Пензенский государственный университет

Кафедра «Математическое обеспечение и применение ЭВМ»

**ОТЧЕТ**

по производственной практике

обучающегося \_\_Каледы Романа Александровича \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

*(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 20 листов, 2 таблицы, 4 рисунка, 4 использованных источника.  С#, КЛИЕНТ-СЕРВЕРНОЕ ПРИЛОЖЕНИЕ, ОБЪЕКТНО-ОРИЕНТИРОВАННОЕ ПРОГРАММИРОВАНИЕ, REST API, HTTP, КОНСОЛЬ  Цель работы: разработка клиент-серверного приложения в виде сервиса, реализующего вывод данных переданного бинарного файла в шестнадцатеричном формате. Взаимодействие между сервисами организовано через http.  Технология разработки – язык программирования C#.  Результат работы – реализация приложения для выполнения вывода данных файла в особом формате. |

**Содержание**

[Введение 3](#_Toc202473129)

[1. Разработка клиент–серверного приложения на языке программирования С# ……………………………………………………………………………….. 5](#_Toc202473130)

[1.1. Анализ структурной организации предприятия 5](#_Toc202473131)

[1.2. Обзор задач, решаемых на предприятии 6](#_Toc202473132)

[1.3. Обзор используемого программного обеспечения 7](#_Toc202473133)

[1.4. Задачи, решаемые с использованием ПО на предприятии 9](#_Toc202473134)

[1.5.1. Постановка задачи 10](#_Toc202473135)

[1.5.2. Анализ требований 11](#_Toc202473136)

[1.5.3. Проектирование. Структура программного обеспечения 13](#_Toc202473137)

[1.5.4. Реализация программы 16](#_Toc202473138)

[Список использованных источников 20](#_Toc202473139)

# Введение

Летняя практика на предприятии предоставила возможность значительно расширить и углубить навыки в создании утилит и серверных приложений, организации и понимании клиент-серверного взаимодействия. В ходе практики осуществлялась разработка программных решений с использованием современных подходов – от обработки бинарных данных до построения клиент-серверной архитектуры.

Работа над проектами не только позволила отточить технические, но и показала, насколько важно тестировать написанный код, оптимизировать используемые подходы и подходить к настройке приложения с различных точек зрения и условий эксплуатации. Был сделан упор на то, чтобы новые модули без проблем могли встраиваться в существующую инфраструктуру и гарантировали стабильную и эффективную работу.

Практика также позволила вникнуть в процесс разработки в крупной IT-компании, понять как выстроено взаимодействие между командами и приобрести практический опыт в применении профессиональных инструментов и методологий. Полученный опыт стал весомым шагом в профессиональном развитии для будущей карьеры в сфере информационных технологий.

# Разработка клиент–серверного приложения на языке программирования С#

## Анализ структурной организации предприятия

АО «ИнфоТеКС» (Информационные Технологии и Коммуникационные Системы) – одна из ведущих отечественных компаний, которая проектирует и внедряет комплексные аппаратно-программные VPN-решения, а также разрабатывает средства криптографической защиты данных. Появившись на рынке 6 сентября 1991 года по инициативе группы специалистов под руководством Андрея Чапчаева, сегодня «ИнфоТеКС» стабильно входит в пятёрку крупнейших российских разработчиков в сфере информационной безопасности [1].

В состав российской части группы входят три ключевых подразделения:

**ОАО «ИнфоТеКС Интернет Траст» (2001)**

Занимается обслуживанием и поддержкой решений ViPNet для защиты каналов связи и организации электронного документооборота. Центр аккредитован как удостоверяющий и обеспечивает электронную отчётность перед ПФР, ФНС и другими ведомствами, регулярно занимая места в ТОП-10 крупнейших российских УЦ.

**ЗАО «Перспективный мониторинг» (2007)**

Проводит всесторонние аудиты ИТ-ландшафта, выявляет уязвимости и скрытые сервисы, помогает формировать или адаптировать корпоративные политики и регламенты по информационной безопасности в соответствии с текущими бизнес-задачами.

