Пензенский государственный университет

Кафедра «Математическое обеспечение и применение ЭВМ»

**ОТЧЕТ**

по производственной практике

обучающегося \_\_Зиновьева Леонида Дмитриевича\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

*(фамилия, имя, отчество)*

3 курса группы \_\_\_\_\_22ВП2\_ факультета вычислительной техники

направления подготовки \_\_\_\_\_\_\_\_\_\_\_\_Программная инженерия (09.03.04),

проходившего практику с \_\_\_25\_\_июня\_\_\_ 20\_\_\_ по \_\_\_июля\_\_20\_\_\_

в \_\_\_АО\_«ИнфоТеКС»\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_.

*(наименование профильной организации)*

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_.

«\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_ 20\_\_\_г. \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ /\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_/

*(подпись обучающегося) (расшифровка подписи)*

«Отчет УТВЕРЖДАЮ»

Руководитель практики \_\_\_\_\_\_\_\_\_\_\_\_\_ /\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_/ \_\_\_\_\_\_\_\_\_\_\_\_\_

*(подпись) (расшифровка подписи) (должность)*

кафедры \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ ПГУ.

*( название кафедры)*

Руководитель практики от АО «ИнфоТеКС»\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

*( наименование организации, предприятия, учреждения)*

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

*(занимаемая должность) (подпись) (расшифровка подписи)*

М.П.

**Содержание**

[Введение 3](#_Toc202716594)

[1. Характеристики организации 4](#_Toc202716595)

[1.1 Анализ структурной организации предприятия 4](#_Toc202716596)

[1.2 Обзор задач, решаемых на предприятии 5](#_Toc202716597)

[1.3 Обзор используемого программного обеспечения 6](#_Toc202716598)

[1.4 Задачи, решаемые с использованием ПО на предприятии 8](#_Toc202716599)

[2. Разработка клиент–серверного приложения на языке программирования С# 9](#_Toc202716600)

[2.1 Постановка задачи 9](#_Toc202716601)

[2.2 Анализ требований 10](#_Toc202716602)

[2.3 Проектирование. Структура программного обеспечения 12](#_Toc202716603)

[2.4 Реализация программы 14](#_Toc202716604)

[Список использованных источников 16](#_Toc202716605)

[Приложение A (обязательное) Код программы 17](#_Toc202716606)

[Приложение Б (рекомендуемое) Диаграммы 22](#_Toc202716607)

# Введение

Летняя практика в компании позволила погрузиться в задачи разработки комплексных программных решений, объединяющих клиентские и серверные компоненты. В рамках работы были реализованы два ключевых проекта: консольная утилита для взаимодействия с бинарными файлами на стороне клиента и веб – сервис для централизованного доступа к функционалу дампа. Такое сочетание позволило получить целостное представление о построении распределенных систем и обеспечении надежности обмена данными между разными компонентами.

В процессе практики широко применялись современные технологии и инструменты экосистемы .NET: от организации асинхронных операций и работы с файловой системой до разработки приложения с REST архитектурой и применения Thrift интерфейсов. Особое внимание уделялось выстраиванию архитектуры с разделением на слои ответственности, что обеспечило гибкость, масштабируемость и легкость сопровождения кода. Важным аспектом стало применение средств валидации входных данных и централизованного логирования, что повысило устойчивость приложений к ошибочным ситуациям.

Полученные знания и навыки представляют собой прочную основу для дальнейшего совершенствования в области разработки программного обеспечения. Практика не только обогатила технические знания, но и укрепила понимание важности тестирования, оптимизации и поддержки качества кода на всех этапах жизненного цикла проекта.

# Характеристики организации

## 1.1 Анализ структурной организации предприятия

АО «ИнфоТеКС» (Информационные Технологии и Коммуникационные Системы) – российская компания, специализирующаяся на разработке программно–аппаратных VPN–решений и средств криптографической защиты информации. Основанная 6 сентября 1991 года группой экспертов по информационной безопасности во главе с Андреем Чапчаевым, компания входит в пятерку крупнейших российских организаций в области защиты информации [1].

