Київський національний університет імені Тараса Шевченка
Факультет комп’ютерних наук та кібернетики
Кафедра математичної інформатики

Звіт
до модуля №2 лабораторної роботи
з дисципліни
«Інформаційні технології»

роботу виконав
студент 4 курсу
групи МІ-4
Є. О. Кирилов

Київ – 2023

Постановка задачі

Загальні вимоги.

Вимоги щодо структури бази:

- кількість таблиць принципово не обмежена (реляції між таблицями не враховувати);
- кількість полів та кількість записів у кожній таблиці також принципово не обмежені.

У кожній роботі для всіх варіантів для полів у таблицях треба забезпечити підтримку таких типів: integer, real, char, string.

Також у кожній роботі треба реалізувати функціональну підтримку для:

- створення бази;
- створення (із валідацією даних) та знищення таблиці з бази;
- створення, валідації, перегляду та редагування рядків таблиці;
- збереження табличної бази на диску та зчитування її з диску.

Додаткові типи та операція над таблицями – **варіант 01**:

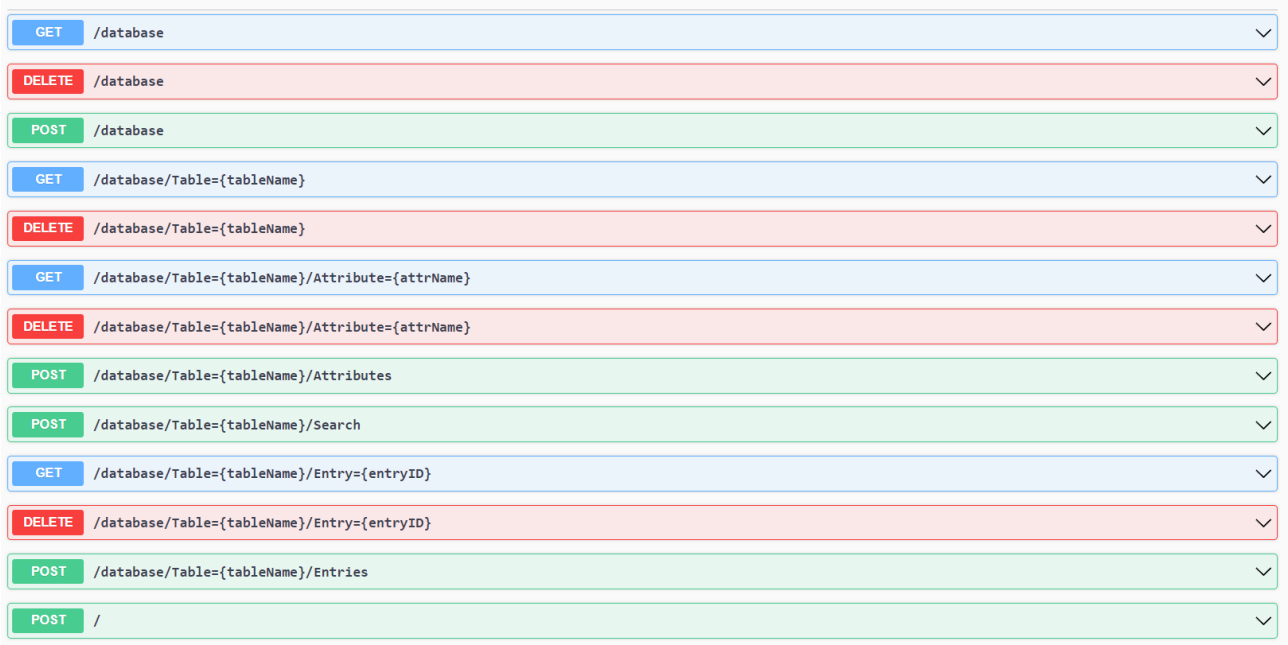
- ТИП – 0) color (RGB код кольору), colorInvl;
- ОПЕРАЦІЯ – 1) пошук (за шаблоном) та перегляд знайдених рядків таблиці.

Реалізація етапів 3-4 та 7-8 відсутня. Реалізовано тільки етапи 5 і 6.

ЕТАП 5. REST

Для виконання етапу REST було використано бібліотеку класів мови C# предметної області (Table, Attribute, Entry, тощо) розроблену під час виконання модуля №1. REST HTTP сервер розроблено мовою C# у IDE Visual Studio з використанням бібліотек Microsoft.AspNetCore.OpenApi, Microsoft.OpenApi, Swashbuckle.AspNetCore. Сервер тримає бази даних у вигляді JSON файлів та не зберігає клієнтську інформацію (сервер є stateless), що є обов'язковою поведінкою в ідеології REST.

