ГУАП

КАФЕДРА № 43

ОТЧЕТ ЗАЩИЩЕН С ОЦЕНКОЙ: _____

ПРЕПОДАВАТЕЛЬ:

доцент, к.т.н., доцент	/		/		/	В. Н. Коромысличенко
(должность, учёная степень, звание)		(подпись)		(дата защиты)		(инициалы, фамилия)

ОТЧЕТ О ЛАБОРАТОРНОЙ РАБОТЕ №2

«Разработать требования на программу деления чисел»

ПО КУРСУ: «Разработка и анализ требований»

РАБОТУ ВЫПОЛНИЛ СТУДЕНТ: 4134К (номер группы) / Самарин Д.В. (инициалы, фамилия)

/ _____ (подпись студента) / _____ (дата отчета)

Санкт-Петербург 2025

Разработка требований для программы деления чисел

1. Введение

Цель данного доклада – разработка требований к программе, выполняющей операцию деления чисел. Программа должна корректно принимать входные данные, выполнять операцию деления, обрабатывать исключения (например, деление на ноль или некорректный ввод) и выводить результат. Доклад составлен с учетом современных методик инженерии требований, использования сценарного подхода, трассировки и спецификации требований

Особое внимание уделено:

1. Функциональным требованиям: обеспечивают основные операции (ввод, проверка, вычисление, вывод).
2. Нефункциональным требованиям: определяют параметры качества, такие как производительность, надежность, удобство и безопасность.
3. Сценариям использования: последовательность действий пользователя и системы.
4. Архитектурным решениям: модульное разделение системы и методы трассировки требований.

2. Анализ предметной области и постановка задачи

Предметная область:

Программа деления чисел является утилитой для выполнения одной из базовых математических операций. Задача состоит в том, чтобы корректно принимать два числа – делимое и делитель, выполнять деление, обрабатывать ошибки (например, деление на ноль, некорректный формат ввода) и представлять результат пользователю.

Основные задачи системы:

1. Прием числовых данных.
2. Проверка корректности ввода.
3. Выполнение операции деления.
4. Вывод результата и сообщение об ошибках.

Эта постановка задачи соответствует требованиям, изложенным в документе по инженерии требований, где выделяются процессы спецификации, валидации и трассировки требований.

3. Функциональные требования

Функциональные требования описывают функции, которые должна выполнять система:

3.1. Ввод и проверка данных

Прием данных:

- Программа должна принимать два числовых значения (делимое и делитель) через графический интерфейс или консоль.

Проверка корректности:

- Вводимые данные проверяются на соответствие числовому формату; в случае некорректного ввода (например, символы вместо цифр) генерируется сообщение об ошибке.

3.2. Вычисление операции деления

Выполнение деления:

Программа выполняет операцию деления с поддержкой как целочисленного, так и дробного деления.

Обработка исключений:

При попытке деления на ноль система должна выводить предупреждение и предлагать повторный ввод.

3.3. Вывод результатов

Интерфейс вывода:

Результат вычисления выводится пользователю в отдельном поле (или в консольном окне).

Логирование ошибок:

При возникновении ошибок данные фиксируются для последующего анализа.

3.4. Дополнительные возможности (опционально)

- Сохранение результатов в файл.
- Отображение истории выполненных операций.

4. Нефункциональные требования

Нефункциональные требования определяют характеристики системы, не связанные с основной функциональностью:

4.1. Производительность
Операция деления должна выполняться практически мгновенно (задержка не более 0,1 секунды).

Система должна корректно работать на различных аппаратных конфигурациях.

4.2. Надежность и устойчивость
Корректная обработка ошибок ввода и математических исключений.

Система не должна аварийно завершать работу при возникновении ошибок.

4.3. Удобство использования (Usability)
Интуитивно понятный интерфейс с четкими инструкциями и пояснениями.

Сообщения об ошибках должны быть ясными и информативными.

4.4. Безопасность
Защита от внедрения некорректных данных.

В случае расширенного функционала (например, сохранение в файл) – контроль доступа к файлам.

4.5. Поддерживаемость и масштабируемость
Модульная архитектура, позволяющая в будущем расширять функционал (например, добавление новых математических операций).

Четкая документация и комментарии в коде для упрощения поддержки.

