Київський національний університет імені Тараса Шевченка

Факультет комп’ютерних наук та кібернетики

Кафедра математичної інформатики

Звіт

з практичного завдання

«Фрагментарна реалізація системи

управління табличними базами даних»

(Модуль 2)

студента 4 курсу групи МІ-4

Скоробогатька Ігоря Олександровича

Київ - 2023

**Загальні вимоги**

Основні 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.

Для полів у таблицях треба забезпечити підтримку таких типів:

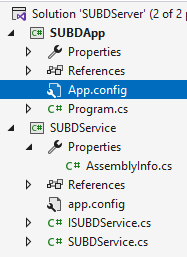
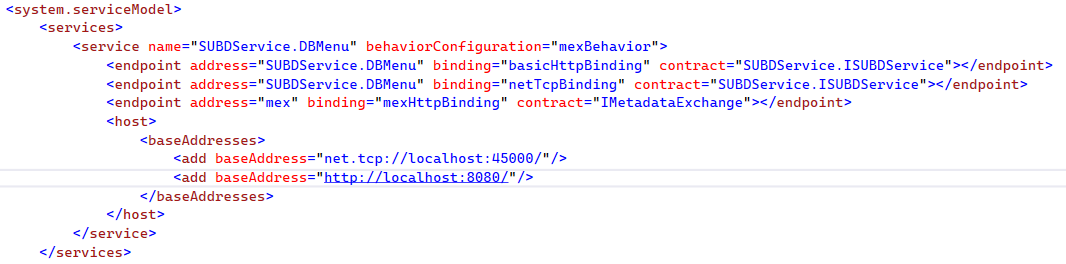
* integer;
* real;
* char;
* string.
* сolor (RGB код кольору)
* colorInvl

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 зчитування її з диску.
* об'єднання таблиць

**Клієнт-серверна версія СУБД**

Для виконання етапу було використано технологію віддаленої взаємодії WCF. Відповідно до логіки роботи, клас системи управління фактично став серверним додатком, а форма з графічним інтерфейсом – клієнтським.

Сервер являє собою основний консольний додаток, що запускає сервіс за вказаними у конфігурації параметрами, та сама реалізація сервісу:

[ServiceContract]

public interface ISUBDService

{

[OperationContract]

void OpenNew();

[OperationContract]

int OpenBase(string Path);

[OperationContract]

bool BaseOpened();

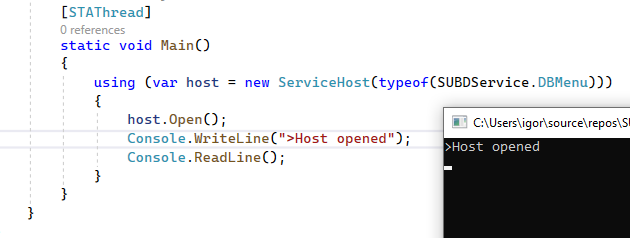
[OperationContract]

bool TableOpened();

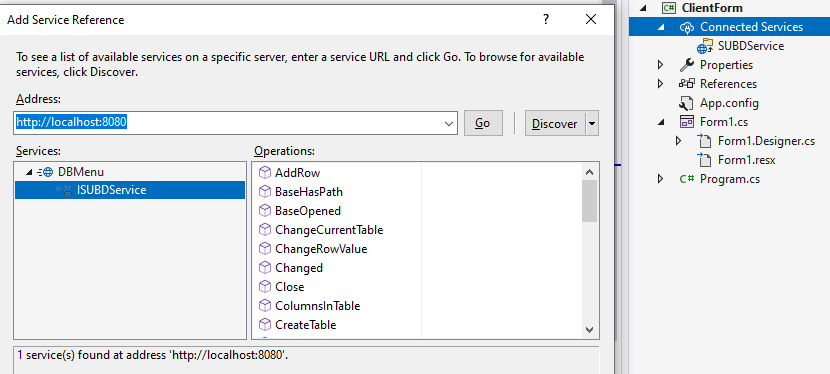
[OperationContract]

… … …

У реалізації сервісу створено потрібний технології інтерфейс сервісу та його імплементація на основі нерозподіленої версії.



