Министерство науки и высшего образования Российской Федерации Федеральное государственное бюджетное образовательное учреждение высшего образования

ТОМСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ СИСТЕМ

УПРАВЛЕНИЯ И РАДИОЭЛЕКТРОНИКИ

(ТУСУР)

Кафедра экономической математики, информатики и статистики (ЭМИС)

ОБЩЕЕ ОПИСАНИЕ СИСТЕМЫ

Отчет по практической работе №1 по дисциплине

«Технология программирования»

Студент гр. з-39-2

\_\_\_\_\_\_\_\_\_\_\_\_ / В.О. Михайленко

« » 20 г

Проверил

Ассистент каф. ЭМИС

\_\_\_\_\_\_\_\_\_\_\_\_ / Я.В. Костелей

« \_\_ » \_\_\_\_\_\_\_ 20\_\_ г

Томск 2023

**Цель.** Познакомится с основами модульного тестирования и написать свои unit-тесты.

**Задание**. Для заданного модуля провести модульное тестирование и решить следующие задачи:

* описать задачу модуля и провести его реализацию;
* написать тест-кейсы;
* на основании тест-кейсов написать модульные тесты.

Сформировать отчет, содержащий титульный лист, цели и задачи лабораторной работы, ход работы, тестируемые функции, тест-кейсы, код и описание unit-тестов, выводы.

# Описание модуля

Определим функциональное назначение модулей. В качестве примера мы будем использовать методы из лабораторной работы по проектированию:

bool RegisterForm.checkDoctorData (string login, string password, string repPassword)

IDoctorEntry RegisterForm.onRegisterClick(string login, string password, string repPassword)

С этими методами связано следующее описание последовательности действий пользователем: «Врач выбирает регистрацию в приложении, открывается форма. Вводит логин и пароль (два раза). Проверяются требования к логину и паролю. Программа подключается к базе данных. Производится попытка добавления записи в базу данных.».

Тестировать мы должны как основной поток событий, так и альтернативные. При описании варианта использования регистрации мы предусмотрели следующие особенности обработки:

a) логин существует. Система предлагает придумать новый логин. Пользователь может вернуться в основной поток, или выйти из системы.

б) логин содержит запрещенные символы. Система предлагает исключить запрещенные символы.

в) пароль ненадежен. Система предлагает ввести надежный пароль.

Кроме этого, мы можем определить то, что пользователь:

1) может оставить пустым какое-то поле

2) или может ввести два разных варианта пароля,

3) пароль и логин совпадают.

Также установим требования к паролю (его надежность):

4) более 10 символов,

5) содержит символы цифр,

6) небуквенные символы «@#$%^&\*!»

7) и один символ в верхнем регистре.

Пусть требование к логину будут в том, что:

8) он состоит из букв или цифр или символа подчеркивания (\_).

Кроме этого, могут возникнуть ситуации, когда невозможно подключится к базе данных. Установим, что разрабатываемые методы должны при возникновении альтернативных потоков выбрасывать исключение (Exception).

# Тест-кейсы

На основании основного и альтернативных потоков составим тест-кейсы. Построим структуру описания тест-кейса следующим образом:

* порядковый номер теста;
* английское наименование теста, которое будет использовано в коде;
* краткое описание теста;
* полное наименование теста;
* значения входных данных;
* описание ожидаемого результата.

Для описанных ситуаций, кроме (а), сформируем тест-кейсы.

Основной поток событий:

**Тест**: 001 . Символ: СheckDoctorData\_BaseFlow .

**Название**: Правильные данные для СheckDoctorData.

В функцию переданы данные логина и паролей, соответствующие требованиям.

**Входные данные**: login= "myname\_doctor"

password = "DoctorSuperBest123!"

repPassword = "DoctorSuperBest123!"

**Ожидаемый результат**: true, без исключений.

Для (1):

**Тест**: 002 . Символ: СheckDoctorData\_EmptyLogin .

**Название**: Пустой логин.

Пользователь оставил поле ввода логина пустым.

**Входные данные**: login= ""

password = "DoctorSuperBest123!"

repPassword = "DoctorSuperBest123!"