4.6. Портируемость
Поддержка работы на разных операционных системах (Windows, Linux, macOS) при использовании кроссплатформенных технологий.

5. Требования к пользовательскому интерфейсу
Для обеспечения удобного взаимодействия с пользователем необходимо детально описать требования к интерфейсу:

5.1. Главное окно приложения
- Два текстовых поля для ввода делимого и делителя.
- Ярлыки (label) для каждого поля с пояснениями.
- Кнопка «Вычислить», запускающая операцию деления.

5.2. Окно вывода результата
- Отдельное поле или диалоговое окно для отображения результата.
- При возникновении ошибки – вывод сообщения с рекомендацией повторить ввод.

5.3. Элементы управления

- Расположение элементов должно обеспечивать логичный порядок действий.
- Интерфейс – адаптивный к разным разрешениям экрана.

Пример:

Пользователь вводит два числа, программа возвращает результат или сообщение об ошибке.

7. Системные требования (System Requirements)

Внешние условия:

- Поддержка целочисленного и дробного деления.

Архитектурные ограничения:

- Модульная структура (ввод/вывод, проверка данных, вычисления).

8. Требования к атрибутам качества (Quality Attributes)

Надежность:

- Обработка деления на ноль, исключение аварийных завершений.

Производительность:

- Время отклика ≤ 0.1 сек.

Портируемость:

- Работа на Windows, Linux, macOS.

9. Схемы и визуальное представление

Для лучшего понимания процессов разработки требований приведены несколько схем.

9.1. Схема 1: Основные разделы разработки требований

Эта схема иллюстрирует основные этапы формирования требований, как представлено в лекционном материале:

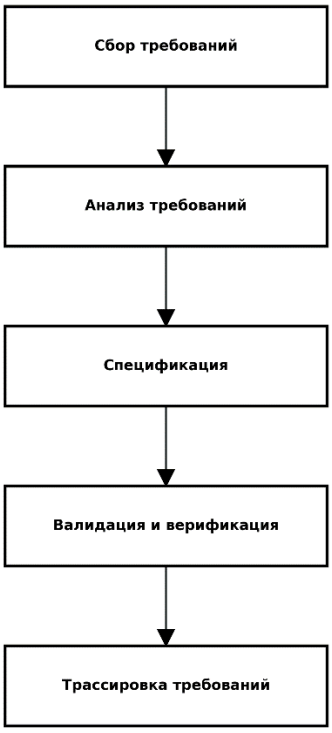

Схема 1 иллюстрирует последовательность этапов разработки требований, начиная от их сбора и заканчивая трассировкой, что позволяет обеспечить контроль корректности и полноты спецификации.

9.2. Схема 2: Компонентная диаграмма

Рассмотрим нашу диаграмму:

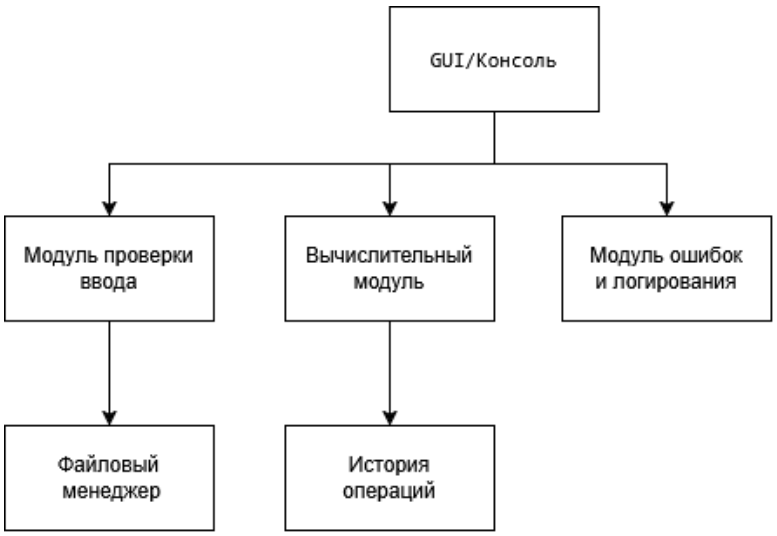

Схема 2 показывает, как система разделена на независимые модули, что обеспечивает удобство поддержки, тестирования и расширения функционала.

