

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

**Web-портал поиска родственников и ведения
генеалогического дерева семьи**

Студент

(подпись, дата)

Малькова Анастасия Сергеевна
(ФИО полностью)

Руководитель

(подпись, дата)

Шибанов С.В.

(фамилия, инициалы)

Нормоконтролёр

(подпись, дата)

Такташкин Д.В.

(фамилия, инициалы)

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

Заведующий кафедрой

(подпись)

Козлов А.Ю.

(фамилия, инициалы)

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

Секретарь ГЭК

(подпись)

Попова Н.А.

(фамилия, инициалы)

Пенза, 2025

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
«20» декабря 2024 г

**ЗАДАНИЕ
НА ВЫПУСКНУЮ КВАЛИФИКАЦИОННУЮ РАБОТУ
БАКАЛАВРА**

1. Студент Малькова Анастасия Сергеевна гр. 21ВП2 факультета ВТ
<Фамилия, имя, отчество полностью>

2. Тема работы Web-портал поиска родственников и ведения генеалогического дерева семьи

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

3. Руководитель работы к.т.н., доцент кафедры «МОиПЭВМ» Шибанов С.В.

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

1) *Назначение: Web-портал поиска родственников и ведения генеалогического дерева семьи*

2) *Среда выполнения: Windows 10.*

Языковые средства разработки: Java, Javascript.

Инструментальные средства: IntelliJ IDEA, Webstorm, Swagger, Visual Paradigm, Microsoft Project.

Технологии разработки: ООП, сервисно-ориентированный подход.

3) *Основные функции разрабатываемого приложения:*

а) создание дерева семьи;

б) ввод и изменение данных о родственниках;

в) поиск родственников;

г) добавление истории жизни человека;

д) оценивание достоверности данных;

е) резервное копирование базы данных.

5. Перечень подлежащих разработке вопросов

1) Анализ предметной области и постановка задачи

2) Анализ требований на разработку

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%	08.04.2025 – 14.04.2025	
2	Проектирование программных средств	20%	15.04.2025 – 25.04.2025	
3	Реализация программных средств	30%	26.04.2025 – 12.05.2025	
4	Тестирование программных средств	20%	13.05.2025 – 22.05.2025	
5	Оформление пояснительной записки	20%	23.05.2025 – 03.06.2025	

Дата выдачи задания «20» декабря 2024г.

Руководитель бакалаврской работы _____
(подпись, дата)

Задание к исполнению принял студент _____
(подпись, дата)

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

Реферат

Пояснительная записка содержит 149 листов, 34 рисунка, 18 таблиц, 33 использованных источника, 8 приложений.

ВЕБ ПРИЛОЖЕНИЕ, РОДСТВЕННИКИ, ГЕНЕАЛОГИЧЕСКОЕ ДЕРЕВО, REST, MS SQL SERVER, JAVA, SPRING, JS, REACT, UML, MVC, AXIOS, TAILWIND

Объект разработки – веб-приложение для поиска родственников и ведения генеалогического дерева семьи.

Цель выпускной квалификационной работы – разработать веб-приложение для поиска родственников и ведения генеалогических древ, обеспечивающее пользователям возможность создания, редактирования и визуализации семейных связей, хранения биографических данных, а также взаимодействия с другими пользователями при помощи комментариев.

Технологии разработки – разработка проводилась на языках программирования Java с использованием фреймворка Spring, JavaScript с использованием фреймворка React. В качестве сред разработки использовались редакторы программного кода IntelliJ IDEA и WebStorm. В качестве СУБД была использована СУБД MS SQL Server.

Среда исполнения – операционная система Windows 10, браузер Chrome.

Результаты работы – в ходе выполнения выпускной квалификационной работы был проведен анализ предметной области с обзором рынка, разработано и протестировано веб-приложение для поиска родственников и ведения родословных. Все требования к разработке были соблюдены.

					ПГУ 09.03.04 – 08BP212.18 ПЗ			
Изм.	Лист	№ докум.	Подп.	Дата	Web-портал поиска родственников и ведения генеалогического дерева семьи.	Лит.	Лист	Листов
Разраб.		Малькова А.С.					4	149
Пров.		Шибанов С.В.			Пояснительная записка	Группа 21ВП2		
Н. контр.		Такташкин Д.В.						
Утв.		Козлов А.Ю.						

Содержание

Введение	8
1 Анализ предметной области и постановка задачи на разработку	10
1.1 Анализ предметной области web-портала поиска родственников и ведения генеалогического дерева семьи	10
1.2 Сравнительный анализ программных продуктов аналогичного назначения	11
1.2.1 Международный сервис «MyHeritage»	11
1.2.2 Российский сервис «Famiry»	13
1.2.3 Программное приложение «Дерево Жизни»	14
1.2.4 Сервис «FamilySearch»	15
1.2.5 Форум «ВГД»	16
1.2.6 Сравнение альтернативных платформ	17
1.3 Постановка задачи на разработку web-портала поиска родственников и ведения генеалогического дерева семьи	18
1.4 Планирование процесса разработки и расчет бюджета	22
2 Анализ требований на разработку web-портала поиска родственников и ведения генеалогического дерева семьи	29
2.1 Анализ функциональных требований	29
2.2 Анализ архитектурных требований	34
2.3 Анализ нефункциональных требований	36
2.3.1 Эксплуатационные характеристики	36
2.3.2 Структурные характеристики	37
2.3.3 Сквозные характеристики	38
2.4 Анализ требований к данным	38

2.4.1	Определение концептуальной модели базы данных	38
2.4.2	Определение логической модели базы данных	41
2.5	Анализ требований к пользовательскому интерфейсу	45
2.6	Обоснование выбора технологий и средств разработки	47
3	Разработка программного обеспечения web-портала поиска родственников и ведения генеалогического дерева семьёй	49
3.1	Разработка физической модели базы данных	49
3.2	Разработка пользовательского интерфейса	51
3.3	Разработка программного обеспечения	55
3.3.1	Разработка программного обеспечения серверной части	55
3.3.2	Реализация клиентской части приложения	60
4	Контроль качества программного продукта	62
4.1	Обоснование выбора модели тестирования	62
4.2	Тестирование web-портала поиска родственников и ведения генеалогического дерева семьи	63
4.2.1	Функциональное тестирование. Тестирование чёрным ящиком ...	63
4.2.2	Интеграционное тестирование	64
4.2.3	Нагрузочное тестирование	66
4.3	Оценка метрик кода	68
4.3.1	Метрика SLOC	69
4.3.2	Метрика Холстеда	70
	Заключение	72
	Список использованных источников	74
	Приложение А (справочное) Глоссарий	78

Приложение Б (обязательное) Фрагменты исходного кода серверной части web-портала	81
Приложение В (обязательное) Фрагменты исходного кода клиентской части web-портала	100
Приложение Г (рекомендуемое) Диаграммы вариантов использования	117
Приложение Д (рекомендуемое) Табличное описание атрибутов сущностей системы	122
Приложение Е (рекомендуемое) Пользовательский интерфейс приложения.....	129
Приложение Ж (рекомендуемое) Результаты функционального тестирования.....	142
Приложение З (рекомендуемое) Исходный код функций тестирования	146

Введение

В настоящее время всё больше людей интересуются своей семейной историей и составляют родословные. Это особенно важно сейчас, когда семьи часто живут далеко друг от друга и связи между поколениями ослабевают.

Правительство Российской Федерации помогает развивать цифровые технологии для сохранения истории. В рамках программы «Цифровая экономика» и национального проекта «Культура» уделяется особое внимание оцифровке архивных данных, что делает поиск информации значительно проще [1, 2]. Каждый год оцифровываются миллионы страниц историй из архивов по всей стране. Также стоит отметить инициативу федеральной службы безопасности России, которая к 80-летию Победы в Великой Отечественной войне опубликовала рассекреченные архивные материалы [3]. Эти документы, содержащие сотни тысяч записей о наградах и службе военнослужащих, могут помочь многим людям восстановить историю своей семьи.

Хотя значительная часть материалов теперь доступна в цифровом формате, процесс самостоятельного составления родословной остаётся непростой задачей. Многие генеалогические сервисы требуют знаний и опыта работы с архивными данными. Также стоит учесть, что большинство зарубежных сервисов не адаптированы для работы с русскоязычными документами и только малая часть генеалогических приложений предоставляет возможность для поиска родственников и сохранения их истории жизни.