В России у ИнфоТеКС есть три дочерние компании:

1. ОАО «ИнфоТеКС Интернет Траст», основанное в 2001 году, предоставляет услуги по защите информации с использованием продуктов и технологий ViPNet, включая электронный документооборот. Компания является удостоверяющим центром и оператором сдачи отчетности в электронном виде для ПФР, ФНС и других государственных учреждений, входя в ТОП–10 крупнейших удостоверяющих центров России.
2. ЗАО «Перспективный мониторинг», созданное в 2007 году, занимается исследованием состояния безопасности информационных систем организаций, выявлением уязвимостей и недокументированных сервисов. Компания оказывает экспертную поддержку в разработке политик, требований и инструкций по информационной безопасности, а также в актуализации существующих регламентов под изменяющиеся бизнес–требования.
3. Учебный центр ИнфоТеКС, функционирующий с 2000 года, сотрудничает с ведущими вузами страны, предоставляя консультации и обучение специалистам в сфере информационной безопасности. Центр обеспечивает теоретическую и практическую подготовку клиентов, сотрудников и партнеров компании в вопросах построения комплексных систем защиты информации и использования средств защиты в автоматизированных системах.

Кроме того, в состав ГК «ИнфоТеКС» входят компании «Системы практической безопасности» и «СФБ лаборатория», предоставляющая услуги по сертификации средств защиты информации и проведению контрольных исследований программных и программно–аппаратных СЗИ по требованиям ФСБ России.

## 1.2 **Обзор задач, решаемых на предприятии**

Компания «ИнфоТеКС» – ведущий отечественный разработчик и производитель высокотехнологичных программных и программно–аппаратных средств защиты информации. Основная продукция компании, известная благодаря решениям для защиты каналов связи корпоративных сетей, была преобразована в современный комплекс, ориентированный на защиту от актуальных угроз информационной безопасности. В портфолио компании также входят решения по обнаружению и предотвращению угроз и вторжений, защите серверов и рабочих станций от несанкционированного доступа, а также решения для защиты мобильных устройств и коммуникаций, ИБ АСУ ТП, IIoT, криптографические сервисы, библиотеки и модули для встраивания [2].

Некоторые из наиболее популярных продуктов компании включают:

1. ViPNet CSS Connect HW: стационарный телефон с сенсорным экраном, предназначенный для общения пользователей сети ViPNet по защищенному каналу.
2. ViPNet SafeBoot 3: новое поколение программного модуля доверенной загрузки (ПМДЗ), сертифицированного ФСБ и ФСТЭК России. Предназначен для создания точки доверия к платформе и ее компонентам, а также загружаемой операционной системе, обеспечивая защиту UEFI BIOS и организацию доверенной загрузки ОС.
3. ViPNet Client: программный комплекс для защиты рабочих мест корпоративных пользователей, надежно защищающий от сетевых атак за счет фильтрации трафика и обеспечивающий защищенную работу с корпоративными данными через зашифрованный канал, включая удаленных пользователей.
4. ViPNet OSSL: программное обеспечение на базе библиотеки OpenSSL, позволяющее использовать российские криптографические алгоритмы ГОСТ через интерфейс OpenSSL.
5. ViPNet Hardware Security Module (ViPNet HSM): универсальный криптографический модуль для реализации криптографических операций по запросу различных прикладных сервисов.
6. ViPNet Quantum Trusted System Lite (ViPNet QTS Lite): квантовая криптографическая система для выработки и распределения ключей, обеспечивающая квантовозащищенными ключами средства криптографической защиты информации (СКЗИ) в автоматическом режиме.
7. ViPNet CryptoSmart: криптопровайдер для работы с распределенными реестрами, предназначенный для защиты распределенных реестров и смарт–контрактов с использованием криптографических алгоритмов ГОСТ, встраиваемый в блокчейн–платформы на базе Hyperledger Fabric.
8. ViPNet HashCalc: утилита для контроля целостности дистрибутивов программных продуктов компании ИнфоТеКС, которые доступны для загрузки на сайте компании.

Эти продукты демонстрируют широкий спектр решений компании «ИнфоТеКС» для защиты информации в различных областях и обеспечивают надежную защиту данных от актуальных угроз.

## 1.3 Обзор используемого программного обеспечения

Microsoft Visual Studio сочетает полноценный редактор кода с мощным компилятором и инструментами анализа. Интегрированные механизмы IntelliSense и статического анализа ускоряют разработку и повышают качество исходного кода, а средства рефакторинга облегчают масштабирование и реструктуризацию проектов. Отладчик на уровне исходного текста с точками останова и профилировщик производительности позволяют оперативно и точно выявлять узкие места в приложениях. Поддержка тестовых фреймворков MSTest, NUnit и xUnit обеспечивает автоматизированную проверку корректности работы модулей.

Прямое подключение к системам контроля версий (Git, Team Foundation Version Control) упрощает управление изменениями, а интеграция с облачными платформами Azure и AWS вместе с DevOps–пайплайнами гарантирует сквозную автоматизацию CI/CD. Универсальная поддержка .NET, Unity и других платформ позволяет создавать решения для десктопа, мобильных устройств, веба и игровых консолей, а богатая экосистема расширений Visual Studio обеспечивает адаптацию среды под любые требования проекта.

