Министерство науки и высшего образования российской федерации

Федеральное государственное автономное образовательное учреждение высшего образования

ТОМСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ СИСТЕМ УПРАВЛЕНИЯ И РАДИОЭЛЕКТРОНИКИ (ТУСУР)

[Кафедра экономической математики, информатики и статистики](http://www.tusur.ru/ru/faculties/ekf/chairs/emis) (ЭМИС)

МОДУЛЬНОЕ ТЕСТИРОВАНИЕ (UNIT-ТЕСТЫ)

Отчет по лабораторной работе по дисциплине

«Технология программирования»

Студент гр. 541-1

\_\_\_\_\_\_\_\_\_\_\_\_/ Федькин М.Д.

«20» апреля 2025 г.

Проверил:

Доцент каф. ЭМИС

/ Костелей Я.В.

оценка

«\_\_\_» \_\_\_\_\_\_\_\_\_\_ 2025 г.

Томск 2025

# Лабораторная работа №3

# «Модульное тестирование (Unit-тесты)»

Цель: познакомится с основами модульного тестирования и написать свои unit-тесты.

Для выбранного 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.

**Ход работы**

Определим функциональное назначение модулей. Далее будут рассматриваться методы из первой лабораторной работы по проектированию:

DoctorEntry onRegisterClick(string login, string password, string repPassword)

static bool checkDoctorData(string login, string password, string repPassword)

С этими методами связано следующее описание последовательности действий пользователем:

Пользователь находится на экране регистрации. Когда он хочет создать новую учётную запись, он вводит логин, пароль и подтверждение пароля, после чего нажимает кнопку «Зарегистрироваться». Это действие инициирует процесс регистрации.

onRegisterClick(string login, string password, string repPassword) – это метод, который вызывается после того, как пользователь нажал на кнопку “Зарегистрироваться” Система проверяет, что введенные данные соответствуют всем условиям регистрации. Если данные валидны, данные проходят регистрацию. Пользователю выводится подтверждение регистрации.

Параметры:

login: логин, введённый пользователем.

password: пароль, введённый пользователем.

repPassword: подтверждение пароля (должно совпадать с password).

checkDoctorData(string login, string password, string repPassword) – это метод, который проверяет данные перед регистрацией. Он выполняет валидацию полей по заданным критериям. После чего возвращает true, если данные корректны, и false, если есть ошибки.

Параметры:

login: логин, введённый пользователем.

password: пароль, введённый пользователем.

repPassword: подтверждение пароля (должно совпадать с password).

При описании варианта использования регистрации предусмотрены следующие альтернативные потоки

а) Пустой логин. Пользователь нажал «Зарегистрироваться», но не ввёл логин. Система выводит сообщение, что логин не введен.

б) Пустой пароль (первый). Пользователь ввёл логин, но оставил поле первого пароля пустым. Система обнаруживает, что у пользователя не введен первый пароль. Система выводит сообщение, что нет первого пароля.

в) Пустой повтор пароля. Пользователь ввёл логин, но оставил поле второго пароля пустым. Система обнаруживает, что у пользователя не введен второй пароль. Система выводит сообщение, что нет второй пароля.

г) Пароли не совпадают. Первый и повтор пароля различаются. Система обнаруживает, что у пользователя не совпадают введенные пароли. Система выводит сообщение, что пароли не совпадают.

д) Логин и пароль совпадают. Пользователь ввёл в оба поля одну и ту же строку. Система обнаруживает, что у пользователя логин и пароль совпадают. Система выводит сообщение, что логин и пароль совпадают.

е) Пароль менее 10 символов. Пользователь ввёл пароль, длина пароля меньше 10. Система обнаруживает, что у пользователя не верная длина пароль. Система выводит сообщение, что пароль слишком маленький.

ж) Пароль без цифр / без спецсимволов / без заглавной буквы. Система должна обнаружить эту несоответствие и вывести сообщение пользователю.