Целью данной работы является создание интуитивно понятного сервиса с поддержкой ведения истории жизни человека, что сделает семейную родословную интереснее, подробнее и доступнее для каждого. А разработка данного решения отвечает не только потребностям общества, но и соответствует стратегическим направлениям развития цифровых технологий в России.

В ходе выполнения выпускной квалификационной работы был проведен анализ предметной области с обзором рынка, планированием и расчетом бюджета на разработку. Также был произведён анализ и выбор используемых

технологий. Разработка произведена с использованием Java Spring и React JS, а также MySQL. Для авторизации и аутентификации реализован механизм JWT. Взаимодействие клиентской и серверной частей производится при помощи Axios. Визуализация древ создается с помощью библиотеки React Flow.

Реализованный веб-сервис позволяет пользователю создавать редактировать дерево семьи, добавлять родственников и устанавливать между ними различные виды связей. Также позволяет вести историю жизни каждого члена семьи с возможностью прикрепления медиафайлов. Помимо этого, пользователи могут находить новых родственников и обмениваться известными им данными при помощи комментариев.

В рамках тестирования были проведены:

- функциональное тестирование;
- интеграционное тестирование;
- нагрузочное тестирование;
- регрессионное тестирование.

Были вычислены и проанализированы метрики кода.

Таким образом, результатом работы стал удобный, современный инструмент для поиска родственников и ведения генеалогических древ.

1 Анализ предметной области и постановка задачи на разработку

Анализ предметной области – это детальное изучение сферы, для которой создается программное обеспечение. Этот этап помогает выявить важные термины, принципы работы и порядок действий, которые нужно учитывать при разработке системы. Если его пропустить, готовый продукт может оказаться бесполезным: он может не удовлетворять потребности пользователей или быть неудобным в использовании [4].

Анализ предметной области помогает понять, какие функции необходимо реализовать, а также помогает избежать серьезных ошибок, возникающих при недостаточном понимании особенностей сферы.

1.1 Анализ предметной области web-портала поиска родственников и ведения генеалогического дерева семьи

Существует множество систем для генеалогических исследований. Они предназначены для сбора, систематизации и анализа истории семьи. Из-за растущих потребностей пользователей данные сервисы постоянно эволюционируют.

Платформы для изучения родословной обычно включают три основных функции:

- инструменты для составления схемы родственных связей в виде генеалогического дерева;
- системы для хранения документов, фотографий, информации о членах семьи и других материалов;
- механизмы для поиска совпадений и установления связей между разными деревьями.

Таким образом, современный web-портал для генеалогических исследований должен обеспечивать простой в использовании интерфейс в сочетании с безопасным хранением и обработкой информации. Эти требования задают основу для разработки системы и её функций.

1.2 Сравнительный анализ программных продуктов аналогичного назначения

Программы для составления родословных можно разделить на два типа:

- онлайн программы, позволяющие искать родственников в базах данных и архивах;
- оффлайн программы, предназначенные для локального использования.

Оффлайн версии более всего подходят для тех, кто заботиться о конфиденциальности информации.

В этом разделе сравниваются популярные сервисы, чтобы понять:

- какие задачи лучше решает каждый из них;
- какое решение выгоднее;
- насколько они удобны для русскоязычных пользователей.

Результаты сравнения позволят определить оптимальный инструмент в зависимости от задач пользователя. Также позволят учесть существующие недостатки и преимущества при проектировании и разработке собственного веб-приложения.

1.2.1 Международный сервис «MyHeritage»

MyHeritage – это популярный международный сервис, который помогает людям изучать историю своей семьи [5].

Главное его преимущество – удобный инструмент для построения генеалогического дерева. Пользователи могут добавлять родственников, указывать важные даты и события, прикреплять фотографии и документы. Интерфейс интуитивно понятен. Дерево автоматически формируется по мере добавления информации.

Особенно удобна функция автоматического поиска – система сама предлагает возможных родственников, основываясь на уже внесённых данных.

Ещё один плюс – возможность сделать ДНК-тест, который покажет этническое происхождение и поможет найти генетических родственников.

Пример дерева представлен на рисунке 1.

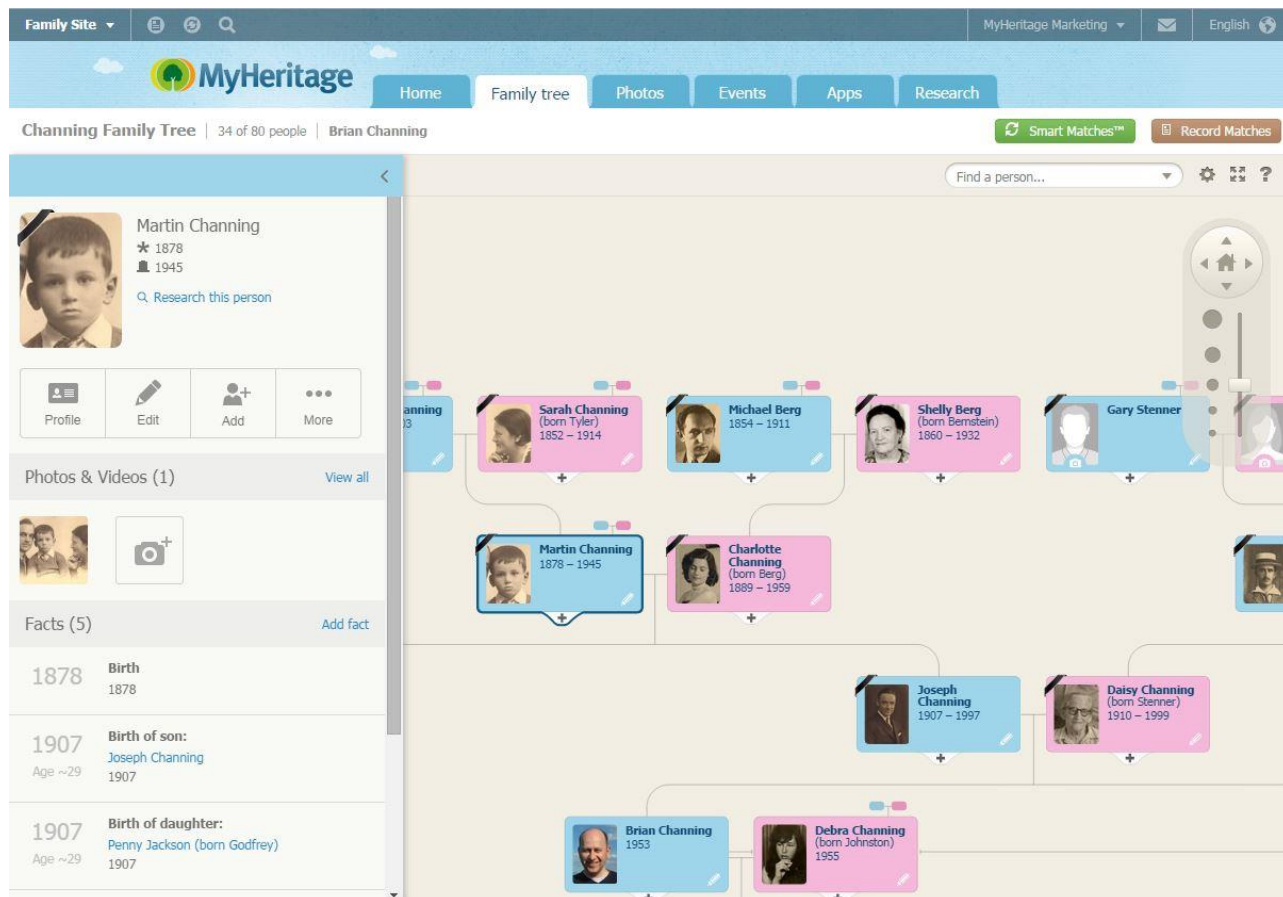


Рисунок 1 – Интерфейс сервиса «MyHeritage»

Из рисунка видно, что структура дерева аккуратная, легка для разбора, есть разделение по гендеру с помощью цветового решения.

Однако не обошлось без недостатков. Подписка на этот сервис платная. Есть пробный период в 14 дней, но с ограниченным функционалом. Кроме того, у российских пользователей есть проблемы с использованием данного сервиса. «MyHeritage» ушел из России. Аккаунты россиян активно блокируются, зайти на сайт можно через VPN, но есть сложности с оплатой из-за международных ограничений. Ещё один минус – информация об умерших родственниках становится доступной для других пользователей, что не всем нравится. А также,

другие пользователи не могут комментировать данные о человеке, что может быть полезным для оценки их достоверности.

