

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РФ

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

Факультет
вычислительной техники

Кафедра
МОиПЭВМ

Направление подготовки 09.03.04 «Программная инженерия»

Направленность Программная инженерия

БАКАЛАВРСКАЯ РАБОТА

на тему

**Менеджер паролей для безопасного
хранения учетных данных**

Студент _____ Сафронов Максим Андреевич
(подпись, дата) (ФИО полностью)

Руководитель _____ Балашова И. Ю.
(подпись, дата) (фамилия, инициалы)

Нормоконтролер _____ Такташкин Д.В.
(подпись, дата) (фамилия, инициалы)

Работа допущена к защите (протокол заседания кафедры от 03.06.2026г. № 12)

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**

Менеджер паролей для безопасного хранения учетных данных

1. Студент Сафронов Максим Андреевич гр. 22ВП2 факультета ВТ
<Фамилия, имя, отчество полностью>
2. Тема работы Менеджер паролей для безопасного хранения учетных данных

Тема утверждена приказом ПГУ №1583/о от "30" декабря 2025 г.

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

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

1) *Назначение: программные средства для безопасного хранения учетных данных.*

2) *Среда выполнения: Windows 10.*

Языковые средства разработки: C++, SQL, UML.

Инструментальные средства: Visual Studio 2022, MS SQL Server,

Visual Paradigm 17.2.

Технологии разработки: ООП, принципы SOLID, шаблоны проектирования.

3) *Основные функции приложения:*

а) регистрация и авторизация;

б) добавление нового устройства;

в) добавление новых учетных данных;

г) просмотр своих учетных данных;

д) добавление нового сложного пароля;

е) экспорт/импорт своих учетных данных между устройствами;

ж) предоставление справочной информации.

5. Перечень подлежащих разработке вопросов

- 1) *Анализ предметной области и постановка задачи;*
- 2) *Проектирование;*
- 3) *Реализация;*
- 4) *Контроль качества программного обеспечения;*
- 5) *Оформление пояснительной записки.*

6. 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 _____ 26.12.2025 г.
(подпись, дата)

Задание к исполнению принял студент _____ 26.12.2025 г.
(подпись, дата)

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

Реферат

Пояснительная записка содержит 100 листов, 35 рисунков, 7 таблиц, 20 использованных источников, 5 приложений и 4 листинга.

МЕНЕДЖЕР ПАРОЛЕЙ, УЧЕТНЫЕ ДАННЫЕ, ИНФОРМАЦИОННАЯ БЕЗОПАСНОСТЬ, КРИПТОГРАФИЯ, АУТЕНТИФИКАЦИЯ, ДВУХФАКТОРНАЯ ЗАЩИТА, P2P-АРХИТЕКТУРА, C++, REACT, PostgreSQL, СИСТЕМА УПРАВЛЕНИЯ БАЗАМИ ДАННЫХ.

Объект исследования – программное средство для безопасного хранения и управления учетными данными пользователей.

Цель выпускной квалификационной работы – спроектировать и разработать менеджер паролей, обеспечивающий конфиденциальность данных и их надежную синхронизацию между различными устройствами пользователя.

Технологии разработки – разработка программного средства выполнялась в среде Microsoft Visual Studio 2022 на языке программирования C++ с использованием React. Для хранения данных использовалась база данных PostgreSQL. При проектировании приложения применялись принципы объектно-ориентированного программирования и язык моделирования UML.

Среда исполнения – Windows 10, веб-браузер.

Результаты работы – в ходе выполнения выпускной квалификационной работы было спроектировано и разработано программное средство для генерации сложных паролей, их безопасного хранения и управления локальными хранилищами. Все требования к разработке приложения соблюдены.

					ПГУ 09.03.04 – 08ВР222.27 ПЗ							
Изм.	Лист	№ докум.	Подпись	Дата	Менеджер паролей для безопасного хранения учетных данных. Пояснительная записка			Лит.	Лист	Листов		
Разраб.		Сафронов М.А.								4	100	
Пров.		Балашова И.Ю.						Группа 22ВП2				
Реценз.												
Н. Контр.		Такташкин Д.В										
Утв.		Козлов А.Ю.										

Содержание

Введение.....	7
1 Анализ предметной области и выявление требований на разработку приложения менеджера паролей для безопасного хранения учетных данных	8
1.1 Анализ предметной области приложения	8
1.2 Анализ архитектурных подходов для защищенного хранения информации	9
1.3 Анализ аналогичных программных средств	10
1.4 Постановка задачи на разработку	14
1.5 Анализ требований	15
1.5.1 Анализ функциональных требований.....	15
1.5.2 Анализ нефункциональных требований.....	20
1.6 Планирование разработки и оценки бюджета	22
2 Проектирование серверной части менеджера паролей	24
2.1 Архитектура приложения.....	24
2.2 Проектирование структур данных	26
2.2.1 Концептуальная модель базы данных	26
2.2.2 Логическая модель базы данных.....	28
2.3 Проектирование серверной части	29
2.3.1 Проектирование сервиса AuthService	29
2.3.2 Проектирование сервиса PasswordService.....	33
2.3.3 Проектирование сервиса UserProfileService.....	36
2.4 Проектирование клиентской части	39
3 Разработка менеджера паролей.....	48
3.1 Разработка серверной части приложения.....	48

3.1.1 Разработка сервиса авторизации клиента.....	48
3.1.2 Разработка сервиса для создания паролей	52
3.1.3 Разработка маршрутизации клиента при помощи nginx.....	54
3.2 Разработка клиентской части менеджера паролей	55
4 Контроль качества программного обеспечения	59
4.1 Выбор режима тестирования и метрик кода	59
4.2 Тестирование серверной части менеджера паролей	60
4.2.1 Модульное тестирование	60
4.2.2 Интеграционное тестирование	64
4.3 Оценка качества программного кода	67
4.3.1 Метрика Холстеда.....	67
4.3.2 Метрика SLOC	72
Заключение	73
Список использованных источников	74
Приложение А (справочное) Глоссарий	77
Приложение Б (обязательное) Листинг программного средства.....	81
Приложение В (рекомендуемое) Графическая часть	89
Приложение Г (рекомендуемое) Листинг интеграционных тестов.....	92
Приложение Д (рекомендуемое) Графическое представление этапов разработки	98

Введение

В условиях стремительного развития цифровых технологий и повсеместного использования интернет-ресурсов, особую актуальность приобретают вопросы защиты личной информации. Современный пользователь каждый день использует десяток сайтов и приложений, каждое из которых требует регистрации. В результате необходимо хранить огромное количество конфиденциальной информации. Использование простых или одинаковых паролей увеличивает вероятность утечки.

Одним из наиболее эффективных способов повышения безопасности хранимых учетных записей являются менеджеры паролей. Такие приложения позволяют хранить в надежном месте свои личные данные, помогают в управлении ими, а также могут помочь сложные, уникальные пароли. Благодаря таким продуктам пользователю не нужно запоминать все свои пароли, достаточно знать только пароль от менеджера.

Актуальность разработки менеджеров паролей обусловлена тем, что в наше время происходит большое количество кибератак, а также повышающиеся требования к защите персональных данных. Современные требования к информационной безопасности включают в себя: использование современных криптографических алгоритмов, разграничение прав доступа, мониторинг действий пользователя.

Целью выпускной квалификационной работы является разработка приложения менеджера паролей, для безопасного хранения учетных данных.

Исходя из цели данной работы, были поставлены следующие задачи:

- проанализировать предметную область и существующие сервисы;
- спроектировать программный продукт;
- разработать программное средство;
- осуществить контроль качества программного обеспечения.

Успешное решение поставленных задач позволит создать качественный и безопасный продукт для хранения личных паролей.

1 Анализ предметной области и выявление требований на разработку приложения менеджера паролей для безопасного хранения учетных данных

1.1 Анализ предметной области приложения

В основе предметной области выпускной квалификационной работы лежит обеспечение информационной безопасности учетных данных пользователей и процессы их безопасного хранения, обработки и использования. Рассмотрим основные термины выбранной области, чтобы лучше понять тему разрабатываемой программы.

Информационная безопасность – защита данных от случайного или намеренного воздействия, которое может нарушить их конфиденциальность, целостность или доступность.

Учетные данные – информация, которая используется для опознания и входа пользователя в систему: логины, пароли, ключи доступ и другие сведения.

Пароль – секретная комбинация символов, с помощью которой подтверждают личность пользователя и дают доступ к системе.

Шифрование – превращение информации в такой вид, который нельзя прочитать без специального ключа.

Аутентификация – проверка, что пользователь действительно тот, за кого себя выдает, через подтверждение своих учетных данных.

Безопасное хранение данных – набор методов и технологий, которые защищают информацию от потери, изменений или несанкционированного доступа на устройстве пользователя или в облаке.

Таким образом, программные средства управления разрабатываются для пользователей, которым необходимо надежное хранение и удобное использование большого количества учетных данных от различных сервисов. Использование менеджеров паролей позволяет повысить уровень информационной безопасности, снизить риск компрометации личных данных и упростить процесс управления учетными данными.

1.2 Анализ архитектурных подходов для защищенного хранения информации

Задача защищенного хранения информации является одной из ключевых в области информационной безопасности. Особенно актуальной она становится в контексте разработки программных приложений, предназначенных для обеспечения безопасного хранения учетных данных пользователей.

В современной практике для решения задачи защищенного хранения информации применяется ряд архитектурных подходов, среди которых наиболее распространены локальное, централизованное (серверное), а также распределенное (P2P) хранение.

Локальная модель предполагает физическое размещение данных непосредственно на конечном устройстве пользователя. Ключевым преимуществом данного подхода является обеспечение автономности и минимизация рисков компрометации информации при передаче через сторонние серверные узлы. Тем не менее, этот метод характеризуется рядом существенных ограничений: в случае физической утраты или аппаратного повреждения устройства происходит безвозвратная потеря данных, а организация бесшовной синхронизации между несколькими устройствами представляет собой сложную техническую задачу.

Централизованная модель, в свою очередь, базируется на использовании выделенного серверного хранилища. Такой архитектурный паттерн существенно упрощает процессы синхронизации информации между различными клиентскими устройствами и позволяет эффективно реализовать механизмы централизованного управления правами доступа и политиками безопасности. Но у централизованной архитектуры есть один большой недостаток – наличие единой точки отказа. При компрометации сервера злоумышленник потенциально может получить доступ к личным данным пользователей, что делает данный подход менее предпочтительным для хранения критически важной информации.

Распределенное хранение данных предполагает размещение информации не на одном сервере, а между несколькими узлами системы. Одним из вариантов такого подхода является P2P-система, в которой участники сети взаимодействуют только друг с другом без передачи данных на центральный сервер. Преимуществом распределенной сети является отсутствие явной точки отказа, что повышает устойчивость приложения к сбоям и атакам на отдельные узлы. Для повышения безопасности в распределенной сети целесообразно использовать предварительное шифрование на стороне клиента, в этом случае в сеть уходят зашифрованные данные, что позволяет уменьшить риск перехвата данных.

Сравнение рассмотренных подходов показывает, что локальное хранение обладает простотой реализации, но ограничено в вопросах надежности и синхронизации. Централизованное хранение удобно для пользователя, но требует усиленной защиты данных и имеет риск компрометации центрального сервера. Распределенная система позволяет совместить возможность синхронизации данных с повышенной отказоустойчивостью и отсутствием точки отказа.

Таким образом, для поставленной задачи наиболее подходит использование распределенной P2P-системы, поскольку данный подход позволяет обеспечить безопасное хранение и передачу информации, повысить устойчивость системы к отказам и снизить зависимость от централизованной инфраструктуры.

1.3 Анализ аналогичных программных средств

Проанализировав рынок менеджеров паролей, схожих по функционалу на разрабатываемое в рамках выпускной квалификационной работы приложение, были выделены следующие альтернативы для сравнения: Bitwarden, KeePass, 1Password, LastPass. Рассмотрим их более подробно, выделим их возможности, преимущества и недостатки.

Bitwarden – наиболее популярный менеджер паролей с открытым

исходным кодом. Приложение дает возможность хранить учетные данные, банковские карты и заметки в зашифрованном виде. Синхронизация данных предусмотрена при помощи централизованного сервера, но можно самостоятельно развернуть серверную часть у себя. Графический интерфейс Bitwarden представлен на рисунке 1.

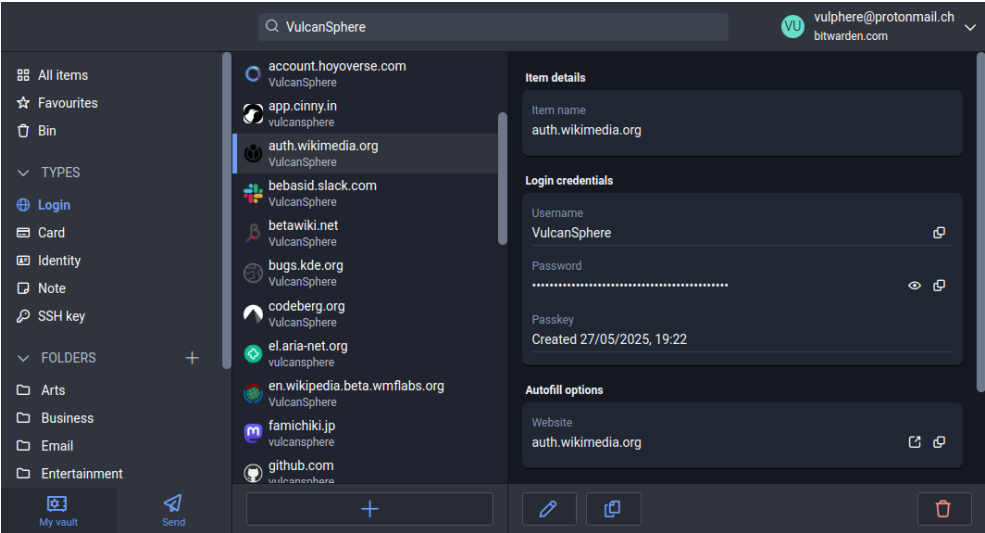


Рисунок 1 – Интерфейс Bitwarden

KeePass – бесплатный парольный менеджер с открытым исходным кодом, использующий локальное хранение данных в зашифрованном файле базы данных. Доступ к базе осуществляется мастер-ключом или файлом с закрытым ключом. Графический интерфейс KeePass представлен на рисунке 2.

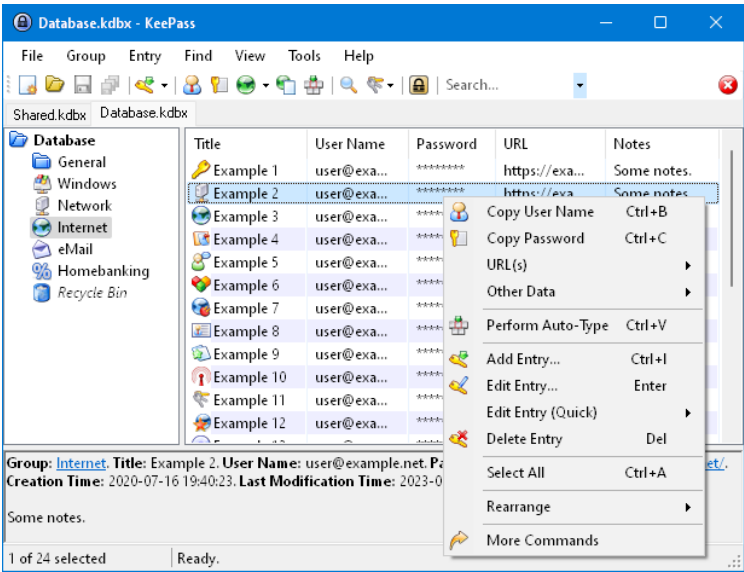


Рисунок 2 – Интерфейс KeePass

1Password – коммерческий менеджер паролей, ориентированный на частных пользователей и корпоративный сектор. Приложение предоставляет функции хранения паролей, документов, платежных данных и имеет синхронизацию через облачную инфраструктуру компании. Интерфейс 1Password представлен на рисунке 3.

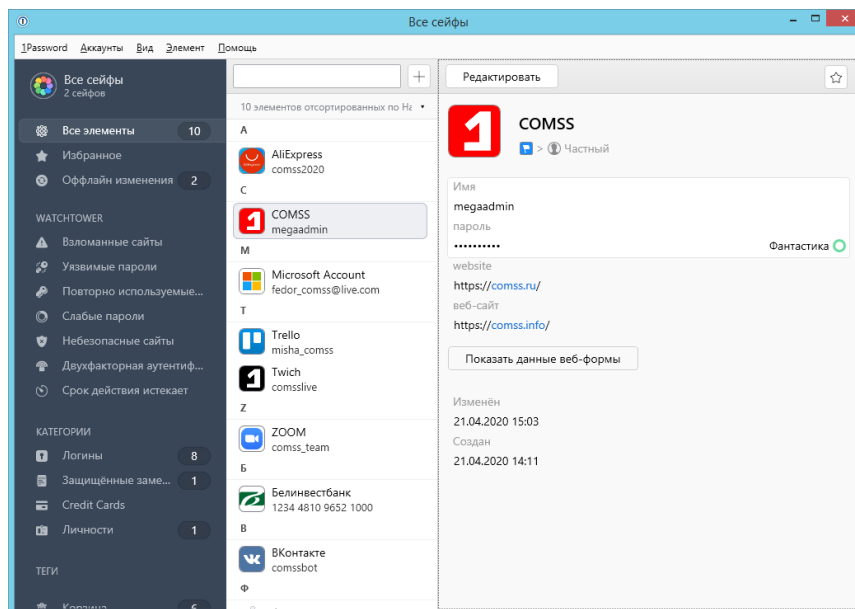


Рисунок 3 – Графический интерфейс 1Password

LastPass – облачный менеджер паролей, обеспечивающий хранение и автоматическое заполнение учетных данных на разных устройствах. Для защиты данных используется шифрование на стороне клиента. Интерфейс представлен на рисунке 4.

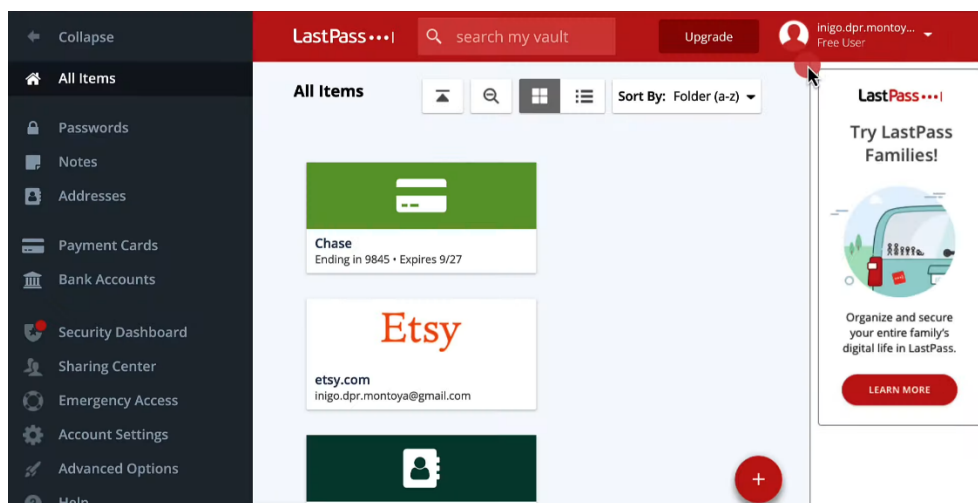


Рисунок 4 – Интерфейс LastPass

Кратко рассмотрим характеристики данных менеджеров паролей и выделим общие сведения об аналогах. Результаты краткого описания приведены в таблице 1.

Таблица 1 – Краткое описание общих сведений об аналогах

Характеристика	Bitwarden	LastPass	1Password	KeePass
Хранение данных	На сервере разработчика	На сервере разработчика	На сервере разработчика	Локальное в зашифрованном виде
Использование мастер-пароля	Да	Да	Да	Да
Применение шифрования	Да (AES-256)	Да (AES-256)	Да (AES-256)	Да (AES-256, ChaCha20)
Генерация сложных паролей	Да	Да	Да	Да
Поддержка двухфакторной аутентификации	Да	Да	Да	Через расширения
Открытый исходный код	Да	Нет	Нет	Да
Модель распространения	Открытое ядро, есть доп платный функционал	Бесплатный тариф, есть доп платный функционал	Бесплатный пробный период, остальное по подписке	Бесплатно
Синхронизация между устройствами	Да	Да	Да	Нет
Зависимость от центрального сервера	Да	Да	Да	Нет

Анализ существующих программных средств показал, что большинство современных менеджеров паролей используют централизованную архитектуру хранения данных, основанную на облачных сервисах. Такое решение дает пользователям удобную синхронизацию между устройствами, но

сохраняет зависимость от центрального сервера. Локальное решение, например, KeePass, дает пользователю полный контроль над своими данными, но не имеет механизма синхронизации.

