

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РФ
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»
ПОЛИТЕХНИЧЕСКИЙ ИНСТИТУТ

<i>Факультет</i>	<i>Кафедра</i>
вычислительной техники	МОиПЭВМ
Направление подготовки	09.03.04 «Программная инженерия»
Направленность	Программная инженерия

БАКАЛАВРСКАЯ РАБОТА

на тему

Автоматизированная система сервиса
облачного хранения данных

Студент	_____	Куприянов Максим Александрович
	(подпись, дата)	(ФИО полностью)
Руководитель	_____	Афонин А.Ю.
	(подпись, дата)	(фамилия, инициалы)
Нормоконтролер	_____	Такташкин Д.В.
	(подпись, дата)	(фамилия, инициалы)

Работа допущена к защите (протокол заседания кафедры от _____ № _____)

Заведующий кафедрой	_____	Козлов А.Ю.
	(подпись)	(фамилия, инициалы)

Работа защищена с отметкой _____ (протокол заседания ГЭК от _____ № _____)

Секретарь ГЭК	_____	Попова Н.А.
	(подпись)	(фамилия, инициалы)

Пенза, 2026

Федеральное государственное бюджетное образовательное учреждение
высшего образования
ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ПОЛИТЕХНИЧЕСКИЙ ИНСТИТУТ

Факультет
Вычислительной техники

Кафедра
МО и ПЭВМ

«Утверждаю»
Заведующий кафедрой
А.Ю. Козлов
«26» декабря 2025 г

**ЗАДАНИЕ
НА ВЫПУСКНУЮ КВАЛИФИКАЦИОННУЮ РАБОТУ
БАКАЛАВРА**

1. Студент Куприянов Максим Александрович гр. 22ВП2 факультета ВТ
<Фамилия, имя, отчество полностью>

2. Тема работы Автоматизированная система сервиса
облачного хранения данных

Тема утверждена приказом ПГУ № _____ от " _____ " _____ 20 _____

3. Руководитель работы к.т.н., доцент кафедры «МО и ПЭВМ» Афонин А.Ю.

4. Задание на работу (назначение разработки, исходные данные и т.п.)

1) *Назначение: серверная часть информационной системы безопасного обмена данными с функциями регистрации, авторизации и защищённой передачи файлов*

2) *Среда выполнения: Windows 11, Linux, Mac OS.*

Языковые средства разработки: C#, SQL, UML.

Инструментальные средства: JetBrains Rider, Swagger, Docker, PostgreSQL, MinIO, Draw.io, Prometheus, Grafana, YARP Reverse Proxy.

Технологии разработки: объектно-ориентированная парадигма, микросервисная архитектура, JWT-аутентификация, серверное шифрование AES-256-GCM, криптография с открытым ключом RSA.

3) *Основные функции приложения:*

а) регистрация и авторизация пользователей с выпуском JWT-токенов;

б) генерация и управление криптографическими ключами пользователей;

в) защищённая загрузка файлов с серверным шифрованием;

г) хранение зашифрованных файлов в объектном хранилище с изоляцией доступа;

<i>д) расшифрование и скачивание файлов только владельцем;</i>
<i>е) получение публичных ключей пользователей для организации обмена;</i>
<i>ж) централизованное журналирование событий безопасности и аудит действий пользователей;</i>
<i>з) администрирование учётных записей и управление ролями;</i>
<i>и) мониторинг состояния микросервисов и сбор метрик производительности.</i>

5. Перечень подлежащих разработке вопросов:

<i>1) Анализ предметной области и постановка задачи;</i>
<i>2) Проектирование;</i>
<i>3) Реализация;</i>
<i>4) Контроль качества программного обеспечения;</i>
<i>5) Оформление пояснительной записки.</i>

5. Календарный график выполнения работы

№ п/п	Наименование этапов работы	Объем работы	Срок выполнения	Подпись руководителя
1	Анализ предметной области и требований к системе	10%	07.04.2026 – 13.04.2026	
2	Проектирование программных средств	20%	14.04.2026 – 25.04.2026	
3	Реализация программных средств	30%	27.04.2026 – 11.05.2026	
4	Тестирование программных средств	20%	12.05.2026 – 21.05.2026	
5	Оформление пояснительной записки	20%	22.05.2026 – 02.06.2026	

Дата выдачи задания «26» декабря 2025г.

Руководитель бакалаврской работы _____
(подпись, дата)

Задание к исполнению принял студент _____
(подпись, дата)

Работу к защите допустить
Декан ФВТ _____ Л.Р. Фионова
(подпись, дата)

Реферат

Пояснительная записка содержит 119 листов, 22 рисунка, 34 таблицы, 20 использованных источников и 10 приложений.

МИКРОСЕРВИСНАЯ АРХИТЕКТУРА, БЕЗОПАСНАЯ ПЕРЕДАЧА ФИЛОВ, C#, ASP.NET CORE, ENTITY FRAMEWORK, JWT, AES-256-GCM, ШИФРОВАНИЕ, POSTGRESQL, MINIO, DOCKER, КЛИЕНТ-СЕРВЕРНОЕ ПРИЛОЖЕНИЕ, API GATEWAY, АУДИТ.

Объект исследования – сервисы обеспечения безопасной передачи конфиденциальных файлов между пользователями с гарантированным шифрованием данных при хранении и транспорте.

Цель выпускной квалификационной работы – спроектировать и разработать программное обеспечение, организующее безопасный обмен файлами с нескольких сторон: со стороны пользователя, загружающего данные, и со стороны получателя, скачивающего расшифрованный контент.

Технологии разработки – реализация программных средств для серверной части сервиса ведётся в среде разработки JetBrains Rider 2024 на языке программирования C#. Конструирование баз данных производится в СУБД PostgreSQL 16. Развёртывание приложения производится посредством контейнеризации в системе Docker на операционной системе Linux Ubuntu 22.04.

Среда исполнения – персональный компьютер пользователя.

Результаты работы – в ходе выполнения выпускной квалификационной работы была спроектирована и разработана серверная часть сервиса, моделирующего защищённую передачу файлов с применением шифрования AES-256-GCM и аутентификации JWT для всех участвующих сторон.

					ПГУ 09.03.04 – 08ВР222.18 ПЗ			
Изм.	Лист	№ докум.	Подпись	Дата	Автоматизированная система сервиса облачного хранения данных. Пояснительная записка	Лит.	Лист	Листов
Разраб.		Куприянов М.А.					4	119
Пров.		Афонин А.Ю.				Группа 22ВП2		
Реценз.								
Н. Контр.		Такташкин Д.В						
Утв.		Козлов А.Ю.						

Содержание

Введение.....	8
1 Анализ предметной области и постановка задачи на разработку автоматизированной системы сервиса облачного шифрования	10
1.1 Анализ предметной области облачных технологий и информационной безопасности.....	10
1.2 Анализ криптографических алгоритмов для облачного шифрования	10
1.2.1 Симметричные алгоритмы шифрования.....	10
1.2.2 Асимметричные алгоритмы шифрования.....	11
1.2.3 Криптографические хеш-функции	11
1.2.4 Выбор криптографических алгоритмов для системы.....	12
1.3 Анализ существующих решений облачного шифрования	12
1.4 Постановка задачи на разработку системы	18
1.4.1 Цели и задачи приложения	18
1.5 Требования к автоматизированной системе	19
1.5.1 Функциональные требования.....	19
1.5.2 Нефункциональные требования.....	25
2 Проектирование системы облачного шифрования.....	29
2.1 Проектирование архитектуры системы	29
2.2 Проектирование структур данных	32
2.2.1 Концептуальная модель баз данных.....	32
2.2.2 Логическая модель баз данных	34
2.3 Обобщённая структура разрабатываемого сервиса.....	36
2.4 Требования к аппаратному обеспечению серверной части.....	38
3 Разработка программного обеспечения системы безопасного облачного хранения файлов.....	39
3.1 Структура исходного кода и архитектура решения	39
3.2 Разработка физической модели базы данных	40
3.3 Реализация API Gateway.....	42
3.4 Реализация сервиса аутентификации.....	45

3.5	Реализация сервиса шифрования и передачи файлов	48
3.6	Реализация сервиса аудита.....	50
3.7	Реализация сервиса мониторинга.....	54
3.7.1	Presentation Layer (Слой представления)	55
3.7.2	Infrastructure Layer (Инфраструктурный слой).....	55
3.8	Реализация баз данных и хранилища.....	56
3.8.1	Реляционные базы данных PostgreSQL.....	56
3.8.2	Объектное хранилище MinIO	57
3.8.3	Prometheus и Grafana	57
3.9	Реализация пользовательского интерфейса	58
3.10	Развёртывание и запуск приложения.....	58
3.10.1	Dockerfile микросервисов	59
3.10.2	Конфигурация Docker Compose	59
3.10.3	Переменные окружения и сетевая конфигурация.....	60
3.10.4	Порядок запуска и проверка работоспособности	60
4	Контроль качества программного обеспечения.....	62
4.1	Обоснование выбора модели тестирования и метрик кода.....	62
4.2	Функциональное тестирование методом чёрного ящика	63
4.2.1	Функциональная диаграмма и таблица решений	63
4.2.2	Оценка сложности по метрикам Холстеда	67
4.2.3	Автоматизированное тестирование	69
4.2.4	Нагрузочное тестирование и оценка коэффициента замедления....	70
4.3	Планирование разработки и оценка бюджета.....	72
	Заключение	75
	Список использованных источников	76
	Приложение А (справочное) Глоссарий	79
	Приложение Б (рекомендуемое) Спецификация атрибутов таблиц	82
	Приложение В (обязательное) Объекты обращения к базам данных.....	85
	Приложение Г (обязательное) SQL-скрипты	88
	Приложение Д (обязательное) Структуры слоя представления данных.....	92

Приложение Е (обязательное) Структуры слоя бизнес-логики	96
Приложение Ж (обязательное) Структуры слоя обращения к базе данных	101
Приложение И (рекомендуемое) Листинг модульных тестов.....	106
Приложение К (рекомендуемое) Иллюстрационные материалы к тестированию API	115
Приложение Л (рекомендуемое) Таблица решений для сервиса файлов	118

Введение

В условиях глобальных изменений и увеличения числа атак на российские информационные системы, в последние годы популярность облачных сервисов информационной безопасности резко возросла [1]. Всё больше внимания и финансовые вложения направляются на защиту данных и развитие этой области.

Риски, связанные с киберугрозами, продолжают набирать обороты, становясь привлекательной целью для злоумышленников. Использование облачных хранилищ получает всё большее распространение, однако не все готовы вкладываться в дополнительную защиту данных [2]. Это приводит к утечкам логинов, паролей, а также личной и корпоративной информации, что влечёт за собой серьёзные финансовые потери для компаний [3].

Тем не менее, ни компании, ни пользователи не прекращают хранить данные в облаке – напротив, рынок облачных услуг демонстрирует устойчивый рост. Объём рынка облачных услуг в Российской Федерации вырос с 42,6 млрд рублей в 2017 году до 167,1 млрд рублей в 2023 году, показывая среднегодовой темп роста около 25% [4]. График роста объема рынка облачных услуг изображен на рисунке 1. Это указывает на то, что, несмотря на риски, облачные технологии остаются популярными, а потребность в защите информации только усиливается.

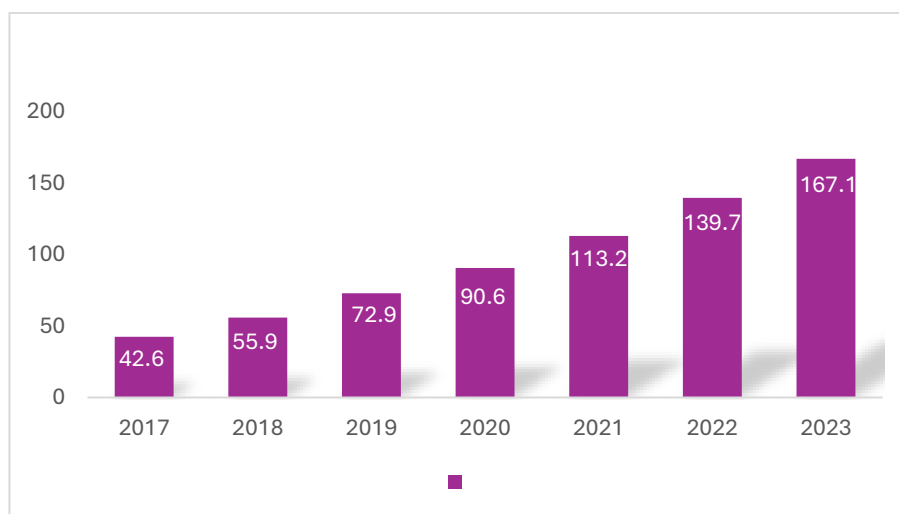


Рисунок 1 – Объём рынка облачных услуг в РФ 2017-2023, млрд.руб.

Одним из ключевых направлений обеспечения информационной безопасности является защита передаваемых файлов с использованием криптографических методов. Стандартные облачные решения зачастую не предоставляют пользователю контроля над процессом шифрования: данные защищаются на стороне провайдера, а не на стороне клиента, что оставляет возможность несанкционированного доступа. В связи с этим актуальной задачей является разработка специализированных систем, обеспечивающих сквозное шифрование данных непосредственно на стороне пользователя до их передачи в облачное хранилище.

Обеспечение безопасности данных в облачных хранилищах требует применения различных механизмов. Основные методы включают аутентификацию, авторизацию и управление доступом, а также шифрование данных [5]. При этом существующие решения облачного шифрования часто либо требуют сложной интеграции с корпоративной инфраструктурой, либо не обеспечивают должного уровня безопасности при передаче данных, либо не поддерживают современные микросервисные архитектуры [6].

Целью данной выпускной квалификационной работы является проектирование и разработка автоматизированной системы сервиса облачного шифрования, обеспечивающей защиту конфиденциальных данных пользователей при их передаче и хранении в облачных хранилищах.

Разработанная система может быть использована организациями для защиты конфиденциальных данных при их передаче в облачные хранилища. Применение микросервисной архитектуры обеспечивает масштабируемость, отказоустойчивость и возможность интеграции с различными облачными провайдерами.

1 Анализ предметной области и постановка задачи на разработку автоматизированной системы сервиса облачного шифрования

1.1 Анализ предметной области облачных технологий и информационной безопасности

Облачные технологии представляют собой модель предоставления вычислительных ресурсов, включая серверы, хранилища, базы данных, сетевые компоненты, программное обеспечение и аналитические инструменты, через интернет по требованию [7]. В отличие от традиционной модели владения инфраструктурой, облачные сервисы позволяют организациям использовать ресурсы сторонних провайдеров, оплачивая только фактическое потребление.

Согласно классификации NIST [7], существуют три основные модели облачных сервисов: Infrastructure as a Service (IaaS), Platform as a Service (PaaS) и Software as a Service (SaaS). Для хранения и обработки конфиденциальных данных наиболее критичной является модель IaaS, где пользователь получает контроль над виртуальной инфраструктурой, но физическая безопасность остаётся на стороне провайдера.

Основные угрозы безопасности данных в облачных средах включают несанкционированный доступ, утечку данных через ошибки конфигурации, атаки инсайдеров со стороны сотрудников провайдера, а также уязвимости в API и веб-интерфейсах [7]. По данным исследований, около 80% компаний испытывают опасения относительно безопасности данных при миграции в облако, при этом только 40% реализуют системы шифрования на уровне данных.

1.2 Анализ криптографических алгоритмов для облачного шифрования

1.2.1 Симметричные алгоритмы шифрования

Симметричное шифрование обеспечивает высокую производительность при шифровании больших объёмов данных. Алгоритм AES (Advanced Encryption Standard) является мировым стандартом симметричного шифрования,

утверждённым NIST в 2001 году [8]. AES поддерживает ключи длиной 128, 192 и 256 бит. Для облачного шифрования рекомендуется использование AES-256 в режиме GCM (Galois/Counter Mode) [8], который обеспечивает аутентифицированное шифрование с проверкой целостности данных (AEAD). Преимущества AES-256-GCM: высокая скорость благодаря параллелизму, встроенная аутентификация, отсутствие необходимости в отдельном MAC.

1.2.2 Асимметричные алгоритмы шифрования

Асимметричное шифрование применяется для защиты ключей симметричного шифрования и аутентификации участников обмена данными. В системе используется алгоритм RSA (Rivest-Shamir-Adleman) с длиной ключа 2048 бит для генерации пар ключей пользователей. Приватный ключ PKCS#8 [9] шифруется с помощью AES-GCM и мастер-ключа системы, публичный ключ экспортируется в формате PEM [9]. Алгоритм RSA обеспечивает надёжную защиту при обмене ключами сессии и цифровых подписей.

1.2.3 Криптографические хеш-функции

Хеш-функции применяются для проверки целостности данных, хранения паролей и генерации ключей. В системе используется SHA-256 (Secure Hash Algorithm 2) для вычисления хешей и деривации ключей из мастер-ключа системы. SHA-256 генерирует дайджест длиной 256 бит и используется для верификации целостности зашифрованных файлов.

Для хеширования паролей пользователей применяется алгоритм BCrypt с адаптивной стоимостью [10]. BCrypt является устойчивым к атакам методом перебора за счёт встроенной соли и настраиваемого параметра work factor (cost factor) [10], который определяет количество итераций хеширования. В отличие от простых хеш-функций типа SHA-256, BCrypt устойчив к атакам с использованием радужных таблиц и GPU-ускоренного перебора [10] паролей. Реализация BCrypt используется в AuthService для регистрации и аутентификации пользователей.

1.2.4 Выбор криптографических алгоритмов для системы

На основе анализа для разрабатываемой системы облачного шифрования выбран следующий набор криптографических алгоритмов: симметричное шифрование файлов – AES-256-GCM для обеспечения высокой производительности и аутентификации через GMAC; асимметричное шифрование ключей – RSA-2048 с шифрованием приватного ключа AES-GCM; хеширование паролей – BCrypt с адаптивной стоимостью; деривация ключей и проверка целостности – SHA-256. Данный набор алгоритмов соответствует современным требованиям информационной безопасности и обеспечивает баланс между уровнем защиты и производительностью системы.

1.3 Анализ существующих решений облачного шифрования

Проанализировав рынок альтернатив, схожих по функциональности на разрабатываемое в рамках выпускной квалификационной работы приложение, были выделены следующие программные средства для проведения сравнения: Cryptomator, Boxcryptor, NordLocker, VeraCrypt. В таблице 1 представлено краткое описание общих сведений о выделенных аналогах систем облачного шифрования.

Таблица 1 – Краткое описание общих сведений об аналогах

Наименование	Производитель	Описание
NordLocker	Nord Security	Решение для шифрования файлов от разработчиков NordVPN. Использует AES-256 и Argon2 для хеширования паролей. Предоставляет облачное хранилище с нулевым знанием и возможность локального шифрования
Boxcryptor	Secomba GmbH (приобретён Dropbox)	Коммерческое решение для шифрования данных в облачных сервисах с поддержкой корпоративных функций. Использует AES-256 и RSA, предоставляет управление группами пользователей и интеграцию с Active Directory
Cryptomator	Skymatic GmbH	Open-source решение для клиентского шифрования файлов перед загрузкой в облачные хранилища. Использует AES-256 и поддерживает работу с Dropbox, Google Drive, OneDrive и другими провайдерами. Обеспечивает прозрачное шифрование с виртуальным диском

Продолжение таблицы 1

Наименование	Производитель	Описание
VeraCrypt	IDRIX	Open-source программа для создания зашифрованных томов на локальных дисках. Использует AES, Serpent, Twofish. Не предназначена специально для облачных хранилищ, но может использоваться для шифрования перед загрузкой

Каждое из представленных приложений включает в себя различный функционал и возможности для взаимодействия с сервисом. Анализ и сравнение с аналогами будет включать в себя выделение плюсов и минусов каждого из выбранных программных средств.

Cryptomator – бесплатное open-source приложение для шифрования файлов перед загрузкой в облако. Разрабатывается с 2016 года и имеет активное сообщество пользователей. Графический интерфейс приложения представлен на рисунке 2.

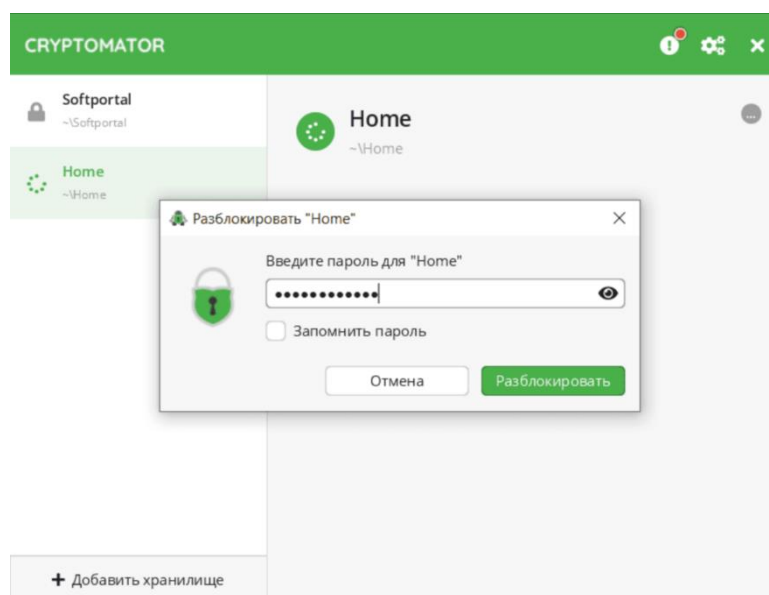


Рисунок 2 – Интерфейс приложения Cryptomator

К основным плюсам Cryptomator можно отнести:

- открытый исходный код и бесплатность использования;
- поддержка множества облачных провайдеров (Dropbox, Google Drive, OneDrive, WebDAV);
- использование проверенных криптографических алгоритмов (AES-256);
- кроссплатформенность (Windows, macOS, Linux, Android, iOS).

К минусам можно отнести:

- отсутствие централизованного управления ключами для корпоративного использования;
- нет встроенного облачного хранилища, только шифрование перед загрузкой;
- ограниченная масштабируемость для enterprise-решений.

Voxcryptor – коммерческое решение для шифрования данных в облаке, ориентированное как на частных пользователей, так и на корпоративный сектор. Приобретён компанией Dropbox в 2023 году. Графический интерфейс приложения представлен на рисунке 3.

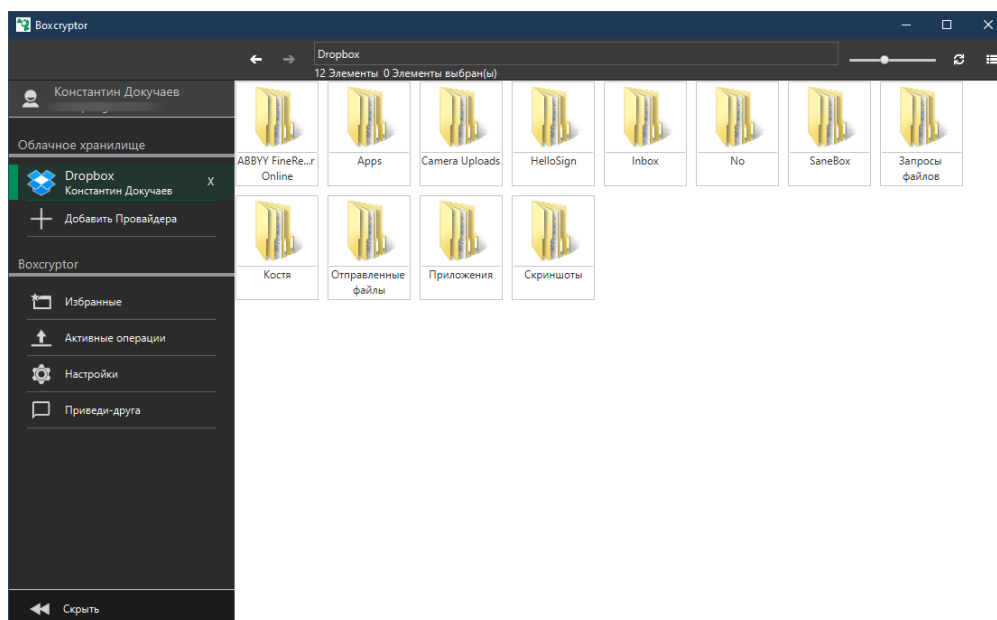


Рисунок 3 – Интерфейс приложения Voxcryptor

Основные преимущества Voxcryptor:

- корпоративные функции: управление группами, интеграция с Active Directory;
- поддержка более 30 облачных провайдеров;
- использование AES-256 и RSA для шифрования; – удобный интерфейс с виртуальным диском.

К основным недостаткам Voxcryptor можно отнести:

- закрытый исходный код;

- ограниченная поддержка российских облачных провайдеров;
- платная модель для корпоративного использования;
- неопределённость в развитии после приобретения Dropbox.

NordLocker – решение для шифрования файлов от разработчиков NordVPN. Предоставляет как облачное хранилище с нулевым знанием, так и возможность локального шифрования. Графический интерфейс приложения представлен на рисунке 4.

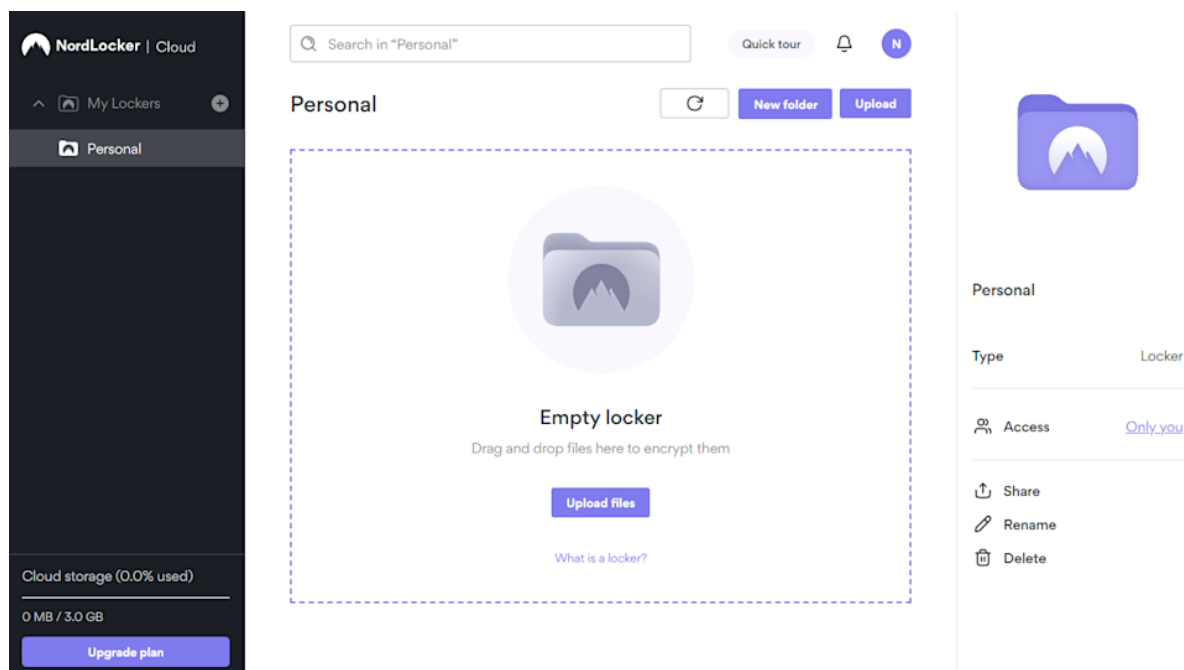


Рисунок 4 – Интерфейс приложения NordLocker

К основным преимуществам NordLocker можно отнести:

- современный интерфейс и простота использования;
- использование AES-256 и Argon2 для хеширования паролей;
- встроенное облачное хранилище с нулевым знанием;
- кроссплатформенность.