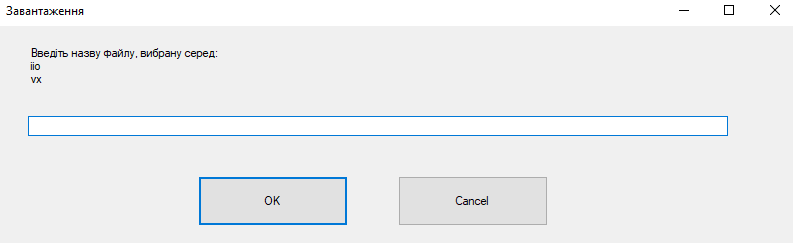
Відкриття хосту за вказаним у конфігурації портом оголошується у консолі.



До сервісу підключаємося за потрібним портом у клієнтській формі.

SUBDService.ISUBDService menu = new SUBDService.SUBDServiceClient("BasicHttpBinding\_ISUBDService");

У використанні методів класу системи управління нічого не змінюється, адже достатньо таким чином створити клієнт замість системи управління.



Проте тепер у клієнтській формі не вийде використати звичайну форму вибору файлу збереження та завантаження. Назву файлу потрібно вводити самостійно, а у випадку завантаження сервер передасть назви файлів, які у нього збережені.

**Web-service версія СУБД**

На цьому етапі для клієнтьської частини на .Net створено ASMX web-сервіс та web-сервіс на Java.

public class DBMenu : System.Web.Services.WebService

{

public static Base CurrentBase;

public static bool IsChanged = false;

public static Table CurrentTable;

[WebMethod]

public void OpenNew()

{

CurrentBase = new Base();

CurrentTable = null;

IsChanged = true;

}

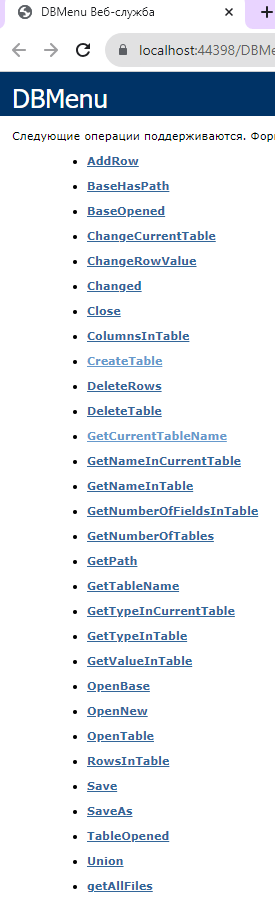
[WebMethod]

public int OpenBase(string Path)

… … …

ASMX web-сервіс визначений класом системи управління, що імплементує web-сервіс та потрібним чином позначає свої методи.

При запуску бачимо у браузері таке:

 .

Агалогічним попередньому етапу чином підключаємо web-сервіс та створюємо клієнт у клієнтській формі.

DBMenuSoapClient menu = new DBMenuSoapClient();

Тоді всі інші частини форми не змінюються у порівнянням з WCF.

Для java оголосимо такий інтерфейс web-сервісу:

import javax.jws.WebService;

import javax.jws.soap.SOAPBinding;

import java.util.ArrayList;

import javax.jws.WebMethod;

@WebService

@SOAPBinding(style = SOAPBinding.Style.DOCUMENT)

public interface interfaceq {

@WebMethod

public void OpenNew();

@WebMethod

public int OpenBase(String Path);

@WebMethod

public boolean BaseOpened();

… … …

Та його імплементацію, що фактично є перекладом з C# на java:

@WebService(endpointInterface = "com.example.interfaceq")

@SOAPBinding(style = SOAPBinding.Style.DOCUMENT)

public class serviceimpl implements interfaceq {

    public static Base CurrentBase;

    public static boolean IsChanged = false;

    public static Table CurrentTable;

@WebMethod

public void OpenNew()

{

    CurrentBase = new Base();

    CurrentTable = null;

    IsChanged = true;

}

@WebMethod

public int OpenBase(String Path)

… … …

Запуск:

public static void main(String[] args){

        Endpoint.publish("http://localhost:45000/q", new serviceimpl());

    }

Та створення клієнту для тієї ж форми на .Net після підключення до цієї ардеси із вказанням ?wsdl.