**Ожидаемый результат**: исключение "Логин не может быть пустым.".

**Тест**: 003 . Символ: СheckDoctorData\_EmptyPassword1 .

**Название**: Пустой первый пароль.

Пользователь оставил поле ввода первого варианта пароля пустым.

**Входные данные**: login= "myname\_doctor"

password = ""

repPassword = "DoctorSuperBest123!"

**Ожидаемый результат**: исключение "Пропущено поле первого ввода пароля.".

**Тест**: 004 . Символ: СheckDoctorData\_EmptyPassword2 .

**Название**: Пустой второй пароль.

Пользователь оставил поле ввода второго варианта пароля пустым.

**Входные данные**: login= "myname\_doctor"

password = "DoctorSuperBest123! "

repPassword = ""

**Ожидаемый результат**: исключение "Пропущено поле второго ввода пароля.".

Для (2):

**Тест**: 005 . Символ: СheckDoctorData\_DifferentPasswords .

**Название**: Пароли не совпадают.

Пользователь ввел в поля пароля разные строки.

**Входные данные**: login= "myname\_doctor"

password = "DoctorSuperBest123!"

repPassword = "DoctorSupeBest123!"

**Ожидаемый результат**: исключение "Пароли не совпадают!".

Для (3):

**Тест**: 006 . Символ: СheckDoctorData\_SameLoginPassword .

**Название**: Логин и пароль совпадают.

Пользователь ввел в поля пароля и логина одну и ту же строку.

**Входные данные**: login= "myname\_doctor"

password = "myname\_doctor"

repPassword = "myname\_doctor"

**Ожидаемый результат**: исключение "Логин и пароль не могут совпадать.".

Для (4):

**Тест**: 007 . Символ: СheckDoctorData\_PasswordLess10Chars .

**Название**: Пароль менее 10 символов.

Пользователь ввел пароль менее 10 символов.

**Входные данные**: login= "myname\_doctor"

password = "Doc123!"

repPassword = "Doc123!"

**Ожидаемый результат**: исключение "Пароль не может быть менее 10 символов.".

Для (5):

**Тест**: 008 . Символ: СheckDoctorData\_PasswordNoNumber .

**Название**: Пароль не содержит цифр.

Пользователь ввел пароль без цифр.

**Входные данные**: login= "myname\_doctor"

password = "DoctorSuperBest!"

repPassword = "DoctorSuperBest!"

**Ожидаемый результат**: исключение "Пароль должен содержать хотя бы один символ цифры.".

Для (6):

**Тест**: 009 . Символ: СheckDoctorData\_PasswordNoExtraChar .

**Название**: Пароль не содержит спецсимволов.

Пользователь ввел пароль без символов «@#$%^&\*!».

**Входные данные**: login= "myname\_doctor"

password = "DoctorSuperBest123"

repPassword = "DoctorSuperBest123"

**Ожидаемый результат**: исключение "Пароль должен содержать хотя бы один символ из @#$%^&\*!".

Для (7):

**Тест**: 010 . Символ: СheckDoctorData\_PasswordNoUpperChar .

**Название**: Пароль не содержит букву в верхнем регистре.

Пользователь ввел пароль без букв в верхнем регистре.

**Входные данные**: login= "myname\_doctor"

password = "doctorsuperbest123!"

repPassword = "doctorsuperbest123!"

**Ожидаемый результат**: исключение "Пароль должен содержать хотя бы один символ в верхнем регистре.".

Для (8):

**Тест**: 011 . Символ: СheckDoctorData\_LoginForbidden .

**Название**: Запрещенный формат логина.

Логин содержит запрещенные символы.

**Входные данные**: login= "myname\_doctor!"

password = "DoctorSuperBest123!"

repPassword = "DoctorSuperBest123!"