Основными недостатками NordLocker являются:

- закрытый исходный код;
- платная модель для больших объёмов хранения;
- зависимость от инфраструктуры Nord Security;
- нет корпоративных функций управления пользователями.

VeraCrypt – мощное open-source решение для создания зашифрованных томов. Развивается как наследник TrueCrypt с 2013 года. Графический интерфейс приложения представлен на рисунке 5.

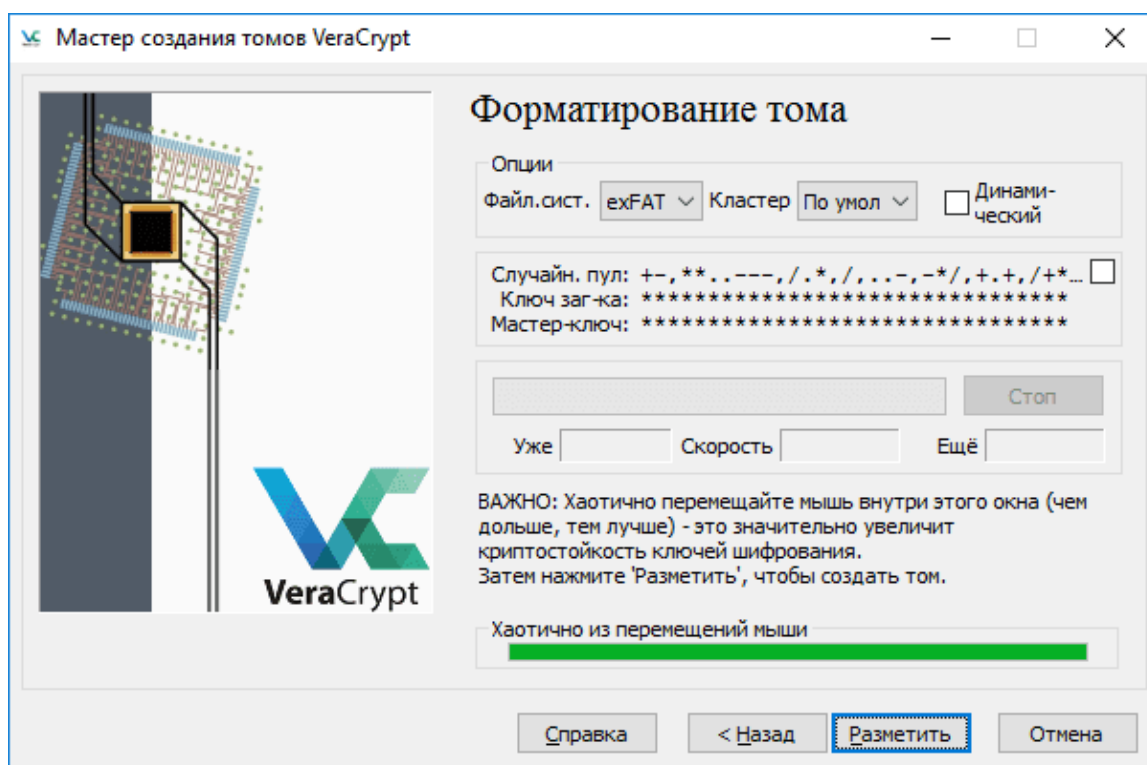


Рисунок 5 – Интерфейс приложения VeraCrypt

К основным преимуществам VeraCrypt можно отнести:

- открытый исходный код и бесплатность;
- использование множества криптографических алгоритмов (AES, Serpent, Twofish);
- создание скрытых томов и защита от атаки методом грубой силы;
- кроссплатформенность.

Основными недостатками VeraCrypt являются:

- не предназначена специально для облачных хранилищ;
- отсутствие API для интеграции с другими приложениями;
- устаревший интерфейс;
- нет централизованного управления ключами.

При сопоставлении существующих решений с проектируемой системой использована методика оценки по ряду выделенных критериев. Шкала

оценивания предполагает три градации: 0 баллов фиксируется при невыполнении требования, 1 балл – при удовлетворении требования не более чем наполовину, 2 балла – при полном соответствии заданному параметру.

Результаты анализа конкурирующих продуктов сведены в таблицу 2.

Таблица 2 – Сравнительный анализ аналогичных программных средств

Критерии	Cryptomator	Boxcryptor	NordLocker	VeraCrypt
Открытый исходный код	2	0	0	2
Поддержка российских облачных провайдеров	1	0	0	0
Централизованное управление ключами	0	2	0	0
Корпоративные функции, аудит	0	2	0	0
REST API для интеграции	0	1	0	0
Клиентское шифрование	2	2	2	2
Итого	5	7	2	4

Таким образом, по итогам сравнительного анализа можно сделать вывод о том, что рассмотренные программные средства предоставляют схожий базовый функционал шифрования и обладают как рядом преимуществ, так и недостатков. Однако ни один из аналогов в полной мере не соответствует поставленным для разрабатываемой системы критериям: поддержка российских облачных провайдеров, открытый исходный код, централизованное управление ключами и корпоративные функции. Поэтому необходимо разработать автоматизированную систему, которая будет предоставлять следующий функционал:

- прозрачное шифрование данных с использованием AES-256-GCM перед загрузкой в облако;
- централизованное управление ключами шифрования с разграничением доступа на основе ролей;
- поддержка российских облачных провайдеров (Яндекс Облако, Selectel, Cloud.ru);
- REST API для интеграции с корпоративными приложениями;
- журнала аудита всех операций с файлами и ключами шифрования для обеспечения контроля доступа и расследования инцидентов информационной безопасности.

- микросервисную архитектуру для обеспечения масштабируемости.

1.4 Постановка задачи на разработку системы

1.4.1 Цели и задачи приложения

Основной целью выпускной квалификационной работы является разработка серверной части программного обеспечения сервиса облачного шифрования с веб-интерфейсом управления. Основной его функцией должно быть шифрование файлов пользователей с использованием алгоритмов AES-256-GCM и RSA-2048, их безопасное хранение в объектном хранилище MinIO, а также предоставление удобного веб-интерфейса для загрузки, скачивания и администрирования файлов и пользователей.

К основным задачам, которые должны решаться при разработке приложения, можно отнести следующие:

- регистрация и авторизация пользователей с использованием алгоритма BCrypt для хеширования паролей и JWT-токенов для управления сессиями;
- генерация асимметричных ключевых пар RSA-2048 для каждого пользователя при регистрации и их защита с помощью мастер-ключа системы через AES-256-GCM;
- шифрование файлов алгоритмом AES-256-GCM с генерацией уникальных ключей шифрования для каждого файла;
- загрузка зашифрованных файлов в объектное хранилище MinIO и сохранение метаданных в PostgreSQL;
- расшифрование файлов с проверкой целостности через тег аутентификации GMAC при скачивании;
- реализация разграничения доступа на основе ролей – роли Пользователя и Администратора;
- администрирование пользователей: изменение ролей, блокировка и разблокировка пользователей с указанием причины;

- сбор метрик производительности с использованием OpenTelemetry и Prometheus;
- ведение аудит-журнала всех операций (регистрация, вход, загрузка, скачивание, блокировка) с сохранением в PostgreSQL и просмотром через админ-панель;
- предоставление REST API для всех операций через единую точку входа ApiGateway с использованием YARP reverse proxy;
- разработка веб-интерфейса на чистом JavaScript для взаимодействия с API: регистрация, вход, загрузка/скачивание файлов, администрирование;
- контейнеризация всех компонентов системы с помощью Docker и оркестрация через Docker Compose.

В результате выполнения этих задач необходимо получить систему, состоящую из микросервисов, работающих по технологии REST API, с единым веб-интерфейсом, базой данных PostgreSQL для хранения метаданных пользователей и файлов, объектным хранилищем MinIO для зашифрованных файлов, и возможностью развёртывания всей системы через Docker Compose.

1.5 Требования к автоматизированной системе

1.5.1 Функциональные требования

Диаграммы вариантов использования описывают взаимоотношения и зависимости между группами вариантов использования и действующих лиц, участвующими в процессе. Важно понимать, что диаграммы вариантов использования не предназначены для отображения проекта и не могут описывать внутреннее устройство системы [11]. Таким образом, диаграмма вариантов использования помогает верхнеуровнево отобразить связи между актерами и необходимыми сценариями для работы приложения. Подобные диаграммы говорят о том, что разрабатываемая система должна выполнять, а не как именно это будет реализовано.

Во время анализа функциональных требований системы облачного шифрования для выпускной квалификационной работы была разработана

диаграмма вариантов использования, отражающая следующие основные сценарии работы системы:

- зарегистрироваться в системе – создание учетной записи с указанием электронной почты и пароля;
- войти в систему – аутентификация пользователя для доступа к функционалу;
- выйти из системы – завершение текущего сеанса пользователя с прекращением доступа к функционалу системы до повторной аутентификации;
- загрузить файл с шифрованием – выбор файла, настройка параметров шифрования и сохранение в облаке;
- скачать файл с расшифрованием – получение ранее загруженного файла с автоматическим расшифрованием;
- просмотреть список файлов – отображение всех загруженных пользователем документов с метаданными;
- просмотреть список пользователей (администратор) – отображение перечня зарегистрированных пользователей с информацией об их ролях, статусе учетной записи и других основных данных;
- управлять пользователями (администратор) – создание, редактирование и удаление учетных записей;
- просматривать аудит-логи (администратор) – мониторинг действий пользователей в системе;
- изменить роль пользователя (администратор) – назначение или отзыв прав доступа;
- заблокировать/разблокировать пользователя (администратор) – временное ограничение доступа к системе;
- получить публичный ключ – экспорт криптографического ключа для обмена с другими участниками.

Диаграмма вариантов использования представлена на рисунке 6.

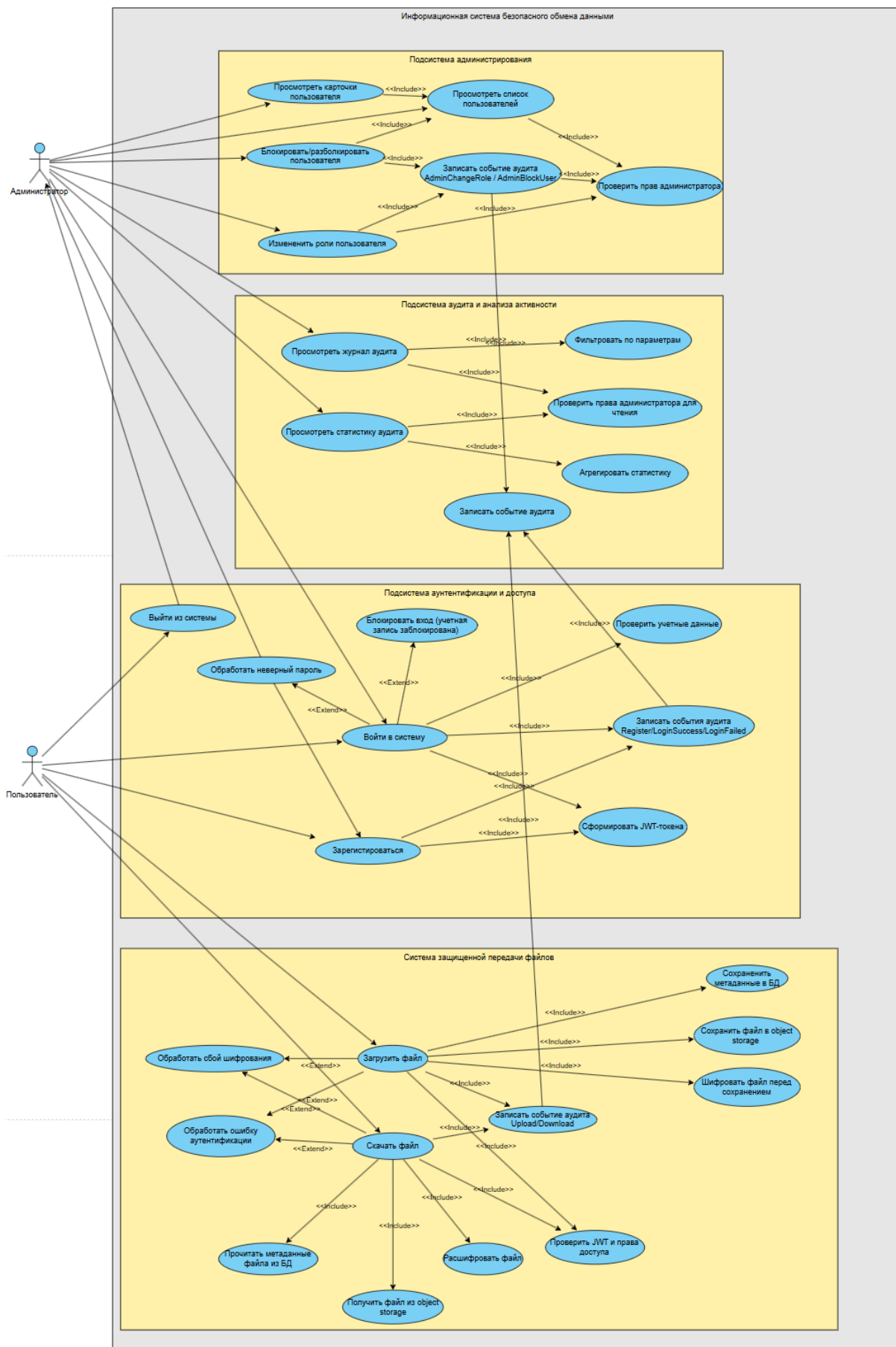


Рисунок 6 – Диаграмма вариантов использования сервиса облачного шифрования

В таблице 3 представлена варианта использования «Зарегистрироваться».

Таблица 3 – Спецификация варианта использования «Зарегистрироваться»

Поле	Значение
Прецедент	Зарегистрироваться в системе
ID	UC-01
Краткое описание	Пользователь создаёт новую учётную запись в системе с автоматической генерацией ключевой пары RSA
Главные актеры	Пользователь
Второстепенные актеры	Система шифрования (генерация ключей)
Предусловия	1. Пользователь не имеет учётной записи в системе 2. Email не занят другим пользователем
Основной поток	1. Пользователь вводит email и пароль в форму регистрации 2. Система хеширует пароль алгоритмом BCrypt 3. Система генерирует пару ключей RSA-2048 4. Приватный ключ шифруется мастер-ключом через AES-256-GCM 5. Учётная запись и ключи сохраняются в базе данных 6. Система возвращает JWT-токен для доступа
Постусловия	1. Пользователь зарегистрирован 2. Пользователю доступен функционал загрузки файлов 3. Создана запись в аудит-логе
Альтернативные потоки	1.a. Email уже занят – система возвращает ошибку конфликта 1.b. Пароль не соответствует требованиям безопасности

В таблице 4 представлена спецификация варианта использования «Войти в систему».

Таблица 4 – Спецификация варианта использования «Войти в систему»

Поле	Значение
Прецедент	Войти в систему
ID	UC-02
Краткое описание	Аутентификация пользователя с проверкой пароля через BCrypt и выдачей JWT-токена
Главные актеры	Пользователь, Администратор
Второстепенные актеры	Нет
Предусловия	1. Пользователь зарегистрирован в системе 2. Учётная запись не заблокирована
Основной поток	1. Пользователь вводит email и пароль 2. Система находит учётную запись по email 3. Система проверяет пароль через BCrypt.Verify 4. Система генерирует JWT-токен с ролью пользователя 5. Токен возвращается пользователю
Постусловия	1. Пользователь получил JWT-токен 2. Создана запись в аудит-логе об успешном входе
Альтернативные потоки	3.a. Пароль неверный – увеличивается счётчик неудачных попыток, создаётся запись в аудит-логе 2.a. Пользователь заблокирован – возвращается ошибка 403 с указанием причины

В таблице 5 представлена спецификация варианта использования «Загрузить файл».

Таблица 5 – Спецификация варианта использования «Загрузить файл»

Поле	Значение
Прецедент	Загрузить файл с шифрованием
ID	UC-03
Краткое описание	Пользователь загружает файл, который автоматически шифруется AES-256-GCM перед сохранением в хранилище
Главные актеры	Пользователь
Второстепенные актеры	Система шифрования, MinIO
Предусловия	1. Пользователь авторизован (валидный JWT) 2. У пользователя есть ключевая пара
Основной поток	1. Пользователь выбирает файл для загрузки 2. Система генерирует случайный ключ AES-256 3. Система шифрует файл через AES-256-GCM 4. Ключ AES шифруется публичным ключом пользователя 5. Зашифрованный файл сохраняется в MinIO 6. Метаданные сохраняются в PostgreSQL 7. Создаётся запись в аудит-логе
Постусловия	1. Файл зашифрован и сохранён 2. Пользователь получает ID загруженного файла 3. Доступ к файлу имеет только владелец
Альтернативные потоки	1.а. Файл не выбран – возвращается ошибка 400 5.а. Ошибка соединения с MinIO – возвращается ошибка 500

В таблице 6 представлена спецификация варианта использования «Скачать файл».

Таблица 6 – Спецификация варианта использования «Скачать файл»

Поле	Значение
Прецедент	Скачать файл с расшифрованием
ID	UC-04
Краткое описание	Пользователь скачивает свой ранее загруженный файл, система автоматически расшифровывает его перед отправкой
Главные актеры	Пользователь (владелец файла)
Второстепенные актеры	Система шифрования
Предусловия	1. Пользователь авторизован 2. Файл существует и принадлежит пользователю
Основной поток	1. Пользователь указывает ID файла 2. Система проверяет право доступа (владелец файла) 3. Система получает метаданные файла из PostgreSQL 4. Приватный ключ пользователя расшифровывается мастер-ключом 5. Ключ AES расшифровывается приватным ключом RSA 6. Файл загружается из MinIO 7. Система расшифровывает файл через AES-256-GCM с проверкой tag 8. Расшифрованный файл отправляется пользователю 9. Создаётся запись в аудит-логе

Продолжение таблицы 6

Поле	Значение
Постусловия	1. Пользователь получил расшифрованный файл
Альтернативные потоки	2.а. Файл не найден или доступ запрещён – возвращается 404 7.а. Неверный тег аутентификации – данные повреждены, возвращается ошибка

В таблице 7 представлена спецификация варианта использования «Просмотреть список пользователей» для администратора.

Таблица 7 – Спецификация варианта использования «Просмотреть список пользователей» (администратор)

Поле	Значение
Прецедент	Просмотреть список пользователей
ID	UC-05
Краткое описание	Администратор просматривает список пользователей с фильтрацией по роли, статусу блокировки и email
Главные актеры	Администратор
Второстепенные актеры	Нет
Предусловия	1. Администратор авторизован с ролью Admin
Основной поток	1. Администратор запрашивает список пользователей 2. Система возвращает данные с пагинацией 3. Применяются фильтры по запросу
Постусловия	1. Администратор получил список пользователей
Альтернативные потоки	Нет

В таблице 8 представлена спецификация варианта использования «Изменить роль пользователя» для администратора.

Таблица 8 – Спецификация варианта использования «Изменить роль пользователя» (администратор)

Поле	Значение
Прецедент	Изменить роль пользователя
ID	UC-06
Краткое описание	Администратор изменяет роль пользователя
Главные актеры	Администратор
Второстепенные актеры	Целевой пользователь
Предусловия	1. Администратор авторизован 2. Целевой пользователь существует
Основной поток	1. Администратор указывает ID пользователя и новую роль 2. Система обновляет роль в базе данных 3. Создаётся запись в аудит-логе с указанием ролей
Постусловия	1. Роль пользователя изменена 2. Создана запись аудита
Альтернативные потоки	2.а. Некорректная роль – возвращается ошибка 400

В таблице 9 представлена спецификация варианта использования «Заблокировать/разблокировать пользователя» для администратора.

Таблица 9 – Спецификация варианта использования «Заблокировать/разблокировать пользователя» (администратор)

Поле	Значение
Прецедент	Заблокировать/разблокировать пользователя
ID	UC-07
Краткое описание	Администратор блокирует или разблокирует учётную запись пользователя с указанием причины
Главные актеры	Администратор
Второстепенные актеры	Целевой пользователь
Предусловия	1. Администратор авторизован 2. Целевой пользователь существует
Основной поток	1. Администратор указывает ID, флаг блокировки и причину 2. Система обновляет статус блокировки, дату и причину 3. Создаётся запись в аудит-логе
Постусловия	1. Пользователь заблокирован/разблокирован 2. Заблокированный пользователь не может войти в систему
Альтернативные потоки	Нет

В таблице 10 представлена спецификация варианта использования «Просмотреть аудит-логи» для администратора.

Таблица 10 – Спецификация варианта использования «Просмотреть аудит-логи» (администратор)

Поле	Значение
Прецедент	Просмотреть аудит-логи
ID	UC-08
Краткое описание	Администратор просматривает журнал аудита с фильтрацией по дате, действию, результату и пользователю
Главные актеры	Администратор
Второстепенные актеры	Нет
Предусловия	1. Администратор авторизован с ролью Admin
Основной поток	1. Администратор запрашивает аудит-логи с фильтрами 2. Система возвращает список событий с пагинацией 3. Включаются: действие, время, пользователь, целевой ресурс, результат, IP, User-Agent
Постусловия	1. Администратор получил данные аудита
Альтернативные потоки	Нет

Таким образом, на основе проведенного анализа функциональных требований, был выведен минимальный набор разрабатываемых функций, необходимый для корректной работы всех бизнес-сценариев логистического сервиса.

1.5.2 Нефункциональные требования

При разработке автоматизированной системы сервиса облачного шифрования необходимо провести систематическую идентификацию и анализ

ключевых нефункциональных требований, определяющих архитектурные решения, способы интеграции компонентов и требования к эксплуатации системы. Для рассматриваемой системы выделены следующие основные нефункциональные требования.

Безопасность – система должна обеспечивать защиту API и административных функций от несанкционированного доступа, а также предотвращать компрометацию передаваемых и хранимых данных. Для обеспечения безопасности в проекте используется JWT-аутентификация с валидацией токенов на всех защищённых endpoints, а также серверная проверка прав доступа на основе ролей. Доступ к административным операциям ограничивается политикой AdminOnly, а все защищённые действия доступны только при наличии валидного токена. Критически важным элементом безопасности является аудит действий, позволяющий фиксировать события (успешные и неуспешные входы, административные операции, изменения ролей, блокировки/разблокировки) с сохранением IP-адреса, User-Agent и идентификатора корреляции для расследований и анализа инцидентов. Дополнительно учитываются типовые требования к защите API: корректная обработка ошибок авторизации, минимизация утечек информации в сообщениях об ошибках, а также строгое разделение прав между ролями Пользователя и Администратора.

Надёжность и отказоустойчивость – система должна сохранять работоспособность при частичных сбоях и обеспечивать предсказуемое поведение при ошибках внешних зависимостей (база данных PostgreSQL, объектное хранилище MinIO). В проекте применяется контейнеризация, а запуск сервисов организован через Docker Compose с учётом зависимостей и ожидания готовности PostgreSQL. Для операций, критичных к внешним ресурсам, предусмотрена обработка ошибок и возврат корректных HTTP-кодов ответа. При недоступности сервиса аудита события не приводят к остановке основных операций: запись аудита выполняется через отдельный компонент AuditPublisher с обработкой исключений, что позволяет продолжать работу даже при сбоях

логирования. Криптографические операции включают очистку чувствительных данных из памяти.

Производительность – система должна обеспечивать приемлемое время отклика для типовых сценариев: вход в систему, регистрация, шифрование и загрузка файлов, расшифрование и скачивание, административные операции. Производительность достигается за счёт разделения ответственности между микросервисами, а также за счёт фильтрации и постраничного вывода данных (пагинация с параметрами `page` и `pageSize`, ограничение выборки до 200 записей). Для операций шифрования используется аппаратное ускорение AES-NI через реализацию `AesGcm` в .NET, что обеспечивает высокую скорость обработки данных. Для административных списков минимизируется объём возвращаемых данных через проекции LINQ и асинхронную обработку запросов.

Масштабируемость и расширяемость – архитектура должна позволять развивать систему и увеличивать нагрузку без существенного изменения существующих модулей. В проекте используется микросервисный подход, при котором функциональные зоны разделены на самостоятельные сервисы. Такой подход облегчает добавление новых сценариев (расширение набора аудируемых действий, введение дополнительных ролей, развитие аналитики) и позволяет масштабировать наиболее нагруженные части независимо. Наличие API-шлюза упрощает унификацию входных точек, централизованное применение политик безопасности и последующее расширение маршрутов. Сбор метрик через OpenTelemetry и Prometheus обеспечивает наблюдаемость при росте нагрузки.

Совместимость и кроссплатформенность – клиентская часть должна работать на различных устройствах и в распространённых браузерах, а сервисы – разворачиваться в типовой инфраструктуре. В проекте используется веб-интерфейс на чистом JavaScript, доступный через любой современный браузер, что обеспечивает кроссплатформенный доступ без привязки к операционной системе. Серверная часть основана на ASP.NET Core и контейнеризована, что обеспечивает воспроизводимость окружения и упрощает развёртывание в разных средах (локальная разработка, тестовый стенд, рабочее окружение).

Интеграция между сервисами построена через HTTP/REST, что повышает переносимость и снижает связанность компонентов. Использование PostgreSQL и MinIO (S3-совместимое API) обеспечивает совместимость с распространёнными инфраструктурными решениями.

Удобство использования – интерфейс системы должен быть понятен пользователям и администраторам, минимизировать число действий в типовых сценариях и обеспечивать прозрачность результатов операций. Для пользовательских сценариев важны быстрая регистрация и вход, интуитивно понятная загрузка файлов с визуальной обратной связью, а также простое скачивание по идентификатору файла. Для административных сценариев предусмотрены фильтрация и поиск пользователей, управление ролями и блокировками через формы с валидацией, а также удобный просмотр аудит-логов с фильтрами по дате, действию и результату. Разделение интерфейса на вкладки и отображение админ-панели только при авторизации с ролью администратора снижает риск ошибок. Наглядное отображение статусов операций, причин блокировки и результатов повышает эффективность работы.

