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.

|  |
| --- |
| РЕФЕРАТ  Пояснительная записка содержит 16 листов, 3 рисунка, использованных источников 5.  С#, КЛИЕНТ–СЕРВЕРНОЕ ПРИЛОЖЕНИЕ, ОБЪЕКТНО–ОРИЕНТИРОВАННОЕ ПРОГРАММИРОВАНИЕ, КОНСОЛЬНАЯ УТИЛИТА, REST API.  Цель работы: разработка клиент–серверного приложения, реализующего вывод данных переданного файла в шестнадцатеричном формате.  Технология разработки – язык программирования С#.  Результат работы – реализация приложения для выполнения вывода данных файла в особом формате. |

Оглавление

[Введение 4](#_Toc202298498)

[1. Разработка клиент–серверного приложения на языке программирования С# 5](#_Toc202298499)

[1.1. Анализ структурной организации предприятия 5](#_Toc202298500)

[1.2. Обзор задач, решаемых на предприятии 6](#_Toc202298501)

[1.3. Обзор используемого программного обеспечения 7](#_Toc202298502)

[1.4. Задачи, решаемые с использованием ПО на предприятии 8](#_Toc202298503)

[1.5. Решение задачи, поставленной руководителем предприятия 9](#_Toc202298504)

[1.5.1. Постановка задачи 9](#_Toc202298505)

[1.5.2. Анализ требований 9](#_Toc202298506)

[1.5.3. Проектирование. Структура программного обеспечения 11](#_Toc202298507)

[1.5.4. Реализация программы 13](#_Toc202298508)

[Список использованных источников 16](#_Toc202298509)

# 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 распределённых систем и обеспечении надёжности 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 лёгкость 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.

# Разработка клиент–серверного приложения на языке программирования С#

## 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.

## 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 её 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.

## Обзор используемого программного обеспечения

**Microsoft Visual Studio**

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**

Docker представляет собой платформу для автоматизации развёртывания 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**

Git представляет собой распределённую 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 развёртывание приложений в различных средах.

**Apache Thrift**

Apache Thrift представляет собой фреймворк удалённого 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, надёжность и эффективность взаимодействия в распределённых системах.

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

## 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.

Распределённая 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 жёстко 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 распределённой 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 надёжность обмена сообщениями между компонентами.

## Решение задачи, поставленной руководителем предприятия

### 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 удалённого получения дампа и расчёта объёма 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.

### Анализ требований

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

**Консольная утилита для вывода HexDump**

Функциональные требования:

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 удалённых 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:

1. Эффективное использование памяти за счёт 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. Лёгкость сопровождения через чёткое разделение слоёв (ввод/вывод, бизнес–логика, взаимодействие с сервисом).

**Серверное приложение в клиент–серверной архитектуре**

Функциональные требования:

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:

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. Надёжность: 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].