Docker представляет собой платформу для автоматизации развертывания и управления контейнерами приложениями. Она позволяет упаковать программу со всеми зависимостями в изолированный контейнер, совместимый с любой Linux–системой, поддерживающей контрольные группы, а встроенные команды обеспечивают создание, запуск и управление жизненным циклом контейнеров.

Git представляет собой распределенную систему контроля версий, широко применяемую для управления исходным кодом в процессе разработки программного обеспечения. Предоставляемые инструменты позволяют отслеживать изменения в файлах, координировать работу нескольких разработчиков и управлять версиями проектов, при этом каждая копия репозитория хранит полную историю изменений для автономного выполнения задач. Применение Git вместе с сопутствующими технологиями обеспечивает эффективную разработку, тестирование и развертывание приложений в различных средах.

Apache Thrift представляет собой фреймворк удаленного вызова процедур (RPC), предназначенный для разработки высокопроизводительных межъязыковых сервисов. Описание структур данных и интерфейсов выполняется в едином языке IDL, на основе которого автоматически генерируются клиентские и серверные компоненты для поддерживаемых языков программирования. Абстракции протоколов и транспортов, включая оптимизированные двоичные и компактные схемы сериализации, обеспечивают гибкость, надежность и эффективность взаимодействия в распределенных системах.

Эти инструменты и технологии обеспечивают эффективную разработку, тестирование и развертывание приложений в различных средах.

## 1.4 Задачи, решаемые с использованием ПО на предприятии

В процессе разработки и сопровождения программных продуктов интегрированная среда Microsoft Visual Studio обеспечивает полный цикл создания приложения: от редактирования исходного кода и анализа до отладки на уровне исходного кода и выполнения модульных тестов. Возможности IntelliSense, рефакторинга и визуального профилирования способствуют повышению производительности разработчиков при создании консольных и графических приложений, веб–сервисов и серверных функций на платформе .NET. Интеграция с системами контроля версий и облачными DevOps–платформами обеспечивает сквозную автоматизацию процессов CI/CD.

Распределенная система контроля версий Git обеспечивает координацию работы команды над единой кодовой базой. Использование изолированных веток для разработки новых функциональных возможностей и исправления ошибок минимизирует конфликты изменений, а процедуры pull–request и code–review упрощают проверку и интеграцию нового функционала. Локальное хранение полной истории изменений обеспечивает возможность быстрого возврата к любому предыдущему состоянию проекта.

Платформа контейнеризации Docker применяется для стандартизации и масштабирования окружений разработки и тестирования. Приложения совместно с необходимыми сервисами (базами данных, брокерами сообщений и пр.) упаковываются в изолированные контейнеры с жестко зафиксированными зависимостями. Это гарантирует воспроизводимость поведения при развертывании на различных инфраструктурах и оптимизирует создание автоматизированных CI/CD–конвейеров с непрерывным тестированием и доставкой образов.

Фреймворк Apache Thrift используется для организации высокопроизводительного межъязыкового взаимодействия внутри распределенной архитектуры. Определение структур данных и интерфейсов в едином IDL позволяет автоматически генерировать клиентские и серверные компоненты для различных языков программирования. Применение Thrift обеспечивает эффективную сериализацию и транспорт данных по оптимизированным двоичным протоколам, упрощает расширение системы новыми сервисами и повышает надежность обмена сообщениями между компонентами.

# Разработка клиент–серверного приложения на языке программирования С#

## 2.1 Постановка задачи

В рамках практики предусматривалось создание комплексное решение для анализа бинарных файлов, включающее консольное приложение для интерактивного просмотра шестнадцатеричного дампа и серверное приложение на платформе ASP .NET Core с Thrift – интерфейсом для удаленного получения дампа и расчета объема данных, при этом описать протокол и модели данных в едином Thrift – IDL, организовать консистентную валидацию входных параметров и обработку исключений, а также обеспечить готовность компонентов к контейнеризации и интеграции с помощью Docker.

## 2.2 Анализ требований

Для решения поставленной задачи необходимо учитывать специфику каждого компонента системы и их взаимодействие. Основные требования делятся на функциональные и нефункциональные [3].