Наблюдаемость и контролируемость – система должна предоставлять средства контроля состояния и ключевых показателей для эксплуатации и анализа инцидентов. С этой целью реализованы технические средства наблюдаемости, включающие endpoint проверки работоспособности системы, endpoint предоставления метрик в формате Prometheus для интеграции с системами мониторинга. Наличие централизованного аудита в PostgreSQL дополняет наблюдаемость с точки зрения безопасности: журнал событий позволяет анализировать активность учётных записей, выявлять подозрительные последовательности действий и подтверждать факты выполнения административных операций. Статистика аудита доступна администраторам для анализа инцидентов.

2 Проектирование системы облачного шифрования

2.1 Проектирование архитектуры системы

После проведенного анализа функциональных и нефункциональных требований, поставленных для разрабатываемого программного средства, была выбрана архитектура на основе микросервисов как основа системы. Такая архитектура обладает умеренной сложностью при разработке и прекрасной гибкостью в выборе решения для написания различных частей сервиса, позволяя выбирать наиболее подходящее средство для обеспечения работы каждого необходимого для работы компонента. Все это делает микросервисную архитектуру отличным выбором для бизнес-приложений, где является необходимым разработать распределенную, но тем временем хорошо управляемую архитектуру. Каждый микросервис системы CloudSecure развёртывается независимо в отдельном Docker-контейнере и взаимодействует с остальными компонентами через чётко определённые API-интерфейсы.

Для лучшего понимания особенностей системы и того, какие компоненты системы необходимы для работы разрабатываемого приложения, была разработана диаграмма компонентов серверной части логистического сервиса. Диаграмма компонентов описывает особенности физического представления системы. Она позволяет определить архитектуру разрабатываемой системы, установив зависимости между программными компонентами, в роли которых могут быть как различные сервисы, написанные на различных языках программирования, так и базы данных, используемые для хранения данные приложений [12]. Диаграмма компонентов предоставляет общую картину архитектуры системы, а также помогает лучше понять ее структуру и взаимосвязи. Более того является удобным решением для документирования и коммуникации выбранных архитектурных решений при разработке приложений.

Диаграмма компонентов для серверной части разрабатываемого логистического сервиса представлена на рисунке 7.

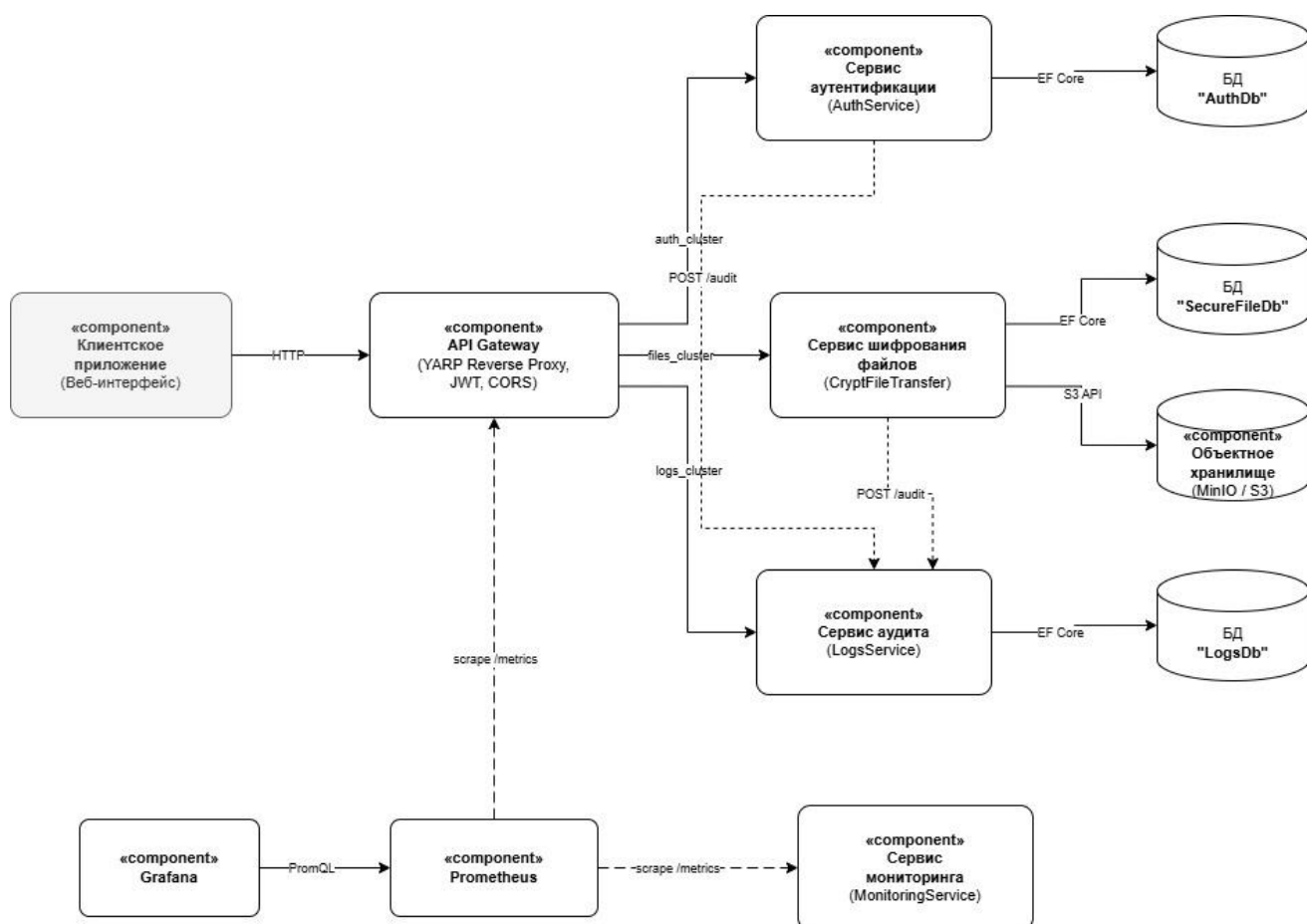


Рисунок 7 – Диаграмма компонентов

Каждый из представленных на диаграмме микросервисов отвечает за свою конкретную предметную область:

- API Gateway (сервис-шлюз) – единая точка входа в систему для всех клиентских запросов, выполняет маршрутизацию к downstream-сервисам через YARP Reverse Proxy, централизованную JWT-аутентификацию, CORS-политику и раздачу статических файлов веб-интерфейса;
- AuthService (сервис аутентификации) – регистрация и аутентификация пользователей с хешированием паролей через BCrypt, генерация и валидация JWT-токенов, управление криптографическими ключевыми парами RSA-2048, администрирование пользователей (изменение ролей, блокировка/разблокировка);
- MonitoringService (сервис мониторинга) – сбор и экспорт метрик приложения через OpenTelemetry с предоставлением endpoint для Prometheus;

- CryptFileTransfer (сервис шифрования файлов) – шифрование загружаемых файлов алгоритмом AES-256-GCM, расшифрование при скачивании с проверкой целостности через GMAC, хранение зашифрованных данных в объектном хранилище MinIO через S3-совместимый API;

- LogsService (сервис аудита) – централизованный сбор и хранение событий безопасности из всех микросервисов (регистрации, входы, операции с файлами, административные действия), предоставление API для просмотра журналов с фильтрацией и статистикой.

Такое разделение на предметные области позволяет максимально гибко взаимодействовать с системой, упрощая процессы модификации и масштабирования разработанного сервиса. Каждый микросервис может быть развёрнут, обновлён и масштабирован независимо от остальных.

Для организации взаимодействия между микросервисами внутри серверной части используется подход REST API. REST API – стиль разработки, обеспечивающий простоту и гибкость во всех взаимодействиях. Используется в системе для обмена данными в реальном времени, применяя следующие HTTP-методы для чёткого разграничения выполняемых операций:

- GET – для получения данных (списки пользователей, события аудита, метаданные файлов);

- POST – для создания ресурсов (регистрация, загрузка файлов, запись событий аудита);

- PATCH – для частичного обновления (изменение роли пользователя, блокировка/разблокировка);

- DELETE – для удаления ресурсов.

Публикация событий аудита из AuthService и CryptFileTransfer в LogsService выполняется асинхронно через компонент AuditPublisher с обработкой исключений. Это позволяет не останавливать выполнение основных операций (регистрация, вход, загрузка файлов) при временной недоступности сервиса аудита, обеспечивая отказоустойчивость системы.

Для хранения зашифрованных файлов используется объектное хранилище MinIO, совместимое с протоколом Amazon S3. Его использование позволяет отделить хранение больших бинарных объектов от реляционной базы данных PostgreSQL, в которой сохраняются только метаданные файлов (имя, размер, зашифрованный ключ, nonce, tag). Такой подход снижает нагрузку на основную базу данных и обеспечивает горизонтальную масштабируемость хранилища.

Для мониторинга состояния сервисов используется связка Prometheus и Grafana. Prometheus периодически собирает метрики с endpoints /metrics каждого микросервиса, а Grafana предоставляет визуализацию собранных данных для оперативного контроля работоспособности системы.

2.2 Проектирование структур данных

В данном разделе описывается проектирование структур данных системы. Рассматриваются концептуальная и логическая модели баз данных, определяются сущности, их атрибуты и связи между ними. Проектирование структур данных является основой для последующей реализации уровня хранения информации и обеспечивает корректное взаимодействие микросервисов системы с базами данных.

2.2.1 Концептуальная модель баз данных

Концептуальная модель – это модель, отображающая данные и связи предметной области с учетом их семантического содержания, независимо от представления в компьютере. Такая модель представления данных строится для отображения вида хранения данных в общем виде и лучшего понимания какие данные необходимы для хранения в системе [13].

Согласно всем определенным требованиям по организации хранения и управления данными для серверной части разрабатываемого логистического сервиса была разработана диаграмма типа «сущность-связь» концептуальной модели базы данных. Диаграмма представлена на рисунке 8.

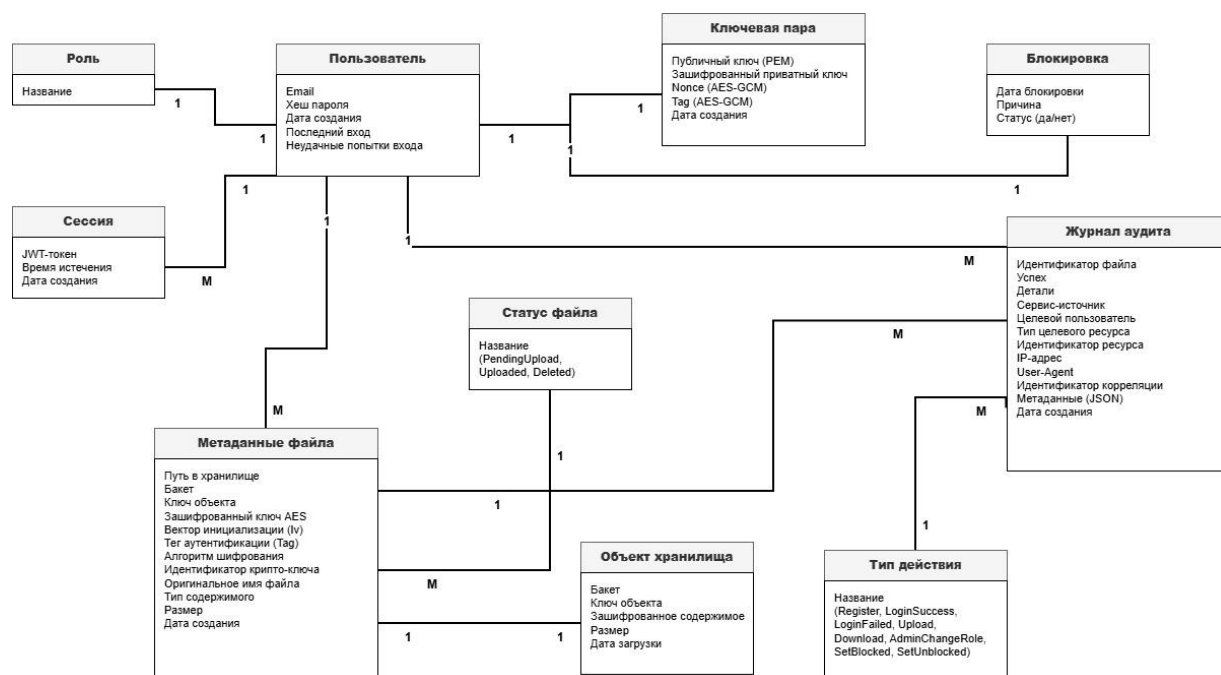


Рисунок 8 – Концептуальная модель БД

Диаграмма, разработанная в процессе проектирования концептуальной модели базы данных, наглядно отображает все сущности и связи между ними. В таблице 11 представлено описание сущностей, которые были выделены для организации работы системы безопасной передачи и хранения файлов.

Таблица 11 – Описание сущностей

Название сущности	Идентификатор	Назначение
Роль	role	Хранит сведения о доступных в системе ролях пользователей
Пользователь	user	Хранит сведения об имеющихся пользователях системы
Ключевая пара	user_key_pair	Хранит криптографические ключи пользователя для шифрования файлов
Блокировка	block	Хранит информацию о блокировке учётной записи пользователя
Сессия	session	Хранит сведения о выданных JWT-токенах авторизации
Метаданные файла	file_metadata	Хранит метаинформацию о загруженных зашифрованных файлах
Статус файла	file_status	Хранит перечень возможных состояний файла в системе
Объект хранилища	storage_object	Хранит информацию о физическом расположении зашифрованного файла в объектном хранилище
Журнал аудита	audit_log	Хранит записи обо всех действиях пользователей в системе
Тип действия	action_type	Хранит перечень возможных типов действий, фиксируемых в журнале аудита

В таблице 12 представлено описание всех возможных связей между выделенными сущностями.

Таблица 12 – Описание связей между сущностями

Название связи	Описание	Мощность
role-user	Определяет принадлежность роли к пользователю	1:1
user-user_key_pair	Определяет принадлежность криптографической ключевой пары к пользователю	1:1
user-block	Определяет принадлежность информации о блокировке к пользователю	1:1
user-session	Определяет принадлежность выданных сессий к пользователю	1:M
user-file_metadata	Определяет принадлежность загруженных файлов к пользователю	1:M
user-audit_log	Определяет принадлежность записей аудита к пользователю	1:M
file_metadata-file_status	Определяет принадлежность файла к текущему статусу	M:1
file_metadata-storage_object	Определяет соответствие метаданных файла и физического объекта в хранилище	1:1
file_metadata-audit_log	Определяет принадлежность записей аудита к файлу	1:M
audit_log-action_type	Определяет принадлежность записи аудита к типу действия	M:1

Данная концептуальная модель базы данных была разработана на основе нотации IDEF1X [14] в приложении draw.io.

2.2.2 Логическая модель баз данных

Логическая модель данных – это расширение концептуальной модели данных. Она включает в себя все сущности, атрибуты, ключи и взаимосвязи, которые представляют бизнес-информацию и определяют бизнес-правила [14]. Также эта модель является основой для физической модели базы данных.

Для разрабатываемой серверной части системы безопасного облачного хранения файлов CloudSecure было выделено 2 базы данных, разделённые по предметным областям: «Аутентификация и ключи» (AuthDb) и «Файловое хранилище и аудит» (SecureFileDb).

Такое разделение баз данных было выполнено по следующим причинам:

- снижение нагрузки на базу данных, распределяя запросы по нескольким экземплярам одновременно;

- возможность масштабировать базы данных и хранить резервные копии отдельно, что повышает отказоустойчивость системы;
- возможность масштабировать базы данных и хранить резервные копии отдельно, что повышает отказоустойчивость системы;
- при таком разделении можно управлять каждой предметной областью независимо от другой, обеспечивая изоляцию сервисов аутентификации и файлового обмена;
- обеспечение принципа наименьших привилегий – компрометация одной базы данных не приводит к утечке данных из другой.

Логическая модель базы данных пользователя представлена на рисунке 9.

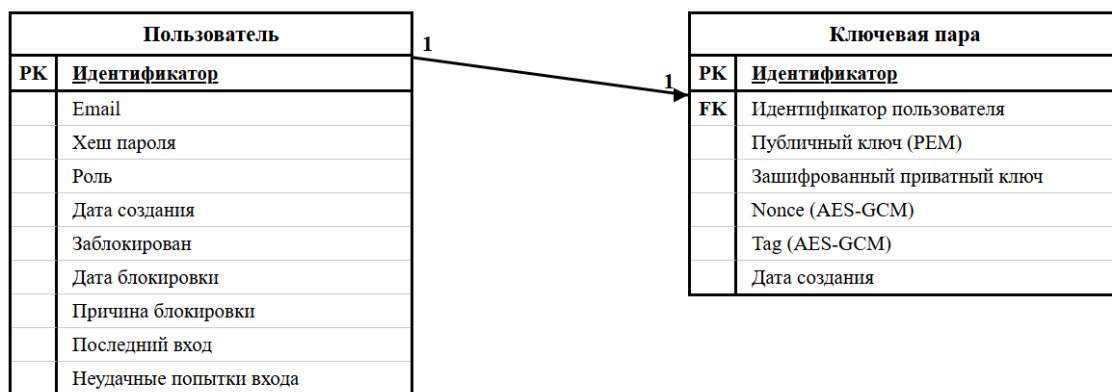


Рисунок 9 – Логическая модель БД «Пользователи»

Логическая модель базы данных метаданных файла представлена на рисунке 10.

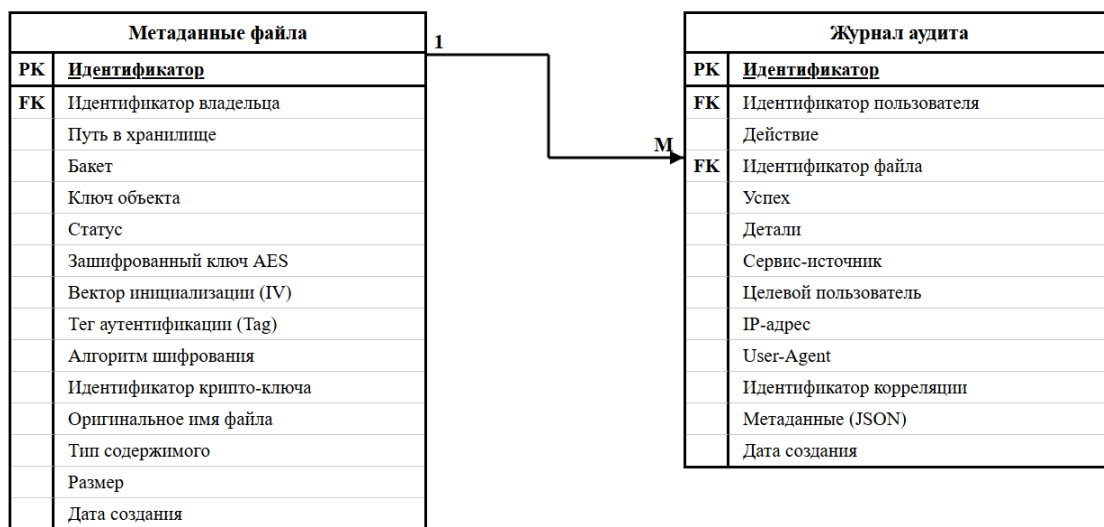


Рисунок 10 – Логическая модель БД «Метаданные файлов»

Разработанные логические модели баз данных для серверной части системы CloudSecure в дальнейшем будут использованы для разработки физической модели базы данных и её формирования.

2.3 Обобщённая структура разрабатываемого сервиса

В общем виде разрабатываемый сервис безопасного облачного хранения файлов CloudSecure представляет собой клиент-серверное приложение с микросервисной архитектурой. Клиент подразумевает под собой приложение, которое использует пользователь для взаимодействия со всем функционалом сервиса. В данном случае таким приложением выступает веб-браузер, установленный на персональном компьютере пользователя. Сервер, в свою очередь, представляет собой вычислительную среду, в которой хранится и исполняется разработанное приложение, развёрнутое в контейнерах Docker. Также в клиент-серверных приложениях используется отдельная машина (сервер), на которой отдельно хранятся все данные, необходимые для приложения – сервер баз данных. В системе CloudSecure используются два экземпляра PostgreSQL, а также объектное хранилище MinIO для хранения зашифрованных файлов.

Основные плюсы таких приложений:

- отсутствие дублирования кода: весь сервис хранится и исполняется на серверной стороне;
- минимальные требования к пользователю: от него требуется только наличие современного веб-браузера;
- безопасность: данные хранятся в базах данных и объектном хранилище, файлы шифруются алгоритмом AES-256-GCM перед сохранением, а пользователи не имеют прямого доступа к хранилищу.

Основные минусы подобных приложений:

- выход из строя сервера или базы данных: это сделает сервис недоступным для всех пользователей;

– перегрузка сервера: возможные DDoS-атаки могут привести к деградации производительности и замедлить работу приложения [15].

Архитектура приложения в общем виде представлена на диаграмме развёртывания на рисунке 11.

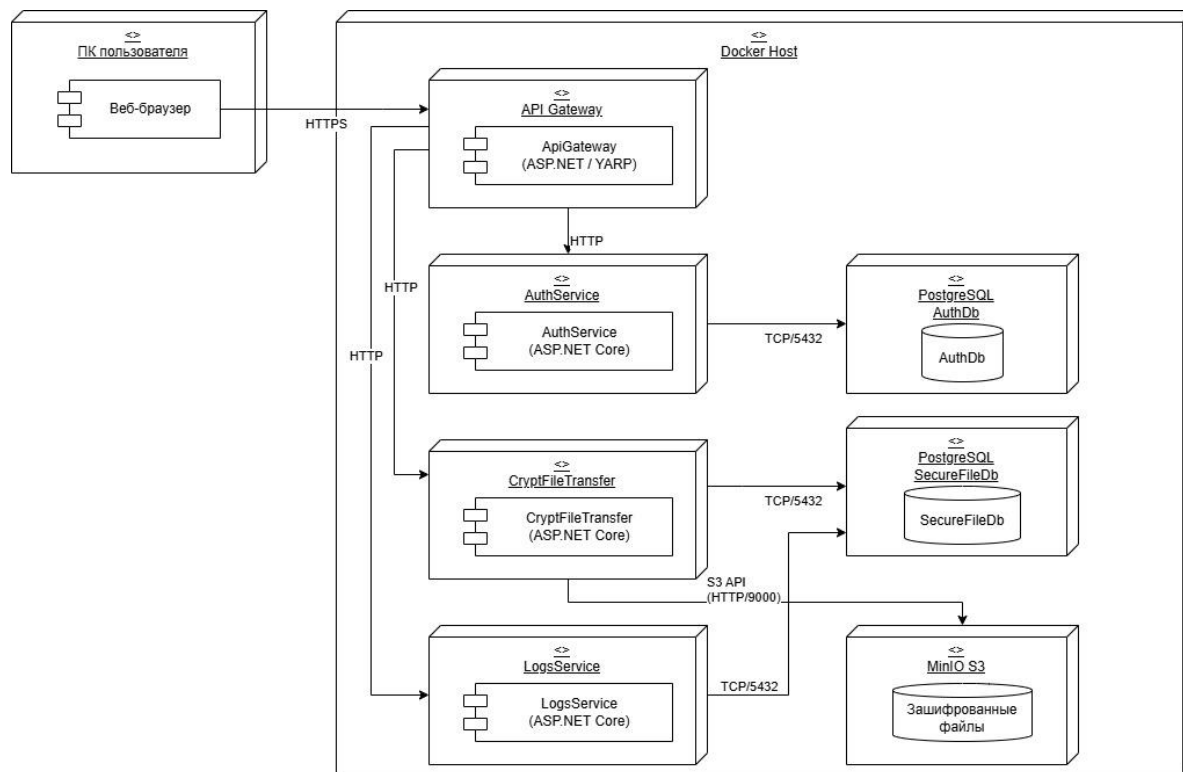


Рисунок 11 – Диаграмма развёртывания

На диаграмме отражены основные компоненты для работы системы CloudSecure: персональный компьютер пользователя, серверная среда Docker Host, на которой развёрнуты контейнеры микросервисов, серверы баз данных PostgreSQL и объектное хранилище MinIO.

Пользователь посредством выполнения HTTPS-запросов из веб-браузера обращается к API Gateway, который маршрутизирует запросы к соответствующим микросервисам для выполнения различных операций: аутентификации, загрузки и скачивания зашифрованных файлов, просмотра журнала аудита. Микросервисы, в свою очередь, при необходимости обращаются к серверам баз данных PostgreSQL, используя протокол TCP/5432, а также к объектному хранилищу MinIO по протоколу S3 API для сохранения и извлечения зашифрованных файлов.

Для обеспечения работы системы на сервере должен быть установлен Docker и Docker Compose для возможности развёртывания всех микросервисов, баз данных и объектного хранилища с помощью инструкций, описанных в Dockerfile и docker-compose файлах.

2.4 Требования к аппаратному обеспечению серверной части

Для обеспечения корректной работы серверной части разрабатываемой системы CloudSecure были выделены минимальные и рекомендуемые требования к техническому оснащению программных средств, на которых будет запускаться приложение. Данные требования описаны в таблице 12.

Таблица 13 – Требования к программному обеспечению

Характеристика	Минимальные требования	Рекомендуемые требования
Процессор	x64 с 2 ГГц, 2 ядра	x64 с 3 ГГц, 4 ядра и более
Оперативная память	4 Гб	8 Гб и более
Тип и объём жёсткого диска	HDD со 100 Гб	SSD с 500 Гб и более
Пропускная способность сети	100 Мбит/сек	1 Гбит/сек
Операционная система	Ubuntu 20.04 / Windows Server 2019	Ubuntu 22.04 и выше
Обязательное ПО	Docker 20.10, Docker Compose 2.0	Docker 24.0, Docker Compose 2.20 и выше

Приведённые требования отражают минимально необходимую конфигурацию для запуска всех микросервисов системы в контейнерном окружении Docker. Соблюдение рекомендуемых характеристик обеспечивает стабильную производительность при одновременной работе нескольких сервисов, хранении зашифрованных файлов и обработке запросов в условиях реальной нагрузки.

3 Разработка программного обеспечения системы безопасного облачного хранения файлов

3.1 Структура исходного кода и архитектура решения

Исходный код системы CloudSecure организован в виде единого Solution, содержащего пять проектов ASP.NET Core, каждый из которых реализует отдельный микросервис с изолированной зоной ответственности. Структура каталогов представлена ниже:

- src/ApiGateway – единая точка входа, YARP reverse proxy, статический веб-интерфейс;
- src/AuthService – регистрация, аутентификация, управление пользователями и ключами;
- src/CryptFileTransfer – шифрование, загрузка и скачивание файлов через MinIO;
- src/LogService – запись и хранение событий аудита;
- src/MonitoringService – экспорт метрик для Prometheus.

Все проекты нацелены на .NET 8 (TargetFramework: net8.0) и используют минимальный хостинг (top-level statements в Program.cs). Взаимодействие между сервисами осуществляется по HTTP через внутреннюю Docker-сеть; внешний трафик проходит исключительно через ApiGateway.

