**МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ**

**УЧРЕЖДЕНИЕ ОБРАЗОВАНИЯ**

**ГОМЕЛЬСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ ИМЕНИ П. О. СУХОГО**

Факультет автоматизированных и информационных систем

Кафедра «Информатика»

ОТЧЕТ ПО ЛАБОРАТОРНОЙ РАБОТЕ № 1

по дисциплине «Распределённая обработка данных»

на тему: «Взаимодействие приложений на основе протокола TCP/IP»

Выполнил: студент гр. ИП-41

Коваленко А.И.

Принял: преподаватель

Шибеко В.Н.

Гомель 2023

**Цель:** Познакомится с возможностями взаимодействия приложений на основе протоколов TCP. Изучить способы реализации клиента и сервера. Изучить и реализовать способы разделенного доступа к файловым источникам данных.

**Задание 1.1:**

Разработать библиотеку, реализующую для заданной предметной области операции редактирования и выборки данных.

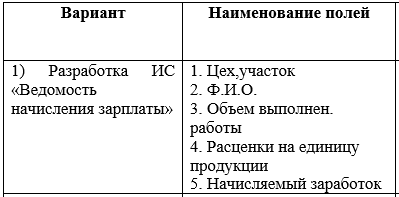
Данные располагаются в файле.

Разработать приложение – клиент для выборки и редактирования данных (CRUD операции) предметной области на основе доступа к серверу и протокола TCP/IP.

**Задание 1.2:**

Разработать средства разделенного доступа к источнику данных при выполнении операций редактирования и выборки. Проверить на основе нескольких параллельно-работающих клиентов.

**Предметная область:** вариант 1



**Реализация:**

TCPClient.cs:

using DataEditLib.Data;

using DataEditLib.Interfaces;

using DataEditLib.Models.MessagesTypes;

using ManagementClient.Interfaces;

using System.Net;

using System.Net.Sockets;

using System.Text;

using System.Text.Json;

namespace ManagementClient.Clients

{

public class TcpClientSocket : IClient

{

private TcpClient \_client;

private StreamReader \_sReader;

private StreamWriter \_sWriter;

private readonly string \_ip;

private readonly int \_port;

private readonly ILogger \_log;

private static ServerMessage? \_messageToGet;

private static ClientMessage? \_messageToSend;

public TcpClientSocket(ILogger log)

{

\_ip = ProjectProperties.TcpClientServerIp;

\_port = ProjectProperties.TcpClientServerPort;

\_log = log;

}

public TcpClientSocket(int port, string ip, ILogger log)

{

\_ip = ip;

\_port = port;

\_log = log;

}

// Making new thread to connect to server

private void StartConnection()

{

try

{

\_client = new TcpClient();

\_client.Connect(\_ip, \_port);

\_messageToSend.IpAddress = ((IPEndPoint)\_client.Client.RemoteEndPoint).Address.ToString();

\_messageToSend.Port = ((IPEndPoint)\_client.Client.RemoteEndPoint).Port;

HandleCommunication();

//Thread t = new Thread(new ThreadStart(HandleCommunication));

//t.Start();

}

catch (Exception ex)

{

Console.WriteLine(ex.ToString());

}

}

public void HandleCommunication()

{

if (\_messageToSend != null)

{

// Creation I/O streams

Task.Delay(10);

\_sReader = new StreamReader(\_client.GetStream(), Encoding.ASCII);

Task.Delay(10);

\_sWriter = new StreamWriter(\_client.GetStream(), Encoding.ASCII);

// Preparing to send message at server

PrepareToSend();

// Sending message as CVS string and getting info about request; cloasing stream

\_sWriter.WriteLine(JsonSerializer.Serialize(\_messageToSend));

\_log.Log(\_messageToSend, DataEditLib.Enums.SenderType.Client);

\_sWriter.Flush();

// Waiting server response and getting info about response

var jsonMessage = \_sReader.ReadLine();

\_messageToGet = new ServerMessage();

\_messageToGet = JsonSerializer.Deserialize<ServerMessage>(jsonMessage);

\_log.Log(\_messageToGet, DataEditLib.Enums.SenderType.Client);

// Cloasing all

Console.WriteLine();

\_sWriter.Close();

\_client.Close();

Clear();

}

}

// Clearing messages variables

private void Clear()

{

\_messageToGet = null;

\_messageToSend = null;

}

private void PrepareToSend()

{

if (\_messageToSend != null)

{

\_messageToSend.Port = \_port;

\_messageToSend.IpAddress = \_ip;

}

}

// Send message method

public void SendMessage(ClientMessage message)

{

\_messageToSend = message;

\_messageToSend.SenderType = DataEditLib.Enums.SenderType.Client;

StartConnection();

}

}

}

TCPServer.cs

using DataEditLib.Data;

using DataEditLib.Interfaces;

using DataEditLib.Models.MessagesTypes;

using ManagementServer.Interfaces;

using System.Net;

using System.Net.Sockets;

using System.Text;

using System.Text.Json;

namespace ManagementServer.Servers

{

public class TcpServerSocket : IServer

{

private TcpListener \_server;

private bool \_isRunning;

private readonly string \_ip;

private readonly int \_port;

private readonly IDataEdit \_data;

private readonly ILogger \_log;

private static ServerMessage? \_messageToSend;

private static ClientMessage? \_messageToGet;

public TcpServerSocket(IDataEdit dataEditService, ILogger log)

{;

\_ip = ProjectProperties.TcpClientServerIp;

\_port = ProjectProperties.TcpClientServerPort;

\_data = dataEditService;

\_log = log;

}

public TcpServerSocket(string ip, int port, IDataEdit dataEditService, ILogger log)

{

\_ip = ip;

\_port = port;

\_data = dataEditService;