Таким образом, по итогам сравнительного анализа можно сделать вывод о том, что рассмотренные сервисы предоставляют схожий функционал и имеют ряд преимуществ, так и определенные ограничения. Облачные менеджеры паролей зависят от централизованной серверной инфраструктуры, что снижает уровень контроля пользователя над своими данными. Локальное решение дает контроль и возможность развертывания, но уступает аналогам в удобстве использования. Поэтому необходимо разработать приложение, которое будет предоставлять следующий функционал:

- хранить в зашифрованном виде учетные данные пользователя локально;
- поддерживать распределенную архитектуру без прямой зависимости от центрального сервера;
- синхронизировать данные между устройствами пользователя;
- поддерживать двухфакторную аутентификацию для дополнительной защиты.

Реализация перечисленного функционала позволит создать программное средство, объединяющее преимущества существующих решений и устраняющее их основной недостаток за счет использования распределенной P2P-системы обмена данными.

1.4 Постановка задачи на разработку

Основной целью выпускной квалификационной работы является разработка приложения для безопасного хранения учетных данных пользователей. Основной его функцией должно быть хранение и синхронизация учетных данных пользователей. Кроме того, разрабатываемое приложение должно предоставлять пользователю удобный и гибкий интерфейс взаимодействия с учетными данными.

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

- авторизация и регистрация пользователей;
- добавление, изменение и удаление учетных записей у пользователя;
- добавление новых учетных данных;
- генерация сложных паролей;
- обеспечение защиты данных от несанкционированного доступа.

В результате выполнения этих задач необходимо получить программное средство, соответствующее требованиям безопасности и автономности.

1.5 Анализ требований

1.5.1 Анализ функциональных требований

Диаграммы вариантов использования описывают взаимоотношения и зависимости между группами вариантов использования и действующими лицами, участвующих в процессе [2].

В процессе анализа существующих менеджеров паролей были выделены следующие функциональные требования:

- регистрация и авторизация пользователя;
- изменение данных пользователя;
- выход из системы;
- просмотр активных сессий;
- импорт хранилища с учетными данными;
- просмотр журнал событий;
- добавление, получение и удаление паролей.

В соответствии с выявленными требованиями разработана диаграмма вариантов использования UML, представленная на рисунке В.1 в приложении В.

Диаграмма отражает функциональные требования, определяет роли пользователей и дает общее понимание о поведении системы на высоком уровне абстракции.

На рисунках ниже описаны наиболее важные бизнес-процессы, которые детализируют основные сценарии работы с системой и показывают порядок выполнения критически значимых функций.

На рисунке 5 представлена сценарий варианта использования «Авторизоваться».

Прецедент: авторизоваться
Идентификатор прецедента: 1
Краткое описание: пользователь входит в систему, используя свои учетные данные. После успешной авторизации получает доступ к функциям менеджера паролей.
Действующие лица: пользователь
Предусловия: пользователь зарегистрирован в системе.
Основной поток: <ol style="list-style-type: none"> 1. система предлагает ввести <u>email</u> и пароль; 2. пользователь вводит учетные данные; 3. система проверяет корректность данных; 4. система предоставляет доступ к личному кабинету.
Постусловия: пользователь авторизован в системе.

Рисунок 5 – Сценарий варианта использования «Авторизоваться»

Сценарий варианта использования «Восстановить доступ» представлен на рисунке 6.

Прецедент: восстановить доступ
Идентификатор прецедента: 2
Краткое описание: пользователь восстанавливает доступ к учетной записи, если забыл пароль. Система отправляет инструкции для установки нового пароля.
Действующие лица: пользователь
Предусловия: пользователь имеет зарегистрированную учетную запись.
Основной поток: <ol style="list-style-type: none"> 1. пользователь выбирает восстановление доступа; 2. система предлагает ввести <u>email</u>; 3. пользователь вводит <u>email</u>; 4. система отправляет инструкции по восстановлению; 5. пользователь устанавливает новый пароль.
Постусловия: доступ к учетной записи восстановлен.

Рисунок 6 – Сценарий варианта использования «Восстановить доступ»

Сценарий варианта использования «Изменить личные данные» представлен на рисунке 7.

Прецедент: изменить личные данные
Идентификатор прецедента: 3
Краткое описание: пользователь изменяет данные своей учетной записи: имя, пароль или фото профиля.
Действующие лица: пользователь
Предусловия: пользователь авторизован в системе.
Основной поток: <ol style="list-style-type: none"> 1. пользователь открывает раздел личных данных; 2. система отображает текущие данные профиля; 3. пользователь изменяет необходимые данные; 4. система сохраняет изменения.
Постусловия: данные учетной записи обновлены.

Рисунок 7 – Сценарий варианта использования «Изменить личные данные»

Сценарий варианта использования «Просмотреть список активных сессий» представлен на рисунке 8.

Прецедент: просмотреть список активных сессий
Идентификатор прецедента: 4
Краткое описание: пользователь просматривает список активных сессий своей учетной записи.
Действующие лица: пользователь
Предусловия: пользователь авторизован в системе.
Основной поток: <ol style="list-style-type: none"> 1. пользователь открывает раздел активных сессий. 2. система отображает список устройств и сессий. 3. пользователь просматривает информацию о входах.
Постусловия: пользователь получил информацию об активных сессиях.

Рисунок 8 – Сценарий варианта использования «Просмотреть список активных сессий»

Сценарий варианта использования «Завершить сессию» представлен на рисунке 9.

Прецедент: завершить сессию
Идентификатор прецедента: 5
Краткое описание: пользователь завершает одну, несколько или все активные сессии.
Действующие лица: пользователь
Предусловия: пользователь авторизован и имеет активные сессии.
Основной поток: <ol style="list-style-type: none"> 1. пользователь открывает список активных сессий; 2. система отображает доступные сессии; 3. пользователь выбирает сессию или все сессии; 4. система завершает выбранные сессии.
Постусловия: выбранные сессии завершены.

Рисунок 9 – Сценарий варианта использования «Завершить сессию»

Сценарий варианта использования «Создать хранилище паролей» представлен на рисунке 10.

Прецедент: создать хранилище паролей
Идентификатор прецедента: 6
Краткое описание: пользователь создает новое хранилище для хранения учетных данных.
Действующие лица: пользователь
Предусловия: пользователь авторизован в системе.
Основной поток: <ol style="list-style-type: none"> 1. пользователь выбирает создание хранилища; 2. система предлагает ввести название хранилища; 3. пользователь вводит название; 4. система создает новое хранилище.
Постусловия: новое хранилище паролей создано.

Рисунок 10 – Сценарий варианта использования «Создать хранилище паролей»

Сценарий варианта использования «Управлять хранилищами» представлен на рисунке 11.

Прецедент: управлять хранилищами
Идентификатор прецедента: 7
Краткое описание: пользователь просматривает список хранилищ, переименовывает их, импортирует или экспортирует данные.
Действующие лица: пользователь
Предусловия: пользователь авторизован в системе.
Основной поток: <ol style="list-style-type: none"> 1. пользователь открывает раздел хранилищ; 2. система отображает список хранилищ; 3. пользователь выбирает нужное действие; 4. система выполняет выбранную операцию.
Постусловия: изменения в хранилищах сохранены.

Рисунок 11 – Сценарий варианта использования «Управлять хранилищами»

Сценарий варианта использования «Добавить пароль в хранилище» представлен на рисунке 12.

Прецедент: добавить пароль в хранилище
Идентификатор прецедента: 8
Краткое описание: пользователь добавляет новую учетную запись в выбранное хранилище.
Действующие лица: пользователь
Предусловия: пользователь авторизован, хранилище создано.
Основной поток: <ol style="list-style-type: none"> 1. пользователь выбирает хранилище; 2. система предлагает ввести название сервиса, логин и пароль; 3. пользователь вводит данные; 4. система сохраняет данные в зашифрованном виде.
Постусловия: пароль добавлен в хранилище.

Рисунок 12 – Сценарий варианта использования «Добавить пароль в хранилище»

Сценарий варианта использования «Просмотреть список паролей» представлен на рисунке 13.

Прецедент: просмотреть список паролей
Идентификатор прецедента: 9
Краткое описание: пользователь просматривает список сохраненных учетных данных в выбранном хранилище.
Действующие лица: пользователь
Предусловия: пользователь авторизован, в хранилище есть сохраненные пароли.
Основной поток: <ol style="list-style-type: none"> 1. пользователь открывает хранилище; 2. система отображает список сохраненных записей; 3. пользователь просматривает доступную информацию.
Постусловия: пользователь получил список сохраненных паролей.

Рисунок 13 – Сценарий варианта использования «Просмотреть список паролей»

Сценарий варианта использования «Получить пароль» представлен на рисунке 14.

Прецедент: получить пароль
Идентификатор прецедента: 10
Краткое описание: пользователь получает сохраненный пароль для выбранного сервиса.
Действующие лица: пользователь
Предусловия: пользователь авторизован, пароль сохранен в хранилище.
Основной поток: <ol style="list-style-type: none"> 1. пользователь выбирает нужную запись; 2. система предлагает показать или скопировать пароль; 3. пользователь выбирает действие; 4. система предоставляет пароль.
Постусловия: пользователь получил доступ к выбранному паролю.

Рисунок 14 – Сценарий варианта использования «Получить пароль»

Сценарий варианта использования «Удалить пароль» представлен на рисунке 15.

Прецедент: удалить пароль
Идентификатор прецедента: 11
Краткое описание: пользователь удаляет учетную запись из хранилища паролей.
Действующие лица: пользователь
Предусловия: пользователь авторизован, пароль существует в хранилище.
Основной поток: <ol style="list-style-type: none"> 1. пользователь выбирает запись для удаления; 2. система запрашивает подтверждение; 3. пользователь подтверждает удаление; 4. система удаляет запись.
Постусловия: пароль удален из хранилища.

Рисунок 15 – Сценарий варианта использования «Удалить пароль»

Сценарий варианта использования «Просмотреть журнал аудита» представлен на рисунке 16.

Прецедент: просмотреть журнал аудита
Идентификатор прецедента: 12
Краткое описание: пользователь просматривает журнал аудита, в котором фиксируются важные действия в системе.
Действующие лица: пользователь
Предусловия: пользователь авторизован в системе.
Основной поток: <ol style="list-style-type: none"> 1. пользователь открывает журнал аудита; 2. система отображает список событий; 3. пользователь просматривает действия, выполненные в системе.
Постусловия: пользователь получил информацию о событиях системы.

Рисунок 16 – Сценарий варианта использования «Просмотреть журнал аудита»

Сценарий варианта использования «Фильтровать события журнала аудита» представлен на рисунке 17.

Прецедент: фильтровать события журнала аудита
Идентификатор прецедента: 13
Краткое описание: пользователь фильтрует события журнала аудита по типу или времени для анализа активности.
Действующие лица: пользователь
Предусловия: пользователь авторизован, журнал аудита содержит события.
Основной поток: <ol style="list-style-type: none"> 1. пользователь открывает журнал аудита; 2. система отображает список событий; 3. пользователь задает фильтр по типу или времени; 4. система отображает отфильтрованные события.
Постусловия: пользователь получил отфильтрованный список событий журнала.

Рисунок 17 – Сценарий варианта использования «Фильтровать события журнала аудита»

Проведя анализ функциональных требований, можно выделить варианты использования для менеджера паролей, которые отображают манипуляции пользователя в отношении системы.

1.5.2 Анализ нефункциональных требований

При проведении анализа нефункциональных требований для разрабатываемого менеджера паролей одной из ключевых задач являлся выбор архитектурного подхода, обеспечивающий высокий уровень безопасности, надежности и стабильности работы системы при хранении и обработке учетных данных пользователей. Специфика приложения заключается в работе с

конфиденциальной информацией, что предъявляет высокие требования к архитектуре системы [3].

На основе анализа предметной области были сформулированы следующие нефункциональные требования:

- безопасность;
- надежность;
- производительность;
- масштабируемость;
- доступность;
- удобство и простота использования.

Данные требования были определены с учетом особенностей приложения, поскольку система должна обеспечивать защищенное хранение данных, быстрый доступ к ним и устойчивую работу при увеличении количества пользователей.

Опираясь на сформулированные требования, в качестве архитектурного подхода была выбрана микросервисная архитектура с четким разделением ответственности между компонентами системы. Такой подход позволяет изолировать критически важные части, связанные с обработкой и шифрованием данных.

Ниже приведен анализ нефункциональных требований, обосновывающий выбранную архитектуру менеджера паролей.

Безопасность – одним из ключевых нефункциональных требований является обеспечение надежной защиты учетных данных пользователей на всех этапах их обработки и хранения. Предлагаемая архитектура предусматривает полную изоляцию процессов хранения и шифрования данных, что позволяет минимизировать риски несанкционированного доступа и утечки конфиденциальной информации.

Надежность – приложение должно обеспечивать стабильную работу и корректную обработку возможных ошибок и исключительных ситуаций.

Применение микросервисной архитектуры с разделением функциональности на независимые сервисы существенно снижает вероятность полного отказа системы при возникновении проблем в одном из ее компонентов, повышая тем самым общую отказоустойчивость приложения.

Масштабируемость и производительность – система должна обеспечивать эффективную поддержку роста пользовательской базы и объемов обрабатываемых данных без деградации производительности. Применение микросервисного подхода позволяет реализовывать горизонтальное масштабирование, выделяя дополнительные вычислительные ресурсы адресно, исключительно для наиболее нагруженных модулей.

Доступность – после успешной аутентификации пользователь должен иметь постоянный доступ к своим данным. Архитектурные решения ориентированы на быстрое восстановление работоспособности системы в случае сбоев и минимизацию времени простоя, что обеспечивает высокий уровень доступности сервиса.

Удобство использования – интерфейс программы должен быть интуитивно понятным и простым в использовании.

1.6 Планирование разработки и оценки бюджета

Планирование процесса разработки и расчет бюджета относятся к ключевым этапам, обеспечивающим успешную реализацию программного продукта [4].

Для управления проектом был выбран Microsoft Project, предоставляющая возможности по планированию работ, мониторингу выполнения задач и контролю ресурсов, задействованных в разработке.

После внесения в программу информации о стадиях создания приложения был сформирован план разработки программного обеспечения и выполнена оценка необходимых финансовых затрат. Для визуализации процесса выполнения работ использовалась диаграмма Ганта, отражающая основные этапы реализации проекта.

Диаграмма Ганта представляет собой инструмент планирования, отображающий задачи проекта в табличном виде. Диаграмма состоит из двух осей: вертикальная, где отображен список задач и горизонтальная со сроками.

Сформированная диаграмма Ганта приведена на рисунке Д.1 приложения Д.

В процессе планирования было определено 18 задач. В качестве даты начала проекта было выбрано 10 апреля 2026 года, а завершение работ согласно графику, запланировано 12 июня 2026 года.

Перечень поставленных задач представлен на рисунке Д.2 приложения Д.

Выполнение всех задач возложено на одного исполнителя – разработчика.

В результате общие трудозатраты на создание менеджера паролей составляют 380 часов, а совокупная стоимость проекта достигла 180000 рублей.

2 Проектирование серверной части менеджера паролей

2.1 Архитектура приложения

После проведения анализа требований, была выбрана микросервисная архитектура. Такая архитектура позволяет разделить систему на разные части, каждая из которых отвечает за свои задачи. Это наиболее востребовано для приложений, связанных с хранением и защитой конфиденциальных данных пользователей.

Для лучшего понимания структуры системы и определения набора компонентов, необходимых для функционирования разрабатываемого приложения, была разработана диаграмма компонентов [5]. Она позволяет наглядно представить архитектуру менеджера паролей, а также установить зависимости между компонентами, в роли которых выступают микросервисы и базы данных, используемые для хранения конфиденциальной информации.

Диаграмма компонентов для менеджера паролей представлена на рисунке 18.

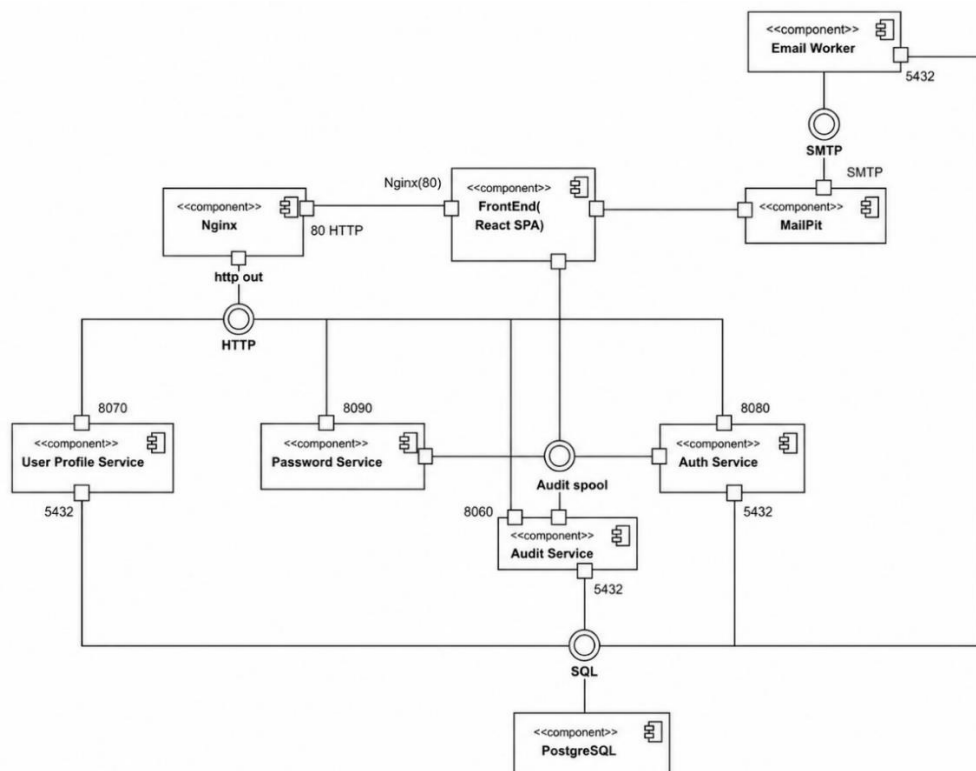


Рисунок 18 – Диаграмма компонентов

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

Auth Service – отвечает за регистрацию и вход пользователей в систему, управление пользовательскими сессиями.

Password Service – сервис для хранения и управления паролями, позволяет создавать, изменять, удалять хранилища с учетными данными. Все действия пользователей фиксируются в виде событий в отдельном сервисе.

User Profile Service – сервис для хранения информации о пользователе и его настройках. Этот сервис не хранит никакой секретной информации.

Audit Service – централизованный сервис журналирования. Получает события от других сервисов и предоставляет возможность просматривать историю действий.

Nginx – единая точка входа для всех запросов перенаправляет запросы к нужным внутренним сервисам и проверяет авторизацию.

Email Worker – сторонний сервис для отправки электронных писем получает запросы из очереди и выполняет рассылку отдельно от остальных запросов.

Mailpit – SMTP-сервер, который принимает письма и позволяет их просматривать.

Kafka – брокер сообщений, через который сервис авторизации пересылает события о создании одноразовых кодов в сервис рассылки.

Такое распределение на предметные области позволяет максимально гибко взаимодействовать с системой, упрощая процессы модификации и масштабирования сервисов.

Также для хранения одноразовых кодов для сервиса авторизации используется база данных ключ-значение Redis. Используя ее, можно заметно снизить нагрузку на основную базу данных приложения.

2.2 Проектирование структур данных

2.2.1 Концептуальная модель базы данных

Концептуальная модель используется для сбора требований перед проектированием базы данных, а также для определения структуры данных на начальном этапе проектирования. Такая модель отражает ключевые сущности и связи между ними [6].

Спецификация первоначального набора сущностей представлена в таблице 2.

Таблица 2 – Описание сущностей

№	Наименование	Описание
1	Пользователь	Участник сети, владеющий хранилищем паролей. Имеет ключи и устройства для доступа и подписи данных.
2	Устройство	Клиент пользователя, обладающий собственным ключом и участвующий в P2P-синхронизации.
3	Хранилище	Контейнер для записей с паролями и заметками. Может быть личным или общим.
4	Запись	Учетные данные: логин, пароль, заметка, URL и т. п., хранящиеся в зашифрованном виде.
5	Пиринговое разрешение	Сущность, описывающая, кто имеет доступ к хранилищу и с какими правами.
6	Пара ключей	Публичный и приватный ключ, используемые для подписи и шифрования данных.
7	Политика доступа	Правила доступа: роли, ограничения, требования MFA и т. д.
8	Журнал аудита	Записи действий пользователя и устройств: вход, изменение, синхронизация.
9	Сессия синхронизации	Событие обмена изменениями между пирами. Содержит статус и время.
10	Приглашение	Токен для добавления нового участника или предоставления доступа к хранилищу.
11	Метка	Категория для классификации записей внутри хранилища.
12	Вложение	Файл, прикрепленный к записи, зашифрованный вместе с ней.