Основные технологии и фреймворки, применяемые в проекте, перечислены в таблице 14.

Таблица 14 – Основные технологии и NuGet-зависимости проекта

Технология	Версия	Назначение
ASP.NET Core	8.0	Веб-фреймворк для всех микросервисов
Entity Framework Core	8.0	ORM, миграции БД (AuthService, LogService)
Npgsql.EntityFrameworkCore.PostgreSQL	8.0	Провайдер PostgreSQL для EF Core
YARP (Reverse Proxy)	2.1	Маршрутизация запросов в ApiGateway
BCrypt.Net-Next	4.0	Хеширование паролей в AuthService

Продолжение таблицы 14

Технология	Версия	Назначение
Minio SDK	6.0	Взаимодействие с объектным хранилищем MinIO
Minio SDK	6.0	Взаимодействие с объектным хранилищем MinIO
OpenTelemetry	1.9	Сбор и экспорт метрик (все сервисы)
OpenTelemetry.Exporter.Prometheus	1.13-beta	Экспорт метрик в формате Prometheus
Swashbuckle (Swagger)	6.6	Документирование API в режиме разработки
Docker / Docker Compose	3.9	Контейнеризация и оркестрация

Перечисленные зависимости обеспечивают реализацию всех функциональных требований: безопасное хранение файлов (AES-256-GCM, RSA-2048), аутентификацию (JWT), аудит действий и наблюдаемость системы.

3.2 Разработка физической модели базы данных

Физическая модель базы данных – финальный этап моделирования базы данных, выполняемый на основе логической модели. Эта модель включает все необходимые таблицы, столбцы, связи и свойства баз данных для физической реализации. Производительность базы данных, стратегия индексации, физическое хранилище и денормализация – важные параметры физической модели [16]. Физическая модель разрабатывается уже для конкретной СУБД, выбранной для программного обеспечения.

В рамках разрабатываемой системы защищённого обмена файлами (CloudSecure) реализована микросервисная архитектура, где каждый сервис использует собственную базу данных. В качестве системы управления базами данных выбрана PostgreSQL – реляционная СУБД с открытым исходным кодом, обеспечивающая надёжность, масштабируемость и поддержку сложных типов данных.

Выделены следующие базы данных:

- AuthDb – база данных микросервиса авторизации, содержит таблицы пользователей (Users) и ключевых пар RSA (UserKeyPairs);

– SecureFileDb – база данных микросервиса обработки файлов, содержит таблицы метаданных файлов (FileMetadata) и журнала аудита (AuditLogs).

На рисунке 12 представлена физическая модель базы данных микросервиса авторизации.

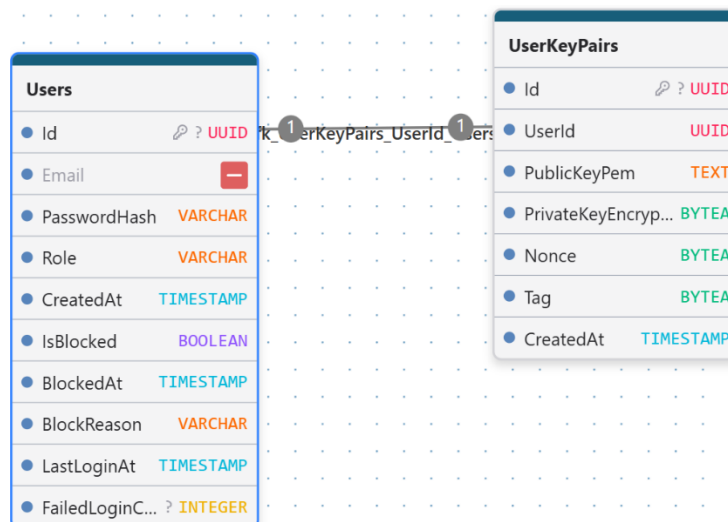


Рисунок 12 – Физическая модель базы данных микросервиса авторизации

На рисунке 13 представлена физическая модель базы данных микросервиса шифрования и аудита.

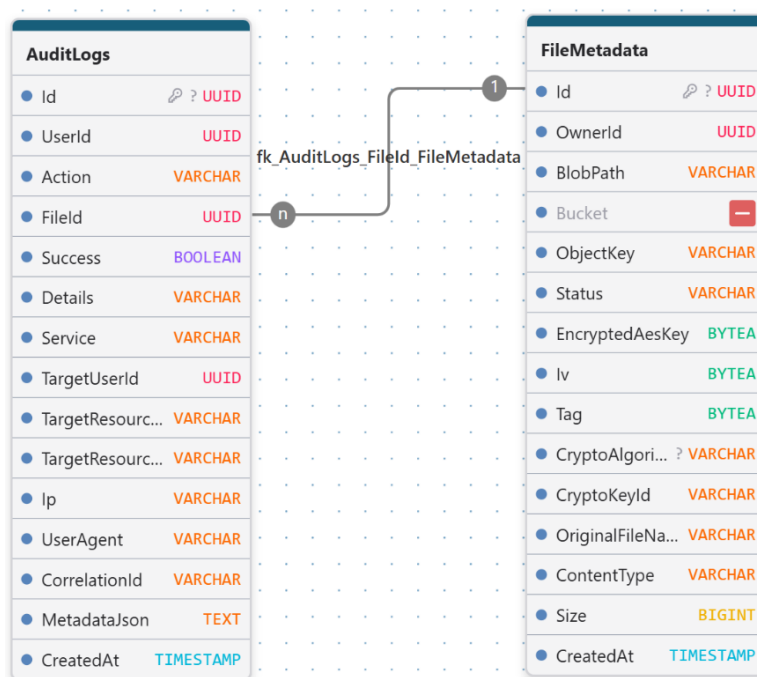


Рисунок 13 – Физическая модель базы данных микросервиса шифрования и аудита

В таблице 15 отражено описание всех сущностей баз данных системы защищённого обмена файлами CloudSecure.

Таблица 15 – Описание сущностей баз данных

№ п/п	Наименование	Описание	Категория	База данных
1	Users	Список всех пользователей системы	Стержневая	AuthDb
2	UserKeyPairs	Ключевые пары RSA пользователей (публичный и зашифрованный приватный ключ)	Характеристическая	AuthDb
3	FileMetadata	Метаданные файлов (путь, шифрование, статус, размер)	Стержневая	SecureFileDb
4	AuditLogs	Журнал аудита операций (загрузка, скачивание, вход, ошибки)	Стержневая	SecureFileDb

Спецификации атрибутов всех сущностей таблиц баз данных представлены в приложении Б.

Структуры данных, используемые в микросервисах в слоях обращения к базам данных, описаны в приложении В.

SQL-скрипты для создания таблиц представлены в приложении Г.

3.3 Реализация API Gateway

Микросервис ApiGateway реализует единую точку входа в систему и выполняет функции маршрутизации запросов к остальным микросервисам, применения механизмов безопасности и предоставления вспомогательных сервисных конечных точек. В проекте он построен на ASP.NET Core 8 и использует:

- YARP Reverse Proxy для проксирования маршрутов на AuthService, CryptFileTransfer и LogsService;
- контроллер AuthProxyController для проксирования запросов регистрации и входа в AuthService через IHttpClientFactory;
- JWT-аутентификацию и политики авторизации (UserOnly, AdminOnly) для централизованного ограничения доступа на уровне шлюза;

- OpenTelemetry + Prometheus exporter для экспонирования метрик на /metrics.

- статический веб-интерфейс (index.html, styles.css, app.js), раздаваемый через UseStaticFiles/UseDefaultFiles;

Для микросервиса можно выделить следующие логические слои:

Presentation Layer (Слой представления). Содержит конечные точки, принимающие HTTP-запросы:

- маршруты, проксируемые через YARP: /auth/*, /files/*, /logs/*, /admin/*, /audit/*;
- контроллер AuthProxyController: POST /auth/register, POST /auth/login
- прямое проксирование запросов к AuthService через HttpClient;
- минимальные endpoints, заданные в Program.cs: /health (проверка состояния), /test (тестовый), /metrics (метрики Prometheus).

Infrastructure Layer (Инфраструктурный слой). Содержит конфигурацию и подключение инфраструктуры:

- YARP Reverse Proxy (AddReverseProxy().LoadFromConfig(...), MapReverseProxy()), маршруты и кластеры определяются в appsettings.json;
- CORS-политика «frontend» для взаимодействия с веб-клиентом (WithOrigins, AllowAnyHeader, AllowAnyMethod);
- HttpClient-фабрика для AuthService с базовым адресом http://authservice:8080;
- OpenTelemetry metrics + Prometheus exporter (MapPrometheusScrapingEndpoint()).

Security Layer (Слой безопасности). Реализован через middleware и конфигурацию:

- AddAuthentication().AddJwtBearer(...) с валидацией Issuer, Audience, IssuerSigningKey и Lifetime;
- AddAuthorization() с политиками: UserOnly (роли User, Admin); AdminOnly (роль Admin);

– YARP-маршруты /admin/* защищены политикой AdminOnly, маршрут /audit/* (GET) защищён политикой AdminOnly, маршрут /audit/* (POST) открыт для записи аудита.

Структура микросервиса представлена на рисунке 14.

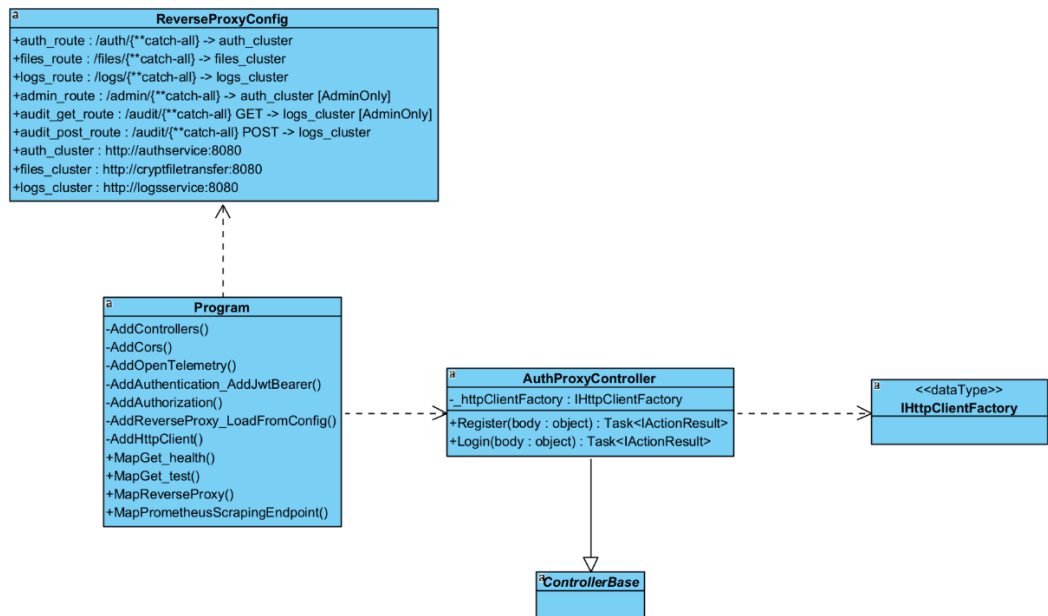


Рисунок 14 – Диаграмма классов микросервиса шлюза

Описание основных компонентов микросервиса приведено ниже. Ниже перечислены публичные точки доступа, которые обрабатываются API Gateway (контроллерами и минимальными endpoints), и основные маршруты Reverse Proxy, заданные конфигурацией.

Таблица 16 – Собственные endpoints ApiGateway

Метод	Точка доступа	Описание	Параметры
GET	/health	Проверка работоспособности API Gateway	–
GET	/test	Тестовый endpoint	–
GET	/metrics	Метрики Prometheus (OpenTelemetry exporter)	–

Для проксирования запросов аутентификации на внутренний AuthService реализован отдельный контроллер AuthProxyController, маршруты которого представлены в таблице 17.

Таблица 17 – Proxy endpoints (AuthProxyController)

Метод	Точка доступа	Описание	Параметры
POST	/auth/register	Проксирование регистрации в AuthService	body: JSON (email, password)
POST	/auth/login	Проксирование входа в AuthService	body: JSON (email, password)

Маршрутизация остальных запросов к внутренним сервисам осуществляется через YARP Reverse Proxy; полный перечень маршрутов с указанием политик доступа приведён в таблице 18.

Таблица 18 – YARP Reverse Proxy маршруты

Метод	Точка доступа	Назначение (куда проксируется)	Политика доступа
ANY	/auth/{**catch-all}	auth_cluster→http://authservice:8080	нет (открыто)
ANY	/files/{**catch-all}	files_cluster→http://cryptfiletransfer:8080	нет (открыто)
ANY	/logs/{**catch-all}	logs_cluster → http://logsservice:8080	нет (открыто)
ANY	/admin/{**catch-all}	auth_cluster → http://authservice:8080	AdminOnly на уровне gateway
GET	/audit/{**catch-all}	logs_cluster → http://logsservice:8080	AdminOnly на уровне gateway
POST	/audit/{**catch-all}	logs_cluster → http://logsservice:8080	без политики (запись аудита разрешена)

Указанными методами обеспечиваются основные точки входа веб-интерфейса и интеграция с сервисами аутентификации, файлового хранилища и аудита через единый шлюз.

Детальное описание реализации каждого слоя всех микросервисов будет представлено в соответствующих приложениях:

- описание слоя представления данных в приложении Д;
- описание слоя бизнес-логики в приложении Е;
- описание слоя обращения к базе данных в приложении Ж.

3.4 Реализация сервиса аутентификации

Микросервис AuthService отвечает за регистрацию пользователей, аутентификацию, выпуск JWT-токенов, генерацию криптографических ключей и административное управление учётными записями. Сервис публикует события аудита в микросервис LogsService. Микросервис реализован как независимое веб-приложение на платформе ASP.NET Core и взаимодействует с другими компонентами системы через HTTP API. Для хранения данных используется

реляционная база данных PostgreSQL, а доступ к данным осуществляется посредством Entity Framework Core с применением подхода Code First.

Архитектура микросервиса разделена на три слоя:

- Data Layer: AuthDbContext (EF Core, PostgreSQL), сущности User и UserKeyPair, конфигурация ограничений и связей;
- Presentation Layer: AuthController (регистрация, логин, получение публичного ключа), AdminUsersController (управление пользователями, политика AdminOnly);
- Service Layer: IJwtService/JwtService (генерация JWT), ICryptoService/CryptoService (RSA-ключи, AES-GCM), IAuditPublisher/AuditPublisher, AuthMetrics (метрики OpenTelemetry).

Основные компоненты:

- AuthDbContext – контекст EF Core с таблицами Users и UserKeyPairs, связь one-to-one с каскадным удалением;
- User – модель пользователя: Id, Email, PasswordHash, Role, CreatedAt, IsBlocked, BlockedAt, BlockReason, LastLoginAt, FailedLoginCount, KeyPair;
- UserKeyPair – ключевая пара: Id, UserId, PublicKeyPem, PrivateKeyEncrypted, Nonce, Tag, CreatedAt;
- RegisterRequest (Email, Password, Role?), LoginRequest (Email, Password), JwtResponse (AccessToken, ExpiresAtUnix) – DTO запросов и ответов;
- AuditEventDto – модель события аудита (Action, TimestampUtc, Service, ActorUserId, TargetResourceId, Success, Ip, Details);
- JwtService – формирует токен с email, role, срок из конфигурации;
- CryptoService – генерация RSA-2048, шифрование приватного ключа мастер-ключом (AES-GCM);
- AuditPublisher – отправка AuditEventDto через именованный HttpClient «LogsService»;
- AuthMetrics – счётчики auth_logins_success, auth_logins_failed, auth_registrations, auth_metrics_heartbeat.

Структура микросервиса представлена на рисунке 15.

Таблица 21 – Технические endpoints

Метод	Точка доступа	Описание
GET	/health	Проверка работоспособности
GET	/metrics	Метрики Prometheus

Приведённый перечень endpoint'ов охватывает полный функциональный периметр микросервиса AuthService: аутентификацию пользователей, административное управление учётными записями и наблюдение за состоянием.

3.5 Реализация сервиса шифрования и передачи файлов

Микросервис CryptFileTransfer предназначен для защищенной загрузки и скачивания файлов. Сервис реализует серверное шифрование содержимого по алгоритму AES-GCM с 256-битными ключами, генерируя уникальный ключ и вектор инициализации для каждого файла. Шифрование выполняется потоково в памяти сервера до передачи данных в хранилище. Шифртекст размещается в объектном хранилище MinIO с S3-совместимым интерфейсом, метаданные файлов – в PostgreSQL. Каждая операция публикует событие аудита для построения журнала действий.

Реализовано на ASP.NET Core с Entity Framework Core для работы с PostgreSQL, JWT-аутентификацией с валидацией через центральный сервис идентификации и политикой авторизации UserOnly. Доступ к файлам строго ограничивается владельцем: идентификатор из claim sub JWT сопоставляется с полем OwnerId, при несовпадении возвращается ошибка.

Data Layer содержит FilesDbContext — контекст EF Core с настроенными DbSet и Fluent API конфигурациями сущностей, ограничениями целостности и индексами по полям OwnerId; сущность FileMeta с валидаторами; репозиторий IFileRepository с асинхронными операциями и cancellation tokens. Presentation Layer содержит FilesController с endpoint'ами upload, download (потоковое расшифрование на лету) и получения метаданных. Infrastructure Layer сконфигурирован в Program.cs: MinIO-клиент, OpenTelemetry, Prometheus (/metrics) и health checks.

Структура микросервиса представлена на рисунке 16.

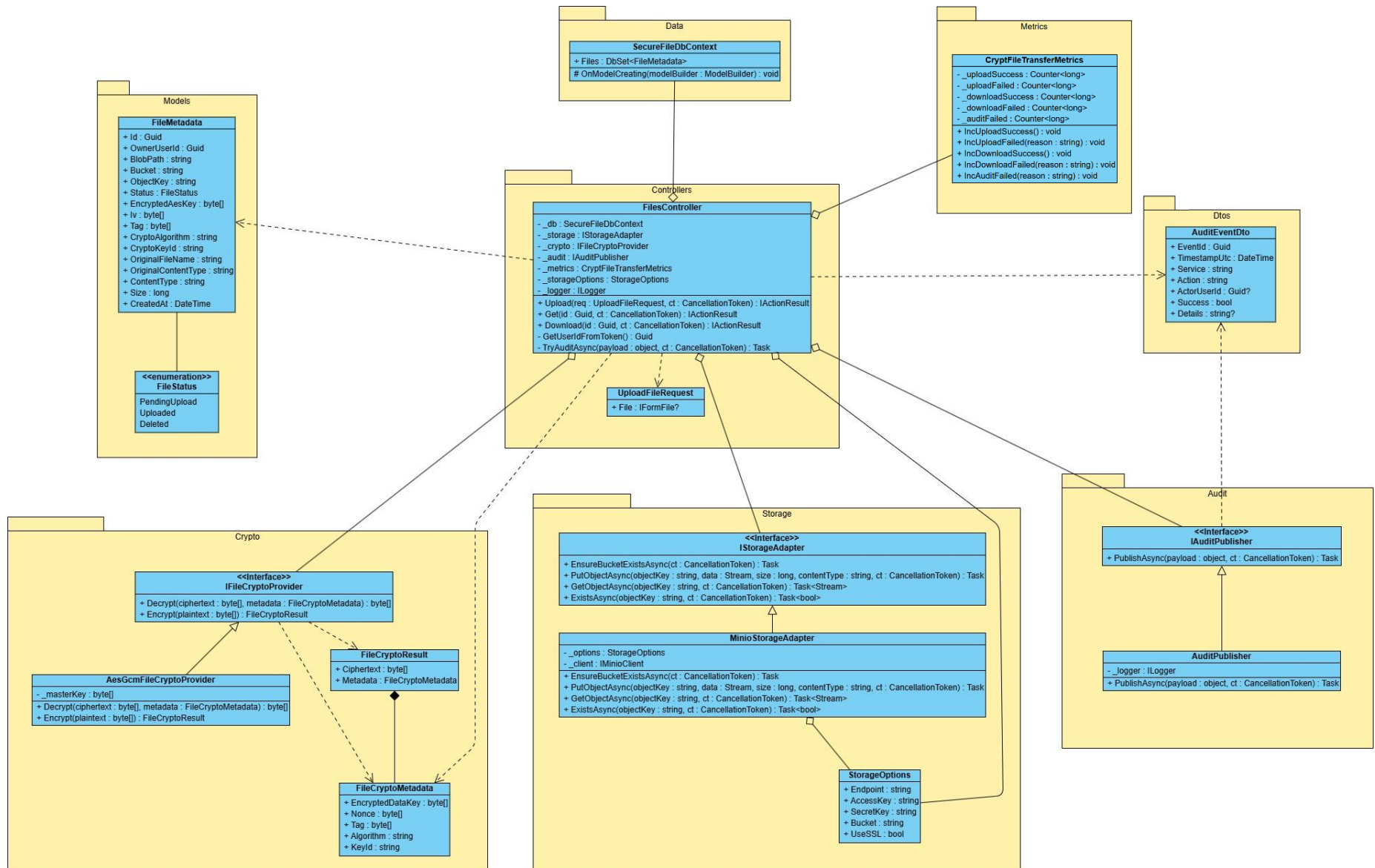


Рисунок 16 – Диаграмма классов микросервиса шифрования и передачи файлов

Ниже представлено описание компонентов микросервиса CryptFileTransfer. FilesDbContext – контекст EF Core для доступа к таблице файлов FileMetas. Настраивает индексы по полям OwnerId, CreatedAt. FileMeta – основная модель метаданных файла (хранится в БД). Содержит Id, CreatedAt, FileName, ContentType, Size; OwnerId (идентификатор владельца); StorageKey (ключ в объектном хранилище MinIO); Iv, Tag (параметры AES-GCM для расшифрования). FilesController – основной API-контроллер работы с файлами: POST /files/upload – загрузка файла (прием, шифрование AES-GCM, сохранение шифртекста в MinIO, сохранение метаданных в БД, возврат идентификатора); GET /files/download/{id} – скачивание файла (проверка прав владельца, получение шифртекста из MinIO, расшифрование, возврат оригинального файла); GET /files/{id}/meta – получение метаданных файла (только для владельца). ICryptoStorageService / CryptoStorageService – криптографический сервис хранения. Реализует шифрование содержимого файла (AES-GCM) с генерацией случайного ключа, расшифрование при скачивании, взаимодействие с MinIO. IMinioClient / MinioClient – клиент объектного хранилища. Обеспечивает загрузку шифртекста в бакет MinIO и его получение по ключу StorageKey.

Таблица 22 – Конечные точки сервиса передачи файлов

Метод	Путь	Назначение	Авторизация
POST	/files/upload	Загрузка и шифрование файла	JWT (UserOnly)
GET	/files/download/{id}	Скачивание и расшифрование файла	JWT (UserOnly, владелец)
GET	/files/{id}/meta	Получение метаданных файла	JWT (UserOnly, владелец)

Приведённые endpoint'ы реализуют полный цикл работы с файлами: загрузку с шифрованием, скачивание с расшифрованием и получение метаданных.

3.6 Реализация сервиса аудита

Микросервис LogsService предназначен для централизованного хранения и обработки событий аудита, поступающих из других микросервисов системы (AuthService, CryptFileTransfer). Сервис обеспечивает:

- приём событий аудита по HTTP и сохранение их в базе данных;
- предоставление административного доступа к журналу аудита с фильтрами и пагинацией;
- формирование агрегированной статистики по действиям и активности пользователей.

Централизованное хранение аудиторских данных позволяет обеспечить прослеживаемость всех критически важных операций в системе и упростить проведение анализа инцидентов безопасности.

Для реализации микросервиса используется ASP.NET Core 8, Entity Framework Core (PostgreSQL), JWT-аутентификация и политика авторизации AdminOnly для защищённого чтения данных.

В рамках LogsService выделяются следующие логические слои:

Data Layer (Слой данных)

Содержит:

- LogsDbContext – контекст EF Core для работы с PostgreSQL;
- сущность AuditLog – модель записи аудита;
- LogsDbContextFactory – фабрика контекста для design-time сценариев (миграции EF Core).

Presentation Layer (Слой представления / API)

Содержит контроллеры:

- AuditController – основной контроллер работы с аудитом (запись, выборка, статистика);
- LogsController – упрощённый контроллер записи и чтения логов.

Infrastructure / Security Layer (Инфраструктура и безопасность)

Реализовано через конфигурацию Program.cs:

- JWT-аутентификация (для административного чтения);
- политики UserOnly и AdminOnly;
- OpenTelemetry + Prometheus exporter (/metrics);
- автоматическое применение миграций БД при старте (с ретраями).

Ниже представлено описание компонентов микросервиса LogsService:

LogsDbContext: контекст EF Core для доступа к таблице аудита AuditLogs. Настраивает индексы по полям CreatedAt, UserId, Action, Success, что повышает производительность выборки и фильтрации.

LogsDbContextFactory: фабрика контекста (IDesignTimeDbContextFactory), используемая при создании и применении миграций EF Core вне процесса исполнения приложения.

AuditLog: основная модель записи аудита (хранится в БД). Содержит:

- Id, CreatedAt, Service, Action;
- UserId (инициатор события), TargetUserId;
- целевые данные: TargetResourceType, TargetResourceId;
- FileId (идентификатор файла при файловых операциях);
- Success, Details;
- контекст запроса: Ip, UserAgent, CorrelationId;
- MetadataJson (дополнительные данные в JSON).

AuditEventDto: контракт входного события аудита, который отправляют другие сервисы. Используется в POST /audit и преобразуется в сущность AuditLog перед сохранением. Содержит 14 полей: EventId, TimestampUtc, Service, Action, ActorUserId, TargetUserId, TargetResourceType, TargetResourceId, Success, Ip, UserAgent, CorrelationId, Details, MetadataJson.

AuditController: основной API-контроллер аудита:

- POST /audit – приём и сохранение событий (доступен без авторизации для межсервисного взаимодействия);
- GET /audit – административный просмотр журнала с фильтрами (from, to, actorUserId, action, success) и пагинацией (page, pageSize);
- GET /audit/stats – административная статистика (топ действий и топ пользователей за период).

LogsController: упрощённый контроллер логирования:

- POST /logs – запись записи аудита напрямую из модели AuditLog;
- GET /logs – получение последних 200 записей.

JWT / Authorization Policies: механизм защиты административного доступа. Методы GET /audit и GET /audit/stats защищены политикой AdminOnly. Метод POST /audit доступен анонимно для приёма событий от внутренних сервисов.

Метрики (/metrics): endpoint Prometheus для наблюдаемости (OpenTelemetry exporter). Включает инструментацию ASP.NET Core, HttpClient и .NET Runtime.

Структура микросервиса представлена на рисунке 17.