\_log = log;

}

public void Start()

{

\_server = new TcpListener(

new IPAddress(FromStringToBytesIp(\_ip)), \_port);

\_isRunning = true;

\_server.Start();

Thread th = new Thread(Listen);

th.Start();

}

public byte[] FromStringToBytesIp(string ip) => ip.Split('.').Select(byte.Parse).ToArray();

private void Listen()

{

while (\_isRunning)

{

try

{

// Waiting client

TcpClient newClient = \_server.AcceptTcpClient();

if (newClient != null)

{

Thread t = new Thread(new ParameterizedThreadStart(HandleClient));

t.Start(newClient);

}

}

catch (Exception ex)

{

\_log.Log(ex);

}

}

}

public async void HandleClient(object obj)

{

// [Need boxing for threads]

TcpClient client = (TcpClient)obj;

// Making I/O streams

StreamWriter \_sWriter = new StreamWriter(client.GetStream(), Encoding.ASCII);

StreamReader \_sReader = new StreamReader(client.GetStream(), Encoding.ASCII);

// Getting request from client (as json string)

var jsonMessage = \_sReader.ReadLine();

\_messageToGet = new ClientMessage();

\_messageToGet = JsonSerializer.Deserialize<ClientMessage>(jsonMessage);

\_log.Log(\_messageToGet, DataEditLib.Enums.SenderType.Server);

// Define operation and sending response

\_messageToSend = await \_data.ParseMessageType(\_messageToGet);

\_sWriter.WriteLine(JsonSerializer.Serialize(\_messageToSend));

\_log.Log(\_messageToSend, DataEditLib.Enums.SenderType.Server);

\_sWriter.Flush();

Clear();

}

public void Pause() => \_isRunning = false;

private void Clear()

{

\_messageToSend = null;

\_messageToGet = null;

}

}

}

MyEntity.cs

using DataEditLib.Interfaces;

namespace DataEditLib.Models

{

public class MyEntity : IEntity

{

public MyEntity() { }

public MyEntity(int id, string name, double scopeOfWork, double unitPrice, double accruedEarnings)

{

Id = id;

Name = name;

ScopeOfWork = scopeOfWork;

UnitPrice = unitPrice;

AccruedEarnings = accruedEarnings;

}

public int Id { get; set; }

public string? Name { get; set; }

public double? ScopeOfWork { get; set; }

public double? UnitPrice { get; set; }

public double? AccruedEarnings { get; set; }

public override string ToString()

{

return Id.ToString() + ',' +

Name.ToString() + ',' +

ScopeOfWork.ToString() + ',' +

UnitPrice.ToString() + ',' +

AccruedEarnings.ToString() + ';';

}

public IEntity ToObjectFromText(string info)

{

info = info.Replace("\r\n", string.Empty).Trim(';');

var props = info.Split(',');

Id = int.Parse(props[0]);

Name = props[1];

ScopeOfWork = double.Parse(props[2]);

UnitPrice = double.Parse(props[3]);

AccruedEarnings = double.Parse(props[4]);

return this;

}

public void UpdateCurrent(IEntity newEntity)

{

var tmp = newEntity as MyEntity;

this.Name = tmp.Name;

this.AccruedEarnings = tmp.AccruedEarnings;

this.ScopeOfWork = tmp.ScopeOfWork;

this.UnitPrice = tmp.UnitPrice;

}

}

}

CsvDataHandler.cs

using DataEditLib.Enums;

using DataEditLib.Interfaces;

using DataEditLib.Interfaces.Crud;

using DataEditLib.Models;

using DataEditLib.Models.MessagesTypes;

namespace DataEditLib.Data

{

public class CsvDataHanler<T>

: IDataEdit, IRead<T>, ICreate<T>, IUpdate<T>, IDelete<T> where T : MyEntity, new()

{

public async Task<IActionResult<T>> Create(ClientMessage message)

{

try

{

var objs = await InsertCurrentData(message);

await File.WriteAllLinesAsync(ProjectProperties.TextFilePath, objs);

return new ActionResult<T>(message, MessageStatus.Ok, ProjectProperties.OkCreatedInfo);

}

catch (Exception ex)

{

return new ActionResult<T>(message, MessageStatus.Error, ex);

}

}

public async Task<ServerMessage> ParseMessageType(ClientMessage message)

{

IActionResult<T> actionResult;

try

{

if (message == null)

throw new ArgumentNullException();

if (!File.Exists(ProjectProperties.TextFilePath))

{

File.Create(ProjectProperties.TextFilePath);

}

if (message.MsgType == MessageType.Create)

actionResult = await Create(message);

else if (message.MsgType == MessageType.ReadOne)

actionResult = await ReadOne(message);

else if (message.MsgType == MessageType.ReadAll)

actionResult = await ReadAll(message);

else if (message.MsgType == MessageType.Update)

actionResult = await Update(message);

else if (message.MsgType == MessageType.Delete)

actionResult = await Delete(message);

else

throw new ArgumentException(ProjectProperties.MessageTypeNotDefine);

return ServerMessage.ServerResponse((IActionResult<MyEntity>)actionResult, message);

}

catch (Exception ex)

{

actionResult = new ActionResult<T>(message, MessageStatus.Error, ex);

return ServerMessage.ServerResponse((IActionResult<MyEntity>)actionResult, message);

}

}

public async Task<IActionResult<T>> ReadOne(ClientMessage message)

{

try

{

var text = (await File.ReadAllTextAsync(ProjectProperties.TextFilePath));

var collection = ToCollection(text);

if (int.TryParse(message.Value, out int id))

foreach (var item in collection)

if (item.Id == id)

return new ActionResult<T>(message, MessageStatus.Ok, item.ToString());

throw new KeyNotFoundException();