з) Логин содержит запрещённые символы. Пользователь ввёл логин, который содержит запрещенные символы. Система обнаруживает, что у пользователя ошибка в логине. Система выводит сообщение, что логин не соответствует требованиям.

При регистрации пользователя система должна убедиться, что он действительно добавляется в базу данных.

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

На основании основного и альтернативных потоков составим тест-кейсы.

Основной поток событий:

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

**Название**: Правильные данные для проверки регистрации.

Все поля заполнены корректно, пароли совпадают, логин и пароль соответствуют требованиям.

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

password = "doctorSuperBest123!""

repPassword = "doctorSuperBest123!"

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

Альтернативные потоки событий:

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

**Название**: Пустой логин.

Поле логина пустое.

**Входные данные**: login = ""

password = "doctorSuperBest123!"

repPassword = "doctorSuperBest123!"

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

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

**Название**: Пустой первый пароль.

Поле первого пароля пустое.

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

password = ""

repPassword = "doctorSuperBest123!"

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

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

**Название**: Пустой второй пароль.

Поле второго пароля пустое.

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

password = "doctorSuperBest123!"

repPassword = ""

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

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

**Название**: Пароли не совпадают.

Пароль и подтверждение разные.

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

password = "passA123!"

repPassword = "passB123!"

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

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

**Название**: Логин и пароль совпадают.

Логин и пароль одинаковые строки.

**Входные данные**: login = "sameValue"

password = "sameValue"

repPassword = "sameValue"

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

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

**Название**: Пароль менее 10 символов.

Длина пароля меньше 10.

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

password = "Short1!A"

repPassword = "Short1!A"

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

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

**Название**: Пароль без цифр.

Пароль не содержит цифр.

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

password = "NoNumber!A"

repPassword = "NoNumber!A"

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

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

**Название**: Пароль без спецсимволов.

Пароль не содержит спецсимволов.

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

password = "NoExtra123A"

repPassword = "NoExtra123A"

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

**Тест**: 0010 . Символ: CheckDoctorData\_PasswordNoUpperChar .

**Название**: Пароль без заглавной буквы.

Пароль не содержит буквы в верхнем регистре.

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

password = "lowercase123!"

repPassword = "lowercase123!"

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

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

**Название**: Запрещённый формат логина.

Логин содержит недопустимые символы.

**Входные данные**: login = "bad!login"

password = "Valid123!A"

repPassword = "Valid123!A"

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

После написания тест-кейсов создан проект Windows Forms .Net (Майкрософт) версия 6.0. Имя формы Form1 изменено на RegisterForm и зарезервировано место под метод (результат представлен на рисунке 1)

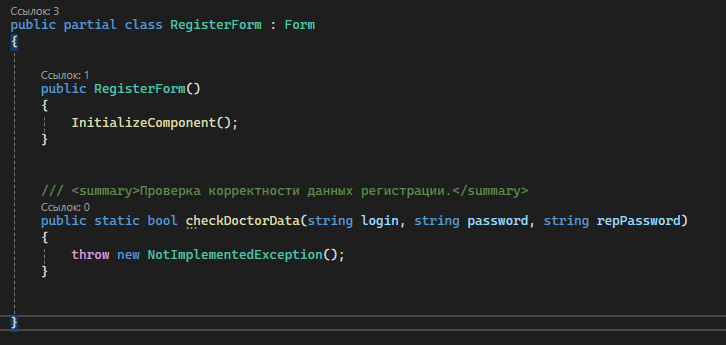


Рисунок 1 – Скриншот кода определения метода checkDoctorData

Далее добавлен еще один проект в решение – Тестовый проект NUnit. Структура решения в обозревателе проектов представлена на рисунке 2. Структура файла тестов представлена на рисунке 3.