**Учебный центр ИнфоТеКС (2000)**

Во взаимодействии с профильными вузами готовит специалистов в области ИБ, сочетая лекции с практическими кейс-занятиями. Программы охватывают построение сквозных систем защиты и работу с решениями ViPNet на предприятиях.

Кроме того, в группу входят компании «Системы практической безопасности» и «СФБ лаборатория», которые специализируются на сертификации криптосредств и выполнении обязательных испытаний программно-аппаратных систем защиты по нормам ФСБ.

За годы работы «ИнфоТеКС» расширил своё присутствие и начал активное участие в международных выставках и конференциях, укрепляя позицию российского вендора на мировом рынке кибербезопасности.

## Обзор задач, решаемых на предприятии

«ИнфоТеКС» сегодня – ключевой российский разработчик и производитель передовых решений для комплексной защиты информации. Основу её ассортимента составляют продукты для надёжного шифрования и охраны корпоративных сетей, которые эволюционировали в целый набор инструментов против современных киберугроз. В портфеле компании есть системы обнаружения и нейтрализации атак, средства защиты серверов и рабочих станций от несанкционированного доступа, решения для безопасного обмена на мобильных устройствах и в промышленных автоматизированных системах (АСУ ТП, IIoT), а также собственные криптографические сервисы, библиотеки и модули для интеграции в сторонние приложения [2].

Ниже – краткий обзор самых востребованных продуктов «ИнфоТеКС»:

* ViPNet CSS Connect HW: cкрытый аппаратно-программный телефон с тач-дисплеем, через который пользователи ViPNet могут вести защищённые голосовые и видеоконференции по шифрованным каналам;
* ViPNet SafeBoot 3: cертифицированный ФСБ/ФСТЭК модуль доверенной загрузки нового поколения. Создаёт «точку доверия» на уровне UEFI BIOS и обеспечивает проверку всех компонентов платформы и операционной системы до старта;
* ViPNet Client: приложение для защиты рабочих мест корпоративных сотрудников – фильтрует весь сетевой трафик, шифрует передаваемые данные и гарантирует безопасную удалённую работу с ресурсами компании;
* ViPNet OSSL: крипто-надстройка над OpenSSL, дающая возможность использовать отечественные алгоритмы ГОСТ через привычный интерфейс OpenSSL без доработок стороннего ПО;
* ViPNet HSM: аппаратно-программный модуль, который выполняет все основные криптографические операции (генерация ключей, подпись, шифрование) по запросу любых прикладных сервисов;
* ViPNet Quantum Trusted System Lite: лёгкая квантово-криптографическая система для автоматической выработки и распределения ключей, которая даёт вашей СКЗИ «квантовую защиту» от перехвата будущими квантовыми компьютерами;
* ViPNet CryptoSmart: криптопровайдер для блокчейн-решений на базе Hyperledger Fabric. Встраивается в распределённые реестры и смарт-контракты, обеспечивая ГОСТ-шифрование транзакций и данных.

Эта линейка продуктов иллюстрирует, как «ИнфоТеКС» закрывает весь спектр задач по информационной безопасности: от надёжного шифрования каналов и платформ до защиты мобильных и промышленных систем. 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 представляет собой полнофункциональную интегрированную среду разработки, объединяющую множество инструментов для программистов:

- редактор кода с расширенными возможностями;

- компиляторы для различных языков;

- средства автоматической сборки проектов;

- интеграция с системами контроля версий;

- набор инструментов для создания графических интерфейсов.

С помощью Visual Studio можно создавать консольные утилиты, игры, приложения с GUI, а также веб-сайты, веб-приложения и сервисы. Поддерживаются платформы .NET Framework, .NET Core/.NET, MAUI, Xbox, .NET Compact Framework и Silverlight, что делает её универсальным решением для разработки разного рода ПО.