SUBDService.interfaceqClient menu = new SUBDService.interfaceqClient();

За винятком незначних проблем з перекладом між мовами, клієнт без змін переключився на сервіс іншої платформи.

**REST**

У проекті ASP.NET Web API створимо свій контролер, який замінить об’єкт системи управління.

public class DefaultController : ApiController

{

static List<Base> Bases = new List<Base>();

[HttpPut]

[Route("{BaseName}/{TableName}/Union/{first}/{second}")]

public IHttpActionResult Union(string BaseName,string TableName,string first,string second)

… … …

Із вказанням типу запиту та шляху перенесемо всі потрібні методи. Замінимо обробку невірної поведінки клієнта поверненням BadRequest із вказанням помилки та будемо повертати разом із правильною відповіддю посилання для інших запитів, що можуть цікавити користувача, як це потрібно в HATEOAS. Наприклад, для додавання таблиці маємо:

[HttpPut]

[Route("{BaseName}/{TableName}/Add")]

public IHttpActionResult TableAdd(string BaseName,string TableName)

{

foreach(Base b in Bases)

{

if (b.Name.Equals(BaseName))

{

foreach(Table t in b.Tables)

{

if (t.Name.Equals(TableName)){

if (t.Fields.Count == 0) return BadRequest("There are no fields in the table");

t.AddRow();

Dictionary<string, string> Links = new Dictionary<string, string>();

Links.Add("GET|GetBase", "https://localhost:44349/" + BaseName);

Links.Add("GET|GetTable", "https://localhost:44349/" + BaseName + "/" + TableName);

Links.Add("PUT|AddRow", "https://localhost:44349/" + BaseName + "/" + TableName + "/Add");

Links.Add("PUT|DeleteRow", "https://localhost:44349/" + BaseName + "/" + TableName + "/Delete/{posistion}/{number}");

Links.Add("POST|AddField", "https://localhost:44349/" + BaseName + "/" + TableName + "/{FieldName}?type={Type}");

Links.Add("DELETE|DeleteField", "https://localhost:44349/" + BaseName + "/" + TableName + "/{FieldName}");

Links.Add("GET|GetField", "https://localhost:44349/" + BaseName + "/" + TableName + "/{FieldName}");

return Ok(Links);

}

}

return BadRequest("There is no table with this name");