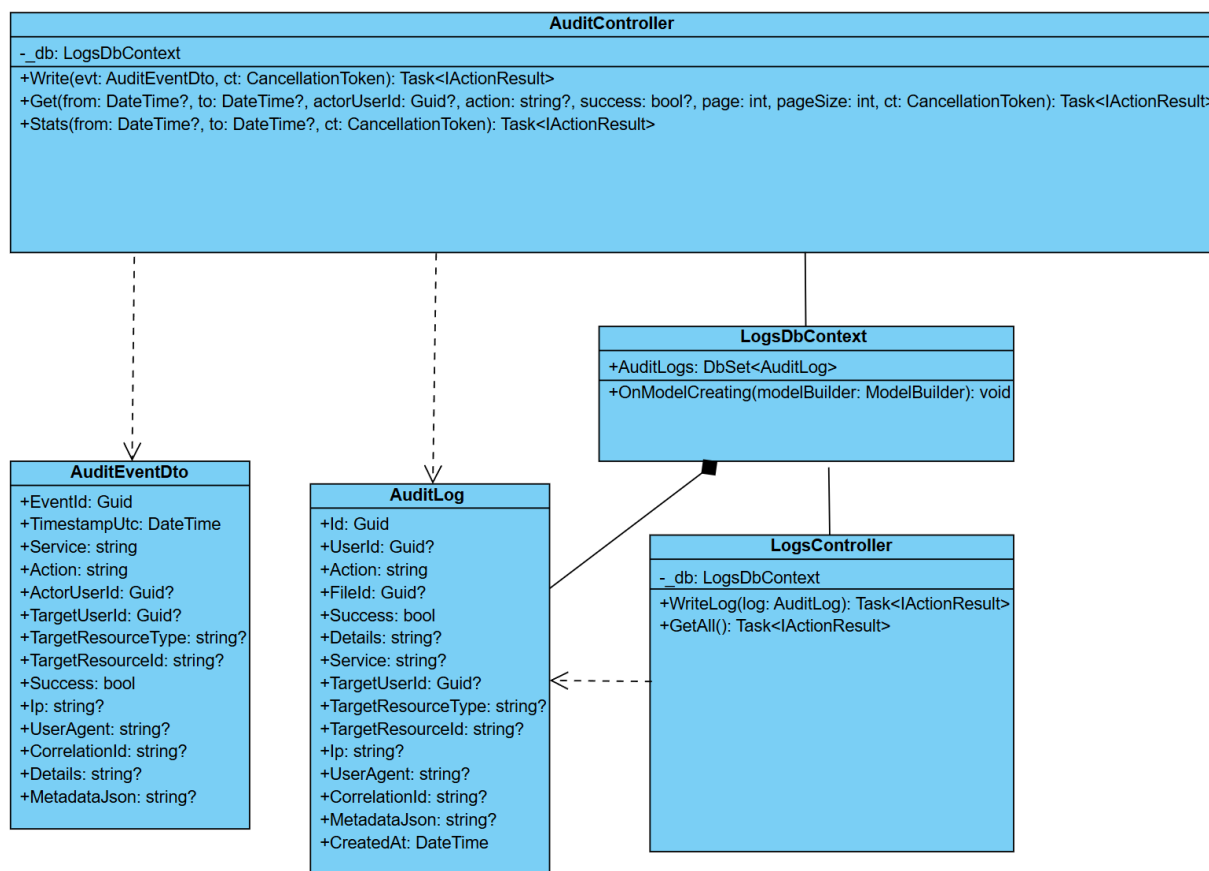


Рисунок 17 – Диаграмма классов микросервиса логирования

Методы, используемые в данном микросервисе, представлены в таблицах 23 – 25.

Таблица 23 – Основные методы аудита (AuditController)

Метод	Точка доступа	Авторизация	Описание	Параметры
POST	/audit	Нет (AllowAnonymous)	Принять событие аудита (AuditEventDto) и сохранить в БД. Валидирует обязательные поля Action и Service	body: AuditEventDto

Продолжение таблицы 23

GET	/audit	AdminOnly	Получить записи аудита с фильтрами и пагинацией (макс. 200 записей на страницу)	query: from?, to?, actorUserId?, action?, success?, page (=1), pageSize (=50)
GET	/audit/stats	AdminOnly	Получить агрегированную статистику: топ-20 действий (с разбивкой success/fail) и топ-20 пользователей	query: from?, to?

Для обратной совместимости и упрощённых сценариев записи в микросервисе дополнительно реализован LogsController с прямым доступом к хранилищу; его методы перечислены в таблице 24.

Таблица 24 – Вспомогательные методы (LogsController)

Метод	Точка доступа	Авторизация	Описание	Параметры
POST	/logs	Нет	Записать AuditLog напрямую (упрощённый сценарий). Валидирует поле Action	body: AuditLog
GET	/logs	Нет	Получить последние 200 записей AuditLog, отсортированных по дате создания	–

Технический endpoint микросервиса обеспечивает интеграцию с системой мониторинга и представлен в таблице 25.

Таблица 25 – Технические endpoints микросервиса LogsService

Метод	Точка доступа	Описание	Параметры
GET	/metrics	Метрики Prometheus (OpenTelemetry exporter)	–

Метрики позволяют осуществлять мониторинг состояния аудита и оперативно выявлять аномалии в потоке событий.

3.7 Реализация сервиса мониторинга

Микросервис MonitoringService предназначен для централизованного сбора и экспорта runtime-метрик приложения в формате, совместимом с Prometheus. Сервис построен на ASP.NET Core 8 с минимальной конфигурацией и не содержит бизнес-логики – его единственная задача состоит в предоставлении наблюдаемости (observability) для всей системы.

В проекте используются следующие компоненты:

- OpenTelemetry SDK (версия 1.9.0) для инструментирования метрик;

- `OpenTelemetry.Instrumentation.AspNetCore` – автоматический сбор метрик HTTP-запросов (длительность, количество, коды ответов);
- `OpenTelemetry.Instrumentation.Runtime` – метрики среды выполнения .NET (GC, ThreadPool, выделение памяти);
- `OpenTelemetry.Exporter.Prometheus.AspNetCore` – экспорт собранных метрик через HTTP-эндпоинт `/metrics` в текстовом формате Prometheus.

Для сервиса можно выделить следующие логические слои:

3.7.1 Presentation Layer (Слой представления)

Содержит минимальные endpoints, определённые в `Program.cs`:

- `GET /health` – проверка работоспособности сервиса, возвращает JSON-объект со статусом и текущей временной меткой;
- `GET /metrics` – экспонирование метрик Prometheus.

3.7.2 Infrastructure Layer (Инфраструктурный слой)

Конфигурация OpenTelemetry реализована через fluent API в `Program.cs`:

- `ConfigureResource(r => r.AddService("MonitoringService"))` – идентификация источника метрик;
- `AddAspNetCoreInstrumentation()` – автоматический сбор метрик;
- `AddRuntimeInstrumentation()` – метрики среды выполнения;
- `AddPrometheusExporter()` – экспорт данных на `/metrics`.

Сервис не использует баз данных, внешних API или механизмов аутентификации. Swagger UI доступен только в режиме Development.

Публичные точки доступа приведены в таблице 26.

Таблица 26 – Endpoints микросервиса MonitoringService

Метод	Точка доступа	Описание	Параметры
GET	<code>/health</code>	Проверка работоспособности MonitoringService	–
GET	<code>/metrics</code>	Метрики Prometheus (OpenTelemetry exporter)	–

MonitoringService развёртывается как отдельный Docker-контейнер на порту 8080 (внутренний). Prometheus опрашивает эндпоинт `/metrics` каждые 5 секунд наряду с аналогичными эндпоинтами остальных четырёх микросервисов.

3.8 Реализация баз данных и хранилища

Подсистема хранения данных CloudSecure включает реляционную СУБД PostgreSQL 16, объектное хранилище MinIO и инфраструктуру мониторинга (Prometheus + Grafana). Все компоненты работают в Docker-контейнерах и связаны общей Docker-сетью.

3.8.1 Реляционные базы данных PostgreSQL

Единый экземпляр PostgreSQL 16 обслуживает две базы данных:

- AuthDb – данные пользователей и ключевых пар (AuthService);
- SecureFileDb – метаданные файлов (CryptFileTransfer) и записи аудита (LogsService).

Миграции обеих баз управляются Entity Framework Core (подход Code-First). При запуске контейнера каждый сервис автоматически применяет ожидающие миграции с механизмом повторных попыток.

Основные индексы базы AuthDb:

- уникальный индекс по полю Email (таблица Users) – гарантия уникальности почтовых адресов;
- индекс по UserId (таблица UserKeyPairs) – быстрый поиск ключевой пары пользователя.

Основные индексы базы SecureFileDb (таблица AuditLogs):

- индекс по CreatedAt – выборка событий по диапазону дат;
- индекс по UserId – фильтрация по пользователю;
- индекс по Action – фильтрация по типу действия;
- индекс по Success – фильтрация по результату операции.

Healthcheck контейнера PostgreSQL выполняется командой `pg_isready` каждые 5 секунд.

Зависимые сервисы (AuthService, CryptFileTransfer, LogsService) запускаются только после успешного прохождения проверки (condition: service_healthy).

3.8.2 Объектное хранилище MinIO

MinIO развёрнут в контейнере на основе образа `minio/minio:RELEASE.2024-05-28T17-19-04Z` и предоставляет S3-совместимый API для хранения зашифрованных файлов.

Конфигурация хранилища:

- бакет: `files` – единый бакет для всех загруженных файлов;
- порт API: `9000` – взаимодействие с `CryptFileTransfer` через Minio SDK;
- порт консоли: `9001` – веб-интерфейс администрирования MinIO;
- данные хранятся в Docker-томе `minio-data` для обеспечения персистентности.

`CryptFileTransfer` обращается к MinIO по внутреннему DNS-имени `minio:9000` без использования SSL. Каждый файл сохраняется с уникальным `ObjectKey`, сформированным на основе идентификатора `FileMetadata`.

3.8.3 Prometheus и Grafana

Prometheus (версия 2.53.0) собирает метрики со всех пяти микросервисов. Конфигурация определена в файле `infra/prometheus/prometheus.yml` и содержит пять scrape-целей:

- `apigateway:8080/metrics`;
- `authservice:8080/metrics`;
- `cryptfiletransfer:8080/metrics`;
- `logsservice:8080/metrics`;
- `monitoringservice:8080/metrics`.

Интервал опроса составляет 5 секунд (`scrape_interval: 5s`). Prometheus доступен на порту `9090` хост-машины.

Grafana (версия 11.2.0) подключена к Prometheus как источник данных и предоставляет визуализацию метрик через веб-интерфейс на порту `3001`. Базовые учётные данные: `admin/admin`.

3.9 Реализация пользовательского интерфейса

Пользовательский интерфейс системы CloudSecure реализован как одностраничное веб-приложение (SPA), встроенное в контейнер ApiGateway. Статические файлы (index.html, styles.css) размещены в каталоге wwwroot и раздаются через middleware UseStaticFiles/UseDefaultFiles.

Интерфейс написан на чистом HTML, CSS и JavaScript без использования внешних фреймворков. Стилизация выполнена с применением CSS-переменных и адаптивных grid-макетов. Шрифт: Plus Jakarta Sans (Google Fonts).

Навигация организована через систему вкладок (табов). Основные разделы интерфейса:

- регистрация – форма создания учётной записи (email, пароль);
- вход – форма аутентификации, при успехе JWT-токен сохраняется в localStorage;
- файлы – загрузка файлов (multipart/form-data через /files/upload);
- скачать – скачивание файла по идентификатору (GET /files/download/{id});
- администрирование – панель управления пользователями (вкладка отображается только при роли Admin).

Все запросы к серверу проходят через единый шлюз ApiGateway. JWT-токен передаётся в заголовке Authorization: Bearer. При отсутствии или истечении токена интерфейс перенаправляет пользователя на вкладку входа.

Административная панель позволяет просматривать список пользователей, изменять их роли и блокировать учётные записи. Доступ к панели ограничен политикой AdminOnly на уровне шлюза.

3.10 Развёртывание и запуск приложения

Система CloudSecure развёртывается с помощью Docker Compose. Файл docker-compose.yml определяет 9 контейнеров: 5 микросервисов приложения и 4 инфраструктурных компонента.

3.10.1 Dockerfile микросервисов

Каждый микросервис использует многоэтапную сборку:

- этап build: образ mcr.microsoft.com/dotnet/sdk:8.0, выполняет dotnet restore и dotnet publish -c Release;
- этап final: образ mcr.microsoft.com/dotnet/aspnet:8.0, содержит только скомпилированные артефакты.

Такой подход минимизирует размер итогового образа (около 220 МБ вместо 900+ МБ с SDK) и исключает утечку исходного кода в production-контейнер.

3.10.2 Конфигурация Docker Compose

Состав контейнеров системы приведён в таблице 27.

Таблица 27 – Контейнеры Docker Compose

Контейнер	Образ / сборка	Порт (хост:контейнер)	Назначение
cloudsecure-apigateway	./src/ApiGateway	8080:8080	API Gateway (YARP)
cloudsecure-postgres	postgres:16	55432:5432	СУБД PostgreSQL
cloudsecure-minio	minio/minio:RELEASE.2024-05-28	9000:9000, 9001:9001	Объектное хранилище S3
cloudsecure-authservice	./src/AuthService	5024:8080	Сервис аутентификации
cloudsecure-cryptfiletransfer	./src/CryptFileTransfer	5240:8080	Сервис шифрования файлов
cloudsecure-logsservice	./src/LogsService	5262:8080	Сервис аудита
cloudsecure-monitoringservice	./src/MonitoringService	5088:8080	Сервис мониторинга
cloudsecure-prometheus	prom/prometheus:v2.53.0	9090:9090	Сбор метрик
cloudsecure-grafana	grafana/grafana:11.2.0	3001:3000	Визуализация метрик

Все контейнеры объединены в общую сеть Docker и взаимодействуют между собой по именам сервисов без экспонирования внутренних портов наружу. Такой подход обеспечивает сетевую изоляцию компонентов системы и исключает прямой доступ к внутренним сервисам в обход API Gateway.

3.10.3 Переменные окружения и сетевая конфигурация

Конфигурация сервисов передаётся через переменные окружения в `docker-compose.yml`. Основные параметры:

- `ConnectionStrings__DefaultConnection` – строка подключения к PostgreSQL (`Host=postgres;Port=5432`);
- `Jwt__Key`, `Jwt__Issuer`, `Jwt__Audience` – единые параметры JWT для всех сервисов;
- `Storage__Endpoint`, `Storage__AccessKey`, `Storage__SecretKey` – подключение к MinIO;
- `Services__LogsBaseUrl` – URL сервиса аудита для `CryptFileTransfer`.

Все контейнеры находятся в общей Docker-сети (`default`), что позволяет обращаться к сервисам по имени контейнера. Дополнительная внешняя сеть `dokploy-network` подключена для интеграции с системой развёртывания `Dokploy`.

3.10.4 Порядок запуска и проверка работоспособности

Docker Compose обеспечивает корректный порядок запуска через директивы `depends_on` с условиями:

- PostgreSQL запускается первым; `AuthService`, `CryptFileTransfer` и `LogsService` ожидают прохождения `healthcheck` (`pg_isready`);
- `ApiGateway` запускается после старта `AuthService`, `CryptFileTransfer` и `LogsService`;
- `Prometheus` запускается после всех микросервисов;
- `Grafana` запускается после `Prometheus`.

Проверка работоспособности системы выполняется последовательными запросами:

- `GET http://localhost:8080/health` – проверка `ApiGateway`;
- `GET http://localhost:8080/auth/register` – проверка маршрутизации к `AuthService`;

- GET `http://localhost:9090/targets` – статус scrape-целей Prometheus (все 5 сервисов должны иметь статус UP);
- GET `http://localhost:3001` – доступность Grafana.

Для запуска всей системы достаточно выполнить команду `docker compose up --build` из корневого каталога проекта. Время полной сборки и запуска составляет 2-3 минуты на стандартном оборудовании.

4 Контроль качества программного обеспечения

После завершения разработки серверной части системы CloudSecure проводилось комплексное тестирование для подтверждения корректности работы микросервисов, оценки безопасности доступа к защищённым файлам и измерения производительности криптографических операций. Контроль качества охватывал функциональные аспекты системы, анализ метрик кода и нагрузочное тестирование, что позволило объективно оценить надёжность разработанного программного обеспечения.

4.1 Обоснование выбора модели тестирования и метрик кода

В процессе разработки серверной части системы CloudSecure особое внимание уделялось обеспечению качества программного кода. Комплексный подход к тестированию и анализу качества кода позволил гарантировать надёжность, безопасность и эффективность функционирования компонентов микросервисной архитектуры.

Основные виды тестирования серверной части включают:

- модульное тестирование – проверка корректности работы отдельных компонентов и модулей системы в изоляции [17];
- функциональное тестирование – проверка соответствия реализованных функций заявленным требованиям [18];
- нагрузочное тестирование – оценка производительности криптографических операций и расчёт коэффициента замедления;
- расчёт метрик Холстеда – анализ сложности алгоритмов и оценка трудоёмкости разработки.

Метрики качества кода играют важную роль в оценке надёжности серверной части системы CloudSecure. В рамках данной работы были применены метрики Холстеда для оценки сложности криптографических операций и нагрузочные метрики для оценки производительности шифрования.

Выбор данных методов обусловлен необходимостью получить объективную оценку как функциональной корректности системы, так и её

производительностных характеристик. Регулярный мониторинг метрик позволяет поддерживать высокий уровень качества серверной части системы CloudSecure.

Таблица 28 – Виды тестирования, применённые в проекте

Вид тестирования	Назначение	Покрываемые компоненты
Модульное	Проверка корректности криптографических преобразований	AesGcmFileCryptoProvider
Функциональное	Проверка API и обработки ошибок методом чёрного ящика	FilesController, JWT-аутентификация
Нагрузочное	Оценка времени шифрования и коэффициента замедления	AesGcmFileCryptoProvider

Из таблицы 28 видно, что выбранный набор тестов покрывает основные технические риски проекта: корректность криптографических операций, проверка бизнес-логики API и оценка производительности шифрования.

4.2 Функциональное тестирование методом чёрного ящика

Тестирование методом чёрного ящика – это подход к тестированию программного обеспечения, при котором тестирующий оценивает функциональность приложения без знания его внутренней структуры или кода. При применении данного метода основное внимание уделяется входным данным и ожидаемым выходным результатам. Тестирующий создаёт тест-кейсы на основе требований и спецификаций, проверяя, соответствует ли поведение программы заданным критериям. Этот метод позволяет сосредоточиться на том, как система работает с точки зрения пользователя, и выявлять ошибки, которые могут быть незаметны при анализе внутренней работы программы.

4.2.1 Функциональная диаграмма и таблица решений

Для выявления потенциальных проблем при обработке запросов загрузки и скачивания файлов в системе CloudSecure (функционал FilesController) были выделены элементарные причины и следствия.

Метод причинно-следственного анализа позволяет систематически охватить все граничные сценарии поведения сервиса и сформировать минимально необходимое покрывающее множество тестовых случаев.

Причины (Входные условия):

- П1: JWT-токен передан и действителен (пользователь аутентифицирован).
- П2: Файл передан в теле запроса (поле file заполнено в multipart-форме).
- П3: Размер файла больше нуля байт.
- П4: Файл с указанным идентификатором существует в базе данных.
- П5: Идентификатор пользователя из токена совпадает с OwnerUserId файла.

Следствия (Выходные результаты):

- C1: 401 Unauthorized – отказ в доступе без авторизации.
- C2: 400 Bad Request – «File is required» (файл не передан или пустой).
- C3: 200 OK + JSON с fileId – успешная загрузка файла.
- C4: 404 Not Found – файл не найден в базе данных.
- C5: 403 Forbidden – попытка доступа к чужому файлу.
- C6: 200 OK + байты файла – успешное скачивание файла.

На основе данных условий построена таблица решений. Таблица отражает строгую логику последовательного анализа: если выполнение условия завершается ошибкой, последующие условия не проверяются. Такой подход соответствует реальному поведению middleware-конвейера ASP.NET Core, в котором проверка аутентификации предшествует авторизации, а авторизация – бизнес-логике обработки запроса. Совокупность выделенных причин и следствий обеспечивает полное покрытие ветвей логики FilesController без дублирования тестовых сценариев.

Выделенные условия охватывают два независимых сценария: загрузку нового файла и скачивание существующего. Для каждого сценария определён минимальный набор проверок, последовательное невыполнение которых однозначно определяет код ответа системы.

Графическое представление причин и следствий в виде функциональной диаграммы представлено на рисунке 18.

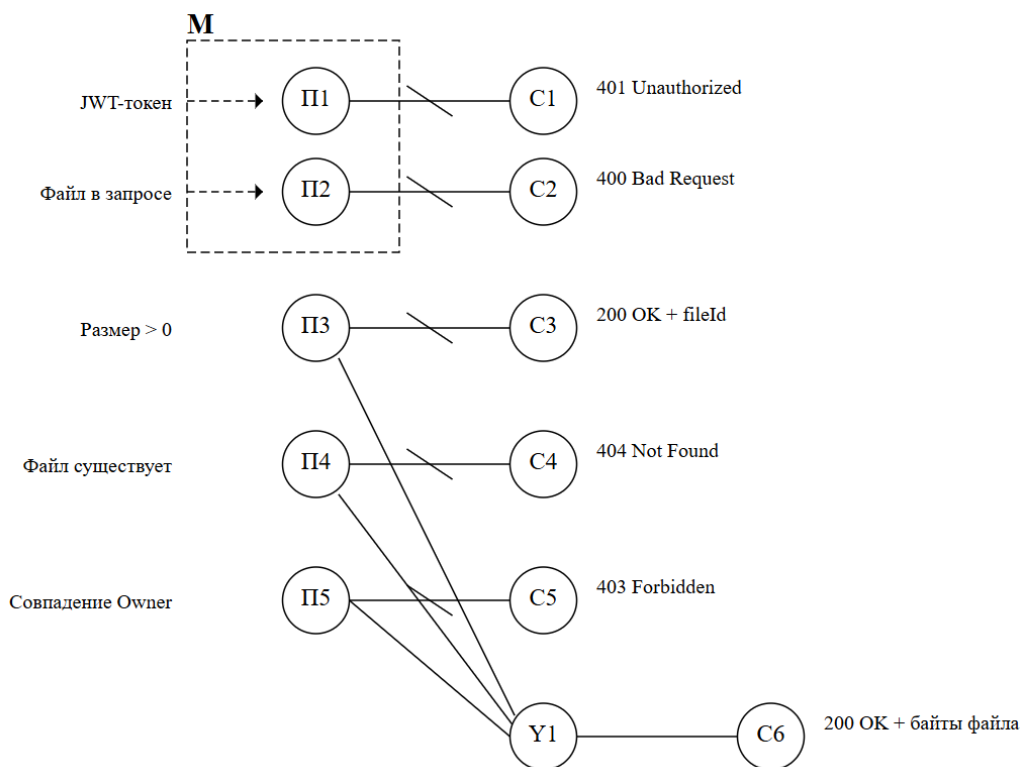


Рисунок 18 – Функциональная диаграмма

Для загрузки файла необходимо одновременное выполнение условий П1, П2, П3. Для скачивания файла необходимо выполнение условий П1, П4, П5. Все наиболее вероятные комбинации входных и выходных данных приведены в приложении Л.

Построенная функциональная диаграмма наглядно демонстрирует разветвлённую логику принятия решений и служит основой для разработки модульных тестов, покрывающих все сценарии отказа и успешного выполнения операций. Каждая ветвь диаграммы соответствует отдельному тестовому случаю в таблице решений, что исключает дублирование проверок и обеспечивает минимально достаточное покрытие без избыточности.

Таким образом, применение метода причинно-следственного анализа позволило формализовать логику работы FilesController и получить структурированное множество тестовых сценариев, охватывающих как штатные пути выполнения, так и все предусмотренные варианты отказа, включая отсутствие аутентификации, некорректные входные данные и попытки несанкционированного доступа к чужим файлам.

В части следствий:

- C1 = 1 только если П1 = 0 (отсутствует валидный JWT-токен) – во всех остальных случаях C1 = 0.
- C2 = 1 если П1 = 1, но П2 = 0 или П3 = 0 (Upload без файла или с пустым файлом).
- C3 = 1 если П1 = 1, П2 = 1, П3 = 1 (Upload с валидным файлом – успех).
- C4 = 1 если П1 = 1, П4 = 0 (Download – файл не найден в БД).
- C5 = 1 если П1 = 1, П4 = 1, П5 = 0 (Download – файл найден, но запрашивающий не владелец).
- C6 = 1 если П1 = 1, П4 = 1, П5 = 1 (Download – успешное скачивание).

Следствия C2/C3 относятся к операции Upload и зависят от П2, П3. Следствия C4/C5/C6 относятся к операции Download и зависят от П4, П5. Условия П4 и П5 не влияют на Upload; условия П2 и П3 не влияют на Download. На основании выявленных зависимостей сформированы тест-кейсы, охватывающие все уникальные комбинации условий, приведённые в таблице 29.

Таблица 29 – Тест-кейсы для FilesController (Upload/Download)

№	Описание тест-кейса	Входные данные	Ожидаемый результат
ТС-01	Загрузка файла без JWT-токена	POST /files/upload без заголовка Authorization	401 Unauthorized
ТС-02	Загрузка без файла (поле file отсутствует)	POST /files/upload, multipart без file	400 Bad Request, тело: {"message": "File is required"}
ТС-03	Успешная загрузка файла	POST /files/upload, валидный JWT, файл в теле	200 OK, тело: {"fileId": "..."}
ТС-04	Доступ к метаданным чужого файла	GET /files/{id}, JWT другого пользователя	403 Forbidden
ТС-05	Запрос несуществующего файла	GET /files/{несуществующий_id}	404 Not Found
ТС-06	Успешное скачивание своего файла	GET /files/{id}/download, JWT владельца	200 OK, тело: байты файла
ТС-07	AES-256-GCM round-trip Encrypt/Decrypt	Произвольный массив байт, мастер-ключ	Дешифрованные данные == исходные

Сформированные тест-кейсы ТС-01 – ТС-07 покрывают все ветви логики FilesController: отказ по аутентификации, валидационные ошибки загрузки,

разграничение доступа по владельцу и корректность криптографического round-trip.

4.2.2 Оценка сложности по метрикам Холстеда

Для оценки сложности алгоритма загрузки файла был проведён анализ метода Upload класса FilesController по метрикам Холстеда. Метрики Холстеда позволяют количественно оценить сложность программы на основе подсчёта операторов и операндов в исходном коде.

При анализе кода метода Upload были выделены следующие операторы и операнды.

Таблица 30 – Операторы метода FilesController.Upload

№	Оператор	Количество (N_{ij})	Уникальных (n_{ij})	Описание
1	await	6	1	Ожидание асинхронной операции
2	if	4	1	Условный оператор
3	return	5	1	Возврат значения
4	=	7	1	Присвоение
5	new	5	1	Создание объекта
6	.	22	1	Доступ к члену
7	()	18	1	Вызов метода
8	[]	2	1	Индексация массива
9	using	2	1	Управление ресурсами
–	Итого	71	9	

Перечень операндов метода, используемых при вычислении метрик, представлен в таблице 31.

