Министерство науки и высшего образования РФ  
ФГАОУ ВО “Омский государственный технический университет”

Кафедра “Автоматизированные системы обработки информации и управления”

**Расчётно-графическая работа**

на тему “Тестирование НАЗВАНИЕ ТЕМЫ”

по дисциплине “Тестирование программного обеспечения”

студента ФАМИЛИЯ ИМЯ ОТЧЕСТВО ПЭ-221

Специальность 09.05.01 Применение и эксплуатация автоматизированных систем специального назначения

Старший преподаватель И. П. Убалехт

Студент И. О. Фамилия

Омск 2025

**СОДЕРЖАНИЕ**

[ВВЕДЕНИЕ 3](#_Toc186814711)

[1. Программный комплекс 4](#_Toc186814712)

[2. Тестирование 6](#_Toc186814713)

[ЗАКЛЮЧЕНИЕ 8](#_Toc186814714)

[Приложение A1 (обязательное) Листинг кода CreditRequestContractTests 9](#_Toc186814715)

[Приложение A2 (обязательное) Листинг кода MortgageCalculationResponseContractTests 11](#_Toc186814716)

[Приложение A3 (обязательное) Листинг кода MortgageRequestContractTests 13](#_Toc186814717)

[Приложение A4 (обязательное) Листинг кода MortgageRequestContractTests 15](#_Toc186814718)

[Приложение B1 (обязательное) Листинг кода MortgageCalculatorTests 17](#_Toc186814719)

[Приложение B2 (обязательное) Листинг кода CreditEligibilityCheckerTests 20](#_Toc186814720)

[Приложение C1 (обязательное) Листинг кода ConsoleServiceTests 22](#_Toc186814721)

[Приложение C2 (обязательное) Листинг кода FileServiceTests 25](#_Toc186814722)

[Приложение D (обязательное) Листинг кода MortgageServiceTests 28](#_Toc186814723)

[Приложение E1 (обязательное) Листинг кода RGR.Core.Tests.IoCModuleTests 31](#_Toc186814724)

[Приложение E2 (обязательное) Листинг кода RGR.IO.Tests.IoCModuleTests 34](#_Toc186814725)

ВВЕДЕНИЕ

Тестирование программного обеспечения является ключевым этапом разработки, обеспечивающим его надёжность, качество и соответствие требованиям. Данная работа посвящена изучению и тестированию консольного приложения, предназначенного для расчёта ипотеки, управления заявками на кредит и проверки их соответствия установленным критериям. В процессе исследования рассмотрены структура программного решения, особенности реализации сервисов и используемые методы тестирования. Работа включает анализ юнит-тестов, разработанных с применением NUnit3 и современных подходов, таких как внедрение зависимостей (IoC) и разделение ответственности между модулями.

1. Программный комплекс

Программный комплекс представляет собой консольное приложение (рисунок 1.1), которое выполняет расчёт кредита и принимает заявки на кредит. В случае успешной проверки заявок данные сохраняются в формате JSON.

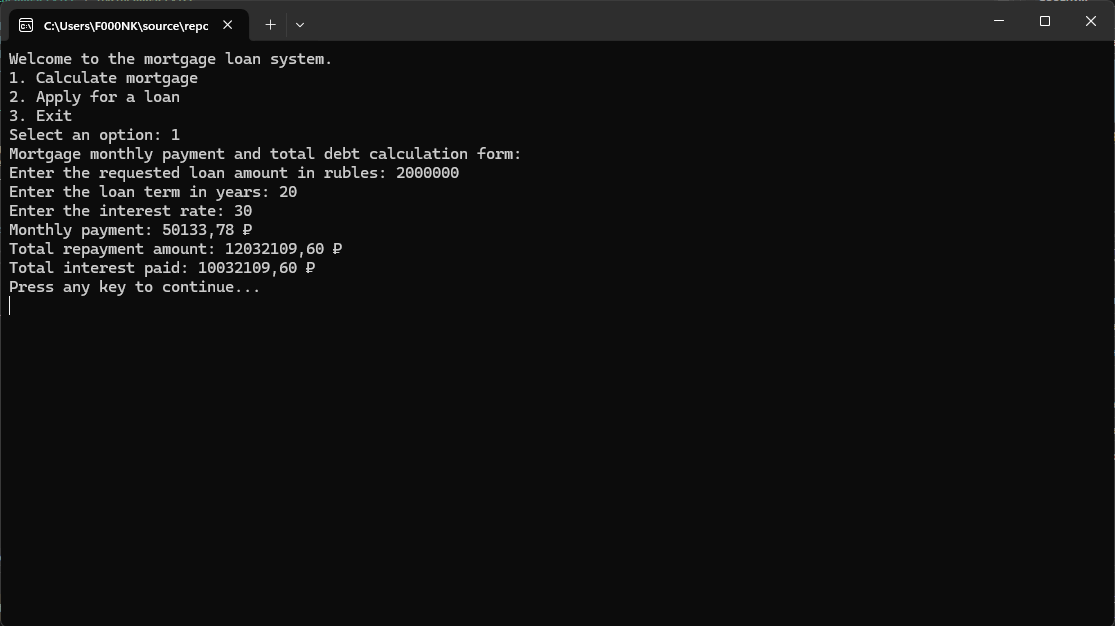


Рисунок 1.1 – Консольный интерфейс приложения

Приложение состоит из пяти проектов (рисунок 1.2):

* **RGR** – главный проект, отвечающий за инициализацию IoC-модулей и запуск основного цикла приложения.
* **RGR.Core** – модуль, содержащий бизнес-логику, такие как сервисы расчётов и работы с данными.
* **RGR.IO** – проект, отвечающий за ввод/вывод данных, включая взаимодействие с консолью и файлами, а также их тестирование.
* **RGR.Core.Tests** – модуль для тестирования бизнес-логики.
* **RGR.IO.Tests** – модуль для тестирования сервисов ввода/вывода.

Каждый из модулей разработан с учётом принципов модульности и переиспользуемости, что обеспечивает лёгкость тестирования и сопровождения.