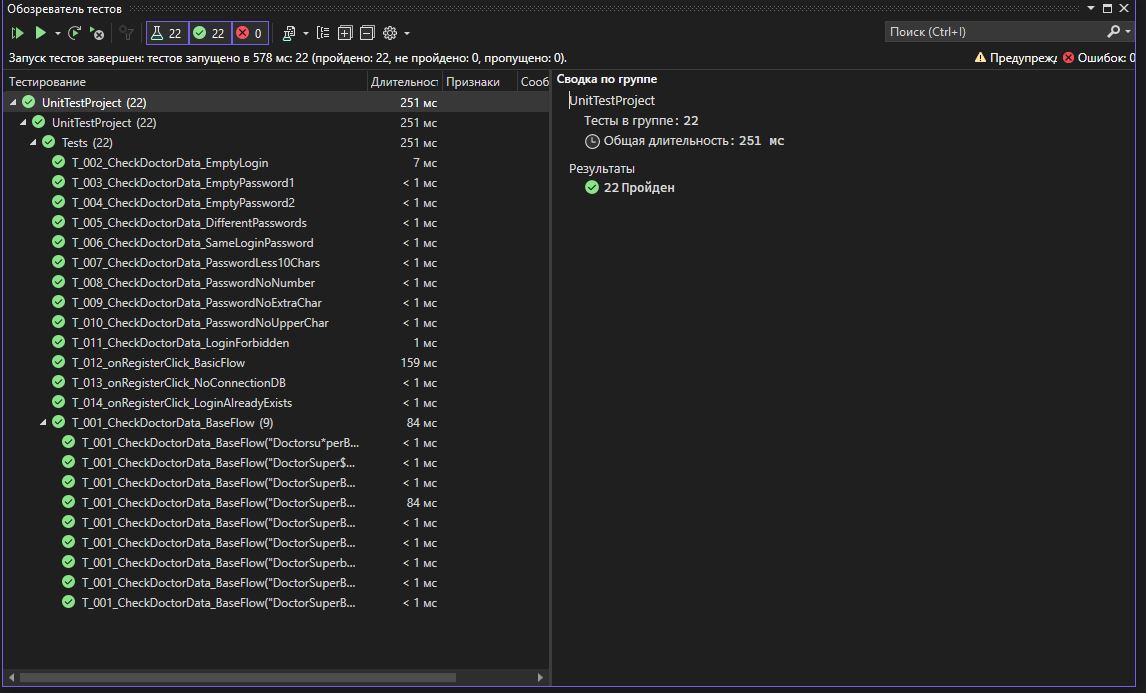
**Ожидаемый результат**: исключение "Логин должен состоять только из цифр, букв и символа \_.".

# Разработка unit-тестов

По методу TDD (Test-Driving Development) разработаем модульные тесты согласно тест-кейсам, используя среду разработки Microsoft Visual Studio 2022.

Затем реализуем модуль, способный пройти созданные тесты и выполнить поставленную задачу. В виде отдельных классов реализуем интерфейсы *IDoctorEntry* и *IDatabaseController*.

Обозреватель тестов с пройденными модульными тестами представлен на рисунке 1.

Рисунок 1 – Результат выполнения тестов

# Вывод

В лабораторной работе мы познакомились с основами модульного тестирования и написали свои unit-тесты. Кроме этого, заданного модуля провели модульное тестирование и решили следующие задачи:

– описали задачу модуля и провели его реализацию;

– написали тест-кейсы для модульного тестирования;

– на основании тест-кейсов написали модульные тесты.

# Приложение A UnitTestProject.Tests

using System;

using NUnit.Framework;

using WinFormsAppLabTest;

namespace UnitTestProject

{

public class Tests

{

/// <summary>

/// Правильные данные для СheckDoctorData.

/// В функцию переданы данные логина и паролей, соответствующие требованиям.

/// </summary>

[TestCase("DoctorSuperBest123!")]

[TestCase("DoctorSuperBest123$")]

[TestCase("DoctorSuperBest13@")]

[TestCase("DoctorSuperBest13#")]

[TestCase("DoctorSuper$Best13")]

[TestCase("DoctorSuperbest13%")]

[TestCase("DoctorSuperBest13^")]

[TestCase("DoctorSuperBes&t13")]

[TestCase("Doctorsu\*perBest13")]

public void T\_001\_СheckDoctorData\_BaseFlow(string value)

{

//подготовка данных

String login = "myname\_doctor";

String password = value;

String repPassword = value;

//ожидаемое значение

bool expectedReturnValue = true;

//подготовка переменной для полученного значения

bool actualReturnValue = false;

//Assert для получения исключения

Assert.DoesNotThrow(() =>

{

actualReturnValue = RegisterForm.checkDoctorData(login, password, repPassword);

});

//Assert для проверки ожидаемого и полученного значения

Assert.AreEqual(expectedReturnValue, actualReturnValue);

}

/// <summary>

/// Пустой логин.

