляді.

					КПІ.ІП-9228.045420.02.81	Арк.
Змін.	Арк.	№ докум.	Підп.	Дата.		64

В третьому розділі проведено аналіз якості програмного забезпечення та тестування створеного програмного забезпечення, розглядається контрольний приклад. Для аналізу якості програмного забезпечення, що створено, введено метрику Ассигасу, згідно якої програмне забезпечення в середньому формує 81% зауважень до синтаксису та стилістики SQL запитів з загальної кількості зауважень, що мали б бути сформовані на підставі Mozilla SQL style guide. Для обчислення метрики було проведено ручне маркування корпусу з двадцяти SQL – запитів. Тестування створеного програмного забезпечення виконувалося у конвенційний спосіб, з використанням модульних та функціональних тестів. Модульні тести забезпечують коректну поведінку окремих методів або класів, функціональні тести дозволили впевнитись в коректній поведінці додатку, що розроблено, в цілому. Порядок проведення функціональних тестів детально описаний в розділі, тексти unit – тестів наведені в додатках. Контрольний приклад також дозволив впевнитись в коректній поведінці додатку в усіх сценаріях використання.

В четвертому розділі розглянуто питання розгортання програмного забезпечення та його оновлення. Розгортання програмного забезпечення виконується на віддаленому сервері з використанням контейнеризації Docker, сервери REST API та HTTPS представлені окремими контейнерами згідно рекомендацій з використання Docker. Для оновлення програмного забезпечення на сервері запропоновано використання DevOps практик CI/CD.

Розроблене та створене програмне забезпечення відповідає сформульованим функціональним та нефункціональним вимогам та меті реалізації проекту, що були сформульовані в розділі 1. Згідно результатам тестування поведінка програмного забезпечення в усіх сценаріях використання відповідає очікуваній поведінці.

					КПІ.ІП-9228.045420.02.81	Арк.
						65
Змін.	Арк.	№ докум.	Підп.	Дата.		

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

- 1) Вивчаємо SQL. Генерація, вибірка та обробка даних, 3-є видання / Больє А. / 2021, 400 стор., іл.
- 2) Практичне керівництво по SQL , 3-є видання / Боуман С.Д., Емерсон С.Л., Дарновскі М. / Діалектика Вільямс – 2020, 336 стор., іл.
- 3) What Is a Linter? [Електронний ресурс]. Режим доступу до ресурсу: <https://www.testim.io/blog/what-is-a-linter-heres-a-definition-and-quick-start-guide/>
- 4) Аналіз і проектування інформаційних систем за допомогою UML 2.0 / Мацяшек Л. / Діалектика Вільямс – 2020, 816 стор.: іл.
- 5) Mozilla SQL style guide, [Електронний ресурс]. Режим доступу до ресурсу: https://docs.telemetry.mozilla.org/concepts/sql_style.html#sql-style-guide
- 6) Застосування UML 2.0 і шаблонів проектування, 3-є видання / Ларман К. / Діалектика Вільямс – 2020, 736 стор.: іл.
- 7) Введення в UML від творців мови / Буч Г., Рамбо Д., Якобсон І. / ДМК Пресс, 2017 – 496 стор.: іл.
- 8) HTML и CSS. Розробка та дизайн веб-сайтів. Дакетт Дж. / Видавництво Ексмо – 2013, 480 стор., іл.
- 9) Javascript та jQuery. Інтерактивна web-розробка. Дакетт Дж. / Видавництво Ексмо – 2019, 560 стор., іл.
- 10) Програмування мовою Python / Васильєв О/ Богдан, 2020, 504 стор.: іл.
- 11) JSON [Електронний ресурс]. Режим доступу до ресурсу: <https://uk.wikipedia.org/wiki/JSON>
- 12) FastAPI. [Електронний ресурс]. Режим доступу до ресурсу: <https://fastapi.tiangolo.com/>
- 13) SQLite [Електронний ресурс]. Режим доступу до ресурсу: <https://www.sqlite.org/docs.html>
- 14) SHA256 [Електронний ресурс]. Режим доступу до ресурсу: <https://www.simplilearn.com/tutorials/cyber-security-tutorial/sha-256-algorithm>

- 15) Pydantic [Електронний ресурс]. Режим доступу до ресурсу:
<https://docs.pydantic.dev/latest/>
- 16) Starlette [Електронний ресурс]. Режим доступу до ресурсу:
<https://www.starlette.io/>
- 17) Parsing Techniques - A Practical Guide, 2-е видання / Грун Д., Джейкобс С. / Springer US - 2008, 622 стор., іл.
- 18) LR Parsing: Theory and Practice / Чапман Н. П. / 1987, 228 стор.

Факультет інформатики та обчислювальної техніки

Кафедра інформатики та програмної інженерії

“ЗАТВЕРДЖЕНО”

Завідувач кафедри

_____ Едуард ЖАРІКОВ

“ ____ ” _____ 2023 р.

Програмне забезпечення для перевірки якості написання

SQL-запитів Текст програми

КП.ПІ-9228.045420.03.12

“ПОГОДЖЕНО”

Керівник проєкту:

_____ Юрій ОЛІЙНИК

Нормоконтроль:

_____ Катерина ЛІЩУК

Виконавець:

_____ Дарія ФРАДИНСЬКА

Київ – 2023

Файл requirements.txt

uvicorn>=0.21.1

SQLAlchemy>=2.0.7

pydantic>=1.10.7

fastapi>=0.95.0

requests>=2.28.2

dacite>=1.8.0

Файл boolean_at_line_beginning.py

```
import re
```

```
from typing import List
```

```
from .single_lint import SingleLint
```

```
from ..models import Lint
```

```
BOOLEANS = """\
```

```
AND
```

```
OR"""
```

```
BOOLEAN_LIST = BOOLEANS.split('\n')
```

```
class BooleanAtLineBeginning(SingleLint):
```

```
    @classmethod
```

```
    def process(cls, query: str) ->
```

```
        List[Lint]: output = []
```

```
        for i in range(len(query_list) :=
```

```
            query.split('\n')): line =
```

```
            query_list[i].rstrip()
```

```
            for bool_str in BOOLEAN_LIST:
```

```
                p = re.compile(bool_str)
```

```
                m = p.search(line)
```

					КП.ІП-9228.045420.03.12	Арк.
Змін.	Арк.	№ докум.	Підп.	Дата.		2

```

        # short circuit m.start() if m is None
        if m is not None and m.start() != 0:
            output.append(tuple([i + 1, m.group(), f'Логічний оператор
{m.group()} не на початку рядка.']))
    return [Lint(line=element[0], keyword=element[1], hint=element[2])
for element in output]

```

Файл code_blocks.py

```

import re
from typing import List

from .common import ROOT_WORD_LIST
from .common import remove_dashdash_comments, remove_multiline_comments,
remove_punctuation
from .single_lint import SingleLint
from ..models import Lint

class CodeBlocks(SingleLint):

    @classmethod
    def process(cls, query: str) -> List[Lint]:
        query = remove_multiline_comments(query)
        query = remove_dashdash_comments(query)
        query = remove_punctuation(query)
        output = []

        for i in range(len(query_list := query.split('\n'))):
            line = query_list[i].rstrip()
            for root_word in ROOT_WORD_LIST:
                m = re.search(root_word, line, re.IGNORECASE)
                # short circuit m.start() if m is None
                if m is not None and m.end() != len(line):