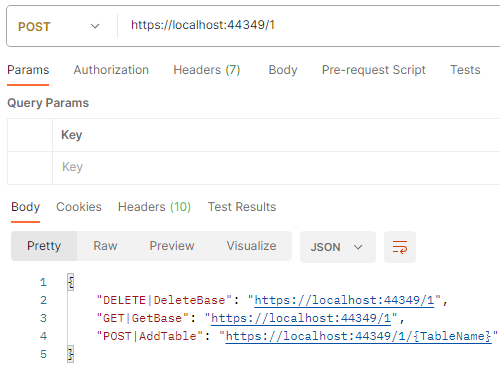
}

}

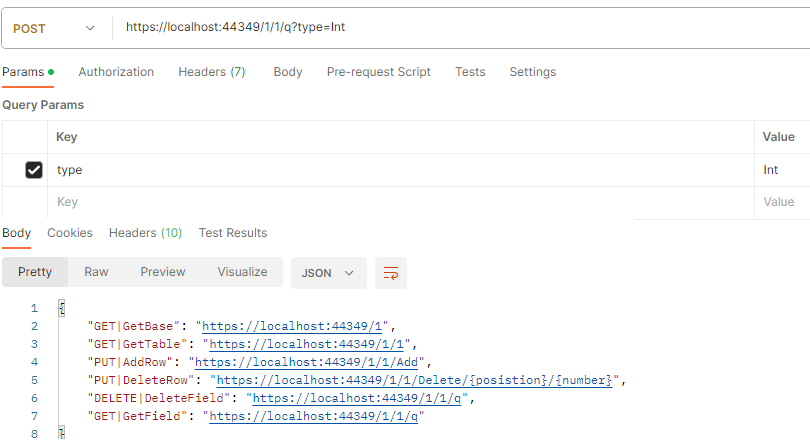
return BadRequest("There is no base with this name");

}

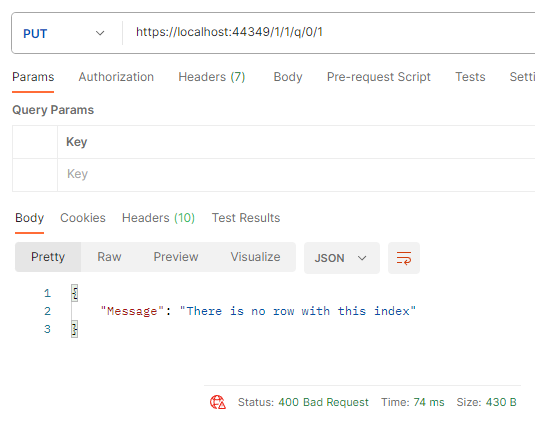
Додавання бази 1 у Postman із поверненням посилань на її видалення, отримання та додачу в ній таблиці виглядає так:



Для створення поля q типу Int у таблиці 1 бази 1 потрібно вказати цей тип у параметрах:



Якщо ж допустити помилку, і змінювати значення поля у недоданому рядку, то отримаємо 400 Bad Request із вказанням помилки.



**GraphQL APІ**

Із встановленням HotChocolate для ASP.NET додамо GraphQLServer до web-додатку із використанням потрібних нам методів у Query та Mutation:

var builder = WebApplication.CreateBuilder(args);

builder.Services

.AddGraphQLServer()

.AddMutationType<Mutation>()

.AddQueryType<Query>();

var app = builder.Build();

Table t = new Table("t1");

t.Fields.Add(new IntField("int1"));

Base b = new Base("B1");

b.Tables.Add(t);

Syst.Bases.Add(b);

app.MapGraphQL();

app.Run();

У Query визначимо початкові Get-запити, що повернуть множину баз чи об’єкт бази за її іменем. Обробка помилок виконується за допомогою кидання GraphQLException із вказанням помилки:

public class Query

{

public IEnumerable<Base> GetBases()

{

return Syst.Bases;

}

public Base GetBaseByName(string name) {

foreach(Base b in Syst.Bases)

{

if(b.Name.Equals(name))

{

return b;

}

}

throw new GraphQLException(new Error("There is no base with this name"));

}

}

Надалі GraphQL дозволяє користувачу отримати Get-доступ до всіх полів об’єктів-баз та Get-методу отримання таблиці за іменем. Це дозволяє транзитивно отримати повну інформацію про бази, таблиці, поля та значення і довільним чином деталізовувати її.

public class Base

{

public List<Table> Tables { get; set; } = new List<Table>();

public string Name { get; set; }

public Base(string n)

{

Name = n;

}

public Table GetTableByName(string name)

{

foreach (Table t in Tables)

{

if (t.Name.Equals(name))

{

return t;

}

}

throw new GraphQLException(new Error("Table not found"));

return new Table("NO TABLE WITH THIS NAME");

}

}

Для Mutation кожен метод є похідним від Post, Put, Delete-методів з REST. Повертається або рядок Ok, або помилка з текстом. Наприклад, так:

public string CreateBase(string name)

{

foreach (Base b in Syst.Bases)

{

if (b.Name.Equals(name))