/// Пользователь оставил поле ввода логина пустым.

/// </summary>

[Test]

public void T\_002\_СheckDoctorData\_EmptyLogin()

{

//подготовка данных

String login = "";

String password = "DoctorSuperBest123!";

String repPassword = "DoctorSuperBest123!";

//ожидаемое значение

String expectedExceptionMessage = RegisterForm.ExceptionStrings.EmptyLogin;

//Assert для получения исключения

Exception? exception = Assert.Throws<Exception>(() =>

{

RegisterForm.checkDoctorData(login, password, repPassword);

});

Assert.IsNotNull(exception);

//Assert для проверки ожидаемого и полученного значения

Assert.AreEqual(expectedExceptionMessage, exception.Message);

}

/// <summary>

/// Пустой первый пароль.

/// Пользователь оставил поле ввода первого варианта пароля пустым.

/// </summary>

[Test]

public void T\_003\_СheckDoctorData\_EmptyPassword1()

{

//подготовка данных

String login = "myname\_doctor";

String password = "";

String repPassword = "DoctorSuperBest123!";

//ожидаемое значение

String expectedExceptionMessage = RegisterForm.ExceptionStrings.EmptyPassword1;

//Assert для получения исключения

Exception? exception = Assert.Throws<Exception>(() =>

{

RegisterForm.checkDoctorData(login, password, repPassword);

});

Assert.IsNotNull(exception);

//Assert для проверки ожидаемого и полученного значения

Assert.AreEqual(expectedExceptionMessage, exception.Message);

}

/// <summary>

/// Пустой второй пароль.

/// Пользователь оставил поле ввода второго варианта пароля пустым.

/// </summary>

[Test]

public void T\_004\_СheckDoctorData\_EmptyPassword2()

{

//подготовка данных

String login = "myname\_doctor";

String password = "DoctorSuperBest123!";

String repPassword = "";

//ожидаемое значение

String expectedExceptionMessage = RegisterForm.ExceptionStrings.EmptyPassword2;

//Assert для получения исключения

Exception? exception = Assert.Throws<Exception>(() =>

{

RegisterForm.checkDoctorData(login, password, repPassword);

});

Assert.IsNotNull(exception);

//Assert для проверки ожидаемого и полученного значения

Assert.AreEqual(expectedExceptionMessage, exception.Message);

}

/// <summary>

/// Пароли не совпадают.

/// Пользователь ввел в поля пароля разные строки.

/// </summary>

[Test]

public void T\_005\_СheckDoctorData\_DifferentPasswords()

{

//подготовка данных

String login = "myname\_doctor";

String password = "DoctorSuperBest123!";

String repPassword = "DoctorSupeBest123!";

//ожидаемое значение

String expectedExceptionMessage = RegisterForm.ExceptionStrings.DifferentPasswords;

//Assert для получения исключения

Exception? exception = Assert.Throws<Exception>(() =>

{

RegisterForm.checkDoctorData(login, password, repPassword);

});

Assert.IsNotNull(exception);

//Assert для проверки ожидаемого и полученного значения

Assert.AreEqual(expectedExceptionMessage, exception.Message);

}

/// <summary>

/// Логин и пароль совпадают.

/// Пользователь ввел в поля пароля и логина одну и ту же строку.

/// </summary>

[Test]

public void T\_006\_СheckDoctorData\_SameLoginPassword()

{

//подготовка данных

String login = "myname\_doctor";

String password = "myname\_doctor";

String repPassword = "myname\_doctor";

//ожидаемое значение

String expectedExceptionMessage = RegisterForm.ExceptionStrings.SameLoginPassword;

//Assert для получения исключения

Exception? exception = Assert.Throws<Exception>(() =>

{

RegisterForm.checkDoctorData(login, password, repPassword);

});

Assert.IsNotNull(exception);

//Assert для проверки ожидаемого и полученного значения

Assert.AreEqual(expectedExceptionMessage, exception.Message);

}

/// <summary>

/// Пароль менее 10 символов.