9.3. Схема 3: Use-case диаграмма

Ниже приведена схема основного сценария использования для программы деления чисел:

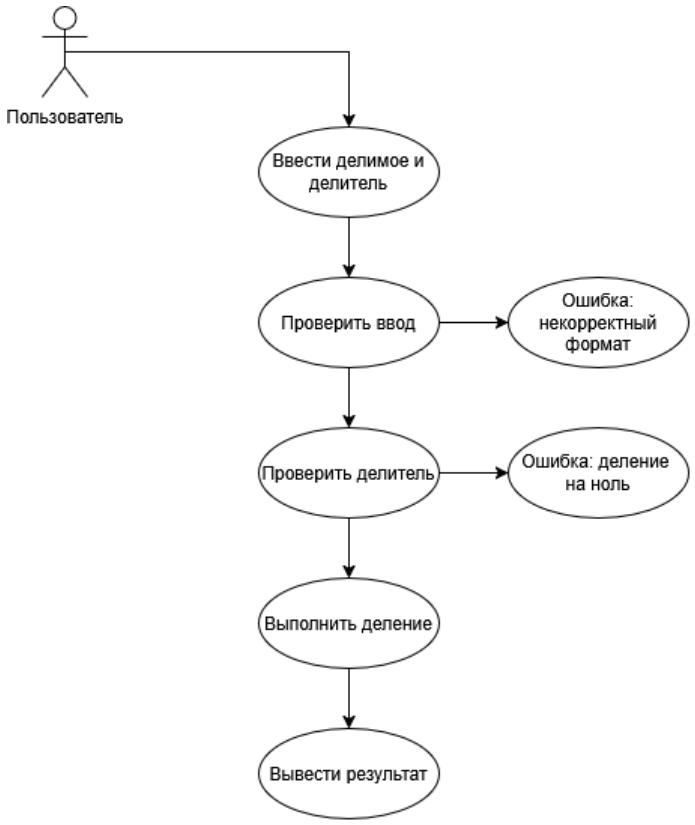

Схема 3 демонстрирует поток взаимодействия между пользователем и системой: от ввода данных до вывода результата с проверкой корректности на каждом этапе.

9.4. Схема 4: Трассировка требований

Данная схема иллюстрирует связь между функциональными требованиями, сценариями использования и тестированием:

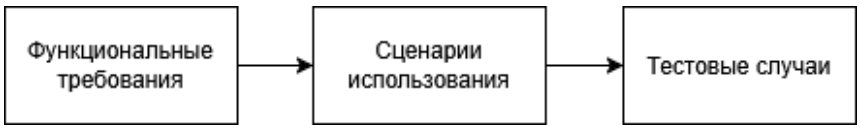

Схема 4 демонстрирует механизм трассировки: каждое требование отображается в сценариях использования, что позволяет разработчикам и тестировщикам контролировать соответствие реализованной функциональности исходным требованиям.

