**КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ**

**ІМЕНІ ТАРАСА ШЕВЧЕНКА**

Проект з дисципліни

**«Інформаційні технології»**

**Виконав:**

студент 4 курсу ОС «Бакалавр»

спеціальності «Комп’ютерні науки»,

освітньої програми «Інформатика»,

групи ТТП-42

*Расахацький Максим Володимирович*

**Київ-2022**

1. **Опис завдання**

Реалізувати Фрагментарну реалізацію систем управління табличними базами даних

* 1. Загальні вимоги
* 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 треба забезпечити підтримку (для полів у таблицях) наступних (загальних для всіх варіантів!) 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;
  + перегляду та редагування рядків таблиці;
  + збереження табличної бази на диску та, навпаки, зчитування її з диску.
  1. Варіанти додаткових типів

Потрібно забезпечити підтримку (для можливого використання у таблицях) двох додаткових типів у відповідності з одним із наступних варіантів: html-файли; stringInvl (інтервал рядків);

* 1. Варіанти додаткових операцiй над таблицями

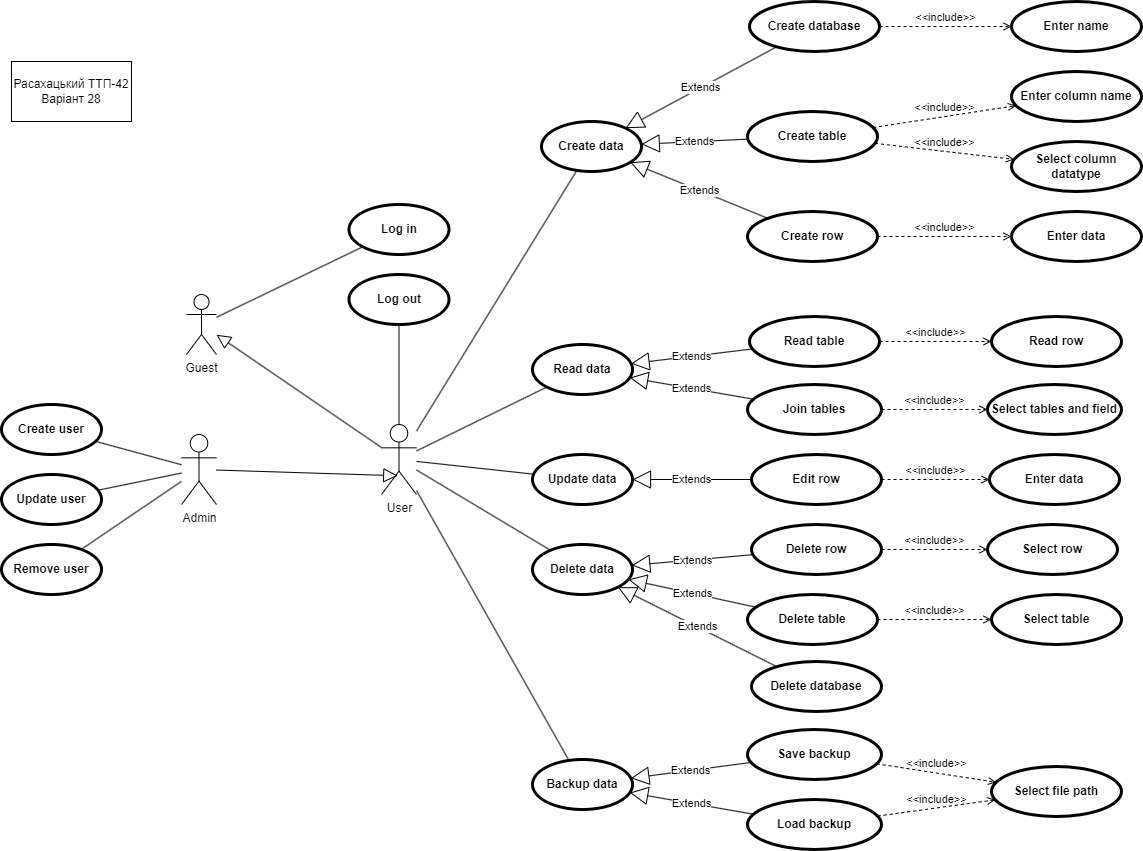
Потрібно реалізувати операцiї над таблицями у відповідності з варіантом 8: сполучення таблиць (за спiльним полем);

1. **Опис реалізації**

Під час виконання лабораторних робіт було виконано наступні етапи:

* 1. **Етап 0 - Попередній етап**

Функціональна специфікація системи управління табличними базами даних (СУТБД) у вигляді однієї або кількох діаграм прецедентів UML.



* 1. **Етап 1-2 - Розробка локальної (не розподіленої) версії СУТБД (із власною реалізацією класів "Таблиця" та "База")**

Розробка власних класів для понять "Таблиця", "База" та, можливо, деяких інших класів, спряжених із поняттям "Таблиця".

Реалізовано завдання було мовою Golang, в якій немає класів, але є структури та функції що реалізують перні інтерфейси.