}

catch (Exception ex)

{

return new ActionResult<T>(message, MessageStatus.Error, ex);

}

}

public async Task<IActionResult<T>> ReadAll(ClientMessage message)

{

try

{

var text = await File.ReadAllTextAsync(ProjectProperties.TextFilePath);

return new ActionResult<T>(message, MessageStatus.Ok, text);

}

catch (IOException ex)

{

return new ActionResult<T>(message, MessageStatus.Error, ex);

}

}

public async Task<IActionResult<T>> Update(ClientMessage message)

{

try

{

var text = (await File.ReadAllTextAsync(ProjectProperties.TextFilePath));

var collection = ToCollection(text);

if (int.TryParse(message.Value.Split(',')[0], out int id))

{

foreach (var item in collection)

{

if (item.Id == id)

{

var tmp = new T();

item.UpdateCurrent(tmp.ToObjectFromText(message.Value));

await RewriteBase(collection);

return new ActionResult<T>(message, MessageStatus.Ok, ProjectProperties.OkUpdatedInfo);

}

}

}

throw new KeyNotFoundException();

}

catch (Exception ex)

{

return new ActionResult<T>(message, MessageStatus.Error, ex);

}

}

public async Task<IActionResult<T>> Delete(ClientMessage message)

{

try

{

var text = (await File.ReadAllTextAsync(ProjectProperties.TextFilePath));

var collection = ToCollection(text);

if (int.TryParse(message.Value.Split(',')[0], out int id))

foreach (var item in collection)

{

if (item.Id == id)

{

(collection as List<T>).Remove(item);

await RewriteBase(collection);

return new ActionResult<T>(message, MessageStatus.Ok, ProjectProperties.OkDeletedInfo);

}

}

throw new KeyNotFoundException();

}

catch (Exception ex)

{

return new ActionResult<T>(message, MessageStatus.Error, ex);

}

}

private IEnumerable<T> ToCollection(string line)

{

var list = new List<T>();

var lines = line.Replace("\r\n", string.Empty)

.Replace("\r\n", string.Empty)

.Split(';')

.Where(x => x != string.Empty)

.Select(x => x += ";").ToArray();

for (int i = 0; i < lines.Count(); i++)

list.Add((T)new T().ToObjectFromText(lines.ElementAt(i)));

return list;

}

private async Task RewriteBase(IEnumerable<T> data)

{

File.Delete(ProjectProperties.TextFilePath);

var list = new List<string>();

foreach (var item in data)

list.Add(item.ToString());

await File.WriteAllLinesAsync(ProjectProperties.TextFilePath, list);

}

private async Task<string[]> InsertCurrentData(ClientMessage message)

{

string toWrite = (await ReadAll(message)).Value;

var objs = message.Value.Insert(0, toWrite)

.Replace("\r\n", string.Empty)

.Split(';')

.Where(x => x != string.Empty)

.Select(x => x += ";").ToArray();

return objs;

}

}

}

Server start:

// Server Console

using DataEditLib.Data;

using DataEditLib.Interfaces;

using DataEditLib.Logger;

using DataEditLib.Models;

using ManagementServer.Servers;

// Setting variables

IDataEdit editService = new CsvDataHanler<MyEntity>();

//IDataEdit editService = new SqliteDataHandler<MyEntity>();

ILogger logger = new CL();

// Starting server

var \_server = new TcpServerSocket(editService, logger);

\_server.Start();

Console.ReadLine();

Client start:

Запросы вызываются через специальный файл с комадами

CreateDatabase|

Create|1,Ilya,60,5.2,2000;2,Isaac,99.23,7.9,3000;

ReadAll|

Create|3,Stepan,70,6.41,2845.19;

ReadAll|

ReadOne|3

Update|3,Aleksander,945,6.01,3957.59;

ReadAll|

Delete|1

ReadAll|

DeleteDatabase|

// Client Console

using DataEditLib.Enums;

using DataEditLib.Interfaces;

using DataEditLib.Logger;

using DataEditLib.Models.MessagesTypes;

using ManagementClient.Clients;

ILogger \_logger = new CL();

var client = new TcpClientSocket(\_logger);

using (StreamReader f = new StreamReader(@"E:\Work\РОД\LR\_1\TestData\Script\_1.txt"))

{

string line = string.Empty;

ClientMessage message = new ClientMessage();

while ((line = f.ReadLine()) != null)

{

var objs = line.Split('|');

var msgType = (MessageType)Enum.Parse(typeof(MessageType), objs[0]);

message = new ClientMessage(objs[1], msgType);

Console.WriteLine(msgType + ": " + objs[1]);

client.SendMessage(message);

Task.Delay(500).Wait();

}

}

Console.ReadLine();

**Результат:**

*Server console output:*

Client: 192.168.56.1:59267 Send message [9/28/2022 11:09:51 AM; Status: Ok; Type: CreateDatabase]:

Server: 192.168.56.1:59267 Send message [9/28/2022 11:09:53 AM; Status: Error; Type: 0]:

Client: 192.168.56.1:59267 Send message [9/28/2022 11:09:54 AM; Status: Ok; Type: Create]:

1,Ilya,60,5.2,2000;2,Isaac,99.23,7.9,3000;

Server: 192.168.56.1:59267 Send message [9/28/2022 11:09:54 AM; Status: Ok; Type: 0]:

Client: 192.168.56.1:59267 Send message [9/28/2022 11:09:54 AM; Status: Ok; Type: ReadAll]:

Server: 192.168.56.1:59267 Send message [9/28/2022 11:09:54 AM; Status: Ok; Type: 0]:

Client: 192.168.56.1:59267 Send message [9/28/2022 11:09:55 AM; Status: Ok; Type: Create]:

3,Stepan,70,6.41,2845.19;

Server: 192.168.56.1:59267 Send message [9/28/2022 11:09:55 AM; Status: Ok; Type: 0]:

Client: 192.168.56.1:59267 Send message [9/28/2022 11:09:55 AM; Status: Ok; Type: ReadAll]:

Server: 192.168.56.1:59267 Send message [9/28/2022 11:09:55 AM; Status: Ok; Type: 0]:

Client: 192.168.56.1:59267 Send message [9/28/2022 11:09:56 AM; Status: Ok; Type: ReadOne]:

3

Server: 192.168.56.1:59267 Send message [9/28/2022 11:09:56 AM; Status: Ok; Type: 0]:

Client: 192.168.56.1:59267 Send message [9/28/2022 11:09:56 AM; Status: Ok; Type: Update]:

3,Aleksander,945,6.01,3957.59;

Server: 192.168.56.1:59267 Send message [9/28/2022 11:09:57 AM; Status: Ok; Type: 0]:

Client: 192.168.56.1:59267 Send message [9/28/2022 11:09:57 AM; Status: Ok; Type: ReadAll]:

Server: 192.168.56.1:59267 Send message [9/28/2022 11:09:57 AM; Status: Ok; Type: 0]:

Client: 192.168.56.1:59267 Send message [9/28/2022 11:09:58 AM; Status: Ok; Type: Delete]:

1

Server: 192.168.56.1:59267 Send message [9/28/2022 11:09:58 AM; Status: Ok; Type: 0]:

Client: 192.168.56.1:59267 Send message [9/28/2022 11:09:58 AM; Status: Ok; Type: ReadAll]:

Server: 192.168.56.1:59267 Send message [9/28/2022 11:09:58 AM; Status: Ok; Type: 0]:

Client: 192.168.56.1:59267 Send message [9/28/2022 11:09:59 AM; Status: Ok; Type: DeleteDatabase]:

Server: 192.168.56.1:59267 Send message [9/28/2022 11:09:59 AM; Status: Error; Type: 0]:

*Client console output:*

CreateDatabase:

Client: Send message [Status: Ok; Type: CreateDatabase]:

Server: 192.168.56.1:59267 Send message [9/28/2022 11:09:53 AM; Status: Error; Type: 0]:

Wrong message type. Can't find [Create/ReadOne/ReadAll/Update/Delete] or [CreateDatabase/DeleteDatabase]

Create: 1,Ilya,60,5.2,2000;2,Isaac,99.23,7.9,3000;

Client: Send message [Status: Ok; Type: Create]:

1,Ilya,60,5.2,2000;2,Isaac,99.23,7.9,3000;

Server: 192.168.56.1:59267 Send message [9/28/2022 11:09:54 AM; Status: Ok; Type: 0]:

Object was created

ReadAll:

Client: Send message [Status: Ok; Type: ReadAll]:

Server: 192.168.56.1:59267 Send message [9/28/2022 11:09:54 AM; Status: Ok; Type: 0]:

1,Ilya,60,5.2,2000;

2,Isaac,99.23,7.9,3000;

Create: 3,Stepan,70,6.41,2845.19;

Client: Send message [Status: Ok; Type: Create]:

3,Stepan,70,6.41,2845.19;

Server: 192.168.56.1:59267 Send message [9/28/2022 11:09:55 AM; Status: Ok; Type: 0]:

Object was created

ReadAll:

Client: Send message [Status: Ok; Type: ReadAll]:

Server: 192.168.56.1:59267 Send message [9/28/2022 11:09:55 AM; Status: Ok; Type: 0]:

1,Ilya,60,5.2,2000;

2,Isaac,99.23,7.9,3000;

3,Stepan,70,6.41,2845.19;

ReadOne: 3

Client: Send message [Status: Ok; Type: ReadOne]:

3

Server: 192.168.56.1:59267 Send message [9/28/2022 11:09:56 AM; Status: Ok; Type: 0]:

3,Stepan,70,6.41,2845.19;

Update: 3,Aleksander,945,6.01,3957.59;

Client: Send message [Status: Ok; Type: Update]:

3,Aleksander,945,6.01,3957.59;

Server: 192.168.56.1:59267 Send message [9/28/2022 11:09:57 AM; Status: Ok; Type: 0]:

Object was updated

ReadAll:

Client: Send message [Status: Ok; Type: ReadAll]:

Server: 192.168.56.1:59267 Send message [9/28/2022 11:09:57 AM; Status: Ok; Type: 0]:

1,Ilya,60,5.2,2000;

2,Isaac,99.23,7.9,3000;

3,Aleksander,945,6.01,3957.59;

Delete: 1

Client: Send message [Status: Ok; Type: Delete]:

1

Server: 192.168.56.1:59267 Send message [9/28/2022 11:09:58 AM; Status: Ok; Type: 0]:

Object was deleted

ReadAll:

Client: Send message [Status: Ok; Type: ReadAll]:

Server: 192.168.56.1:59267 Send message [9/28/2022 11:09:58 AM; Status: Ok; Type: 0]:

2,Isaac,99.23,7.9,3000;

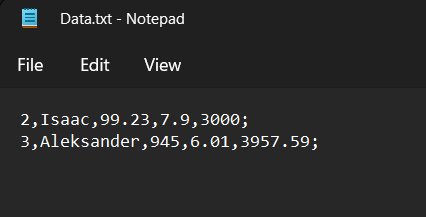
3,Aleksander,945,6.01,3957.59;

DeleteDatabase:

Client: Send message [Status: Ok; Type: DeleteDatabase]:

Server: 192.168.56.1:59267 Send message [9/28/2022 11:09:59 AM; Status: Error; Type: 0]:

Wrong message type. Can't find [Create/ReadOne/ReadAll/Update/Delete] or [CreateDatabase/DeleteDatabase]



**Вывод:** В ходе выполнения лабораторной работы я познакомился с возможностями взаимодействия приложений на основе протоколов TCP. Изучил способы реализации клиента и сервера. Изучил и реализовать способы разделенного доступа к файловым источникам данных.

**МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ**

**УЧРЕЖДЕНИЕ ОБРАЗОВАНИЯ**

**ГОМЕЛЬСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ ИМЕНИ П. О. СУХОГО**

Факультет автоматизированных и информационных систем

Кафедра «Информатика»

ОТЧЕТ ПО ЛАБОРАТОРНОЙ РАБОТЕ № 2

по дисциплине «Распределённая обработка данных»

на тему: «Архитектура “Сервер приложений”»

Выполнил: студент гр. ИП-42

Бурин И. А.

Принял: преподаватель

Шибеко В.Н.

Гомель 2022

**Цель:** Изучить архитектуру «AS». Познакомиться с возможностями взаимодействия приложений на основе протокола REST.

**Задание 2.1:**

* Разработать Application-Сервер, взяв в качестве СУБД – SqLite.
* Разработать библиотеку, реализующую (для заданной предметной области) операции редактирования и выборки данных. Данные располагаются в СУБД SqLite.
* Разработать серверное приложение для реализации операций предметной области
* Разработать клиент-приложение для реализации общения с сервером приложений.
* Форматы передачи данных – JSON. Общение клиента и сервера приложений на основе протокола REST и TCP/IP.
* Ввести временные задержки в выполнение операций на сервере для проверки одновременного доступа к данным.
* Реализовать многопользовательский доступ к AS-серверу.

Проверить одновременный доступ в одной строке в одной таблице данных SqLite для операций модификации.

**Реализация:**

SqliteDataHandler.cs

using DataEditLib.Enums;

using DataEditLib.Interfaces;

using DataEditLib.Interfaces.Crud;

using DataEditLib.Models;

using DataEditLib.Models.MessagesTypes;

using Microsoft.Data.Sqlite;

using System.Diagnostics;

namespace DataEditLib.Data

{

public class SqliteDataHandler<T> : IDataEdit, IDataBaseOptions<T>,

ICreate<T>, IRead<T>, IUpdate<T>, IDelete<T> where T : MyEntity, new()

{

public async Task<IActionResult<T>> Create(ClientMessage message)

{

using (var connection = new SqliteConnection(ProjectProperties.SqliteDataString))

{

try

{

connection.Open();

var list = ToCollection(message.Value);

SqliteCommand command = new SqliteCommand();

command.Connection = connection;

foreach (var item in list)

{

command.CommandText = $"INSERT INTO MyEntities (Name, ScopeOfWork, UnitPrice, AccruedEarnings) " +

$"VALUES ('{item.Name}', '{item.ScopeOfWork}', '{item.UnitPrice}', '{item.AccruedEarnings}')";

var number = await command.ExecuteNonQueryAsync();

}

await connection.CloseAsync();

await connection.DisposeAsync();

return new ActionResult<T>(message, MessageStatus.Ok, ProjectProperties.OkCreatedInfo);

}

catch (Exception ex)

{

await connection.CloseAsync();

await connection.DisposeAsync();

return new ActionResult<T>(message, MessageStatus.Error, ex);

}

}

}

public async Task<IActionResult<T>> CreateDataBase(ClientMessage message)

{

using (var connection = new SqliteConnection(ProjectProperties.SqliteDataString))

{

try

{

connection.Open();

var list = ToCollection(message.Value);

SqliteCommand command = new SqliteCommand();

command.Connection = connection;

command.CommandText = "CREATE TABLE MyEntities(Id INTEGER NOT NULL PRIMARY KEY AUTOINCREMENT UNIQUE, Name TEXT, ScopeOfWork DOUBLE, UnitPrice DOUBLE, AccruedEarnings DOUBLE)";

int number = await command.ExecuteNonQueryAsync();

await connection.CloseAsync();

await connection.DisposeAsync();

return new ActionResult<T>(message, MessageStatus.Ok, $"Database was created");

}

catch (Exception ex)

{

await connection.CloseAsync();

await connection.DisposeAsync();

return new ActionResult<T>(message, MessageStatus.Error, ex);

}

}

}

public async Task<ServerMessage> ParseMessageType(ClientMessage message)

{

IActionResult<T> actionResult;

try

{

if (message == null)

throw new ArgumentNullException();

if (message.MsgType == MessageType.CreateDatabase)

actionResult = await CreateDataBase(message);

else if (message.MsgType == MessageType.DeleteDatabase)

actionResult = await DeleteDataBase(message);

else if (message.MsgType == MessageType.Create)

actionResult = await Create(message);

else if (message.MsgType == MessageType.ReadOne)

actionResult = await ReadOne(message);

else if (message.MsgType == MessageType.ReadAll)

actionResult = await ReadAll(message);

else if (message.MsgType == MessageType.Update)

actionResult = await Update(message);

else if (message.MsgType == MessageType.Delete)

actionResult = await Delete(message);

else

throw new ArgumentException(ProjectProperties.MessageTypeNotDefine);

return ServerMessage.ServerResponse((IActionResult<MyEntity>)actionResult, message);

}

catch (Exception ex)

{

actionResult = new ActionResult<T>(message, MessageStatus.Error, ex);

return ServerMessage.ServerResponse((IActionResult<MyEntity>)actionResult, message);

}

}

public async Task<IActionResult<T>> Delete(ClientMessage message)

{

using (var connection = new SqliteConnection(ProjectProperties.SqliteDataString))

{

try

{

connection.Open();

if (int.TryParse(message.Value, out int id))