9.5. Схема 5: Диаграмма классов

Рассмотрим диаграмму классов

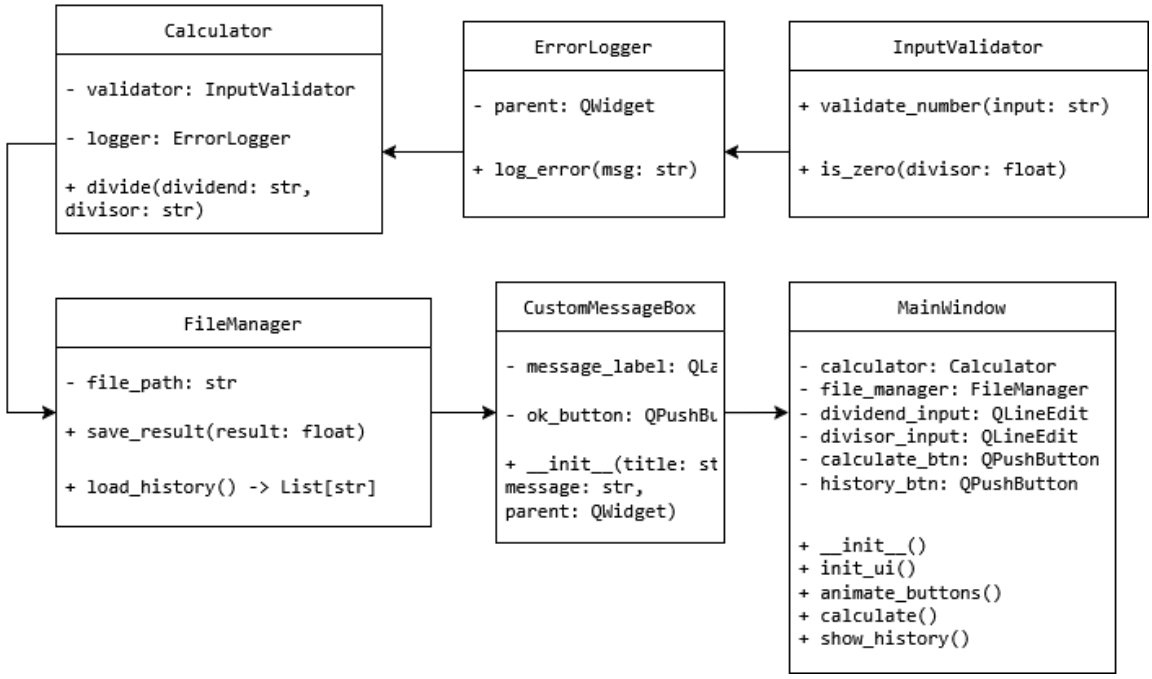

Схема 5 демонстрирует диаграмму классов

Описание классов:

1. InputValidator

Атрибуты: Нет.

Методы:

- validate_number(input: str) -> bool: Проверяет, является ли ввод числом.
- is_zero(divisor: float) -> bool: Проверяет, равен ли делитель нулю.

2. ErrorLogger

Атрибуты:

- parent: QWidget: Родительский виджет для отображения диалоговых окон.

Методы:

- log_error(msg: str): Отображает сообщение об ошибке в кастомном диалоговом окне.

3. Calculator

Атрибуты:

- validator: InputValidator: Объект для проверки ввода.

- logger: ErrorLogger: Объект для логирования ошибок.

Методы:

- divide(dividend: str, divisor: str) -> float: Выполняет операцию деления и возвращает результат.

4. FileManager

Атрибуты:

- file_path: str: Путь к файлу для сохранения истории.

Методы:

- save_result(result: float): Сохраняет результат в файл.
- load_history() -> List[str]: Загружает историю операций из файла

5. CustomMessageBox

Атрибуты:

- message_label: QLabel: Текст сообщения.
- ok_button: QPushButton: Кнопка "OK".

Методы:

- __init__(title: str, message: str, parent: QWidget): Конструктор для создания кастомного диалогового окна.

6. MainWindow

Атрибуты:

- calculator: Calculator: Объект для выполнения вычислений.
- file_manager: FileManager: Объект для работы с файлами.
- dividend_input: QLineEdit: Поле ввода делимого.
- divisor_input: QLineEdit: Поле ввода делителя.
- calculate_btn: QPushButton: Кнопка "Calculate".
- history_btn: QPushButton: Кнопка "Show History".

Методы:

- __init__(): Конструктор основного окна.
- init_ui(): Инициализация интерфейса.
- animate_buttons(): Анимация кнопок.
- calculate(): Обработка нажатия кнопки "Calculate".

- show_history(): Обработка нажатия кнопки "Show History".

Взаимосвязи между классами

MainWindow использует:

- Calculator для выполнения операций деления.
- FileManager для сохранения и загрузки истории.
- CustomMessageBox для отображения результатов, ошибок и истории

Calculator использует:

- InputValidator для проверки ввода.
- ErrorLogger для логирования ошибок.

ErrorLogger использует:

- CustomMessageBox для отображения ошибок.
- FileManager работает с файловой системой для сохранения и загрузки данных.