Таблица 31 – Операнды метода FilesController.Upload

№	Операнд	N_{2i} (вхождений)	n_{2i} (уникальных)	Тип
1	userId	4	1	Guid
2	req	4	1	UploadFileRequest
3	req.File	3	1	IFormFile
4	fileId	3	1	Guid
5	objectKey	2	1	string
6	plaintext	4	1	byte[]
7	cryptoResult	3	1	FileCryptoResult
8	meta	5	1	FileMetadata
9	ct	3	1	CancellationTokens
10	Guid.Empty	1	1	Guid
11	0	2	1	int literal
12	application/octet-stream	1	1	string literal
–	Итого	35	12	

По данным таблиц 30 и 31 рассчитаны метрики Холстеда.

Вычисление словаря программы производится по формуле:

$$n = n_1 + n_2,$$

где n_1 – число уникальных операторов;

n_2 – число уникальных операндов.

Вычисление длины программы производится по формуле:

$$N = N_1 + N_2,$$

где N_1 – общее число операторов;

N_2 – общее число операндов.

Вычисление объёма программы производится по формуле:

$$V = N \times \log_2(n),$$

где N – длина программы;

n – словарь программы.

Вычисление сложности программы производится по формуле:

$$D = \frac{n_1}{2} \times \frac{N_2}{n_2},$$

где N_2 – общее число операндов;

n_1 – число уникальных операторов; n_2 – число уникальных операндов.

Вычисление усилия реализации производится по формуле:

$$E = D \times V,$$

где D – сложность программы;

V – объём программы.

Вычисление времени реализации производится по формуле:

$$T = \frac{E}{18},$$

где E – усилие реализации.

Подставляя значения $n_1 = 9$, $n_2 = 12$, $N_1 = 71$, $N_2 = 35$, получаем: $n = 21$, $N = 106$, $V \approx 465,5$, $D \approx 13,1$, $E \approx 6098$, $T \approx 339$ сек $\approx 5,6$ мин.

Полученные значения свидетельствуют об умеренной сложности метода Upload и подтверждают его функциональную лаконичность при достаточном уровне криптографической и бизнес-логики.

4.2.3 Автоматизированное тестирование

Для автоматизированного тестирования функционала FilesController применялся фреймворк xUnit совместно с Microsoft.AspNetCore.Mvc.Testing. Данный подход позволяет запускать полноценное ASP.NET Core-приложение в памяти, заменяя реальные зависимости тестовыми двойниками: in-memory базой данных Entity Framework Core и Mock-объектами Moq.

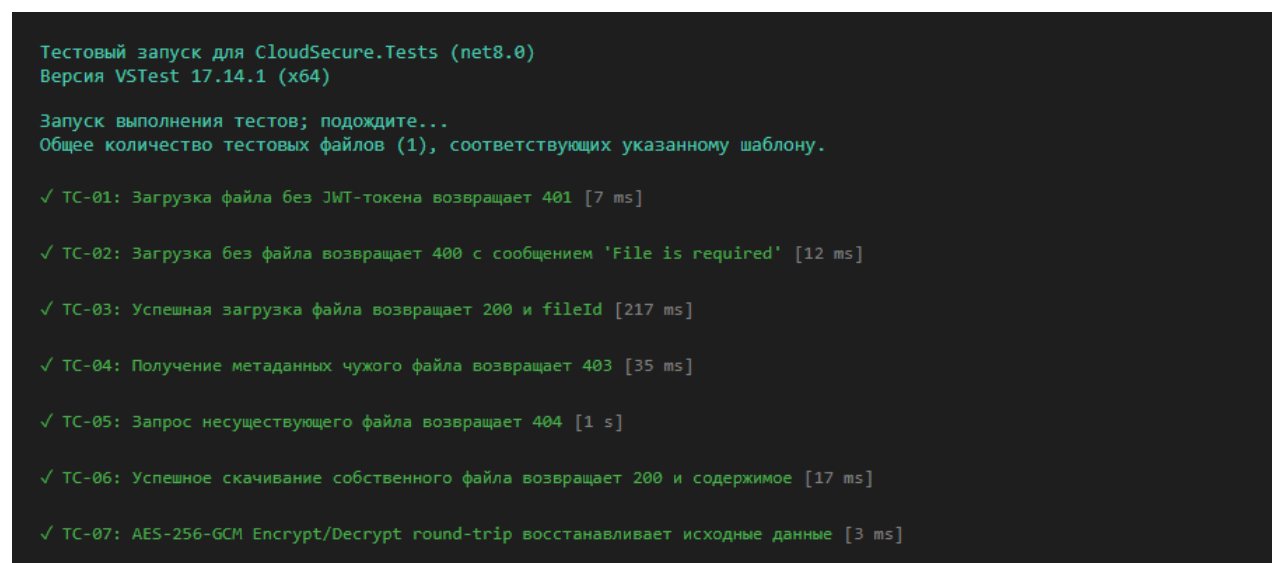
Тестовый класс FilesControllerTests содержит 7 тест-кейсов, покрывающих основные сценарии работы с файлами: загрузку, скачивание, проверку прав доступа и криптографический round-trip AES-256-GCM. Каждый тест создаёт независимый HTTP-клиент с собственным JWT, что исключает взаимное влияние тестов.

Ниже представлен фрагмент модульного теста криптографического провайдера.

Листинг 1 – Фрагмент модульного теста криптографического провайдера

```
[Fact(DisplayName="TC-07:AES-256-GCM Encrypt/Decrypt round-trip")]
public async Task TC07_AesGcm_EncryptDecrypt_RoundTrip()
{
    var result = provider.Encrypt(originalData);
    var decrypted = provider.Decrypt(result.Ciphertext,
    result.Metadata);
    Assert.Equal(originalData, decrypted);
    Assert.Equal("AES-256-GCM", result.Metadata.Algorithm);
}
```

На рисунке 19 приведены общие результаты автоматизированного тестирования.



```
Тестовый запуск для CloudSecure.Tests (net8.0)
Версия VSTest 17.14.1 (x64)

Запуск выполнения тестов; подождите...
Общее количество тестовых файлов (1), соответствующих указанному шаблону.

✓ TC-01: Загрузка файла без JWT-токена возвращает 401 [7 ms]

✓ TC-02: Загрузка без файла возвращает 400 с сообщением 'File is required' [12 ms]

✓ TC-03: Успешная загрузка файла возвращает 200 и fileId [217 ms]

✓ TC-04: Получение метаданных чужого файла возвращает 403 [35 ms]

✓ TC-05: Запрос несуществующего файла возвращает 404 [1 s]

✓ TC-06: Успешное скачивание собственного файла возвращает 200 и содержимое [17 ms]

✓ TC-07: AES-256-GCM Encrypt/Decrypt round-trip восстанавливает исходные данные [3 ms]
```

Рисунок 19 – Результаты выполнения тестов

Все 7 тест-кейсов успешно пройдены. Общее время выполнения тестового набора составило 3,13 секунды. Тест ТС-05 занял наибольшее время (1 с) из-за внутреннего таймаута ожидания ответа при несуществующем файле.

4.2.4 Нагрузочное тестирование и оценка коэффициента замедления

Помимо функциональной корректности системы, для проекта CloudSecure важна производительность криптографической обработки файлов. По этой причине было проведено отдельное тестирование времени шифрования и расшифрования, а также рассчитан коэффициент замедления по сравнению с базовой операцией копирования данных в памяти.

Коэффициент замедления показывает, во сколько раз криптографическая операция выполняется медленнее по сравнению с простым копированием массива байтов. Данный показатель позволяет оценить практическую стоимость применения шифрования в серверной части системы.

Измерения проводились для файлов размером 1 КБ, 1 МБ и 10 МБ.

Результаты тестирования шифрования представлены в таблице 32.

Таблица 32 – Результаты измерения времени шифрования и коэффициента замедления

Размер данных	Загрузка, мс	Шифрование, мс	Коэффициент замедления	Скорость, МВ/с
1 КБ	0,000	0,004	18,00х	226,06
1 МБ	0,210	0,643	3,06х	1555,67
10 МБ	2,740	5,073	1,85х	1971,31

Из таблицы 32 видно, что на малых объёмах коэффициент замедления выше из-за накладных расходов на инициализацию. Для файлов 10 МБ скорость шифрования составила 1971 МВ/с.

Из таблицы 32 видно, что на малых объёмах данных коэффициент замедления выше, что связано с постоянными накладными расходами на инициализацию криптографической операции. При увеличении размера обрабатываемого файла коэффициент снижается, а пропускная способность возрастает. Для файла объёмом 10 МБ скорость шифрования составила 1971,31 МВ/с, что свидетельствует о высокой эффективности выбранного алгоритма.

Результаты тестирования расшифрования представлены в таблице 33.

Таблица 33 – Результаты измерения времени расшифрования и коэффициента замедления

Размер данных	Baseline, мс	Расшифрование, мс	Коэффициент замедления	Скорость, МВ/с
1 КБ	0,000	0,003	14,60х	334,44
1 МБ	0,282	0,474	1,68х	2109,88
10 МБ	2,742	5,413	1,97х	1847,49

Полученные результаты показывают, что расшифрование также выполняется с приемлемой производительностью. Для прикладных объёмов файлов в 1-10 МБ коэффициент замедления находится в пределах, не создающих критических ограничений для эксплуатации системы.

Дополнительно было выполнено сравнение средних времён шифрования и расшифрования для файла объёмом 1 МБ. Результаты в таблице 34.

Таблица 34 – Сравнение времени шифрования и расшифрования

Показатель	Значение
Среднее время шифрования	0,657 мс
Среднее время расшифрования	0,510 мс
Разница	0,148 мс
Отношение шифрование/расшифрование	1,29

Значение отношения шифрование/расшифрование = 1,29 показывает, что операция шифрования выполняется несколько медленнее операции расшифрования, однако различие остаётся незначительным. Это позволяет сделать вывод о сопоставимой производительности обоих направлений криптографической обработки.

Таким образом, результаты тестирования подтверждают, что использование AES-256-GCM в составе серверной части системы CloudSecure обеспечивает не только конфиденциальность и контроль целостности файлов, но и приемлемый уровень производительности. Для файлов прикладного размера выбранная схема шифрования не создаёт критического замедления и может использоваться в эксплуатационной среде. Полученные данные могут служить обоснованием выбора алгоритма AES-256-GCM для дальнейшего масштабирования системы и обработки файлов большего объёма.

4.3 Планирование разработки и оценка бюджета

Планирование разработки и оценка бюджета являются критически важными этапами для успешной реализации программного продукта [19]. Грамотное планирование позволяет заблаговременно оценить трудоёмкость задач, распределить ресурсы и минимизировать риски срыва сроков. Структурированный подход к планированию облегчает коммуникацию внутри команды и позволяет своевременно выявлять узкие места в процессе реализации.

Управление проектом проводилось в программе ProjectLibre – свободно распространяемом аналоге Microsoft Project, позволяющем планировать разработку, отслеживать выполнение поставленных задач и контролировать ресурсы. ProjectLibre поддерживает построение иерархических структур работ, назначение исполнителей, расчёт критического пути и экспорт отчётов. В систему были внесены все этапы разработки с указанием длительности, зависимостей и назначенных ресурсов; связи типа «финиш-старт» отражают технологическую последовательность работ. После заполнения данных программа автоматически рассчитала даты этапов, построила диаграмму Ганта и сформировала оценку финансовых затрат.

Диаграмма Ганта – это горизонтальный ленточный график, отображающий перечень задач на вертикальной оси и временную шкалу на горизонтальной [20]. Длина полосы соответствует продолжительности задачи, а их взаимное расположение отражает зависимости между этапами и параллельно выполняемые работы. Диаграмма Ганта является одним из наиболее распространённых инструментов визуализации проектного расписания и позволяет наглядно оценить общую продолжительность проекта, загруженность ресурсов и критический путь выполнения работ. Применение данного инструмента в рамках проекта CloudSecure обеспечило прозрачность планирования и позволило контролировать соблюдение сроков на каждом этапе разработки.

Построенная диаграмма Ганта проекта CloudSecure представлена на рисунке 20.

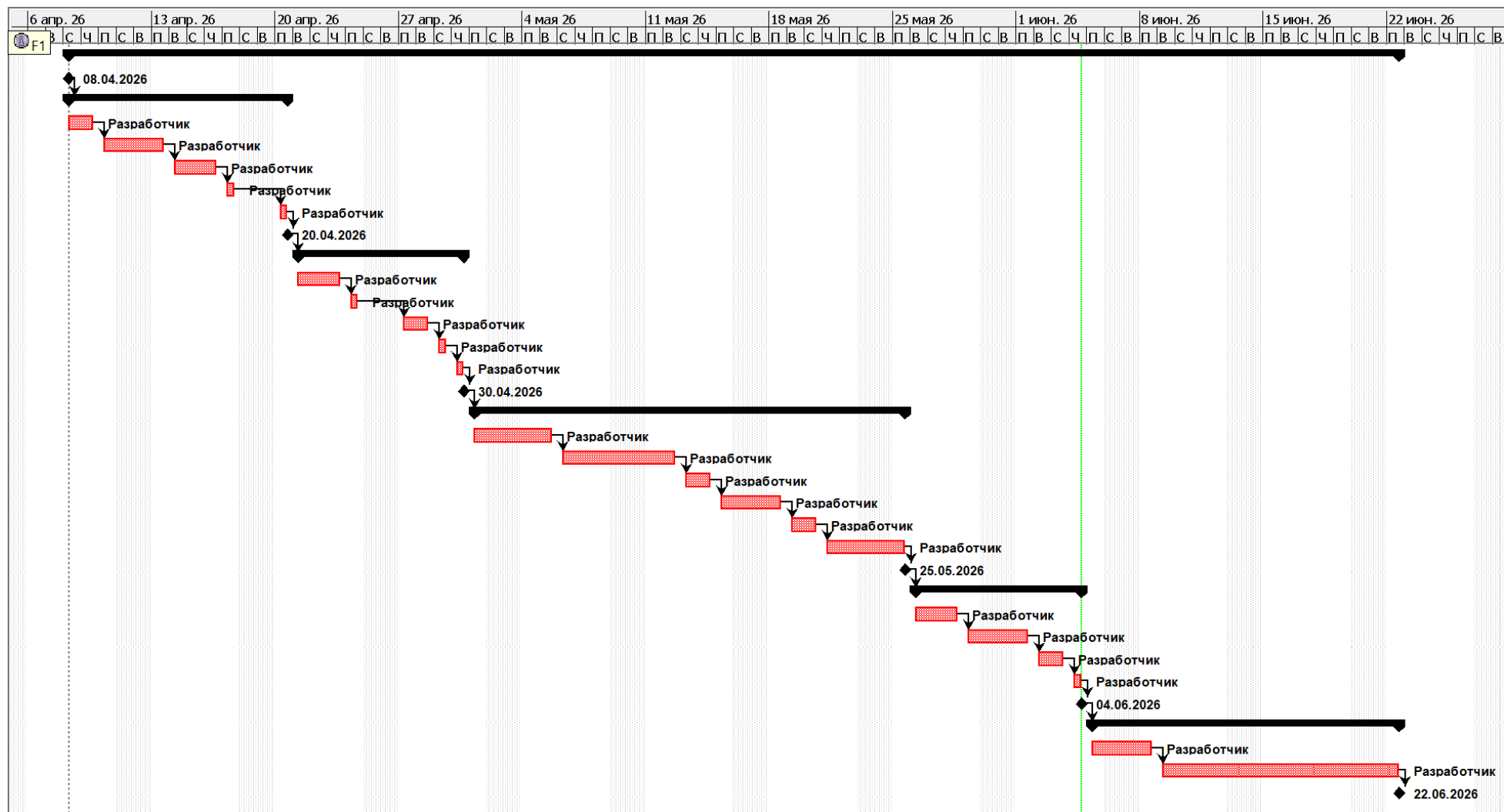


Рисунок 20 – Диаграмма Ганта

Заключение

В ходе прохождения преддипломной практики были выполнены задачи, поставленные в рамках индивидуального задания: сформулирована цель работы, определены основные задачи и выбраны методы проектирования системы. Было проведено исследование предметной области облачных технологий и криптографических методов защиты данных, в результате которого были выделены ключевые процессы: аутентификация пользователей с использованием JWT-токенов и двухфакторной верификации, шифрование файлов с применением алгоритма AES-256-GCM, асимметричное шифрование ключей с использованием RSA-2048, безопасное хранение приватных ключей в формате PKCS#8, хеширование паролей с помощью BCrypt, управление сессиями и разграничение доступа, а также проверка целостности данных через SHA-256.

Также был выполнен сравнительный анализ программных продуктов аналогов, в ходе которого были выявлены основные ограничения существующих решений, связанные с архитектурной гибкостью, масштабируемостью и удобством интерфейса.

На основе анализа предметной области и существующих решений были сформулированы функциональные и нефункциональные требования к разрабатываемому программному продукту.

В процессе проектирования была обоснована микросервисная архитектура приложения, позволяющая разделить систему на независимые функциональные компоненты. Были спроектированы структуры данных, разработаны концептуальная и логическая модели базы данных, а также описан пользовательский интерфейс платформы. Также были разработаны UML-диаграммы классов программных средств, отражающие структуру основных компонентов системы и связи между ними.

Список использованных источников

1. Российский рынок облачных инфраструктурных сервисов: итоги 2023 г. // iKS-Consulting : аналитическое агентство. – [Электронный ресурс]. – URL: <http://survey.iksconsulting.ru/page44202947.html> (дата обращения: 12.05.2026).
2. Облачные сервисы (рынок России) // TAdviser : портал корпоративных ИТ-решений. – [Электронный ресурс]. – URL: [https://www.tadviser.ru/index.php/Статья:Облачные_сервисы_\(рынок_России\)](https://www.tadviser.ru/index.php/Статья:Облачные_сервисы_(рынок_России)) (дата обращения: 12.05.2026).
3. 10 фактов об утечках данных в России в 2024 году // Habr : блог компании SearchInform. – [Электронный ресурс]. – URL: <https://habr.com/ru/companies/searchinform/articles/1012214/> (дата обращения: 12.05.2026).
4. Облачный провайдинг 2018–2022: экономика, технологии, рынок // iKS-Consulting : аналитическое агентство. – [Электронный ресурс]. – URL: <http://survey.iksconsulting.ru/page4680670.html> (дата обращения: 13.05.2026).
5. Безопасность облачной инфраструктуры: как защитить свои данные в облаке // ITGLOBAL : официальный блог. – [Электронный ресурс]. – URL: <https://itglobal.com/ru-uz/company/blog/bezopasnost-oblachnoj-infrastruktury-kak-zashhitit-svoi-dannye-v-oblake/> (дата обращения: 13.05.2026).
6. Шифрование данных в облаке: стандарты и практики // Cloud4Y : официальный блог. – [Электронный ресурс]. – URL: <https://www.cloud4y.ru/en/blog/data-encryption-in-the-cloud-standards-and-practices/> (дата обращения: 15.05.2026).
7. The NIST Definition of Cloud Computing // National Institute of Standards and Technology : официальный сайт. – [Электронный ресурс]. – URL: <https://nvlpubs.nist.gov/nistpubs/legacy/sp/nistspecialpublication800-145.pdf> (дата обращения: 15.05.2026).

8. Recommendation for Block Cipher Modes of Operation: Galois/Counter Mode (GCM) and GMAC // National Institute of Standards and Technology : официальный сайт. – [Электронный ресурс]. – URL: <https://nvlpubs.nist.gov/nistpubs/legacy/sp/nistspecialpublication800-38d.pdf> (дата обращения: 16.05.2026).

9. Public-Key Cryptography Standards (PKCS) #8: Private-Key Information Syntax Specification Version 1.2 // RFC Editor : официальный репозиторий. – [Электронный ресурс]. – URL: <https://www.rfc-editor.org/rfc/rfc5208> (дата обращения: 18.05.2026).

10. Password Storage Cheat Sheet // OWASP : официальная документация. – [Электронный ресурс]. – URL: https://cheatsheetseries.owasp.org/cheatsheets/Password_Storage_Cheat_Sheet.html (дата обращения: 18.05.2026).

11. UML – диаграмма вариантов использования (use case diagram) // Habr : публикация. – [Электронный ресурс]. – URL: <https://habr.com/ru/articles/47940/> (дата обращения: 17.05.2026).

12. UML: обзор основных типов диаграмм, диаграмма компонентов. Часть 2 // Habr : публикация. – [Электронный ресурс]. – URL: <https://habr.com/ru/articles/756552/> (дата обращения: 17.05.2026).

13. Концептуальная модель // mlkv.ru : учебные материалы по базам данных. – [Электронный ресурс]. – URL: <https://mlkv.ru/sample-page/материалы-по-базам-данных/концептуальная-модель/> (дата обращения: 18.05.2026).

14. Верников, Г. Основы методологии IDEF1X // Корпоративный менеджмент : проект. – [Электронный ресурс]. – URL: <https://www.cfin.ru/vernikov/idef/idef1x.shtml> (дата обращения: 20.05.2026).

15. Моделирование данных: обзор // Habr : публикация. – [Электронный ресурс]. – URL: <https://habr.com/ru/articles/556790/> (дата обращения: 18.05.2026).

16. Моделирование данных: что это и зачем нужно // Яндекс Практикум : образовательный портал. – [Электронный ресурс]. – URL: <https://practicum.yandex.ru/blog/chto-takoe-modelirovanie-dannyh-i-kak-ego-ispolzovat/> (дата обращения: 05.06.2026).
17. Юнит-тестирование для чайников // Habr : публикация. – [Электронный ресурс]. – URL: <https://habr.com/ru/articles/169381/> (дата обращения: 05.06.2026).
18. Фундаментальная теория тестирования // Habr : публикация. – [Электронный ресурс]. – URL: <https://habr.com/ru/articles/549054/> (дата обращения: 05.06.2026).
19. Иванов, П.В. Управление проектами : учебное пособие / П.В. Иванов. – Ростов н/Д : Феникс, 2018. – 102 с.
20. Создаём диаграмму Ганта // Unisender : официальный блог. – [Электронный ресурс]. – URL: <https://www.unisender.com/ru/blog/chto-takoe-diagramma-ganta-i-kak-ee-sdelat/> (дата обращения: 05.06.2026).

Приложение А
(справочное)
Глоссарий

Серверное приложение – это программное обеспечение, которое работает на сервере и предоставляет различные сервисы или функциональность для клиентских приложений. Серверные приложения обычно используются в клиент-серверной архитектуре.

API (Application Programming Interface) – это набор правил и протоколов, которые позволяют различным программным приложениям взаимодействовать друг с другом.

REST (Representational State Transfer) – это стиль архитектуры программного обеспечения для распределённых систем, который, как правило, используется для построения веб-служб.

Микросервисная архитектура – это подход к проектированию программных систем, при котором приложение разбивается на набор небольших независимых сервисов, каждый из которых выполняет одну конкретную функцию и взаимодействует с остальными через API.

Слой представления данных – это первый и самый верхний уровень, который присутствует в приложении. Этот уровень представляет собой UI, то есть представление контента конечному пользователю через интерфейс.

HTTP-запрос – это запрос, который отправляется от клиента к веб-серверу для получения или отправки данных.

Контроллер – это компонент приложения, который обрабатывает входящие запросы от пользователей или других систем.

Сервис – это слой бизнес-логики приложения, реализующий основные операции и правила обработки данных.

Бизнес-логика – это набор правил, необходимых для запуска приложения в соответствии с руководящими принципами, установленными организацией.

Репозиторий – это компонент, отвечающий за взаимодействие с хранилищем данных.

JWT (JSON Web Token) – это открытый стандарт (RFC 7519) для безопасной передачи данных между сторонами в виде JSON-объекта.

Используется для аутентификации и авторизации пользователей в веб-приложениях.

Зашифрование – зависящий от криптографического ключа процесс обратимого преобразования открытого текста в зашифрованный текст.

Расшифрование – зависящий от криптографического ключа процесс преобразования зашифрованного текста в соответствующий ему открытый текст.

Шифрование – термин, объединяющий термины «зашифрование» и «расшифрование».

AES-GCM – это режим работы симметричного блочного шифра AES, обеспечивающий одновременно шифрование данных и их аутентификацию с помощью кода аутентификации сообщения (MAC).

RSA – это асимметричный криптографический алгоритм, использующий пару ключей – открытый и закрытый. Применяется для шифрования данных и цифровой подписи.

Аудит – это процесс систематической записи и проверки действий пользователей и событий в информационной системе с целью обеспечения безопасности и подотчётности.

Docker – это платформа для разработки, доставки и запуска приложений в изолированных контейнерах, обеспечивающая единообразие среды выполнения.

API Gateway – это сервер, выступающий в роли единой точки входа для всех клиентских запросов к микросервисной системе. Обеспечивает маршрутизацию, аутентификацию и балансировку нагрузки.

MinIO – это высокопроизводительное объектное хранилище с открытым исходным кодом, совместимое с Amazon S3 API. Используется для хранения зашифрованных файлов в системе CloudSecure.