**Docker**

Docker представляет собой платформу контейнеризации, позволяющую упаковать приложение вместе со всеми зависимостями в лёгкий, переносимый образ. Такой контейнер запускается из этого образа и изолируется от хоста и других контейнеров средствами ядра Linux, что обеспечивает идентичность окружения на любых машинах. Для создания образов используется Dockerfile – текстовый сценарий, описывающий последовательность команд сборки, а готовые образы хранятся в реестрах (например, Docker Hub). Docker упрощает масштабирование, автоматизацию развёртывания и интеграцию с оркестраторами вроде Kubernetes или Docker Swarm.

**GIT**

Git – это распределённая система контроля версий, в которой каждый разработчик работает с полной копией репозитория, включая всю историю изменений. Изменения в коде фиксируются локально в коммитах, а затем синхронизируются с удалёнными серверами (GitHub, GitLab и др.) с помощью команд pull и push. Механизм ветвления позволяет легко изолировать новые возможности, исправления или эксперименты, после чего их можно объединить в основную ветку через merge или rebase. Благодаря скорости операций, гибкости рабочих процессов и широкому набору инструментов для код-ревью и CI/CD, Git стал стандартом управления исходным кодом в большинстве команд разработчиков.

**xUnit**

xUnit – это современный фреймворк для модульного тестирования приложений на платформе .NET. Основные особенности:

- простая и понятная атрибутивная модель для описания тестов;

- поддержка параметризованных (data-driven) тестов;

- встроенные механизмы фикстур для подготовки и очистки тестового окружения;

- интеграция с консолями сборки и CI/CD (Azure DevOps, GitHub Actions и др.);

- расширяемая архитектура: создание собственных атрибутов и расширений.

В сочетании с такими средами разработки, как Visual Studio или VS Code (через расширения), а также с системами сборки и контейнеризации (например, Docker), xUnit позволяет организовать надёжный процесс автоматического тестирования и обеспечить качество кода на всех этапах разработки.

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

## Задачи, решаемые с использованием ПО на предприятии

С помощью Microsoft Visual Studio обеспечивается полный цикл разработки .NET-приложений: от написания кода в интеллектуальном редакторе с подсветкой синтаксиса и автодополнением до глубокого профилирования, отладки и анализа производительности. Встроенные мастера и шаблоны позволяют быстро создать проекты для консольных утилит, десктопных GUI-приложений, веб-сервисов и мобильных клиентских решений на базе MAUI. Visual Studio тесно интегрируется с системами контроля версий, облачными сервисами Azure и популярными платформами CI/CD, что упрощает непрерывную поставку и развёртывание.

Git выступает основой современной командной работы над кодом: каждый разработчик получает полную локальную копию репозитория со всей историей изменений и может безопасно создавать ветки для новых фич или исправления багов. Быстрые операции commit, branch, merge и rebase, а также продвинутые рабочие процессы – от Gitflow до Trunk-Based Development –позволяют гибко организовать релизы и минимизировать конфликтность. Интеграция в Visual Studio и других IDE делает управление ветками и пулл-реквестами максимально наглядным, а автоматические хуки (pre-commit, pre-push) помогают поддерживать качество кода на каждом этапе.

Docker гарантирует, что среда разработки, тестирования и продакшена будет идентичной: приложение вместе со всеми библиотеками и настройками «упаковывается» в образ, из которого мгновенно создаётся контейнер. Описанный в Dockerfile процесс сборки обеспечивает воспроизводимость, а Docker Compose позволяет единым файлом описать многокомпонентный стэк (БД, кэш, бэкэнд, фронтэнд) и поднять его одной командой. Контейнеризация упрощает масштабирование, упрощает интеграцию с Kubernetes и другими оркестраторами, а также ускоряет CI/CD-пайплайны за счёт кеширования слоёв и параллельной сборки.