Функциональные требования для консольной утилиты для вывода HexDump:

1. Чтение бинарного файла порциями заданного размера и вычисление шестнадцатеричного дампа с ASCII–интерпретацией каждой порции.
2. Интерактивный постраничный вывод дампа с навигацией по строкам и столбцам (стрелочные клавиши, PageUp/PageDown, Home/End, выход по Esc).
3. Конфигурирование параметров работы: выбор кодировки, размера блока, режима вывода (консоль или файл) и указание пути к файлу через аргументы командной строки, конфигурационный файл или интерактивный ввод.
4. Вызов удаленных методов Thrift–сервиса для получения общего числа блоков и фрагментов дампа.
5. Корректная обработка ошибок доступа к файлам и ошибок сетевого взаимодействия с сервером, вывод информативных сообщений.
6. Наличие модульных тестов для проверки основных сценариев работы (чтение блоков, навигация, обработка ошибок).

Нефункциональные требования для консольной утилиты для вывода HexDump:

1. Эффективное использование памяти за счет блочного чтения без загрузки всего файла в оперативную память.
2. Высокая отзывчивость интерфейса при навигации по дампу.
3. Устойчивость к ошибкам: отсутствие аварийного завершения при некорректных входных данных или сбоях связи.
4. Легкость сопровождения через четкое разделение слоев (ввод/вывод, бизнес–логика, взаимодействие с сервисом).

Функциональные требования для серверного приложения в клиент – серверной архитектуре:

1. Реализация Thrift – интерфейса с методами для работы с дампом бинарного файла.
2. Централизованная валидация входных параметров и возврат ошибок в формате ProblemDetails.
3. Логирование всех запросов, ответов и исключений через Serilog с возможностью ротации лог–файлов.
4. Описание протокола и моделей данных в едином Thrift–IDL для автоматической генерации клиентских и серверных модулей.

Нефункциональные требования для серверного приложения в клиент – серверной архитектуре:

1. Обработка параллельных RPC–запросов без блокировок, обеспечение масштабируемости и высокой пропускной способности.
2. Конфигурация параметров сервиса (порт, пути к хранилищам, настройки логирования) через внешние файлы или переменные окружения.
3. Надежность: корректное освобождение ресурсов Thrift–соединений и отсутствие утечек.
4. Модульная архитектура («API», «Application», «Domain», «Infrastructure») для упрощения расширения и поддержки.

На рисунке 1 представлена диаграмма вариантов использования для пользователя [4].

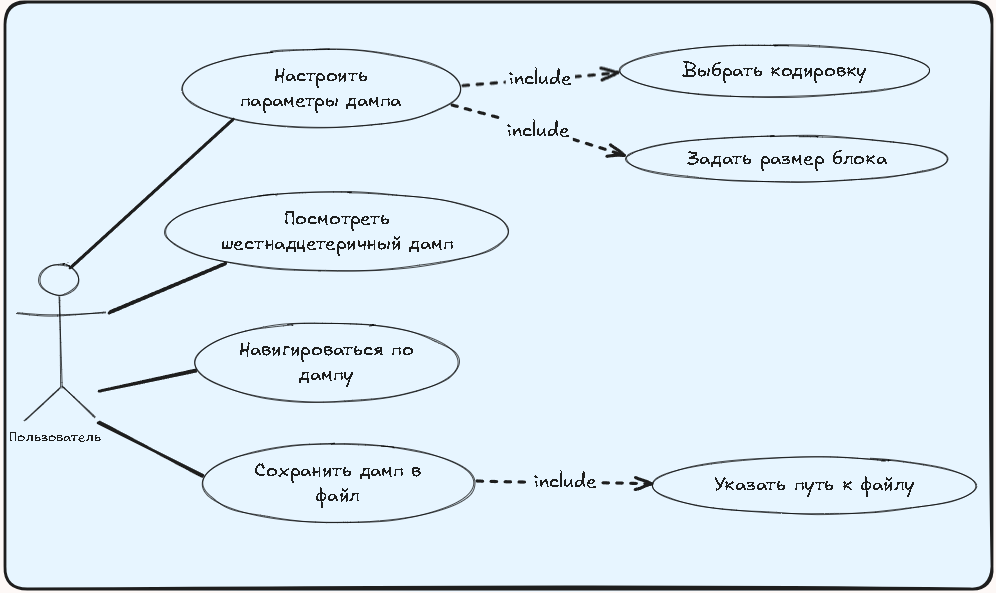


Рисунок 1 – Диаграмма вариантов использования

Представленная диаграмма вариантов использования демонстрирует ключевые сценарии взаимодействия пользователя с системой. Учёт как функциональных, так и нефункциональных требований позволяет добиться надёжности, масштабируемости и удобства эксплуатации приложения.