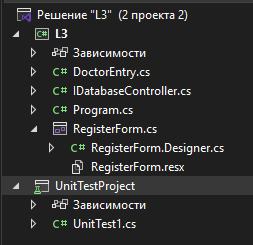


Рисунок 2 – Заготовка структуры решения

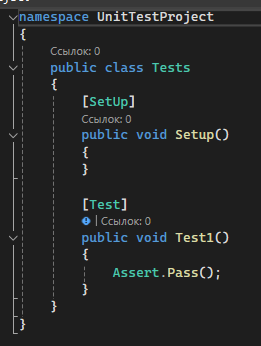


Рисунок 3 – Структура файла тестов

Для того, чтобы в тестовом проекте можно было тестировать модули из проекта WinForms определены зависимости для тестового проекта через контекстное меню на последовательную сборку и на доступ к классам (результаты представлены на рисунках 4-5).

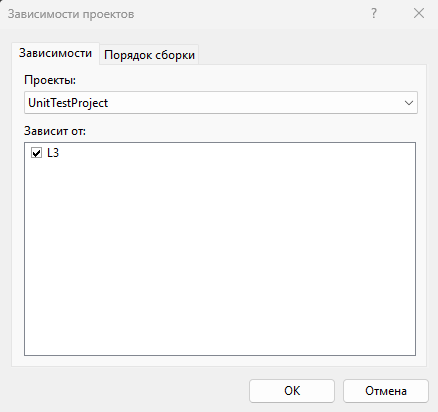


Рисунок 4 – Определение зависимости для тестового проекта

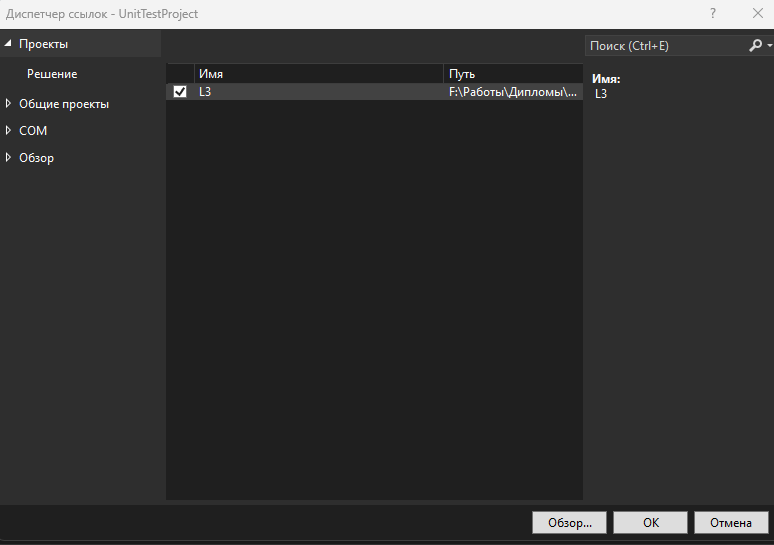


Рисунок 5 – Добавление ссылки на проект

Также для проекта тестирования через контекстное меню вызвана вкладка Свойства проекта и указана целевая операционная система для того, чтобы проекты были согласованы. Результат представлен на рисунке 6.