Для автоматизации модульного тестирования .NET-проектов используется xUnit.net – лёгкий, но мощный фреймворк с понятной атрибутивной моделью тестов. Поддержка параметризованных тестов позволяет легко пробегать один и тот же тест на разных наборах данных, а встроенные фикстуры (IClassFixture, ICollectionFixture) упрощают подготовку и очистку общего состояния. xUnit интегрируется с Visual Studio Test Explorer, консольными утилитами dotnet test и внешними CI-системами, что делает прогон тестов и сбор отчётов о покрытии частью каждого билда и помогает оперативно выявлять регрессии

* 1. **Решение задачи, поставленной руководителем предприятия**

### Постановка задачи

Задание включает разработку и доработку нескольких программных компонентов, а также их интеграцию в единую систему.

**Консольная утилита для вывода HexDump**

В качестве первого компонента разработана лёгкая консольная программа, которая порционно считывает любой бинарный файл и выводит его шестнадцатеричный дамп. Путь к файлу можно передать через аргументы командной строки или ввести интерактивно, а считывание по частям защищает от чрезмерного потребления памяти. Пользователь настраивает режим вывода (консоль или файл), количество байт в строке, отображение смещения и ASCII-представления. Для проверки корректного поведения покрытие кода обеспечивают модульные тесты на xUnit.

**Серверное приложение на ASP.NET Core**

Второй компонент – HTTP-сервер на базе ASP.NET Core, принимающий через GET-запрос параметры (путь к файлу, номер страницы, высоту окна, число байт в строке и кодировку) и возвращающий готовую «страницу» HexDump. Входные данные проверяются валидатором FluentValidation, чтение и форматирование фрагментов файла выполняет сервис HexDumpCreator, а кодировка текста – StringEncoder. Ошибки обработки запросов централизованно обрабатывает кастомное middleware, возвращая понятные JSON-ответы. Надёжность сервиса подтверждена Unit-тестами.

Все эти задачи были выполнены с использованием различных инструментов и технологий, включая Microsoft Visual Studio, Docker и GIT, что обеспечило эффективное и надежное решение поставленных задач.

### Анализ требований

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

**Консольная утилита для вывода HexDump**

**Функциональные требования:**

- возможность выбора кодировки и количества символов в строке для формирования и вывода HexDump;

- настройка способа вывода данных: поддержка вывода данных в файл или в консоль;

- избегание полного чтения содержимого файла в память: данные должны считываться и обрабатываться порциями (постранично);

- получение пути к файлу из командной строки: возможность передавать путь к файлу через аргументы командной строки;

- для каждой «страницы» дампа формируется HTTP-запрос к серверу с параметрами;

- навигация по блокам дампа с помощью стрелок (←, →, ↑, ↓), PageUp/PageDown, Home/End, Escape для выхода;

- автоматическое отслеживание и перерасчёт при изменении размеров окна.

**Нефункциональные требования:**

**Производительность**

- утилита должна обрабатывать файлы порционно и работать асинхронно, чтобы минимизировать задержки и нагрузку на память и процессор, особенно при сканировании больших файлов.

**Надёжность**

- все ошибки должны перехватываться и преобразовываться в понятные сообщения, без «падения» приложения с необработанными исключениями.

**Тестируемость**

- логика вынесена в отдельные сервисы и интерфейсы, что позволяет покрыть ключевые сценарии unit-тестами.

**Серверное приложение на ASP.NET Core**

**Функциональные требования:**

- использование эндпоинтов HTTP для взаимодействия с клиентом, который будет запрашивать порционно сформированный дамп;

- чтение данных из файла и их форматирование: сервер считывает данные из файла, формирует и записывает в файл или передает клиенту в консоль;

- поддержка конвертации дампа в запрошенную кодировку;

- интеграционные тесты функций: тесты для проверки взаимодействия и корректной работы функций сервера [3].

**Нефункциональные требования:**

- сервер должен обрабатывать запросы быстро и эффективно;

- обработка ошибок и устойчивость к сбоям;

- возможность проведения интеграционных тестов.