## 2.3 Проектирование. Структура программного обеспечения

Диаграмма классов, консольного клиента соответствующая структуре программного обеспечения, представлена на рисунке Б.1 в приложении Б.

Описание основного класса FileDumpPresenter.

Поля:

1. \_ui – экземпляр IUserInterface для взаимодействия с пользователем (вывод сообщений, чтение ввода).
2. \_dumper – экземпляр IFileDumper для отправки запросов к удаленному сервису и получения данных дампа.
3. \_settingsProvider – экземпляр ISettingsProvider для получения настроек дампа.
4. \_settingsValidator – экземпляр ISettingsValidator для проверки корректности настроек.
5. \_currentSettings – объект DumpSettingsDto, хранящий текущие параметры дампа (кодировка, путь к файлу, размер блока и пр.).
6. \_totalLinesResponse – объект TotalLinesResponse, содержащий общее число доступных строк (блоков) в файле.

Методы:

1. FileDumpPresenter(IUserInterface ui, IFileDumper dumper, ISettingsProvider settingsProvider, ISettingsValidator settingsValidator) – конструктор, инициализирующий все зависимости и получающий начальные настройки.
2. void Run() – основной метод: выполняет валидацию настроек, запрашивает общее число строк, затем в зависимости от режима вывода либо сохраняет дамп в файл, либо запускает постраничный вывод в консоль.
3. private void DumpToConsole() – настраивает и запускает объект Pager для интерактивной навигации по получаемым строкам дампа.
4. private void OnPageRequested(object sender, PageRequestedEventArgs e) – обработчик события PageRequested от Pager: отправляет запрос за очередным фрагментом дампа и передает полученные строки в Pager.DisplayLines.
5. private int CalculateMaxLineLength(int blockSize) – рассчитывает максимальную ширину выводимой строки дампа (адрес + hex – часть + ASCII) по заданному размеру блока.

Диаграмма классов, веб – приложения соответствующая структуре программного обеспечения, представлена на рисунке Б.2 в приложение Б.

Основной класс – FileDumpServiceHandler реализующий thrift – интерфейс представлен в листинге А.1 приложения А.

Описание основного класса FileDumpServiceHandler**.**

Поля:

1. \_fileDumpService – экземпляр IFileDumpService для выполнения бизнес–логики по созданию дампа.
2. \_logger – экземпляр ILogger<FileDumpServiceHandler> для записи информационных и ошибочных событий.
3. \_validator – экземпляр IValidator<DumpSettingsDto> (реализован DumpRequestDtoValidator) для проверки корректности входных настроек.

Методы:

1. Task<BinaryDump> DumpToConsole(DumpSettings settings, long startLine, int numberOfLines, CancellationToken) – выполняет валидацию параметров, вызывает DumpFileAsync бизнес–сервиса и возвращает список строк дампа.
2. Task<SharedDumpFileResponseDto> DumpToFile(DumpSettings settings, long startLine, int numberOfLines, CancellationToken) – валидирует настройки, производит асинхронную запись дампа в файл и возвращает информацию о результате.
3. Task<long> GetTotalLines(string filePath, int blockSize, CancellationToken) – проверяет входные аргументы и возвращает общее количество блоков (строк) в файле.

## 2.4 Реализация программы

В данном разделе подробно описывается процесс создания и интеграции двух основных компонентов системы – консольного клиента и веб–сервиса на базе ASP .NET Core с Thrift.

Консольное приложение реализовано в трех слоях. На уровне пользовательского интерфейса присутствуют интерфейсы IUserInterface и IPager, а их конкретные реализации ConsoleUserInterface и Pager отвечают за ввод–вывод и постраничную навигацию в консоли. Логика получения настроек вынесена в SettingsProvider и SettingsValidator: первый класс читает конфигурацию из файла или интерактивно запрашивает параметры у пользователя, второй – проверяет их корректность. Слой доступа к данным представляет собой фасад FileDumper поверх асинхронного Thrift–клиента ThriftProtocolService, реализующего интерфейс IClientProtocolService. Весь процесс работы описан в классе FileDumpPresenter: сначала здесь валидируются настройки и вычисляется общее число блоков, затем, в зависимости от выбранного режима, либо отправляется запрос на сохранение дампа в файл, либо инициализируется постраничный вывод, где для каждой страницы через IPager генерируются события PageRequested, приводящие к последовательным вызовам FileDumper и отображению результата.