1.2.2 Российский сервис «Famiry»

Famiry – это современный российский IT-сервис для изучения семейной истории, объединяющий технологии и генеалогию. Название проекта происходит от английских слов family (семья) и history (история), отражая его суть – цифровое сохранение родословных. Платформа позволяет создавать интерактивные семейные деревья, искать предков. Разработана командой экспертов в области генеалогии, архивного дела и IT с 20-летним опытом. Веб-сервис сочетает международные стандарты работы с родословными и специальные решения для российских пользователей [6].

Сервис помимо создания деревьев предоставляет возможность ведения истории событий в жизни человека с добавлением фотографий. Также возможен поиск родственников по базе данных. Имеет современный интерфейс, которым приятно пользоваться (рисунок 2).

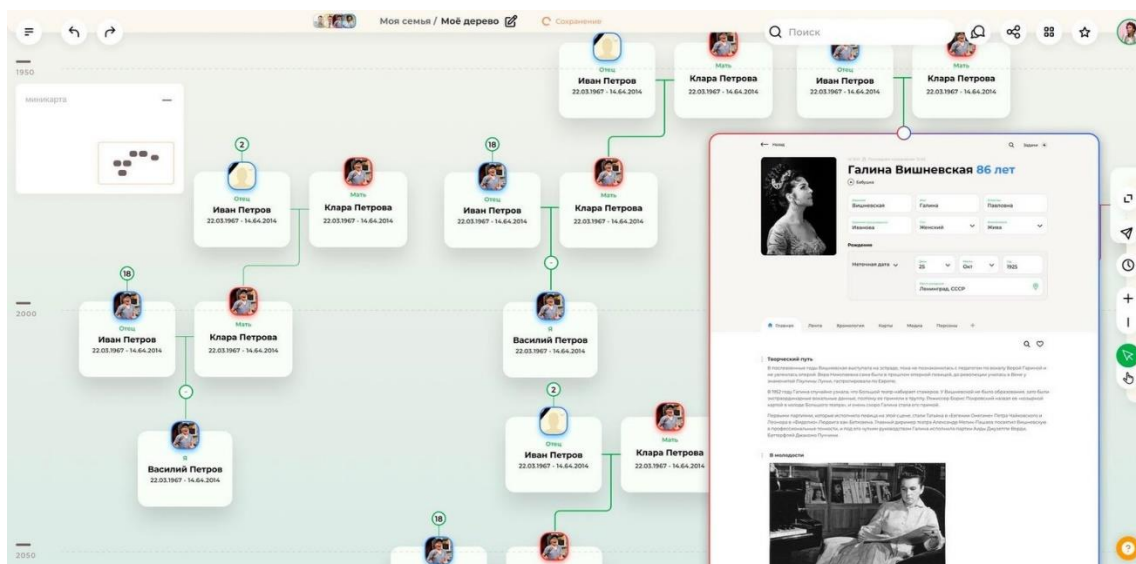


Рисунок 2 – Интерфейс платформы «Famiry» [7]

Но недостатки присутствуют и здесь. Деревья доступны для просмотра только членам семьи. Но при изучении своей родословной и расширении её

хотелось бы посмотреть известные данные родственников в других поколениях. Кроме этого, отсутствует возможность комментирования данных.

У сервиса есть как бесплатный доступ к ограниченному функционалу, так и платная подписка с дополнительными возможностями. В бесплатной версии ограничено добавление персон в дерево до 100 человек, можно формировать только до 4-х поколений родства, загрузка до 50 фотографий. И можно создать только одно дерево, что совсем неудобно при ведении родословных разных ветвей семьи.

1.2.3 Программное приложение «Дерево Жизни»

Дерево Жизни – это российская десктопная программа для построения генеалогических деревьев, работающая на операционной системе Windows [8].

Данное приложение идеально подойдет для тех, кто беспокоится о конфиденциальности данных, так как они хранятся локально на вашем компьютере. Также «Дерево Жизни» поддерживает неограниченное число персон и поколений. Можно выбрать графическую схему для отрисовки дерева (круговое, вертикальное, горизонтальное). Поддерживает добавление фото и ведение хронологии событий.

Пример созданной родословной приведён на рисунке 3.

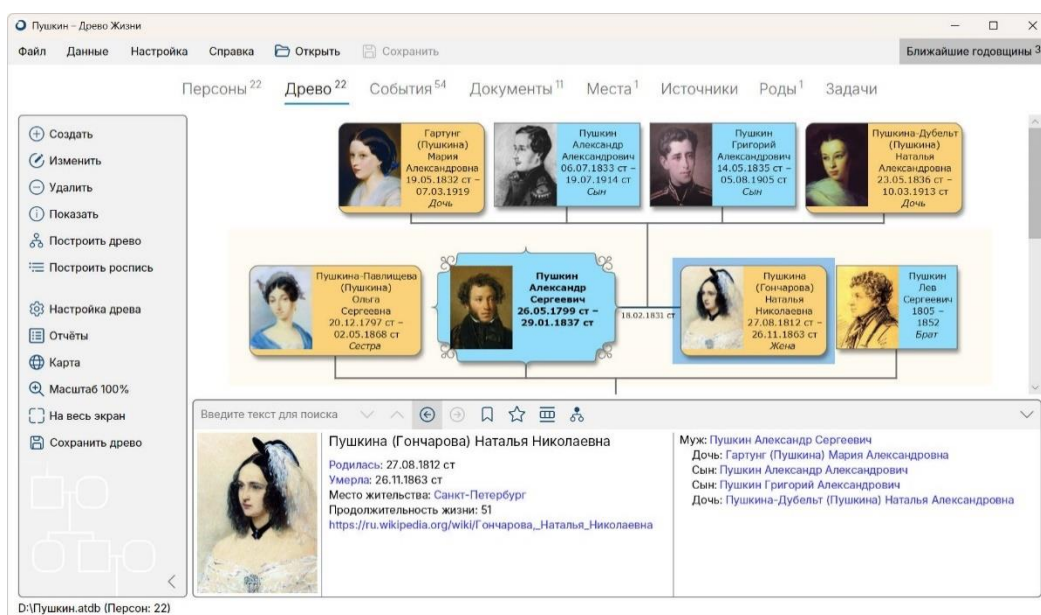


Рисунок 3 – Интерфейс приложения «Дерево Жизни»

Как плюсом, так и минусом одновременно является отсутствие подписок, но программа всё равно не бесплатная. Для работы необходимо единоразово купить бессрочную лицензию. Также есть демоверсия с ограничением на 40 персон. Однако, в бесплатной версии нельзя сохранить проект после закрытия программы. Все данные стираются, что совершенно неудобно.

К недостаткам можно отнести отсутствие версий для macOS и Linux и облачной синхронизации, из-за чего для доступа с разных устройств нужен ручной перенос файлов. Работа через эмуляторы возможна, но нестабильна. Также недоступны поиск родственников в общей базе и комментирование достоверности данных.

1.2.4 Сервис «FamilySearch»

FamilySearch – один из крупнейших бесплатных генеалогических веб-сервисов в мире. Это некоммерческий проект, предоставляющий доступ к миллиардам записей и инструментам для построения семейного дерева [9].

Функционал сайта полностью бесплатный и без ограничений. Большим плюсом является огромная база данных. Можно объединять данные с другими пользователями, например, если кто-то уже исследовал нужных предков. Есть возможность комментирования корректности данных любыми пользователями. Сайт является международным, поддерживает русский язык.

Также FamilySearch предоставляет доступ к оцифрованным архивным материалам, включая записи о рождении, браке, переписи населения и другим историческим инструментам.

Кроме того, в FamilySearch реализован механизм поиска совпадений с существующими профилями, что упрощает процесс расширения дерева.

К недостаткам можно отнести возможные сложности при работе с русскоязычными архивами, которые представлены не так широко, как западные источники.

Одно из заполненных генеалогических древ изображено на рисунке 4.