## Проектирование. Структура программного обеспечения

Диаграмма классов для консольного клиента представлена на рисунке 1 [4].

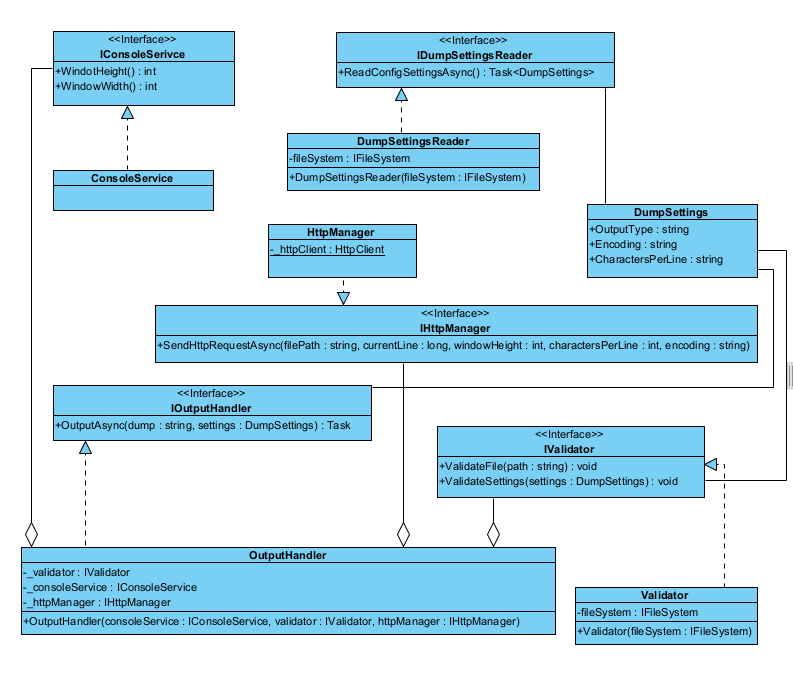


Рисунок 1 – Диаграмма классов для консольного клиента

**Описание класса OutputHandler**

**Поля:**

- \_validator – ссылка на экземпляр класса, реализующего интерфейс IValidator для валидации данных;

- \_consoleService – ссылка на экземпляр класса, реализующего интерфейс IConsoleService для возврата параметров консоли;

- \_httpManager – ссылка на экземпляр класса, реализующего интерфейс IHttpManager для отправки http-запросов.

**Методы:**

- OutputAsync – производит валидацию настроек и файла, выводит дамп и управляет пользовательским взаимодействием через клавиши.

**Описание класса Validator**

**Поля:**

- \_fileSystem – абстракция файловой системы.

**Методы:**

- ValidateSettings – производит валидацию количества символов в строке дампа с минимальным и максимальным значениями;

- ValidateFile – производит валидацию пути к файлу.

**Описание класса ConsoleService**

**Методы**

- WindowHeight – возвращает высоту консоли;

- WindowWidth – возвращает ширину консоли.

**Описание класса HttpManager**

**Поля:**

- \_httpClient – объект для взаимодействия с http-клиентом для отправки запросов на сервер.

**Методы:**

- SendHttpRequestAsync – отправляет http-запрос на серверную часть для получения страницы дампа.

**Описание класса DumpSettingsReader**

**Поля:**

- filesystem – абстракция файловой системы.

**Методы:**

- ReadConfigSettingsAsync – читает настройки из JSON файла.

Диаграмма классов для http-сервера на ASP .NET CORE представлена на рисунке 2.