Приложение Б
(рекомендуемое)
Спецификация атрибутов таблиц

Таблица Б.1 – Спецификация атрибутов таблицы Users

№ п/п	Наименование	Описание	Тип данных	Размер	Роль	Обязательный	Ограничения	Значение по умолчанию
1	Id	Идентификатор пользователя	UUID	16 байт	PK	Да		
2	Email	Email пользователя	VARCHAR	255		Да	UNIQUE	
3	PasswordHash	Хеш пароля (BCrypt)	VARCHAR	255		Да		
4	Role	Роль пользователя	VARCHAR	50		Да		
5	CreatedAt	Дата создания записи	TIMESTAMP	8 байт		Да		now()
6	IsBlocked	Флаг блокировки аккаунта	BOOLEAN	1 байт		Да		false
7	BlockedAt	Дата и время блокировки	TIMESTAMP	8 байт		Нет		
8	BlockReason	Причина блокировки	VARCHAR	500		Нет		
9	LastLoginAt	Время последнего успешного входа	TIMESTAMP	8 байт		Нет		
10	FailedLoginCount	Количество неудачных попыток входа	INTEGER	4 байт		Нет		0

Таблица Б.2 – Спецификация атрибутов таблицы FileMetadata

№ п/п	Наименование	Описание	Тип данных	Размер	Роль	Обязательный	Ограничения	Значение по умолчанию
1	Id	Идентификатор файла	UUID	16 байт	PK	Да		
2	OwnerId	Идентификатор владельца файла	UUID	16 байт		Да		
3	BlobPath	Путь к файлу в хранилище	VARCHAR	500		Нет		
4	Bucket	Имя бакета хранилища	VARCHAR	255		Нет		
5	ObjectKey	Ключ объекта в хранилище	VARCHAR	500		Нет		
6	Status	Статус файла	VARCHAR	50		Да		PendingUpload
7	EncryptedAesKey	Зашифрованный ключ AES	BYTEA	–		Да		
8	Iv	Вектор инициализации AES-GCM	BYTEA	12 байт		Да		
9	Tag	Тег аутентификации AES-GCM	BYTEA	16 байт		Да		
10	CryptoAlgorithm	Алгоритм шифрования	VARCHAR	50		Нет		AES-256-GCM
11	CryptoKeyId	Идентификатор криптографического ключа	VARCHAR	255		Нет		
12	OriginalFileName	Оригинальное имя файла	VARCHAR	255		Нет		
13	ContentType	MIME-тип содержимого	VARCHAR	100		Нет		
14	Size	Размер файла в байтах	BIGINT	8 байт		Нет		
15	CreatedAt	Дата и время загрузки файла	TIMESTAMP	8 байт		Да		now()

Таблица Б.3 – Спецификация атрибутов таблицы AuditLogs

№ п/п	Наименование	Описание	Тип данных	Размер	Роль	Обязательный	Ограничения	Значение по умолчанию
1	Id	Идентификатор записи аудита	UUID	16 байт	PK	Да		
2	UserId	Идентификатор пользователя-инициатора	UUID	16 байт	FK	Нет		
3	Action	Тип действия (Upload/Download/Login)	VARCHAR	100		Да		
4	FileId	Идентификатор файла	UUID	16 байт	FK	Нет	ON DELETE SET NULL	
5	Success	Успешность выполнения операции	BOOLEAN	1 байт		Да		
6	Details	Детальное описание события	VARCHAR	1000		Нет		
7	Service	Сервис-источник события	VARCHAR	100		Нет		
8	TargetUserId	Идентификатор целевого пользователя	UUID	16 байт		Нет		
9	TargetResourceType	Тип целевого ресурса	VARCHAR	100		Нет		
10	TargetResourceId	Идентификатор целевого ресурса	VARCHAR	255		Нет		
11	Ip	IP-адрес клиента	VARCHAR	45		Нет		
12	UserAgent	User-Agent клиента	VARCHAR	500		Нет		
13	CorrelationId	Идентификатор корреляции запроса	VARCHAR	100		Нет		
14	MetadataJson	Дополнительные данные в формате JSON	TEXT	–		Нет		
15	CreatedAt	Дата и время создания записи	TIMESTAMP	8 байт		Да		now()

Таблица Б.4 – Спецификация атрибутов таблицы UserKeyPairs

№ п/п	Наименование	Описание	Тип данных	Размер	Роль	Обязательный	Ограничения	Значение по умолчанию
1	Id	Идентификатор ключевой пары	UUID	16 байт	PK	Да		
2	UserId	Ссылка на пользователя	UUID	16 байт	FK	Да	UNIQUE, ON DELETE CASCADE	
3	PublicKeyPem	Публичный ключ RSA в формате PEM	TEXT	–		Да		
4	PrivateKeyEncrypted	Зашифрованный приватный ключ	BYTEA	–		Да		
5	Nonce	Nonce для AES-GCM	BYTEA	12 байт		Да		
6	Tag	Tag аутентификации AES-GCM	BYTEA	16 байт		Да		
7	CreatedAt	Дата создания ключевой пары	TIMESTAMP	8 байт		Да		now()

Приложение В
(обязательное)
Объекты обращения к базам данных

Листинг В.1 – Объекты пользователей (User)

```
public class User
{
    public Guid Id { get; set; }
    public string Email { get; set; } = default!;
    public string PasswordHash { get; set; } = default!;
    public string Role { get; set; } = "User";
    public DateTime CreatedAt { get; set; } = DateTime.UtcNow;
    public bool IsBlocked { get; set; } = false;
    public DateTime? BlockedAt { get; set; }
    public string? BlockReason { get; set; }
    public DateTime? LastLoginAt { get; set; }
    public int FailedLoginCount { get; set; } = 0;
    public UserKeyPair? KeyPair { get; set; }
}
```

Листинг В.2 – Объекты ключевых пар (UserKeyPair)

```
public class UserKeyPair
{
    public Guid Id { get; set; }
    public Guid UserId { get; set; }
    public string PublicKeyPem { get; set; } = default!;
    public byte[] PrivateKeyEncrypted { get; set; } = default!;
    public byte[] Nonce { get; set; } = default!;
    public byte[] Tag { get; set; } = default!;
    public DateTime CreatedAt { get; set; } = DateTime.UtcNow;
    public User User { get; set; } = default!;
}
```

Листинг В.3 – Объекты метаданных файлов (FileMetadata)

```
public class FileMetadata
{
    public Guid Id { get; set; }
    public Guid OwnerId { get; set; }
    public string? BlobPath { get; set; }
    public string? Bucket { get; set; }
    public string? ObjectKey { get; set; }
    public FileStatus Status { get; set; } =
FileStatus.PendingUpload;
    public byte[] EncryptedAesKey { get; set; } = default!;
    public byte[] Iv { get; set; } = default!;
    public byte[] Tag { get; set; } = default!;
    public string CryptoAlgorithm { get; set; } = "AES-256-GCM";
    public string? CryptoKeyId { get; set; }
    public string? OriginalFileName { get; set; }
    public string? ContentType { get; set; }
    public long? Size { get; set; }
    public DateTime CreatedAt { get; set; } = DateTime.UtcNow;
}

public enum FileStatus
{
    PendingUpload,
}
```

```
        Uploaded,  
        Deleted  
    }
```

Листинг В.4 – Объекты журнала аудита (AuditLog)

```
public class AuditLog  
{  
    public Guid Id { get; set; }  
    public Guid? UserId { get; set; }  
    public string Action { get; set; } = default!;  
    public Guid? FileId { get; set; }  
    public bool Success { get; set; }  
    public string? Details { get; set; }  
    public string? Service { get; set; }  
    public Guid? TargetUserId { get; set; }  
    public string? TargetResourceType { get; set; }  
    public string? TargetResourceId { get; set; }  
    public string? Ip { get; set; }  
    public string? UserAgent { get; set; }  
    public string? CorrelationId { get; set; }  
    public string? MetadataJson { get; set; }  
    public DateTime CreatedAt { get; set; } = DateTime.UtcNow;  
}
```

Листинг В.5 – Объекты событий аудита (AuditEventDto)

```
public record AuditEventDto(  
    Guid EventId,  
    DateTime TimestampUtc,  
    string Service,  
    string Action,  
    Guid? ActorUserId,  
    Guid? TargetUserId,  
    string? TargetResourceType,  
    string? TargetResourceId,  
    bool Success,  
    string? Ip,  
    string? UserAgent,  
    string? CorrelationId,  
    string? Details,  
    string? MetadataJson);
```

Приложение Г
(обязательное)
SQL-скрипты

Листинг Г.1 – Скрипты создания таблиц

```
CREATE TABLE IF NOT EXISTS "Users" (  
    "Id" UUID PRIMARY KEY,  
    "Email" VARCHAR(255) NOT NULL UNIQUE,  
    "PasswordHash" VARCHAR(255) NOT NULL,  
    "Role" VARCHAR(50) NOT NULL,  
    "CreatedAt" TIMESTAMP NOT NULL DEFAULT NOW(),  
    "IsBlocked" BOOLEAN NOT NULL DEFAULT FALSE,  
    "BlockedAt" TIMESTAMP NULL,  
    "BlockReason" VARCHAR(500) NULL,  
    "LastLoginAt" TIMESTAMP NULL,  
    "FailedLoginCount" INTEGER DEFAULT 0  
);  
  
CREATE INDEX "IX_Users_Email" ON "Users"("Email");  
  
CREATE TABLE IF NOT EXISTS "UserKeyPairs" (  
    "Id" UUID PRIMARY KEY,  
    "UserId" UUID NOT NULL UNIQUE,  
    "PublicKeyPem" TEXT NOT NULL,  
    "PrivateKeyEncrypted" BYTEA NOT NULL,  
    "Nonce" BYTEA NOT NULL,  
    "Tag" BYTEA NOT NULL,  
    "CreatedAt" TIMESTAMP NOT NULL DEFAULT NOW(),  
    CONSTRAINT "FK_UserKeyPairs_Users" FOREIGN KEY ("UserId")  
        REFERENCES "Users"("Id") ON DELETE CASCADE  
);  
  
CREATE INDEX "IX_UserKeyPairs_UserId" ON "UserKeyPairs"("UserId");  
  
CREATE TABLE IF NOT EXISTS "FileMetadata" (  
    "Id" UUID PRIMARY KEY,  
    "OwnerId" UUID NOT NULL,  
    "BlobPath" VARCHAR(500) NULL,  
    "Bucket" VARCHAR(255) NULL,  
    "ObjectKey" VARCHAR(500) NULL,  
    "Status" VARCHAR(50) NOT NULL DEFAULT 'PendingUpload',  
    "EncryptedAesKey" BYTEA NOT NULL,  
    "Iv" BYTEA NOT NULL,  
    "Tag" BYTEA NOT NULL,  
    "CryptoAlgorithm" VARCHAR(50) DEFAULT 'AES-256-GCM',  
    "CryptoKeyId" VARCHAR(255) NULL,  
    "OriginalFileName" VARCHAR(255) NULL,  
    "ContentType" VARCHAR(100) NULL,  
    "Size" BIGINT NULL,  
    "CreatedAt" TIMESTAMP NOT NULL DEFAULT NOW()  
);  
  
CREATE INDEX "IX_FileMetadata_OwnerId" ON  
"FileMetadata"("OwnerId");  
CREATE INDEX "IX_FileMetadata_Status" ON "FileMetadata"("Status");
```

```

CREATE INDEX "IX_FileMetadata_CreatedAt" ON
"FileMetadata"("CreatedAt");

CREATE TABLE IF NOT EXISTS "AuditLogs" (
    "Id" UUID PRIMARY KEY,
    "UserId" UUID NULL,
    "Action" VARCHAR(100) NOT NULL,
    "FileId" UUID NULL,
    "Success" BOOLEAN NOT NULL,
    "Details" VARCHAR(1000) NULL,
    "Service" VARCHAR(100) NULL,
    "TargetUserId" UUID NULL,
    "TargetResourceType" VARCHAR(100) NULL,
    "TargetResourceId" VARCHAR(255) NULL,
    "Ip" VARCHAR(45) NULL,
    "UserAgent" VARCHAR(500) NULL,
    "CorrelationId" VARCHAR(100) NULL,
    "MetadataJson" TEXT NULL,
    "CreatedAt" TIMESTAMP NOT NULL DEFAULT NOW(),
    CONSTRAINT "FK_AuditLogs_FileMetadata" FOREIGN KEY ("FileId")
        REFERENCES "FileMetadata"("Id") ON DELETE SET NULL
);

CREATE INDEX "IX_AuditLogs_CreatedAt" ON "AuditLogs"("CreatedAt");
CREATE INDEX "IX_AuditLogs_UserId" ON "AuditLogs"("UserId");
CREATE INDEX "IX_AuditLogs_Action" ON "AuditLogs"("Action");
CREATE INDEX "IX_AuditLogs_Success" ON "AuditLogs"("Success");
CREATE INDEX "IX_AuditLogs_FileId" ON "AuditLogs"("FileId");

```

Листинг Г.2 – Скрипты вставки данных

```

INSERT INTO "Users" ("Id", "Email", "PasswordHash", "Role",
"CreatedAt", "IsBlocked")
VALUES (:id, :email, :password_hash, :role, :created_at,
:is_blocked)
RETURNING "Id";

INSERT INTO "UserKeyPairs" ("Id", "UserId", "PublicKeyPem",
"PrivateKeyEncrypted", "Nonce", "Tag")
VALUES (:id, :user_id, :public_key_pem, :private_key_encrypted,
:nonce, :tag)
RETURNING "Id";

INSERT INTO "FileMetadata" ("Id", "OwnerId", "Bucket",
"ObjectKey", "Status", "EncryptedAesKey", "Iv", "Tag",
"OriginalFileName", "ContentType", "Size")
VALUES (:id, :owner_id, :bucket, :object_key, :status,
:encrypted_aes_key, :iv, :tag, :original_file_name, :content_type,
:size)
RETURNING "Id";

```

```
INSERT INTO "AuditLogs" ("Id", "UserId", "Action", "FileId",  
"Success", "Details", "Service", "Ip", "UserAgent",  
"CorrelationId", "MetadataJson")  
VALUES (:id, :user_id, :action, :file_id, :success, :details,  
:service, :ip, :user_agent, :correlation_id, :metadata_json)  
RETURNING "Id";
```

Листинг Г.3 – Скрипты выборки данных

```
SELECT * FROM "Users"  
WHERE "Email" = :email;
```

```
SELECT * FROM "UserKeyPairs"  
WHERE "UserId" = :user_id;
```

```
SELECT * FROM "FileMetadata"  
WHERE "OwnerId" = :owner_id  
ORDER BY "CreatedAt" DESC  
LIMIT 50;
```

```
SELECT * FROM "AuditLogs"  
WHERE "UserId" = :user_id  
    AND "CreatedAt" BETWEEN :from_date AND :to_date  
ORDER BY "CreatedAt" DESC  
LIMIT 200;
```

Приложение Д
(обязательное)
Структуры слоя представления данных

Листинг Д.1 – Слой представления AuthService (AuthController)

```
[ApiController]
[Route("auth")]
public class AuthController : ControllerBase
{
    [HttpPost("register")]
    public async Task<IActionResult> Register([FromBody]
RegisterRequest req)
    { ... }
    [HttpPost("login")]
    public async Task<IActionResult> Login([FromBody] LoginRequest
req)
    { ... }
    [HttpGet("public-key")]
    public async Task<IActionResult> GetPublicKey()
    { ... }
}
```

Листинг Д.2 – Слой представления AuthService (AdminUsersController)

```
[ApiController]
[Route("admin/users")]
[Authorize(Policy = "AdminOnly")]
public class AdminUsersController : ControllerBase
{
    [HttpGet]
    public async Task<IActionResult> GetUsers(
        [FromQuery] string? query, [FromQuery] string? role,
        [FromQuery] bool? isBlocked, [FromQuery] int page,
        [FromQuery] int pageSize, CancellationToken ct)
    { ... }
    [HttpGet("{id:guid}")]
    public async Task<IActionResult> GetUser(Guid id,
CancellationToken ct)
    { ... }
    [HttpPatch("{id:guid}/role")]
    public async Task<IActionResult> ChangeRole(
        Guid id, [FromBody] ChangeUserRoleRequest req,
CancellationToken ct)
    { ... }
    [HttpPatch("{id:guid}/block")]
    public async Task<IActionResult> SetBlocked(
        Guid id, [FromBody] SetUserBlockedRequest req,
CancellationToken ct)
    { ... }
}
```

Листинг Д.3 – Слой представления CryptFileTransferService (FilesController)

```
[ApiController]
[Route("files")]
public class FilesController : ControllerBase
{
    [HttpPost("upload")]
    [Authorize]
    [RequestSizeLimit(long.MaxValue)]
```

```

        public async Task<IActionResult> Upload(
            [FromForm] UploadFileRequest req, CancellationToken ct)
        { ... }
        [HttpGet("{id:guid}")]
        [Authorize]
        public async Task<IActionResult> Get(Guid id,
            CancellationToken ct)
        { ... }
        [HttpGet("{id:guid}/download")]
        [Authorize]
        public async Task<IActionResult> Download(Guid id,
            CancellationToken ct)
        { ... }
    }

```

Листинг Д.4 – Слой представления LogsService (AuditController)

```

[ApiController]
[Route("audit")]
public class AuditController : ControllerBase
{
    [HttpPost]
    [AllowAnonymous]
    public async Task<IActionResult> Write(
        [FromBody] AuditEventDto evt, CancellationToken ct)
    { ... }
    [HttpGet]
    [Authorize(Policy = "AdminOnly")]
    public async Task<IActionResult> Get(
        [FromQuery] DateTime? from, [FromQuery] DateTime? to,
        [FromQuery] Guid? actorUserId, [FromQuery] string? action,
        [FromQuery] bool? success, [FromQuery] int page,
        [FromQuery] int pageSize, CancellationToken ct)
    { ... }
    [HttpGet("stats")]
    [Authorize(Policy = "AdminOnly")]
    public async Task<IActionResult> Stats(
        [FromQuery] DateTime? from, [FromQuery] DateTime? to,
        CancellationToken ct)
    { ... }}

```

Листинг Д.5 – Слой представления LogsService (LogsController)

```

[ApiController]
[Route("logs")]
public class LogsController : ControllerBase
{
    [HttpPost]
    public async Task<IActionResult> WriteLog([FromBody] AuditLog
log)
    { ... }
    [HttpGet]
    public async Task<IActionResult> GetAll()
    { ... }
}

```

Листинг Д.6 – Слой представления ApiGateway (AuthProxyController)

```
[ApiController]
[Route("auth")]
public class AuthProxyController : ControllerBase
{
    [HttpPost("register")]
    public async Task<IActionResult> Register([FromBody] object
body)
    {
        var client =
_httpClientFactory.CreateClient("AuthService");
        using var resp = await
client.PostAsJsonAsync("/auth/register", body);
        var content = await resp.Content.ReadAsStringAsync();
        return StatusCode((int)resp.StatusCode, content);
    }
    [HttpPost("login")]
    public async Task<IActionResult> Login([FromBody] object body)
    {
        var client =
_httpClientFactory.CreateClient("AuthService");
        using var resp = await
client.PostAsJsonAsync("/auth/login", body);
        var content = await resp.Content.ReadAsStringAsync();
        return StatusCode((int)resp.StatusCode, content);
    }
}
```

Листинг Д.7 – Конфигурация маршрутов обратного прокси ApiGateway (appsettings.json)

```
"ReverseProxy": {
  "Routes": {
    "auth_route": { "ClusterId": "auth_cluster",
                    "Match": { "Path": "/auth/{**catch-all}" } },
    "files_route": { "ClusterId": "files_cluster",
                     "Match": { "Path": "/files/{**catch-all}" }
  },
  "logs_route": { "ClusterId": "logs_cluster",
                  "Match": { "Path": "/logs/{**catch-all}" } },
  "admin_route": { "ClusterId": "auth_cluster",
                   "AuthorizationPolicy": "AdminOnly",
                   "Match": { "Path": "/admin/{**catch-all}" } },
  "audit_get_route": { "ClusterId": "logs_cluster",
                       "AuthorizationPolicy": "AdminOnly",
                       "Match": { "Path": "/audit/{**catch-
all}",
                                "Methods": ["GET"] } },
  "audit_post_route": { "ClusterId": "logs_cluster",
                        "Match": { "Path": "/audit/{**catch-
all}",
                                "Methods": ["POST"] } }
  }
}
```

Приложение Е
(обязательное)
Структуры слоя бизнес-логики

Листинг Е.1 – Слой бизнес-логики AuthService (IJwtService, JwtService)

```
public interface IJwtService
{
    string GenerateToken(User user, out DateTime expiresAt);
}

public class JwtService : IJwtService
{
    private readonly IConfiguration _config;

    public JwtService(IConfiguration config)
    {
        _config = config;
    }

    public string GenerateToken(User user, out DateTime expiresAt)
    {
        var claims = new List<Claim>
        {
            new(JwtRegisteredClaimNames.Sub, user.Id.ToString()),
            new(JwtRegisteredClaimNames.Email, user.Email),
            new(ClaimTypes.Role, user.Role)
        };
        var creds = new SigningCredentials(
            new SymmetricSecurityKey(Encoding.UTF8.GetBytes(key)),
            SecurityAlgorithms.HmacSha256);
        expiresAt = DateTime.UtcNow.AddMinutes(expiryMinutes);
        var token = new JwtSecurityToken(
            issuer, audience, claims,
            notBefore: DateTime.UtcNow,
            expires: expiresAt,
            signingCredentials: creds);
        return new JwtSecurityTokenHandler().WriteToken(token);
    }
}
```

Листинг Е.2 – Слой бизнес-логики AuthService (ICryptoService, CryptoService)

```
public interface ICryptoService
{
    (string publicPem, byte[] privateEncrypted, byte[] nonce,
    byte[] tag)
        GenerateAndProtectRsa(string masterKey);

    byte[] ProtectPrivateKey(byte[] privateKeyPkcs8, string
    masterKey,
        out byte[] nonce, out byte[] tag);

    byte[] UnprotectPrivateKey(byte[] encrypted, string masterKey,
        byte[] nonce, byte[] tag);
}

public class CryptoService : ICryptoService
{
}
```

```

public (string, byte[], byte[], byte[])
    GenerateAndProtectRsa(string masterKey)
{
    using var rsa = RSA.Create(2048);
    string publicPem = rsa.ExportSubjectPublicKeyInfoPem();
    byte[] privatePkcs8 = rsa.ExportPkcs8PrivateKey();
    var encrypted = ProtectPrivateKey(
        privatePkcs8, masterKey, out var nonce, out var tag);
    CryptographicOperations.ZeroMemory(privatePkcs8);
    return (publicPem, encrypted, nonce, tag);
}

public byte[] ProtectPrivateKey(byte[] privateKeyPkcs8,
    string masterKey, out byte[] nonce, out byte[] tag)
{
    byte[] key = DeriveKey(masterKey);
    nonce = RandomNumberGenerator.GetBytes(12);
    tag = new byte[16];
    byte[] ciphertext = new byte[privateKeyPkcs8.Length];
    using var aesGcm = new AesGcm(key);
    aesGcm.Encrypt(nonce, privateKeyPkcs8, ciphertext, tag);
    CryptographicOperations.ZeroMemory(key);
    return ciphertext;
}

public byte[] UnprotectPrivateKey(byte[] encrypted,
    string masterKey, byte[] nonce, byte[] tag)
{
    byte[] key = DeriveKey(masterKey);
    byte[] plaintext = new byte[encrypted.Length];
    using var aesGcm = new AesGcm(key);
    aesGcm.Decrypt(nonce, encrypted, tag, plaintext);
    CryptographicOperations.ZeroMemory(key);
    return plaintext;
}

private static byte[] DeriveKey(string masterKey)
{
    using var sha = SHA256.Create();
    return sha.ComputeHash(Encoding.UTF8.GetBytes(masterKey));
}
}

```

Листинг Е.3 – Слой бизнес-логики CryptFileTransferService (IFileCryptoProvider)

```

public interface IFileCryptoProvider
{
    FileCryptoResult Encrypt(byte[] plaintext);
    byte[] Decrypt(byte[] ciphertext, FileCryptoMetadata
metadata);
}

public sealed record FileCryptoMetadata(
    byte[] EncryptedDataKey,

```

```

        byte[] Nonce,
        byte[] Tag,
        string Algorithm,
        string KeyId);

public sealed record FileCryptoResult(
    byte[] Ciphertext,
    FileCryptoMetadata Metadata);

```

Листинг Е.4 – Слой бизнес-логики CryptFileTransferService (AesGcmFileCryptoProvider)

```

public sealed class AesGcmFileCryptoProvider : IFileCryptoProvider
{
    private readonly byte[] _masterKey;

    public AesGcmFileCryptoProvider(IConfiguration config)
    {
        var master = config["MasterKey:Key"] ?? "dev-master-key-
change";
        using var sha = SHA256.Create();
        _masterKey =
sha.ComputeHash(Encoding.UTF8.GetBytes(master));
    }

    public FileCryptoResult Encrypt(byte[] plaintext)
    {
        var dataKey = RandomNumberGenerator.GetBytes(32);
        var nonce    = RandomNumberGenerator.GetBytes(12);
        var tag      = new byte[16];
        var ciphertext = new byte[plaintext.Length];
        using (var aes = new AesGcm(dataKey))
            aes.Encrypt(nonce, plaintext, ciphertext, tag);

        var wrappedNonce = RandomNumberGenerator.GetBytes(12);
        var wrappedTag    = new byte[16];
        var wrappedKey    = new byte[dataKey.Length];
        using (var wrapAes = new AesGcm(_masterKey))
            wrapAes.Encrypt(wrappedNonce, dataKey, wrappedKey,
wrappedTag);

        var encryptedDataKey = new byte[12 + 16 +
wrappedKey.Length];
        Buffer.BlockCopy(wrappedNonce, 0, encryptedDataKey, 0,
12);
        Buffer.BlockCopy(wrappedTag,    0, encryptedDataKey, 12,
16);
        Buffer.BlockCopy(wrappedKey,    0, encryptedDataKey, 28,
wrappedKey.Length);
        CryptographicOperations.ZeroMemory(dataKey);

        return new FileCryptoResult(ciphertext,
            new FileCryptoMetadata(encryptedDataKey, nonce, tag,
                Algorithm: "AES-256-GCM", KeyId: "master"));
    }
}

```

```

    }

    public byte[] Decrypt(byte[] ciphertext, FileCryptoMetadata
metadata)
    {
        var wrappedNonce = metadata.EncryptedDataKey.AsSpan(0,
12).ToArray();
        var wrappedTag    = metadata.EncryptedDataKey.AsSpan(12,
16).ToArray();
        var wrappedKey    =
metadata.EncryptedDataKey.AsSpan(28).ToArray();
        var dataKey = new byte[wrappedKey.Length];
        using (var wrapAes = new AesGcm(_masterKey))
            wrapAes.Decrypt(wrappedNonce, wrappedKey, wrappedTag,
dataKey);
        var plaintext = new byte[ciphertext.Length];
        using (var aes = new AesGcm(dataKey))
            aes.Decrypt(metadata.Nonce, ciphertext, metadata.Tag,
plaintext);
        CryptographicOperations.ZeroMemory(dataKey);
        return plaintext;
    }
}

```

Листинг Е.5 – Слой бизнес-логики CryptFileTransferService (IStorageAdapter)

```

public interface IStorageAdapter
{
    Task EnsureBucketExistsAsync(CancellationTokens ct);

    Task PutObjectAsync(string objectKey, Stream data,
        long size, string contentType, CancellationTokens ct);

    Task<Stream> GetObjectAsync(string objectKey,
CancellationTokens ct);

    Task<bool> ExistsAsync(string objectKey, CancellationTokens
ct);
}

```

Приложение Ж
(обязательное)
Структуры слоя обращения к базе данных

Листинг Ж.1 – Модель сущности User (AuthService)

```
public class User
{
    public Guid      Id                { get; set; }
    public string    Email              { get; set; } = default!;
    public string    PasswordHash      { get; set; } = default!;
    public string    Role               { get; set; } = "User";
    public DateTime  CreatedAt          { get; set; } =
DateTime.UtcNow;
    public bool      IsBlocked          { get; set; } = false;
    public DateTime? BlockedAt          { get; set; }
    public string?   BlockReason        { get; set; }
    public DateTime? LastLoginAt        { get; set; }
    public int       FailedLoginCount   { get; set; } = 0;

    public UserKeyPair? KeyPair         { get; set; }
}
```

Листинг Ж.2 – Модель сущности UserKeyPair (AuthService)

```
public class UserKeyPair
{
    public Guid      Id                { get; set; }
    public Guid      UserId             { get; set; }
    public string    PublicKeyPem       { get; set; } = default!;
    public byte[]    PrivateKeyEncrypted { get; set; } = default!;
    public byte[]    Nonce              { get; set; } = default!; //
AES-GCM nonce
    public byte[]    Tag                { get; set; } = default!; //
AES-GCM tag
    public DateTime  CreatedAt          { get; set; } =
DateTime.UtcNow;

    public User? User { get; set; }
}
```