Створені структури:

|  |
| --- |
| type Database struct { |
|  |

|  |
| --- |
| Name string |
|  |

|  |
| --- |
| Tables []Table |
|  |

|  |
| --- |
| } |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| type DBPathJSON struct { |
|  |

|  |
| --- |
| Name string `json:"name"` |
|  |

}

|  |
| --- |
| type TableHeaderJSON struct { |
|  |

|  |
| --- |
| Name string `json:"name"` |
|  |

|  |
| --- |
| Type string `json:"type"` |
|  |

|  |
| --- |
| } |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| type TableJSONValues struct { |
|  |

|  |
| --- |
| Name string `json:"name"` |
|  |

|  |
| --- |
| Headers []TableHeaderJSON `json:"headers"` |
|  |

|  |
| --- |
| Values [][]interface{} `json:"values"` |
|  |

|  |
| --- |
| } |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| type TableJSON struct { |
|  |

|  |
| --- |
| Name string `json:"name"` |
|  |

|  |
| --- |
| Headers []TableHeaderJSON `json:"headers"` |
|  |

|  |
| --- |
| } |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| type DatabaseInfoJSON struct { |
|  |

|  |
| --- |
| Tables []TableJSON `json:"tables"` |
|  |

}

|  |
| --- |
| type Table struct { |
|  |

|  |
| --- |
| Name string |
|  |

|  |
| --- |
| Types []string |
|  |

|  |
| --- |
| Headers []string |
|  |

|  |
| --- |
| Values [][]DBType |
|  |

|  |
| --- |
| } |
|  |
| type DBType interface { | |
|  | |

|  |
| --- |
| Value() interface{} |
|  |

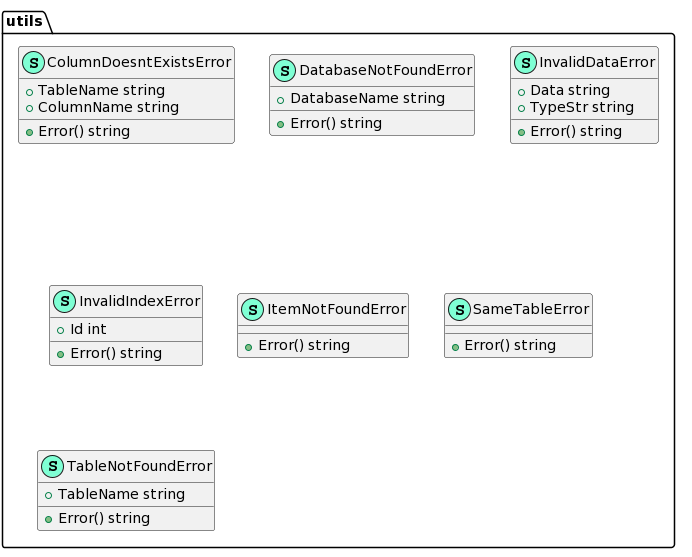
|  |
| --- |
| TypeName() string |
|  |

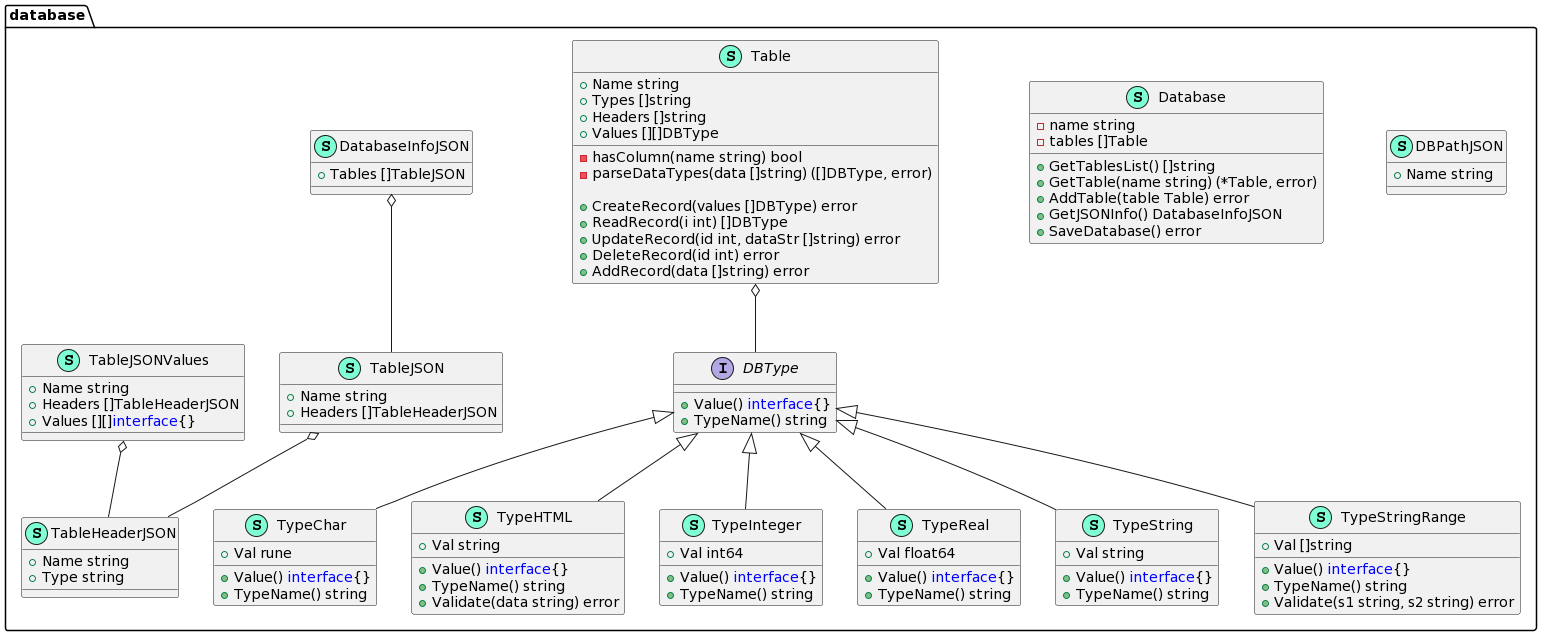
}

Створення UML-діаграми класів (з наявними між класами відношеннями).