```

					КПІ.ІП-9228.045420.03.12	Арк.
Змін.	Арк.	№ докум.	Підп.	Дата.		3

```

        output.append(tuple([i + 1, m.group(), f'Кореневе слово
знаходиться не на окремому рядку'])))
    return [Lint(line=element[0], keyword=element[1], hint=element[2])
for element in output]

```

Файл **common.py**

```

import re

from typing import List

from .keywords import RESERVED_KEYWORDS

RESERVED_KEYWORD_ARRAY =
RESERVED_KEYWORDS.split('\n') RESERVED_KEYWORD_DICT =
{}
for keyword in RESERVED_KEYWORD_ARRAY:
    RESERVED_KEYWORD_DICT[keyword] = 1

ROOT_WORDS = """"\
FROM
WHERE
GROUP BY
HAVING
SELECT
ORDER BY
LIMIT""""

ROOT_WORD_LIST = ROOT_WORDS.split('\n')

def remove_dashdash_comments(query: str) -> str:
    query = query.split('\n')
    p = re.compile('--')
    for i in range(len(query)):
        if (m := p.search(query[i])): query[i] =
query[i][0:m.start()] return '\n'.join(query)

```

					КП.ІП-9228.045420.03.12	Арк.
Змін.	Арк.	№ докум.	Підп.	Дата.		4

```

def remove_multiline_comments(query: str) -> str:
    p = re.compile(r'\^*')
    p2 = re.compile(r'\^*\^/')
    query = query.split('\n')
    output_query = []
    in_comment = False
    for line in query:
        if not in_comment: m = p.search(line)
        else: m2 = p2.search(line)
        if not in_comment and m:
            in_comment = True
            if m3 := p2.search(line):
                in_comment = False
                output_query.append(line[0:m.start()] + line[(m.end() + 1):-
1]) else:
                output_query.append(line[0:m.start()])
        elif in_comment and not m2:
            continue
        elif in_comment and m2:
            end_of_comment_line = line[m2.end():-1]

            if end_of_comment_line != "":
                output_query.append()
            in_comment = False
        else:
            0

```

Файл left_align_root_words.py

```

import re
from typing import List

from .common import ROOT_WORD_LIST
from .single_lint import SingleLint

```

					КПІ.ІП-9228.045420.03.12	Арк.
Змін.	Арк.	№ докум.	Підп.	Дата.		5

```
from ..models import Lint
```

```
class LeftAlignKeywords(SingleLint):
```

```
    @classmethod
```

```
    def process(cls, query: str) -> List[Lint]:
```

```
        output = []
```

```
        for i in range(len(query_list := query.split('\n'))):
```

```
            for word in ROOT_WORD_LIST:
```

```
                if m := re.search(word, query_list[i], re.IGNORECASE):
```

```
                    if m.start() != 0:
```

```
                        output.append(tuple([i + 1, m.group(), 'Відсутнє  
вирівнювання вліво для кореневого слова']))
```

```
            return [Lint(line=element[0], keyword=element[1], hint=element[2])  
                    for element in output]
```

Файл **linter.py**

```
from typing import List
```

```
from .boolean_at_line_beginning import
```

```
BooleanAtLineBeginning from .code_blocks import CodeBlocks
```

```
from .left_align_root_words import
```

```
LeftAlignKeywords from .reserved_words import
```

```
ReservedWords
```

```
from ..models import Hint
```

```
class Linter:
```

```
    @staticmethod
```

```
    def process(query_text: str) -> List[Hint]:
```

```
        lint_list = ReservedWords.process(query_text)
```

```
        for processor in [LeftAlignKeywords,  
                          BooleanAtLineBeginning, CodeBlocks]:
```

```
            lint_list.extend(processor.process(query_text))
```

```
        sorted_reserved_words_list = sorted(lint_list, key=lambda x:  
        x.line) hints_for_response = []
```

```
        for lint in sorted_reserved_words_list:
```

					КПІ.ІП-9228.045420.03.12	Арк.
Змін.	Арк.	№ докум.	Підп.	Дата.		6

```

hints_for_response.append(Hint(row_number=lint.line, hint_text=f'-
    >{lint.keyword}<-{lint.hint}'))
return hints_for_response

```

Файл reserved_words.py

```

import re

from typing import List

from .common import (remove_multiline_comments,
remove_dashdash_comments, remove_punctuation,
                    remove_spacing, find_lowercase_sql_keywords)
from .single_lint import SingleLint
from ..models import Lint

class ReservedWords(SingleLint):

    @classmethod
    def process(cls, query: str) -> List[Lint]:
        output = []
        original_query = query
        query = remove_multiline_comments(query)
        query = remove_dashdash_comments(query)
        query = remove_punctuation(query)
        query = remove_spacing(query)
        lowercase_keywords =
        find_lowercase_sql_keywords(query)
        set_lowercase_keywords = set(lowercase_keywords)
        for keyword in set_lowercase_keywords:
            p = re.compile(r'\b' + keyword + r'\b')
            m = p.finditer(original_query)
            for i in m:
                p2 = re.compile("\n")

```

					КПІ.ІП-9228.045420.03.12	Арк.
Змін.	Арк.	№ докум.	Підп.	Дата.		7

```

        m2 = p2.findall(original_query[0:i.start()])
        line_number = len(m2) + 1
        output.append(tuple([line_number, i.group(), 'Зарезервоване слово
в нижньому регістрі']))
    return [Lint(line=element[0], keyword=element[1], hint=element[2])
for element in output]

```

Файл **single_lint.py**

```

from abc import ABC
from typing import List

from ..models import Lint

class SingleLint(ABC):

    @classmethod
    def process(cls, query_text: str) ->
        List[Lint]: pass

```

Файл **models.py**

```

from dataclasses import dataclass
from typing import List, Dict, Any

from pydantic import BaseModel
from sqlalchemy import Column, Integer,

String from .server.database import Base

@dataclass
class Lint:

    line: int

```

keyword: str

hint: str

```
class Credentials(Base):
```

```
    __tablename__ = "users"
```

```
    id = Column(Integer, primary_key=True,  
index=True) username = Column(String)
```

```
    password = Column(String)
```

```
class User(BaseModel):
```

```
    id: int
```

```
    name: str
```

```
    is_active: bool
```

```
class Config:
```

```
    orm_mode = True
```

```
class Hint(BaseModel):
```

```
    row_number: int
```

```
    hint_text: str
```

```
class Response(BaseModel):
```

```
    initial_text: str
```

```
    hints: Dict[int, str]
```

```
    formatted_request: str | None
```

```
@staticmethod
```

```
def get_response_from_values(initial_text: str,
```

					КПІ.ІП-9228.045420.03.12	Арк.
Змін.	Арк.	№ докум.	Підп.	Дата.		9

```

        response_list: List[Hint],
        formatted_sql_request: str) -> "Response":
    return Response(initial_text=initial_text,
                    hints={hint.row_number: hint.hint_text for hint in
                          response_list}, formatted_request=formatted_sql_request)

```

```
class Text(BaseModel):
```

```
    value: str
```

```
    formatText: bool
```