Листинг Ж.3 – Контекст базы данных AuthDbContext (AuthService)

```
public class AuthDbContext : DbContext
{
    public AuthDbContext(DbContextOptions<AuthDbContext> options)
        : base(options) { }

    public DbSet<User> Users => Set<User>();
    public DbSet<UserKeyPair> UserKeyPairs => Set<UserKeyPair>();

    protected override void OnModelCreating(ModelBuilder
modelBuilder)
    {
        modelBuilder.Entity<User>(e =>
        {
            e.HasKey(x => x.Id);
            e.HasIndex(x => x.Email).IsUnique();
            e.Property(x => x.Email).IsRequired();
            e.Property(x => x.PasswordHash).IsRequired();
            e.Property(x => x.Role).IsRequired();
        });
    }
}
```

```

        e.HasOne(x => x.KeyPair)
        .WithOne(k => k.User!)
        .HasForeignKey<UserKeyPair>(k => k.UserId)
        .OnDelete(DeleteBehavior.Cascade);
    });

modelBuilder.Entity<UserKeyPair>(e =>
{
    e.HasKey(x => x.Id);
    e.Property(x => x.PublicKeyPem).IsRequired();
    e.Property(x => x.PrivateKeyEncrypted).IsRequired();
    e.Property(x => x.Nonce).IsRequired();
    e.Property(x => x.Tag).IsRequired();
});
}
}

```

Листинг Ж.4 – Модель сущности FileMetadata (CryptFileTransferService)

```

public enum FileStatus
{
    PendingUpload = 0,
    Uploaded      = 1,
    Deleted       = 2
}

public class FileMetadata
{
    public Guid      Id { get; set; }
    public Guid      OwnerUserId { get; set; }
    public string    BlobPath { get; set; } =
default!;
    public string    Bucket { get; set; } =
default!;
    public string    ObjectKey { get; set; } =
default!;
    public FileStatus Status { get; set; } =
FileStatus.PendingUpload;

    public byte[]    EncryptedAesKey { get; set; } =
Array.Empty<byte>();
    public byte[]    Iv { get; set; } =
Array.Empty<byte>();
    public byte[]    Tag { get; set; } =
Array.Empty<byte>();
    public string    CryptoAlgorithm { get; set; } =
default!;
    public string    CryptoKeyId { get; set; } =
default!;

    public string    OriginalFileName { get; set; } =
default!;
    public string    OriginalContentType { get; set; } =
"application/octet-stream";
}

```

```

        public string ContentType { get; set; } =
"application/octet-stream";
        public long Size { get; set; }
        public DateTime CreatedAt { get; set; } =
DateTime.UtcNow;
    }

```

Листинг Ж.5 – Контекст базы данных SecureFileDbContext (CryptFileTransferService)

```

public class SecureFileDbContext : DbContext
{
    public
SecureFileDbContext(DbContextOptions<SecureFileDbContext> options)
        : base(options) { }

    public DbSet<FileMetadata> Files => Set<FileMetadata>();

    protected override void OnModelCreating(ModelBuilder
modelBuilder)
    {
        modelBuilder.Entity<FileMetadata>(e =>
        {
            e.HasKey(x => x.Id);
            e.Property(x => x.BlobPath).IsRequired();
            e.Property(x => x.Bucket).IsRequired();
            e.Property(x => x.ObjectKey).IsRequired();
            e.Property(x => x.Status).IsRequired();
            e.Property(x => x.OwnerUserId).IsRequired();
            e.Property(x => x.CreatedAt)
                .HasDefaultValueSql("now() at time zone 'utc'");
            e.Property(x => x.EncryptedAesKey).IsRequired();
            e.Property(x => x.Iv).IsRequired();
            e.Property(x => x.Tag).IsRequired();
            e.Property(x => x.CryptoAlgorithm).IsRequired();
            e.Property(x => x.CryptoKeyId).IsRequired();
            e.Property(x => x.OriginalFileName).IsRequired();
            e.Property(x => x.OriginalContentType).IsRequired();
        });
    }
}

```

Листинг Ж.6 – Модель сущности AuditLog (LogsService)

```

public class AuditLog
{
    public Guid Id { get; set; }
    public Guid? UserId { get; set; }
    public string Action { get; set; } = default!;
    public Guid? FileId { get; set; }
    public bool Success { get; set; }
    public string? Details { get; set; }
    public string? Service { get; set; }
    public Guid? TargetUserId { get; set; }
    public string? TargetResourceType { get; set; }
    public string? TargetResourceId { get; set; }
}

```

```

        public string? Ip { get; set; }
        public string? UserAgent { get; set; }
        public string? CorrelationId { get; set; }
        public string? MetadataJson { get; set; }
        public DateTime CreatedAt { get; set; } =
DateTime.UtcNow;
    }

```

Листинг Ж.7 – Контекст базы данных LogsDbContext (LogsService)

```

public class LogsDbContext : DbContext
{
    public LogsDbContext(DbContextOptions<LogsDbContext> options)
        : base(options) { }

    public DbSet<AuditLog> AuditLogs => Set<AuditLog>();

    protected override void OnModelCreating(ModelBuilder
modelBuilder)
    {
        modelBuilder.Entity<AuditLog>(e =>
        {
            e.HasKey(x => x.Id);
            e.Property(x => x.Action).IsRequired();
            e.Property(x => x.Success).IsRequired();
            e.Property(x => x.CreatedAt)
                .HasDefaultValueSql("now() at time zone 'utc'");

            e.HasIndex(x => x.CreatedAt);
            e.HasIndex(x => x.UserId);
            e.HasIndex(x => x.Action);
            e.HasIndex(x => x.Success);
        });
    }
}

```

Приложение И
(рекомендуемое)
Листинг модульных тестов

Листинг И.1 – Файл FilesControllerTests.cs – фабрика тестового веб-приложения

```
public class TestWebApplicationFactory :
WebApplicationFactory<Program>
{
    public const string JwtKey    = "your-super-secret-jwt-key-
here-32+chars";
    public const string Issuer    = "CloudSecure";
    public const string Audience = "CloudSecureUsers";

    protected override void ConfigureWebHost(IWebHostBuilder
builder)
    {
        builder.UseEnvironment("Testing");

        builder.ConfigureAppConfiguration((_, cfg) =>
        {
            cfg.Sources.Clear();
            cfg.AddInMemoryCollection(new Dictionary<string,
string?>
            {
                ["Jwt:Key"]           = JwtKey,
                ["Jwt:Issuer"]        = Issuer,
                ["Jwt:Audience"]      = Audience,
                ["Jwt:ExpiryMinutes"] = "60",
                ["MasterKey:Key"]     = "test-master-key-change-
in-prod!!!",
                ["Storage:Bucket"]    = "test-bucket",
                ["ConnectionStrings:DefaultConnection"] = "Data
Source=:memory:"
            });
        });

        builder.ConfigureServices(services =>
        {
            // Заменить реальный DbContext на In-Memory
            var dbDesc = services.SingleOrDefault(
                d => d.ServiceType ==
typeof(DbContextOptions<SecureFileDbContext>));
            if (dbDesc is not null) services.Remove(dbDesc);
            services.AddDbContext<SecureFileDbContext>(opts =>
                opts.UseInMemoryDatabase("TestDb_Shared"));

            // Мок IStorageAdapter - хранит данные в памяти по
objectKey
            var storedObjects = new Dictionary<string, byte[]>();
            var storageMock = new Mock<IStorageAdapter>();
            storageMock
                .Setup(s => s.PutObjectAsync(
                    It.IsAny<string>(), It.IsAny<Stream>(),
                    It.IsAny<long>(), It.IsAny<string>(),
                    It.IsAny<CancellationToken>()))
                .Returns<string, Stream, long, string,
CancellationToken>(
```

```

        async (key, stream, _, _, _) =>
        {
            using var ms = new MemoryStream();
            await stream.CopyToAsync(ms);
            lock (storedObjects) storedObjects[key] =
ms.ToArray();
        });
        storageMock
            .Setup(s => s.GetObjectAsync(
                It.IsAny<string>(),
                It.IsAny<CancellationTokens>()))
            .Returns<string, CancellationTokens>((key, _) =>
            {
                lock (storedObjects)
                {
                    var data = storedObjects.TryGetValue(key,
out var bytes)
                        ? bytes : Array.Empty<byte>();
                    return Task.FromResult<Stream>(new
MemoryStream(data));
                }
            });
        var storageDesc = services.SingleOrDefault(
            d => d.ServiceType == typeof(IAStorageAdapter));
        if (storageDesc is not null)
services.Remove(storageDesc);
        services.AddSingleton(storageMock.Object);

        // Mock IAuditPublisher
        var auditMock = new Mock<IAuditPublisher>();
        auditMock.Setup(a => a.PublishAsync(
            It.IsAny<object>(),
            It.IsAny<CancellationTokens>()))
            .Returns(Task.CompletedTask);
        var auditDesc = services.SingleOrDefault(
            d => d.ServiceType == typeof(IAuditPublisher));
        if (auditDesc is not null) services.Remove(auditDesc);
        services.AddScoped(_ => auditMock.Object);
    });
}

public static string GenerateJwt(Guid userId)
{
    var key = new
SymmetricSecurityKey(Encoding.UTF8.GetBytes(JwtKey));
    var creds = new SigningCredentials(key,
SecurityAlgorithms.HmacSha256);
    var claims = new[]
    {
        new Claim(JwtRegisteredClaimNames.Sub,
userId.ToString()),
        new Claim(ClaimTypes.NameIdentifier,
userId.ToString()),
    }
}

```

```

        new Claim(JwtRegisteredClaimNames.Email,
"test@cloudsecure.ru"),
        new Claim(ClaimTypes.Role, "User")
    };
    var token = new JwtSecurityToken(
        Issuer, Audience, claims,
        notBefore: DateTime.UtcNow,
        expires:    DateTime.UtcNow.AddHours(1),
        signingCredentials: creds);
    return new JwtSecurityTokenHandler().WriteToken(token);
}
}

```

Листинг И.2 – Файл FilesControllerTests.cs – интеграционные тесты Upload/Download (TC-01-TC-06)

```

public class FilesControllerTests :
    IClassFixture<TestWebApplicationFactory>
{
    private HttpClient CreateAuthenticatedClient(Guid userId)
    {
        var client = _factory.CreateClient();
        client.DefaultRequestHeaders.Authorization =
            new AuthenticationHeaderValue("Bearer",
                TestWebApplicationFactory.GenerateJwt(userId));
        return client;
    }

    // TC-01: Загрузка файла без JWT-токена -> 401 Unauthorized
    [Fact(DisplayName = "TC-01: Загрузка файла без JWT-токена
возвращает 401")]
    public async Task TC01_Upload_WithoutAuth_Returns401()
    {
        var client = _factory.CreateClient();
        using var content = new MultipartFormDataContent();
        var fileContent = new
        ByteArrayContent(Encoding.UTF8.GetBytes("test-data"));
        fileContent.Headers.ContentType =
        MediaTypeHeaderValue.Parse("text/plain");
        content.Add(fileContent, "file", "test.txt");

        var response = await client.PostAsync("/files/upload",
        content);

        Assert.Equal(HttpStatusCode.Unauthorized,
        response.StatusCode);
    }

    // TC-02: Загрузка без файла (пустое тело) -> 400 Bad Request
    [Fact(DisplayName = "TC-02: Загрузка без файла возвращает
400")]
    public async Task TC02_Upload_WithoutFile_Returns400()
    {

```

```

        using var client =
CreateAuthenticatedClient(Guid.NewGuid());
        using var content = new MultipartFormDataContent();
        content.Add(new StringContent("dummy"), "dummy_field");
        var response = await client.PostAsync("/files/upload",
content);
        Assert.Equal(HttpStatusCode.BadRequest,
response.StatusCode);
        var body = await response.Content.ReadAsStringAsync();
        Assert.Contains("File is required", body);
    }

    // TC-03: Успешная загрузка файла -> 200 OK + { fileId }
    [Fact(DisplayName = "TC-03: Успешная загрузка файла возвращает
200 и fileId")]
    public async Task TC03_Upload_ValidFile_Returns200WithFileId()
    {
        using var client =
CreateAuthenticatedClient(Guid.NewGuid());
        using var content = new MultipartFormDataContent();
        var fileBytes = Encoding.UTF8.GetBytes("secure document
content");
        var fileContent = new ByteArrayContent(fileBytes);
        fileContent.Headers.ContentType =
MediaTypeHeaderValue.Parse("text/plain");
        content.Add(fileContent, "file", "document.txt");

        var response = await client.PostAsync("/files/upload",
content);

        Assert.Equal(HttpStatusCode.OK, response.StatusCode);
        var body = await response.Content.ReadAsStringAsync();
        Assert.Contains("fileId", body);
    }

    // TC-04: Получение метаданных чужого файла -> 403 Forbidden
    [Fact(DisplayName = "TC-04: Получение метаданных чужого файла
возвращает 403")]
    public async Task TC04_Get_OtherUsersFile_Returns403()
    {
        var ownerUserId = Guid.NewGuid();
        using var ownerClient =
CreateAuthenticatedClient(ownerUserId);
        using var uploadContent = new MultipartFormDataContent();
        var fileBytes = Encoding.UTF8.GetBytes("owner-secret-
data");
        var fileContent = new ByteArrayContent(fileBytes);
        fileContent.Headers.ContentType =
MediaTypeHeaderValue.Parse("text/plain");
        uploadContent.Add(fileContent, "file", "owner_file.txt");
        var uploadResp = await
ownerClient.PostAsync("/files/upload", uploadContent);

```

```

        var uploadBody = await
uploadResp.Content.ReadAsStringAsync();
        var fileId =
JsonSerializer.Deserialize<FileIdResponse>(uploadBody,
        new JsonSerializerOptions {
PropertyNameCaseInsensitive = true })!.FileId;

        using var attackerClient =
CreateAuthenticatedClient(Guid.NewGuid());
        var response = await
attackerClient.GetAsync($" /files/{fileId}");

        Assert.Equal(HttpStatusCode.Forbidden,
response.StatusCode);
    }

    // TC-05: Запрос несуществующего файла -> 404 Not Found
    [Fact(DisplayName = "TC-05: Запрос несуществующего файла
возвращает 404")]
    public async Task TC05_Get_NonExistentFile_Returns404()
    {
        using var client =
CreateAuthenticatedClient(Guid.NewGuid());
        var response = await
client.GetAsync($" /files/{Guid.NewGuid()}");
        Assert.Equal(HttpStatusCode.NotFound,
response.StatusCode);
    }

    // TC-06: Успешное скачивание собственного файла -> 200 OK +
содержимое
    [Fact(DisplayName = "TC-06: Успешное скачивание собственного
файла возвращает 200")]
    public async Task
TC06_Download_OwnFile_Returns200WithContent()
    {
        var userId = Guid.NewGuid();
        using var client = CreateAuthenticatedClient(userId);
        var originalContent =
Encoding.UTF8.GetBytes("confidential-report.pdf-content");
        using var uploadContent = new MultipartFormDataContent();
        var fileContent = new ByteArrayContent(originalContent);
        fileContent.Headers.ContentType =
        MediaTypeHeaderValue.Parse("application/octet-
stream");
        uploadContent.Add(fileContent, "file", "report.pdf");

        var uploadResp = await client.PostAsync("/files/upload",
uploadContent);
        var uploadBody = await
uploadResp.Content.ReadAsStringAsync();
        var fileId =
JsonSerializer.Deserialize<FileIdResponse>(uploadBody,

```

```

        new JsonSerializerOptions {
PropertyNamesCaseInsensitive = true })!.FileId;

        var downloadResp = await
client.GetAsync($"/files/{fileId}/download");

        Assert.Equal(HttpStatusCode.OK, downloadResp.StatusCode);
        var downloadedBytes = await
downloadResp.Content.ReadAsByteArrayAsync();
        Assert.True(downloadedBytes.Length > 0);
    }

    private record FileIdResponse(Guid FileId);
}

```

Листинг И.3 – Файл FilesControllerTests.cs – unit-тест AES-GCM и тесты производительности (TC-07, PERF-01-03)

```

// TC-07: AES-256-GCM Encrypt/Decrypt round-trip
[Fact(DisplayName = "TC-07: AES-256-GCM Encrypt/Decrypt
восстанавливает исходные данные")]
public async Task TC07_AesGcm_EncryptDecrypt_RoundTrip()
{
    var config = new ConfigurationBuilder()
        .AddInMemoryCollection(new Dictionary<string, string?>
        {
            ["MasterKey:Key"] = "test-master-key-change-in-
prod!!!"
        })
        .Build();
    var provider = new AesGcmFileCryptoProvider(config);
    var originalData = Encoding.UTF8.GetBytes(
        "CloudSecure - защищённая передача файлов");
    var result = provider.Encrypt(originalData);
    var decrypted = provider.Decrypt(result.Ciphertext,
result.Metadata);
    Assert.Equal(originalData, decrypted);
    Assert.Equal("AES-256-GCM", result.Metadata.Algorithm);
    Assert.Equal("master", result.Metadata.KeyId);
    await Task.CompletedTask;
}

// PERF-01: Измерение времени шифрования и коэффициента
замедления
[Theory(DisplayName = "PERF-01: Время шифрования при разных
размерах данных")]
[InlineData(1024)] // 1 KB
[InlineData(1024 * 1024)] // 1 MB
[InlineData(10 * 1024 * 1024)] // 10 MB
public void PERF01_Encrypt_Performance_Measurement(int
dataSizeBytes)
{
    var provider = BuildProvider();
    var testData = new byte[dataSizeBytes];
    RandomNumberGenerator.Fill(testData);
}

```

```

var baselineSw = Stopwatch.StartNew();
var iterations = 10;
for (int i = 0; i < iterations; i++)
{
    var copy = new byte[testData.Length];
    Buffer.BlockCopy(testData, 0, copy, 0, testData.Length);
}
baselineSw.Stop();
var baselineMs = baselineSw.Elapsed.TotalMilliseconds /
iterations;
var encryptSw = Stopwatch.StartNew();
for (int i = 0; i < iterations; i++)
{
    var result = provider.Encrypt(testData);
    CryptographicOperations.ZeroMemory(result.Ciphertext);
}
encryptSw.Stop();
var encryptMs = encryptSw.Elapsed.TotalMilliseconds /
iterations;
_output.WriteLine($"=== ШИФРОВАНИЕ ({dataSizeBytes /
1024.0:F1} KB) ===");
_output.WriteLine($"Baseline: {baselineMs:F3}
мс");
_output.WriteLine($"Шифрование: {encryptMs:F3}
мс");
_output.WriteLine($"Коэффициент замедления: {encryptMs /
baselineMs:F2}x");
_output.WriteLine($"Скорость: "
+ $"{(dataSizeBytes / 1024.0 / 1024.0) / (encryptMs /
1000.0):F2} MB/s");

Assert.True(encryptMs < 5000,
    $"Шифрование {dataSizeBytes} байт должно занимать менее 5
секунд");
}
// PERF-02: Измерение времени расшифрования
[Theory(DisplayName = "PERF-02: Время расшифрования при разных
размерах данных")]
[InlineData(1024)]
[InlineData(1024 * 1024)]
[InlineData(10 * 1024 * 1024)]
public void PERF02_Decrypt_Performance_Measurement(int
dataSizeBytes)
{
    var provider = BuildProvider();
    var testData = new byte[dataSizeBytes];
    RandomNumberGenerator.Fill(testData);
    var encrypted = provider.Encrypt(testData);

    var iterations = 10;
    var decryptSw = Stopwatch.StartNew();
    for (int i = 0; i < iterations; i++)

```

```

        {
            var decrypted = provider.Decrypt(encrypted.Ciphertext,
encrypted.Metadata);
            CryptographicOperations.ZeroMemory(decrypted);
        }
        decryptSw.Stop();
        var decryptMs = decryptSw.Elapsed.TotalMilliseconds /
iterations;
        Assert.True(decryptMs < 5000,
            $"Расшифрование {dataSizeBytes} байт должно занимать менее
5 секунд");
    }
    // PERF-03: Сравнение Encrypt vs Decrypt - накладные расходы
    [Fact(DisplayName = "PERF-03: Сравнение Encrypt vs Decrypt (1
MB)")]
    public void PERF03_EncryptVsDecrypt_Overhead()
    {
        var provider = BuildProvider();
        var testData = new byte[1024 * 1024];
        RandomNumberGenerator.Fill(testData);
        var encrypted = provider.Encrypt(testData);
        var iterations = 50;
        var encryptSw = Stopwatch.StartNew();
        for (int i = 0; i < iterations; i++)
        {
            var r = provider.Encrypt(testData);
            CryptographicOperations.ZeroMemory(r.Ciphertext);
        }
        encryptSw.Stop();
        var decryptSw = Stopwatch.StartNew();
        for (int i = 0; i < iterations; i++)
        {
            var d = provider.Decrypt(encrypted.Ciphertext,
encrypted.Metadata);
            CryptographicOperations.ZeroMemory(d);
        }
        decryptSw.Stop();
        var encAvg = encryptSw.Elapsed.TotalMilliseconds /
iterations;
        var decAvg = decryptSw.Elapsed.TotalMilliseconds / iterations;
        var ratio = encAvg / decAvg;

        Assert.True(ratio > 0.3 && ratio < 3.0,
            $"E/D должно быть сопоставимым. Отношение: {ratio:F2}");
    }
    private static AesGcmFileCryptoProvider BuildProvider() =>
        new(new ConfigurationBuilder()
            .AddInMemoryCollection(new Dictionary<string, string?>
            {
                ["MasterKey:Key"] = "test-master-key-change-in-
prod!!!")
            })
        .Build());

```

Приложение К
(рекомендуемое)
Иллюстрационные материалы к тестированию API

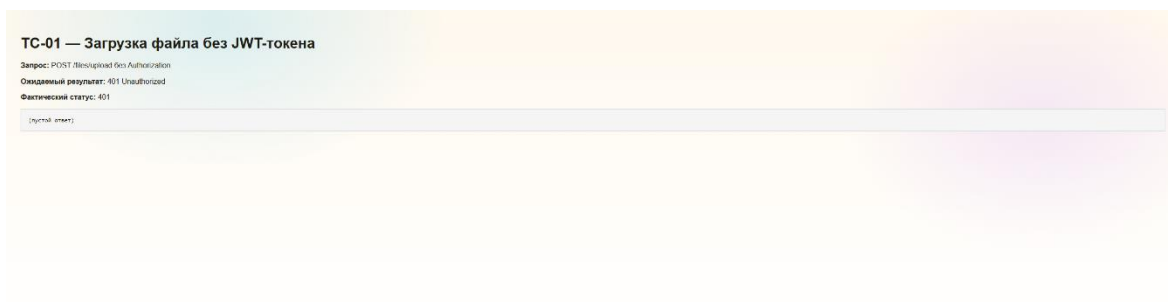


Рисунок К.1 – Результат тест-кейса TC-01: загрузка файла без JWT-токена

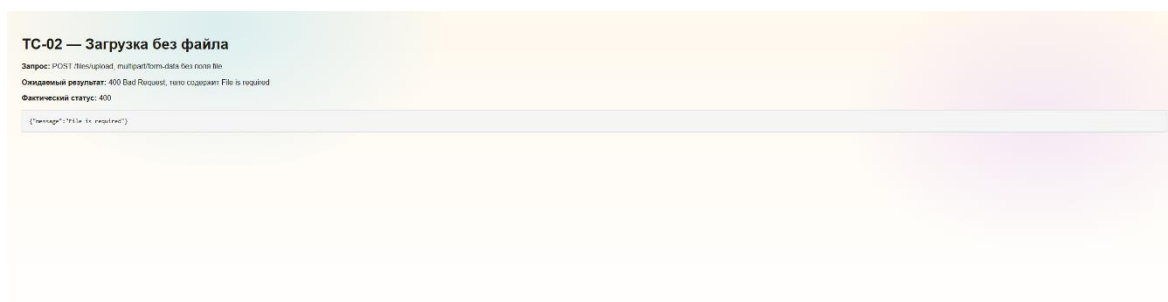


Рисунок К.2 – Результат тест-кейса TC-02: загрузка без файла

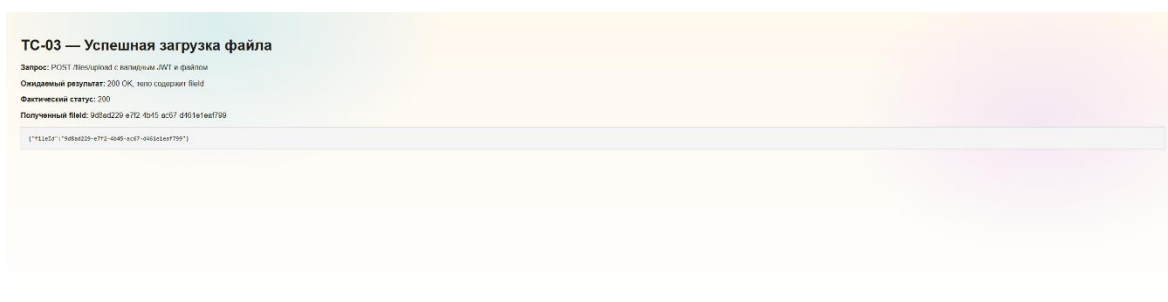


Рисунок К.3 – Результат тест-кейса TC-03: успешная загрузка файла

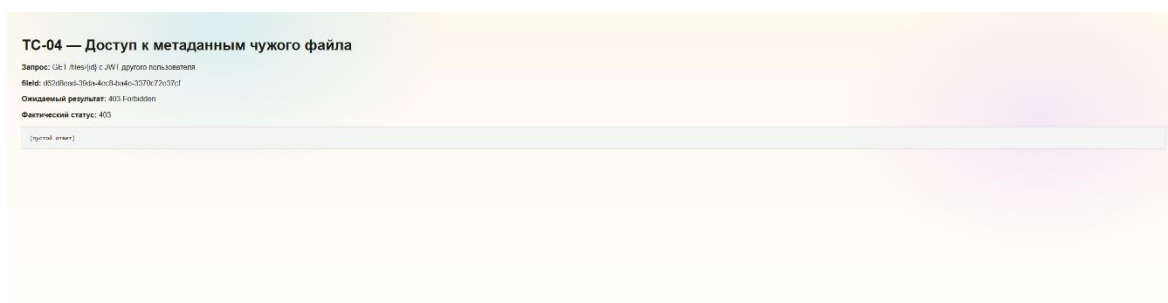


Рисунок К.4 – Результат тест-кейса TC-04: доступ к метаданным чужого файла

Приложение Л
(рекомендуемое)
Таблица решений для сервиса файлов

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32
Π1	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
Π2	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1
Π3	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1
Π4	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1
Π5	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
C1	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
C2	0	0	0	0	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1	0	0	0	0
C3	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1
C4	0	0	0	0	0	0	0	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0
C5	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
C6	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1	0	1	0	1