Для генерації діаграми класів використано утиліту [goplantuml](https://github.com/jfeliu007/goplantuml)

Для рендеру діаграми використано [dumels](https://www.dumels.com/)

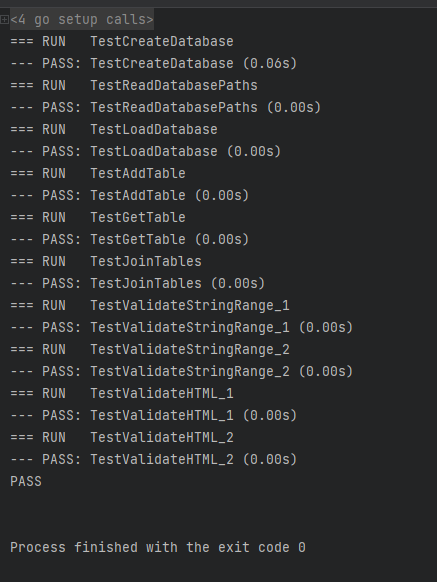




Проведення unit-тестування. Надати 3..\* тести, один з яких має бути призначеним для тестуванням “індивідуальної” (варіантної) операції з розділу III.

Юніт тести знаходяться в файлі DBMS/main\_test.go

Результат виконання тестів:



Забезпечення інтерфейсу користувача на основі форм.

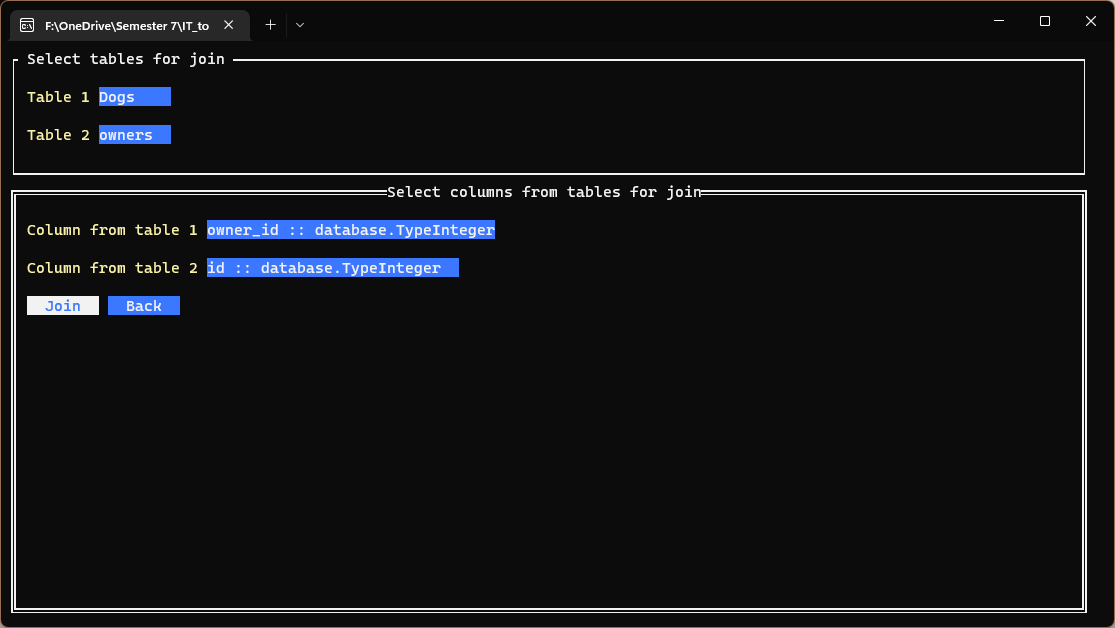
Було використано CLI бібліотеку [tview](https://github.com/rivo/tview)



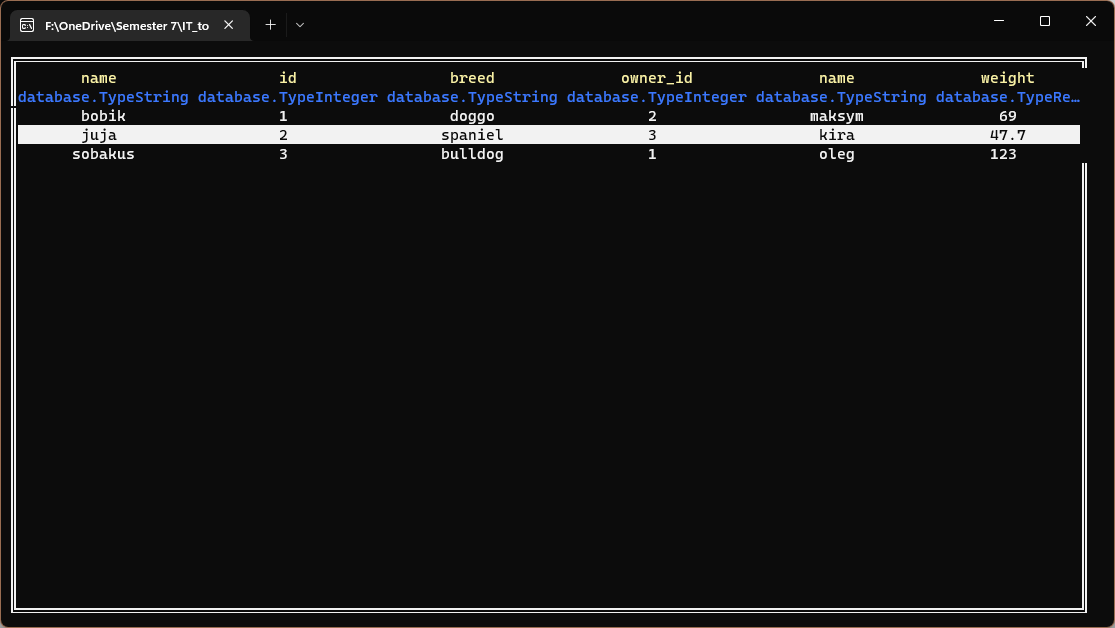
Навігація таблицею



Редагування поля



Запуск операції Join



Результат операції Join

* 1. **10) REST web-сервіси. Реалізація операцій над даними, орієнтуючись на їх ієрархічну структуру: база -> таблиця -> ... та на використання HTTP-запитів (як мінімум GET, POST та DELETE). Потрібно розробити REST API сервер та продемонструвати його роботу на відповідних тестових HTTP-запитах (Postman, cURL тощо).**

Спочатку всі запити та інтерфейси до них були написані вручну. Коли було вивчено генерацію коду, також додатково були реалізовані генеровані інтерфейси.