{

throw new GraphQLException(new Error("Base with this name exists"));

}

}

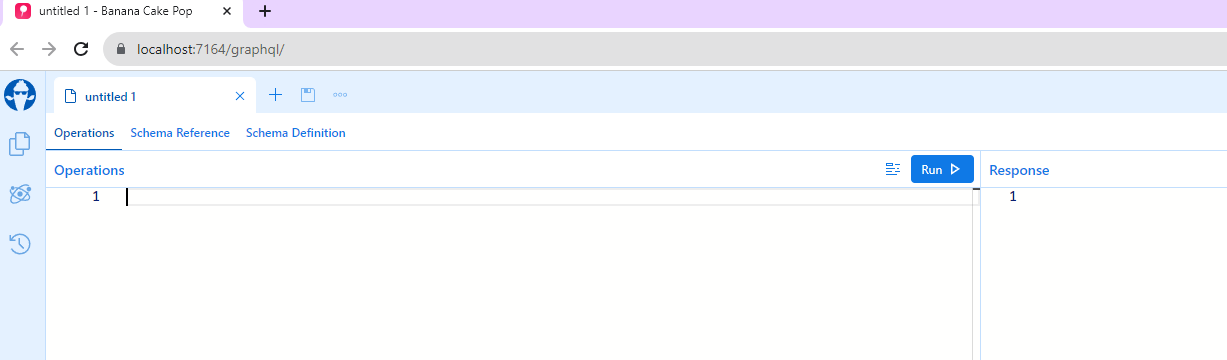
Base ba = new Base(name);

Syst.Bases.Add(ba);

return "Ok";

}

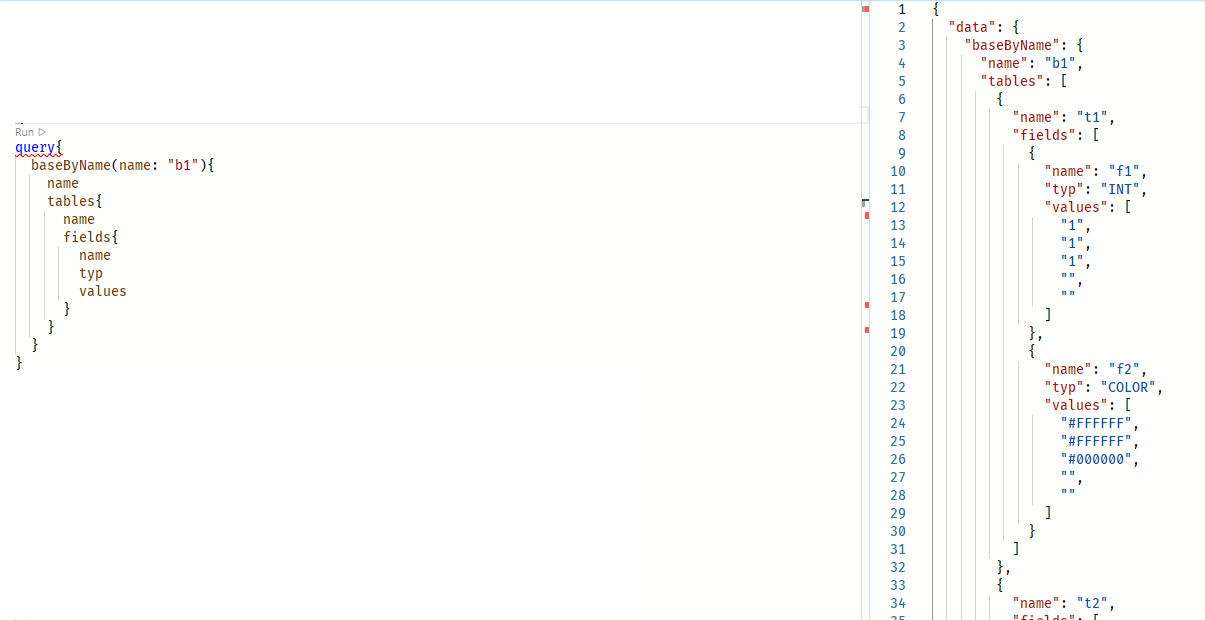
При запуску отримуємо таке вікно GraphQL:



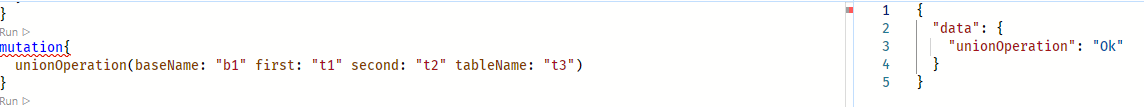
Виконаємо спочатку таке створення бази:



Далі отримаємо повну інформацію про базу b1. Ми могли б, наприклад, не виводити values, а просто перевірити, що у деяких таблиць однакова множина name-typ.



Виконаємо варіантну операцію із створенням таблиці t3:



Та отримаємо тільки цю нову таблицю



**gRPC**

Аналогічно етапу з веб-сервісами створимо одну клієнтську форму та два варіанти сервісу: C# та Java.

У .Net встановимо пакет grpc та додамо у веб-додатку grpc-сервіс:

builder.Services.AddGrpc(o =>

{

o.EnableDetailedErrors = true;

});

var app = builder.Build();

app.MapGrpcService<SUBDService>();

Цей сервіс створений на основі побудованих до цього стабів та реалізовує ті ж самі методи, що і раніше:

public class SUBDService : GrpcService.Protos.SUBDServiceGRPC.SUBDServiceGRPCBase

{

public static Base CurrentBase;

public static bool IsChanged = false;

public static Table CurrentTable;

public override Task<Empty> OpenNew(Empty request, ServerCallContext context)

{

CurrentBase = new Base();

CurrentTable = null;

IsChanged = true;

return Task.FromResult(new Empty());

}

public override Task<oneindex> OpenBase(onlystring request, ServerCallContext context)

{

… … …

Самі стаби побудовані на основі файлу-протобуфера, у якому вказані всі методи, можливі повідомлення та параметри для С# і Java:

syntax="proto3";

import "google/protobuf/empty.proto";

option java\_package="org.example";

option java\_multiple\_files = true;

option csharp\_namespace = "GrpcService.Protos";

service SUBDServiceGRPC{

rpc OpenNew(google.protobuf.Empty) returns (google.protobuf.Empty);

rpc OpenBase(onlystring) returns (oneindex);

rpc BaseOpened(google.protobuf.Empty) returns (logical);

rpc TableOpened(google.protobuf.Empty) returns (logical);

rpc RowsInTable(google.protobuf.Empty) returns (oneindex);

rpc ColumnsInTable(google.protobuf.Empty) returns (oneindex);

rpc GetNameInCurrentTable(oneindex) returns (onlystring);

rpc GetTypeInCurrentTable(oneindex) returns (onlystring);

rpc GetValueInTable(twoindex) returns (onlystring);

rpc GetNumberOfTables(google.protobuf.Empty) returns (oneindex);

rpc GetNumberOfFieldsInTable(oneindex) returns (oneindex);

rpc GetTableName(oneindex) returns (onlystring);

rpc GetCurrentTableName(google.protobuf.Empty) returns (onlystring);

rpc GetNameInTable(twoindex) returns (onlystring);

rpc GetTypeInTable(twoindex) returns (onlystring);

rpc ChangeCurrentTable(oneindex) returns (oneindex);

rpc BaseHasPath(google.protobuf.Empty) returns (logical);

rpc Changed(google.protobuf.Empty) returns (logical);

rpc Save(google.protobuf.Empty) returns (oneindex);

rpc getAllFiles(google.protobuf.Empty) returns (onlystring);

rpc SaveAs(onlystring) returns (oneindex);

rpc CreateTable(table) returns (oneindex);

rpc OpenTable(onlystring) returns (oneindex);

rpc DeleteTable(onlystring) returns (oneindex);

rpc AddRow(google.protobuf.Empty) returns (google.protobuf.Empty);

rpc DeleteRows(twoindex) returns (oneindex);

rpc ChangeRowValue(changevalue) returns (oneindex);

rpc GetPath(google.protobuf.Empty) returns (onlystring);

rpc Union(union) returns (oneindex);

rpc Close(google.protobuf.Empty) returns (google.protobuf.Empty);