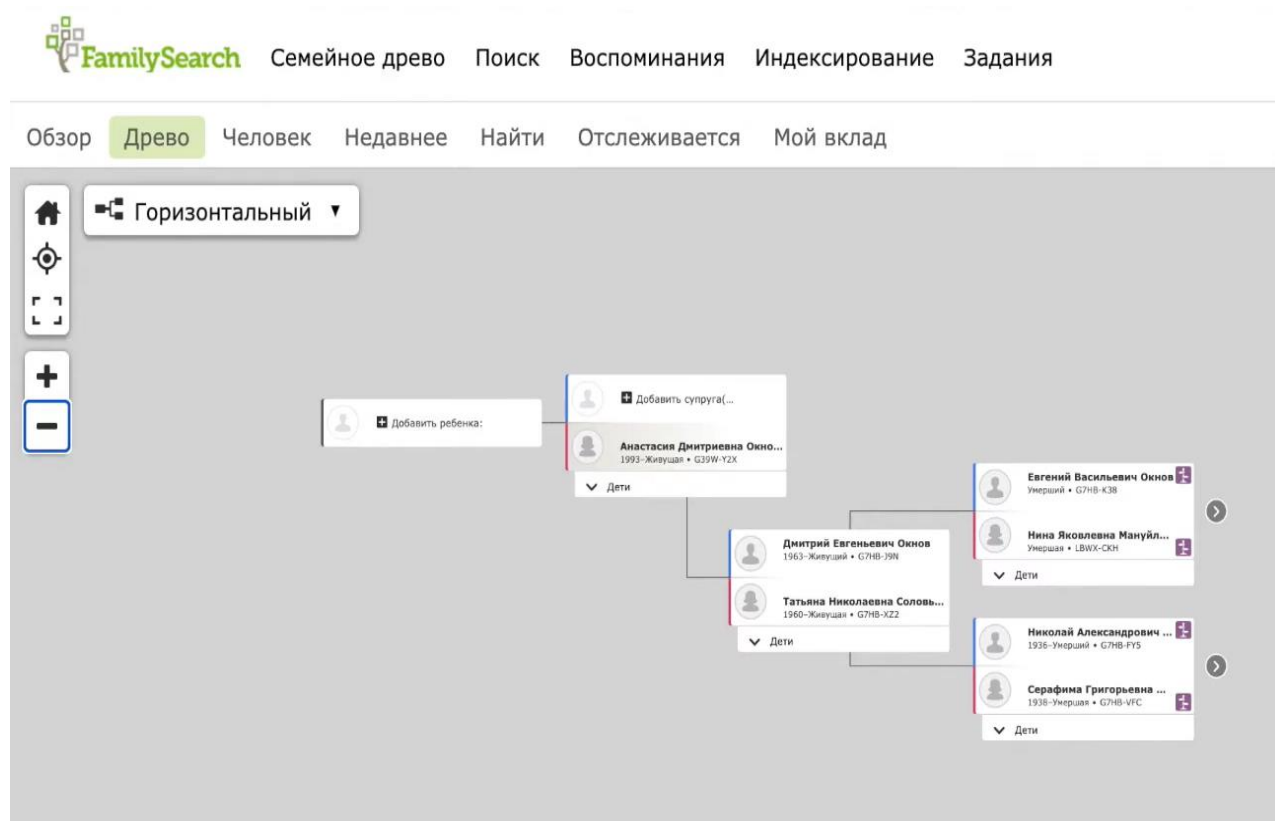


Рисунок 4 – Интерфейс сервиса «FamilySearch»

Но данный сервис недоступен для работы из России, можно использовать VPN. Также минусом «FamilySearch» является возможность объединения деревьев в одно общее дерево. Любой пользователь может изменить данные ваших предков.

1.2.5 Форум «ВГД»

Форум ВГД – крупнейшее некоммерческое сообщество генеалогов-любителей и профессионалов в России [10]. Все знания на форуме создают сами пользователи. Такого масштаба и детализации нет даже в государственных базах.

Хотя ВГД – это всё же форум, а не специализированный конструктор деревьев, здесь есть инструменты для систематизации родословных. Можно схематично изобразить дерево, но, честно говоря, интерфейс довольно устаревший и неудобный. Пример построенного дерева приведен на рисунке 5.

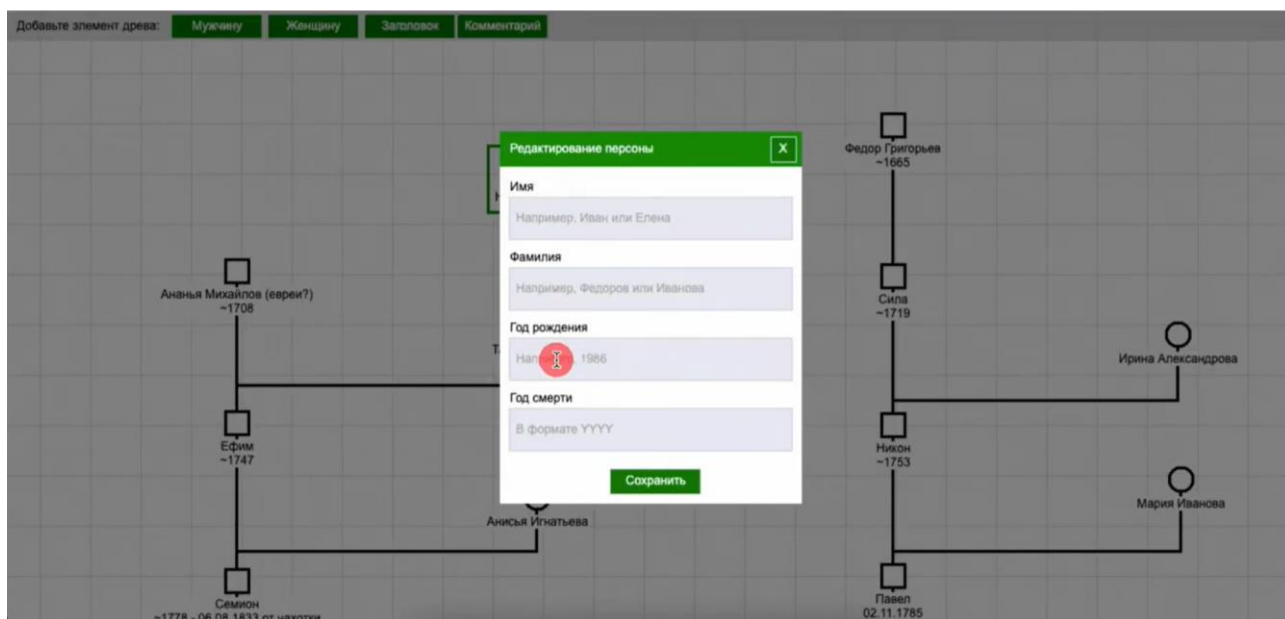


Рисунок 5 – Интерфейс функционала для построения генеалогического дерева на форуме «ВГД»

В сервисе нет возможности ведения истории жизни человека, деревья недоступны для просмотра другим пользователям.

1.2.6 Сравнение альтернативных платформ

На основании проведенного сравнения характеристик аналогичных разработок сформирована обобщающая таблица 1.

Таблица 1 – Сравнение аналогов

Сервис	MyHeritage	Famiry	Дерево Жизни	FamilySearch	Форум ВГД
Стоимость	1200 – 2000 руб./мес.	Бесплатно с ограничениями; Платная подписка 4400–10600 руб./год	Бесплатно до 40 персон; Лицензия 1900–6400 руб.	Бесплатно	Бесплатно
Наличие пробного периода	Да, 14 дней	Нет	Да, 30 дней	-	-
Тип приложения	Веб, десктоп, мобильное	Веб	Десктоп (Windows)	Веб, мобильное	Веб
Ведение нескольких деревьев	Да, до 250 шт.	С подпиской	С лицензией	Нет	Нет

Продолжение таблицы 1

Сервис	MyHeritage	Famiry	Дерево Жизни	FamilySearch	Форум ВГД
Поиск людей в общей базе данных	Да	Да	Нет	Да	Да
Комментирование	Нет	Нет	Нет	Да	Нет
Ведение истории жизни	Да	Да	Да	Да	Нет
Доступность в России	Нет	Да	Да	Нет	Да
Доступность деревьев к просмотру для всех	Нет	Нет	Нет	Да	Нет

Проведя анализ популярных генеалогических сервисов, был найден ряд серьёзных ограничений: международные платформы труднодоступны для россиян, а отечественные аналоги либо платные, либо имеют слишком устаревший функционал. Именно поэтому было принято решение о разработке нового приложения, которое устранит эти недостатки.