На основе полученной информации была подготовлена концептуальная диаграмма в нотации «сущность–связь». Были определены основные типы связей между сущностями, а также их атрибуты. В процессе моделирования были определены основные типы связей между сущностями, включая

отношения «один ко многим» и ассоциативные связи, возникающие через дополнительные сущности. Важным аспектом являлось корректное отражение обязательности связей [7].

Графическое представление концептуальной модели представлено на рисунке 19.

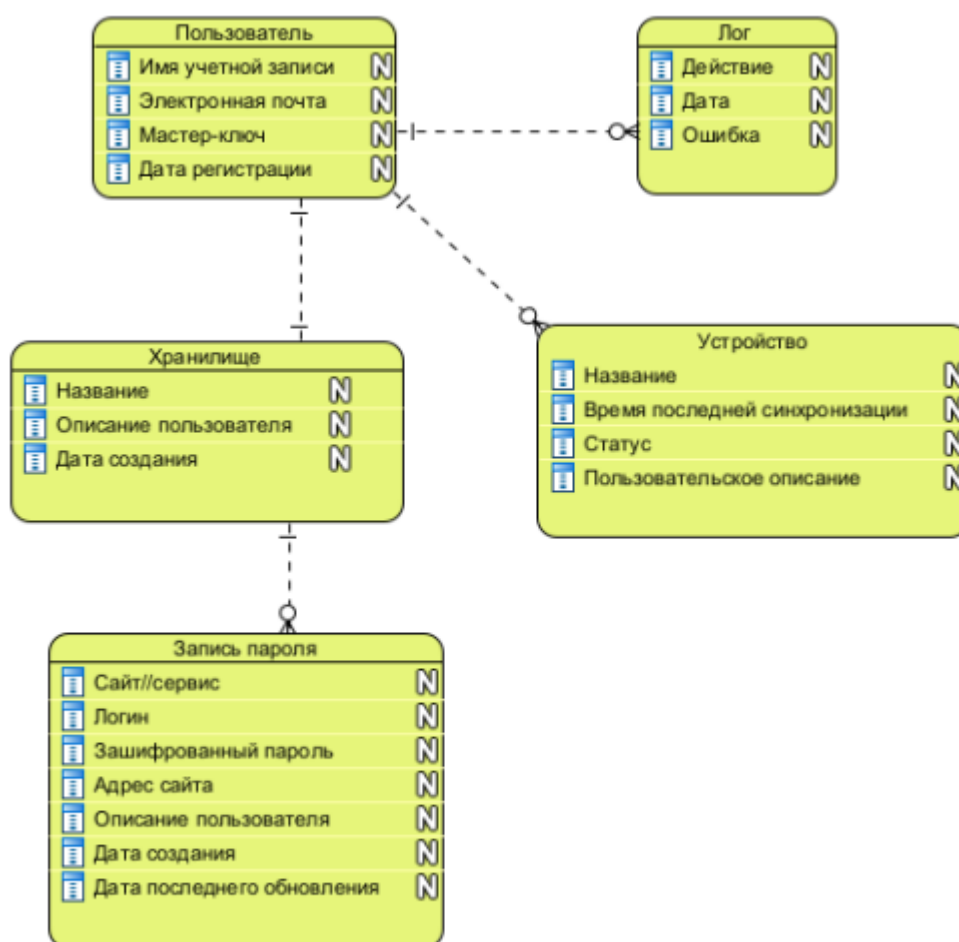


Рисунок 19 – Концептуальная модель базы данных

Построенная концептуальная модель отражает целостную логику функционирования информационной системы, описывая, каким образом организованы данные и какие зависимости между ними существуют.

Данная модель служит фундаментом для последующего проектирования логической и физической структуры базы данных, обеспечивая согласованность хранения информации и минимизацию избыточности.

2.2.2 Логическая модель базы данных

Логическая модель данных представляет из себя расширение концептуальной модели. Она включает в себя все сущности, атрибуты, ключи и взаимосвязи. Также на этом этапе уточняется тип данных для каждого поля. В ходе логического проектирования базы данных элементы концептуальной

На рисунке 20 представлена логическая модель базы данных.

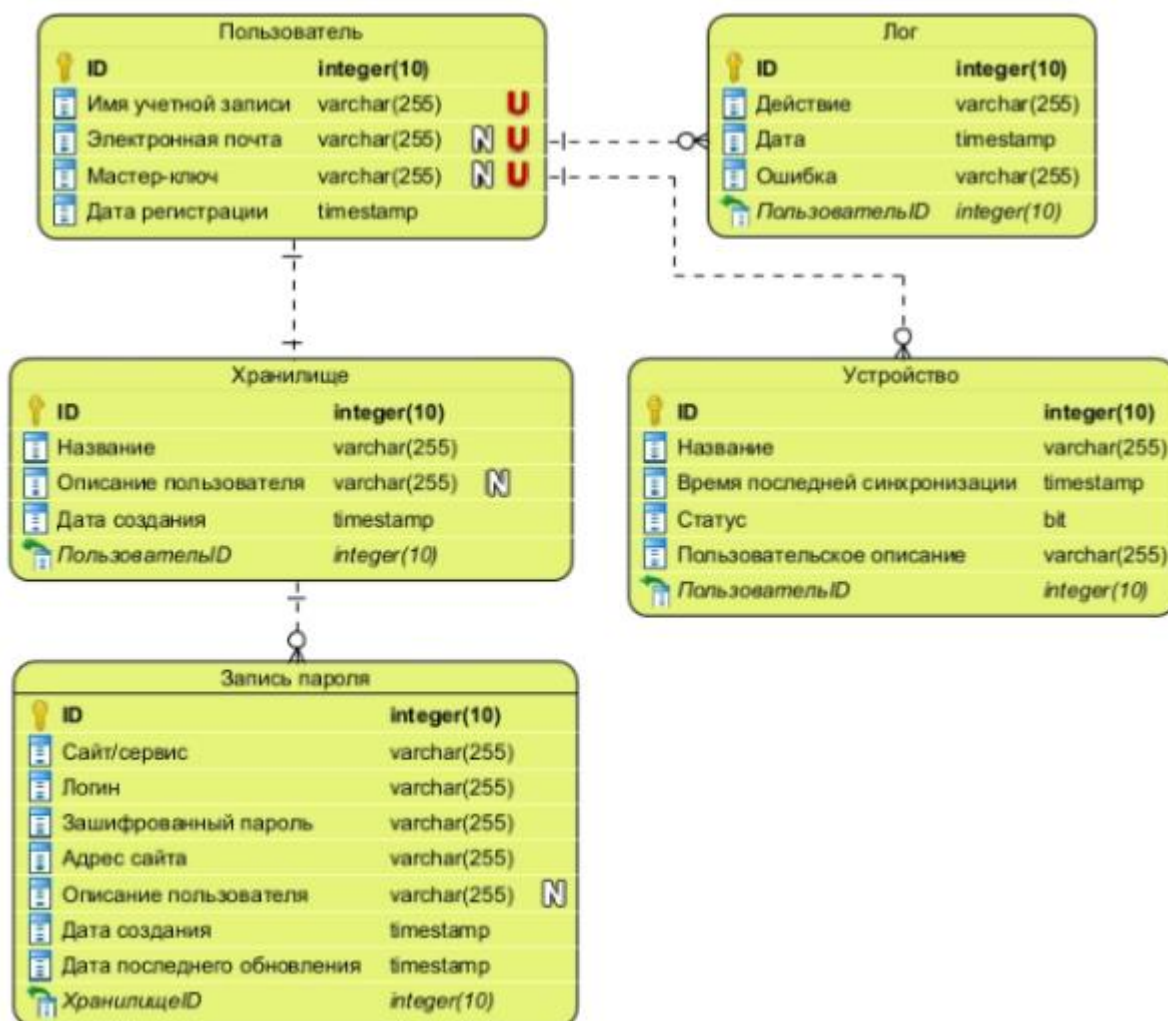


Рисунок 20 – Диаграмма логической модели базы данных

Таким образом на этапе логического проектирования базы данных особое внимание уделялось правилам ссылочной целостности, так как они обеспечивают корректность взаимодействия данных и сохраняют связи между таблицами при обновлении или удалении записей.

2.3 Проектирование серверной части

2.3.1 Проектирование сервиса AuthService

Сервис AuthService предназначен для управления учетными записями пользователей и обеспечения безопасной аутентификации в программном средстве управления текстовыми документами. Он отвечает за регистрацию новых пользователей, подтверждение email-адресов, выдачу и обновление JWT-токенов, а также управление профилем и паролем. Сервис взаимодействует с базой данных, брокером сообщений Kafka и системой мониторинга Prometheus, обеспечивая надежность и безопасность.

AuthService предназначен для управления учетными записями пользователей и обеспечения безопасной аутентификации и авторизации в системе. Сервис реализует полный жизненный цикл работы с учетной записью пользователя: регистрацию, подтверждение email-адреса, вход в систему, управление сессиями, смену и восстановление пароля, а также вход с другого устройства с использованием одноразового кода (QR-login).

AuthService разработан как самостоятельный микросервис и разворачивается во внутренней сети Docker, будучи доступным извне исключительно через reverse-проxy Nginx. Такое решение повышает безопасность и упрощает централизованное управление HTTP-доступом.

Архитектура AuthService построена по многослойному принципу и включает следующие логические уровни:

- слой обработки HTTP-запросов (AuthController);
- сервисный слой (бизнес-логика) (RegistrationService, LoginService, SessionManagementService, PasswordResetService, EmailBindService, DeviceLoginService);
- слой репозитория доступа к данным (UserRepository, SessionRepository, EmailVerificationRepository, PasswordResetRepository и др.);
- инфраструктурный слой (JWT, криптография, outbox-механизм, email-worker).

Такое разделение обеспечивает слабую связанность компонентов, упрощает тестирование и повышает сопровождаемость кода.

Слой HTTP реализован с использованием фреймворка Drogon и представлен контроллером AuthController.

Контроллер принимает внешние HTTP-запросы, выполняет базовую валидацию входных данных, проверку access-токенов и делегирует выполнение соответствующим сервисам бизнес-логики.

Для защищенных эндпоинтов используется Access Guard, который проверяет корректность JWT-токена, его срок действия и актуальность версии токена пользователя (token_version), что позволяет отзывать access-токены на уровне пользователя.

Сервисный слой содержит основную бизнес-логику AuthService:

- RegistrationService – регистрация пользователя, хеширование пароля и запуск процесса подтверждения email;
- LoginService – аутентификация по email или master-key, контроль количества неудачных попыток входа и временная блокировка учетной записи;
- SessionManagementService – управление пользовательскими сессиями, выход из системы, отзыв отдельных или всех сессий;
- PasswordResetService – восстановление пароля по email с использованием одноразового токена;
- EmailBindService – привязка нового email-адреса с обязательным подтверждением;
- DeviceLoginService – вход с другого устройства по одноразовому коду.

Токены доступа и обновления генерируются с использованием асимметричной криптографии. Access-токены являются короткоживущими, refresh-токены хранятся в базе данных в виде хеша и поддерживают строгую ротацию.

Слой репозитория реализует доступ к данным в PostgreSQL и

абстрагирован через интерфейсы, что позволяет использовать моки при модульном тестировании.

Основные репозитории:

- UserRepository – управление пользователями и их учетными данными;
- SessionRepository – управление сессиями и refresh-токенами;
- EmailVerificationRepository – хранение токенов подтверждения email;
- PasswordResetRepository – хранение токенов восстановления пароля;
- DeviceLoginRequestRepository – запросы на вход с другого устройства;
- OutboxRepository – реализация паттерна outbox.

Для отправки email-сообщений используется паттерн Transactional Outbox.

При регистрации, восстановлении пароля или привязке email AuthService сохраняет сообщение в таблицу outbox_messages в рамках одной транзакции с бизнес-операцией.

Отдельный сервис email-worker периодически опрашивает outbox, конкурентно безопасно извлекает сообщения и отправляет письма через SMTP-сервер. Такой подход обеспечивает:

- надежность доставки сообщений;
- отсутствие потери писем при сбоях;
- слабую связанность между AuthService и почтовой подсистемой.

В качестве reverse-проxy используется Nginx, который:

- проксирует запросы /auth/* к AuthService;
- отдает публичный ключ JWT (/well-known/jwks.json);
- скрывает сервис от прямого доступа извне.

Для локальной разработки и тестирования используется SMTP-сервер Mailpit, предоставляющий web-интерфейс для просмотра отправленных писем.

AuthService реализует следующий набор функций:

- регистрация пользователя с подтверждением email;
- аутентификация по логину и паролю с выдачей JWT-токенов;
- управление пользовательскими сессиями;
- смена и восстановление пароля;
- привязка и подтверждение email-адреса;
- вход с другого устройства по одноразовому коду;
- асинхронная отправка email-уведомлений.

Диаграмма классов данного сервиса представлена на рисунке В.2 в приложении В.

Ниже в таблице 3 представлено описание API сервиса.

Таблица 3 – Спецификация API сервиса авторизации

Метод	Адрес	Описание
POST	/auth/register	Регистрация нового пользователя по email или master-key. При наличии email инициируется отправка письма подтверждения.
POST	/auth/login	Аутентификация пользователя по email или master-key и паролю. Возвращает access и refresh токены.
POST	/auth/refresh	Обновление access-токена по refresh-токену с поддержкой строгой ротации.
POST	/auth/password/forgot	Инициация восстановления пароля. Отправляет письмо с одноразовым токеном.
POST	/auth/password/reset	Сброс пароля по одноразовому токену, полученному по email.
POST	/auth/device-login/start	Запуск процесса входа с другого устройства. Возвращает одноразовый код и идентификатор запроса.
GET	/auth/device-login/poll	Проверка статуса входа с другого устройства и получение токенов после подтверждения.
POST	/auth/logout	Завершение текущей пользовательской сессии.
POST	/auth/logout-all	Завершение всех сессий пользователя на всех устройствах.
GET	/auth/sessions	Получение списка активных пользовательских сессий.
POST	/auth/sessions/revoke	Отзыв конкретной пользовательской сессии по ее идентификатору.
POST	/auth/logout-others	Завершение всех сессий пользователя, кроме текущей.
POST	/auth/password/change	Смена пароля текущего пользователя с проверкой старого пароля.

Продолжение таблицы 3

Метод	Адрес	Описание
POST	/auth/email/bind/start	Инициация привязки нового email-адреса с отправкой письма подтверждения.
POST	/auth/email/bind/confirm	Подтверждение привязки нового email-адреса по одноразовому токenu.
POST	/auth/device-login/approve	Подтверждение входа с другого устройства по одноразовому коду.

Все операции, связанные с управлением учетной записью, доступны только авторизованным пользователям и защищены с помощью механизма JWT-аутентификации.

2.3.2 Проектирование сервиса PasswordService

Сервис генерации паролей предназначен для автоматического создания криптографически стойких паролей в системе управления паролями. Он используется для формирования безопасных паролей по заданным пользователем или системой политикам и интегрируется с основным PasswordService. Основной целью сервиса является снижение риска использования слабых или предсказуемых паролей за счет централизованной генерации с контролируемыми параметрами.

Сервис не взаимодействует напрямую с внешними пользователями и не имеет собственного хранилища данных. Он используется как вспомогательный компонент внутри password-service и вызывается через HTTP API или сервисный слой при создании новой записи пароля. Генерация выполняется исключительно на стороне сервера, что исключает утечку логики формирования паролей на клиент.

Архитектура сервиса генерации паролей разделена на следующие логические слои:

- утилитарный слой (PasswordGenerator);
- сервисный слой (PasswordService);
- слой доставки HTTP-запросов (PasswordController).

Утилитарный слой представлен компонентом PasswordGenerator, который инкапсулирует всю логику генерации паролей. Он использует криптографически стойкий генератор случайных чисел (libsodium) и поддерживает гибкую настройку параметров генерации, включая:

- длину пароля;
- использование строчных и заглавных букв;
- включение цифр;
- включение специальных символов;
- обязательное присутствие символов каждого выбранного типа.

PasswordGenerator не содержит бизнес-логики и не зависит от HTTP, баз данных или контекста пользователя. Это позволяет использовать его как независимый и тестируемый компонент, а также повторно применять в других сервисах системы при необходимости.

Сервисный слой представлен компонентом PasswordService, который отвечает за валидацию параметров генерации, проверку прав доступа пользователя к выбранному хранилищу (vault) и интеграцию с доменной моделью. При генерации нового пароля PasswordService вызывает PasswordGenerator, после чего результат немедленно шифруется и сохраняется в виде зашифрованного значения, не допуская хранения открытого текста пароля в памяти дольше необходимого времени.

Слой обработки HTTP-запросов реализован в PasswordController и предоставляет API для генерации паролей. Эндпоинт принимает параметры генерации в формате JSON, передает их в сервисный слой и возвращает результат клиенту. Контроллер не содержит логики генерации и выполняет исключительно функции маршрутизации, валидации входных данных и обработки ошибок.

Основной функционал сервиса генерации паролей включает:

- генерацию паролей по заданным политикам безопасности;

- обеспечение криптографической стойкости с использованием проверенных библиотек;
- централизованный контроль параметров генерации;
- интеграцию с сервисом хранения паролей без сохранения открытых значений;
- возможность расширения политик генерации без изменения клиентских приложений.

Таким образом, сервис генерации паролей повышает общий уровень безопасности системы, упрощает использование менеджера паролей для конечных пользователей и обеспечивает единый подход к формированию надежных паролей в рамках всей архитектуры.

Ниже на рисунке 21 представлена диаграмма классов данного сервиса.

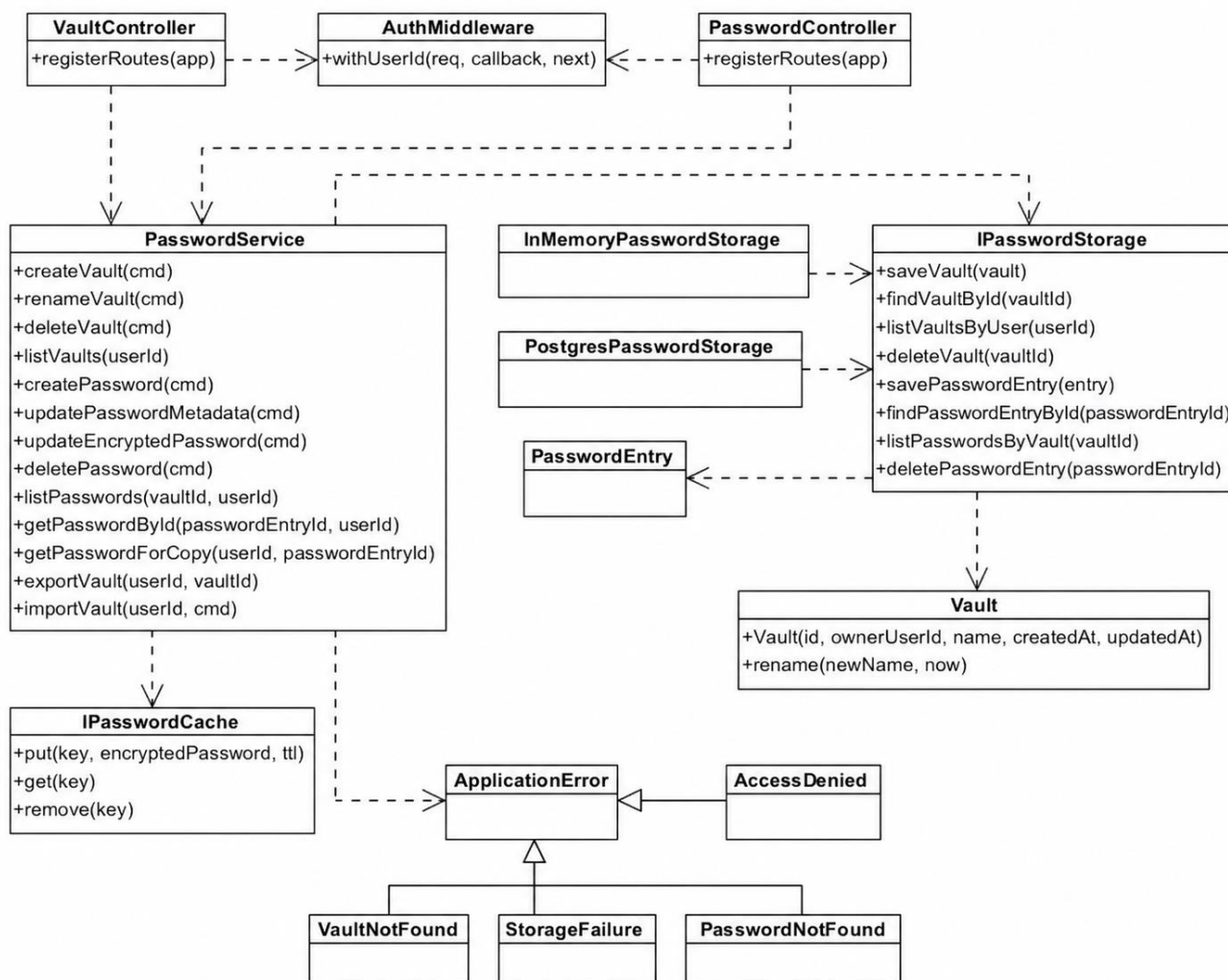


Рисунок 21 – Диаграмма классов сервиса паролей

Ниже в таблице 4 представлено описание API сервиса.