Файл fast_api_server.py

```
import logging
```

```
from fastapi import FastAPI, Depends, HTTPException
```

```
from fastapi.middleware.cors import CORSMiddleware
```

```
from fastapi.security import HTTPBasic,
HTTPBasicCredentials from sqlalchemy.orm import Session
```

```
from . import crud
```

```
from .database import SessionLocal, engine
```

```
from .pass_enc_dec import PasswordEncryptDecrypt
```

```
from .. import models
```

```
from ..linter.linter import Linter
```

```
from ..models import Response, Text
```

```
from sql_formatter.core import format_sql
```

```
models.Base.metadata.create_all(bind=engine)
```

```
app = FastAPI()
```

```
app.add_middleware(
```

```
    CORSMiddleware,
```

					КПІ.ІП-9228.045420.03.12	Арк.
Змін.	Арк.	№ докум.	Підп.	Дата.		10

```

        allow_origins=["*"],
        allow_credentials=True,
        allow_methods=["*"],
        allow_headers=["*"],
    )

    security = HTTPBasic()
    logger = logging.getLogger('main_logger')

def get_db():
    db = SessionLocal()
    try:
        yield db
    finally:
        db.close()

def check_credentials(db, credentials):
    user = crud.get_user(db, credentials.username)
    if user is None:
        raise HTTPException(status_code=401, detail=f"{credentials.username} is
not valid user")
    else:
        check_result =
PasswordEncryptDecrypt.check(credentials.password, user.password)

        if not check_result:
            raise HTTPException(status_code=401, detail="Not valid password")

@app.get("/")
async def health_check():
    return {"health_check": "ok"}

```

					КП.ІП-9228.045420.03.12	Арк.
Змін.	Арк.	№ докум.	Підп.	Дата.		11

```
@app.post("/get_hints/", response_model=Response)
async def get_skills_with_text(db: Session = Depends(get_db), credentials:
    HTTPBasicCredentials = Depends(security),
    text: Text | None = None):

    check_credentials(db, credentials)

    if text:
        linter_result = Linter.process(text.value)
        return Response.get_response_from_values(text.value,
            linter_result,
            format_sql(text.value) if text.formatText else None)
    else:
        return None
```

Файл test_linter.py

```
from unittest import TestCase

from rest_api.application.linter.linter import Linter

class TestLinter(TestCase):

    def setUp(self) -> None:
        self.sample_text = """
        SELECT *
        FROM
            TableA AS a -- Name = 'Doe,
John' INNER JOIN
            TableB AS b -- Name = 'Doe.John'
        on /* */ a.id = b.id
        WHERE
            LEFT(a.name, 1) = 'n' AND
            LEFT(a.name, 1) = 'z';"""
```

```
def test_linter(self):
    response =
    Linter.process(self.sample_text) pass
```

Файл pass_enc_dec.py

```
import hashlib
import os
```

```
class PasswordEncryptDecrypt:
```

```
@classmethod
```

```
def encrypt(cls, data: str) -> bytes:
```

```
    salt = os.urandom(32)
```

```
    encrypted = cls._encrypt_with_salt(data,
    salt) return salt + encrypted
```

```
@classmethod
```

```
def check(cls, data: str, encrypted: bytes) -> bool:
```

```
    salt_from_storage = encrypted[:32]
```

```
    check_bytes = salt_from_storage +
cls._encrypt_with_salt(data, salt_from_storage)
    return check_bytes == encrypted
```

```
@staticmethod
```

```
def _encrypt_with_salt(data: str, salt: bytes) -> bytes:
```

```
    return hashlib.pbkdf2_hmac('sha256', data.encode('utf-8'), salt,
100000, dklen=128)
```

Файл app.js

```
const form = document.getElementById("formDiv");
const url = "http://localhost:8000/get_hints/";
```

					КП.ІП-9228.045420.03.12	Арк.
Змін.	Арк.	№ докум.	Підп.	Дата.		13

```

const username = "user_2";
const password = "password_2";

async function getHints(textField, checkField) {
  let response = await fetch(
    url,
    {method: "POST",
      headers: new Headers({
        "Accept": "application/json",
        "Content-Type": "application/json",
        "Authorization": "Basic " + btoa(username + ":" +
          password), "Permissions": ["http://127.0.0.1:8000" ]
      })
    },
    body: JSON.stringify({
      "value": textField,
      "formatText": checkField,
      Cache: 'default'
    })
  )
  let responseJson = await response.json();

  let textToDisplay = "";
  for (const [line, hint] of
    Object.entries(responseJson.hints)) { textToDisplay +=
    "Рядок " + line + " " + hint + "\n"
  }
  if (responseJson.formatted_request != null) {
    textToDisplay += "\n--- --- --- Форматований SQL запит --- --- ---
    \n\n" textToDisplay += responseJson.formatted_request
  }

  return textToDisplay;

```

					КП.ІП-9228.045420.03.12	Арк.
Змін.	Арк.	№ докум.	Підп.	Дата.		14

```

}

form.addEventListener("submit", (e) => {
    e.preventDefault();
    const textField =
document.getElementById("requestString").value; const
checkField = document.getElementById("formatTheQuery"); let
hintsField = document.getElementById("responseString");
console.log(checkField.checked)

    getHints(textField, checkField.checked).then(r => {

        hintsField.value = r;
    });
});

form.addEventListener("reset", () => {
    const hintsField =
document.getElementById("responseString"); hintsField.value
= "";
})

```

Файл crud.py

```

from sqlalchemy.orm import Session

from ..models import Credentials

def get_user(db: Session, username: str):
    return db.query(Credentials).filter(Credentials.username == username).first()

```

Файл database.py

```

from sqlalchemy import create_engine

from .. import config

```

```
from sqlalchemy.ext.declarative import declarative_base
```

```
from sqlalchemy.orm import sessionmaker
```

```
SQLALCHEMY_DATABASE_URL =
```

```
f"sqlite:/// {config.db_path}" engine = create_engine(
```

```
    SQLALCHEMY_DATABASE_URL,  
    connect_args={"check_same_thread": False}  
)
```

```
SessionLocal = sessionmaker(autocommit=False, autoflush=False, bind=engine)
```

```
Base = declarative_base()
```

Файл manage.py

```
import uvicorn
```

```
from application.server.fast_api_server import app
```

```
if __name__ == "__main__":
```

```
    log_config = uvicorn.config.LOGGING_CONFIG
```

```
    log_config["formatters"]["access"]["fmt"] = "%(asctime)s - %(levelname)s -  
%(message)s"
```

```
    log_config["formatters"]["default"]["fmt"] = "%(asctime)s - %(levelname)s -  
%(message)s"
```

```
    uvicorn.run(app, host="127.0.0.1", port=8000, log_config=log_config)
```

Файл index.html

```
<!DOCTYPE html>
```

```
<html lang="uk">
```

```
<head>
```

					КП.ІП-9228.045420.03.12	Арк. 16
Змін.	Арк.	№ докум.	Підп.	Дата.		

```

<meta charset="UTF-8">
<title>SQL Linter</title>
<style>
    .button-area{
        display: flex;
        justify-content: space-between;

        align-items: center;
        width: 20%;
        margin-left: 100px;
        margin-right: 100px;
    }
    .container {
        display: flex;
        flex-wrap: wrap;
        text-align: center;
        justify-content: center;
        width: 70%;
        margin-left: 100px;
        margin-right: 100px;
    }
    h3{
        font-size: x-large;
    }
    h5{
        font-size: medium;
    }
    .header-area{
        justify-content: center;
        align-items: center;
        width: 70%;
        margin-left: 100px;
        margin-right: 100px;