/// Пользователь ввел пароль менее 10 символов.

/// </summary>

[Test]

public void T\_007\_СheckDoctorData\_PasswordLess10Chars()

{

//подготовка данных

String login = "myname\_doctor";

String password = "Doc123!";

String repPassword = "Doc123!";

//ожидаемое значение

String expectedExceptionMessage = RegisterForm.ExceptionStrings.PasswordLess10Chars;

//Assert для получения исключения

Exception? exception = Assert.Throws<Exception>(() =>

{

RegisterForm.checkDoctorData(login, password, repPassword);

});

Assert.IsNotNull(exception);

//Assert для проверки ожидаемого и полученного значения

Assert.AreEqual(expectedExceptionMessage, exception.Message);

}

/// <summary>

/// Пароль не содержит цифр.

/// Пользователь ввел пароль без цифр.

/// </summary>

[Test]

public void T\_008\_СheckDoctorData\_PasswordNoNumber()

{

//подготовка данных

String login = "myname\_doctor";

String password = "DoctorSuperBest!";

String repPassword = "DoctorSuperBest!";

//ожидаемое значение

String expectedExceptionMessage = RegisterForm.ExceptionStrings.PasswordNoNumber;

//Assert для получения исключения

Exception? exception = Assert.Throws<Exception>(() =>

{

RegisterForm.checkDoctorData(login, password, repPassword);

});

Assert.IsNotNull(exception);

//Assert для проверки ожидаемого и полученного значения

Assert.AreEqual(expectedExceptionMessage, exception.Message);

}

/// <summary>

/// Пароль не содержит спецсимволов.

/// Пользователь ввел пароль без символов «@#$%^&\*!».

/// </summary>

[Test]

public void T\_009\_СheckDoctorData\_PasswordNoExtraChar()

{

//подготовка данных

String login = "myname\_doctor";

String password = "DoctorSuperBest123";

String repPassword = "DoctorSuperBest123";

//ожидаемое значение

String expectedExceptionMessage = RegisterForm.ExceptionStrings.PasswordNoExtraChar;

//Assert для получения исключения

Exception? exception = Assert.Throws<Exception>(() =>

{

RegisterForm.checkDoctorData(login, password, repPassword);

});

Assert.IsNotNull(exception);

//Assert для проверки ожидаемого и полученного значения

Assert.AreEqual(expectedExceptionMessage, exception.Message);

}

/// <summary>

/// Пароль не содержит букву в верхнем регистре.

/// Пользователь ввел пароль без букв в верхнем регистре.

/// </summary>

[Test]

public void T\_010\_СheckDoctorData\_PasswordNoUpperChar()

{

//подготовка данных

String login = "myname\_doctor";

String password = "doctorsuperbest123!";

String repPassword = "doctorsuperbest123!";

//ожидаемое значение

String expectedExceptionMessage = RegisterForm.ExceptionStrings.PasswordNoUpperChar;

//Assert для получения исключения

Exception? exception = Assert.Throws<Exception>(() =>

{

RegisterForm.checkDoctorData(login, password, repPassword);

});

Assert.IsNotNull(exception);

//Assert для проверки ожидаемого и полученного значения

Assert.AreEqual(expectedExceptionMessage, exception.Message);

}

/// <summary>

/// Запрещенный формат логина.

/// Логин содержит запрещенные символы.