Інтерфейс, що сервер надає клієнтам, задокументовано у вигляді специфікації OpenAPI мовою C#. Ось перелік усіх ендпоінтів розробленого REST API:



Окрім явно описаних ендпоінтів, сервер також надає доступ до специфікації OpenAPI за посиланням https://localhost:44308/swagger/v1/swagger.json. Початок файлу має наступний вигляд:

```
{
  "openapi": "3.0.1",
  "info": {
    "title": "ServerSide",
    "version": "1.0"
  },
  "paths": {
    "/database": {
      "get": {
        "tags": [
          "ServerSide"
        ],
        "operationId": "GET Database",
        "responses": {
          "200": {
            "description": "OK",
            "content": {
              "application/json": {
                "schema": {
                  "$ref": "#/components/schemas/Database"
                }
              }
            }
          }
        }
      },
      "delete": {
        "tags": [
          "ServerSide"
        ],
        "operationId": "DELETE Database",
        "responses": {
          "200": {
            "description": "OK"
          }
        }
      },
      "post": {
        "tags": [
          "ServerSide"
        ],
        "operationId": "POST Table",
        "requestBody": {
          "content": {
            "application/json": {
              "schema": {
                "$ref": "#/components/schemas/Table"
              }
            }
          },
          "required": true
```


Для доступу до такого API використовується Swagger. Приклад запиту GET /database в Swagger:


```
{
  "tables": [
    {
      "name": "HairColor",
      "attributes": [
        {
          "name": "personId",
          "type": "Integer"
        },
        {
          "name": "hairColor",
          "type": "Color"
        }
      ],
      "entries": [
        {
          "values": [
            {
              "value": "0",
              "attribute": {
```

```
                "name": "personId",
                "type": "Integer"
              }
            },
            {
              "value": "(255, 255, 255)",
              "attribute": {
                "name": "hairColor",
                "type": "Color"
              }
            }
          ]
        },
        {
          "values": [
            {
              "value": "1",
              "attribute": {
                "name": "personId",
                "type": "Integer"
              }
            },
            {
              "value": "(0, 0, 255)",
              "attribute": {
                "name": "hairColor",
                "type": "Color"
              }
            }
          ]
        }
      ]
    }
  ]
}
```

ЕТАП 6. GraphQL

У цьому етапі я розробив GraphQL API для СУБД. Код написано мовою Python з використанням бібліотек graphene, starlette_graphene3, fastapi та uvicorn:

- За допомогою бібліотеки graphene визначаються класи, що можуть бути повернуті у відповідь на запит.
- Після створення усіх запитів та мутацій, бібліотека starlette_graphene3 об’єднує їх в опис GraphQL.
- Бібліотека fastapi потрібна, щоб задати ендпойнт, за яким буде доступний GraphQL – в даному випадку я задав ендпоінт /graphql.
- Бібліотека uvicorn потрібна для запуску сервера, описаного за допомогою попередніх бібліотек.

На відміну від REST у користувачів GraphQL є можливість точніше задавати поля, що має повернути сервер, що є також і необхідністю. Під час обробки запиту обчислюються і повертаються лише ті поля, які задав користувач, таким чином зменшуючи навантаження на обчислювальні ресурси та мережу; під час обробки мутації змінюється стан об’єктів на сервері і повертається інформація про них.

Для доступу до такого API можна використовувати Postman. Приклад запиту, що запитує назву усіх таблиць таблиці та імена людей в них в БД "DMBS-Ex":

```
query GetDatabaseByName {
    getDatabaseByName(databaseName: "DBMS-Ex") {
        name
        tables {
            name
            entries {
                values {
                    ... on StringValue {
                        string
                    }
                }
            }
        }
    }
}
```


```
{
    "data": {
        "getDatabaseByName": {
            "name": "DBMS-Ex",
            "tables": [
                {
                    "name": "HairColor",
                    "entries": [
                        {
                            "values": [
                                {
                                    "string": "Mary"
                                },
                                {}
                            ]
                        },
                        {
                            "values": [
                                {
                                    "string": "Alex"
                                },
                                {}
                            ]
                        }
                    ]
                }
            ]
```