```

```

        line-height: 10%;
    }
    textarea {
        background: #fce9c0; /* Колір фону */

        border: 2px solid #a9c358; /* Параметри рамки */
        padding-left: 10px; /* Поля */
        padding-right: 10px; /* Поля */
        margin-left: 100px;
        margin-right: 100px;
        width: 70%; /* Ширина */
        height: 160px; /* Висота */
        box-sizing: border-box; /* Алгоритм розрахунку ширини
        */ font-size: 14px; /* Розмір шрифту */
        resize: none /* Заборона зміни розмірів */
    }
</style>
</head>
<body>
    <div class="header-area" style="margin-bottom: -10px">
        <p><h3>SQLint</h3>
        <p><h5>Синтаксичний та стилістичний аналіз SQL
        запитів</h5> <hr style="margin-bottom: 30px">
    </div>
    <div id="formDiv">
        <form id="requestForm">
            <p><label for="requestString"></label><textarea id="requestString"
            name="requestText" placeholder="Введіть текст SQL запиту."></textarea></p>
            <div class="container">
                <div class="check-box-area">
                    <input type="checkbox" id="formatTheQuery"
                    name="format_the_query" value="QueryFormat">
                    <label for="formatTheQuery"> Форматувати
                    запит</label><br> </div>
                    <div class="button-area">

```

```

        <input type="submit" value="Відправити">
        <input type="reset" value="Очистити">
    </div>
</div>
</form>
</div>
<p><label>
    <textarea id="responseString" name="responseText" placeholder="Тут
буде результат аналізу SQL запиту"></textarea>
</label></p>
    <script type="module" src="app.js"></script>
</body>
</html>

```

					КП.ІП-9228.045420.03.12	Арк. 19
Змін.	Арк.	№ докум.	Підп.	Дата.		

Факультет інформатики та обчислювальної техніки

Кафедра інформатики та програмної інженерії

“ЗАТВЕРДЖЕНО”

Завідувач кафедри

_____ Едуард ЖАРІКОВ

“ ____ ” _____ 2023 р.

Програмне забезпечення для перевірки якості написання

SQL-запитів

Програма та методика тестування

КПІ.ПІ-9228.045420.04.51

“ПОГОДЖЕНО”

Керівник проєкту:

_____ Юрій ОЛІЙНИК

Нормоконтроль:

_____ Катерина ЛІЩУК

Виконавець:

_____ Дарія ФРАДИНСЬКА

Київ – 2023

ЗМІСТ

1	ОБ’ЄКТ ВИПРОБУВАНЬ.....	3
2	МЕТА ТЕСТУВАННЯ	3
3	МЕТОДИ ТЕСТУВАННЯ.....	4
4	ЗАСОБИ ТА ПОРЯДОК ТЕСТУВАННЯ	5

					КПІ. ІП-9228.045420.04.51	Арк.
Змін	Арк.	№ докум.	Підп.	Дата.		2

1 ОБ'ЄКТ ВИПРОБУВАНЬ

Об'єктом випробування є програмне забезпечення для перевірки якості написання SQL-запитів. Серверна частина додатку реалізована за мікросервісним шаблоном архітектури мовою програмування Python, за допомогою фреймворку FastApi, а клієнтська мовою програмування JavaScript.

					КПІ. ІП-9228.045420.04.51	Арк.
Змін	Арк.	№ докум.	Підп.	Дата.		3

2 МЕТА ТЕСТУВАННЯ

Метою тестування є наступне:

- перевірка правильності роботи програмного забезпечення відповідно до функціональних вимог;
- перевірка збереження даних;
- перевірка сумісності веб-додатку з останніми версіями сучасних браузерів;
- знаходження проблем, помилок і недоліків з метою їх усунення;
- перевірка зручності графічного інтерфейсу.

					КПІ. ІП-9228.045420.04.51	Арк.
Змін	Арк.	№ докум.	Підп.	Дата.		4

3 МЕТОДИ ТЕСТУВАННЯ

Для тестування програмного забезпечення використовуються такі методи:

- статичне тестування перевіряється програма разом з усією документацією, яка аналізується на предмет дотримання стандартів програмування;
- динамічне тестування застосовується в процесі виконання програми.

Коректність програмного засобу перевіряється на певній кількості тестів. При прогоні кожного з них збираються та аналізуються дані про проблеми та помилки в роботі програми;

- функціональне тестування – полягає у перевірці відповідності реальної поведінки програмного забезпечення очікуваній;
- системне тестування – перевіряється усе програмне забезпечення в цілому;
- мануальне тестування – тестування без використання автоматизації, тест-кейси пише особа, що тестує програмне забезпечення;
- тестування «чорної скриньки» об'єктом тестування тут є функції присутні у програмі. Перевіряється коректність вихідних даних при заданих вхідних;
- тестування «білої скриньки» об'єктом тестування тут є внутрішня поведінка програми. Перевіряється коректність побудови всіх елементів програми та правильність їхньої взаємодії один з одним;
- тестування «сірої скриньки» об'єктом тестування тут є деякі особливості внутрішньої поведінки програми. Перевіряється коректність вихідних даних при заданих вхідних, застосовується для тестування окремих алгоритмів (функцій).

					КПІ. ІІ-9228.045420.04.51	Арк.
						5
Змін	Арк.	№ докум.	Підп.	Дата.		

4 ЗАСОБИ ТА ПОРЯДОК ТЕСТУВАННЯ

Тестування виконується мануально та написанням unit-тестів, з метою знаходження помилок та недоліків як у функціональній частині програмного забезпечення так і в зручності користування. Для того, щоб перевірити працездатність та відмовостійкість застосунку, необхідно провести наступні тестування:

- динамічне тестування на відповідність функціональним вимогам;
- тестування на виведення повідомлень про помилку, коли це необхідно;
- тестування зміни орієнтації екрану;
- тестування інтерфейсу користувача;
- тестування зручності використання;

					КПІ. ІП-9228.045420.04.51	Арк.
Змін	Арк.	№ докум.	Підп.	Дата.		6

Факультет інформатики та обчислювальної техніки

Кафедра інформатики та програмної інженерії

“ЗАТВЕРДЖЕНО”

Завідувач кафедри

_____ Едуард ЖАРІКОВ

“ ____ ” _____ 2023 р.

**ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ ДЛЯ ПЕРЕВІРКИ ЯКОСТІ
НАПИСАННЯ SQL-ЗАПИТІВ**

Керівництво користувача

КПІ.ПІ-9228.045420.05.34

“ПОГОДЖЕНО”

Керівник проєкту:

_____ Юрій ОЛІЙНИК

Нормоконтроль:

_____ Ірина ВІТКОВСЬКА

Виконавець:

_____ Дарія ФРАДИНСЬКА

Київ – 2023

ЗМІСТ

1	ПРИЗНАЧЕННЯ ПРОГРАМИ	3
2	ПІДГОТОВКА ДО РОБОТИ З ПРОГРАМНИМ ЗАБЕЗПЕЧЕННЯМ.....	4
2.1	Системні вимоги для коректної роботи.....	4
2.2	Завантаження застосунку.....	4
2.3	Перевірка коректної роботи.....	4
3	ВИКОНАННЯ ПРОГРАМИ	5

					КПІ.ІП-9228.045420.05.34	Арк.
						2
Змін.	Арк.	№ докум.	Підп.	Дата.		