/// </summary>

[Test]

public void T\_011\_СheckDoctorData\_LoginForbidden()

{

//подготовка данных

String login = "myname\_doctor!";

String password = "DoctorSuperBest123!";

String repPassword = "DoctorSuperBest123!";

//ожидаемое значение

String expectedExceptionMessage = RegisterForm.ExceptionStrings.LoginForbidden;

//Assert для получения исключения

Exception? exception = Assert.Throws<Exception>(() =>

{

RegisterForm.checkDoctorData(login, password, repPassword);

});

Assert.IsNotNull(exception);

//Assert для проверки ожидаемого и полученного значения

Assert.AreEqual(expectedExceptionMessage, exception.Message);

}

public class MockDoctorEntry : IDoctorEntry

{

public string ID { get; set; }

public string Login { get; set; }

public string Password { get; set; }

}

public class MockDatabaseController\_NoConnection : IDatabaseController

{

public IDoctorEntry getNewDoctorEntry() { throw new NotImplementedException(); }

public bool login(string login, string password) { throw new NotImplementedException(); }

public bool tryConnectDB() { return false; }

public bool tryCreateAccount(string login, string password) { throw new NotImplementedException(); }

}

public class MockDatabaseController\_LoginExists : IDatabaseController

{

public IDoctorEntry getNewDoctorEntry() { throw new NotImplementedException(); }

public bool login(string login, string password) { return true; }

public bool tryConnectDB() { return true; }

public bool tryCreateAccount(string login, string password) { return false; }

}

public class MockDatabaseController\_OK : IDatabaseController

{

public IDoctorEntry getNewDoctorEntry() { return new MockDoctorEntry() { ID="1", Login= "myname\_doctor", Password= "DoctorSuperBest123!" }; }

public bool login(string login, string password) { return true; }

public bool tryConnectDB() { return true; }

public bool tryCreateAccount(string login, string password) { return true; }

}

/// <summary>

/// Регистрация успешна.

/// Процесс регистрации успешный.

/// </summary>

[Test]

public void T\_012\_onRegisterClick\_BasicFlow()

{

//подготовка данных

String login = "myname\_doctor";

String password = "DoctorSuperBest123!";

String repPassword = "DoctorSuperBest123!";

//подготовка данных

RegisterForm registerForm = new RegisterForm();

registerForm.controller = new MockDatabaseController\_OK();

IDoctorEntry doctorEntry = null;

//Assert для получения исключения

Assert.DoesNotThrow(() =>

{

doctorEntry = registerForm.onRegisterClick(login, password, repPassword);

});

//Assert для проверки ожидаемого и полученного значения

Assert.IsNotNull(doctorEntry);

Assert.AreEqual(doctorEntry.Login, login);

Assert.AreEqual(doctorEntry.Password, password);

}

/// <summary>

/// Невозможно подключится к БД.

/// Нет доступа к базе данных, поэтому не можем зарегистрироваться.

/// </summary>

[Test]

public void T\_013\_onRegisterClick\_NoConnectionDB()

{

//подготовка данных

String login = "myname\_doctor";

String password = "DoctorSuperBest123!";

String repPassword = "DoctorSuperBest123!";

//ожидаемое значение

String expectedExceptionMessage = RegisterForm.ExceptionStrings.NoConnectionDB;

RegisterForm registerForm = new RegisterForm();

registerForm.controller = new MockDatabaseController\_NoConnection();

//Assert для получения исключения

Exception? exception = Assert.Throws<Exception>(() =>

{

registerForm.onRegisterClick(login, password, repPassword);

});

Assert.IsNotNull(exception);

//Assert для проверки ожидаемого и полученного значения

Assert.AreEqual(expectedExceptionMessage, exception.Message);

}

/// <summary>

/// Невозможно подключится к БД.

/// Нет доступа к базе данных, поэтому не можем зарегистрироваться.

/// </summary>

[Test]

public void T\_014\_onRegisterClick\_LoginAlreadyExists()

{

//подготовка данных

String login = "myname\_doctor";

String password = "DoctorSuperBest123!";

String repPassword = "DoctorSuperBest123!";

//ожидаемое значение

String expectedExceptionMessage = RegisterForm.ExceptionStrings.LoginAlreadyExists;

RegisterForm registerForm = new RegisterForm();

registerForm.controller = new MockDatabaseController\_LoginExists();

//Assert для получения исключения

Exception? exception = Assert.Throws<Exception>(() =>

{

registerForm.onRegisterClick(login, password, repPassword);

});

Assert.IsNotNull(exception);

//Assert для проверки ожидаемого и полученного значения

Assert.AreEqual(expectedExceptionMessage, exception.Message);

