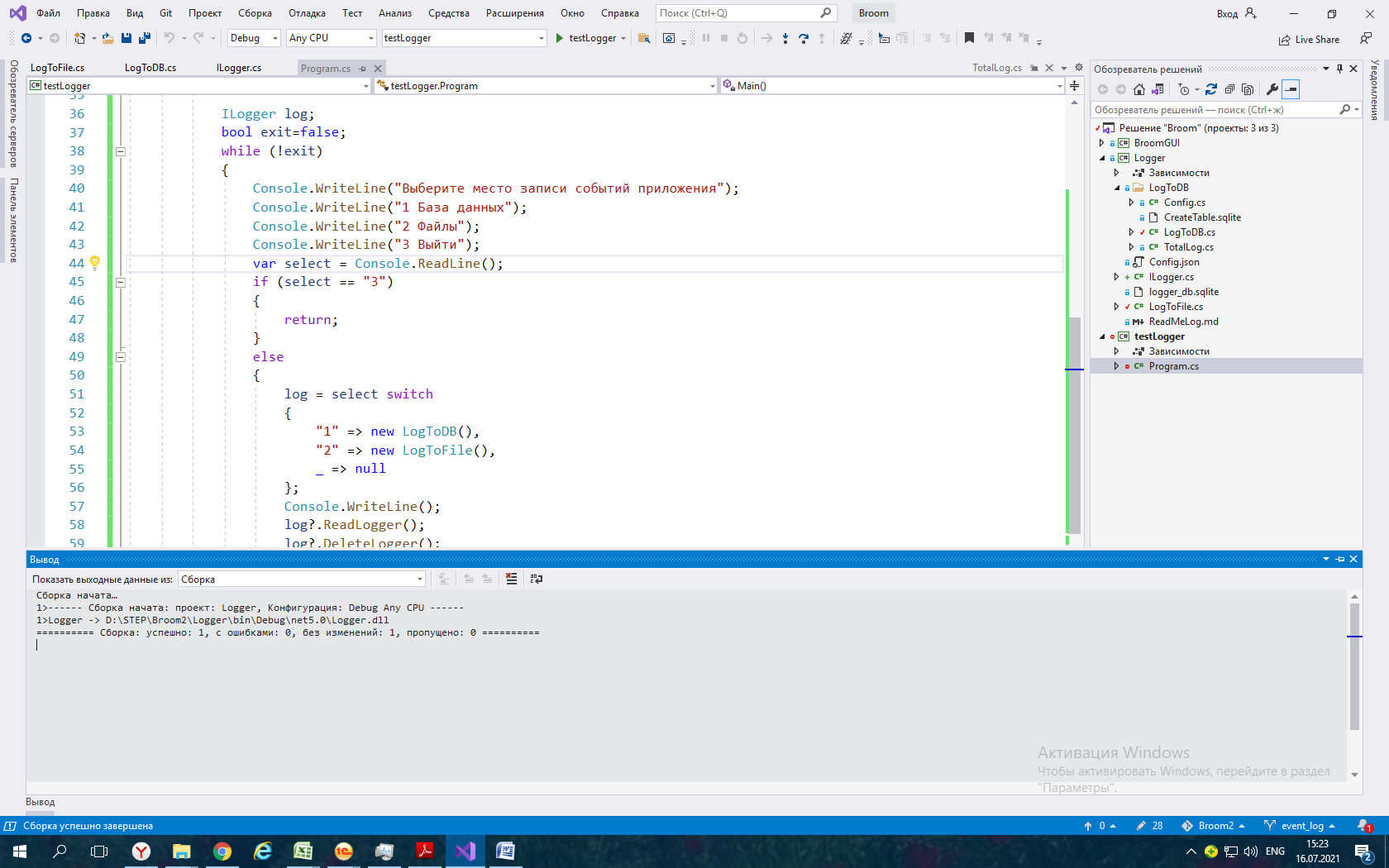
**16.07.2021**



**Logger**

**LogTo DB**

**Config.cs**

**CreateTable.sqlite**

**LogTo DB.cs**

**TotalLog.cs**

**Config.json**

**ILogger.cs**

**logger\_db.sqlite**

**LogToFile.cs**

**Config.cs**

namespace Logger

{

public class Config

{

public string DataSource { get; set; }

public string Mode { get; set; }

}

}

**CreateTable.sqlite**

create table tab\_total\_log

(

id integer not null primary key autoincrement ,

type\_event text not null ,

date\_time\_event text not null,

user text not null,

message text not null

);