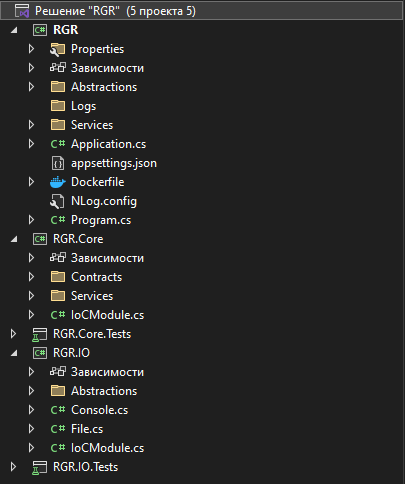


Рисунок 1.2 – Обозрение каталогов решения

1. Тестирование

Для тестирования был создан отдельные проекты **RGR.Core.Tests** и **RGR.IO.Tests** (рисунок 2.1), которые используют **NUnit3** и **.NET 9** для проверки основного проекта **RGR.Core и вспомогательного RGR.IO**, также написанных на **.NET 9**.

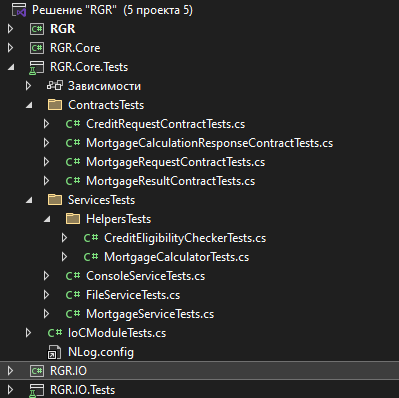


Рисунок 2.1 – Обозрение структуры проекта RGR.Core.Tests

Тестирование приложения проводится с использованием **NUnit3** и библиотеки **Moq** для создания имитаций зависимостей. Для каждого из ключевых модулей выполнены следующие проверки:

1. **Тестирование контрактов** (приложения A1, A2, A3, A4):

* Проверяется корректная работа конструкторов и соответствие создаваемых объектов ожидаемым данным.

1. **Тестирование сервисов расчёта ипотеки** (MortgageCalculator, приложение B1):

* Проверка корректности расчёта ежемесячного платежа, общей суммы выплат и переплаты.
* Тесты охватывают как позитивные сценарии (корректные данные), так и негативные (некорректные вводные).

1. **Тестирование проверки кредитоспособности** (CreditEligibilityChecker, приложение B2):

* Проверяется соответствие возрастным и финансовым критериям.
* Обеспечивается корректная обработка пограничных случаев, таких как минимально допустимый возраст или доход.

1. **Тестирование сервисов ввода/вывода** (ConsoleService и FileService, приложения C1, C2):

* Проверка обработки пользовательского ввода и исключительных ситуаций (например, пустой ввод или некорректный формат).
* Проверяется корректность сериализации и сохранения данных в файлы JSON.

1. **Тестирование основного сервиса** (MortgageService, приложение D):

* Проверяется обработка заявок на кредит, включая взаимодействие с сервисами расчёта и проверки кредитоспособности.
* Тестируются как успешные сценарии (одобрение кредита), так и отказ в кредите с выводом соответствующих сообщений.

1. **Тестирование IoC-модулей** (приложения E1):

* Проверяется корректность регистрации и взаимодействия зависимостей, таких как **MortgageService** и **ConsoleService**.
* Убедились, что все зарегистрированные сервисы имеют ожидаемый жизненный цикл (например, Singleton или Instance Per Dependency).

Тестированию RGR.IO не уделено большое внимание т.к. там тестируется только **IoCModule** (приложение E2).

Проведённое тестирование продемонстрировало высокий уровень стабильности приложения. Выявленные ошибки были оперативно исправлены, что улучшило общее качество системы.

ЗАКЛЮЧЕНИЕ

В результате работы было разработано консольное приложение, выполняющее расчёт ипотеки и проверку кредитных заявок. Приложение обладает модульной архитектурой, что позволило обеспечить высокое покрытие тестами и упростить процесс сопровождения. Проведённое тестирование подтвердило корректность работы всех ключевых модулей, включая расчёты, проверку вводных данных и взаимодействие сервисов.

Использование современных инструментов разработки и тестирования, таких как .NET 9, NUnit3 и внедрение зависимостей, обеспечило гибкость и масштабируемость приложения. Работа продемонстрировала значимость комплексного тестирования на всех этапах разработки для достижения высокого уровня качества программного обеспечения.

Приложение A1  
(обязательное)  
Листинг кода CreditRequestContractTests

using RGR.Core.Contracts;

namespace RGR.Core.Tests.ContractsTests

{

[TestFixture]

public class CreditRequestContractTests

{

// Позитивный тест для создания объекта CreditRequestContract с валидными значениями

[Test]

public void Constructor\_ValidInput\_CreatesCreditRequestContract()

{

// Arrange

var clientName = "John Doe";

var clientAge = 30;

var clientIncome = 50000m;

var loanAmount = 100000m;

var loanTerm = 15;

var interestRate = 5.5m;

var monthlyPayment = 8500m;

var totalRepayment = 153000m;

var totalInterest = 53000m;

// Act

var creditRequest = new CreditRequestContract(clientName, clientAge, clientIncome, loanAmount, loanTerm, interestRate, monthlyPayment, totalRepayment, totalInterest);

// Assert

Assert.That(creditRequest.ClientName, Is.EqualTo(clientName));

Assert.That(creditRequest.ClientAge, Is.EqualTo(clientAge));

Assert.That(creditRequest.ClientIncome, Is.EqualTo(clientIncome));

Assert.That(creditRequest.LoanAmount, Is.EqualTo(loanAmount));

Assert.That(creditRequest.LoanTerm, Is.EqualTo(loanTerm));

Assert.That(creditRequest.InterestRate, Is.EqualTo(interestRate));

Assert.That(creditRequest.MonthlyPayment, Is.EqualTo(monthlyPayment));

Assert.That(creditRequest.TotalRepayment, Is.EqualTo(totalRepayment));

Assert.That(creditRequest.TotalInterest, Is.EqualTo(totalInterest));

}

// Тест на создание объекта с нулевыми значениями

[Test]

public void Constructor\_ZeroValues\_CreatesCreditRequestContract()

{

// Arrange

var clientName = "John Doe";

var clientAge = 0;

var clientIncome = 0m;

var loanAmount = 0m;

var loanTerm = 0;

var interestRate = 0m;

var monthlyPayment = 0m;

var totalRepayment = 0m;

var totalInterest = 0m;