Серверная часть выстроена по многослойной архитектуре «API – Application – Domain – Infrastructure». Входной API–уровень представлен Thrift – хендлером FileDumpServiceHandler, реализующим сгенерированный интерфейс FileDumpServiceThrift.IAsync. Этот класс берет на себя валидацию параметров через FluentValidation, логирование операций с помощью Serilog и обработку исключений, которые через глобальный ExceptionHandlingMiddleware преобразуются в стандартные ProblemDetails. В Application–слое интерфейс IFileDumpService и его реализация FileDumpService отвечают за конвертацию DTO в доменные модели и делегирование работы инфраструктурному компоненту. Domain – уровень включает сущности DumpSettings и BinaryDump, а Infrastructure–уровень – класс FileDumper, работающий с файловой системой через абстракцию IFileSystem: здесь читаются данные блоками, формируется hex/ASCII–дамп и, при необходимости, выполняется запись в файловую систему.

Для унификации взаимодействия определен единый Thrift – IDL, содержащий перечисление SharedEncodingEnum, структуры DumpSettings, BinaryDump и SharedDumpFileResponseDto, а также сервис FileDumpServiceThrift с тремя методами. На его основе автоматически генерируются C#–клиенты и серверный прокси, что исключает рассогласование DTO.

В результате реализована надежная и производительная система, обеспечивающая как локальное интерактивное исследование бинарных данных, так и масштабируемый удаленный доступ через стандартизованный RPC–интерфейс.

# Список использованных источников

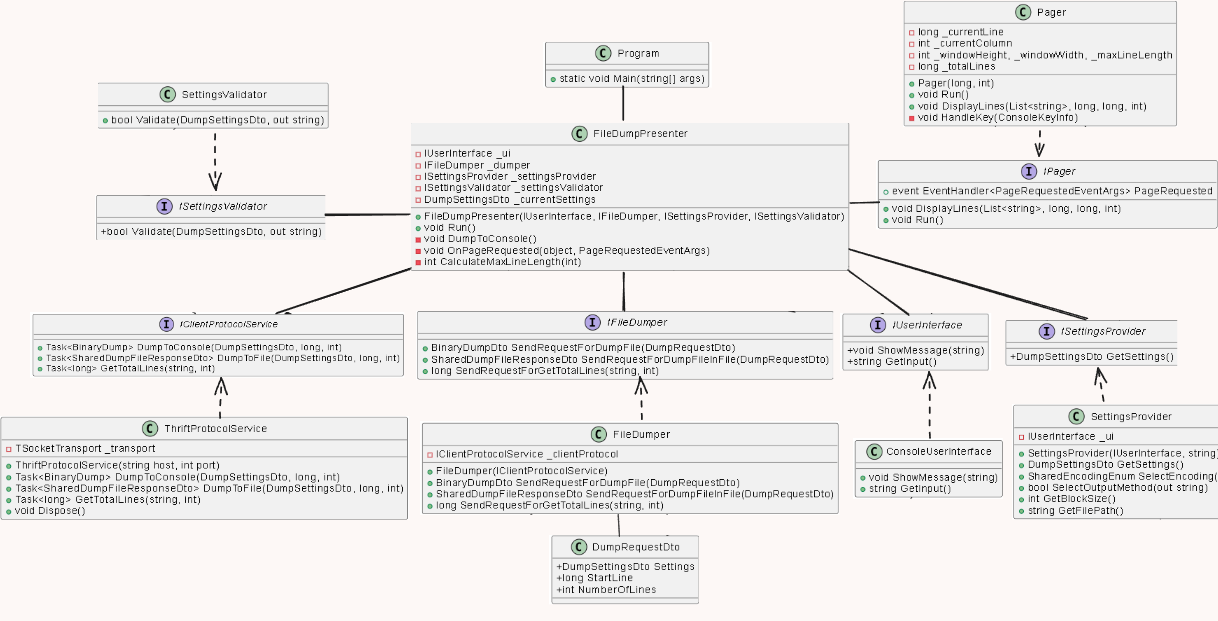
1. Компания ИнфоТеКС [электронный ресурс] // ru.wikipedia.org – URL: <https://ru.wikipedia.org/wiki/ИнфоТеКС#cite_note–1> (дата обращения: 29.06.2025).
2. ИнфоТеКС (Infotecs) [электронный ресурс] // tedviser.ru – URL: https://www.tadviser.ru/index.php/Компания:ИнфоТеКС\_(Infotecs) (дата обращения: 01.07.2025).
3. Мартин Р. Чистая архитектура. Искусство разработки программного обеспечения / Р. Мартин. – Спб.: Питер, 2018. – 352 с.
4. Арлоу Д. UML2 и Унифицированный процесс. Практический объектно-ориентированный анализ и проектирование / Д. Арлоу, А. Нейштадт. – Санкт-Петербург, Издательство Символ-Плюс, 2007. – 624 с.