**LogTo DB.cs**

using System;

using System.IO;

using System.Text.Json;

using Microsoft.Data.Sqlite;

using System.Collections.Generic;

namespace Logger

{

public class LogToDB: ILogger

{

private static FileStream file = new("config.json", FileMode.Open);

private static Config config = JsonSerializer.DeserializeAsync<Config>(file).Result;

private string connectionString = $"Data Source={config.DataSource};Mode={config.Mode};";

public SqliteConnection \_connection;

public SqliteCommand \_query;

public LogToDB()

{

\_connection = new SqliteConnection(connectionString);

\_query = new SqliteCommand { Connection = \_connection };

}

public void Open()

{

try

{

\_connection.Open();

}

catch (InvalidOperationException)

{

throw new Exception("Ошибка открытия БД");

}

catch (SqliteException)

{

throw new Exception("Подключаемся к уже открытой БД");

}

}

public void Close()

{

\_connection.Close();

}

public SqliteDataReader SelectQuery(string sql)

{

\_query.CommandText = sql;

var result = \_query.ExecuteReader();

return result;

}

public void RecordToLog(string typeevent, string message)

{

\_connection.Open();

\_query.CommandText = $"INSERT INTO tab\_total\_log (type\_event, date\_time\_event, user, message)" +

$"VALUES ('{typeevent}', '{DateTime.Now}', '{Environment.UserName}', '{message}')";

\_query.ExecuteNonQuery();

\_connection.Close();

}

public void ReadTheLog()

{

\_connection.Open();

var sql = "SELECT \* FROM tab\_total\_log";

using var result = SelectQuery(sql);

if (!result.HasRows)

{

Console.WriteLine("Нет данных");

return;

}

var totals = new List<TotalLog>();

while (result.Read())

{

var total = new TotalLog

{

Id = result.GetInt32(0),

TypeEvent = result.GetString(1),

DateTimeEvent = result.GetString(2),

User = result.GetString(3),

Message = result.GetString(4)

};

totals.Add(total);

}

foreach (var total in totals)

{

Console.WriteLine($"{total.Id} | {total.TypeEvent} | {total.DateTimeEvent} | {total.User} | {total.Message}");

}

\_connection.Close();

}

public void ClearLog()

{

\_connection.Open();

\_query.CommandText = "DELETE FROM tab\_total\_log";

\_query.ExecuteNonQuery();

\_connection.Close();

}

/\*

public void RecordEventToDB(string typeevent, string message)

{

\_connection.Open();

\_query.CommandText = $"INSERT INTO tab\_total\_log (type\_event, date\_time\_event, user, message)" +

$"VALUES ('{typeevent}', '{DateTime.Now}', '{Environment.UserName}', '{message}')";

\_query.ExecuteNonQuery();

\_connection.Close();

}

//Выводит данные таблицы в консоль на этапе отладки

public void GetTotalLog()

{

\_connection.Open();

var sql = "SELECT \* FROM tab\_total\_log";

using var result = SelectQuery(sql);

if (!result.HasRows)

{

Console.WriteLine("Нет данных");

return;

}

var totals = new List<TotalLog>();

while (result.Read())

{

var total = new TotalLog

{

Id = result.GetInt32(0),

TypeEvent = result.GetString(1),

DateTimeEvent = result.GetString(2),

User = result.GetString(3),

Message = result.GetString(4)

};

totals.Add(total);

}

foreach (var total in totals)

{

Console.WriteLine($"{total.Id} | {total.TypeEvent} | {total.DateTimeEvent} | {total.User} | {total.Message}");

}

\_connection.Close();

}

public void DeleteToDB()

{

\_connection.Open();

\_query.CommandText = "DELETE FROM tab\_total\_log";

\_query.ExecuteNonQuery();

\_connection.Close();

}\*/

}

}

**TotalLog.cs**

using System;

namespace Logger

{

public class TotalLog

{

public int Id { get; set; }

public string TypeEvent { get; set; }

public string DateTimeEvent { get; set; }

public string User { get; set; }

public string Message { get; set; }

public TotalLog(int id, string typeevent, string datetimeevent, string user, string message)

{

Id = id;

TypeEvent = typeevent;

DateTimeEvent = datetimeevent;

User = user;

Message = message;

}

public TotalLog() { }

}

}

**Config.json**

{

"DataSource": "logger\_db.sqlite",

"Mode": "ReadWrite"