}

message onlystring{

string ar=1;

}

message oneindex{

int32 ar=1;

}

message twoindex{

int32 ar1=1;

int32 ar2=2;

}

message table{

string name=1;

repeated string names=2;

repeated string types=3;

}

message changevalue{

int32 i=1;

int32 j=2;

string val=3;

}

message union{

string name=1;

repeated string args=2;

}

message logical{

bool ans=1;

}

Як бачимо у коді, методи сервісу були суттєво змінені, адже тепер треба повертати Task<повідомлення-відповідь> та використовувати параметри повідомлення-запиту, притримуючись всюди формату повідомлень:

public override Task<onlystring> GetNameInCurrentTable(oneindex request, ServerCallContext context)

{

onlystring output = new onlystring();

try

{

output.Ar = CurrentTable.Fields[request.Ar].Name;

return Task.FromResult(output);

}

catch (Exception ex)

{

output.Ar = "ERR";

return Task.FromResult(output);

}

}

Для Java потрібно у Maven додати той самий протобуфер та визначити у файлі конфігурації побудову стабів:

<dependency>  
 <groupId>com.google.protobuf</groupId>  
 <artifactId>protobuf-java</artifactId>  
 <version>3.6.1</version>  
</dependency>  
  
<dependency>  
 <groupId>io.grpc</groupId>  
 <artifactId>grpc-netty-shaded</artifactId>  
 <version>1.15.1</version>  
</dependency>  
<dependency>  
 <groupId>io.grpc</groupId>  
 <artifactId>grpc-protobuf</artifactId>  
 <version>1.15.1</version>  
</dependency>  
<dependency>  
 <groupId>io.grpc</groupId>  
 <artifactId>grpc-stub</artifactId>  
 <version>1.15.1</version>  
</dependency>

Та

<inputDirectories>  
 <include>src/main/resources</include>  
</inputDirectories>  
  
<outputTargets>  
 <outputTarget>  
 <type>java</type>  
 <outputDirectory>src/main/java</outputDirectory>  
 </outputTarget>  
 <outputTarget>  
 <type>grpc-java</type>  
 <pluginArtifact>io.grpc:protoc-gen-grpc-java:1.15.0</pluginArtifact>  
 <outputDirectory>src/main/java</outputDirectory>  
 </outputTarget>  
</outputTargets>

Запускаємо сервіс:

Server server= ServerBuilder.*forPort*(9090).addService( new SUBD() ).build();

Перекладемо версію на C# на Java

public class SUBD extends SUBDServiceGRPCGrpc.SUBDServiceGRPCImplBase {  
 public static Base *CurrentBase*;  
 public static boolean *IsChanged* = false;  
 public static Table *CurrentTable*;  
 oneindex.Builder a=oneindex.*newBuilder*().setAr(1);  
 @Override  
 public void openNew(Empty request, StreamObserver<Empty> responseObserver) {  
 Empty.Builder res=Empty.*newBuilder*();  
 *CurrentBase* = new Base();  
 *CurrentTable* = null;  
 *IsChanged* = true;  
  
 responseObserver.onNext(res.build());  
 responseObserver.onCompleted();  
 }  
  
 @Override  
 public void openBase(onlystring request, StreamObserver<oneindex> responseObserver) {

… … …

Відповідно реалізації grpc у Java, потрібно будувати відповіді на основі використовуваних типів повідомлень та отримувати параметри з запиту:

@Override  
public void getNameInCurrentTable(oneindex request, StreamObserver<onlystring> responseObserver) {  
 onlystring.Builder res=onlystring.*newBuilder*();  
 try  
 {  
 res.setAr(*CurrentTable*.Fields.get(request.getAr()).Name);  
 responseObserver.onNext(res.build());  
 responseObserver.onCompleted();  
 }  
 catch (Exception ex)  
 {  
 res.setAr("ERR");  
 responseObserver.onNext(res.build());  
 responseObserver.onCompleted();  
 }  
}

Клієнтська форма так само будує свої стаби та створює grpc-канал по потрібному порту залежно від того, потрібні Java чи C#:

bool Java = true;

static GrpcChannel channel = GrpcChannel.ForAddress("http://localhost:9090");

SUBDServiceGRPC.SUBDServiceGRPCClient client = new SUBDServiceGRPC.SUBDServiceGRPCClient(channel);

У іншому робота форми не змінюється, проте замість виклику звичайних методів потрібно створити повідомлення-запит, викликати метод та розібрати повідомлення-відповідь, як тут для отримання значення в таблиці за індексами:

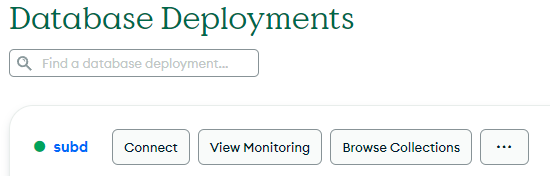
twoindex arg2 = new twoindex { Ar1 = j, Ar2 = i - 1 };

var reply5 = client.GetValueInTable(arg2);

Values[i].Add(reply5.Ar);

**Вибірковий блок**

Було обрано варіант проєкту із використанням СУБД Mongo (замість використання серіалізації об'єктів для збереження даних). Зареєструємося на Mongodb та створимо проект:



У початковій нерозподіленій формі зв’яжемося з проектом:

string connectionUri = "mongodb+srv://igor:12345@subd.2e7zo8o.mongodb.net/?retryWrites=true&w=majority";

client = new MongoClient(connectionUri);

Змінимо відкриття за назвою бази таким чином. Отримаємо від серверу базу, її таблиці та трансформуємо таблиці бази без параметру \_id з mongodb у стандартний об’єкт списку таблиць у базі:

public int OpenBase(string Path)

{

try

{

IMongoDatabase db = client.GetDatabase(Path);

IMongoCollection<Table> col= db.GetCollection<Table>("tables");

Base NewBase = new Base();

NewBase.Path= Path;

var filter = Builders<Table>.Filter.Empty;

NewBase.Tables = col.Find(filter).Project<Table>("{\_id: 0}").ToList();

CurrentBase = NewBase;

CurrentTable = null;

IsChanged = false;

return 0;

}

catch (Exception ex)

{

Console.WriteLine(ex.Message);

return 1;

}

}

Також змінимо збереження так, що база за вказаною назвою видаляється та створюється заново напряму за вставкою у колекцію таблиць бази на сервері всієї множини таблиць.

public int SaveAs(string Path)

{

IMongoDatabase db = client.GetDatabase(Path);

client.DropDatabase(Path);

db = client.GetDatabase(Path);

IMongoCollection<Table> col= db.GetCollection<Table>("tables");

if (CurrentBase.Tables.Count > 0)

{

col.InsertMany(CurrentBase.Tables);

}

CurrentBase.Path = Path;

IsChanged = false;

return 0;

}

Крім того, для визначення множини файлів вибору завантаження запитаємо у mongodb множину назв баз, що є на сервері.

public string getAllFiles()

{

try

{

var databases = client.ListDatabaseNames();

List<string> dblist = new List<string>();

databases.ForEachAsync(s => dblist.Add(s));

string ans = String.Join("\n", dblist);

return ans;

}

catch (System.Exception ex)

{

return "ERR";

}

}

Зокрема, збереження бази з таблицею 1 із встановленням назви 1 дасть такий результат(зліва – форма, справа – сайт mongodb).