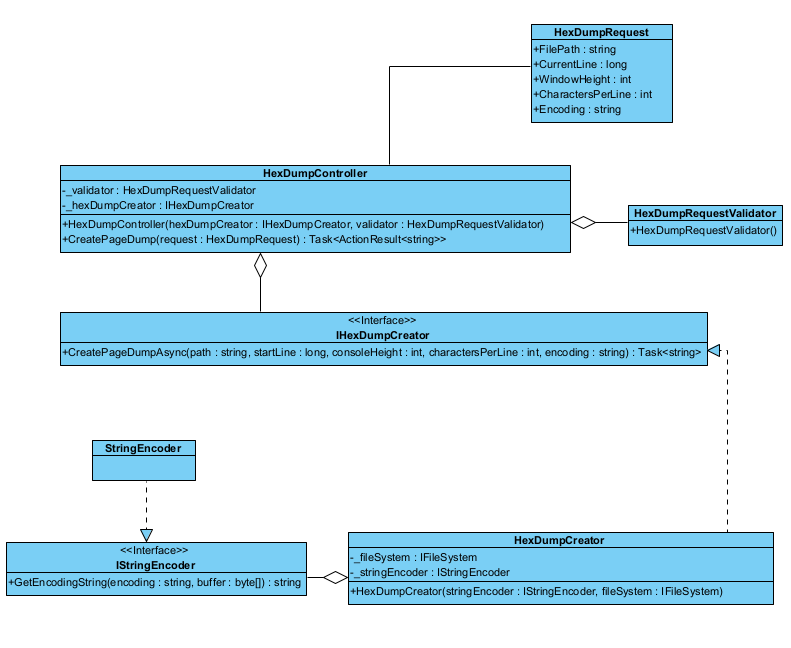


Рисунок 2 – Диаграмма классов для сервера

**Описание класса HexDumpController**

**Поля:**

- \_validator – валидатор полей запроса;

- \_hexDumpCreator – объект для реализации создания шестнадцатеричного дампа.

**Методы:**

- CreatePageDump – эндпоинт на http-get запрос на получение страницы дампа.

**Описание класса HexDumpRequest**

**Поля:**

- FilePath – путь к файлу;

- CurrentLine – текущая линия в файле;

- WindowHeight – высота окна консоли;

- CharactersPerLine – количество символов в строке дампа;

- Encoding – кодировка для вывода дампа.

**Описание класса StringEncoder**

**Методы:**

- GetEncodingString – метод возврата строки в нужной кодировке.

**Описание класса HexDumpCreator**

**Поля:**

- \_fileSystem – абстракция файловой системы;

- \_stringEncoder – ссылка на экземпляр класса, реализующий интерфейс для перевода строки в нужную кодировку.

**Методы:**

- CreatePageDumpAsync – метод для создания одной страницы дампа по входным данным.

## Реализация программы

В данном разделе подробно описывается процесс создания и интеграции двух основных компонентов системы – консольного клиента и http-сервера на базе ASP .NET Core. Диаграмма компонентов для консольного клиента приведена на рисунке 3.