// Act

var creditRequest = new CreditRequestContract(clientName, clientAge, clientIncome, loanAmount, loanTerm, interestRate, monthlyPayment, totalRepayment, totalInterest);

// Assert

Assert.That(creditRequest.ClientName, Is.EqualTo(clientName));

Assert.That(creditRequest.ClientAge, Is.EqualTo(clientAge));

Assert.That(creditRequest.ClientIncome, Is.EqualTo(clientIncome));

Assert.That(creditRequest.LoanAmount, Is.EqualTo(loanAmount));

Assert.That(creditRequest.LoanTerm, Is.EqualTo(loanTerm));

Assert.That(creditRequest.InterestRate, Is.EqualTo(interestRate));

Assert.That(creditRequest.MonthlyPayment, Is.EqualTo(monthlyPayment));

Assert.That(creditRequest.TotalRepayment, Is.EqualTo(totalRepayment));

Assert.That(creditRequest.TotalInterest, Is.EqualTo(totalInterest));

}

// Тест на создание объекта с максимальными значениями

[Test]

public void Constructor\_MaxValues\_CreatesCreditRequestContract()

{

// Arrange

var clientName = "John Doe";

var clientAge = int.MaxValue;

var clientIncome = decimal.MaxValue;

var loanAmount = decimal.MaxValue;

var loanTerm = int.MaxValue;

var interestRate = decimal.MaxValue;

var monthlyPayment = decimal.MaxValue;

var totalRepayment = decimal.MaxValue;

var totalInterest = decimal.MaxValue;

// Act

var creditRequest = new CreditRequestContract(clientName, clientAge, clientIncome, loanAmount, loanTerm, interestRate, monthlyPayment, totalRepayment, totalInterest);

// Assert

Assert.That(creditRequest.ClientName, Is.EqualTo(clientName));

Assert.That(creditRequest.ClientAge, Is.EqualTo(clientAge));

Assert.That(creditRequest.ClientIncome, Is.EqualTo(clientIncome));

Assert.That(creditRequest.LoanAmount, Is.EqualTo(loanAmount));

Assert.That(creditRequest.LoanTerm, Is.EqualTo(loanTerm));

Assert.That(creditRequest.InterestRate, Is.EqualTo(interestRate));

Assert.That(creditRequest.MonthlyPayment, Is.EqualTo(monthlyPayment));

Assert.That(creditRequest.TotalRepayment, Is.EqualTo(totalRepayment));

Assert.That(creditRequest.TotalInterest, Is.EqualTo(totalInterest));

}

}

}

Приложение A2  
(обязательное)  
Листинг кода MortgageCalculationResponseContractTests

using NUnit.Framework;

using RGR.Core.Contracts;

namespace RGR.Core.Tests.ContractsTests

{

[TestFixture]

public class MortgageCalculationResponseContractTests

{

// Позитивный тест для конструктора

[Test]

public void Constructor\_ValidValues\_CreatesMortgageCalculationResponseContract()

{

// Arrange

var request = new MortgageRequestContract(100000m, 10, 5.5m); // Пример инициализации (зависит от конструктора MortgageRequestContract)

var result = new MortgageResultContract(1000m, 12000m, 50000m); // Пример инициализации (зависит от конструктора MortgageResultContract)

// Act

var response = new MortgageCalculationResponseContract(request, result);

// Assert

Assert.That(response.Request, Is.EqualTo(request));

Assert.That(response.Result, Is.EqualTo(result));

}

// Тест на проверку корректности значений в Request

[Test]

public void Constructor\_ValidRequest\_CreatesCorrectRequest()

{

// Arrange

var request = new MortgageRequestContract(100000m, 10, 5.5m);

var result = new MortgageResultContract(1000m, 12000m, 50000m);

// Act

var response = new MortgageCalculationResponseContract(request, result);

// Assert

Assert.That(response.Request.Amount, Is.EqualTo(100000m));

Assert.That(response.Request.Years, Is.EqualTo(10));

Assert.That(response.Request.InterestRate, Is.EqualTo(5.5m));

}

// Тест на проверку корректности значений в Result

[Test]

public void Constructor\_ValidResult\_CreatesCorrectResult()

{

// Arrange

var request = new MortgageRequestContract(100000m, 10, 5.5m);

var result = new MortgageResultContract(1000m, 12000m, 50000m);

// Act

var response = new MortgageCalculationResponseContract(request, result);

// Assert

Assert.That(response.Result.MonthlyPayment, Is.EqualTo(1000m));

Assert.That(response.Result.TotalRepayment, Is.EqualTo(12000m));

Assert.That(response.Result.TotalInterest, Is.EqualTo(50000m));

}

}

}

Приложение A3  
(обязательное)  
Листинг кода MortgageRequestContractTests

using NUnit.Framework;

using RGR.Core.Contracts;

using System;

namespace RGR.Core.Tests.ContractsTests

{

[TestFixture]

public class MortgageRequestContractTests

{

// Позитивный тест для конструктора с валидными значениями

[Test]

public void Constructor\_ValidValues\_CreatesMortgageRequestContract()

{

// Arrange

decimal amount = 500000m;

int years = 20;

decimal interestRate = 5.5m;

// Act

var request = new MortgageRequestContract(amount, years, interestRate);

// Assert

Assert.That(request.Amount, Is.EqualTo(amount));

Assert.That(request.Years, Is.EqualTo(years));

Assert.That(request.InterestRate, Is.EqualTo(interestRate));

}

// Тест для проверки корректности свойства Amount

[Test]

public void Constructor\_ValidAmount\_CreatesCorrectAmount()

{

// Arrange

decimal amount = 500000m;

int years = 20;

decimal interestRate = 5.5m;

// Act

var request = new MortgageRequestContract(amount, years, interestRate);

// Assert

Assert.That(request.Amount, Is.EqualTo(amount));

}

// Тест для проверки корректности свойства Years

[Test]

public void Constructor\_ValidYears\_CreatesCorrectYears()

{

// Arrange

decimal amount = 500000m;

int years = 20;

decimal interestRate = 5.5m;

// Act

var request = new MortgageRequestContract(amount, years, interestRate);

// Assert

Assert.That(request.Years, Is.EqualTo(years));