}

}

}

# Приложение Б WinFormsAppLabTest

## WinFormsAppLabTest.RegisterForm

namespace WinFormsAppLabTest

{

public partial class RegisterForm : Form

{

public RegisterForm()

{

InitializeComponent();

checkDoctorData("myname\_doctor", "doctorSuperBest123!", "doctorSuperBest123!");

}

public static class ExceptionStrings

{

public const string EmptyLogin = "Логин не может быть пустым.";

public const string EmptyPassword1 = "Пропущено поле первого ввода пароля.";

public const string EmptyPassword2 = "Пропущено поле второго ввода пароля.";

public const string DifferentPasswords = "Пароли не совпадают!";

public const string SameLoginPassword = "Логин и пароль не могут совпадать.";

public const string PasswordLess10Chars = "Пароль не может быть менее 10 символов.";

public const string PasswordNoNumber = "Пароль должен содержать хотя бы один символ цифры.";

public const string PasswordNoExtraChar = "Пароль должен содержать хотя бы один символ из @#$%^&\*! .";

public const string PasswordNoUpperChar = "Пароль должен содержать хотя бы один символ в верхнем регистре.";

public const string LoginForbidden = "Логин должен состоять только из цифр, букв и символа \_.";

public const string NoConnectionDB = "Нет доступа к базе данных, проверьте подключение.";

public const string LoginAlreadyExists = "Уже существует пользователь с данным логином.";

}

public IDatabaseController controller=null;

public IDoctorEntry onRegisterClick(string login, string password, string repPassword)

{

if (checkDoctorData(login, password, repPassword))

{

if (controller.tryConnectDB())

{

if (controller.tryCreateAccount(login, password))

{

IDoctorEntry doctor = controller.getNewDoctorEntry();

controller.login(doctor.Login, doctor.Password);

return doctor;

}

else

{

throw new Exception(ExceptionStrings.LoginAlreadyExists);

}

}

else

{

throw new Exception(ExceptionStrings.NoConnectionDB);

}

}

return null;

}

public static bool checkDoctorData(string login,

string password,

string repPassword)

{

if ( login==null || login.Length==0 )

{

throw new Exception(ExceptionStrings.EmptyLogin);

}

if (password == null || password.Length == 0)

{

throw new Exception(ExceptionStrings.EmptyPassword1);

}

if (repPassword == null || repPassword.Length == 0)

{

throw new Exception(ExceptionStrings.EmptyPassword2);

}

if (password != repPassword)

{

throw new Exception(ExceptionStrings.DifferentPasswords);

}

if (login == password)

{

throw new Exception(ExceptionStrings.SameLoginPassword);

}

if (password.Length < 10)

{

throw new Exception(ExceptionStrings.PasswordLess10Chars);

}

System.Text.RegularExpressions.Regex regex = new System.Text.RegularExpressions.Regex(@"[0-9]");

if (!regex.IsMatch(password))

{

throw new Exception(ExceptionStrings.PasswordNoNumber);

}

regex = new System.Text.RegularExpressions.Regex(@"[@#$%^&\*!]");

if (!regex.IsMatch(password))

{

throw new Exception(ExceptionStrings.PasswordNoExtraChar);

}

regex = new System.Text.RegularExpressions.Regex(@"[A-ZА-Я]");

if (!regex.IsMatch(password))

{

throw new Exception(ExceptionStrings.PasswordNoUpperChar);

}

regex = new System.Text.RegularExpressions.Regex(@"^[0-9A-ZА-Яa-zа-я\_]+$");

if (!regex.IsMatch(login))

{

throw new Exception(ExceptionStrings.LoginForbidden);

}

return true;

}

}

}

# WinFormsAppLabTest. IDatabaseController

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace WinFormsAppLabTest

{

public interface IDatabaseController

{

public bool tryConnectDB();

public bool tryCreateAccount(string login, string password);

public IDoctorEntry getNewDoctorEntry();

public bool login(string login, string password);

}

}

# WinFormsAppLabTest. IDoctorEntry

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace WinFormsAppLabTest

{

public interface IDoctorEntry

{

public string ID { get; set; }

public string Login { get; set; }

public string Password { get;set; }

}

}