{

SqliteCommand command = new SqliteCommand();

command.Connection = connection;

command.CommandText = $"DELETE FROM MyEntities WHERE Id='{id}'";

var number = await command.ExecuteNonQueryAsync();

}

else throw new ArgumentException(message.Value);

await connection.CloseAsync();

await connection.DisposeAsync();

return new ActionResult<T>(message, MessageStatus.Ok, ProjectProperties.OkDeletedInfo);

}

catch (Exception ex)

{

await connection.CloseAsync();

await connection.DisposeAsync();

return new ActionResult<T>(message, MessageStatus.Error, ex);

}

}

}

public async Task<IActionResult<T>> DeleteDataBase(ClientMessage message)

{

try

{

if (File.Exists(ProjectProperties.SqliteFilePath))

{

SqliteConnection.ClearAllPools();

GC.Collect();

GC.WaitForPendingFinalizers();

File.Delete(ProjectProperties.SqliteFilePath);

return new ActionResult<T>(message, MessageStatus.Ok, $"Database was deleted");

}

else throw new Exception("File error");

}

catch (Exception ex)

{

return new ActionResult<T>(message, MessageStatus.Error, ex);

}

}

public async Task<IActionResult<T>> ReadAll(ClientMessage message)

{

string result = string.Empty;

using (var connection = new SqliteConnection(ProjectProperties.SqliteDataString))

{

try

{

connection.Open();

SqliteCommand command = new SqliteCommand("SELECT \* FROM MyEntities", connection);

using (SqliteDataReader reader = await command.ExecuteReaderAsync())

{

if (reader.HasRows)

{

var list = new List<T>();

while (reader.Read())

{

var tmp = new T();

tmp.Id = reader.GetInt32(0);

tmp.Name = reader.GetString(1);

tmp.ScopeOfWork = reader.GetDouble(2);

tmp.UnitPrice = reader.GetDouble(3);

tmp.AccruedEarnings = reader.GetDouble(4);

list.Add(tmp);

}

for (int i = 0; i < list.Count; i++)

{

result += list[i].ToString() + '\n';

}

await reader.CloseAsync();

await reader.DisposeAsync();

}

else throw new Exception("Empty databse");

}

await connection.CloseAsync();

await connection.DisposeAsync();

return new ActionResult<T>(message, MessageStatus.Ok, result);

}

catch (Exception ex)

{

await connection.CloseAsync();

await connection.DisposeAsync();

return new ActionResult<T>(message, MessageStatus.Error, ex);

}

}

}

public async Task<IActionResult<T>> ReadOne(ClientMessage message)

{

using (var connection = new SqliteConnection(ProjectProperties.SqliteDataString))

{

try

{

connection.Open();

if (int.TryParse(message.Value, out int id))

{

SqliteCommand command = new SqliteCommand($"SELECT \* FROM MyEntities WHERE Id = '{id}'", connection);

using (SqliteDataReader reader = await command.ExecuteReaderAsync())

{

if (reader.HasRows)

{

var tmp = new T();

while (reader.Read())

{

tmp.Id = reader.GetInt32(0);

tmp.Name = reader.GetString(1);

tmp.ScopeOfWork = reader.GetDouble(2);

tmp.UnitPrice = reader.GetDouble(3);

tmp.AccruedEarnings = reader.GetDouble(4);

}

await reader.CloseAsync();

await reader.DisposeAsync();

await connection.CloseAsync();

await connection.DisposeAsync();

return new ActionResult<T>(message, MessageStatus.Ok, tmp.ToString());

}

else throw new Exception("Empty db response");

}

}

else throw new KeyNotFoundException();

}

catch (Exception ex)

{

await connection.CloseAsync();

await connection.DisposeAsync();

return new ActionResult<T>(message, MessageStatus.Error, ex);

}

}

}

public async Task<IActionResult<T>> Update(ClientMessage message)

{

using (var connection = new SqliteConnection(ProjectProperties.SqliteDataString))

{

try

{

connection.Open();

var obj = ToCollection(message.Value).First();

SqliteCommand command = new SqliteCommand();

command.Connection = connection;

if (obj != null)

{

command.CommandText = $"UPDATE MyEntities SET Name='{obj.Name}', ScopeOfWork='{obj.ScopeOfWork}', UnitPrice='{obj.UnitPrice}', AccruedEarnings='{obj.AccruedEarnings}' WHERE Id='{obj.Id}'";

var number = await command.ExecuteNonQueryAsync();

}

else throw new NullReferenceException();

await connection.CloseAsync();

await connection.DisposeAsync();

return new ActionResult<T>(message, MessageStatus.Ok, ProjectProperties.OkUpdatedInfo);

}

catch (Exception ex)

{

await connection.CloseAsync();

await connection.DisposeAsync();

return new ActionResult<T>(message, MessageStatus.Error, ex);

}

}

}

private IEnumerable<T> ToCollection(string line)

{

var list = new List<T>();

var lines = line.Replace("\r\n", string.Empty)

.Replace("\r\n", string.Empty)

.Split(';')

.Where(x => x != string.Empty)

.Select(x => x += ";").ToArray();

for (int i = 0; i < lines.Count(); i++)

list.Add((T)new T().ToObjectFromText(lines.ElementAt(i)));

return list;

}

}

}

Server start:

Меняем зависимость в конструкторе TcpServer на SqliteGataHandler

// Server Console

using DataEditLib.Data;

using DataEditLib.Interfaces;

using DataEditLib.Logger;

using DataEditLib.Models;

using ManagementServer.Servers;

// Setting variables

//IDataEdit editService = new CsvDataHanler<MyEntity>();

IDataEdit editService = new SqliteDataHandler<MyEntity>();