Приклади запитів

**GET databases list**

GET http://localhost:1323/databases

HTTP/1.1 200 OK

Content-Type: application/json; charset=UTF-8

Vary: Origin

Date: Thu, 01 Dec 2022 22:42:09 GMT

Content-Length: 37

[

{

"name": "Animals"

},

{

"name": "test"

}

]

#### GET table Dogs from database Animals

GET http://localhost:1323/databases/Animals/Dogs

HTTP/1.1 200 OK

Content-Type: application/json; charset=UTF-8

Vary: Origin

Date: Thu, 01 Dec 2022 22:49:49 GMT

Content-Length: 295

{

"name": "Dogs",

"headers": [

{

"name": "name",

"type": "database.TypeString"

},

{

"name": "id",

"type": "database.TypeInteger"

},

{

"name": "breed",

"type": "database.TypeString"

},

{

"name": "owner\_id",

"type": "database.TypeInteger"

}

],

"values": [

[

"bobik",

1,

"doggo",

2

],

[

"juja",

2,

"spaniel",

3

],

[

"sobakus",

3,

"bulldog",

1

]

]

}

#### GET joined table by column 'owner\_id' from table 'Dogs' and 'id' from table 'owners'

GET http://localhost:1323/databases/animals/joined\_tables?t1=Dogs&t2=owners&c1=owner\_id&c2=id

HTTP/1.1 200 OK

Content-Type: application/json; charset=UTF-8

Vary: Origin

Date: Thu, 01 Dec 2022 22:54:41 GMT

Content-Length: 487

{

"name": "Dogs\_join\_owners",

"headers": [

{

"name": "name",

"type": "database.TypeString"

},

{

"name": "id",

"type": "database.TypeInteger"

},

{

"name": "breed",

"type": "database.TypeString"

},

{

"name": "owner\_id",

"type": "database.TypeInteger"

},

{

"name": "name",

"type": "database.TypeString"

},

{

"name": "weight",

"type": "database.TypeReal"

},

{

"name": "license",

"type": "database.TypeChar"

}

],

"values": [

[

"bobik",

1,

"doggo",

2,

"maksym",

69,

82

],

[

"juja",

2,

"spaniel",

3,

"kira",

47.7,

81

],

[

"sobakus",

3,

"bulldog",

1,

"oleg",

123,

72

]

]

}

#### POST new owner

POST http://localhost:1323/databases/animals/owners/new\_row

HTTP/1.1 201 Created

Content-Type: application/json; charset=UTF-8

Vary: Origin

Date: Thu, 01 Dec 2022 23:02:27 GMT

Content-Length: 25

[

"Andrii",

"4",

"110",

"G"

]

**DELETE 5th row in table owners**

DELETE http://localhost:1323/databases/animals/owners/5

HTTP/1.1 200 OK

Content-Type: text/plain; charset=UTF-8

Vary: Origin

Date: Thu, 01 Dec 2022 23:05:36 GMT

Content-Length: 7

deleted

Response code: 200 (OK); Time: 3ms (3 ms); Content length: 7 bytes (7 B)

#### PUT edit owner 4

PUT http://localhost:1323/databases/animals/owners/4

HTTP/1.1 200 OK

Content-Type: text/plain; charset=UTF-8

Vary: Origin

Date: Thu, 01 Dec 2022 23:13:05 GMT

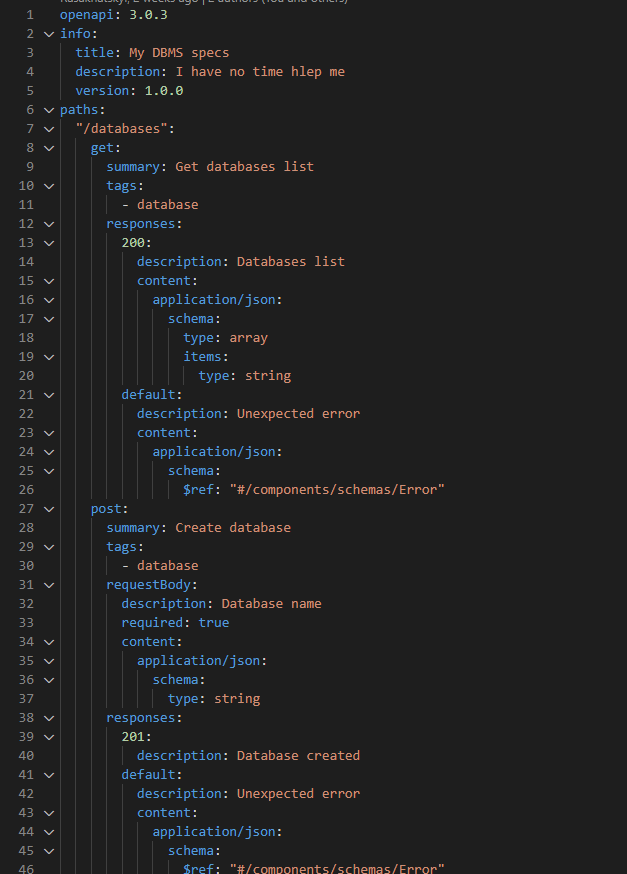
Content-Length: 8

modified

* 1. **REST web-сервіси. Розробка OpenAPI Specification для взаємодії з ієрархічними даними (база, таблиця, ...).**