1.3 Постановка задачи на разработку web-портала поиска родственников и ведения генеалогического дерева семьи

Разрабатываемое веб-приложение должно обеспечить его пользователям набор удобных инструментов для хранения и управления данными о людях и их семейных связях.

Основными целями разработки являются:

- создание инструмента для хранения информации о человеке и его жизни;
- визуализация семейной структуры;
- создание инструмента для поиска родственников.

Необходимо предусмотреть возможность работы для двух категорий пользователей: обычный пользователь и администратор. Каждый тип

пользователей в системе имеет собственный набор функциональных возможностей в зависимости от своей роли.

Были определены следующие подсистемы:

- подсистема регистрации, аутентификации и авторизации – позволяет регистрироваться и входить в систему;
- подсистема управления данными о персонах – позволяет добавлять и редактировать информацию о людях;
- подсистема управления родственными связями – дает возможность создавать и редактировать связи между людьми в их семейных деревьях, но пользователи могут управлять только своими записями;
- подсистема поиска персон – позволяет искать людей по имени или другим параметрам, например, по дате рождения;
- подсистема ведения истории жизни персоны – позволяет управлять событиями из жизни персоны с возможностью добавления фото и видео;
- подсистема оценки достоверности информации – позволяет добавлять комментарии к информации о персоне любому пользователю;
- подсистема резервного копирования и восстановления данных – создает регулярные резервные копии, чтобы избежать потери данных, дает возможность восстанавливать данные в случае неисправности.

Большинство подсистем предназначены для использования пользователями, кроме последних двух. Подсистема оценки и достоверности информации предназначена для обеих категорий. А подсистема резервного копирования и восстановления данных ориентирована на использование только под ролью администратора. Подсистема авторизации и аутентификации доступна для пользователей независимо от их роли, в том числе и для незарегистрированных пользователей.

Сценарий выполнения основных информационных процессов представлен на рисунке 6 в виде диаграммы бизнес-процессов UML, созданной в case-средстве Visual Paradigm [11].

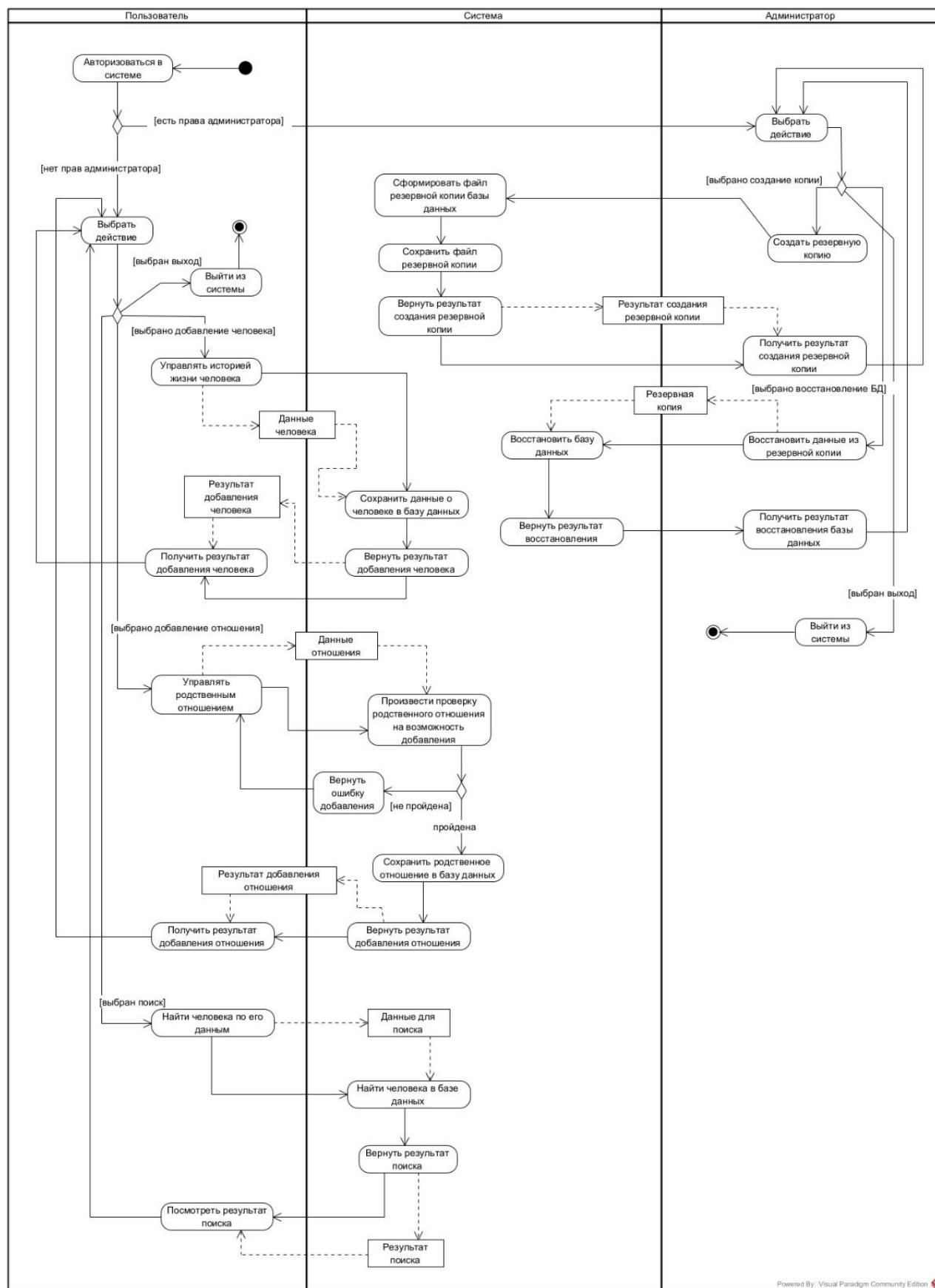


Рисунок 6 – Диаграмма бизнес-процессов, отображающая сценарий выполнения информационных процессов

На рисунке 7 представлено продолжение диаграммы бизнес-процессов.