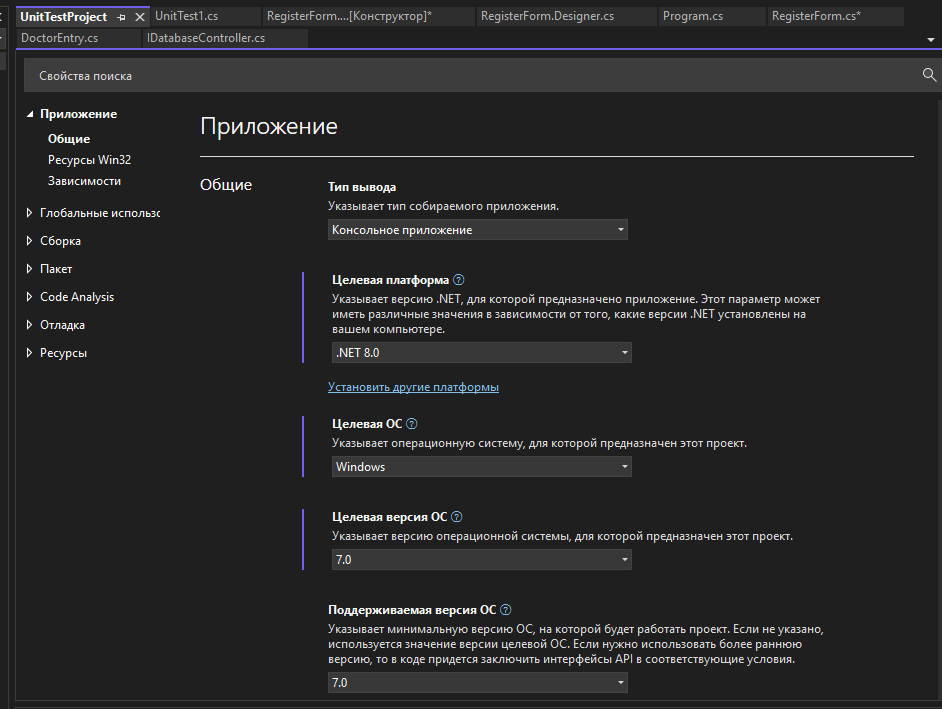


Рисунок 6 – Настройка проекта тестирование

Далее сформирован блок первого теста, скриншот кода которого представлен на рисунке 7.



Рисунок 7 – Скриншот блока первого теста

Запущены тесты через панель Обозревателя тестов. На рисунке 8 представлены результаты прохождения теста. Из рисунка видно, что тесты провалены.

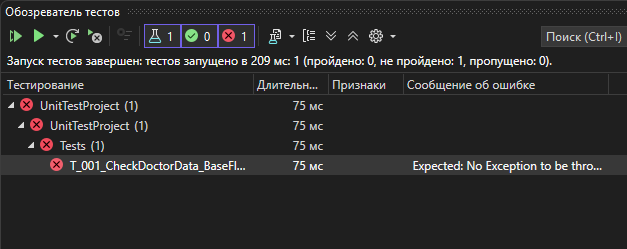


Рисунок 8 – Скриншот первого запуска теста

Написаны блоки остальных тестов. На рисунке 9 представлены результаты прохождения тестов. Код реализации представлен в приложении А.

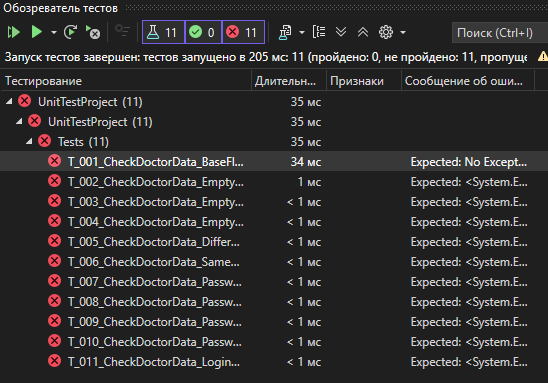


Рисунок 9 – Скриншот обозревателя тестов

Внесены условия для тестов в тестируемый проект и запущены тесты. На рисунке 10 представлены результаты прохождения тестов после добавления блоков кода в тестируемый проект. Код реализации представлен в приложении Б.

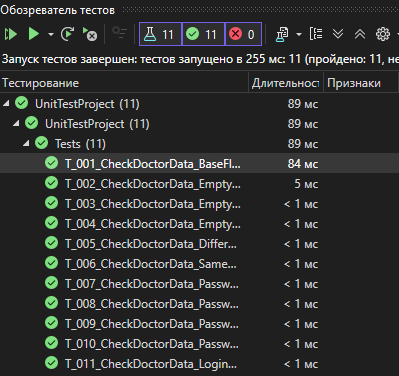


Рисунок 10 – Скриншот обозревателя тестов после добавления блоков кода в тестируемый проект

Чтобы полностью реализовать ситуацию, когда логин уже существует, нужно получить доступ к базе данных в классе IDatabaseController через метод tryCreateAccount. Также, чтобы реализовать весь запланированный код метода onRegisterClick, необходимы следующие классы, которые далее будут заменены на заглушки в тестах.