Було розроблено OpenAPI Specification, файл DBMS/spec.yaml

Фрагмент реалізації



* 1. **13) REST web-сервіси. Реалізація серверного проєкту, використовуючи кодогенерацію стабу за OpenAPI Specification.**

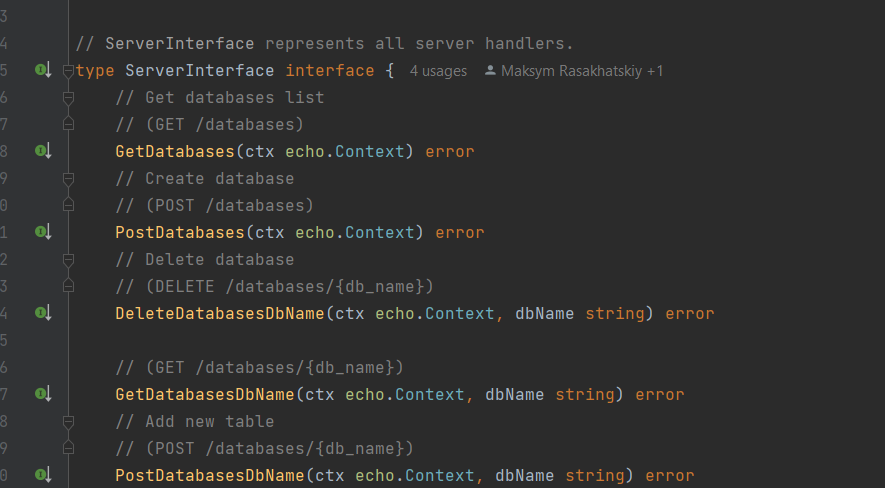
Генерація серверного Golang коду виконана за допомогою [oapi-codegen](https://github.com/deepmap/oapi-codegen)

Команда генерації

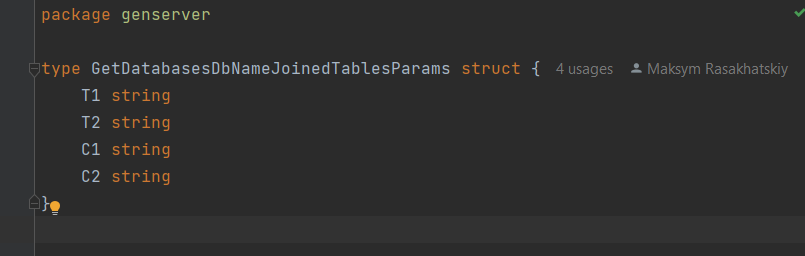
oapi-codegen -generate server -package genserver .\spec.yaml > .\genserver\my.gen.go

Генерований код міститься в файлі /genserver/my.gen.go

Фрагмент генерованого коду

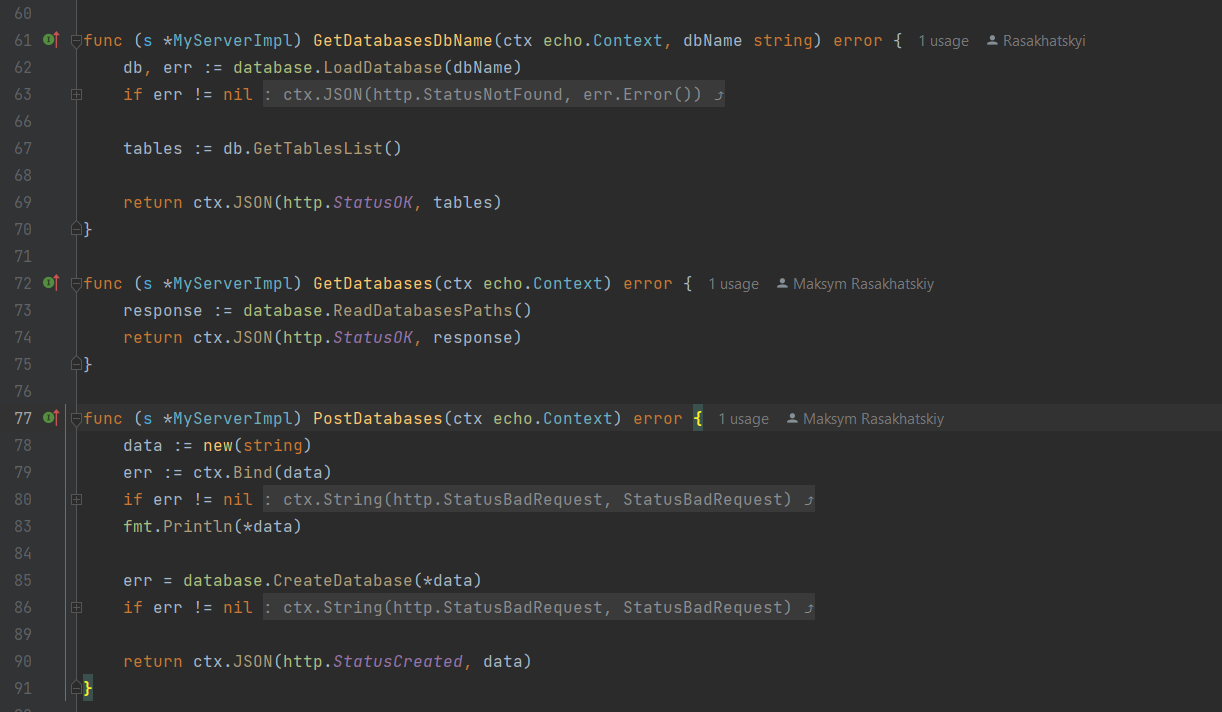


Допоміжні типи створені вручну для передачі параметру в запиті /genserver/utils.go



Імплементація серверних методів в файлі /serverImpl/impl.go

Фрагмент коду



* 1. **14) REST web-сервіси. Реалізація клієнтського проєкту за OpenAPI Specification.**

Так як веб інтерфейс та CLI були розроблені до генерації, вони були згодом переналаштовані на генерований код та його імплементації.