# Приложение A (обязательное) Код программы

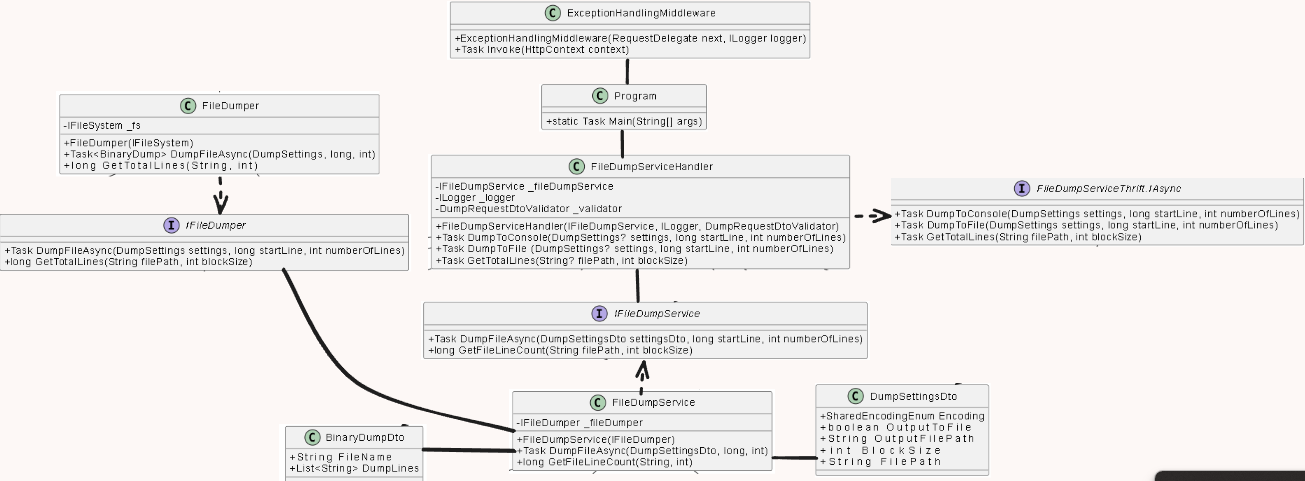
Листинг А.1 – Реализация Thrift – интерфейса

using FluentValidation;  
using HexDumpWeb.Application.DTOs;  
using HexDumpWeb.Application.Interfaces;  
using HexDumpWeb.Thrift.Generated;  
using SharedLibrary.SharedHelpers;  
  
namespace HexDumpWeb.API.ThriftHandler  
{  
 /// <summary>  
 /// Реализует интерфейс сервера <see cref="FileDumpServiceThrift"/>  
 /// </summary>  
 public class FileDumpServiceHandler : FileDumpServiceThrift.IAsync  
 {  
 private readonly ILogger<FileDumpServiceHandler> \_logger;  
 private readonly IFileDumpService \_fileDumpService;  
 private readonly IValidator<DumpSettingsDto> \_validator;  
  
 /// <summary>  
 /// Конструктор  
 /// </summary>  
 /// <param name="fileDumpService">Объект для выполнения дампа</param>  
 /// <param name="logger">Объект для работы с логгированием  
 /// </param>  
 public FileDumpServiceHandler(IFileDumpService fileDumpService,  
 ILogger<FileDumpServiceHandler> logger, IValidator<DumpSettingsDto> validator)  
 {  
 \_fileDumpService = fileDumpService;  
 \_logger = logger;  
 \_validator = validator;  
 }  
  
 /// <summary>  
 /// Выполненяет дамп содержимого файла по указанным параметрам дампа в консоль   
 /// </summary>  
 /// <param name="settings">Настройки дампа.</param>  
 /// <param name="startLine">Номер начальной строки.</param>  
 /// <param name="numberOfLines">Колличество требуемых строк.</param>  
 /// <param name="cancellationToken">Колличество требуемых строк.</param>  
 /// <returns>Список строк дампа для вывода</returns>  
 /// <exception cref="ArgumentNullException">Исключение при пустых настройках</exception>  
 public async Task<BinaryDump> DumpToConsole(DumpSettings? settings, long startLine,  
 int numberOfLines, CancellationToken cancellationToken = default)  
 {  
 if (settings == null)  
 {  
 throw new ArgumentNullException(nameof(settings));  
 }  
   