1 ПРИЗНАЧЕННЯ ПРОГРАМИ

Програмне забезпечення призначено для виявлення та виправлення проблеми з SQL-запитами користувачів. Програмне забезпечення дозволяє проводити синтаксичний, стилістичний та оптимізаційний аналізи запитів та отримувати результати аналізу у вигляді підказок, або сформованого звіту. Крім того, користувач має змогу отримати відформатований, відповідно до стандартів, запит.

Система складається з мікросервісу, який виконує синтаксичний та стилістичний аналіз SQL – запиту, поданого у вигляді тексту, та однієї з можливих реалізацій інтерфейсу користувача – web-інтерфейсу, що є одиночною web-сторінкою з інтерактивними елементами.

Основний мікросервіс з API реалізовано з використанням Python - фреймворку FastAPI, web – інтерфейс – HTML5 сторінка з динамічними елементами JavaScript.

					КП.ІП-9228.045420.05.34	Арк.
						3
Змін.	Арк.	№ докум.	Підп.	Дата.		

2 ПІДГОТОВКА ДО РОБОТИ З ПРОГРАМНИМ ЗАБЕЗПЕЧЕННЯМ

2.1 Системні вимоги для коректної роботи

Для успішної роботи даного застосунку необхідне виконання наступних вимог:

- наявність комп'ютеру чи планшету з операційною системою Windows, MacOS чи Linux;
- наявність встановлених web-браузера, Docker Desktop.

2.2 Завантаження застосунку

На даний момент застосунок можна встановити локально на пристрій. Для цього необхідно мати встановленим Docker Desktop та запустити контейнери з API та клієнтською частиною.

2.3 Перевірка коректної роботи

По завершенню встановлення програмного забезпечення у web-браузері відкривається інтерфейс користувача з коректним відображенням усіх полів для введення та кнопок. У разі, якщо інтерфейс не відкрився, це означає про виникнення проблем з клієнтською частиною. Якщо ж при введенні SQL-запиті та натисканні кнопки “Відправити запит” нічого не відбувається, чи результати перевірки відображаються некоректно - виникли проблеми із серверною частиною.

					КПІ.ІП-9228.045420.05.34	Арк.
						4
Змін.	Арк.	№ докум.	Підп.	Дата.		

3 ВИКОНАННЯ ПРОГРАМИ

Після запуску програмного забезпечення користувачу буде відображено сторінку для введення запиту та отримання результатів аналізу. Користуючись полями для введення користувач може відправити введений запит на перевірку і отримати результат перевірки.

Ілюстрація екранної форми зображена на рисунку 3.1

SQLint
Синтаксичний та стилістичний аналіз SQL запитів

Введіть текст SQL запиту.

☐ Форматувати запит

Тут буде результат аналізу SQL запиту

Рисунок 3.1 – Головна сторінка програмного забезпечення

Користуючись верхнім полем для вводу користувач може ввести свій SQL-запит і натиснувши кнопку “Відправити” запустити процес перевірки. Результати роботи програми будуть відображатись у нижньому полі. При натисканні кнопки “Очистити” поля з введеним запитом та результатами аналізу будуть очищені та сторінка повернеться до свого початкового вигляду

Ілюстрація сторінки наведена на рисунку 3.2

					КП.ІП-9228.045420.05.34	Арк.
						5
Змін.	Арк.	№ докум.	Підп.	Дата.		

SQLint

Синтаксичний та стилістичний аналіз SQL запитів

```
SELECT *
FROM
  TableName AS a      -- Name = 'Lee, Barry'
INNER JOIN
  TableFullNames AS b -- Name = 'Lee.Barry'
on /* */ a.id = b.id
WHERE
  LEFT(a.name, 1) = 'n' AND
  LEFT(a.name, 1) = 'z';
```

☐ Форматувати запит

Відправити

Очистити

Рядок 1 ->SELECT-Кореневе слово знаходиться не на окремому рядку
Рядок 2 ->FROM-Відсутнє вирівнювання вліво для кореневого слова
Рядок 6 ->on-Зарезервоване слово в нижньому регістрі
Рядок 8 ->AND-Логічний оператор AND не на початку рядка.

Рисунок 3.2 – Результати роботи програмного забезпечення

При виборі прапорцю “Форматувати запит” користувач, окрім результатів аналізу, може отримати відформатований запит у полі з результатами аналізу.

Ілюстрація наведена на рисунку 3.3

SQLint

Синтаксичний та стилістичний аналіз SQL запитів

```
SELECT *
FROM
  TableName AS a      -- Name = 'Lee, Barry'
INNER JOIN
  TableFullNames AS b -- Name = 'Lee.Barry'
on /* */ a.id = b.id
WHERE
  LEFT(a.name, 1) = 'n' AND
  LEFT(a.name, 1) = 'z';
```

☒ Форматувати запит

Відправити

Очистити

РЯДОК 8 ->AND-Логічний оператор AND не на початку рядка.

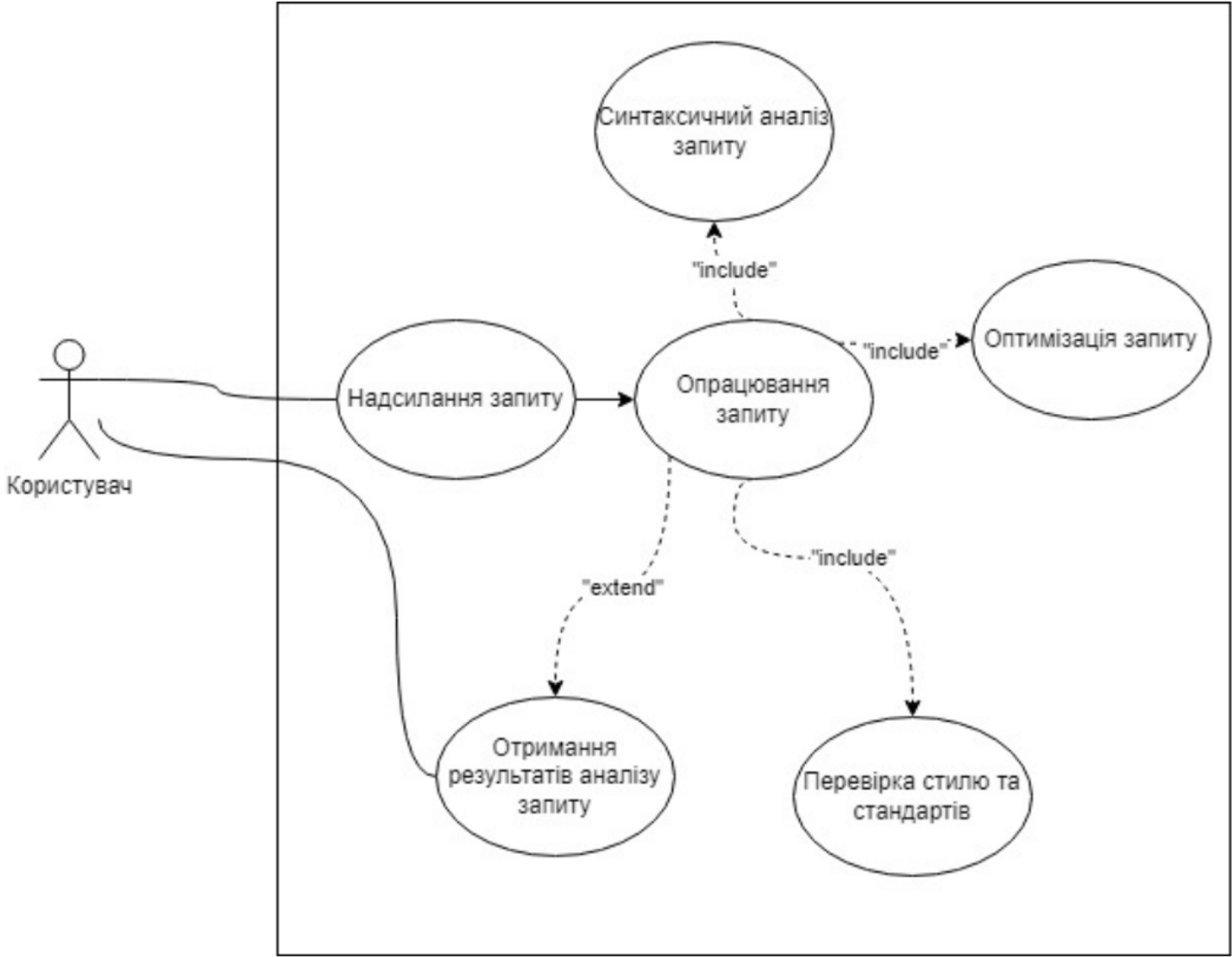
--- Форматований SQL запит ---

```
SELECT *
FROM  tablename as a -- Name = 'Lee, Barry'
      INNER JOIN tablefullnames as b -- Name = 'Lee.Barry'
        ON a.id = b.id /* */
WHERE left(a.name, 1) = 'n'
      and left(a.name, 1) = 'z';
```