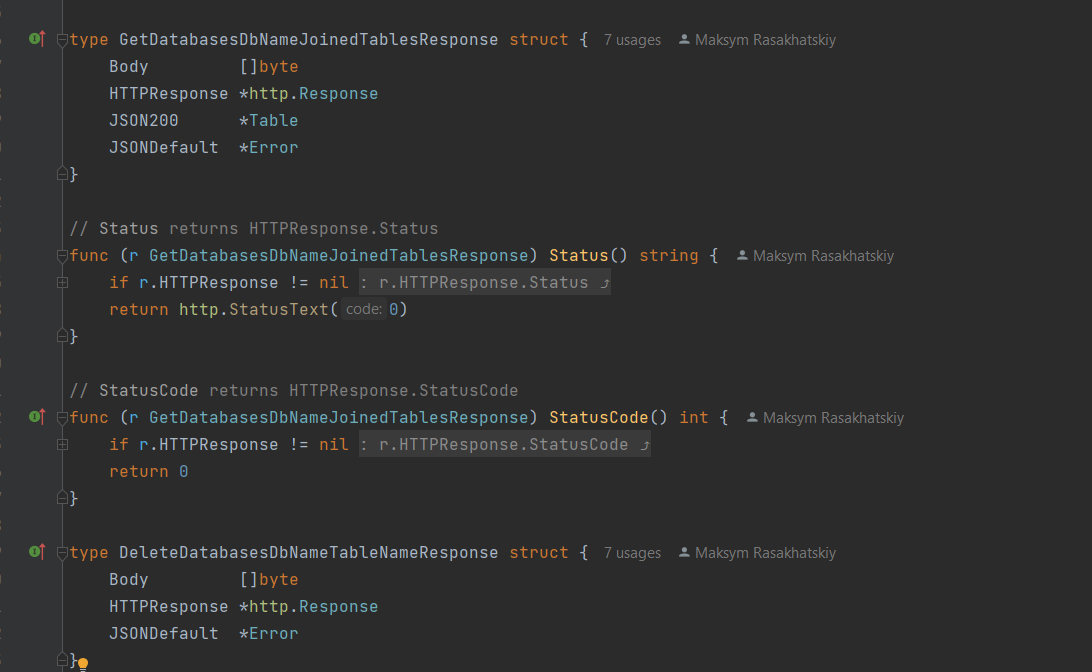
Генерація клієнтського Golang коду виконана за допомогою [oapi-codegen](https://github.com/deepmap/oapi-codegen)

Команда для генерації

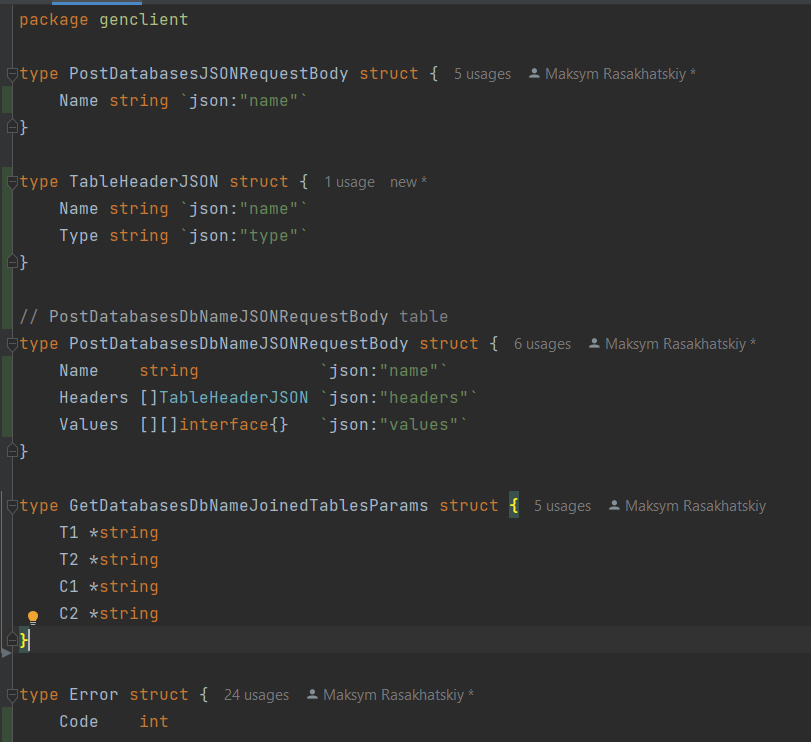
oapi-codegen -generate client -package genclient .\spec.yaml > .\genclient\my.gen.go

Генерований код в файлі /genclient/my.gen.go

Фрагмент генерованого коду



Допоміжні типи створені вручну в файлі /genclient/utils.go



Імплементація клієнтських методів в файлі / genImpl/impl.go

Фрагмент коду



* 1. **18-19) Один або два варіанти Web-проектів. (AspNetWebApi, ASP .NET, ASP .NET MVC, WPF, JSP, JavaServlet, Spring, Struts, Struts 2, JSF, Tapestry, Wicket, GWT тощо). Кожен проект може бути функціонально обмеженим (\*).**

Бекенд написаний використовуючи пакет [Echo](https://echo.labstack.com/) для golang