9.6. Схема 6: Пример интерфейса

Рассмотрим вариант интерфейса программы

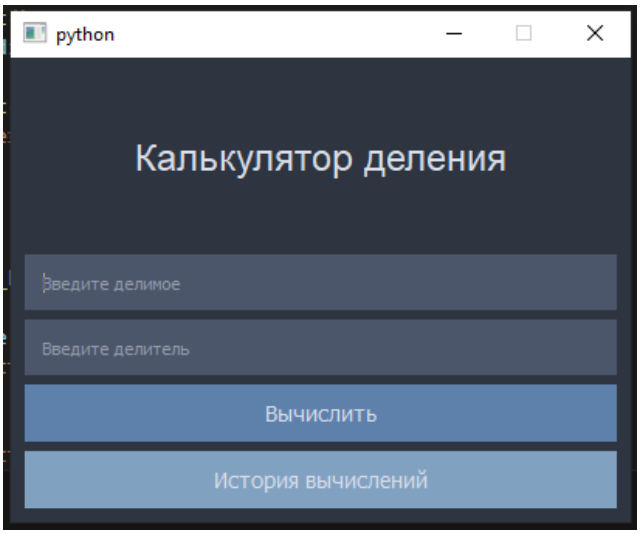

Схема 6 демонстрирует макет программы

10. Трассировка и спецификация требований

Для обеспечения качества и непротиворечивости требований применяется следующая система:

Матрица трассировки:

- Каждое требование получает уникальный идентификатор, связывающий функциональные и нефункциональные требования с соответствующими сценариями и тестовыми случаями.

Валидация и верификация:

- Экспертная оценка требований с привлечением потенциальных пользователей, прототипирование функций и согласование документа с заказчиком.

Обновление документа:

- При внесении изменений матрица трассировки обновляется, что позволяет отследить связь между изменениями в требованиях и компонентами системы.

ID	Тип	Сценарий использования	Тестовый случай
FR-01	Функциональное	Ввод данных	TC-01: Проверка ввода чисел
NFR-03	Надежность	Обработка деления на ноль	TC-05: Деление на 0 → ошибка

11. Технические аспекты реализации

11.1. Выбор платформы и языка программирования

Язык:

- Рекомендуется использовать кроссплатформенные языки (например, Java, C# или Python с библиотеками для создания GUI).

Инструменты:

- Использование современных средств разработки и фреймворков, позволяющих обеспечить модульность и масштабируемость:

  Например:

  PyQt5:

  - Для создания графического интерфейса с поддержкой анимаций и стилей.

  Qt Designer:

- Для визуального проектирования интерфейса.

Git:

- Для контроля версий и совместной разработки.

Pytest:

- Для модульного и интеграционного тестирования.

Sphinx:

- Для автоматической генерации документации из комментариев в коде.

11.2. Архитектурные решения

Модульность:

- Четкое разделение логики ввода/вывода, проверки данных, вычислительной логики и обработки ошибок:

  Модуль ввода/вывода:

  - Обрабатывает взаимодействие с пользователем через графический интерфейс.

  Модуль проверки данных:

  - Проверяет корректность введенных данных.

  Модуль вычислений:

  - Выполняет математические операции (деление).

  Модуль обработки ошибок:

  - Логирует и отображает ошибки.

  Модуль работы с файлами:

  - Сохраняет результаты и загружает историю операций.

Объектно-ориентированный подход:

- Использование классов и объектов для реализации основных функциональных блоков системы.
- Повторное использование кода.

- Упрощение тестирования.

11.3. Документация

1. Подробные комментарии в исходном коде.
2. Разработка документации, описывающей архитектуру, логику работы и инструкции по сборке/развертыванию.

   Архитектура системы:

   - Описание модулей, их взаимодействия и диаграммы классов.

   Логика работы:

   - Пошаговое описание работы программы, включая обработку ошибок и анимации.

   Инструкции по сборке/развертыванию:

   - Установка зависимостей.
   - Запуск программы.

3. Руководство пользователя

   Описание интерфейса:

   - Как вводить данные, выполнять операции и просматривать историю.

   Примеры использования:

   - Ввод: 10 (делимое), 2 (делитель).
   - Результат: 5.0.

   Обработка ошибок:

   - Что делать при некорректном вводе или делении на ноль.

4. Руководство программиста

   - Описание архитектуры и модулей.
   - Инструкции по добавлению новых функций (например, других математических операций).
   - Примеры тестов и их запуск.

12. Тестирование и контроль качества

Для обеспечения высокого качества разрабатываемой программы необходимо провести комплексное тестирование:

12.1. Функциональное тестирование

1. Проверка корректности выполнения операции деления при корректном вводе.
2. Тестирование обработки ошибок: некорректный ввод, деление на ноль.