}

// Тест для проверки корректности свойства InterestRate

[Test]

public void Constructor\_ValidInterestRate\_CreatesCorrectInterestRate()

{

// Arrange

decimal amount = 500000m;

int years = 20;

decimal interestRate = 5.5m;

// Act

var request = new MortgageRequestContract(amount, years, interestRate);

// Assert

Assert.That(request.InterestRate, Is.EqualTo(interestRate));

}

}

}

Приложение A4  
(обязательное)  
Листинг кода MortgageRequestContractTests

using RGR.Core.Contracts;

namespace RGR.Core.Tests.ContractsTests

{

[TestFixture]

public class MortgageResultContractTests

{

// Позитивный тест для конструктора

[Test]

public void Constructor\_ValidValues\_CreatesMortgageResultContract()

{

// Arrange

decimal monthlyPayment = 1000m;

decimal totalRepayment = 12000m;

decimal totalInterest = 5000m;

// Act

var result = new MortgageResultContract(monthlyPayment, totalRepayment, totalInterest);

// Assert

Assert.That(result.MonthlyPayment, Is.EqualTo(monthlyPayment));

Assert.That(result.TotalRepayment, Is.EqualTo(totalRepayment));

Assert.That(result.TotalInterest, Is.EqualTo(totalInterest));

}

// Тест на проверку значения MonthlyPayment

[Test]

public void Constructor\_ValidMonthlyPayment\_CreatesCorrectMonthlyPayment()

{

// Arrange

decimal monthlyPayment = 1000m;

decimal totalRepayment = 12000m;

decimal totalInterest = 5000m;

// Act

var result = new MortgageResultContract(monthlyPayment, totalRepayment, totalInterest);

// Assert

Assert.That(result.MonthlyPayment, Is.EqualTo(1000m));

}

// Тест на проверку значения TotalRepayment

[Test]

public void Constructor\_ValidTotalRepayment\_CreatesCorrectTotalRepayment()

{

// Arrange

decimal monthlyPayment = 1000m;

decimal totalRepayment = 12000m;

decimal totalInterest = 5000m;

// Act

var result = new MortgageResultContract(monthlyPayment, totalRepayment, totalInterest);

// Assert

Assert.That(result.TotalRepayment, Is.EqualTo(12000m));

}

// Тест на проверку значения TotalInterest

[Test]

public void Constructor\_ValidTotalInterest\_CreatesCorrectTotalInterest()

{

// Arrange

decimal monthlyPayment = 1000m;

decimal totalRepayment = 12000m;

decimal totalInterest = 5000m;

// Act

var result = new MortgageResultContract(monthlyPayment, totalRepayment, totalInterest);

// Assert

Assert.That(result.TotalInterest, Is.EqualTo(5000m));

}

}

}

Приложение B1  
(обязательное)  
Листинг кода MortgageCalculatorTests

using Moq;

using Microsoft.Extensions.Logging;

using RGR.Core.Services.Helpers;

using RGR.Core.Contracts;

namespace RGR.Core.Tests.ServicesTests.HelpersTests

{

public class MortgageCalculatorTests

{

private Mock<ILogger<MortgageCalculator>> \_loggerMock;

private MortgageCalculator \_calculator;

[SetUp]

public void Setup()

{

\_loggerMock = new Mock<ILogger<MortgageCalculator>>();

\_calculator = new MortgageCalculator(\_loggerMock.Object);

}

[Test]

public async Task CalculateMortgage\_ValidInput\_ReturnsCorrectResult()

{

// Arrange

var request = new MortgageRequestContract

(

amount: 200000,

years: 30,

interestRate: 5.0m

);

// Act

var result = await \_calculator.CalculateMortgage(request);

// Assert

Assert.That(result, Is.Not.Null);

Assert.That(result.Request, Is.EqualTo(request));

Assert.That(result.Result.MonthlyPayment, Is.GreaterThan(0));

Assert.That(result.Result.TotalRepayment, Is.GreaterThan(request.Amount));

Assert.That(result.Result.TotalInterest, Is.EqualTo(result.Result.TotalRepayment - request.Amount));

\_loggerMock.Verify(

x => x.Log(

LogLevel.Information,

It.IsAny<EventId>(),

It.Is<It.IsAnyType>((v, t) => v.ToString().Contains("Starting mortgage calculation")),

null,

It.IsAny<Func<It.IsAnyType, Exception, string>>()

),

Times.Once);

\_loggerMock.Verify(

x => x.Log(

LogLevel.Information,

It.IsAny<EventId>(),

It.Is<It.IsAnyType>((v, t) => v.ToString().Contains("Mortgage calculation completed")),

null,

It.IsAny<Func<It.IsAnyType, Exception, string>>()

),

Times.Once);

}

[Test]

public void CalculateMortgage\_NullRequest\_ThrowsArgumentException()

{

// Arrange

MortgageRequestContract request = null;

// Act & Assert

var ex = Assert.ThrowsAsync<ArgumentNullException>(async () => await \_calculator.CalculateMortgage(request));

Assert.That(ex.Message, Is.EqualTo("Request contract cannot be null (Parameter 'requestContract')"));

\_loggerMock.Verify(

x => x.Log(

LogLevel.Error,

It.IsAny<EventId>(),

It.Is<It.IsAnyType>((v, t) => v.ToString().Contains("Invalid request contract")),

null,

It.IsAny<Func<It.IsAnyType, Exception, string>>()

),

Times.Once);

}

[Test]

public void CalculateMortgage\_InvalidAmount\_ThrowsArgumentException()

{

// Arrange

var request = new MortgageRequestContract

(

amount: 0,

years: 30,

interestRate: 5.0m

);

// Act & Assert

var ex = Assert.ThrowsAsync<ArgumentException>(async () => await \_calculator.CalculateMortgage(request));

Assert.That(ex.Message, Is.EqualTo("Loan amount must be greater than zero. (Parameter 'Amount')"));

\_loggerMock.Verify(

x => x.Log(

LogLevel.Error,

It.IsAny<EventId>(),

It.Is<It.IsAnyType>((v, t) => v.ToString().Contains("Invalid loan amount")),

null,

It.IsAny<Func<It.IsAnyType, Exception, string>>()

),

Times.Once);