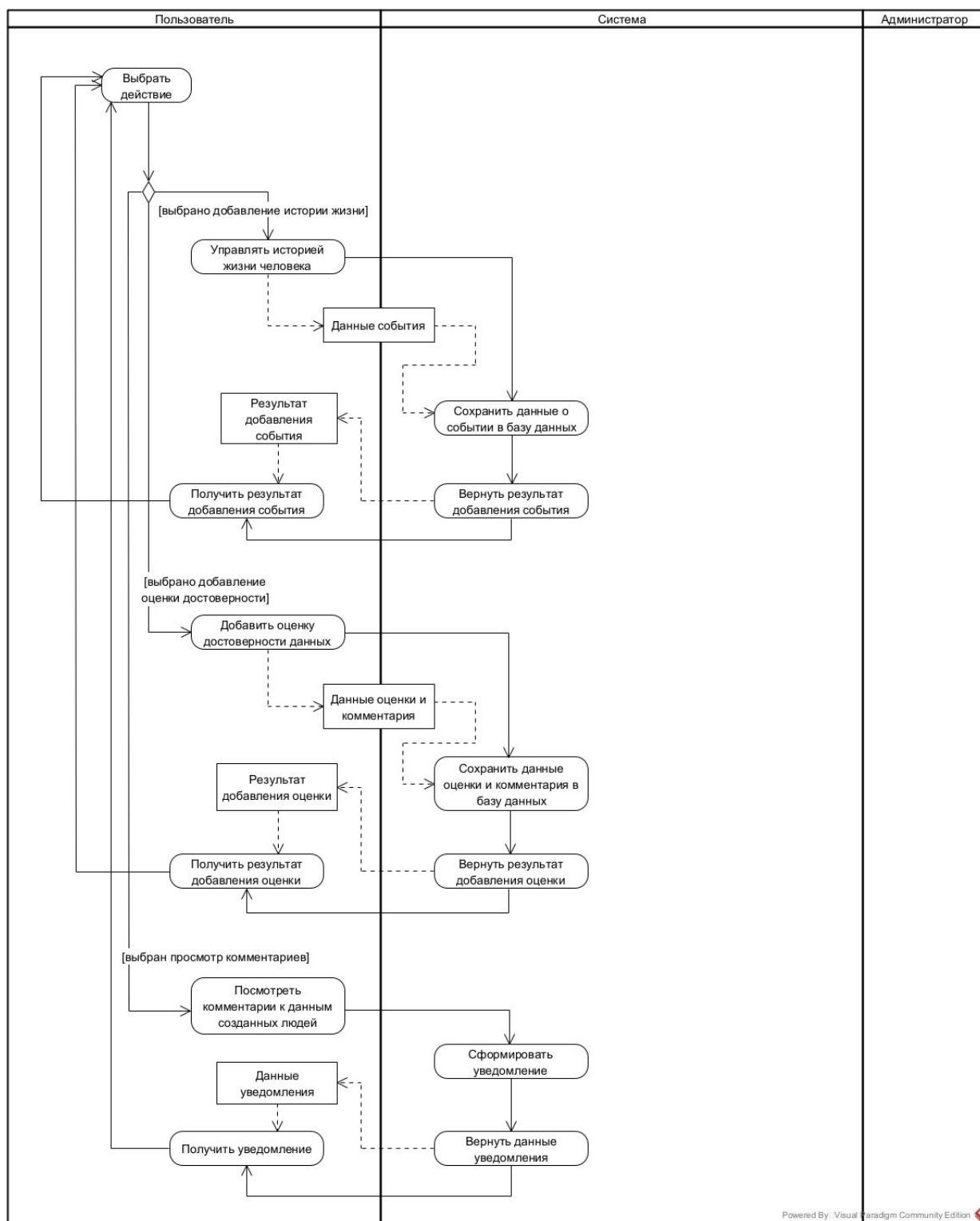


Рисунок 7 – Продолжение диаграммы бизнес-процессов

Сформулированные требования создают четкую основу для последующего проектирования архитектуры системы и разработки её компонентов. Поставленные задачи полностью соответствуют основной цели проекта –

созданию удобного и доступного инструмента для сохранения и изучения семейной истории.

1.4 Планирование процесса разработки и расчет бюджета

Качественное планирование – это основа успешного IT-проекта. Оно помогает организовать работу так, чтобы достичь целей с минимальными затратами времени и ресурсов, не теряя качество конечного продукта.

Эффективное планирование проекта включает несколько ключевых аспектов:

- четкая формулировка целей и задач;
- оценка необходимых ресурсов, включая человеческие и технические, и распределить их в соответствии с этапами реализации;
- прогнозирование потенциальных рисков и разработка плана их минимизации.

В IT-проектах эти шаги особенно актуальны при учёте динамичных технологических изменений и сложность интеграции систем.

Тщательное планирование способствует снижению вероятности превышения бюджета и сроков, уменьшению количества непредвиденных задач и обеспечению высокого качества конечного продукта. Поэтому инвестиции времени и усилий в тщательную подготовку проекта всегда оправдываются в процессе его выполнения [12 – 14].

В таблице 2 представлены ключевые задачи проекта. При формировании календарного графика работ принята шестидневная рабочая неделя.

Таблица 2 – Перечень задач проекта

№	Название задачи	Длительность	Предшественники
1	Начало проекта	–	–
2	Анализ предметной области и постановка задачи на разработку	–	–
3	Анализ предметной области	0,5 дня	1
4	Обзор аналогичных разработок	2 дня	3
5	Сравнение аналогичных разработок	1 день	4
6	Постановка задачи	1 день	5

Продолжение таблицы 2

№	Название задачи	Длительность	Предшественники
7	Анализ предметной области и постановка задачи на разработку завершена	–	6
8	Анализ требований на разработку программного продукта	–	7
9	Анализ функциональных требований	2 дня	7
10	Анализ нефункциональных требований	1 день	7
11	Анализ требований к данным	1 день	9, 10
12	Анализ архитектурных требований	1 день	11
13	Анализ требований к пользовательскому интерфейсу	2 дня	11
14	Анализ существующих технологий разработки программного обеспечения	2 дня	13
15	Выбор технологий и средств разработки программного обеспечения	3 дня	14
16	Анализ требований на разработку программного продукта завершен	–	15
17	Реализация программных средств	–	16
18	Разработка базы данных	2 дня	16
19	Разработка пользовательского интерфейса	21 день	18
20	Разработка серверного модуля программного средства	21 день	18
21	Разработка программного обеспечения	2 дня	20
22	Реализация программных средств завершена	–	19, 21
23	Оценка качества программного продукта	–	22
24	Выбор способа оценивания	0,5 дня	22
25	Функциональное тестирование	2 дня	24
26	Интеграционное тестирование	3 дня	24
27	Оценка метрик кода	1 день	24
28	Оценка качества программного продукта завершена	–	25, 26, 27
29	Составление и оформление документации	5 дней	28
30	Конец работы над проектом	–	29

Следующим этапом планирования является подбор и распределение необходимых ресурсов для выполнения каждой задачи проекта. В таблице 3 приведён полный список требуемых ресурсов с их детализацией.

Таблица 3 – Список ресурсов проекта

Название ресурса	Тип ресурса	Стандартная ставка	Сверхурочная ставка
Аналитик	Трудовой	250 Руб/ч	300 Руб/ч
Программист	Трудовой	300 Руб/ч	400 Руб/ч
Дизайнер	Трудовой	200 Руб/ч	250 Руб/ч

Продолжение таблицы 3

Название ресурса	Тип ресурса	Стандартная ставка	Сверхурочная ставка
Тестировщик	Трудовой	200 Руб/ч	250 Руб/ч
Интернет	Затраты	600 Руб/мес	–
Компьютер	Материальный	40 000 Руб	–

Завершающей стадией ресурсного планирования является назначение имеющихся ресурсов на задачи проекта (таблица 4).

Таблица 4 – Распределение ресурсов проекта по задачам

№	Название задачи	Ресурсы	Единицы
1	Начало проекта	–	–
2	Анализ предметной области и постановка задачи на разработку	–	–
3	Анализ предметной области	Аналитик Компьютер Интернет	100% 1 шт.
4	Обзор аналогичных разработок	Аналитик Компьютер Интернет	100% 1 шт.
5	Сравнение аналогичных разработок	Аналитик Компьютер Интернет	100% 1 шт.
6	Постановка задачи	Аналитик Компьютер Интернет	100% 1 шт.
7	Анализ предметной области и постановка задачи на разработку завершена	–	–
8	Анализ требований на разработку программного продукта	–	–
9	Анализ функциональных требований	Аналитик Программист Компьютер Интернет	50% 50% 2 шт.
10	Анализ нефункциональных требований	Аналитик Программист Компьютер Интернет	50% 50% 2 шт.
11	Анализ требований к данным	Аналитик Программист Компьютер Интернет	100% 100% 2 шт.
12	Анализ архитектурных требований	Аналитик Программист Компьютер Интернет	50% 50% 2 шт.

Продолжение таблицы 4

№	Название задачи	Ресурсы	Единицы
13	Анализ требований к пользовательскому интерфейсу	Аналитик Дизайнер Компьютер Интернет	50% 50% 2 шт.
14	Анализ существующих технологий разработки программного обеспечения	Программист Компьютер Интернет	100% 1 шт.
15	Выбор технологий и средств разработки программного обеспечения	Программист Компьютер Интернет	100% 1 шт.
16	Анализ требований на разработку программного продукта завершен	—	—
17	Реализация программных средств	—	—
18	Разработка базы данных	Программист Компьютер Интернет	100% 1 шт.
19	Разработка пользовательского интерфейса	Дизайнер Программист Компьютер Интернет	100% 50% 2 шт.
20	Разработка серверного модуля программного средства	Программист Компьютер Интернет	50% 1 шт.
21	Разработка программного обеспечения	Программист Компьютер Интернет	100% 1 шт.
22	Реализация программных средств завершена	—	—
23	Оценка качества программного продукта	—	—
24	Выбор способа оценивания	Тестировщик Компьютер Интернет	100% 1 шт.
25	Функциональное тестирование	Тестировщик Компьютер Интернет	50% 1 шт.
26	Интеграционное тестирование	Тестировщик Компьютер Интернет	50% 1 шт.
27	Оценка метрик кода	Тестировщик Компьютер Интернет	100% 1 шт.
28	Оценка качества программного продукта завершена	—	—

Окончание таблицы 4

№	Название задачи	Ресурсы	Единицы
29	Составление и оформление документации	Аналитик	100%
		Программист	100%
		Дизайнер	100%
		Тестировщик	100%
		Компьютер	4 шт.
		Интернет	
30	Конец работы над проектом	—	—

Для эффективного планирования и управления проектом был выбран Microsoft Project – специализированный программный инструмент для профессионального управления проектами [15].