ILogger logger = new CL();

// Starting server

var \_server = new TcpServerSocket(editService, logger);

\_server.Start();

Message.cs

using DataEditLib.Enums;

using System.Net;

using System.Net.Sockets;

using System.Text.Json.Serialization;

namespace DataEditLib.Models

{

public abstract class Message

{

public Message()

{

Date = DateTime.Now;

}

[JsonConstructor]

public Message(string Value, string IpAddress, int Port, DateTime Date,

SenderType SenderType, MessageStatus MsgStatus, MessageType MsgType)

{

this.Value = Value;

this.IpAddress = IpAddress;

this.Port = Port;

this.Date = Date;

this.SenderType = SenderType;

this.MsgStatus = MsgStatus;

this.MsgType = MsgType;

}

public Message(string value, MessageType msgType)

{

Value = value;

MsgType = msgType;

Date = DateTime.Now;

}

public Message(MessageType msgType)

{

MsgType = msgType;

Date = DateTime.Now;

}

public string Value { get; set; }

public string IpAddress { get; set; }

public int Port { get; set; }

public DateTime Date { get; set; }

public SenderType SenderType { get; set; }

public MessageStatus MsgStatus { get; set; }

public MessageType MsgType { get; set; }

public abstract string Info(SenderType loggerDefine);

public static string CollectionToOneString<T>(IEnumerable<T> collection) where T : class

{

var dataToTransfer = string.Empty;

foreach (var entity in collection)

dataToTransfer += entity.ToString() + "\r\n";

return dataToTransfer;

}

}

}

ActionResult.cs

using DataEditLib.Enums;

using DataEditLib.Interfaces;

using DataEditLib.Models.MessagesTypes;

namespace DataEditLib.Models

{

public class ActionResult<T> : IActionResult<T>

{

public ActionResult() {}

public ActionResult(ClientMessage client, MessageStatus messageStatus, Exception exception)

{

ClientMsg = client;

MessageStatus = messageStatus;

Exception = exception;

Value = exception.Message;

}

public ActionResult(ClientMessage client, MessageStatus messageStatus, string value)

{

ClientMsg = client;

MessageStatus = messageStatus;

Value = value;

}

public ClientMessage ClientMsg { get; }

public MessageStatus MessageStatus { get; }

public Exception Exception { get; }

public string Value { get; }

}

}

**Результат:**

Server console output:

Client: 192.168.56.1:59267 Send message [9/28/2022 11:45:41 AM; Status: Ok; Type: CreateDatabase]:

Server: 192.168.56.1:59267 Send message [9/28/2022 11:45:42 AM; Status: Ok; Type: 0]:

Client: 192.168.56.1:59267 Send message [9/28/2022 11:45:43 AM; Status: Ok; Type: Create]:

1,Ilya,60,5.2,2000;2,Isaac,99.23,7.9,3000;

Server: 192.168.56.1:59267 Send message [9/28/2022 11:45:43 AM; Status: Ok; Type: 0]:

Client: 192.168.56.1:59267 Send message [9/28/2022 11:45:43 AM; Status: Ok; Type: ReadAll]:

Server: 192.168.56.1:59267 Send message [9/28/2022 11:45:44 AM; Status: Ok; Type: 0]:

Client: 192.168.56.1:59267 Send message [9/28/2022 11:45:44 AM; Status: Ok; Type: Create]:

3,Stepan,70,6.41,2845.19;

Server: 192.168.56.1:59267 Send message [9/28/2022 11:45:44 AM; Status: Ok; Type: 0]:

Client: 192.168.56.1:59267 Send message [9/28/2022 11:45:45 AM; Status: Ok; Type: ReadAll]:

Server: 192.168.56.1:59267 Send message [9/28/2022 11:45:45 AM; Status: Ok; Type: 0]:

Client: 192.168.56.1:59267 Send message [9/28/2022 11:45:45 AM; Status: Ok; Type: ReadOne]:

3

Server: 192.168.56.1:59267 Send message [9/28/2022 11:45:45 AM; Status: Ok; Type: 0]:

Client: 192.168.56.1:59267 Send message [9/28/2022 11:45:46 AM; Status: Ok; Type: Update]:

3,Aleksander,945,6.01,3957.59;

Server: 192.168.56.1:59267 Send message [9/28/2022 11:45:46 AM; Status: Ok; Type: 0]:

Client: 192.168.56.1:59267 Send message [9/28/2022 11:45:46 AM; Status: Ok; Type: ReadAll]:

Server: 192.168.56.1:59267 Send message [9/28/2022 11:45:46 AM; Status: Ok; Type: 0]:

Client: 192.168.56.1:59267 Send message [9/28/2022 11:45:47 AM; Status: Ok; Type: Delete]:

1

Server: 192.168.56.1:59267 Send message [9/28/2022 11:45:47 AM; Status: Ok; Type: 0]:

Client: 192.168.56.1:59267 Send message [9/28/2022 11:45:47 AM; Status: Ok; Type: ReadAll]:

Server: 192.168.56.1:59267 Send message [9/28/2022 11:45:47 AM; Status: Ok; Type: 0]:

Client: 192.168.56.1:59267 Send message [9/28/2022 11:45:48 AM; Status: Ok; Type: DeleteDatabase]:

Server: 192.168.56.1:59267 Send message [9/28/2022 11:45:48 AM; Status: Ok; Type: 0]:

Console console output:

CreateDatabase:

Client: Send message [Status: Ok; Type: CreateDatabase]:

Server: 192.168.56.1:59267 Send message [9/28/2022 11:45:42 AM; Status: Ok; Type: 0]:

Database was created

Create: 1,Ilya,60,5.2,2000;2,Isaac,99.23,7.9,3000;

Client: Send message [Status: Ok; Type: Create]:

1,Ilya,60,5.2,2000;2,Isaac,99.23,7.9,3000;

Server: 192.168.56.1:59267 Send message [9/28/2022 11:45:43 AM; Status: Ok; Type: 0]:

Object was created

ReadAll:

Client: Send message [Status: Ok; Type: ReadAll]:

Server: 192.168.56.1:59267 Send message [9/28/2022 11:45:44 AM; Status: Ok; Type: 0]:

1,Ilya,60,5.2,2000;

2,Isaac,99.23,7.9,3000;

Create: 3,Stepan,70,6.41,2845.19;

Client: Send message [Status: Ok; Type: Create]:

3,Stepan,70,6.41,2845.19;

Server: 192.168.56.1:59267 Send message [9/28/2022 11:45:44 AM; Status: Ok; Type: 0]:

Object was created

ReadAll:

Client: Send message [Status: Ok; Type: ReadAll]:

Server: 192.168.56.1:59267 Send message [9/28/2022 11:45:45 AM; Status: Ok; Type: 0]:

1,Ilya,60,5.2,2000;

2,Isaac,99.23,7.9,3000;

3,Stepan,70,6.41,2845.19;

ReadOne: 3

Client: Send message [Status: Ok; Type: ReadOne]:

3

Server: 192.168.56.1:59267 Send message [9/28/2022 11:45:45 AM; Status: Ok; Type: 0]:

3,Stepan,70,6.41,2845.19;

Update: 3,Aleksander,945,6.01,3957.59;

Client: Send message [Status: Ok; Type: Update]:

3,Aleksander,945,6.01,3957.59;

Server: 192.168.56.1:59267 Send message [9/28/2022 11:45:46 AM; Status: Ok; Type: 0]:

Object was updated

ReadAll:

Client: Send message [Status: Ok; Type: ReadAll]:

Server: 192.168.56.1:59267 Send message [9/28/2022 11:45:46 AM; Status: Ok; Type: 0]:

1,Ilya,60,5.2,2000;

2,Isaac,99.23,7.9,3000;

3,Aleksander,945,6.01,3957.59;

Delete: 1

Client: Send message [Status: Ok; Type: Delete]:

1

Server: 192.168.56.1:59267 Send message [9/28/2022 11:45:47 AM; Status: Ok; Type: 0]:

Object was deleted

ReadAll:

Client: Send message [Status: Ok; Type: ReadAll]:

Server: 192.168.56.1:59267 Send message [9/28/2022 11:45:47 AM; Status: Ok; Type: 0]:

2,Isaac,99.23,7.9,3000;

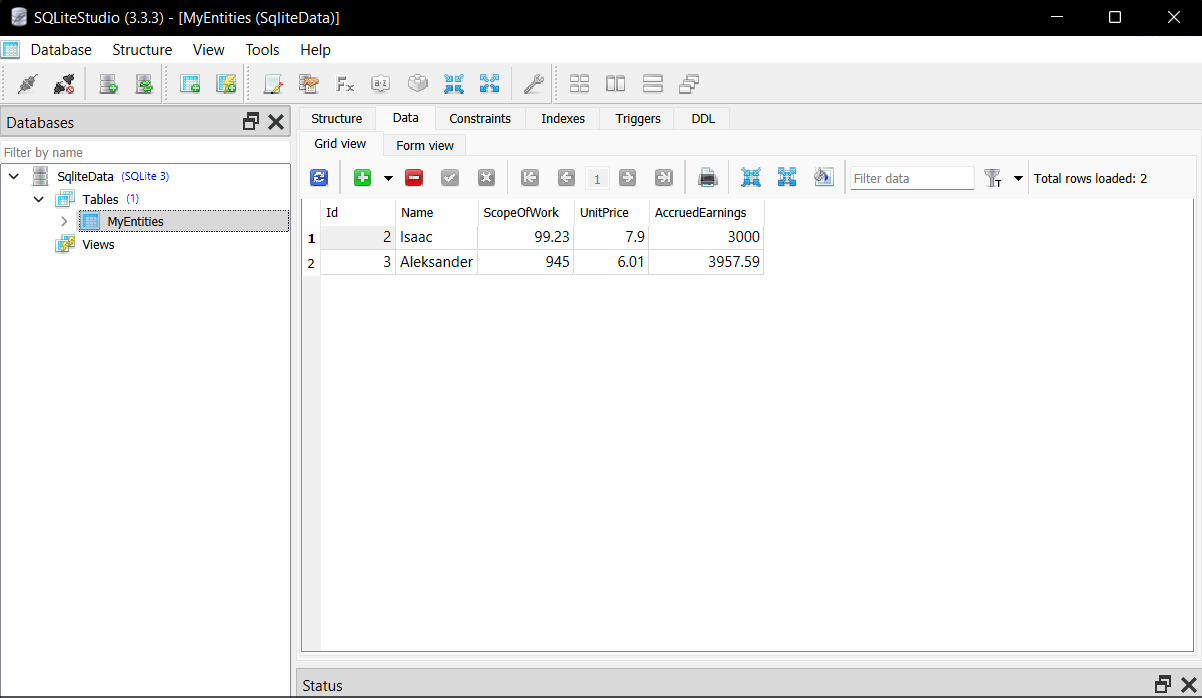
3,Aleksander,945,6.01,3957.59;

DeleteDatabase:

Client: Send message [Status: Ok; Type: DeleteDatabase]:

Server: 192.168.56.1:59267 Send message [9/28/2022 11:45:48 AM; Status: Ok; Type: 0]:

Database was deleted



**Вывод:** В ходе выполнения лабораторной работы я изучил архитектуру «AS». Познакомился с возможностями взаимодействия приложений на основе протокола REST.