}

**ILogger.cs**

namespace Logger

{

public interface ILogger

{

public void RecordToLog(string typeevent, string message) { }

public void ReadTheLog() { }

public void ClearLog() { }

}

}

**LogToFile.cs**

using System;

using System.IO;

using System.Reflection;

using System.Collections.Generic;

namespace Logger

{

public class LogToFile: ILogger

{

private readonly string TotalPath;

private readonly string SuccessPath;

private readonly string ErrorsPath;

private readonly string loggerDir=Path.Combine(Path.GetDirectoryName(Assembly.GetExecutingAssembly().Location),"BroomLogger");

//в разработке вариант с config файлом

public LogToFile()

{

//создается 1 раз при 1 запуске программы

Directory.CreateDirectory(loggerDir);

//создается 1 раз при 1 запуске программы

TotalPath = Path.Combine(loggerDir, "TotalLog.log");

//создаются по 1 на каждую текущую дату

var creatTime = DateTime.Now.ToShortDateString();

var sLog = "SuccessLog" + "\_" + creatTime + ".log";

SuccessPath = Path.Combine(loggerDir, sLog);

var eLog = "ErrorsLog" + "\_" + creatTime + ".log";

ErrorsPath = Path.Combine(loggerDir, eLog);

}

public async void RecordToLog(string typeevent, string message)

{

var text = typeevent + " | " + DateTime.Now + " | " + Environment.UserName + " | " + message + " \n";

Console.WriteLine(text);

switch (typeevent)

{

case "INFO":

case "WARN":

await File.AppendAllTextAsync(TotalPath, text);

break;

case "SUCCESS":

await File.AppendAllTextAsync(SuccessPath, text);

break;

case "ERROR":

await File.AppendAllTextAsync(ErrorsPath, text);

break;

}

}

public async void ReadTheLog()

{

if (File.Exists(TotalPath))

{

StreamReader readerTotal = new(TotalPath);

Console.WriteLine(await readerTotal.ReadToEndAsync());

readerTotal.Close();

}

if (File.Exists(SuccessPath))

{

StreamReader readerSuccess = new(SuccessPath);

Console.WriteLine(await readerSuccess.ReadToEndAsync());

readerSuccess.Close();

}

if (File.Exists(ErrorsPath))

{

StreamReader readerErrors = new(ErrorsPath);

Console.WriteLine(await readerErrors.ReadToEndAsync());

readerErrors.Close();

}

}

public void ClearLogr()

{

if (Directory.Exists(loggerDir))

{

File.WriteAllText(TotalPath, null);

var directoryInfo = new DirectoryInfo(loggerDir);

foreach (var file in directoryInfo.GetFiles())

{

if (file.Name != "TotalLog.log") file.Delete();

}

}

}

/\*

public async void RecordEventToFile(string type, string message)

{

var text = type + " | " + DateTime.Now + " | " + Environment.UserName + " | " + message + " \n";

switch (type)

{

case "INFO":

case "WARN":

await File.AppendAllTextAsync(TotalPath, text);

break;

case "SUCCESS":

await File.AppendAllTextAsync(SuccessPath, text);

break;

case "ERROR":

await File.AppendAllTextAsync(ErrorsPath, text);

break;

}

}

//Выводит данные таблицы в консоль на этапе отладки

public async void ReadFromFile()

{

StreamReader readerTotal = new StreamReader(TotalPath);

Console.WriteLine(await readerTotal.ReadToEndAsync());

readerTotal.Close();

StreamReader readerSuccess = new StreamReader(SuccessPath);

Console.WriteLine(await readerSuccess.ReadToEndAsync());

readerSuccess.Close();

StreamReader readerErrors = new StreamReader(ErrorsPath);

Console.WriteLine(await readerErrors.ReadToEndAsync());

readerErrors.Close();

}\*/

}

}

**Programm.cs**

using Logger;

using System.Collections.Generic;

using System.IO;

using System.Linq;

using System.Text.Json;

namespace testLogger

{

class Program

{

static void Main()

{

/\*

LogToFile log1=new LogToFile();

log1.RecordEventToFile("INFO", "launching the program");

log1.RecordEventToFile("WARN", "no files found in the / C / temp folder");

log1.RecordEventToFile("SUCCESS", "/C/temp/broom\_160\_option delete");

log1.RecordEventToFile("ERROR", "access denied to / C / temp folder");

//Thread.Sleep(1000);

log1.ReadFromFile();

\*/

LogToDB log = new();

log.GetTotalLog();

log.RecordEventToDB("INFO", "launching the program");

/\*

log.RecordEventToDB("WARN", "no files found in the / C / temp folder");

log.RecordEventToDB("SUCCESS", "/C/temp/broom\_160\_option delete");

log.RecordEventToDB("ERROR", "access denied to / C / temp folder");

log.GetTotalLog();\*/

}

}

}