12.2. Тестирование производительности

1. Измерение времени отклика и выполнения операции.
2. Оценка работы системы на различных платформах.
    - Тестирование на Windows, Linux и macOS для проверки кроссплатформенности.
    - Убедиться, что интерфейс и функциональность работают одинаково на всех платформах.

12.3. Тестирование удобства использования

1. Оценка интуитивности интерфейса.
    - Провести тестирование с реальными пользователями.
    - Убедиться, что интерфейс понятен и не требует дополнительных инструкций.
2. Проверка понятности сообщений об ошибках и инструкций для пользователя.
    - Сообщения об ошибках должны быть четкими и информативными.
    - Пример:
        - Некорректный ввод: "Invalid input: not a number".
        - Деление на ноль: "Division by zero".

13. Заключение

В данном докладе приведена детальная спецификация требований для разработки программы деления чисел с интеграцией визуальных схем, иллюстрирующих основные этапы:

1. от сбора и анализа требований,
2. через построение сценариев использования,
3. до модульной архитектуры и механизмов трассировки.

Основные выводы:

- Система должна обеспечивать корректный ввод, проверку, выполнение операции деления и вывод результата.

- Особое внимание уделяется обработке исключений и обеспечению стабильной работы программы.
- Использование диаграмм и схем позволяет обеспечить наглядное представление процессов разработки требований, что способствует лучшему пониманию и дальнейшей реализации проекта.