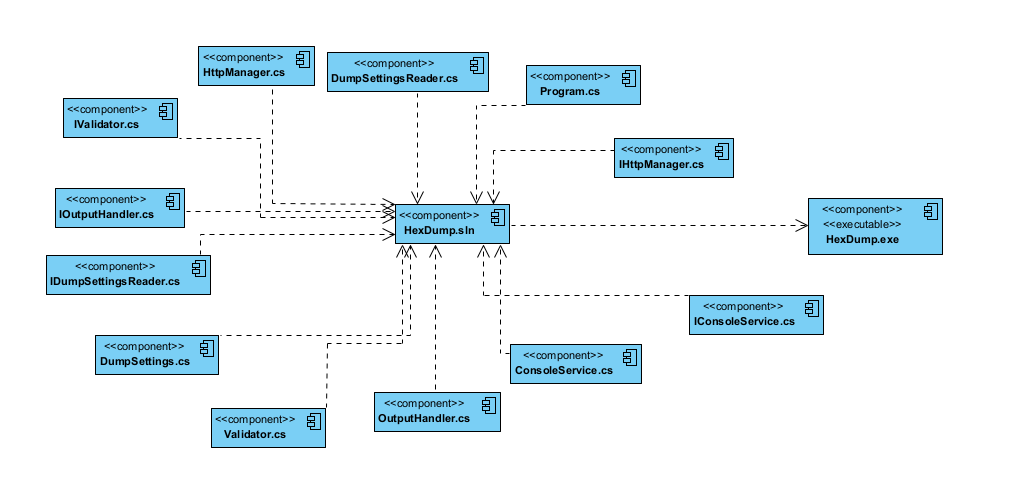


Рисунок 3 – Диаграмма компонентов консольного клиента

**Консольная утилита** HexDump построена по модульному принципу: при старте она загружает настройки из appsettings.json через DumpSettingsReader (или использует значения по умолчанию), затем определяет путь к обрабатываемому файлу – из аргументов командной строки, интерактивного ввода или из stdin. С помощью Validator проверяются существование и непустота файла, а диапазон параметра CharactersPerLine. Данные считываются и обрабатываются порциями через IFileSystem, что минимизирует расход памяти. Вывод организован в OutputHandler совместно с ConsoleService: в интерактивном режиме реализована постраничная навигация (стрелки, PageUp/PageDown, Home/End, Escape), горизонтальная прокрутка и автоматическая перерисовка при изменении размеров окна; при выборе режима «в файл» весь дамп асинхронно собирается частями и сохраняется в текстовый файл с отметкой времени. HTTP-запросы к серверу отправляет HttpManager, формируя GET-запрос с параметрами дампа, а все зависимости (файловая система, HTTP-клиент, консоль) инвертированы через интерфейсы и внедряются через DI-контейнер, что позволяет полноценно покрывать xUnit-тестами и гарантировать надёжность.

Описание компонентов для консольного клиента описано в таблице 1.

Таблица 1 – Описание компонентов консольного клиента

|  |  |
| --- | --- |
| Название компонента | Описание |
| HexDump.sln | Главный файл решения |
| HexDump.exe | Исполняемый файл |
| Program.cs | Файл, являющийся точкой входа |
| IConsoleService.cs | Интерфейс для реализации ConsoleService |
| IDumpSettingsReader.cs | Интерфейс для реализации DumpSettingsReader |
| IHttpManager.cs | Интерфейс для реализации HttpManager |
| IOutputHandler.cs | Интерфейс для реализации OutputHandler |
| IValidator.cs | Интерфейс для реализациия Validator |
| DumpSettings.cs | Модель настроек для дампа |
| ConsoleService.cs | Реализация интерфейса IConsoleService |
| DumpSettingsReader.cs | Реализация интерфейса IDumpSettingsReader |
| HttpManager.cs | Реализация интерфейса IHttpManager |
| OutputHandler.cs | Реализация интерфейса IOutputHandler |
| Validator.cs | Реализация интерфейса IValidator |

Диаграмма компонентов для http-сервера представлена на рисунке 4.

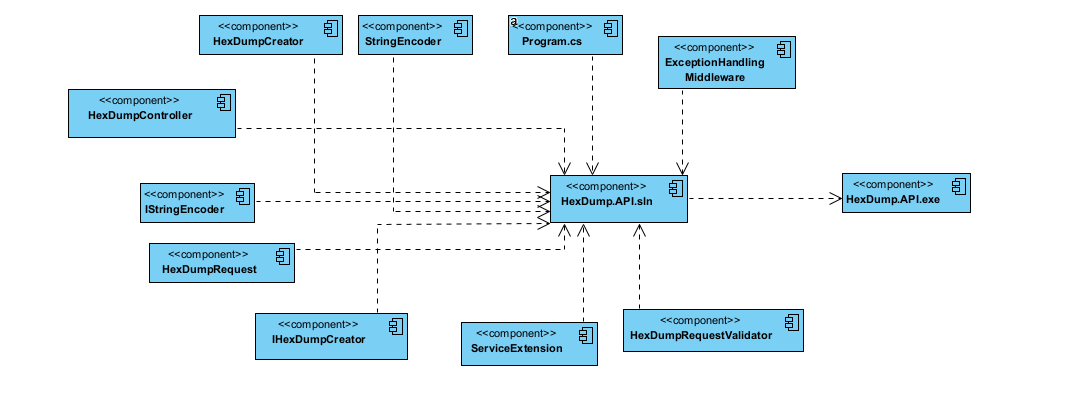


Рисунок 4 – Диаграмма компонентов серверного приложения

Описание компонентов серверного приложения описано в таблице 2.