Таблица 4 – Спецификация API сервиса авторизации

Метод	Адрес	Описание
GET	/health	Проверка доступности сервиса.
POST	/vaults	Создание нового хранилища (vault) для пользователя.
PUT	/vaults/{vaultId}	Переименование существующего хранилища.
GET	/vaults	Получение списка всех хранилищ пользователя.
POST	/vaults/{vaultId}/passwords	Создание новой записи пароля с заранее сформированным секретом (уже зашифрованным клиентом).
GET	/vaults/{vaultId}/passwords	Получение списка паролей в хранилище. Метаданные возвращаются без секрета.
GET	/passwords/{passwordId}	Получение одной записи с шифротекстом секрета.
PUT	/passwords/{passwordId}/metadata	Обновление метаданных (название, логин, URL, заметки), без изменения секрета.
PUT	/passwords/{passwordId}/secret	Обновление только секрета (замена зашифрованного пароля).
DELETE	/passwords/{passwordId}	Удаление записи пароля.
POST	/vaults/{vaultId}/passwords/generate	Генерация нового пароля на сервере по заданной политике и сохранение в хранилище.
GET	/passwords/{passwordId}/decrypt	Расшифровка секрета на стороне сервиса при наличии ключа (опционально, зависит от модели безопасности).

Таким образом, сервис управления паролями обеспечивает как синхронное взаимодействие через RESTful API.

2.3.3 Проектирование сервиса UserProfileService

User Profile Service предназначен для хранения и управления пользовательским состоянием и предпочтениями в системе менеджера паролей. В отличие от сервиса аутентификации, данный сервис не участвует в обеспечении безопасности, не принимает решений о доступе и не реализует политики «разрешено/запрещено». Его задача заключается в представлении пользователя как сущности системы, а не как субъекта аутентификации.

Сервис получает идентификатор пользователя (user_id) от AuthService и предоставляет информацию о профиле пользователя, его настройках и текущем состоянии аккаунта. Таким образом, AuthService отвечает на вопрос «кто это?», а User Profile Service – «какой это пользователь и в каком он состоянии».

User Profile Service хранит следующие категории данных:

- идентификационные данные: идентификатор пользователя, отображаемое имя, ссылка на аватар, дата создания профиля;
- пользовательские настройки: язык интерфейса, часовой пояс, тема оформления, параметры автозаполнения и работы с буфером обмена, флаги уведомлений безопасности;
- состояние аккаунта: подтвержден ли email, завершено ли первичное ознакомление с системой, признак отключения аккаунта и причина отключения (при административном или системном вмешательстве).

Архитектура User Profile Service построена по многоуровневому принципу и включает следующие логические слои:

- слой обработки HTTP-запросов (ProfileController);
- сервисный слой (ProfileService);
- слой репозитория (ProfileRepository).

Слой обработки HTTP-запросов реализован с помощью контроллера ProfileController, который принимает внешние запросы и выполняет базовую валидацию входных данных. Контроллер не содержит бизнес-логики и делегирует все операции сервисному слою.

Сервисный слой представлен компонентом ProfileService, который реализует основную доменную логику: создание профиля пользователя с настройками по умолчанию, обновление отдельных полей профиля, а также обеспечение инвариантов домена (например, невозможность хранения некорректных значений пользовательских настроек). ProfileService не выполняет проверок прав доступа и не принимает решений на основе состояния аккаунта, что соответствует принципу «факт, а не политика».

Слой репозитория отвечает за взаимодействие с базой данных и представлен интерфейсом ProfileRepository и его реализацией. Репозиторий инкапсулирует операции сохранения и извлечения профиля пользователя и не содержит бизнес-логики. Для хранения данных используется реляционная база

данных (PostgreSQL), обеспечивающая целостность и надежность хранения пользовательского состояния.

User Profile Service предоставляет минималистичное HTTP API, включающее операции получения и частичного обновления профиля текущего пользователя. Такой подход упрощает интеграцию сервиса с пользовательским интерфейсом и другими компонентами системы, а также повышает сопровождаемость и расширяемость решения.

Ниже на рисунке 22 представлена диаграмма классов данного сервиса.

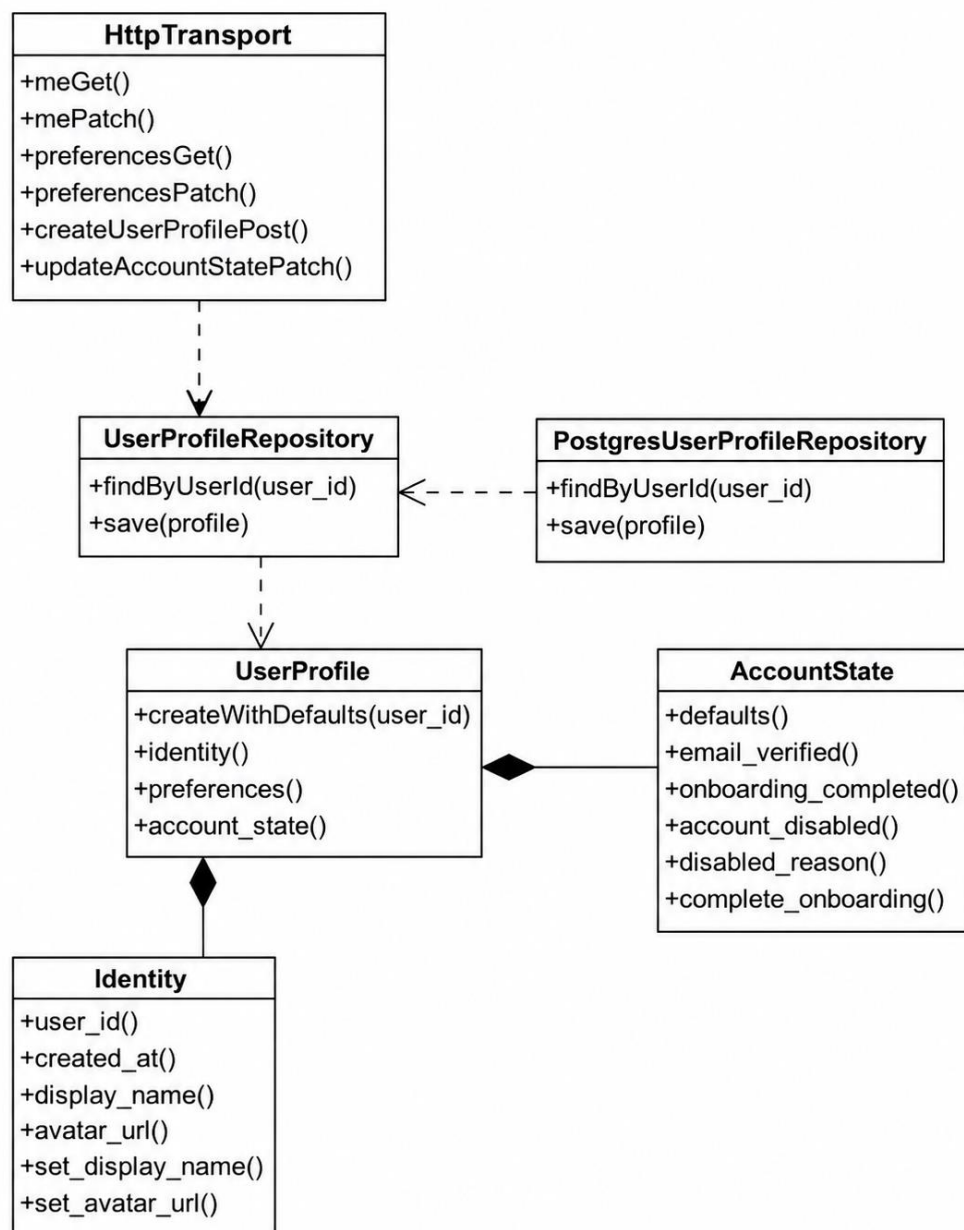


Рисунок 22 – Диаграмма классов сервиса управления аккаунтом

Ниже в таблице 5 представлено описание API сервиса.

Таблица 5 – Спецификация API сервиса авторизации

Метод	Адрес	Описание
GET	/profile/me	Получение полного профиля текущего пользователя. Возвращает пользовательские данные, его настройки и текущее состояние аккаунта.
PATCH	/profile/me	Частичное обновление данных профиля текущего пользователя. Позволяет изменять отображаемое имя, аватар и другие изменяемые атрибуты профиля.
GET	/profile/me/preferences	Получение пользовательских настроек текущего пользователя, включая язык интерфейса, тему оформления и параметры поведения клиента.
PATCH	/profile/me/preferences	Частичное обновление пользовательских настроек. Изменяет только переданные параметры, остальные сохраняются без изменений.

Таким образом, сервис управления профилем пользователя предоставляет доступ к пользовательскому состоянию в системе менеджера паролей.

2.4 Проектирование клиентской части

Клиентская часть менеджера паролей разработана в виде одностраничного приложения с использованием React. Такой подход позволяет обновлять содержимое страницы без ее полной перезагрузки, благодаря чему интерфейс работает быстрее и плавнее. Это повышает удобство использования приложения, так как пользователь взаимодействует с системой без задержек и лишних переходов между страницами. Использование библиотеки React способствует эффективной разработке интерактивных компонентов пользовательского интерфейса и обеспечивает оптимизированную обработку динамически изменяющихся данных.

Архитектура разрабатываемого программного обеспечения базируется на модульном принципе. Исходный код декомпозирован на изолированные логические блоки в строгом соответствии с их функциональным назначением. Данный подход существенно повышает эффективность процесса разработки, обеспечивая возможность независимой реализации и модульного тестирования отдельных частей системы.

Кроме того, модульная организация кода облегчает последующее

сопровождение и масштабирование приложения, поскольку модификации в рамках одного компонента минимизируют риск возникновения побочных эффектов в других частях системы. Каждый модуль инкапсулирует строго определенную бизнес-логику и регламентирует взаимодействие с серверной подсистемой, что в совокупности обеспечивает высокую структурную целостность, читаемость и прозрачность архитектуры.

Прототип интерфейса формы авторизации представлен на рисунке 23.

Auth UI
Local playground

Auth Device login 7

Account
Create an account or sign in

Email or MasterKey 1

Password 2

DeviceId 3
dev-ui

Login 4

Forgot password 5

Login Register 6

Рисунок 23 – Прототип формы авторизации

Рассмотрим основные элементы интерфейса главной формы приложения, представленные на рисунке 23:

- 1 – поле для ввода адреса электронной почты или мастер-ключа;
- 2 – поле для ввода пароля;
- 3 – поле для идентификатора устройства, который заполняется сам;
- 4 – кнопка, для входа в систему;
- 5 – кнопка, для восстановления доступа к аккаунту;
- 6 – переключатель формы входа или регистрации;
- 7 – кнопка для входа в систему с использованием другого устройства.

Прототип интерфейса формы журнала событий представлен на рисунке 24.

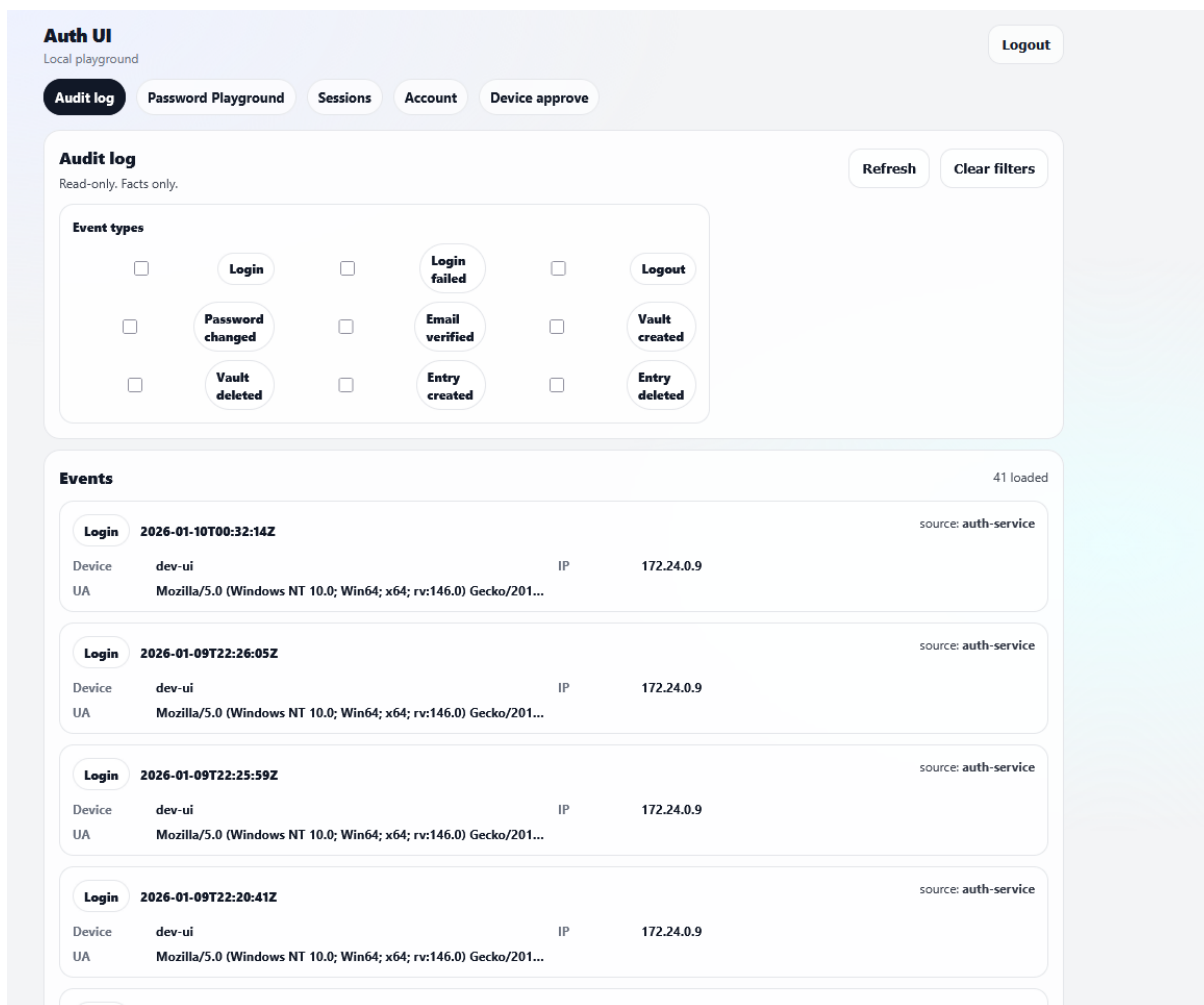


Рисунок 24 – Форма для просмотра журнала событий

Экран Audit log предназначен для просмотра журнала событий и содержит следующие элементы интерфейса.

Навигационное меню сервиса в верхней части экрана, позволяющее переходить между разделами: Audit log, Password Playground, Sessions, Account, Device approve. В правом верхнем углу расположена кнопка Logout для выхода из системы.

Блок фильтрации событий (Event types), содержащий набор чекбоксов, позволяющих отфильтровать события по типу, включая:

- Login;
- Login failed;
- Logout;
- Password changed;

- Email verified;
- Vault created;
- Vault deleted;
- Entry created;
- Entry deleted.

Кнопка Refresh, предназначенная для обновления списка событий с учетом выбранных фильтров.

Кнопка Clear filters, позволяющая сбросить все выбранные фильтры и отобразить полный список событий.

Список событий (Events), отображающий журнал действий пользователя в хронологическом порядке. Для каждого события отображается:

- тип события (например, Login);
- дата и время события;
- идентификатор устройства;
- IP-адрес;
- User-Agent;
- источник события (например, auth-service).

Прототип интерфейса управления хранилищами представлен на рисунке 25.

The screenshot displays the 'Auth UI' interface, specifically the 'Password Playground' section. The top navigation bar includes 'Audit log', 'Password Playground' (active), 'Sessions', 'Account', and 'Device approve'. A 'Logout' button is in the top right. Below the navigation, the 'Password Playground' section is titled 'Calls password-service through Nginx gateway: /password/*'. It features a 'Vaults' section with a 'Refresh' button and a 'Personal' filter. A table lists vaults with columns 'ID' and 'Name'. Below this is a 'Passwords' section with a 'Create' button and a table listing passwords with columns 'ID', 'Title', and 'Login'. The interface is clean and modern, with a light blue and white color scheme.

Рисунок 25 – Форма для управления хранилищами учетных данных пользователя

Экран Password Playground предназначен для управления хранилищами паролей и учетными данными пользователя и содержит следующие элементы интерфейса.

Навигационное меню сервиса в верхней части экрана, позволяющее переходить между разделами: Audit log, Password Playground, Sessions, Account, Device approve. В правом верхнем углу расположена кнопка Logout для завершения пользовательской сессии.

Информационный блок экрана, содержащий название раздела Password Playground. Блок управления хранилищами (Vaults), предназначенный для работы с наборами паролей и содержащий:

- кнопку Refresh для обновления списка хранилищ;
- поле ввода имени хранилища;
- кнопку Create для создания нового хранилища;
- поле отображения идентификатора выбранного хранилища;
- кнопку Load passwords для загрузки паролей выбранного хранилища;
- поле ввода нового имени хранилища;
- кнопку Rename для переименования хранилища;
- кнопку Delete vault для удаления выбранного хранилища;
- кнопки Export и Import для экспорта и импорта данных хранилища.

Таблица хранилищ, отображающая список доступных хранилищ с указанием:

- уникального идентификатора хранилища;
- имени хранилища;
- кнопок Select и Delete для выбора или удаления соответствующего хранилища.

Блок управления паролями (Passwords), предназначенный для работы с учетными данными внутри выбранного хранилища и содержащий:

- поле ввода заголовка записи (Title);
- поле ввода логина (Login);

- поле ввода пароля (Password) с маскированием вводимых символов;
- кнопку Create для создания новой записи;
- поле отображения идентификатора выбранного пароля;
- кнопку Update secret для обновления значения пароля;
- кнопку Delete для удаления выбранной записи.

Таблица паролей, отображающая список сохраненных учетных данных и содержащая:

- идентификатор записи;
- заголовок (Title);
- логин (Login);
- кнопку Select для выбора записи;
- кнопку Copy для копирования пароля в буфер обмена.

Прототип формы управления активными сессиями представлен на рисунке 26.

Auth UI
Local playground

Logout

Audit log Password Playground **Sessions** Account Device approve

Sessions

Load Logout others Logout all

sessionId	deviceId	ip	userAgent	createdAt	lastSeen	revokedAt	action
0279cb2e-3384-42e4-a33b-a21f846ec4a2	dev-ui	172.24.0.9	Mozilla/5.0 (Windows NT 1...	2026-01-10 00:32:13.987869+00			Revoke

Рисунок 26 – Форма для управления активными пользовательскими сессиями

Рассмотрим подробнее форму для управления активными сессиями:

Блок управляющих кнопок, включающий:

- Load – загрузка списка активных сессий пользователя;
- Logout others – завершение всех сессий пользователя, за исключением текущей;
- Logout all – завершение всех пользовательских сессий, включая текущую.

Таблица сессий, отображающая список пользовательских сессий и содержащая следующие столбцы:

- sessionId – уникальный идентификатор сессии;
- deviceId – идентификатор устройства, с которого была создана сессия;
- ip – IP-адрес, с которого выполнен вход;
- userAgent – информация о клиентском приложении или браузере;
- createdAt – дата и время создания сессии;
- lastSeen – дата и время последней активности в рамках сессии;
- revokedAt – дата и время отзыва сессии (при наличии);
- action – набор доступных действий над сессией.

Кнопка «Revoke» в строке таблицы, предназначенная для отзыва (принудительного завершения) выбранной пользовательской сессии.

Прототип формы управления профилем пользователя представлен на рисунке 27.