}

[Test]

public void CalculateMortgage\_InvalidYears\_ThrowsArgumentException()

{

// Arrange

var request = new MortgageRequestContract

(

amount: 200000,

years: 0,

interestRate: 5.0m

);

// Act & Assert

var ex = Assert.ThrowsAsync<ArgumentException>(async () => await \_calculator.CalculateMortgage(request));

Assert.That(ex.Message, Is.EqualTo("Loan term in years must be greater than zero. (Parameter 'Years')"));

\_loggerMock.Verify(

x => x.Log(

LogLevel.Error,

It.IsAny<EventId>(),

It.Is<It.IsAnyType>((v, t) => v.ToString().Contains("Invalid loan term in years")),

null,

It.IsAny<Func<It.IsAnyType, Exception, string>>()

),

Times.Once);

}

[Test]

public void CalculateMortgage\_InvalidInterestRate\_ThrowsArgumentException()

{

// Arrange

var request = new MortgageRequestContract

(

amount: 200000,

years: 30,

interestRate: 0m

);

// Act & Assert

var ex = Assert.ThrowsAsync<ArgumentException>(async () => await \_calculator.CalculateMortgage(request));

Assert.That(ex.Message, Is.EqualTo("Interest rate must be greater than zero. (Parameter 'InterestRate')"));

\_loggerMock.Verify(

x => x.Log(

LogLevel.Error,

It.IsAny<EventId>(),

It.Is<It.IsAnyType>((v, t) => v.ToString().Contains("Invalid interest rate")),

null,

It.IsAny<Func<It.IsAnyType, Exception, string>>()

),

Times.Once);

}

}

}

Приложение B2  
(обязательное)  
Листинг кода CreditEligibilityCheckerTests

using Moq;

using Microsoft.Extensions.Logging;

using RGR.Core.Services.Helpers;

namespace RGR.Core.Tests.ServicesTests.HelpersTests

{

public class CreditEligibilityCheckerTests

{

private Mock<ILogger<CreditEligibilityChecker>> \_loggerMock;

private CreditEligibilityChecker \_checker;

[SetUp]

public void Setup()

{

\_loggerMock = new Mock<ILogger<CreditEligibilityChecker>>();

\_checker = new CreditEligibilityChecker(\_loggerMock.Object);

}

[Test]

public void CheckEligibility\_Approved\_ReturnsTrue()

{

// Arrange

int clientAge = 25;

decimal clientIncome = 50000;

decimal loanAmount = 100000;

int years = 10;

// Act

bool result = \_checker.CheckEligibility(clientAge, clientIncome, loanAmount, years);

// Assert

Assert.That(result, Is.True);

\_loggerMock.Verify(

x => x.Log(

LogLevel.Information,

It.IsAny<EventId>(),

It.Is<It.IsAnyType>((v, t) => v.ToString().Contains("approved")),

null,

It.IsAny<Func<It.IsAnyType, Exception, string>>()

),

Times.Once);

}

[Test]

public void CheckEligibility\_RejectedByAge\_ReturnsFalse()

{

// Arrange

int clientAge = 20;

decimal clientIncome = 50000;

decimal loanAmount = 100000;

int years = 10;

// Act

bool result = \_checker.CheckEligibility(clientAge, clientIncome, loanAmount, years);

// Assert

Assert.That(result, Is.False);

\_loggerMock.Verify(

x => x.Log(

LogLevel.Information,

It.IsAny<EventId>(),

It.Is<It.IsAnyType>((v, t) => v.ToString().Contains("rejected")),

null,

It.IsAny<Func<It.IsAnyType, Exception, string>>()

),

Times.Once);

}

[Test]

public void CheckEligibility\_RejectedByIncome\_ReturnsFalse()

{

// Arrange

int clientAge = 25;

decimal clientIncome = 1000;

decimal loanAmount = 100000;

int years = 10;

// Act

bool result = \_checker.CheckEligibility(clientAge, clientIncome, loanAmount, years);

// Assert

Assert.That(result, Is.False);

\_loggerMock.Verify(

x => x.Log(

LogLevel.Information,

It.IsAny<EventId>(),

It.Is<It.IsAnyType>((v, t) => v.ToString().Contains("rejected")),

null,

It.IsAny<Func<It.IsAnyType, Exception, string>>()

),

Times.Once);

}

}

}

Приложение C1  
(обязательное)  
Листинг кода ConsoleServiceTests

using Moq;

using NUnit.Framework;

using RGR.Core.Services;

using RGR.IO.Abstractions;

namespace RGR.Core.Tests.ServicesTests

{

public class ConsoleServiceTests

{

private Mock<IConsole> \_mockConsole;

private ConsoleService \_consoleService;

[SetUp]

public void Setup()

{

\_mockConsole = new Mock<IConsole>();

\_consoleService = new ConsoleService(\_mockConsole.Object);

}

// Позитивный тест для GetClientFullName

[Test]

public void GetClientFullName\_ValidInput\_ReturnsFullName()

{

// Arrange

\_mockConsole.Setup(c => c.ReadLine()).Returns("John Doe");

// Act

var result = \_consoleService.GetClientFullName();

// Assert

Assert.That(result, Is.EqualTo("John Doe"));

\_mockConsole.Verify(c => c.Write(It.IsAny<string>()), Times.Once);

}

// Негативный тест для GetClientFullName (пустое имя)

[Test]

public void GetClientFullName\_EmptyInput\_PromptsForValidInput()

{

// Arrange

\_mockConsole.SetupSequence(c => c.ReadLine())

.Returns("") // Пустой ввод на первом шаге

.Returns("John Doe"); // Вводим правильное имя на втором шаге

// Act

var result = \_consoleService.GetClientFullName();

// Assert

Assert.That(result, Is.EqualTo(string.Empty));

\_mockConsole.Verify(c => c.WriteLine("Client name cannot be empty. Please enter a valid name."), Times.Exactly(1));

}

// Тест на GetClientAge с правильным вводом

[Test]

public void GetClientAge\_ValidAge\_ReturnsAge()

{

// Arrange

\_mockConsole.Setup(c => c.ReadLine()).Returns("25");

// Act

var result = \_consoleService.GetClientAge();

// Assert

Assert.That(result, Is.EqualTo(25));