Таблица 2 – Описание компонентов серверного приложения

|  |  |
| --- | --- |
| Название компонента | Описание |
| HexDump.API.sln | Главный файл решения |
| HexDump.API.exe | Исполняемый файл |
| Program.cs | Файл, являющийся точкой входа |

Продолжение таблицы 2

|  |  |
| --- | --- |
| HexDumpController.cs | Контроллер для приёма http-запросов |
| HexDumpRequest.cs | Модель для запроса на сервер |
| ServiceExtension.cs | Класс, предоставляющий расширения для DI |
| IHexDumpCreator.cs | Интерфейс для реализации HexDumpCreator |
| IStringEncoder.cs | Интерфейс для реализации StringEncoder |
| ExceptionHandlingMiddleware.cs | Middleware для отлова исключений |
| HexDumpCreator.cs | Реализация интерфейса IHexDumpCreator |
| StringEncoder.cs | Реализация интерфейса IStringEncoder |
| HexDumpRequestValidator.cs | Реализация класс валидатора |

**Серверная часть на ASP.NET Core** – лёгковесный HTTP-сервис с одним endpoint /api/HexDump, принимающим query-параметры FilePath, CurrentLine, WindowHeight, CharactersPerLine и Encoding, автоматически биндящимся в модель HexDumpRequest и проверяемым через FluentValidation. В контроллере HexDumpController при успешной валидации вызывается HexDumpCreator, который рассчитывает смещение (CurrentLine × CharactersPerLine), асинхронно читает из файла ровно нужный фрагмент через абстракцию IFileSystem и формирует текстовый дамп вида {offset:xN}: xx xx … | ASCII или выбранная кодировка | с помощью StringEncoder. Все исключения централизованно перехватывает ExceptionHandlingMiddleware: ошибки валидации отвечают 400 Bad Request с JSON-массивом ошибок, остальные – 500 Internal Server Error с подробным описанием. Регистрация зависимостей осуществляется в ServiceExtension, а в режиме разработки включён Swagger, что вместе с асинхронным I/O и слоистой архитектурой обеспечивает высокую производительность, простоту тестирования (unit- и интеграционные тесты через WebApplicationFactory) и лёгкость дальнейшего расширения.

# Список использованных источников

1. Компания ИнфоТеКС // Википедия [Электронный ресурс]. URL: https://ru.wikipedia.org/wiki/ИнфоТеКС#cite\_note-1 (дата обращения: 01.07.2025).
2. ИнфоТеКС (Infotecs) // TAdviser [Электронный ресурс]. URL: https://www.tadviser.ru/index.php/Компания:ИнфоТеКС\_(Infotecs)#.D0.A1.D1.82.D1.80.Д1.83.Д0.BA.Д1.82.Д1.83.Д1.80.Д0.B0\_.Д0.БA.Д0.ОЕ.Д0.МП.Д0.БА.НИ.Д0.БИ (дата обращения: 03.07.2025).
3. Клиент-серверная архитектура // ServerGate [Электронный ресурс]. URL: <https://servergate.ru/articles/klient-servernaya-arkhitektura/> (дата обращения: 04.07.2025).
4. Джим Арлоу. UML2 и Унифицированный процесс. Практический объектно-ориентированный анализ и проектирование/Джим Арлоу, Айла Нейштадт. – Санкт-Петербург, Издательство Символ-Плюс, 2007. – 624с