Рисунок 27 – Форма для управления профилем пользователя

Блок профиля пользователя, содержащий:

- изображение аватара пользователя;
- текстовое поле «Имя», предназначенное для отображения и изменения отображаемого имени пользователя;
- текстовое поле «Аватар (URL)», предназначенное для указания ссылки на изображение аватара в формате HTTP/HTTPS или Data URL;
- кнопку «Выбрать файл», позволяющую загрузить изображение аватара с локального устройства;
- кнопку «Вставить из буфера», позволяющую вставить изображение или ссылку из буфера обмена.

Изменения в блоке профиля применяются после сохранения.

Блок пользовательских настроек, предназначенный для управления поведением и внешним видом интерфейса и содержащий:

- выпадающий список Тема оформления (например, светлая/темная), применяется немедленно;
- выпадающий список Автозаполнение, позволяющий включить или отключить автоматическую подстановку данных в формы;
- числовое поле Таймаут буфера обмена, определяющее время хранения данных в буфере обмена (в секундах);
- выпадающий список Уведомления безопасности, предназначенный для включения или отключения уведомлений о значимых событиях безопасности.

Кнопка «Сохранить изменения» предназначен для фиксации и отправки на сервер модифицированных пользователем параметров профиля. Для предотвращения избыточных сетевых запросов и оптимизации нагрузки реализована проверка состояния формы: в случае отсутствия внесенных правок система блокирует отправку данных и выводит на экран информационное уведомление об отсутствии несохраненных изменений. Блок управления учетной записью предоставляет пользователю инструменты для мониторинга и

модификации базовых параметров идентификации и безопасности. В состав данного блока входят следующие функциональные элементы:

- информационное поле текущего адреса электронной почты, сопряженное с визуальным индикатором статуса верификации («подтвержден» / «не подтвержден»), обеспечивающее контроль прохождения процедуры подтверждения контактных данных;

- кнопка «Сменить почту», инициирующая многоэтапный процесс обновления адреса электронной почты, включающий обязательную верификацию нового адреса для предотвращения несанкционированного доступа;

- кнопка «Сменить пароль», обеспечивающая навигацию к форме обновления аутентификационных данных с соблюдением политик безопасности системы;

- индикатор текущего состояния учетной записи (например, «Активен»), позволяющий пользователю осуществлять контроль доступности своего профиля и прав доступа в системе.

3 Разработка менеджера паролей

3.1 Разработка серверной части приложения

Серверная часть системы реализована как набор микросервисов на C++ с использованием Drogon. Каждый сервис структурирован по слоистой архитектуре, где каждый слой четко отделен от других. На рисунке 28 представлен типовой вид многослойной архитектуры, используемый в каждом разработанном микросервисе системы [8].



Рисунок 28 – Типовая архитектура микросервисов

Такая реализация обеспечивает масштабируемость, отказоустойчивость и четкое разделение ответственности между компонентами системы.

3.1.1 Разработка сервиса авторизации клиента

Сервис авторизации в системе реализует полный контур безопасности для распределенной архитектуры: проверку личности пользователя (аутентификация), выдачу и проверку токенов (авторизация), управление жизненным циклом сессий, а также вспомогательные сценарии безопасности (смена/сброс пароля, привязка email, device-login). В рамках проекта сделан акцент на том, чтобы любое изменение «состояния доверия» пользователя приводило к предсказуемой и массовой инвалидации токенов, а управление сессиями оставалось прозрачным.

В сервисе используется пара токенов:

- Access Token (JWT) – в короткоживущий, используется микросервисами для авторизации запросов;

- Refresh Token – долгоживущий, используется только для обновления пары токенов (выпуска нового access token) и привязан к серверной записи сессии.

Ключевая особенность реализации – access token включает идентификатор сессии. Такой подход делает авторизацию более «контекстной», система может отличить одного и того же пользователя в разных сессиях или на разных устройствах, а также точно завершать нужные сессии.

В JWT используются только необходимые для безопасности поля:

- sub – идентификатор пользователя;
- sid – идентификатор сессии;
- tv – версия токена пользователя.

Это дает несколько плюсов:

- можно отслеживать, из какой именно сессии пришел запрос;
- можно сразу сделать все старые токены недействительными, если увеличить значение tv (даже если срок их действия еще не закончился);
- в токене нет лишних данных, что уменьшает риск утечек и ошибок.

Access token подписывается с использованием асимметричного алгоритма.

Это удобное решение для микросервисной архитектуры, потому что:

- приватный ключ есть только в сервисе авторизации, и именно он подписывает токены;
- остальные сервисы могут проверять токен только публичным ключом, не имея доступа к секрету.

В сервисе формируется JWKS-представление ключа: вычисляется kid и публикуются параметры публичного ключа. Это позволяет организовать ротацию ключей и кэширование ключей в потребителях.

При верификации JWT проверяются:

- корректность структуры токена;

- алгоритм (alg);
- корректность подписи;
- iss и aud;
- время жизни (exp);
- наличие и типы обязательных claims (sub, sid, tv).

Refresh token генерируется криптографически стойко и никогда не хранится в базе в открытом виде. Сервер работает с парой значений:

- raw refresh – отдается клиенту;
- hash refresh – вычисляется и сохраняется в БД.

Это дает практическую безопасность, даже при компрометации таблицы сессий злоумышленник не сможет использовать refresh токены напрямую.

Сессии хранятся в таблице sessions и содержат:

- user_id;
- device_id;
- refresh_token_hash;
- метаданные (ip, user_agent);
- временные поля (created_at, last_seen_at);
- состояние отзыва (revoked_at) и связь замены (replaced_by).

Сервис поддерживает функциональные операции:

- создание сессии;
- поиск активной сессии по refresh hash;
- получение списка активных сессий пользователя;
- отзыв конкретной сессии;
- отзыв всех сессий пользователя;
- отзыв всех сессий кроме текущей;
- отзыв всех сессий на конкретном устройстве;
- отметка last_seen_at.

Такой набор делает управление входами пользователя управляемым и наблюдаемым: можно реализовать «выйти со всех устройств», «выйти со всех

кроме текущего», «закрыть конкретную сессию».

Процесс входа состоит из нескольких шагов:

- проверка запроса – система убеждается, что переданы все нужные данные (пароль, deviceId), а также что идентификатор пользователя (email или masterKey) указан корректно;
- проверка блокировки – если пользователь временно заблокирован (например, из-за множества неудачных попыток входа), доступ запрещается до окончания блокировки после завершения ограничения снимаются;
- проверка email – при входе по email система проверяет, подтвержден ли он, в случае если нет, войти не удастся, что должно помочь защитить временные или неподтвержденные аккаунты;
- проверка пароля – если пароль неверный, учитываются неудачные попытки входа.

При успешном входе создается новая сессия, а все предыдущие сессии на этом устройстве завершаются (для одного устройства – одна активная сессия).

Это снижает вероятность одновременного использования нескольких refresh-токенов на одном устройстве и упрощает контроль за ними.

Дополнительная мера безопасности – использование поля token_version у пользователя и параметра tv в JWT. Когда выдается access_token, в него записывается текущее значение token_version.

Пароли в сервисе не хранятся в открытом виде и не хэшируются «самописным SHA». Используется libsodium и алгоритм Argon2id (через crypto_pwhash).

Argon2id – современный парольный хеш, устойчивый к GPU/ASIC перебору значительно лучше простых функций. Хеш хранится в формате libsodium (внутри уже содержит salt и параметры), а проверка делается через crypto_pwhash_str_verify. Реализация не требует вручную хранить salt/итерации – формат хеша самодостаточен и стандартизирован. Дополнительные сценарии безопасности: reset/change password, email bind, device-login

Помимо базовых login/refresh/logout, сервис реализует типовые «сложные» сценарии:

Password Reset – «Forgot password» не раскрывает существование email (для несуществующего email – без заметного эффекта). При успешном reset – пароль обновляется, повышается token_version, отзываются старые сессии пользователя, reset-токен помечается использованным.

Password Change – при смене пароля из активной сессии: проверяется текущий пароль, обновляется пароль, повышается token_version, отзываются остальные сессии и создается новая сессия.

Email Bind – сценарий безопасной привязки email. Запрещено привязывать уже занятый email для подтверждения используется временный verification-токен и подтверждение приводит к установке email и отметке токена использованным.

3.1.2 Разработка сервиса для создания паролей

Логика строится вокруг двух основных объектов:

- vault – это хранилище, где лежат данные пользователя (например, «личное», «работа»);
- password entry – отдельная запись внутри хранилища, в которой есть информация вроде названия, логина, ссылки, заметок и зашифрованного пароля.

Такое разделение удобно, потому что:

- данные можно разделить по категориям;
- проще переносить данные – можно экспортировать или импортировать сразу все хранилище вместе с его содержимым.

Важная особенность, сервис работает только с зашифрованными данными, то есть:

- при создании записи клиент отправляет уже зашифрованный пароль;
- сервис не расшифровывает его и не проверяет сам пароль;

- при запросе данных сервис возвращает либо только метаданные, либо сам зашифрованный пароль без изменений.

Это снижает риски того, что если систему взломают, то получить исходные пароли не получится. Доступ к данным контролируется через middleware, который берет ID пользователя из заголовка X-User-Id, после этого:

- пользователь работает только со своими хранилищами;
- нет доступа к чужим данным.

Проверка прав встроена в логику сервиса, поэтому даже при угадывании ID хранилища или записи, доступа все равно не будет.

Генератор проверяет корректность политики:

- минимальная длина,
- хотя бы один выбранный класс символов,
- выдает ошибку при некорректной политике.

Отдельно реализован endpoint «получить секрет для копирования» (copy). Это сделано не как обычный «get entry», а как специальная операция, так как она:

- чаще всего вызывается много раз (UX: пользователь нажимает Copy),
- должна иметь защиту от злоупотреблений.

В сервисе присутствуют два важных механизма.

Кэширование секрета на короткое время (TTL). Если пользователь запрашивает копирование несколько раз подряд, сервис может не ходить в БД каждый раз, а отдавать из кэша в пределах TTL. Это снижает нагрузку и ускоряет UX.

Rate limit для операции copy. Есть ограничение вида «несколько запросов за интервал времени», после чего возвращается «429 Too Many Requests». Это уменьшает риск массового вытягивания секретов автоматизированными запросами в случае компрометации клиентской сессии.

При обновлении секрета или удалении записи кэш инвалидируется, чтобы исключить ситуацию «старый секрет продолжает отдаваться из кэша».

3.1.3 Разработка маршрутизации клиента при помощи nginx

В проекте Nginx используется как единая точка входа (reverse proxy) для всей распределенной системы. Он решает сразу несколько задач:

- маршрутизация запросов по путям (path-based routing) к нужным микросервисам;
- единый периметр авторизации: проверка access token выполняется на уровне Nginx через auth_request, а в микросервисы передается уже извлеченный user_id;
- сокрытие внутренних API: часть эндпоинтов помечена как internal и недоступна извне;
- проксирование UI: фронтенд отдается отдельным контейнером Nginx со статикой, а внешний Nginx проксирует на него корневой /.

Ниже – как это реализовано в конфигурации nginx/conf.d/app.conf.

В конфигурации определены upstream-группы (по сути, имена сервисов в docker-сети) и их порты:

- auth_service – auth-service:8080;
- password_service – password-service:8090;
- user_profile_service – user-profile-service:8070;
- audit_service – audit-service:8060.

Сервер слушает 80 внутри контейнера Nginx, а наружу пробрасывается порт (в docker-compose это NGINX_HTTP_PORT, по умолчанию 8088).

Для всех проксируемых location выставляется стандартный набор заголовков. Это нужно для:

- корректного логирования и аудита;
- корректного определения схемы/хоста в редиректах;
- унификации поведения в разных окружениях.

Также везде задан proxy_read_timeout 30s и ограничение размера тела client_max_body_size 1m.

Ключевой интересный момент маршрутизации – использование механизма

auth_request «/internal/auth/validate». Как это работает:

- перед обработкой «защищенного» запроса Nginx делает внутренний подзапрос на специальный location;
- этот location проксирует на auth-service «/internal/auth/validate»;
- auth-service проверяет Authorization и возвращает результат.

В конфиге явно запрещены «internal» обращения для профиля и аудита. Делается это возвращением 404 на пути:

- /profile/internal и /profile/internal/... – 404;
- /api/profile/internal и /api/profile/internal/... – 404;
- /api/audit/internal и /api/audit/internal/... – 404.

Даже если сервис внутри контейнеров имеет внутренние endpoint'ы (админские/служебные), внешний клиент не сможет добраться до них через Nginx, это поддерживает разделение: публичные API доступны UI, внутренние только внутри сети/для внутренних процессов.

UI обслуживается отдельным контейнером Nginx со статическими файлами (ui/ui-nginx.conf). Корневой location / использует try_files /index.html. Это стандартный режим SPA: любой неизвестный путь отдает index.html, а маршрутизация делается на фронтенде. Внешний Nginx проксирует / – http://ui:80

Дополнительно сделан редиректы запросы на /ui/... переписываются в корневые пути (permanent redirect), чтобы UI был доступен в одном каноничном виде.

3.2 Разработка клиентской части менеджера паролей

Клиентская часть системы реализована в виде одностраничного веб-приложения (Single Page Application, SPA), предоставляющего пользователю единый интерфейс для аутентификации, управления сессиями, работы с паролями и мониторинга событий безопасности. Технологический стек приложения включает:

- библиотеку React 18 для декларативного построения пользовательского интерфейса;
- библиотеку react-router-dom (v6) для реализации клиентской маршрутизации;
- сборщик Vite, обеспечивающий быструю сборку проекта и работу локального сервера разработки.

Стратегия маршрутизации разделяет интерфейс приложения на публичные и защищенные области.

К публичным маршрутам относятся:

- страница аутентификации и регистрации (/, /auth, а также алиасы /login, /register);
- сценарии восстановления доступа (/forgot, /reset-password);
- процедуры верификации адреса электронной почты (/verify-email, /v/:token и их алиасы);
- страницы подтверждения привязки email (/confirm-email-bind, /b/:token).

Доступ к защищенным маршрутам контролируется механизмом защиты на уровне пользовательского интерфейса (UI Route Guard). При отсутствии валидного токена аутентификации система автоматически инициирует перенаправление пользователя на страницу входа. К защищенным разделам относятся:

- /sessions – управление активными пользовательскими сессиями;
- /profile – просмотр и редактирование персональных данных и настроек профиля;
- /password-playground – основной интерфейс для управления защищенным хранилищем (vault) и записями (entries) с учетными данными;
- /audit – визуализация и анализ журнала событий аудита;
- /device-approve – подтверждение входа с нового устройства (device-login) через доверенную авторизованную сессию;

- /change-password – процедура безопасной смены мастер-пароля в рамках текущей сессии.

Для обеспечения единообразия пользовательского опыта (UX) реализована система макетов (layouts). Основной макет включает постоянные элементы навигации (шапку и боковое меню). Для страниц восстановления доступа и подтверждения критических операций применяется минималистичный макет (режим «minimal»), в котором основная навигация скрыта для фокусировки внимания пользователя на целевом действии.

Состояние аутентификации пользователя сохраняется между перезагрузками страницы посредством использования веб-хранилища браузера localStorage, что обеспечивает непрерывность работы с приложением.

Логика выхода реализована в два шага:

- попытка вызвать logout на сервере (по refresh token);
- независимо от результата – очистка токенов локально и переход на страницу логина.

Для взаимодействия с backend используется единый слой API (обертка над fetch), который реализует:

- автоматическое добавление заголовка Authorization: Bearer «accessToken» ко всем запросам (кроме запросов с флагом noAuth);
- безопасное чтение тела ответа (JSON или текст);
- единообразное формирование ошибок (в ошибку добавляются HTTP status и тело ответа);
- timeout на POST-запросы (отдельная защита от зависаний сети).

Если сервер возвращает 401 на запрос с авторизацией, то токены очищаются и пользователь принудительно перенаправляется на базовый URL UI (чтобы убрать его с защищенных страниц, где дальнейшие запросы будут падать).

Это обеспечивает корректное поведение при:

- истекшем access token;

- отзыве текущей сессии;
- изменении tokenVersion на сервере (массовая инвалидизация токенов).

Клиентская часть разделена на несколько функциональных страниц/модулей, каждая соответствует отдельной подсистеме backend:

Модуль авторизации выполняет вход/регистрацию, получает access/refresh токены, сохраняет их локально и переводит пользователя в защищенную часть UI.

Модуль сессий показывает список активных сессий и поддерживает операции отзыва. Важный UX-момент, если пользователь отзывает текущую сессию, UI сразу очищает токены и переводит на страницу входа, чтобы не оставаться в «частично авторизованном» состоянии.

Профиль (Account) – интерфейс управления данными профиля и пользовательскими настройками. Данные берутся из user-profile-service, при этом UI работает только с публичными endpoint'ами, предназначенными для клиента.

Audit log – интерфейс для получения списка событий и показывает их пользователю в удобной форме. При этом клиент обращается к сервису через Nginx, а доступ к журналу защищен системой авторизации.

4 Контроль качества программного обеспечения

4.1 Выбор режима тестирования и метрик кода

Тестирование программного обеспечения является неотъемлемым этапом жизненного цикла разработки, направленным на верификацию соответствия системы заданным требованиям и выявление дефектов. Ключевой целью данного процесса является обнаружение программных ошибок, сбоев и неполноты функционала, а также комплексная оценка корректности функционирования и производительности программного продукта.

В современной инженерии программного обеспечения традиционно выделяют следующие основные виды тестирования:

- функциональное тестирование направлено на проверку соответствия системы заявленным требованиям и корректности реализации основных пользовательских сценариев;
- модульное тестирование предполагает изолированную проверку отдельных программных компонентов с целью подтверждения корректности их внутренней логики, независимо от остальных частей системы;
- интеграционное тестирование применяется для верификации корректности взаимодействия и обмена данными между различными модулями системы;
- нагрузочное тестирование позволяет оценить стабильность и производительность системы в условиях, имитирующих пиковые эксплуатационные нагрузки.

Для формализации процесса проверки разрабатываются тест-кейсы – детализированные алгоритмы, описывающие последовательность шагов для верификации конкретной функции на основе ожидаемого поведения пользователя.

В рамках данной выпускной квалификационной работы в качестве основного метода верификации выбрано модульное тестирование. Дополнительно применяется интеграционное тестирование для оценки

корректности взаимодействия между компонентами и подтверждения общей работоспособности системы.

В условиях роста архитектурной сложности современных программных систем существенно возрастают требования к их надежности и сопровождаемости. Для объективной оценки качества исходного кода применяются специализированные программные метрики, позволяющие идентифицировать фрагменты кода, требующие рефакторинга или более глубокого тестирования [18].

Для количественной оценки качества кода разрабатываемого менеджера паролей были выбраны следующие метрики: цикломатическая сложность, метрики Холстеда и метрика объема кода (SLOC – Source Lines of Code).

Цикломатическая сложность характеризует структурную сложность алгоритма и определяется количеством линейно независимых путей выполнения в программном коде.

Метрики Холстеда базируются на статистическом анализе количества уникальных и общих операторов и операндов, что позволяет оценить вычислительную сложность, а также прогнозировать трудозатраты на разработку и последующее сопровождение программного продукта.

Метрика SLOC используется для оценки физического размера программного обеспечения и представляет собой общее количество строк исходного кода, включая комментарии и пустые строки [19].

4.2 Тестирование серверной части менеджера паролей

4.2.1 Модульное тестирование

В качестве инструмента тестирования использовался фреймворк GoogleTest. Набор тестов запускался из отдельной тестовой сборки на C++, а проверка корректности выполнялась по возвращаемым кодам, состоянию сущностей и ожидаемым побочным эффектам сервисов.

Объектом тестирования выступала подсистема аутентификации,

включающая контроль версий токенов доступа, вход пользователя, ротацию refresh-токенов, завершение сеансов, управление активными сессиями, привязку email, восстановление и смену пароля, а также сценарий device login.

Для верификации корректности работы серверной логики и обеспечения надежности системы был разработан набор модульных тестов. Данные тесты покрывают большинство сценариев использования, включая процессы аутентификации, управление жизненным циклом сессий и механизма защиты. Перечень реализованных тестовых сценариев представлен в таблице 6.