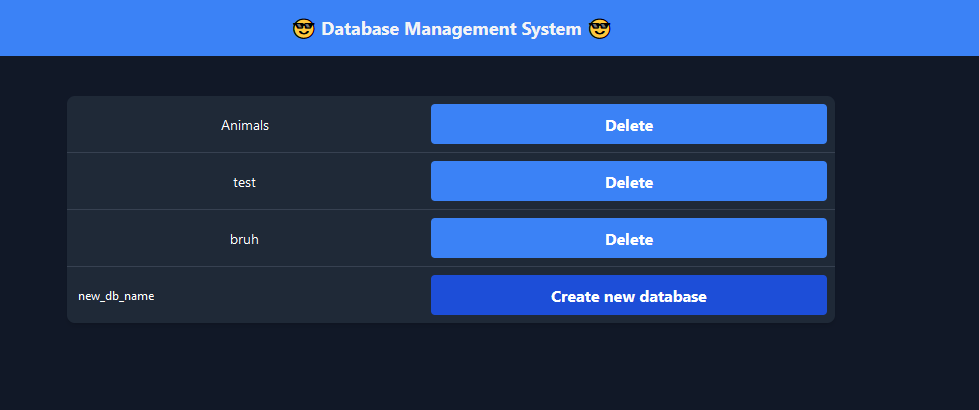
Інтерфейс побудовано за допомогою [Svelte Kit](https://kit.svelte.dev/)

Використано компоненти [tailwind](https://tailwindcss.com/)

Тулінг - [Vite](https://github.com/vitejs/vite)

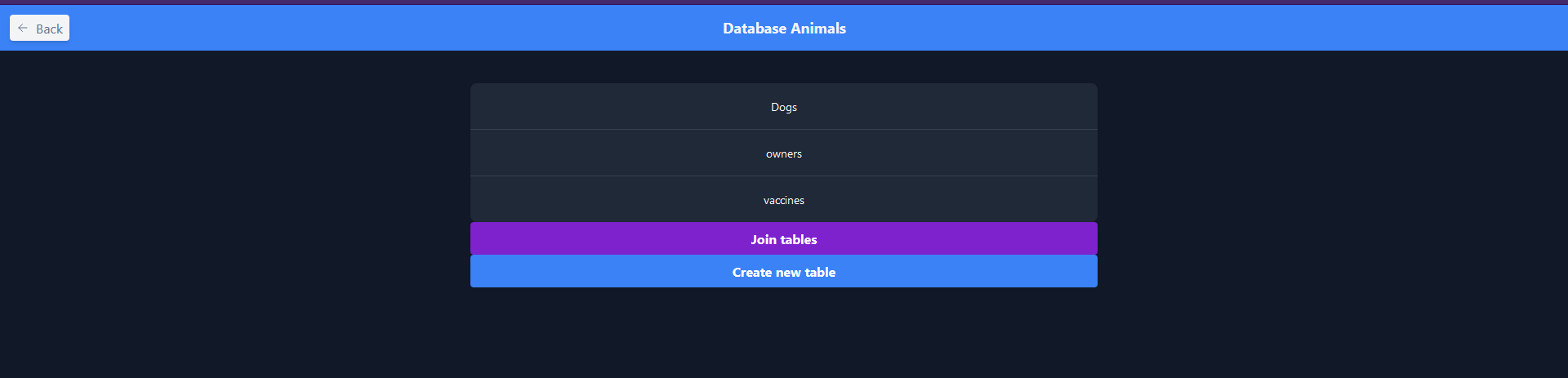
Список баз даних  
Доступні операції

* Створення бази даних
* Видалення бази даних



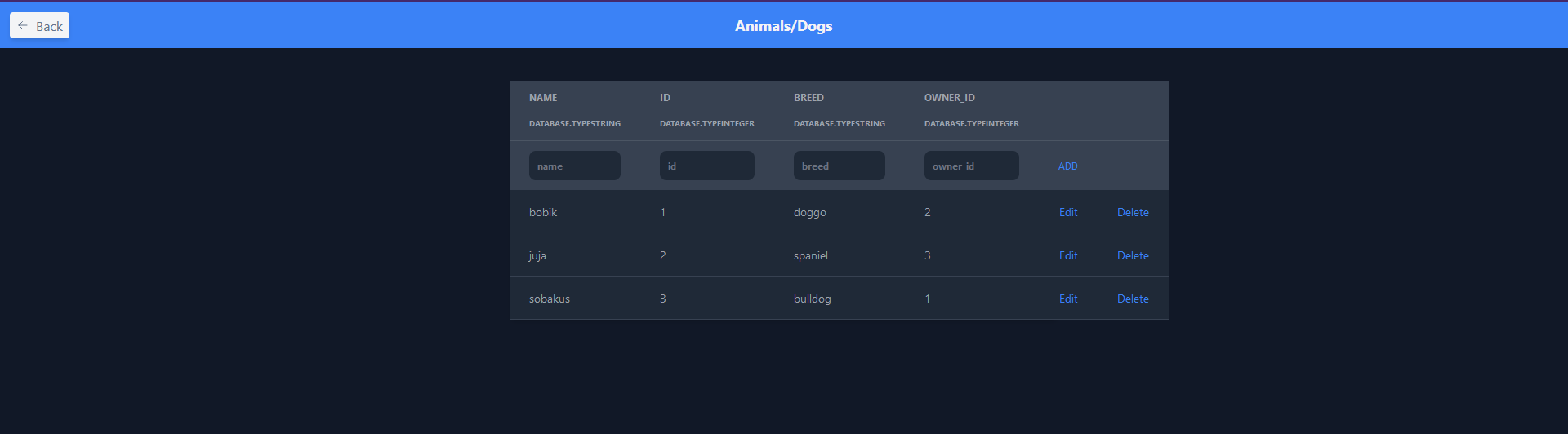
Список таблиць в базі даних Доступні операції