}

// Негативный тест на GetClientAge с неправильным вводом

[Test]

public void GetClientAge\_InvalidInput\_RetriesUntilValidInput()

{

// Arrange

\_mockConsole.SetupSequence(c => c.ReadLine())

.Returns("abc")

.Returns("25");

// Act

var result = \_consoleService.GetClientAge();

// Assert

Assert.That(result, Is.EqualTo(25));

\_mockConsole.Verify(c => c.WriteLine("Incorrect format. Please enter a valid age (over 21)."), Times.Once);

}

// Тест на GetClientIncome с правильным вводом

[Test]

public void GetClientIncome\_ValidIncome\_ReturnsIncome()

{

// Arrange

\_mockConsole.Setup(c => c.ReadLine()).Returns("50000");

// Act

var result = \_consoleService.GetClientIncome();

// Assert

Assert.That(result, Is.EqualTo(50000));

}

// Тест на GetLoanAmount с правильным вводом

[Test]

public void GetLoanAmount\_ValidLoanAmount\_ReturnsLoanAmount()

{

// Arrange

\_mockConsole.Setup(c => c.ReadLine()).Returns("100000");

// Act

var result = \_consoleService.GetLoanAmount();

// Assert

Assert.That(result, Is.EqualTo(100000));

}

// Проверка исключений для неверного ввода на ReadDecimal

[Test]

public void ReadDecimal\_InvalidInput\_ThrowsException()

{

// Arrange

\_mockConsole.SetupSequence(c => c.ReadLine())

.Returns("abc")

.Returns("1000");

// Act

var result = \_consoleService.ReadDecimal("Enter amount: ", "Invalid input!");

// Assert

Assert.That(result, Is.EqualTo(1000));

\_mockConsole.Verify(c => c.WriteLine("Invalid input!"), Times.Once);

}

// Тест на GetMortgageRequest с использованием параметров

[Test]

public void GetMortgageRequest\_ValidRequest\_ReturnsRequest()

{

// Arrange

\_mockConsole.Setup(c => c.ReadLine()).Returns("5"); // Для interest rate

// Act

var request = \_consoleService.GetMortgageRequest(100000, 10);

// Assert

Assert.That(request.Amount, Is.EqualTo(100000));

Assert.That(request.Years, Is.EqualTo(10));

Assert.That(request.InterestRate, Is.EqualTo(5m));

}

// Проверка исключения для неверного ввода (позитивный случай)

[Test]

public void GetYears\_InvalidInput\_PromptsUntilValidInput()

{

// Arrange

\_mockConsole.SetupSequence(c => c.ReadLine())

.Returns("abc")

.Returns("10");

// Act

var result = \_consoleService.GetYears();

// Assert

Assert.That(result, Is.EqualTo(10));

\_mockConsole.Verify(c => c.WriteLine("Incorrect format. Please enter a valid interest years"), Times.Once);

}

}

}

Приложение C2  
(обязательное)  
Листинг кода FileServiceTests

using Moq;

using RGR.Core.Contracts;

using RGR.Core.Services;

using RGR.IO.Abstractions;

using System.Text.Json;

namespace RGR.Core.Tests.ServicesTests

{

[TestFixture]

public class FileServiceTests

{

private Mock<IFile> \_mockFile;

private FileService \_fileService;

[SetUp]

public void Setup()

{

\_mockFile = new Mock<IFile>();

\_fileService = new FileService(\_mockFile.Object);

}

// Позитивный тест на успешное сохранение запроса

[Test]

public void SaveCreditRequest\_ValidRequest\_SavesToFile()

{

// Arrange

var creditRequest = new CreditRequestContract(

"John Doe",

30,

50000m,

100000m,

15,

5.5m,

8500m,

153000m,

53000m

);

var filePath = "creditRequest.json";

// Act

\_fileService.SaveCreditRequest(creditRequest, filePath);

// Assert

var expectedJson = JsonSerializer.Serialize(creditRequest, new JsonSerializerOptions { WriteIndented = true });

\_mockFile.Verify(f => f.WriteAllText(filePath, expectedJson), Times.Once);

}

// Негативный тест на SaveCreditRequest при ошибке записи в файл

[Test]

public void SaveCreditRequest\_WriteToFileThrowsException\_ThrowsIOException()

{

// Arrange

var creditRequest = new CreditRequestContract(

"John Doe",

30,

50000m,

100000m,

15,

5.5m,

8500m,

153000m,

53000m

);

var filePath = "creditRequest.json";

// Настроим мок так, чтобы при вызове WriteAllText возникало исключение

\_mockFile.Setup(f => f.WriteAllText(filePath, It.IsAny<string>())).Throws(new IOException("Failed to write to file"));

// Act & Assert

var ex = Assert.Throws<IOException>(() => \_fileService.SaveCreditRequest(creditRequest, filePath));

Assert.That(ex.Message, Is.EqualTo("Failed to write to file"));

}

// Тест на проверку сериализации объекта

[Test]

public void SaveCreditRequest\_SerializesObjectCorrectly()

{

// Arrange

var creditRequest = new CreditRequestContract(

"John Doe",

30,

50000m,

100000m,

15,

5.5m,

8500m,

153000m,

53000m

);

var filePath = "creditRequest.json";

// Act

\_fileService.SaveCreditRequest(creditRequest, filePath);

// Assert

var expectedJson = JsonSerializer.Serialize(creditRequest, new JsonSerializerOptions { WriteIndented = true });

\_mockFile.Verify(f => f.WriteAllText(filePath, expectedJson), Times.Once);

}

// Тест на неправильный путь файла (например, пустой путь)

[Test]

public void SaveCreditRequest\_EmptyFilePath\_ThrowsArgumentException()

{

// Arrange

var creditRequest = new CreditRequestContract(

"John Doe",

30,

50000m,

100000m,

15,

5.5m,

8500m,

153000m,

53000m

);

var filePath = string.Empty;

// Act & Assert

var ex = Assert.Throws<ArgumentNullException>(() => \_fileService.SaveCreditRequest(creditRequest, filePath));

Assert.That(ex.Message, Is.EqualTo("Value cannot be null. (Parameter 'filePath')"));

}

// Тест на сериализацию пустого объекта

[Test]

public void SaveCreditRequest\_EmptyCreditRequest\_SavesToFile()