По результатам запуска выполнено 34 теста, объединенных в 9 test suite. Все тесты завершились успешно: [PASSED] 34 tests. Общее время выполнения составило 169 мс, что указывает на изолированный характер тестов и отсутствие тяжелых внешних зависимостей. Результаты выполнения модульных тестов продемонстрированы на рисунке 29.

```
[ OK ] PasswordResetServiceTest.Forgot_NonExistingEmail_DoesNothing (0 ms)
[ RUN ] PasswordResetServiceTest.Forgot_ExistingEmail_CreatesTokenAndOutbox
20260322 19:56:31.969906 UTC 9042 INFO Password reset token for u@e.com: raw - PasswordResetService.cpp:97
[ OK ] PasswordResetServiceTest.Forgot_ExistingEmail_CreatesTokenAndOutbox (0 ms)
[ RUN ] PasswordResetServiceTest.Reset_InvalidToken_ReturnsInvalidOrExpired
[ OK ] PasswordResetServiceTest.Reset_InvalidToken_ReturnsInvalidOrExpired (0 ms)
[ RUN ] PasswordResetServiceTest.Reset_Success_UpdatesAndRevokes
[ OK ] PasswordResetServiceTest.Reset_Success_UpdatesAndRevokes (80 ms)
[-----] 4 tests from PasswordResetServiceTest (80 ms total)

[-----] 3 tests from PasswordChangeServiceTest
[ RUN ] PasswordChangeServiceTest.WeakPassword_NoSideEffects
[ OK ] PasswordChangeServiceTest.WeakPassword_NoSideEffects (0 ms)
[ RUN ] PasswordChangeServiceTest.InvalidCurrentPassword_NoSideEffects
[ OK ] PasswordChangeServiceTest.InvalidCurrentPassword_NoSideEffects (0 ms)
[ RUN ] PasswordChangeServiceTest.Success_RevokesAndBumpsAndReturnsNewTokens
[ OK ] PasswordChangeServiceTest.Success_RevokesAndBumpsAndReturnsNewTokens (84 ms)
[-----] 3 tests from PasswordChangeServiceTest (84 ms total)

[-----] 7 tests from DeviceLoginServiceTest
[ RUN ] DeviceLoginServiceTest.Start_CreatesPendingAndReturnsCode
[ OK ] DeviceLoginServiceTest.Start_CreatesPendingAndReturnsCode (0 ms)
[ RUN ] DeviceLoginServiceTest.Approve_HappyPath
[ OK ] DeviceLoginServiceTest.Approve_HappyPath (0 ms)
[ RUN ] DeviceLoginServiceTest.Approve_WrongCode_ReturnsInvalidOrExpired
[ OK ] DeviceLoginServiceTest.Approve_WrongCode_ReturnsInvalidOrExpired (0 ms)
[ RUN ] DeviceLoginServiceTest.Poll_Pending
[ OK ] DeviceLoginServiceTest.Poll_Pending (0 ms)
[ RUN ] DeviceLoginServiceTest.Poll_Approved_IssuesTokensAndConsumes
[ OK ] DeviceLoginServiceTest.Poll_Approved_IssuesTokensAndConsumes (0 ms)
[ RUN ] DeviceLoginServiceTest.Poll_Consumed_ReturnsInvalidOrExpired
[ OK ] DeviceLoginServiceTest.Poll_Consumed_ReturnsInvalidOrExpired (0 ms)
[ RUN ] DeviceLoginServiceTest.Poll_Expired_ReturnsInvalidOrExpired
[ OK ] DeviceLoginServiceTest.Poll_Expired_ReturnsInvalidOrExpired (0 ms)
[-----] 7 tests from DeviceLoginServiceTest (1 ms total)

[-----] Global test environment tear-down
[=====] 34 tests from 9 test suites ran. (169 ms total)
[ PASSED ] 34 tests.
```

Рисунок 29 – Юнит-тесты

Таблица 6 – Модульные тесты, покрывающие основные сценарии приложения

№	Название теста	Ожидаемый результат	Проверяемое поведение
1	AccessGuardTest. TokenVersionMismatch_ReturnsInvalidAccess	Возвращается статус InvalidAccess.	Контроль версии access token и защита от использования устаревшего токена.
2	LoginServiceTest. InvalidRequest_WhenNoIdentifier	Возвращается ошибка некорректного запроса.	Вход невозможен без идентификатора пользователя.
3	LoginServiceTest. InvalidCredentials_WhenUserNotFound	Возвращается ошибка неверных учетных данных.	Обработка попытки входа для несуществующего пользователя.
4	LoginServiceTest. .EmailNotVerified_WhenEmailLoginAndNotVerified	Возвращается признак неподтвержденного email.	Ограничение входа для пользователей с неподтвержденной почтой.
5	LoginServiceTest. Locked_WhenThirdFailedAttemptTriggersLock	После третьей ошибки вход блокируется.	Проверка счетчика неудачных попыток и механизма блокировки аккаунта.
6	LoginServiceTest. Success_ResetsAuthState_RevokesDeviceSessions _CreatesSession_ReturnsTokens	Вход завершается успешно, создается новая сессия, возвращаются токены.	Сброс auth-state после успешного входа и отзыв device-сессий.
7	RefreshServiceTest. InvalidRequest_WhenEmptyToken	Возвращается ошибка некорректного запроса.	Запрос на обновление (refresh) без токена не должен обрабатываться.
8	RefreshServiceTest. InvalidRefresh_WhenNoActiveSession	Возвращается ошибка InvalidRefresh.	Обновление токенов невозможно без активной сессии.
9	RefreshServiceTest. Success_Rotation_CreatesNewSession _RevokesOld_ReturnsNewTokens	Старая сессия отзывается, создается новая, возвращаются новые токены.	Проверка корректной ротации refresh-токена.
10	LogoutServiceTest.InvalidRequest_WhenEmptyToken	Возвращается ошибка некорректного запроса.	Запрос на выход (logout) без токена должен быть отклонен.
11	LogoutServiceTest. Idempotent_WhenSessionNotFound	Операция проходит идиempотентно (без ошибок).	Повторный вызов logout не приводит к ошибке при отсутствии сессии.
12	LogoutServiceTest.RevokesActiveSession	Активная сессия переводится в статус revoked.	Корректное завершение текущего сеанса.
13	SessionManagementServiceTest. ListSessions_ReturnsActiveSessions	Возвращается список активных сессий пользователя.	Проверка чтения информации о текущих сеансах.

Продолжение таблицы 6

№	Название теста	Ожидаемый результат	Проверяемое поведение
14	SessionManagementServiceTest. LogoutOthers_CallsRevokeAllExcept	Вызывается отзыв всех сессий, кроме текущей.	Массовый выход (logout) на других устройствах.
15	SessionManagementServiceTest. RevokeSession_CallsRevokeById	Вызывается отзыв сессии по идентификатору.	Адресное завершение выбранного сеанса.
16	SessionManagementServiceTest. RefreshOfRevokedSession_ReturnsInvalidRefresh	Возвращается ошибка InvalidRefresh.	Отозванная сессия не должна участвовать в процессе refresh.
17	EmailBindServiceTest. StartBind_EmailAlreadyRegistered_Returns409	Возвращается код ответа 409 (Conflict).	Нельзя привязать уже зарегистрированный email.
18	EmailBindServiceTest. StartBind_Success_CreatesTokenAndOutbox	Создается токен подтверждения и сообщение в outbox.	Успешный запуск процедуры привязки email.
19	EmailBindServiceTest. ConfirmBind_InvalidToken_ReturnsInvalidOrExpired	Возвращается ошибка invalid or expired.	Подтверждение по некорректному токenu отклоняется.
20	EmailBindServiceTest. ConfirmBind_Success_SetsEmailAndMarksUsed	Email привязывается, токен помечается как использованный.	Корректное завершение процедуры привязки (bind).
21	PasswordResetServiceTest. Forgot_NonExistingEmail_DoesNothing	Операция не вызывает побочных эффектов.	Безопасная обработка запроса на восстановление для несуществующего email.
22	PasswordResetServiceTest. Forgot_ExistingEmail_CreatesTokenAndOutbox	Создается reset-токен и запись в outbox.	Запуск процедуры восстановления пароля.
23	PasswordResetServiceTest. Reset_InvalidToken_ReturnsInvalidOrExpired	Возвращается ошибка invalid or expired.	Сброс пароля по неверному или просроченному токenu отклоняется.
24	PasswordResetServiceTest. Reset_Success_UpdatesAndRevokes	Пароль обновляется, связанные сессии отзываются.	Успешный сброс пароля приводит к инвалидации старых авторизаций.
25	PasswordChangeServiceTest. WeakPassword_NoSideEffects	Слабый пароль отклоняется без побочных эффектов.	Валидация сложности нового пароля.
26	PasswordChangeServiceTest. InvalidCurrentPassword_NoSideEffects	Некорректный текущий пароль отклоняется.	Пользователь не может сменить пароль без знания текущего.
27	PasswordChangeServiceTest. Success_RevokesAndBumpsAndReturnsNewTokens	Пароль изменяется, версия токенов увеличивается, выдаются новые токены.	Полный цикл успешной смены пароля.
28	DeviceLoginServiceTest.Start_CreatesPendingAndReturnsCode	Создается pending-заявка и возвращается код.	Старт процедуры device login на новом устройстве.

Окончание таблицы 6

№	Название теста	Ожидаемый результат	Проверяемое поведение
29	DeviceLoginServiceTest.Approve_HappyPath	Запрос успешно подтверждается.	Пользователь подтверждает вход по корректному коду.
30	DeviceLoginServiceTest.Approve_WrongCode_ReturnsInvalidOrExpired	Возвращается ошибка invalid or expired.	Неверный код подтверждения не принимается системой.
31	DeviceLoginServiceTest.Poll_Pending	Возвращается состояние pending.	Устройство ожидает подтверждения действия пользователем.
32	DeviceLoginServiceTest.Poll_Approved_IssuesTokensAndConsumes	После подтверждения выдаются токены, запрос помечается как consumed.	Завершение device login и выдача авторизации устройству.
33	DeviceLoginServiceTest.Poll_Consumed_ReturnsInvalidOrExpired	Повторный опрос consumed-запроса отклоняется.	Исключается повторное использование уже обработанного device flow.
34	DeviceLoginServiceTest.Poll_Expired_ReturnsInvalidOrExpired	Просроченный запрос отклоняется.	Контроль срока жизни device-кода.

Выполнение тестовой сборки показало, что все заявленные сценарии завершаются успешно: 34 теста из 34 прошли без ошибок. Набор тестов покрывает как позитивные сценарии, так и защитные механизмы: валидацию входных данных, обработку неверных токенов, блокировку после серии неудачных попыток, отзыв сессий и контроль одноразового использования подтверждающих кодов.

Особое внимание в тестовом наборе уделено безопасности и управлению жизненным циклом сессии. Проверяются ротация refresh-токенов, отзыв активных сеансов при logout, массовый logout на других устройствах, инвалидирование авторизации после смены или сброса пароля, а также корректная работа device login с промежуточными состояниями pending, approved, consumed и expired.

4.2.2 Интеграционное тестирование

Дополнительно для проверки корректности взаимодействия между компонентами системы было проведено интеграционное тестирование. В отличие от модульных тестов, где проверяются отдельные сервисы и методы

изолированно, интеграционные тесты охватывают полный сценарий работы нескольких подсистем одновременно: HTTP-слой, бизнес-логику аутентификации, обработку токенов, работу с сессиями и связанными сущностями.

Интеграционные тесты запускались с использованием `pytest` и располагались в каталоге `tests/integration`. В ходе выполнения было обнаружено и успешно пройдено 3 интеграционных теста, все тесты завершились со статусом `PASSED`, общее время выполнения составило 0.69 s. Листинг интеграционных тестов представлен в приложении Г. Результаты прохождения интеграционного тестирования продемонстрировано на рисунке 30, а также в таблице 7.

В рамках интеграционного тестирования были проверены следующие пользовательские сценарии.

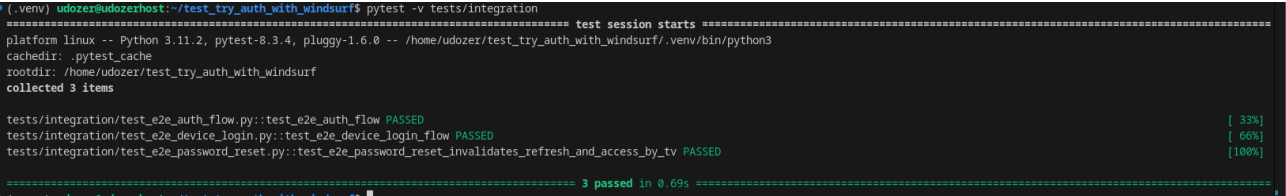
`test_e2e_auth_flow` тест, который проверяет сквозной процесс входа в систему. В рамках сценария выполняется аутентификация пользователя, получение токенов доступа и обновления, а также проверяется корректность дальнейшей работы механизма авторизации. Данный тест подтверждает, что основные компоненты системы аутентификации корректно взаимодействуют друг с другом в реальном потоке выполнения.

`test_e2e_device_login_flow` тест, который проверяет интеграцию механизма `device login`. В этом сценарии тестируется создание запроса на вход, подтверждение входа и завершение процедуры авторизации устройства с выдачей токенов. Проверка подтверждает, что подсистема входа через устройство функционирует корректно на уровне полного пользовательского сценария

`test_e2e_password_reset_invalidates_refresh_and_access_by_tv` тест, который подтверждает, что после инициирования процедуры смены пароля все ранее выданные токены доступа и токены обновления, а также ассоциированные с ними пользовательские сессии, переводятся в недействительное состояние. С точки зрения информационной безопасности данная мера является критически важной: она гарантирует принудительный отзыв прав доступа при смене

учетных данных, предотвращая несанкционированное использование ранее активных или потенциально скомпрометированных сеансов связи.

По итогам интеграционного тестирования установлено, что ключевые бизнес-процессы системы функционируют корректно не только на уровне изолированных модулей, но и в условиях их межкомпонентного взаимодействия. Это свидетельствует о функциональной и архитектурной целостности реализованных механизмов аутентификации, управления сессиями, авторизации устройств и восстановления доступа.



```
(.venv) udozer@udozerhost:~/test_try_auth_with_windsurf$ pytest -v tests/integration
===== test session starts =====
platform linux -- Python 3.11.2, pytest-8.3.4, pluggy-1.6.0 -- /home/udozer/test_try_auth_with_windsurf/.venv/bin/python3
cachedir: .pytest_cache
rootdir: /home/udozer/test_try_auth_with_windsurf
collected 3 items

tests/integration/test_e2e_auth_flow.py::test_e2e_auth_flow PASSED [ 33%]
tests/integration/test_e2e_device_login.py::test_e2e_device_login_flow PASSED [ 66%]
tests/integration/test_e2e_password_reset.py::test_e2e_password_reset_invalidates_refresh_and_access_by_tv PASSED [100%]

===== 3 passed in 0.69s =====
```

Рисунок 30 – Интеграционное тестирование

Таблица 7 – Результаты интеграционного тестирования

№	Имя теста	Назначение	Результат
1	test_e2e_auth_flow	Проверка полного сценария аутентификации пользователя	Успешно
2	test_e2e_device_login_flow	Проверка полного сценария входа через устройство	Успешно
3	test_e2e_password_reset_invalidates_refresh_and_access_by_tv	Проверка сброса пароля и отзыва ранее выданных токенов	Успешно

В результате проведенного функционального тестирования подтверждена корректность работы всех сервисов в изолированной среде, что обеспечивает высокую надежность системы при дальнейшей интеграции и эксплуатации.

4.3 Оценка качества программного кода

4.3.1 Метрика Холстеда

Метрики Холстеда предназначены для количественной оценки сложности программного кода, базирующейся на статистическом анализе используемых в нем операторов и операндов. Данный подход позволяет вычислить алгоритмический объем программы, оценить сложность ее практической реализации и спрогнозировать трудозатраты на разработку и последующее сопровождение. В отличие от сугубо количественных метрик, таких как SLOC (Source Lines of Code), метрики Холстеда обеспечивают оценку внутренней логической и алгоритмической сложности программного модуля.

В качестве объекта эмпирического анализа для расчета метрик Холстеда выбрана функция login компонента AuthService. Данная функция реализует бизнес-логику создания нового документа и управления правами доступа, добавляя указанных пользователей в список соавторов с назначением соответствующих ролей.

Для вычисления базовых показателей Холстеда необходимо определить количество уникальных операторов (n_1) и уникальных операндов (n_2), составляющих словарь программы, а также их общее количество в анализируемом фрагменте кода (N_1 и N_2 соответственно). На основе этих значений в дальнейшем рассчитываются производные метрики, такие как объем, сложность и трудоемкость алгоритма. Листинг кода функции login представлена в приложении Б.

Операторы функции login и число их вхождений приведены в таблице 8.

Таблица 8 – Операторы функции login

№	Оператор	Число вхождений
1	if	13
2	else	1
3	return	10

Продолжение таблицы 8

№	Оператор	Число вхождений
4	&&	6
5		3
6	!	8
7	=	25
8	>	1
9	->	13
10	*	7
11	.	33
12	,	14
13	;	41
14	(	44
15	)	44
16	{	15
17	}	15
18	login	1
19	empty	6
20	toLowerCase	1
21	trimCopy	2
22	looksLikeEmail	1
23	move	2
24	findByEmail	1
25	findByMasterKey	1
26	now	1
27	parseTimestampUtc	1
28	normalizeTimestampToIsoZ	2
29	resetAuthState	2
30	verify	1
31	incrementFailedAttemptsAndMaybeLock	1
32	revokeAllForDevice	1
33	generate	1
34	createSession	1
35	createAccessToken	1
36	accessTtlSeconds	1
	Итого	347

Операнды функции login и число их вхождений приведены в таблице 9.

Таблица 9 – Операнды функции login

№	Операнд	Число вхождений
1	user	16
2	LoginError	10
3	out	9
4	lockedUntil	9
5	id	9
6	empty	6
7	normEmail	5
8	userOpt	5
9	userRepo_	5
10	email	4
11	masterKey	4
12	deviceId	4
13	InvalidRequest	4
14	hasEmail	4
15	hasMasterKey	4
16	password	3
17	normMasterKey	3
18	mk	3
20	lockTp	3
21	state	3
22	refresh	3
23	session	3
24	nullopt	3
25	ip	2
26	userAgent	2
27	trimCopy	2
28	move	2
29	InvalidCredentials	2
30	nowTp	2

Продолжение таблицы 9

№	Операнд	Число вхождений
31	normalizeTimestampToIsoZ	2
32	Locked	2
33	resetAuthState	2
34	ok	2
35	sessionRepo_	2
36	accessTokenIssuer_	2
37	LoginResult	1
38	login	1
39	LoginService	1
40	toLowerCopy	1
41	looksLikeEmail	1
42	User	1
43	findByEmail	1
44	findByMasterKey	1
45	parseTimestampUtc	1
46	failedAttempts	1
48	emailVerified	1
49	EmailNotVerified	1
50	passwordVerifier_	1
51	verify	1
52	passwordHash	1
53	incrementFailedAttemptsAndMaybeLock	1
54	revokeAllForDevice	1
55	refreshTokenGenerator_	1
56	generate	1
57	createSession	1
58	hash	1
59	userId	1
60	sessionId	1
61	createAccessToken	1

Окончание таблицы 9

№	Операнд	Число вхождений
62	tokenVersion	1
63	refreshToken	1
64	raw	1
65	expiresIn	1
	Итого	180

Данные, полученные в ходе обработки программного кода функции login компонента авторизации были использованы для расчета характеристик программы по формулам из метрики Холстеда. Результаты вычислений представлены в таблице 10.

Таблица 10 – Результаты вычислений по метрике Холстеда

Значения	Результат
Словарь программы (n)	106
Длина программы (N)	527
Теоретическая длина программы (N')	611,81
Объем программы (V)	3545,61
Теоретический объем программы (V')	4116,48
Уровень качества программирования (L)	1,16
Уровень качества программирования (L')	0,93

Кроме того, был также рассчитан комплексный показатель качества кода – Maintainability Index. Этот показатель помогает оценить, насколько легко поддерживать и модифицировать код. Высокий индекс означает более легко поддерживаемый код. Расчет показателя происходит по формуле:

$$MI = 171 - 5.2 \cdot \ln HV - 0.23 \cdot CC - 16.2 \cdot \ln LOC,$$

где HV – вычислительная сложность кода;

CC – цикломатическая сложность кода;

LOC – количество строк кода.

Показатель CC, для данного кода равен 14, количество строк кода насчитывает 94 строк, а вычислительная сложность была рассчитана выше и составляет 3545,61.

Таким образом комплексный показатель равен 52

Результаты анализа находятся в пределах нормы, что соответствует среднему уровню сопровождаемости и сложности программного кода, а также соответствуют установленным критериям оценки качества кода. Это свидетельствует о невысокой сложности и прозрачности кодовой базы для разработчиков и простоты ее модификации.

4.3.2 Метрика SLOC

Был проведен анализ качества кода реализованной программы с помощью Visual Studio. Метрики кода приведены на рисунке 31.