Microsoft Project позволяет разрабатывать детальные планы работ оптимально распределять ресурсы между задачами и отслеживать прогресс выполнения. Есть возможность автоматического формирования диаграмм Ганта, которые наглядно отображают временные рамки задач в рамках проекта.

В Microsoft Project были загружены списки задач и назначенных ресурсов, на основании чего системой была построена диаграмма Ганта, изображенная на рисунке 8. В качестве даты начала работ было взято 8 апреля 2025 года.

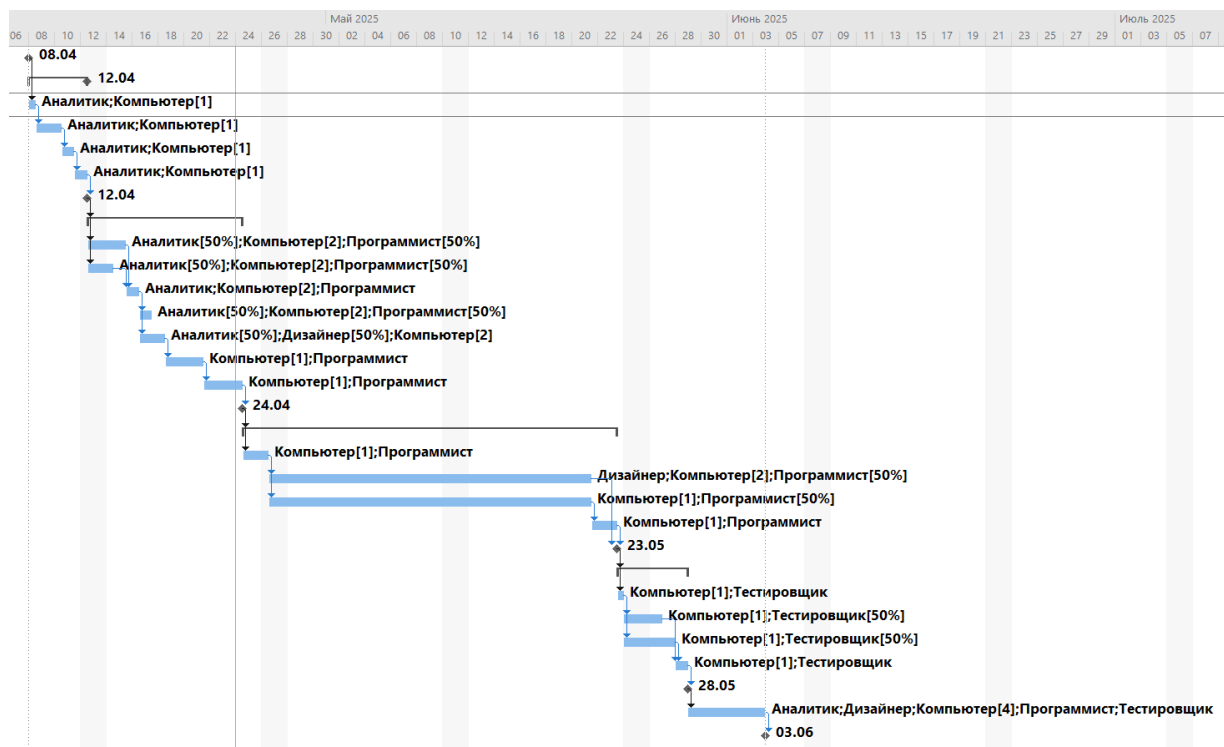


Рисунок 8 – Диаграмма Ганта

По полученной диаграмме можно увидеть, что дата окончания работы над проектом запланирована на 3 июня.

Важное преимущество Microsoft Project – возможность его интеграции с другими продуктами Microsoft Office, что упрощает процесс подготовки отчетов.

Программами с подобным функционалом являются:

- «ProjectLibre» [16];
- «Oracle Primavera P6» [17];
- «GanttPRO» [18].

Также программой были сгенерированы аналитические отчеты. Полный расчет затрат по проекту изображен на рисунках 9, 10. Информация об использовании ресурсов приведена на рисунке 11. Эти отчеты показывают распределение ресурсов между задачами и общую стоимость реализации проекта.

Название	Трудозатраты	Затраты	Названия ресурсов
Начало проекта	0 ч	0,00 ₽	
Анализ предметной области и постановка задачи на разработку	36 ч	9 000,00 ₽	
Анализ предметной области	4 ч	1 000,00 ₽	Аналитик
Обзор аналогичных разработок	16 ч	4 000,00 ₽	Аналитик
Сравнение аналогичных разработок	8 ч	2 000,00 ₽	Аналитик
Постановка задачи	8 ч	2 000,00 ₽	Аналитик
Анализ предметной области и постановка задачи на разработку завершена	0 ч	0,00 ₽	
Анализ требований на разработку программного продукта	100 ч	27 600,00 ₽	
Анализ функциональных требований	12 ч	3 200,00 ₽	Аналитик[50%];Программист[50%]
Анализ нефункциональных требований	8 ч	2 200,00 ₽	Аналитик[50%];Программист[50%]
Анализ требований к данным	16 ч	4 400,00 ₽	Аналитик;Программист
Анализ архитектурных требований	8 ч	2 200,00 ₽	Аналитик[50%];Программист[50%]
Анализ требований к пользовательскому интерфейсу	16 ч	3 600,00 ₽	Аналитик[50%];Дизайнер[50%]
Анализ существующих технологий разработки программного обеспечения	16 ч	4 800,00 ₽	Программист
Выбор технологий и средств разработки программного обеспечения	24 ч	7 200,00 ₽	Программист
Анализ требований на разработку программного продукта завершен	0 ч	0,00 ₽	

Рисунок 9 – Отчет об использовании ресурсов

Реализация программных средств	368 ч	93 600,00 Р
Разработка базы данных	16 ч	4 800,00 Р Программист
Разработка пользовательского интерфейса	252 ч	58 800,00 Р Дизайнер;Программист[50%]
Разработка серверного модуля программного средства	84 ч	25 200,00 Р Программист[50%]
Разработка программного обеспечения	16 ч	4 800,00 Р Программист
Реализация программных средств завершена	0 ч	0,00 Р
Оценка качества программного продукта	32 ч	6 400,00 Р
Выбор способа оценивания	4 ч	800,00 Р Тестировщик
Функциональное тестирование	8 ч	1 600,00 Р Тестировщик[50%]
Интеграционное тестирование	12 ч	2 400,00 Р Тестировщик[50%]
Оценка метрик кода	8 ч	1 600,00 Р Тестировщик
Оценка качества программного продукта завершена	0 ч	0,00 Р
Составление и оформление документации	160 ч	Аналитик;Дизайнер;Компьютер[4];П 79 200,00 Р рограммист;Тестировщик;Интернет[1 200,00 Р]
Конец работы над проектом	0 ч	0,00 Р
ИТОГО	696 ч	215 800,00 Р

Рисунок 10 – Продолжение отчета об использовании ресурсов

По данным отчета можно увидеть, что занятость всех трудовых ресурсов равняется 696 часам, а общая стоимость реализуемого проекта составляет 215800 рублей.

Данные	Ресурсы	Итог
Затраты	Затраты задачи	0
	Не назначен	0
	Аналитик	27000
	Программист	90000
	Дизайнер	43200
	Тестировщик	14400
	Интернет	1200
	Компьютер	40000
Трудозатр	Затраты задачи	0
	Не назначен	0
	Аналитик	108
	Программист	300
	Дизайнер	216
	Тестировщик	72
	Интернет	0
	Компьютер	0
Итог Затраты		215800
Итог Трудозатраты		696

Рисунок 11 – Отчет по сводке затрат и трудозатрат ресурсов

Данный отчет показывает общее время, затраченное каждым из ресурсов за период работы над проектом, и общую стоимость каждого ресурса.

2 Анализ требований на разработку web-портала поиска родственников и ведения генеалогического дерева семьи

2.1 Анализ функциональных требований

Основным актером разрабатываемого приложения является пользователь. У него должна быть возможность создания и редактирования родословных деревьев, добавлять в них людей, указывать информацию о них, а также устанавливать семейные связи (родители, дети, супруги). Пользователю должна быть предоставлена возможность регистрации, входа в систему и выхода из неё. Кроме того, пользователь должен иметь возможность просматривать дерево в графическом виде, осуществлять поиск людей, открывать профиль конкретного человека и оставлять комментарии.