 \_logger.LogInformation("Thrift: Запрос на дамп в файл: FilePath={FilePath}," +  
 " OutputFilePath={OutputFilePath}",  
 settings.FilePath, settings.OutputFilePath);  
   
 var dumpSettings = new DumpSettingsDto  
 {  
 Encoding = ConverterEncodingEnum.ConvertByIndex(settings.Encoding),  
 OutputToFile = true,  
 OutputFilePath = settings.OutputFilePath,  
 BlockSize = settings.BlockSize,  
 FilePath = settings.FilePath  
 };  
  
 var validationResult = await \_validator.ValidateAsync(dumpSettings);  
 if (!validationResult.IsValid)  
 {  
 throw new ValidationException(validationResult.Errors);  
 }  
  
 var dumpDto = await \_fileDumpService.DumpFileAsync(dumpSettings, startLine, numberOfLines);  
  
 \_logger.LogInformation("Thrift: Дамп файла успешно получен" +  
 "Содержание дампа: {@dumpDto}", dumpDto);  
  
 return new BinaryDump  
 {  
 DumpLines = dumpDto.DumpLines  
 };  
 }  
  
 /// <summary>  
 /// Выполненяет дамп содержимого файла по указанным параметрам дампа в файл.  
 /// </summary>  
 /// <param name="settings">Настройки дампа.</param>  
 /// <param name="startLine">Номер начальной строки.</param>  
 /// <param name="numberOfLines">Колличество требуемых строк.</param>  
 /// <param name="cancellationToken"></param>  
 /// <returns>Сообщение о результате и путь к файлу</returns>  
 /// <exception cref="ArgumentNullException">Исключение при пустых настройках</exception>  
 public async Task<SharedDumpFileResponseDto> DumpToFile(DumpSettings? settings, long startLine,  
 int numberOfLines, CancellationToken cancellationToken = default)  
 {  
 if (settings == null)  
 {  
 throw new ArgumentNullException(nameof(settings));  
 }  
  
 \_logger.LogInformation("Thrift: Запрос на дамп в файл: FilePath={FilePath}," +  
 " OutputFilePath={OutputFilePath}",  
 settings.FilePath, settings.OutputFilePath);  
  
 var dumpSettings = new DumpSettingsDto  
 {  
 Encoding = ConverterEncodingEnum.ConvertByIndex(settings.Encoding),  
 OutputToFile = true,  
 OutputFilePath = settings.OutputFilePath,  
 BlockSize = settings.BlockSize,  
 FilePath = settings.FilePath  
 };  
  
 var validationResult = await \_validator.ValidateAsync(dumpSettings);  
 if (!validationResult.IsValid)  
 {  
 throw new ValidationException(validationResult.Errors);  
 }  
  
 var dumpDto = await \_fileDumpService.DumpFileAsync(dumpSettings, startLine, numberOfLines);  
  
 \_logger.LogInformation("Thrift: Дамп файла успешно записан в файл: {OutputFilePath}" +  
 "Содержание дампа: {@dumpDto}", dumpSettings.OutputFilePath, dumpDto);  
  
 return new SharedDumpFileResponseDto  
 {  
 Message = "Дамп успешно записан в файл.",  
 OutputFilePath = dumpSettings.OutputFilePath  
 };  
 }  
  
 /// <summary>  
 /// Выполняет получение общего количества строк  
 /// </summary>  
 /// <param name="filePath">Путь к файлу для дампа.</param>  
 /// <param name="blockSize">Размер одного блока</param>  
 /// <param name="cancellationToken"></param>  
 /// <returns>Общее кол-во строк.</returns>  
 /// <exception cref="ArgumentNullException">Исключение при пустых настройках</exception>  
 public Task<long> GetTotalLines(string? filePath, int blockSize, CancellationToken cancellationToken = default)  
 {  
 if (filePath == null)  
 {  
 throw new ArgumentNullException(nameof(filePath));  
 }  
 if (blockSize == 0)  
 {  
 throw new ArgumentNullException(nameof(blockSize));  
 }  
  
 \_logger.LogInformation("Получен запрос на получение общего количества строк. FilePath: {FilePath}, BlockSize: {BlockSize}", filePath, blockSize);  
  
 var totalLines = \_fileDumpService.GetFileLineCount(filePath, blockSize);  
 \_logger.LogInformation("Общее количество строк для файла {FilePath}: {TotalLines}", filePath, totalLines);  
 return Task.FromResult(totalLines);  
 }  
 }  
}

# Приложение Б (рекомендуемое) Диаграммы



Приложение Б.1 – Диаграмма классов консольного клиента



Приложение Б.2 – Диаграмма классов веб – сервера