Пример проекта, в который необходимо внесение требований и доработки:

```
import sys
import os
from PyQt5.QtWidgets import (
    QApplication, QMainWindow, QWidget, QVBoxLayout, QLabel, QLineEdit,
QPushButton, QMessageBox, QDialog, QHBoxLayout
)
from PyQt5.QtCore import Qt, QPropertyAnimation, QEasingCurve, QRect
from PyQt5.QtGui import QFont

class CustomMessageBox(QDialog):
    """
    Кастомное диалоговое окно для отображения сообщений.
    """
    def __init__(self, title, message, parent=None):
        super().__init__(parent)
        self.setWindowTitle(title)
        self.setStyleSheet("background-color: #2E3440; color: #D8DEE9;")
        self.setFixedSize(300, 150)

        layout = QVBoxLayout(self)

        # Текст сообщения
        self.message_label = QLabel(message)
        self.message_label.setFont(QFont("Arial", 12))
        self.message_label.setAlignment(Qt.AlignCenter)
        layout.addWidget(self.message_label)

        # Кнопка "OK"
        self.ok_button = QPushButton("OK")
        self.ok_button.setStyleSheet(
            "background-color: #5E81AC; color: #D8DEE9; border: none; padding:
10px; font-size: 14px;"
        )
        self.ok_button.clicked.connect(self.close)
        layout.addWidget(self.ok_button, alignment=Qt.AlignCenter)


class InputValidator:
    """
    Класс для проверки корректности ввода данных.
    """
```

```
    def validate_number(self, input_str):
        try:
            float(input_str)
            return True
        except ValueError:
            return False

    def is_zero(self, divisor):
        return divisor == 0


class ErrorLogger:
    """
    Класс для логирования ошибок.
    """
    def __init__(self, parent):
        self.parent = parent

    def log_error(self, msg):
        dialog = CustomMessageBox("Ошибка", msg, self.parent)
        dialog.exec_()


class Calculator:
    """
    Класс для выполнения операции деления.
    """
    def __init__(self, parent):
        self.validator = InputValidator()
        self.logger = ErrorLogger(parent)

    def divide(self, dividend_str, divisor_str):
        if not self.validator.validate_number(dividend_str) or not
self.validator.validate_number(divisor_str):
            self.logger.log_error("Ошибка ввода данных")
            return None
        dividend = float(dividend_str)
        divisor = float(divisor_str)
        if self.validator.is_zero(divisor):
            self.logger.log_error("Деление на ноль невозможно")
            return None
        return dividend / divisor


class FileManager:
    """
    Класс для работы с файлами (сохранение и загрузка истории операций).
    """
    def __init__(self, file_path="history.txt"):
        self.file_path = file_path
        if not os.path.exists(self.file_path):
```

```
            open(self.file_path, 'w').close()

    def save_result(self, result):
        with open(self.file_path, 'a') as file:
            file.write(f"{result}\n")

    def load_history(self):
        with open(self.file_path, 'r') as file:
            return file.readlines()


class MainWindow(QMainWindow):
    """
    Основное окно приложения.
    """
    def __init__(self):
        super().__init__()
        self.setWindowTitle("")
        self.setFixedSize(400, 300)
        self.setStyleSheet("background-color: #2E3440; color: #D8DEE9;")

        # Инициализация классов
        self.calculator = Calculator(self)
        self.file_manager = FileManager()

        # Создание интерфейса
        self.init_ui()

    def init_ui(self):
        # Основной контейнер
        central_widget = QWidget()
        self.setCentralWidget(central_widget)
        layout = QVBoxLayout(central_widget)

        # Заголовок
        title = QLabel("Калькулятор деления")
        title.setFont(QFont("Arial", 18))
        title.setAlignment(Qt.AlignCenter)
        layout.addWidget(title)

        # Поля ввода
        input_layout = QVBoxLayout()
        self.dividend_input = QLineEdit()
        self.dividend_input.setPlaceholderText("Введите делимое")
        self.dividend_input.setStyleSheet("background-color: #4C566A; color:
#D8DEE9; border: none; padding: 10px;")
        input_layout.addWidget(self.dividend_input)

        self.divisor_input = QLineEdit()
        self.divisor_input.setPlaceholderText("Введите делитель")
```

```
        self.divisor_input.setStyleSheet("background-color: #4C566A; color:
#D8DEE9; border: none; padding: 10px;")
        input_layout.addWidget(self.divisor_input)

        layout.addLayout(input_layout)

        # Кнопка "Calculate"
        self.calculate_btn = QPushButton("Вычислить")
        self.calculate_btn.setStyleSheet(
            "background-color: #5E81AC; color: #D8DEE9; border: none; padding:
10px; font-size: 14px;"
        )
        self.calculate_btn.clicked.connect(self.calculate)
        layout.addWidget(self.calculate_btn)

        # Кнопка "Show History"
        self.history_btn = QPushButton("История вычислений")
        self.history_btn.setStyleSheet(
            "background-color: #81A1C1; color: #D8DEE9; border: none; padding:
10px; font-size: 14px;"
        )
        self.history_btn.clicked.connect(self.show_history)
        layout.addWidget(self.history_btn)

        # # Анимация кнопок
        # self.animate_buttons()

    def animate_buttons(self):
        # Анимация для кнопки "Calculate"
        self.anim_calculate = QPropertyAnimation(self.calculate_btn,
b"geometry")
        self.anim_calculate.setDuration(1000)
        self.anim_calculate.setStartValue(self.calculate_btn.geometry())
        self.anim_calculate.setEndValue(QRect(
            self.calculate_btn.x(), self.calculate_btn.y() - 10,
            self.calculate_btn.width(), self.calculate_btn.height()
        ))
        self.anim_calculate.setEasingCurve(QEasingCurve.OutBounce)
        self.anim_calculate.start()

        # Анимация для кнопки "Show History"
        self.anim_history = QPropertyAnimation(self.history_btn, b"geometry")
        self.anim_history.setDuration(1000)
        self.anim_history.setStartValue(self.history_btn.geometry())
        self.anim_history.setEndValue(QRect(
            self.history_btn.x(), self.history_btn.y() + 10,
            self.history_btn.width(), self.history_btn.height()
        ))
        self.anim_history.setEasingCurve(QEasingCurve.OutBounce)
        self.anim_history.start()
```

```
    def calculate(self):
        dividend = self.dividend_input.text()
        divisor = self.divisor_input.text()
        result = self.calculator.divide(dividend, divisor)
        if result is not None:
            self.file_manager.save_result(result)
            dialog = CustomMessageBox("Результат", f"Результат: {result}",
self)
            dialog.exec_()

    def show_history(self):
        history = self.file_manager.load_history()
        if history:
            history_str = "\n".join([line.strip() for line in history])
            dialog = CustomMessageBox("История", history_str, self)
            dialog.exec_()
        else:
            dialog = CustomMessageBox("История", "История операций пуста",
self)
            dialog.exec_()


if __name__ == "__main__":
    app = QApplication(sys.argv)
    window = MainWindow()
    window.show()
    sys.exit(app.exec_())
```