* Створити таблицю
* Join Tables

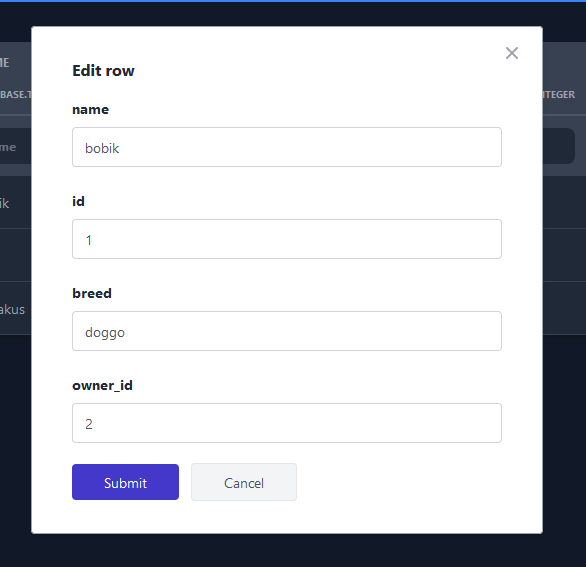


Таблиця бази даних Доступні операції

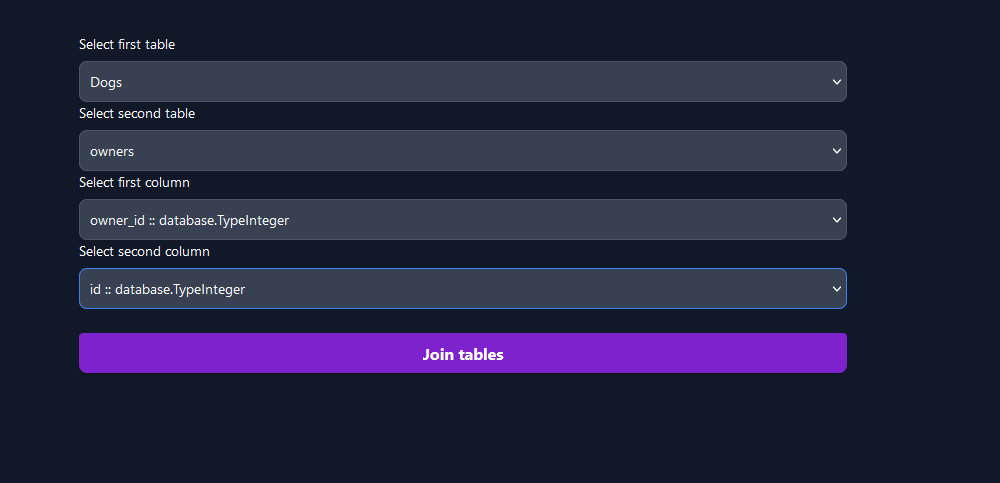
* Додати рядок
* Редагувати рядок
* Видалити рядок



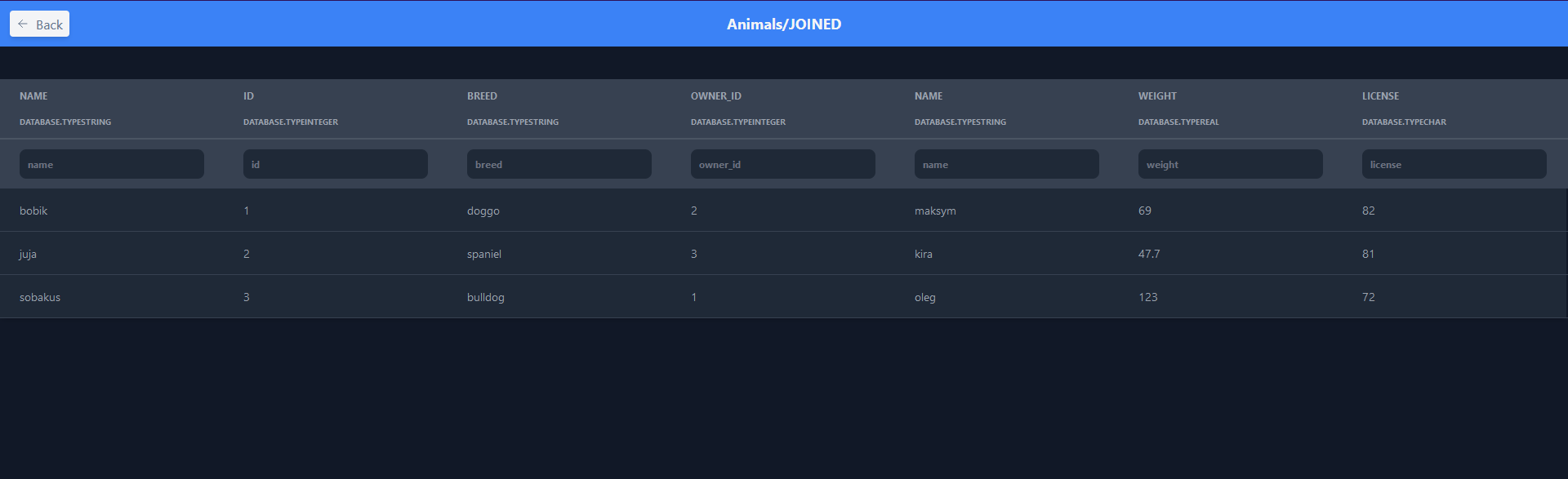
Вікно редагування рядка



Вікно операції Join Tables



Результат операції Join Tables



1. Посилання на GitHub репозиторій

https://github.com/Rasakhatskiy/Labs\_S7\_IT