Рисунок 3.2 – Результати роботи програмного забезпечення з вибором функції форматування

Звіт складається з таких розділів:

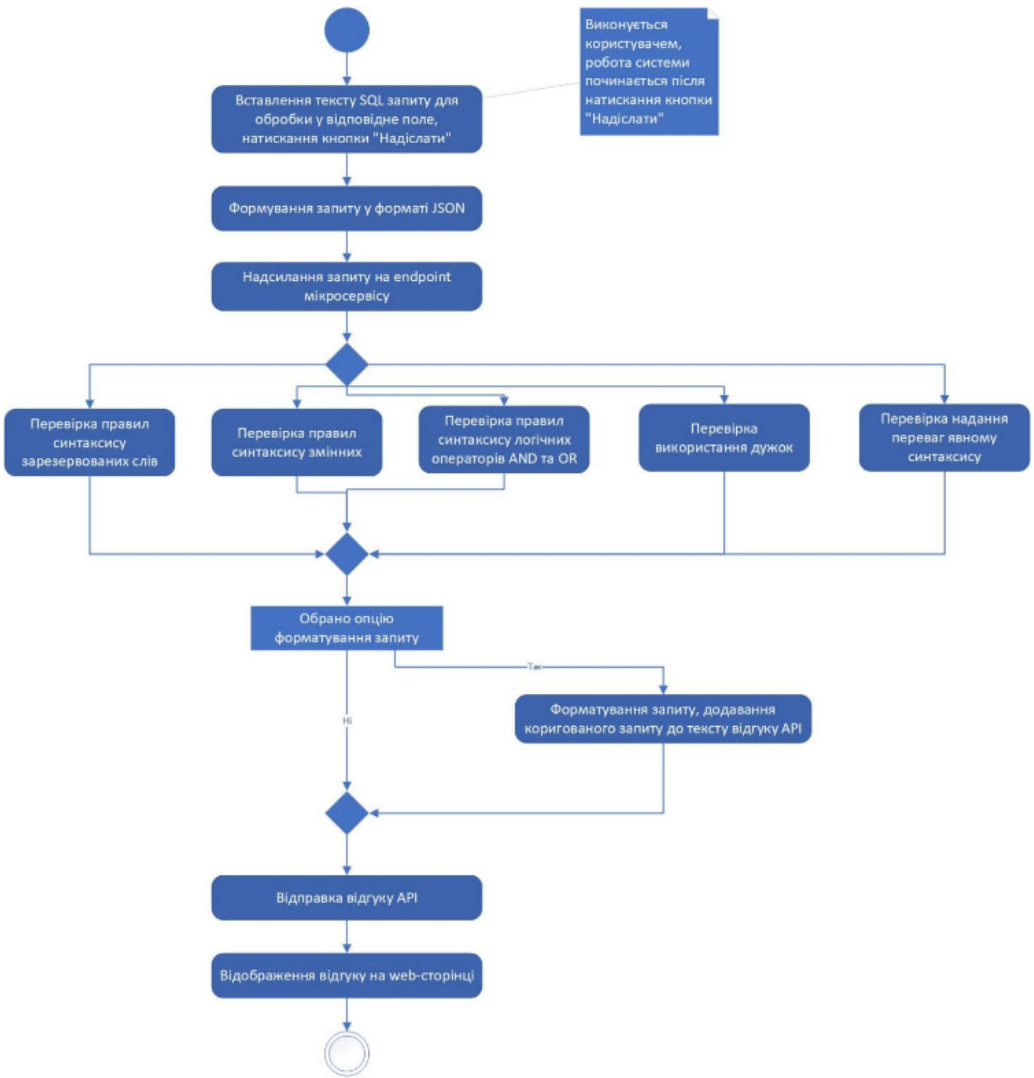
- результати синтаксичного аналізу;
- результати оптимізаційного аналізу;
- результати стилістичного аналізу;
- відформатований код.