Общая схема работы onRegisterClick в том, что Сначала вызывается checkDoctorData(login, password, repPassword) — валидация введённых данных, далее делаем попытку подключения к базе данных (в случае невозможности подключится нужно бросить исключение), далее пробуем добавить пользователя, в случае неуспеха должны сообщить о том, что это невозможно. На рисунке 11 представлен скриншот UML-диаграммы последовательностей. На рисунке 12 представлен скриншот UML-диаграммы классов.

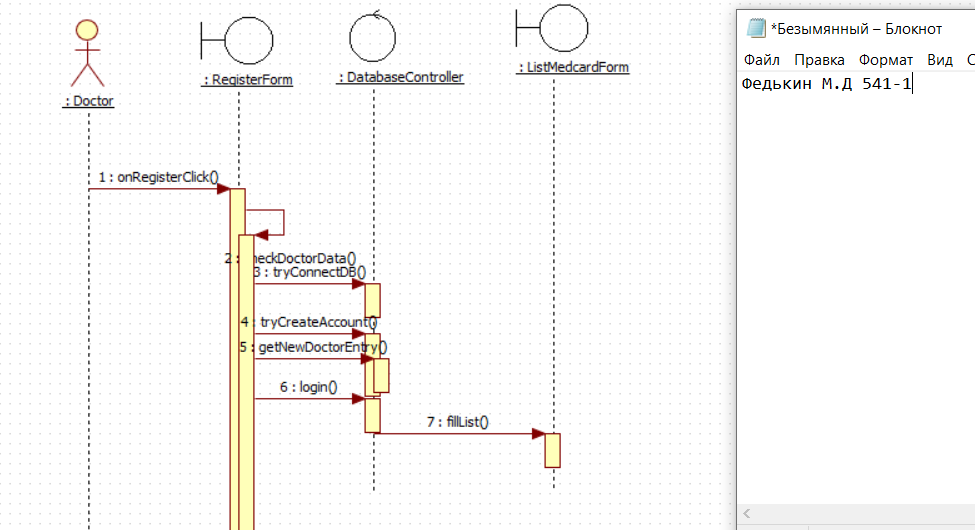


Рисунок 11 – Скриншот UML-диаграммы последовательностей

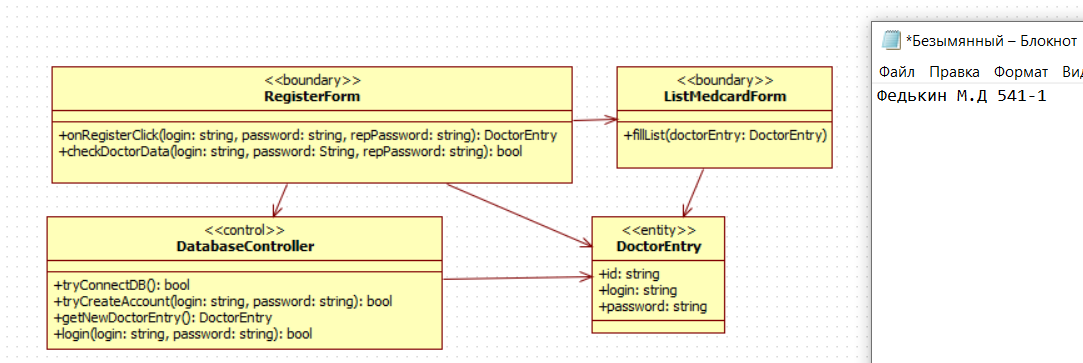


Рисунок 12 – Скриншот UML-диаграммы классов

Сформированы следующие тест-кейсы.

**Тест**: 012 . Символ:  onRegisterClick\_Success .

**Название**: Регистрация прошла успешно.

Процесс регистрации успешный.

**Входные данные**: login = "user1"

password = "Passw0rd!A"

repPassword = "Passw0rd!A"

**Ожидаемый результат**: нет исключений, возвращается DoctorEntry с id = 42, Login = "user1"

**Тест**: 013 . Символ:  onRegisterClick\_NoConnectionDB .

**Название**: Невозможно подключиться к БД.

Метод onRegisterClick вызывается, но контроллер не подключается.