{

// Arrange

var creditRequest = new CreditRequestContract(

"",

0,

0m,

0m,

0,

0m,

0m,

0m,

0m

);

var filePath = "emptyCreditRequest.json";

// Act

\_fileService.SaveCreditRequest(creditRequest, filePath);

// Assert

var expectedJson = JsonSerializer.Serialize(creditRequest, new JsonSerializerOptions { WriteIndented = true });

\_mockFile.Verify(f => f.WriteAllText(filePath, expectedJson), Times.Once);

}

}

}

Приложение D  
(обязательное)  
Листинг кода MortgageServiceTests

using Moq;

using RGR.Core.Contracts;

using RGR.Core.Services;

using RGR.Core.Services.Abstractions;

using RGR.Core.Services.Helpers.Abstractions;

using RGR.IO.Abstractions;

using Microsoft.Extensions.Logging;

namespace RGR.Core.Tests.ServicesTests

{

[TestFixture]

public class MortgageServiceTests

{

private Mock<ILogger<MortgageService>> \_mockLogger;

private Mock<ICreditEligibilityChecker> \_mockEligibilityChecker;

private Mock<IMortgageCalculator> \_mockMortgageCalculator;

private Mock<IConsole> \_mockConsole;

private Mock<IFileService> \_mockFileService;

private Mock<IConsoleService> \_mockConsoleService;

private MortgageService \_mortgageService;

[SetUp]

public void SetUp()

{

\_mockLogger = new Mock<ILogger<MortgageService>>();

\_mockEligibilityChecker = new Mock<ICreditEligibilityChecker>();

\_mockMortgageCalculator = new Mock<IMortgageCalculator>();

\_mockConsole = new Mock<IConsole>();

\_mockFileService = new Mock<IFileService>();

\_mockConsoleService = new Mock<IConsoleService>();

\_mortgageService = new MortgageService(

\_mockLogger.Object,

\_mockEligibilityChecker.Object,

\_mockMortgageCalculator.Object,

\_mockConsole.Object,

\_mockFileService.Object,

\_mockConsoleService.Object

);

}

// Тест для HandleCreditRequest (позитивный случай)

[Test]

public async Task HandleCreditRequest\_Approved\_SavesCreditRequestAndDisplaysResults()

{

// Arrange

var mockMortgageResult = new MortgageCalculationResponseContract(

new MortgageRequestContract(1000000m, 20, 5.5m),

new MortgageResultContract(10000m, 1200000m, 200000m)

);

\_mockConsoleService.Setup(c => c.GetClientFullName()).Returns("John Doe");

\_mockConsoleService.Setup(c => c.GetClientAge()).Returns(30);

\_mockConsoleService.Setup(c => c.GetClientIncome()).Returns(50000m);

\_mockConsoleService.Setup(c => c.GetLoanAmount()).Returns(1000000m);

\_mockConsoleService.Setup(c => c.GetYears()).Returns(20);

\_mockEligibilityChecker.Setup(e => e.CheckEligibility(30, 50000m, 1000000m, 20)).Returns(true);

\_mockConsoleService.Setup(c => c.GetMortgageRequest(1000000m, 20)).Returns(new MortgageRequestContract(1000000m, 20, 5.5m));

\_mockMortgageCalculator.Setup(m => m.CalculateMortgage(It.IsAny<MortgageRequestContract>())).ReturnsAsync(mockMortgageResult);

\_mockFileService.Setup(f => f.SaveCreditRequest(It.IsAny<CreditRequestContract>(), It.IsAny<string>()));

// Act

await \_mortgageService.HandleCreditRequest();

// Assert

\_mockConsole.Verify(c => c.WriteLine(It.IsAny<string>()), Times.AtLeastOnce);

\_mockFileService.Verify(f => f.SaveCreditRequest(It.IsAny<CreditRequestContract>(), It.IsAny<string>()), Times.Once);

}

// Тест для HandleCreditRequest (отказ в кредите)

[Test]

public async Task HandleCreditRequest\_Rejected\_DisplaysRejectionMessage()

{

// Arrange

\_mockConsoleService.Setup(c => c.GetClientFullName()).Returns("John Doe");

\_mockConsoleService.Setup(c => c.GetClientAge()).Returns(30);

\_mockConsoleService.Setup(c => c.GetClientIncome()).Returns(50000m);

\_mockConsoleService.Setup(c => c.GetLoanAmount()).Returns(1000000m);

\_mockConsoleService.Setup(c => c.GetYears()).Returns(20);

\_mockEligibilityChecker.Setup(e => e.CheckEligibility(30, 50000m, 1000000m, 20)).Returns(false);

// Act

await \_mortgageService.HandleCreditRequest();

// Assert

\_mockConsole.Verify(c => c.WriteLine(It.Is<string>(s => s.Contains("Credit request rejected"))), Times.Once);

\_mockFileService.Verify(f => f.SaveCreditRequest(It.IsAny<CreditRequestContract>(), It.IsAny<string>()), Times.Never);

}

// Тест для HandleMortgageCalculation (успешный расчет)

[Test]

public async Task HandleMortgageCalculation\_Success\_DisplaysResults()

{

// Arrange

var mockMortgageResult = new MortgageCalculationResponseContract(

new MortgageRequestContract(1000000m, 20, 5.5m),

new MortgageResultContract(10000m, 1200000m, 200000m)

);

\_mockConsoleService.Setup(c => c.GetMortgageRequest()).Returns(new MortgageRequestContract(1000000m, 20, 5.5m));

\_mockMortgageCalculator.Setup(m => m.CalculateMortgage(It.IsAny<MortgageRequestContract>())).ReturnsAsync(mockMortgageResult);

// Act

await \_mortgageService.HandleMortgageCalculation();

// Assert

\_mockConsole.Verify(c => c.WriteLine(It.IsAny<string>()), Times.AtLeastOnce);

}

// Тест для HandleMortgageCalculation (ошибка при расчете)

[Test]

public async Task HandleMortgageCalculation\_Failure\_DisplaysErrorMessage()

{

// Arrange

\_mockConsoleService.Setup(c => c.GetMortgageRequest()).Returns(new MortgageRequestContract(1000000m, 20, 5.5m));

\_mockMortgageCalculator.Setup(m => m.CalculateMortgage(It.IsAny<MortgageRequestContract>())).ThrowsAsync(new Exception("Calculation error"));