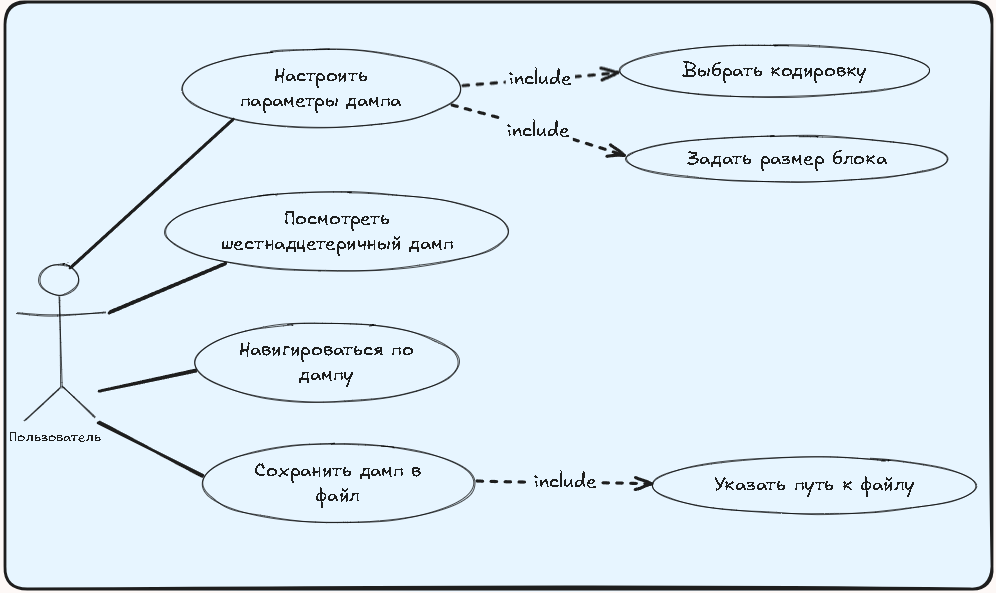


Рисунок 1 – Диаграмма вариантов использования

## Проектирование. Структура программного обеспечения

Диаграмма классов, консольного клиента соответствующая структуре программного обеспечения, представлена на рисунке 2.

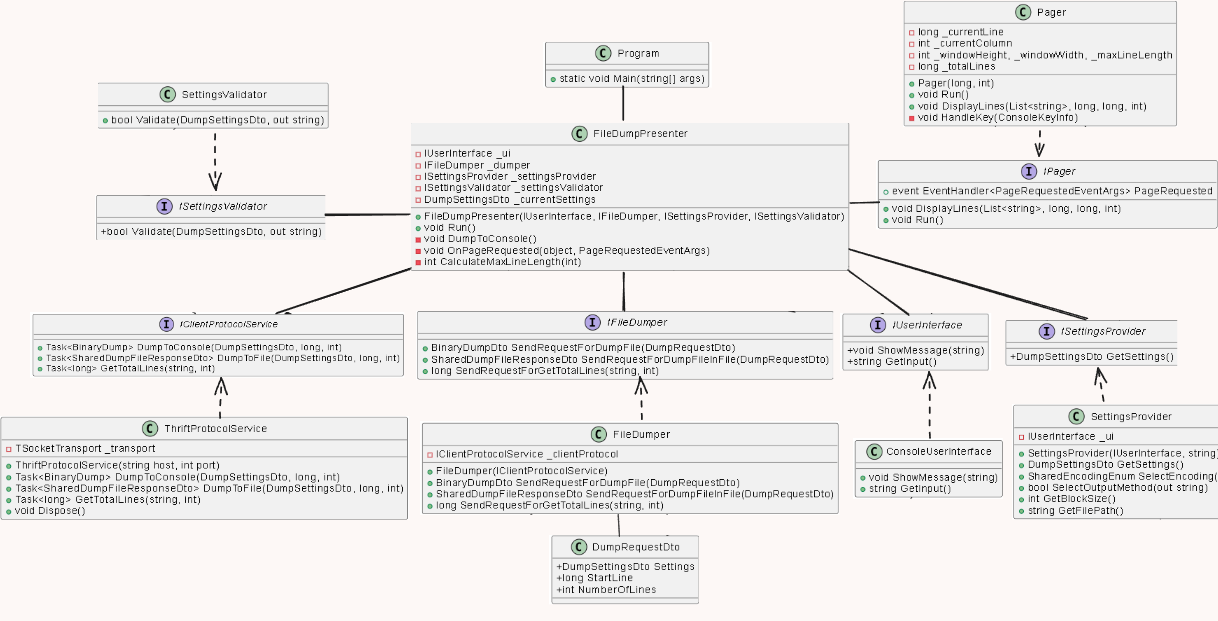


Рисунок 2 – Диаграмма классов

**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 удалённому 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 передаёт 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 3.