**Входные данные**: login = "user1"

password = "Passw0rd!A"

repPassword = "Passw0rd!A"

dbMock.CanConnect = false

**Ожидаемый результат**: исключение с сообщением "Нет доступа к базе данных, проверьте подключение."

**Тест**: 014 . Символ:  onRegisterClick\_LoginAlreadyExists .

**Название**: Логин уже существует.

Метод onRegisterClick вызывается, контроллер подключается, но tryCreateAccount возвращает false.

**Входные данные**: login = "user1"

password = "Passw0rd!A"

repPassword = "Passw0rd!A"

dbMock.CanConnect = false

dbMock.CanCreate = false

**Ожидаемый результат**: исключение с сообщением " Уже существует пользователь с данным логином."

Для класса RegisterForm внесены изменения. Определено поле controller типа IDatabaseController, а также сам метод onRegisterClick, который должен в зависимости от метода tryConnectDB сообщать об ошибках исключениями, или проверять возможность регистрации пользователя.

В проекте тестов создан класс-заглушка, реализующий интерфейс DoctorEntry. Также создан класс, реализующий IDatabaseController. Результат представлен на рисунке 13.

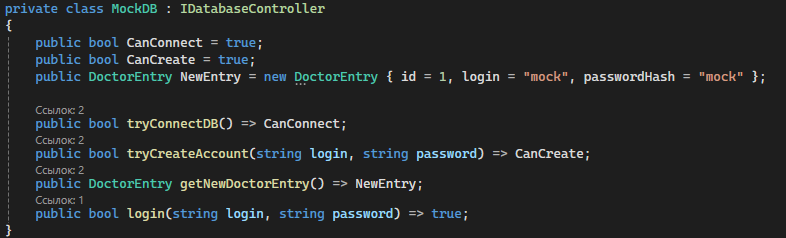


Рисунок 13 – Скриншот блока кода, реализующий класс мок контроллера

После того, как все готово, написаны тесты. Код реализации представлен в приложении А. Результат прохождения текстов представлен на рисунке 14.

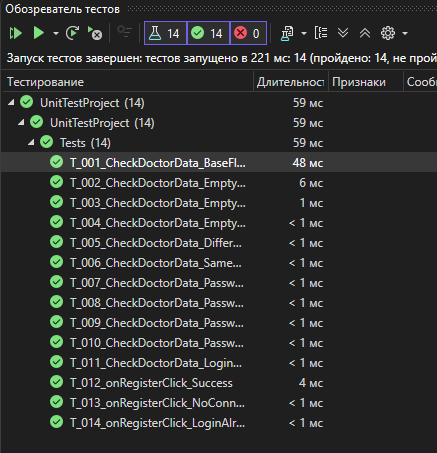


Рисунок 14 – Скриншот обозревателя тестов после добавления блоков кода в тестируемый проект

Код реализации IDatabaseController и DoctorEntry представлен в приложениях В и Г соответственно.

**Выводы**

В результате выполнения лабораторной работы написаны unit-тесты.

Для выбранного модуля проведено модульное тестирование и решены следующие задачи:

* описана задача модуля и проведена его реализация;
* написаны тест-кейсы;
* на основании тест-кейсов написаны модульные тесты.

**Приложение А**

**UnitTestProject.Tests**

using L3;

namespace UnitTestProject

{

public class Tests

{

private RegisterForm form;

private MockDB dbMock;

[SetUp]

public void Setup()

{

form = new RegisterForm();

dbMock = new MockDB();

form.controller = dbMock;

}

[TearDown]

public void TearDown()

{

form.Dispose();

}

// 001: Базовый поток проверки данных

[Test]

public void T\_001\_CheckDoctorData\_BaseFlow()

{

Assert.DoesNotThrow(() =>

Assert.IsTrue(RegisterForm.checkDoctorData(

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

}

// 002: Пустой логин

[Test]

public void T\_002\_CheckDoctorData\_EmptyLogin()