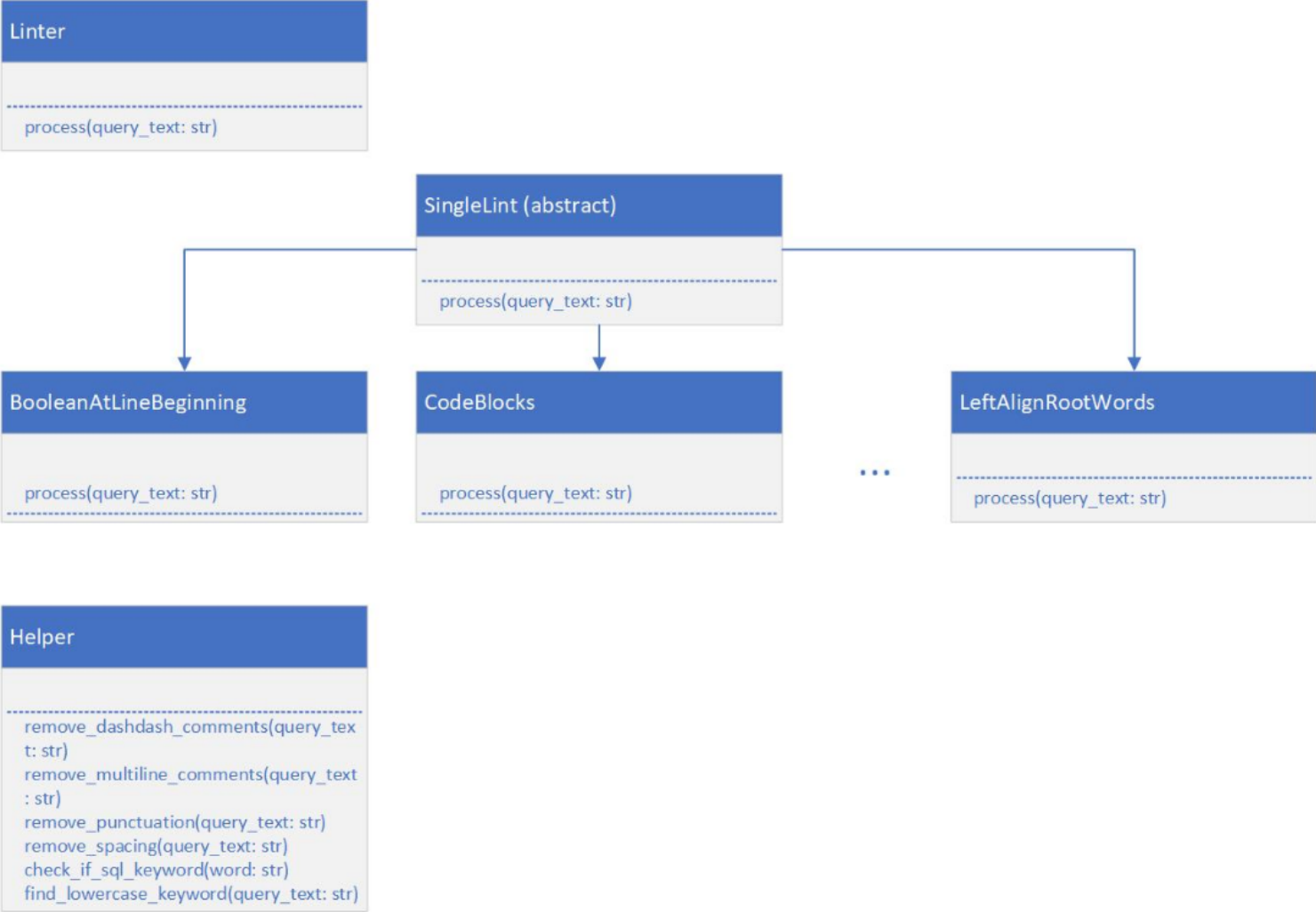
						КПІ.ІП-9228.0450420.06.99.CCB				
						Схема структурна варіантів використання	Літера	Маса	Масштаб	
Зм.	Арк.	№ документа	Підпис	Дата						
Розробив		Фрадинська Д.В.								
Перевірів		Олійник Ю.О.								
Т. кон.							Аркуш 1	Аркушів 1		
Н. кон.		Ліщук К.І.				Програмне забезпечення для перевірки якості написання SQL-запитів	КПІ ім.Ігоря Сікорського Кафедра ІПІ гр. ІП-92			
Затвердив		Жаріков Е.В.								



					КПІ.ІП-9228.0450420.07.99.CCC			
					Схема структурна станів	Лит.	Арк.	Аркуші
Зм.	Арк.	№ докум.	Підп.	Дата				
Розроб.		Фрадинська Д.В.					1	1
Перев.		Олійник Ю.О.						
Т. Кон.						Аркуш		Аркуші
					Програмне забезпечення для перевірки якості написання SQL-запитів	КПІ ім.Ігоря Сікорського Кафедра ІПІ гр. ІП-92		
Н. Кон.		Ліщук К.І.						
Затв.		Жаріков Е.В.						



						КПІ.ІП-9228.0450420.08.99.ССД			
						Схема структурна діяльності	Літера	Маса	Масштаб
Зм.	Арк.	№ документа	Підпис	Дата					
Розробив		Фрадинська Д.В.							
Перевірив		Олійник Ю.О.							
Т. кон.							Аркуш 1	Аркушів 1	
Н. кон.		Ліщук К.І.				Програмне забезпечення для перевірки якості написання SQL-запитів	КПІ ім.Ігоря Сікорського Кафедра ІПІ гр. ІП-92		
Затвердив		Жаріков Е.В.							



					КПІ.ІП-9228.0450420.09.99.ССК				
Зм.	Арк.	№ документа	Підпис	Дата	Схема структурна класів програмного забезпечення		Літера	Маса	Масштаб
Розробив		Фрадинська Д.В.							
Перевірів		Олійник Ю.О.							
Т. кон.									
							Аркуш 1	Аркушів 1	
Н. кон.		Ліщук К.І.			Програмне забезпечення для перевірки якості написання SQL-запитів		КПІ ім.Ігоря Сікорського Кафедра ІПІ гр. ІП-92		
Затвердив		Жаріков Е.В.							

SQLint

Синтаксичний та стилістичний аналіз SQL запитів

Введіть текст SQL запиту.

☐ Форматувати запит

Відправити

Очистити

Тут буде результат аналізу SQL запиту

Екранна форма головної сторінки програмного застосунку

SQLint

Синтаксичний та стилістичний аналіз SQL запитів

```
SELECT *
FROM
  TableNames AS a -- Name = 'Lee, Barry'
INNER JOIN
  TableFullNames AS b -- Name = 'Lee,Barry'
on /* */ a.id = b.id
WHERE
  LEFT(a.name, 1) = 'n' AND
  LEFT(a.name, 1) = 'z';
```

☒ Форматувати запит

Відправити

Очистити

Рядок 8 ->AND-логічний оператор AND не на початку рядка.

--- --- --- форматований SQL запит --- --- ---

```
SELECT *
FROM  tableNames as a -- Name = 'Lee, Barry'
INNER JOIN tableFullNames as b -- Name = 'Lee,Barry'
ON a.id = b.id /* */
WHERE left(a.name, 1) = 'n'
and left(a.name, 1) = 'z';
```

Екранна форма результатів роботи програмного забезпечення з відформатованим запитом

SQLint

Синтаксичний та стилістичний аналіз SQL запитів

```
SELECT *
FROM
  TableNames AS a -- Name = 'Lee, Barry'
INNER JOIN
  TableFullNames AS b -- Name = 'Lee,Barry'
on /* */ a.id = b.id
WHERE
  LEFT(a.name, 1) = 'n' AND
  LEFT(a.name, 1) = 'z';
```