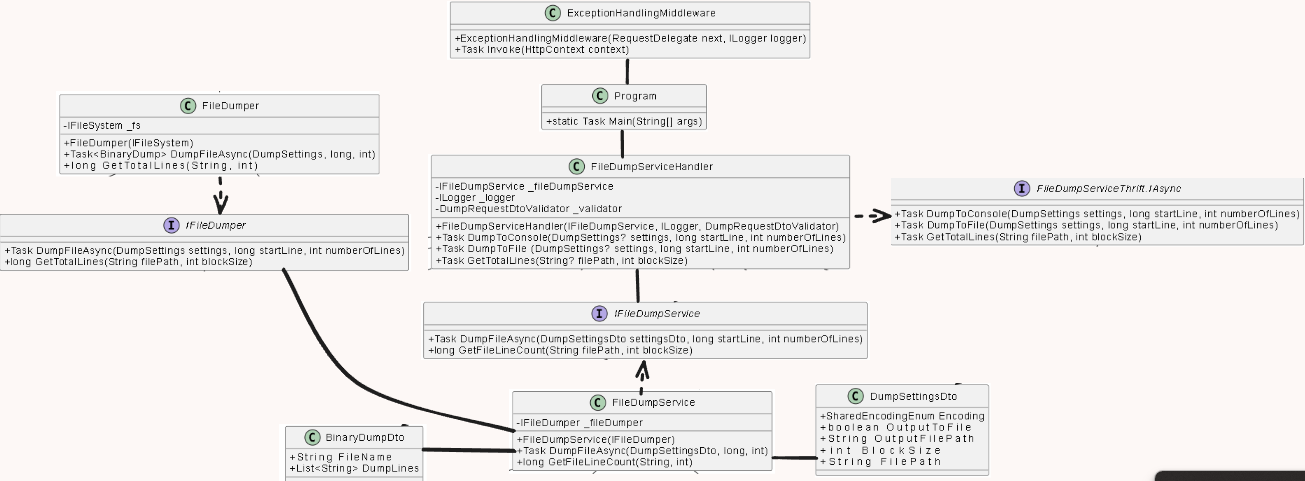


Рисунок 3 – Диаграмма классов

Описание основного класса **FileDumpServiceHandler**

**Поля:**

* \_fileDumpService – экземпляр IFileDumpService для выполнения бизнес–логики по созданию дампа.
* \_logger – экземпляр ILogger<FileDumpServiceHandler> для записи информационных и ошибочных событий.
* \_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.

## 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.

**Консольный клиент**

Консольное приложение реализовано в трёх 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.

**Веб–сервис**

Серверная часть выстроена по многослойной архитектуре «API – Application – Domain – Infrastructure». Входной API–уровень представлен Thrift – хендлером FileDumpServiceHandler, реализующим сгенерированный интерфейс FileDumpServiceThrift.IAsync. Этот класс берёт 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.

**Обмен данными и Thrift – IDL**

Для унификации взаимодействия определён 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 надёжная и производительная система, обеспечивающая как локальное интерактивное исследование бинарных данных, так и масштабируемый удалённый доступ через стандартизованный RPC–интерфейс.

# Список использованных источников

1. Компания ИнфоТеКС (электронный ресурс) – https://ru.wikipedia.org/wiki/ИнфоТеКС#cite\_note–1;
2. ИнфоТеКС (Infotecs) (электронный ресурс) – <https://www.tadviser.ru/index.php/Компания:ИнфоТеКС_(Infotecs)#.D0.A1.D1.82.D1.80.D1.83.D0.BA.D1.82.D1.83.D1.80.D0.B0_.D0.BA.D0.BE.D0.BC.D0.BF.D0.B0.D0.BD.D0.B8.D0.B8>;
3. Клиент–серверная архитектура (электронный ресурс) – [https://servergate.ru/articles/klient–servernaya–arkhitektura/](https://servergate.ru/articles/klient-servernaya-arkhitektura/);
4. Джим Арлоу. UML2 и Унифицированный процесс. Практический объектно–ориентированный анализ и проектирование/Джим Арлоу, Айла Нейштадт. – Санкт–Петербург, Издательство Символ–Плюс, 2007. – 624с.