{

var ex = Assert.Throws<Exception>(() =>

RegisterForm.checkDoctorData("", "doctorSuperBest123!", "doctorSuperBest123!"));

Assert.AreEqual(RegisterForm.ExceptionStrings.EmptyLogin, ex.Message);

}

// 003: Пустой первый пароль

[Test]

public void T\_003\_CheckDoctorData\_EmptyPassword1()

{

var ex = Assert.Throws<Exception>(() =>

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

Assert.AreEqual(RegisterForm.ExceptionStrings.EmptyPassword1, ex.Message);

}

// 004: Пустой второй пароль

[Test]

public void T\_004\_CheckDoctorData\_EmptyPassword2()

{

var ex = Assert.Throws<Exception>(() =>

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

Assert.AreEqual(RegisterForm.ExceptionStrings.EmptyPassword2, ex.Message);

}

// 005: Несовпадение паролей

[Test]

public void T\_005\_CheckDoctorData\_DifferentPasswords()

{

var ex = Assert.Throws<Exception>(() =>

RegisterForm.checkDoctorData("myname\_doctor", "passA123!", "passB123!"));

Assert.AreEqual(RegisterForm.ExceptionStrings.DifferentPasswords, ex.Message);

}

// 006: Логин и пароль совпадают

[Test]

public void T\_006\_CheckDoctorData\_SameLoginPassword()

{

var ex = Assert.Throws<Exception>(() =>

RegisterForm.checkDoctorData("sameValue", "sameValue", "sameValue"));

Assert.AreEqual(RegisterForm.ExceptionStrings.SameLoginPassword, ex.Message);

}

// 007: Пароль менее 10 символов

[Test]

public void T\_007\_CheckDoctorData\_PasswordLess10Chars()

{

var ex = Assert.Throws<Exception>(() =>

RegisterForm.checkDoctorData("myname\_doctor", "Short1!A", "Short1!A"));

Assert.AreEqual(RegisterForm.ExceptionStrings.PasswordLess10Chars, ex.Message);

}

// 008: Пароль без цифр

[Test]

public void T\_008\_CheckDoctorData\_PasswordNoNumber()

{

var ex = Assert.Throws<Exception>(() =>

RegisterForm.checkDoctorData("myname\_doctor", "NoNumber!A", "NoNumber!A"));

Assert.AreEqual(RegisterForm.ExceptionStrings.PasswordNoNumber, ex.Message);

}

// 009: Пароль без спецсимволов

[Test]

public void T\_009\_CheckDoctorData\_PasswordNoExtraChar()

{

var ex = Assert.Throws<Exception>(() =>

RegisterForm.checkDoctorData("myname\_doctor", "NoExtra123A", "NoExtra123A"));

Assert.AreEqual(RegisterForm.ExceptionStrings.PasswordNoExtraChar, ex.Message);

}

// 010: Пароль без заглавной буквы

[Test]

public void T\_010\_CheckDoctorData\_PasswordNoUpperChar()

{

var ex = Assert.Throws<Exception>(() =>

RegisterForm.checkDoctorData("myname\_doctor", "lowercase123!", "lowercase123!"));

Assert.AreEqual(RegisterForm.ExceptionStrings.PasswordNoUpperChar, ex.Message);

}

// 011: Запрещенный формат логина

[Test]

public void T\_011\_CheckDoctorData\_LoginForbidden()

{

var ex = Assert.Throws<Exception>(() =>

RegisterForm.checkDoctorData("bad!login", "Valid123!A", "Valid123!A"));

Assert.AreEqual(RegisterForm.ExceptionStrings.LoginForbidden, ex.Message);

}

// 012: onRegisterClick - успех

[Test]

public void T\_012\_onRegisterClick\_Success()

{

dbMock.CanConnect = true;

dbMock.CanCreate = true;

dbMock.NewEntry = new DoctorEntry { id = 42, login = "user1", passwordHash = "Passw0rd!A" };

DoctorEntry result = null;