Индекс удобства поддержки	Сложность организации цикл...	Глубина наследования	Взаимозависимость классов	Строки исходного кода	Строки исполняемого кода
■ 90	105	3	42	477	131
■ 68	64	1	38	342	123
■ 98	13	3	3	31	0
■ 91	1	1	2	13	1

Рисунок 31 – Метрики кода

Индекс удобства поддержки показывает, насколько удобно и легко поддерживать данный код. Рассчитывается по формуле:

$$MI = 171 - 5.2 \cdot \ln HV - 0.23 \cdot CC - 16.2 \cdot \ln LOC,$$

где HV – вычислительная сложность кода;

CC – Cyclomatic Complexity, структурная сложность кода;

LoC – количество строк исходного кода.

Нормой является значение от 20 до 100. В данном случае значения метрики для каждого модуля близки к 100, что означает, что код удобен в поддержке.

Заключение

В рамках выполнения выпускной квалификационной работы осуществлен комплексный анализ предметной области и спроектировано программное обеспечение, предназначенное для защищенного хранения и управления учетными данными пользователей.

На исследовательском этапе изучены ключевые аспекты информационной безопасности, регламентирующие работу с конфиденциальной информацией. Проведен сравнительный анализ существующих программных продуктов аналогичного назначения (Bitwarden, KeePass, 1Password, LastPass). По результатам исследования выявлены их функциональные достоинства и архитектурные ограничения, что позволило сформировать обоснованный перечень требований к разрабатываемой системе.

В ходе работы формализованы цели и задачи, определены функциональные и нефункциональные требования, а также смоделированы пользовательские сценарии взаимодействия с приложением. Приоритетное внимание уделено вопросам криптографической защиты данных, архитектуре безопасного локального хранения и обеспечению эргономичности пользовательского интерфейса.

На этапе проектирования обоснован выбор архитектуры программного комплекса, методов организации локального хранилища, криптографических алгоритмов и стека технологических решений. Детально проработаны механизмы аутентификации, разграничения прав доступа и обеспечения целостности информации.

Таким образом, в результате выполнения выпускной квалификационной работы не только проведен сравнительный анализ и сформулированы требования, но и разработана архитектурная модель системы, которая обеспечивает высокий уровень конфиденциальности, целостности и доступности пользовательских данных, полностью отвечая современным стандартам информационной безопасности.

Список использованных источников

1. Таненбаум, Э. С., ван Стеен, М. Распределенные системы: принципы и парадигмы / Таненбаум Э. С., ван Стеен М. / – СПб.: Питер, 2015. – 877 с.
2. UML – диаграмма вариантов использования (use case diagram) // [Электронный ресурс]. URL: <https://habr.com/ru/articles/47940/> (дата обращения: 27.05.2026)
3. UML: обзор основных типов диаграмм, диаграмма компонентов. Часть 2 // [Электронный ресурс]. URL: <https://habr.com/ru/articles/756552/> (дата обращения: 26.05.2026)
4. Просто о сложном. Концептуальная модель. // [Электронный ресурс]. URL: <https://mlkv.ru/sample-page/материалы-по-базам-данных/концептуальная-модель/> (дата обращения: 05.06.2026)
5. Глава 14. Проектирование и разработка баз данных. Руководство для разработчиков баз данных. Концептуальная, логическая и физическая модели данных. // Официальная документация Visual Paradigm. [Электронный ресурс]. URL: <https://circle.visual-paradigm.com/docs/database-design-engineering/database-designers-guide/conceptual-logical-and-physical-data-model/> (дата обращения: 08.06.2026)
6. Федорова К.П. Объектный подход в проектировании баз данных // Портал научно-практических публикаций [Электронный ресурс]. URL: <http://portalnp.ru/2014/06/1987> (дата обращения: 08.06.2026)
7. Клиент-серверная архитектура. // [Электронный ресурс]. URL: https://ru.hexlet.io/courses/internet-fundamentals/lessons/client-server/theory_unit (дата обращения: 09.06.2026)
8. Чикилев, Д. Д. Виды архитектуры приложений // Журнал «Научный лидер». 2024. С. 172. [Электронный ресурс]. URL: <https://scilead.ru/article/6673-vidi-arkhitekturi-prilozhenij> (дата обращения: 12.06.2026)

9. Ньюмен, С. От монолита к микросервисам. Эволюционные шаблоны для трансформации монолитной системы / С. Ньюмен // «БХВ-Петербург» 2022. С. 21.
10. Чикилев, Д. Д. Виды архитектуры приложений / Д. Д. Чикилев // Журнал «Научный лидер». 2024. С. 172. [Электронный ресурс]. URL: <https://scilead.ru/article/6673-vidi-arkhitekturi-prilozhenij> (дата обращения: 15.06.2026)
11. Юнит-тестирование для чайников // Хабр [Электронный ресурс]. URL: <https://habr.com/ru/articles/169381/> (дата обращения: 16.06.2026)
12. М. А. Сафронов. Менеджер паролей // Github [Электронный ресурс]. URL: https://github.com/udozer0/Coursovy_PiKPO (дата обращения 17.06.2026)
13. Юнит-тестирование для чайников // Хабр [Электронный ресурс]. URL: <https://habr.com/ru/articles/169381/> (дата обращения: 17.06.2026)
14. Интеграционное тестирование: виды, примеры и инструменты // <https://habr.com> [Электронный ресурс]. URL: <https://habr.com/ru/articles/843534/> (дата обращения: 05.06.2025)
15. Программный код и его метрики // Хабр [Электронный ресурс]. URL: <https://habr.com/ru/companies/intel/articles/106082/> (дата обращения: 18.06.2026)
16. Иванов, П.В. Управление проектами: Учебное пособие / П.В. Иванов. – Рн/Д: Феникс, 2018. –102 с.
17. Создаем диаграмму Ганта // UNISENDER [Электронный ресурс] URL: <https://www.unisender.com/ru/blog/chto-takoe-diagramma-ganta-i-kak-ee-sdelat/> (Дата обращения: 19.06.2026)
18. Значения метрик кода // Официальная документация Microsoft [Электронный ресурс]. URL: <https://learn.microsoft.com/ru-ru/visualstudio/code-quality/code-metrics-values?view=vs-2022> (дата обращения: 19.06.2026)
19. Метрики программного обеспечения Холстеда – разработка программного обеспечения. // Обучающий портал «GeeksforGeeks». [Электронный ресурс]. URL: <https://www.geeksforgeeks.org/software-engineering-halsteads-software-metrics/> (дата обращения: 19.06.2026)

20. Строки кода SLOC: раскрываем скрытые показатели качества. // Metridev. [Электронный ресурс]. URL: <https://www.metridev.com/metrics/sloc-lines-of-code-unveiling-the-hidden-quality-metrics/> (дата обращения: 19.06.2026)

Приложение А
(справочное)
Глоссарий

Учетные данные – уникальная совокупность идентифицирующих сведений и секретных атрибутов, предоставляемых субъектом доступа для прохождения процедур идентификации и аутентификации в информационной системе.

Пароль – конфиденциальная последовательность символов, выступающая в роли секретного компонента учетных данных и служащая для верификации подлинности субъекта доступа.

Мастер-пароль – главный криптографический ключ или секретная последовательность, применяемая субъектом для разблокирования зашифрованного хранилища и получения доступа ко всему массиву содержащихся в нем учетных данных.

Информационная безопасность – состояние защищенности информационной среды и самих данных, характеризующееся обеспечением их конфиденциальности, целостности и доступности для авторизованных субъектов в условиях противодействия внутренним и внешним угрозам.

Шифрование – алгоритмический процесс криптографического преобразования открытого текста в зашифрованный вид (шифротекст), делающий его нечитаемым для лиц, не обладающих соответствующим секретным ключом дешифрования.

Аутентификация – процедура проверки подлинности субъекта доступа посредством сопоставления предъявленных им идентификаторов и учетных данных с эталонными значениями, зарегистрированными в системе.

Авторизация – процедура предоставления субъекту доступа, успешно прошедшему аутентификацию, полномочий на выполнение определенных действий или получение доступа к конкретным ресурсам информационной системы в соответствии с заданной политикой безопасности.

Двухфакторная аутентификация (2FA) – механизм контроля доступа, при котором для подтверждения подлинности субъекта требуется последовательное предоставление двух или более независимых факторов аутентификации (знания, владения или биометрических характеристик).

Хэширование – одностороннее математическое преобразование массива данных произвольной длины в битовую строку фиксированной длины (хэш-значение), обладающее свойством устойчивости к коллизиям и вычислительной необратимостью.

Сессия пользователя – временной интервал активного взаимодействия субъекта доступа с информационной системой, начинающийся с момента успешного прохождения аутентификации и завершающийся явным разрывом соединения или истечением времени ожидания (тайм-аута).

База данных – структурированная, логически организованная совокупность данных, предназначенная для их долговременного хранения в памяти вычислительной машины и управляемая системой управления базами данных (СУБД).

P2P-сеть (одноранговая сеть) – децентрализованная вычислительная архитектура, в которой все узлы (пиры) обладают равным статусом и взаимодействуют друг с другом непосредственно для обмена ресурсами и данными, минуя выделенные централизованные серверы.

Синхронизация данных – процесс обеспечения согласованности и идентичности состояния информационных массивов на множестве распределенных узлов или конечных устройств посредством двустороннего обмена изменениями.

Клиент-серверная архитектура – архитектурный паттерн построения распределенных систем, основанный на строгом разделении функциональных обязанностей между сервером и их клиентом, взаимодействующими посредством сетевых протоколов.

Микросервисная архитектура – методология проектирования программных систем, при которой монолитное приложение декомпозируется на совокупность слабосвязанных, автономно развертываемых и функционально независимых модулей, взаимодействующих посредством легковесных механизмов.

Журнал аудита – хронологически упорядоченный, защищенный от

модификаций массив логических записей, фиксирующий факты событий информационной безопасности, действий субъектов доступа и изменений состояния системы для последующего анализа и расследования инцидентов.

Конфиденциальность данных – атрибут безопасности информации, заключающийся в обеспечении ее доступности исключительно для круга субъектов, обладающих явным образом делегированными полномочиями, и предотвращении ее раскрытия несанкционированным лицам.

Приложение Б
(обязательное)
Листинг программного средства

Листинг Б.1 – Функция для входа в аккаунт

```
std::optional<LoginError> LoginService::login(const
std::optional<std::string> &email, const
std::optional<std::string> &masterKey, const std::string
&password, const std::string &deviceId, const
std::optional<std::string> &ip, const std::optional<std::string>
&userAgent, LoginResult &out)
{
    if (password.empty() || deviceId.empty())
    {
        return LoginError::InvalidRequest;
    }

    const bool hasEmail = email && !email->empty();
    const bool hasMasterKey = masterKey && !masterKey->empty();
    if ((hasEmail && hasMasterKey) || (!hasEmail &&
!hasMasterKey))
    {
        return LoginError::InvalidRequest;
    }

    std::optional<std::string> normEmail;
    std::optional<std::string> normMasterKey;

    if (hasEmail)
    {
        auto e =
auth::utils::toLowerCopy(auth::utils::trimCopy(*email));
        if (e.empty() || !looksLikeEmail(e))
        {
            return LoginError::InvalidRequest;
        }
        normEmail = std::move(e);
    }

    if (hasMasterKey)
    {
        auto mk = auth::utils::trimCopy(*masterKey);
        if (mk.empty())
        {
            return LoginError::InvalidRequest;
        }
        normMasterKey = std::move(mk);
    }

    std::optional<auth::models::User> userOpt;
    if (normEmail)
    {
        userOpt = userRepo_->findByEmail(*normEmail);
    }
    else
    {

```

```

        userOpt = userRepo_>findByMasterKey(*normMasterKey);
    }

    if (!userOpt)
    {
        return LoginError::InvalidCredentials;
    }

    auto user = *userOpt;

    const auto nowTp = std::chrono::system_clock::now();

    if (user.lockedUntil)
    {
        const auto lockTp = parseTimestampUtc(*user.lockedUntil);
        if (lockTp && *lockTp > nowTp)
        {
            out.lockedUntil =
normalizeTimestampToIsoZ(user.lockedUntil);
            return LoginError::Locked;
        }
        userRepo_>resetAuthState(user.id);
        user.failedAttempts = 0;
        user.lockedUntil = std::nullopt;
    }

    if (normEmail && !user.emailVerified)
    {
        return LoginError::EmailNotVerified;
    }

    const bool ok = passwordVerifier_>verify(password,
user.passwordHash);
    if (!ok)
    {
        const auto state = userRepo_
>incrementFailedAttemptsAndMaybeLock(user.id);
        if (state.lockedUntil)
        {
            out.lockedUntil =
normalizeTimestampToIsoZ(state.lockedUntil);
            return LoginError::Locked;
        }
        return LoginError::InvalidCredentials;
    }

    userRepo_>resetAuthState(user.id);

    (void)sessionRepo_>revokeAllForDevice(user.id, deviceId);

    const auto refresh = refreshTokenGenerator_>generate();

```

```

        const auto session =
            sessionRepo_>createSession(user.id, deviceId,
refresh.hash, ip, userAgent);

        out.userId = user.id;
        out.sessionId = session.id;
        out.accessToken = accessTokenIssuer_
>createAccessToken(user.id, session.id, user.tokenVersion);
        out.refreshToken = refresh.raw;
        out.expiresIn = accessTokenIssuer_>accessTtlSeconds();
        out.lockedUntil = std::nullopt;

        return std::nullopt;
    }

```

Листинг Б.2 – Тест LoginServiceTest.cpp

```

#include <auth/services/LoginService.h>

#include "mocks/MockAccessTokenIssuer.h"
#include "mocks/MockPasswordVerifier.h"
#include "mocks/MockRefreshTokenGenerator.h"
#include "mocks/MockSessionRepository.h"
#include "mocks/MockUserRepository.h"

#include <gmock/gmock.h>
#include <gtest/gtest.h>

using ::testing::_;
using ::testing::Return;

namespace
{
    TEST(LoginServiceTest, InvalidRequest_WhenNoIdentifier)
    {
        auth::repositories::MockUserRepository userRepo;
        auth::repositories::MockSessionRepository sessionRepo;
        auth::security::MockPasswordVerifier passwordVerifier;
        auth::crypto::MockRefreshTokenGenerator refreshGen;
        auth::jwt::MockAccessTokenIssuer issuer;

        auth::services::LoginService svc(userRepo, sessionRepo,
passwordVerifier, refreshGen, issuer);

        auth::services::LoginResult out;
        const auto err = svc.login(std::nullopt,
                                std::nullopt,
                                "123",
                                "dev-1",
                                std::nullopt,
                                std::nullopt,
                                out);
    }

```

```

        ASSERT_TRUE(err.has_value());
        EXPECT_EQ(*err, auth::services::LoginError::InvalidRequest);
    }

TEST(LoginServiceTest, InvalidCredentials_WhenUserNotFound)
{
    auth::repositories::MockUserRepository userRepo;
    auth::repositories::MockSessionRepository sessionRepo;
    auth::security::MockPasswordVerifier passwordVerifier;
    auth::crypto::MockRefreshTokenGenerator refreshGen;
    auth::jwt::MockAccessTokenIssuer issuer;

    EXPECT_CALL(userRepo, findByMasterKey("MK"))
        .WillOnce(Return(std::nullopt));

    auth::services::LoginService svc(userRepo, sessionRepo,
    passwordVerifier, refreshGen, issuer);

    auth::services::LoginResult out;
    const auto err = svc.login(std::nullopt,

std::make_optional<std::string>("MK"),
                                "123",
                                "dev-1",
                                std::nullopt,
                                std::nullopt,
                                out);

    ASSERT_TRUE(err.has_value());
    EXPECT_EQ(*err,
auth::services::LoginError::InvalidCredentials);
}

TEST(LoginServiceTest,
EmailNotVerified_WhenEmailLoginAndNotVerified)
{
    auth::repositories::MockUserRepository userRepo;
    auth::repositories::MockSessionRepository sessionRepo;
    auth::security::MockPasswordVerifier passwordVerifier;
    auth::crypto::MockRefreshTokenGenerator refreshGen;
    auth::jwt::MockAccessTokenIssuer issuer;

    auth::models::User u;
    u.id = "u1";
    u.email = "user@example.com";
    u.emailVerified = false;
    u.passwordHash = "ph";

    EXPECT_CALL(userRepo, findByEmail("user@example.com"))
        .WillOnce(Return(u));

```

```

        auth::services::LoginService      svc(userRepo,      sessionRepo,
passwordVerifier, refreshGen, issuer);

        auth::services::LoginResult out;
        const auto err =
svc.login(std::make_optional<std::string>("user@example.com"),
        std::nullopt,
        "123",
        "dev-1",
        std::nullopt,
        std::nullopt,
        out);

        ASSERT_TRUE(err.has_value());
        EXPECT_EQ(*err, auth::services::LoginError::EmailNotVerified);
    }

TEST(LoginServiceTest, Locked_WhenThirdFailedAttemptTriggersLock)
{
    auth::repositories::MockUserRepository userRepo;
    auth::repositories::MockSessionRepository sessionRepo;
    auth::security::MockPasswordVerifier passwordVerifier;
    auth::crypto::MockRefreshTokenGenerator refreshGen;
    auth::jwt::MockAccessTokenIssuer issuer;

    auth::models::User u;
    u.id = "u1";
    u.masterKey = "MK";
    u.emailVerified = true;
    u.passwordHash = "ph";

    EXPECT_CALL(userRepo, findByMasterKey("MK"))
        .WillOnce(Return(u));

    EXPECT_CALL(passwordVerifier, verify("wrong", "ph"))
        .WillOnce(Return(false));

    auth::repositories::AuthStateUpdate state;
    state.failedAttempts = 3;
    state.lockedUntil = std::make_optional<std::string>("2026-01-
01T00:00:00Z");

    EXPECT_CALL(userRepo,
incrementFailedAttemptsAndMaybeLock("u1"))
        .WillOnce(Return(state));

    auth::services::LoginService      svc(userRepo,      sessionRepo,
passwordVerifier, refreshGen, issuer);

    auth::services::LoginResult out;
    const auto err = svc.login(std::nullopt,

```

```

std::make_optional<std::string>("MK"),
                                "wrong",
                                "dev-1",
                                std::nullopt,
                                std::nullopt,
                                out);

    ASSERT_TRUE(err.has_value());
    EXPECT_EQ(*err, auth::services::LoginError::Locked);
    ASSERT_TRUE(out.lockedUntil.has_value());
}

TEST(LoginServiceTest,
    Success_ResetsAuthState_RevokesDeviceSessions_CreatesSession_ReturnsTokens)
{
    auth::repositories::MockUserRepository userRepo;
    auth::repositories::MockSessionRepository sessionRepo;
    auth::security::MockPasswordVerifier passwordVerifier;
    auth::crypto::MockRefreshTokenGenerator refreshGen;
    auth::jwt::MockAccessTokenIssuer issuer;

    auth::models::User u;
    u.id = "u1";
    u.masterKey = "MK";
    u.emailVerified = true;
    u.passwordHash = "ph";
    u.tokenVersion = 7;

    EXPECT_CALL(userRepo, findByMasterKey("MK"))
        .WillOnce(Return(u));

    EXPECT_CALL(passwordVerifier, verify("123", "ph"))
        .WillOnce(Return(true));

    EXPECT_CALL(userRepo, resetAuthState("u1"))
        .Times(1);

    EXPECT_CALL(sessionRepo, revokeAllForDevice("u1", "dev-1"))
        .WillOnce(Return(1));

    auth::crypto::RefreshTokenPair pair;
    pair.raw = "r1";
    pair.hash = "h1";
    EXPECT_CALL(refreshGen, generate())
        .WillOnce(Return(pair));

    auth::models::Session s;
    s.id = "sid1";
    s.userId = "u1";

    EXPECT_CALL(issuer, createAccessToken("u1", "sid1", 7))

```

```

        .WillOnce(Return("jwt1"));
    EXPECT_CALL(issuer, accessTtlSeconds())
        .WillOnce(Return(900));

    auth::services::LoginService      svc(userRepo,      sessionRepo,
passwordVerifier, refreshGen, issuer);

    auth::services::LoginResult out;
    const auto err = svc.login(std::nullopt,

std::make_optional<std::string>("MK"),
                                "123",
                                "dev-1",
                                std::nullopt,
                                std::nullopt,
                                out);

    EXPECT_FALSE(err.has_value());
    EXPECT_EQ(out.userId, "u1");
    EXPECT_EQ(out.sessionId, "sid1");
    EXPECT_EQ(out.accessToken, "jwt1");
    EXPECT_EQ(out.refreshToken, "r1");
    EXPECT_EQ(out.expiresIn, 900);
}

} // namespace

