

**Катедра „Компютърни системи”**

**Специалност „Компютърно и софтуерно инженерство”, степен бакалавър**

Курсова работа по Програмни среди

**Тема: Реализация на Logger в отделен проект, който да се компилира до библиотека и да се преизползва в другите проекти.**

**Изготвил: Петьо Николаев Цветков**

**Курс III, група 37**

**Факултетен номер: 121217078**

**Дата: 04.06.2020**

**Среда за разработка**

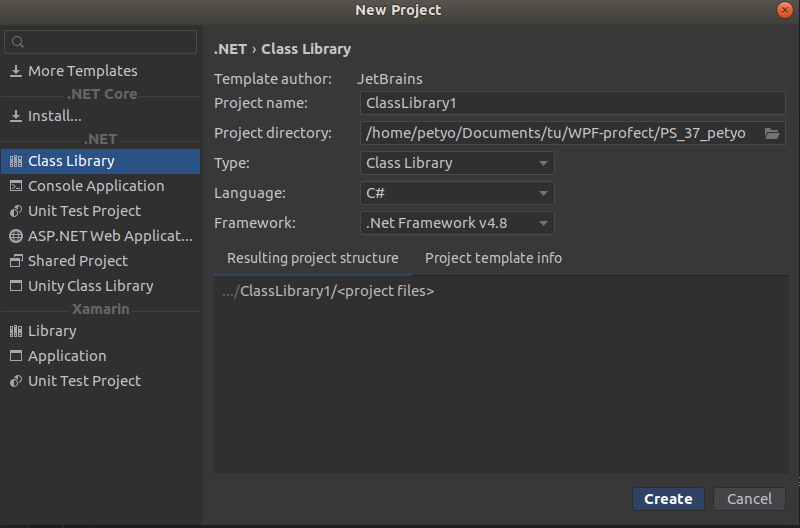
За разработването на проекта съм използвал jetbrains rider за развойна среда. Тя ни предоставя лесен интерфейс за работа с C#. Проекта е разработен на C# .net framework 4.8 .

**Анализ на изготвените промени**

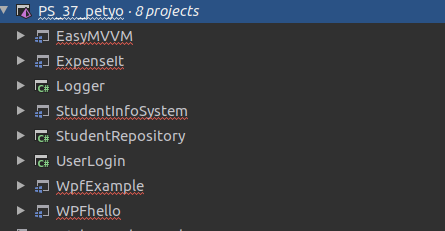
За разработката на курсовата работа съм изготвил нов модул в отделен проект, който да се компилира до библиотека и да може да се преизползва в проектите. Новия модул съдържа Основен клас, който да бъде използван от другите проекти за логване на активности. Той има възможноста да бъде преконфигуриран, да бъдаt зададени нови връзки към базаta данни или ново име за файл, в който да запазваме активностите.

**Изпълнение на функционалностите**

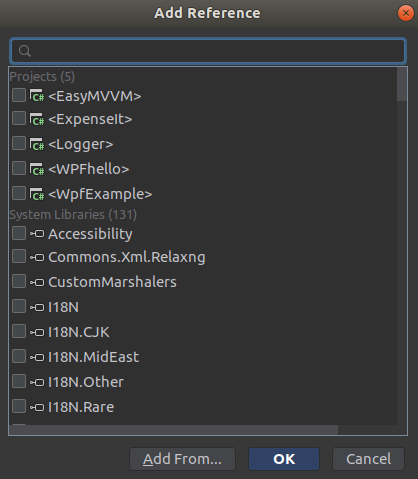
За създаването на новия модъл трябва да създадем нов проект от тъп библиотека. За да създадем нов проект става чрез „десен бътон върху главния проект → add → new project“. След, което ще ви покаже прозорец, в който да конфигурирате новият проект(избор на име, версия над framework-а и т.н.)



След като успешно сме създали проект можем да започнем да добавяме новата функционалност в него.



За да можем да използваме новосъздадения проект в другите проекти, ние трябва да добавим като reference. Това можем да го направим като натиснем десен бутон върху проекта, в който искаме да го добавим, и след това add → reference.



От прозореца можем да изберем, кои библиотеки искаме да добавим в проекта.

В проект Logger, който създадохме, ще добавим 4 нови класа, които ще се грижат за работата на логване на активности. Новите класове ще бъдат DbLogger, FileLogger, Log, Logger.

DbLogger — Грижи се за добавянето на лога към базата данни.

FileLogger — Грижи се за добавянето на логове във файл.

Log — Този клас служи за запазването на логовете в базата с entity framework-а

Logger — Класъм, който ще се грижи да менажира добавянето на логове в базата данни и във файла.

public class DbLogger : DbContext

{

private static string connection =

"Data Source=(local);Initial Catalog=StudentInfoDatabase;Integrated Security=True";

private static DbLogger \_instance = null;

public static void plant(string connectionString)

{

if (!String.IsNullOrEmpty(connectionString))

{

connection = connectionString;

}

\_instance = new DbLogger(connection);

}

public static DbLogger Instance

{

get

{

if (\_instance == null)

\_instance = new DbLogger(connection);

return \_instance;

}

}

public DbSet<Log> Logs { get; set; }

protected DbLogger(string connection) : base(connection)

{

}

public void log(string user, string role,string activity)

{

Logs

.Add(new Log(DateTime.Now, user,

role, activity));

SaveChanges();

}

}

DbLogger класа наследява Dbcontext за да може да работи с базата данни. В метода “plant” можем да преконфигурираме низ-а за връзка с базата ако не се намира на тя на същата машина. Ако не добавим низ за връзка ще преизползваме този по подразбиране, който вече имаме добавен в класа. Класа следва Singleton модел. Имаме поле Instance, което то преставлява вече създадената на инстанция на обекта или създава такава ако не е била създавана до сега.