Assert.DoesNotThrow(() => result = form.onRegisterClick("user1", "Passw0rd!A", "Passw0rd!A"));

Assert.IsNotNull(result);

Assert.AreEqual(42, result.id);

Assert.AreEqual("user1", result.login);

}

// 013: onRegisterClick - нет подключения

[Test]

public void T\_013\_onRegisterClick\_NoConnectionDB()

{

dbMock.CanConnect = false;

var ex = Assert.Throws<Exception>(() => form.onRegisterClick("user1", "Passw0rd!A", "Passw0rd!A"));

Assert.AreEqual(RegisterForm.ExceptionStrings.NoConnectionDB, ex.Message);

}

// 014: onRegisterClick - логин занят

[Test]

public void T\_014\_onRegisterClick\_LoginAlreadyExists()

{

dbMock.CanConnect = true;

dbMock.CanCreate = false;

var ex = Assert.Throws<Exception>(() => form.onRegisterClick("user1", "Passw0rd!A", "Passw0rd!A"));

Assert.AreEqual(RegisterForm.ExceptionStrings.LoginAlreadyExists, ex.Message);

}

// Мок контроллера

private class MockDB : IDatabaseController

{

public bool CanConnect = true;

public bool CanCreate = true;

public DoctorEntry NewEntry = new DoctorEntry { id = 1, login = "mock", passwordHash = "mock" };

public bool tryConnectDB() => CanConnect;

public bool tryCreateAccount(string login, string password) => CanCreate;

public DoctorEntry getNewDoctorEntry() => NewEntry;

public bool login(string login, string password) => true;

}

}

}

**Приложение Б**

**L3.RegisterForm**

namespace L3

{

public partial class RegisterForm : Form

{

public IDatabaseController controller { get; set; } = null;

public RegisterForm()

{

InitializeComponent();

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

}

/// <summary>Строки исключений для модуля регистрации.</summary>

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.";

}

/// <summary>Проверка корректности данных регистрации.</summary>

public static bool checkDoctorData(string login, string password, string repPassword)

{

if (string.IsNullOrWhiteSpace(login))

throw new Exception(ExceptionStrings.EmptyLogin);

if (string.IsNullOrWhiteSpace(password))

throw new Exception(ExceptionStrings.EmptyPassword1);

if (string.IsNullOrWhiteSpace(repPassword))

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);

var 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-Za-zА-Яа-я\_]+$");

if (!regex.IsMatch(login))

throw new Exception(ExceptionStrings.LoginForbidden);

return true;

}

/// <summary>Обработчик кнопки "Зарегистрироваться".</summary>

private void buttonRegister\_Click(object sender, EventArgs e)

{

try

{

var doctor = onRegisterClick(textBoxLogin.Text, textBoxPassword.Text, textBoxRepPassword.Text);

MessageBox.Show($"Регистрация успешна. ID={doctor.id}", "Успех", MessageBoxButtons.OK, MessageBoxIcon.Information);

}

catch (Exception ex)

{

labelError.Text = ex.Message;

labelError.Visible = true;

}

}

/// <summary>Основная логика регистрации.</summary>

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

{

checkDoctorData(login, password, repPassword);

if (!controller.tryConnectDB())

throw new Exception(ExceptionStrings.NoConnectionDB);

if (!controller.tryCreateAccount(login, password))

throw new Exception(ExceptionStrings.LoginAlreadyExists);

return controller.getNewDoctorEntry();

}

}

}

**Приложение В**

**L3.IDatabaseController**

namespace L3

{

/// <summary>

/// Интерфейс контроллера БД для регистрации врачей.

/// </summary>

public interface IDatabaseController

{

bool tryConnectDB();

bool tryCreateAccount(string login, string password);

DoctorEntry getNewDoctorEntry();

bool login(string login, string password);

}

}

**Приложение Г**

**L3.DoctorEntry**

namespace L3

{

public class DoctorEntry

{

public int id{ get; set; }

public string login { get; set; }

public string passwordHash { get; set;}

}

}