☐ Форматувати запит

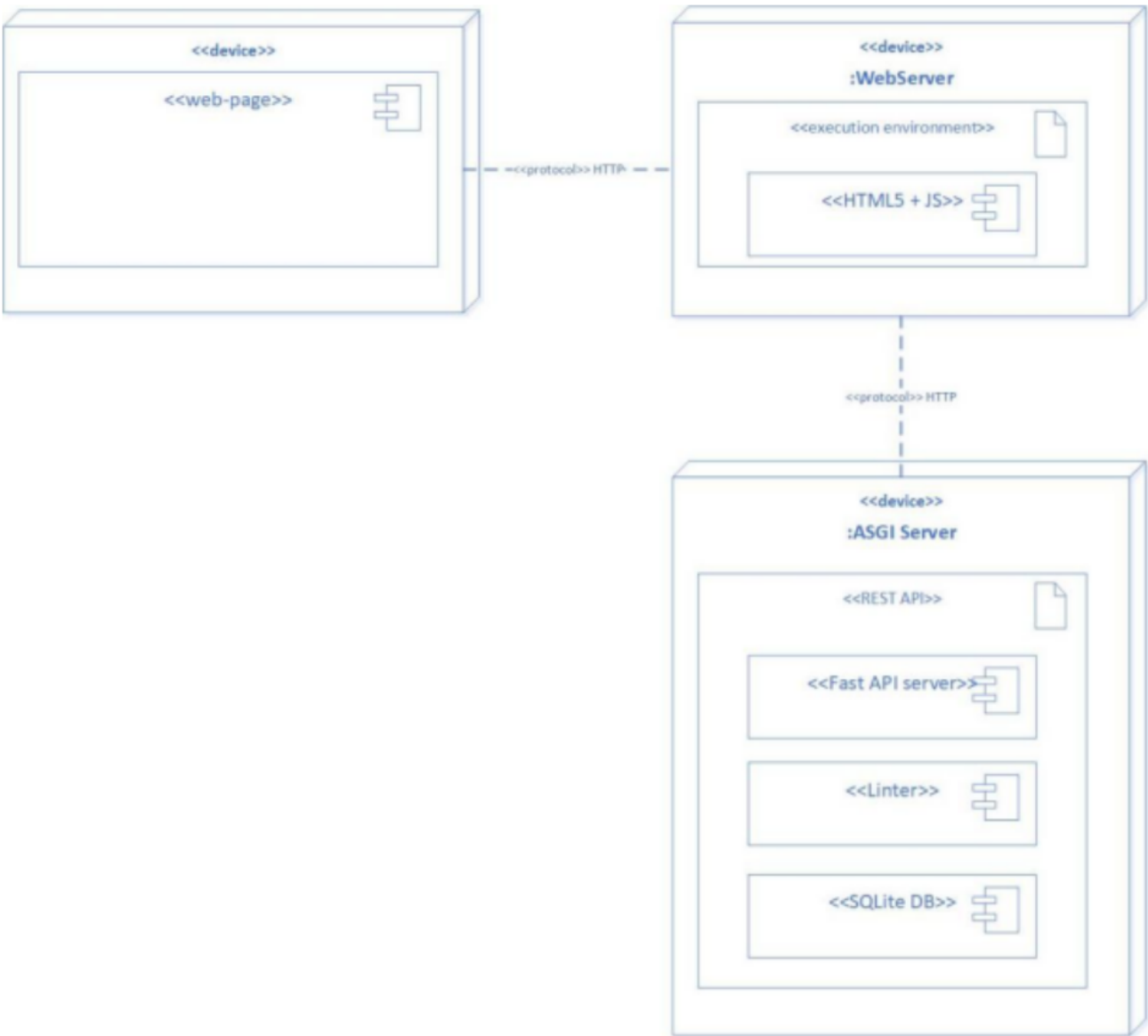
Відправити

Очистити

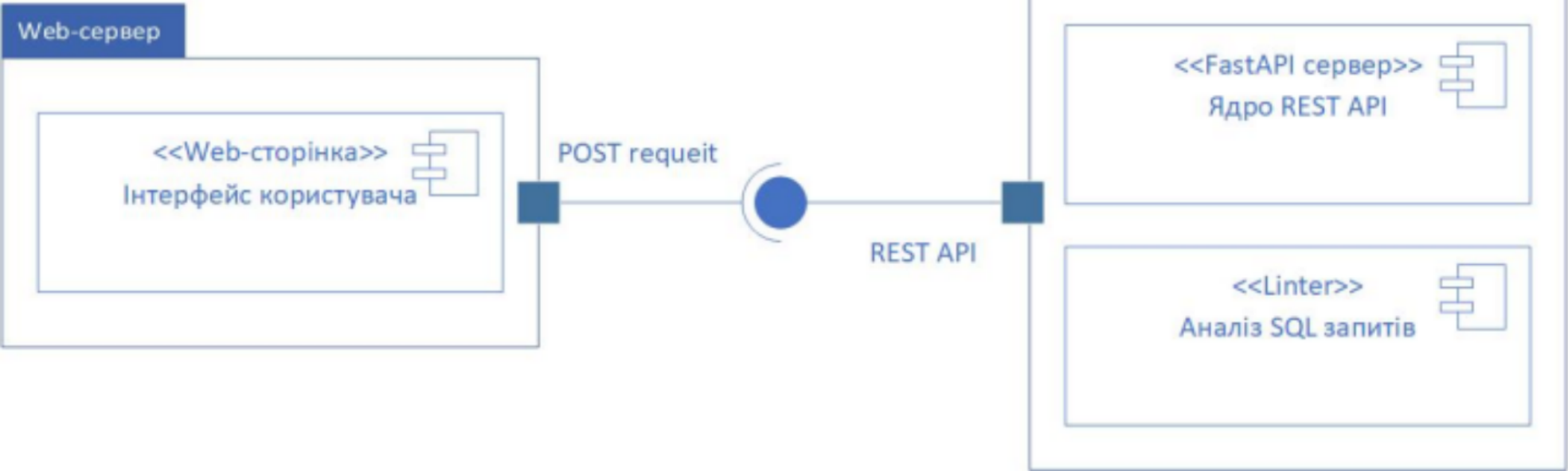
Рядок 1 ->SELECT-кореневе слово знаходиться не на окремому рядку
Рядок 2 ->FROM-Відсутнє вирівнювання впливо для кореневого слова
Рядок 6 ->on-Зарезервоване слово в нижньому регістрі
Рядок 8 ->AND-логічний оператор AND не на початку рядка.

Екранна форма результатів роботи програмного забезпечення

					КПІ.ІП-9228.0450420.11.99.KE				
					Креслення вигляду екранних форм	Літера		Маса	Масштаб
Зм.	Арк.	№ документа	Підпис	Дата					
Розробив		Фрадинська Д.В.							
Перевірив		Олійник Ю.О.							
Т. кон.						Аркуш 1	Аркушів 1		
Н. кон.		Ліщук К.І.			Програмне забезпечення для перевірки якості написання SQL-запитів	КПІ ім.Ігоря Сікорського Кафедра ІПІ гр. ІП-92			
Затвердив		Жаріков Е.В.							



					КПІ.ІП-9228.0450420.11.99.CCP					
Зм.	Арк.	№ документа	Підпис	Дата	Схема структурна розгортання програмного забезпечення			Літера	Маса	Масштаб
Розробив		Фрадинська Д.В.								
Перевішив		Олійник Ю.О.								
Т. кон.										
								Аркуш 1	Аркушів 1	
Н. кон.		Ліщук К.І.			Програмне забезпечення для перевірки якості написання SQL-запитів			КПІ ім.Ігоря Сікорського Кафедра ІПІ гр. ІП-92		
Затвердив		Жаріков Е.В.								



					КПІ.ІП-9228.0450420.12.99.CCM				
					Схема структурна компонентів програмного забезпечення				
Зм.	Арк.	№ документа	Підпис	Дата					
Розробив		Фрадинська Д.В.							
Перевірів		Олійник Ю.О.							
Т. кон.									
Н. кон.		Ліщук К.І.			Програмне забезпечення для перевірки якості написання SQL-запитів			Літера	
Затвердив		Жаріков Е.В.						Маса	Масштаб
								Аркуш	Аркушів
					КПІ ім.Ігоря Сікорського Кафедра ІПІ гр. ІП-92				