```

Приложение В
(рекомендуемое)
Графическая часть

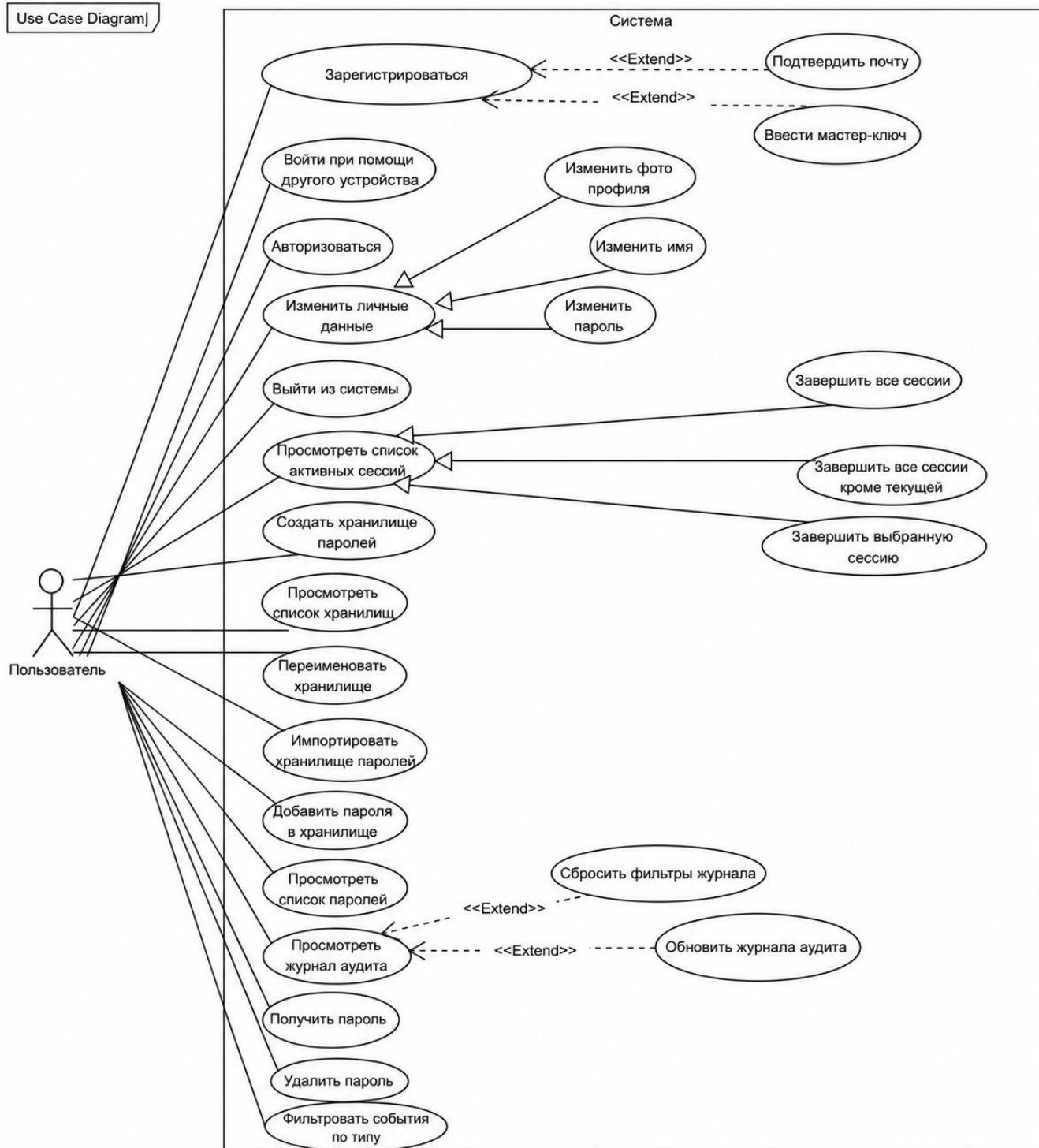


Рисунок В.1 – Диаграмма вариантов использования

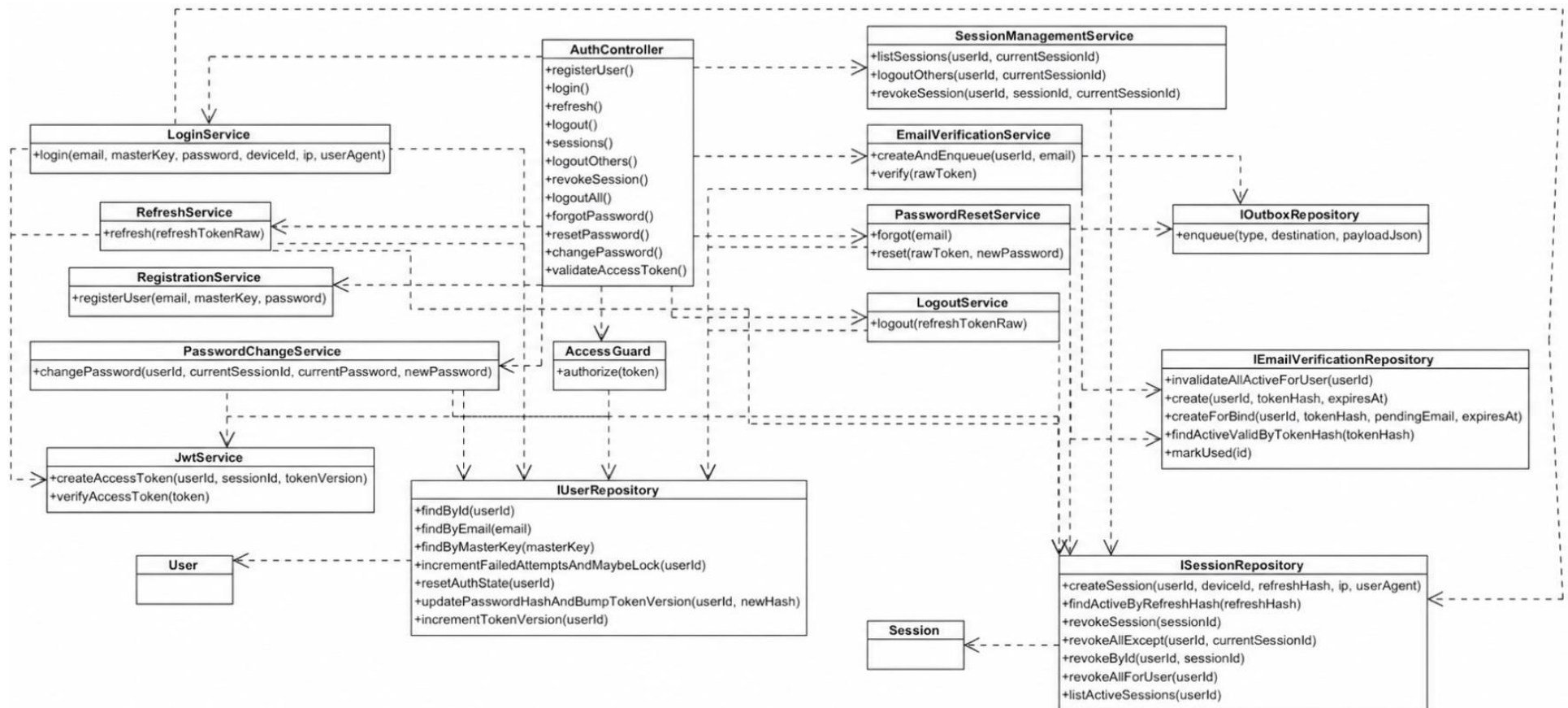


Рисунок В.2 – Диаграмма классов

Приложение Г
(рекомендуемое)
Листинг интеграционных тестов

Листинг Г.1 – test_e2e_auth_flow.py

```
import time
import uuid

from tests.integration.helpers.http import get, post_json
from tests.integration.helpers.tokens import
get_latest_outbox_token

def _wait_outbox_token(db, msg_type: str, destination: str,
timeout_seconds: int = 10) -> str:
    deadline = time.time() + timeout_seconds
    while time.time() < deadline:
        token = get_latest_outbox_token(db, msg_type,
destination=destination)
        if token:
            return token
        time.sleep(0.2)
    raise AssertionError(f"Outbox token not found: type={msg_type}
destination={destination}")

def test_e2e_auth_flow(base_url, db):
    email = f"it9_{uuid.uuid4().hex[:12]}@e2e.local"
    password = "12345678"

    st, body = post_json(f"{base_url}/auth/register", {"email":
email, "password": password})
    assert st == 201, body
    assert body.get("status") == "pending_email_verification"

    token = _wait_outbox_token(db, "email_verification",
destination=email)

    st, body = post_json(f"{base_url}/auth/email/verify",
{"token": token})
    assert st == 200, body
    assert body.get("ok") is True

    st, body = post_json(
        f"{base_url}/auth/login",
        {"email": email, "password": password, "deviceId": "it9-
dev-1"},
    )
    assert st == 200, body
    access1 = body.get("accessToken")
    refresh1 = body.get("refreshToken")
    assert access1 and refresh1

    st, body = post_json(f"{base_url}/auth/refresh",
{"refreshToken": refresh1})
    assert st == 200, body
```

```

refresh2 = body.get("refreshToken")
access2 = body.get("accessToken")
assert refresh2 and access2 and refresh2 != refresh1

st, body = post_json(f"{base_url}/auth/refresh",
{"refreshToken": refresh1})
assert st == 401, body
assert body.get("error") == "invalid_refresh"

st, body = post_json(f"{base_url}/auth/logout",
{"refreshToken": refresh2})
assert st == 200, body
assert body.get("ok") is True

st, body = post_json(f"{base_url}/auth/refresh",
{"refreshToken": refresh2})
assert st == 401, body
assert body.get("error") == "invalid_refresh"

st, body = get(f"{base_url}/health")
assert st == 200

```

Листинг Г.2 – test_e2e_device_login.py

```

import time
import uuid

from tests.integration.helpers.http import get, post_json
from tests.integration.helpers.tokens import
get_latest_outbox_token

def _wait_outbox_token(db, msg_type: str, destination: str,
timeout_seconds: int = 10) -> str:
    deadline = time.time() + timeout_seconds
    while time.time() < deadline:
        token = get_latest_outbox_token(db, msg_type,
destination=destination)
        if token:
            return token
        time.sleep(0.2)
    raise AssertionError(f"Outbox token not found: type={msg_type}
destination={destination}")

def test_e2e_device_login_flow(base_url, db):
    email = f"it9_{uuid.uuid4().hex[:12]}@dl.local"
    password = "12345678"

    st, body = post_json(f"{base_url}/auth/register", {"email":
email, "password": password})
    assert st == 201, body

```

```

    ver_token = _wait_outbox_token(db, "email_verification",
destination=email)
    st, body = post_json(f"{base_url}/auth/email/verify",
{"token": ver_token})
    assert st == 200, body

    st, body = post_json(
        f"{base_url}/auth/login",
        {"email": email, "password": password, "deviceId": "it9-
dev-b"},
    )
    assert st == 200, body
    access_b = body.get("accessToken")
    assert access_b

    st, body = post_json(
        f"{base_url}/auth/device-login/start",
        {"deviceId": "it9-dev-a"},
    )
    assert st == 201, body
    request_id = body.get("requestId")
    code = body.get("code")
    assert request_id and code

    st, body = post_json(
        f"{base_url}/auth/device-login/approve",
        {"requestId": request_id, "code": code},
        headers={"Authorization": f"Bearer {access_b}"},
    )
    assert st == 200, body
    assert body.get("ok") is True

    st, body = get(
        f"{base_url}/auth/device-
login/poll?requestId={request_id}&code={code}",
    )
    assert st == 200, body
    assert body.get("status") == "approved"
    access_a = body.get("accessToken")
    refresh_a = body.get("refreshToken")
    assert access_a and refresh_a

    st, body = get(
        f"{base_url}/auth/device-
login/poll?requestId={request_id}&code={code}",
    )
    assert st == 400, body
    assert body.get("error") == "invalid_or_expired_request"

    st, body = get(
        f"{base_url}/auth/sessions",
        headers={"Authorization": f"Bearer {access_a}"},

```

```
)
assert st == 200, body
assert body.get("ok") is True
```

Листинг Г.3 – test_e2e_password_reset.py

```
import time
import uuid

from tests.integration.helpers.http import get, post_json
from tests.integration.helpers.tokens import
get_latest_outbox_token

def _wait_outbox_token(db, msg_type: str, destination: str,
timeout_seconds: int = 10) -> str:
    deadline = time.time() + timeout_seconds
    while time.time() < deadline:
        token = get_latest_outbox_token(db, msg_type,
destination=destination)
        if token:
            return token
        time.sleep(0.2)
    raise AssertionError(f"Outbox token not found: type={msg_type}
destination={destination}")

def
test_e2e_password_reset_invalidates_refresh_and_access_by_tv(base_
url, db):
    email = f"it9_{uuid.uuid4().hex[:12]}@reset.local"
    password = "12345678"

    st, body = post_json(f"{base_url}/auth/register", {"email":
email, "password": password})
    assert st == 201, body

    ver_token = _wait_outbox_token(db, "email_verification",
destination=email)
    st, body = post_json(f"{base_url}/auth/email/verify",
{"token": ver_token})
    assert st == 200, body

    st, body = post_json(
        f"{base_url}/auth/login",
        {"email": email, "password": password, "deviceId": "it9-
dev-reset"},
    )
    assert st == 200, body
    access1 = body.get("accessToken")
    refresh1 = body.get("refreshToken")
    assert access1 and refresh1
```

```

    st, body = post_json(f"{base_url}/auth/password/forgot",
{"email": email})
    assert st == 200, body
    assert body.get("ok") is True

    reset_token = _wait_outbox_token(db, "password_reset",
destination=email)

    st, body = post_json(
        f"{base_url}/auth/password/reset",
        {"token": reset_token, "newPassword": "newpass123"},
    )
    assert st == 200, body
    assert body.get("ok") is True

    st, body = post_json(f"{base_url}/auth/refresh",
{"refreshToken": refresh1})
    assert st == 401, body
    assert body.get("error") == "invalid_refresh"

    st, body = get(
        f"{base_url}/auth/sessions",
        headers={"Authorization": f"Bearer {access1}"},
    )
    assert st == 401, body
    assert body.get("error") == "invalid_access"

```

Приложение Д
(рекомендуемое)

Графическое представление этапов разработки

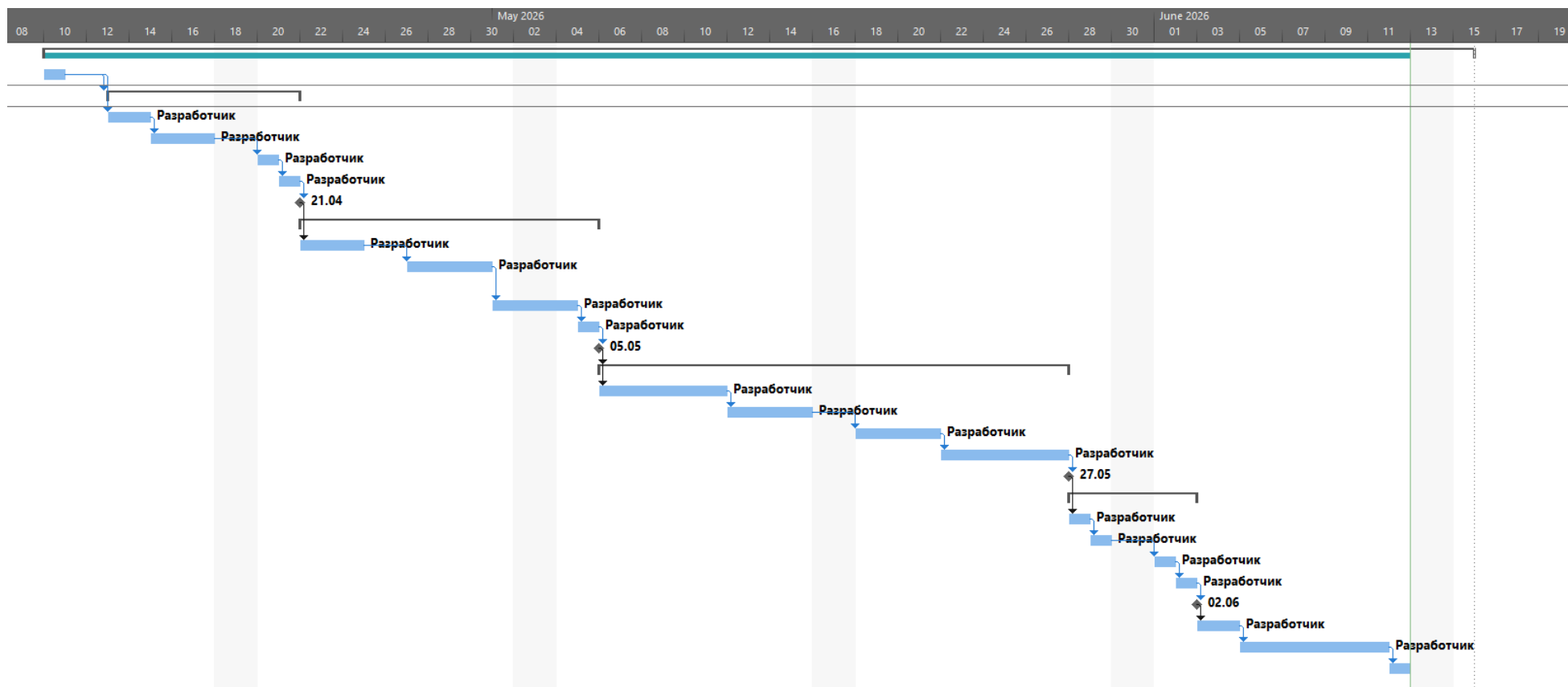


Рисунок Д.1 – Диаграмма Ганта

Task Mode ▾	Task Name ▾	Duration ▾	Start ▾	Finish ▾	Resource Names ▾	Cost ▾
	Разработка приложения	47 days?	Fri 10.04.26	Mon 15.06.26		\$2 464,00
	Начало проекта	8 hrs	Fri 10.04.26	Fri 10.04.26		\$0,00
	Анализ предметной области приложения	7 days	Mon 13.04.26	Tue 21.04.26		\$392,00
	Изучение предметной области	2 days	Mon 13.04.26	Tue 14.04.26	Разработчик	\$112,00
	Сравнение аналогов	3 days	Wed 15.04.26	Fri 17.04.26	Разработчик	\$168,00
	Составление требований к системе	1 day	Mon 20.04.26	Mon 20.04.26	Разработчик	\$56,00
	Постановка задачи на разработку	1 day	Tue 21.04.26	Tue 21.04.26	Разработчик	\$56,00
	Конец анализа предметной области	0 days	Tue 21.04.26	Tue 21.04.26		\$0,00
	Проектирование приложения	10 days	Wed 22.04.26	Tue 05.05.26		\$560,00
	Проектирование архитектуры	3 days	Wed 22.04.26	Fri 24.04.26	Разработчик	\$168,00
	Проектирование концептуальной модели данных	4 days	Mon 27.04.26	Thu 30.04.26	Разработчик	\$224,00
	Проектирование логической модели данных	2 days	Fri 01.05.26	Mon 04.05.26	Разработчик	\$112,00
	Выбор технологий разработки	1 day	Tue 05.05.26	Tue 05.05.26	Разработчик	\$56,00
	Конец проектирования приложения	0 days	Tue 05.05.26	Tue 05.05.26		\$0,00
	Разработка приложения	16 days	Wed 06.05.26	Wed 27.05.26		\$896,00
	Разработка архитектуры	4 days	Wed 06.05.26	Mon 11.05.26	Разработчик	\$224,00
	Разработка слоев представления данных	4 days	Tue 12.05.26	Fri 15.05.26	Разработчик	\$224,00
	Разработка слоев бизнес-логики	4 days	Mon 18.05.26	Thu 21.05.26	Разработчик	\$224,00
	Разработка слоев обращения к базе данных	4 days	Fri 22.05.26	Wed 27.05.26	Разработчик	\$224,00
	Конец разработки приложения	0 days	Wed 27.05.26	Wed 27.05.26		\$0,00
	Контроль качества приложения	4 days	Thu 28.05.26	Tue 02.06.26		\$224,00
	Выбор метрик программного кода	1 day	Thu 28.05.26	Thu 28.05.26	Разработчик	\$56,00
	Модульное тестирование	1 day	Fri 29.05.26	Fri 29.05.26	Разработчик	\$56,00
	Интеграционное тестирование	1 day	Mon 01.06.26	Mon 01.06.26	Разработчик	\$56,00
	Вычисление метрик программного кода	1 day	Tue 02.06.26	Tue 02.06.26	Разработчик	\$56,00
	Конец контроля качества приложения	0 days	Tue 02.06.26	Tue 02.06.26		\$0,00
	Составление глоссария	16 hrs	Wed 03.06.26	Thu 04.06.26	Разработчик	\$112,00
	Составление отчета	5 days	Fri 05.06.26	Thu 11.06.26	Разработчик	\$280,00
	Конец проекта	1 day	Fri 12.06.26	Fri 12.06.26		\$0,00

Рисунок Д.2 – Список задач проекта