Функцията “log” като параметри получава потребителя извършил дейноста, ролята на потребителя и каква активност той е извършил. Тя създава нова инстанция от класа “Log” с данните получе като входни и запазва новосъздадения обект в базата данни.

public class FileLogger

{

private static FileLogger \_instance = null;

private static String \_fileName = "log.txt";

public static string FileName

{

get => \_fileName;

}

public static void plant(String fileName)

{

if (!string.IsNullOrEmpty(fileName))

{

\_fileName = fileName;

}

\_instance = new FileLogger();

}

public static FileLogger Instance

{

get

{

if (\_instance == null)

\_instance = new FileLogger();

return \_instance;

}

}

public void log(string user,string role, String activity)

{

string activityLine = DateTime.Now + ";"

+ user + ";"

+ role + ";"

+ activity;

if (File.Exists(FileName))

File.AppendAllText(FileName, activityLine + "\n");

}

}

Класа FileLogger служи за добавяне на логовете в файл, който може да бъде избран или да използваме този по подразбиране “log.txt”. Класа следва Singleton модел-а като в “Instance” поелоте връщаме като резултат инстанция към класа. Финкцията “plant” зададем новия клас, който искаме да ползваме за запис на логовете.

public class Log

{

public System.Int32 LogId { get; set; }

public System.DateTime Date { get; set; }

public System.String Role { get; set; }

public System.String User { get; set; }

public System.String Activity { get; set; }

public Log()

{

}

public Log(DateTime date, string role, string user, string activity)

{

Date = date;

Role = role;

User = user;

Activity = activity;

}

}

Класа “Log” представлява клас, в който описваме полетата, които искаме. в таблицата в базата данни. Празния конструктор на класа е задължителен за да може Entity framework-a да създадед истанция за всеки запис в базата и след това да добави полетата.

public class Logger

{

private static Logger \_instance = null;

private static List<string> currentSessionActivities = new List<string>();

public static void plant()

{

\_instance = new Logger();

}

public static void plant(string fileName, string connection)

{

\_instance = new Logger(fileName, connection);

}

private Logger() : this(null, null)

{

}

private Logger(string fileName, string connection)

{

DbLogger.plant(connection);

FileLogger.plant(fileName);

}

public static Logger Instance

{

get

{

if (\_instance == null)

\_instance = new Logger();

return \_instance;

}

}

public void LogActivity(string user, string role, string activity)

{

FileLogger.Instance.log(user, role, activity );

DbLogger.Instance.log(user, role, activity );

}

public string GetCurrentSessionActivities()

{

StringBuilder sb = new StringBuilder();

foreach (var activities in currentSessionActivities)

sb.Append(activities);

return sb.ToString();

}

}

Logger е основния клас за работа от новия проект. Чрез него можеш да конфигурираш низа за връзка с базата данни и файл-а за запис. Той следва Singleton модел-а, чрез което прави работата ни с него по лесна. Имаме две “plant” функции, които да използваме за конфигурация на класа. В единия той не приема никакви входни параметри, с който искаме да използваме стойностите по подразбиране. С втората функция, което като входни параметри получава името на файл-а, в който да запазваме активностите, и низ за връзка към базата данни, която искаме да използваме.

LogActivity функцията получава като входни параметри, потребителя извършил активноста, ролята на потребителя и каква активност е извършил. Тя извиква отделните функции на двата класа за запазвазване на логовете с съответните параметри необходими за тяхната работа.

GetCurrentSessionActivities връща символен низ от всички извършени дейности от текущата сесия на потребителя.

След като сме добавили новия проект като референция към проекта, в който искаме да го ползваме, препоръчително е в началото на програмата да инициализираме този клас по следния начин :

Logger.Logger.plant();

за да могат да бъдат добавени в началото таблиците в базата данни ако не съсществуват и да конфигурира връзката с базата, защото тази операция отнема повече време и не искаме да забавим рабатата с библиотеката по време на нейния процес на работа.

След като вече сме конфигурирали всичко необходимо за работата с новата библиотека, можем да започнем да я използваме. Можем да заменим старата логика за записване на логовете :

Logger.Logger.Instance.LogActivity(LoginValidation.currentUserUsername, LoginValidation.currentUserRole.ToString(), "Update activity to " + username);

След като сме конфигурирали нашият клас Logger, то той вече има създадена инстанция на себе си и няма нужда да създаваме нова. За използването на основната фунцкия ние използваме LoginValidation класа за да можем, да вземем необходимата информация за потренителя.

Може да преизползваме новата функционалност и в LoginValidation, когато потребител успешно влезе в системата:

Logger.Logger.Instance.LogActivity(LoginValidation.currentUserUsername, LoginValidation.currentUserRole.ToString(),"Success Login");

**Приложение**

Github repository: <https://github.com/petiocvetkov/WPF-profect>