// Act

await \_mortgageService.HandleMortgageCalculation();

// Assert

\_mockConsole.Verify(c => c.WriteLine(It.Is<string>(s => s.Contains("An error occurred"))), Times.Once);

}

}

}

Приложение E1  
(обязательное)  
Листинг кода RGR.Core.Tests.IoCModuleTests

using Autofac;

using Microsoft.Extensions.Configuration;

using RGR.Core.Services;

using RGR.Core.Services.Abstractions;

using RGR.Core.Services.Helpers.Abstractions;

using Moq;

using RGR.Core.Services.Helpers;

using Microsoft.Extensions.Logging;

using RGR.IO.Abstractions;

using NLog.Extensions.Logging;

namespace RGR.Core.Tests

{

[TestFixture]

public class IoCModuleTests

{

private IContainer \_container;

private Mock<IConfiguration> \_mockConfiguration;

[SetUp]

public void SetUp()

{

\_mockConfiguration = new Mock<IConfiguration>();

var builder = new ContainerBuilder();

// Регистрация ILoggerFactory и ILogger

builder.RegisterInstance(LoggerFactory.Create(loggingBuilder =>

{

loggingBuilder.AddNLog();

})).As<ILoggerFactory>().SingleInstance();

builder.RegisterGeneric(typeof(Logger<>)).As(typeof(ILogger<>)).SingleInstance();

// Регистрация конфигурации

var configuration = new ConfigurationBuilder()

.SetBasePath(Directory.GetCurrentDirectory())

.AddJsonFile("appsettings.json", optional: true, reloadOnChange: true)

.Build();

// Регистрация модуля IoC

builder.RegisterModule(new RGR.IO.IoCModule(\_mockConfiguration.Object));

builder.RegisterModule(new RGR.Core.IoCModule(\_mockConfiguration.Object));

// Регистрация IConsole для ConsoleService

builder.RegisterType<RGR.IO.Console>().As<IConsole>().SingleInstance();

\_container = builder.Build();

}

[TearDown]

public void TearDown()

{

// Dispose the container to release resources

\_container.Dispose();

}

[Test]

public void TestMortgageCalculatorRegistration()

{

// Act

var mortgageCalculator = \_container.Resolve<IMortgageCalculator>();

// Assert

Assert.That(mortgageCalculator, Is.Not.Null);

Assert.That(mortgageCalculator, Is.InstanceOf<MortgageCalculator>());

}

[Test]

public void TestCreditEligibilityCheckerRegistration()

{

// Act

var eligibilityChecker = \_container.Resolve<ICreditEligibilityChecker>();

// Assert

Assert.That(eligibilityChecker, Is.Not.Null);

Assert.That(eligibilityChecker, Is.InstanceOf<CreditEligibilityChecker>());

}

[Test]

public void TestFileServiceRegistration()

{

// Act

var fileService = \_container.Resolve<IFileService>();

// Assert

Assert.That(fileService, Is.Not.Null);

Assert.That(fileService, Is.InstanceOf<FileService>());

}

[Test]

public void TestConsoleServiceRegistration()

{

// Act

var consoleService = \_container.Resolve<IConsoleService>();

// Assert

Assert.That(consoleService, Is.Not.Null);

Assert.That(consoleService, Is.InstanceOf<ConsoleService>());

}

[Test]

public void TestMortgageServiceRegistration()

{

// Act

var mortgageService = \_container.Resolve<IMortgageService>();

// Assert

Assert.That(mortgageService, Is.Not.Null);

Assert.That(mortgageService, Is.InstanceOf<MortgageService>());

}

[Test]

public void TestLifetimeForSingletons()

{

// Act

var mortgageCalculator1 = \_container.Resolve<IMortgageCalculator>();

var mortgageCalculator2 = \_container.Resolve<IMortgageCalculator>();

// Assert

Assert.That(mortgageCalculator1, Is.SameAs(mortgageCalculator2)); // Ensure they are the same instance

}

[Test]

public void TestLifetimeForInstancePerDependency()

{

// Act

var mortgageService1 = \_container.Resolve<IMortgageService>();

var mortgageService2 = \_container.Resolve<IMortgageService>();

// Assert

Assert.That(mortgageService1, Is.Not.SameAs(mortgageService2)); // Ensure they are different instances

}

}

}

Приложение E2  
(обязательное)  
Листинг кода RGR.IO.Tests.IoCModuleTests

using Autofac;

using Microsoft.Extensions.Configuration;

using Moq;

using RGR.IO.Abstractions;

namespace RGR.IO.Tests

{

[TestFixture]

public class IoCModuleTests

{

private IContainer \_container;

private Mock<IConfiguration> \_mockConfiguration;

[SetUp]

public void SetUp()

{

\_mockConfiguration = new Mock<IConfiguration>();

var builder = new ContainerBuilder();

// Регистрация модуля IoC

builder.RegisterModule(new IoCModule(\_mockConfiguration.Object));

// Строим контейнер

\_container = builder.Build();

}

[TearDown]

public void TearDown()

{

// Освобождаем ресурсы

\_container.Dispose();

}

[Test]

public void TestFileServiceRegistration()

{

// Act

var fileService = \_container.Resolve<IFile>();

// Assert

Assert.That(fileService, Is.Not.Null);

Assert.That(fileService, Is.InstanceOf<IO.File>());

}

[Test]

public void TestConsoleServiceRegistration()

{

// Act

var consoleService = \_container.Resolve<IConsole>();

// Assert

Assert.That(consoleService, Is.Not.Null);

Assert.That(consoleService, Is.InstanceOf<IO.Console>());

}

[Test]

public void TestLifetimeForFileService()

{

// Act

var fileService1 = \_container.Resolve<IFile>();

var fileService2 = \_container.Resolve<IFile>();

// Assert

Assert.That(fileService1, Is.Not.SameAs(fileService2)); // Проверяем, что это разные экземпляры

}

[Test]

public void TestLifetimeForConsoleService()

{

// Act

var consoleService1 = \_container.Resolve<IConsole>();

var consoleService2 = \_container.Resolve<IConsole>();

// Assert

Assert.That(consoleService1, Is.Not.SameAs(consoleService2)); // Проверяем, что это разные экземпляры

}

}

}