Вторым актёром является администратор. Администратор системы выполняют вспомогательные функции, направленные на обеспечение стабильной и корректной работы приложения. Администратор проверяет поступившие жалобы и принимает решение: отклонить жалобу (если комментарий не нарушает правил) либо согласиться с ней и удалить соответствующий комментарий (если он содержит недопустимый контент). В случае технических сбоев или потери данных администратор должен иметь возможность восстановления базы данных из ранее созданной резервной копии.

Учитывая изложенные выше функциональные требования, были разработаны диаграммы вариантов использования для каждого из актеров, учитывая выявленные ранее информационные процессы.

Диаграмма вариантов использования (use case diagram) – это один из инструментов UML, предназначенный для описания функциональности системы. Она позволяет наглядно представить, какие функции доступны пользователям и каким образом они взаимодействуют с системой [19].

На рисунке 12 представлена диаграмма для подсистемы управления данными о персонах.

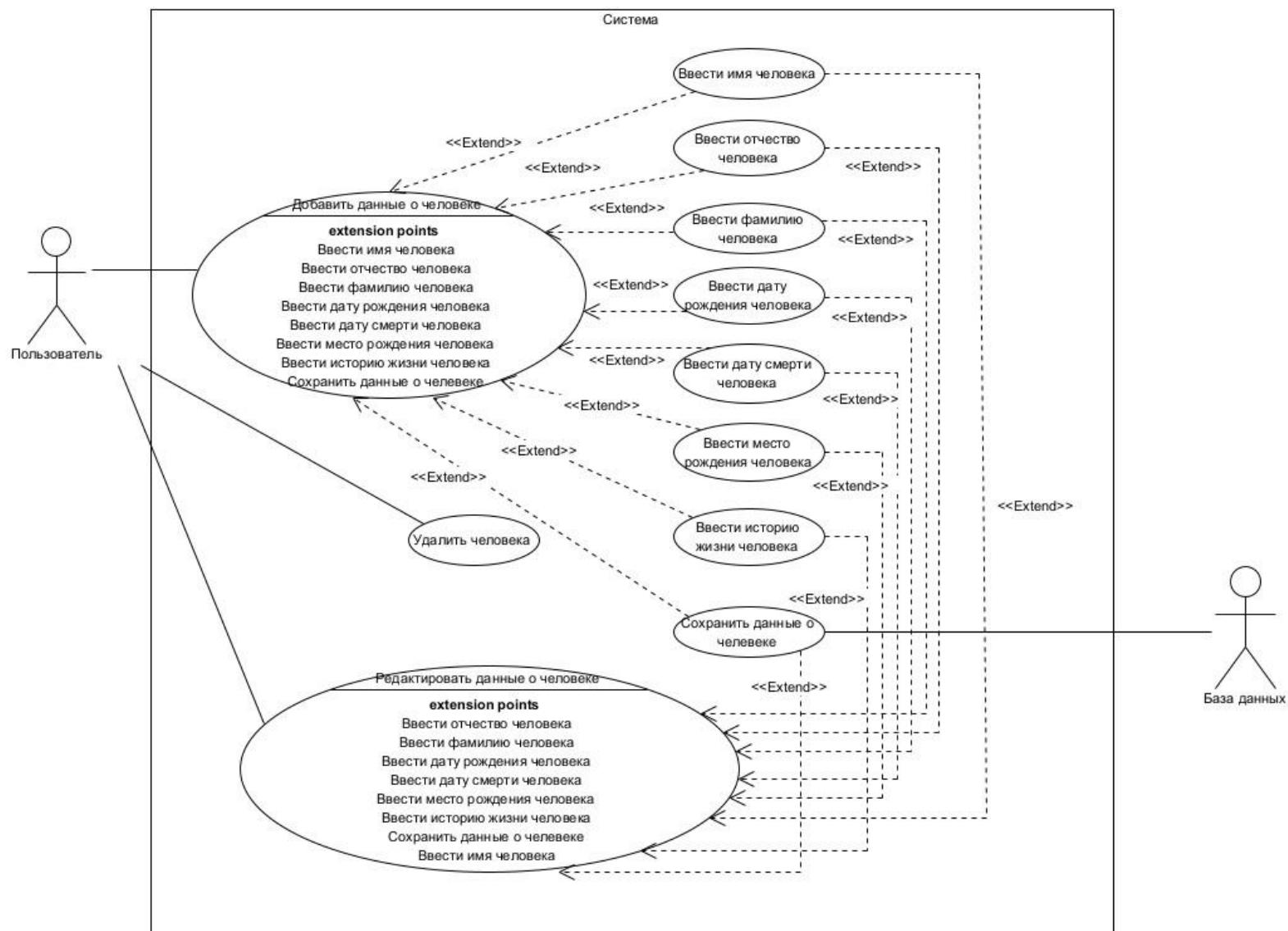


Рисунок 12 – Диаграмма вариантов использования подсистемы управления данными о персонах

На рисунке 13 представлена диаграмма для подсистемы управления родственными связями.



Рисунок 13 – Диаграмма вариантов использования подсистемы управления родственными связями

На рисунках 14 приведена диаграмма для подсистем поиска персон и оценки достоверности данных.

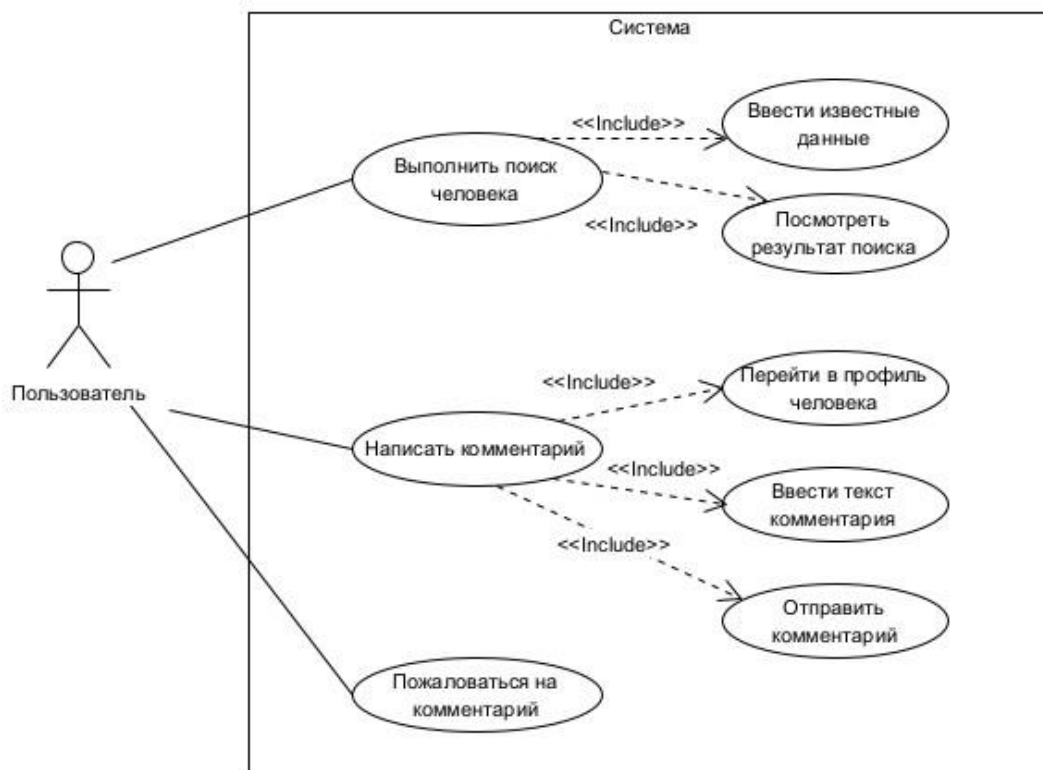


Рисунок 14 – Диаграмма вариантов использования подсистемы поиска персон и оценки достоверности данных

В приложении Г также приведены диаграммы для подсистем авторизации, управления деревьями и администрирования.

На рисунках 15 – 17 приведены спецификации некоторых наиболее важных вариантов использования.

Наименование: Добавить данные о человеке
ID: 1
Краткое описание: Пользователь вводит данные о человеке, система сохраняет их в базу данных
Действующие лица: Пользователь, система
Предусловие:
Пользовательский поток: 1. Пользователь вводит данные о человеке и отправляет их на сохранение. 2. Пользователь получает результат сохранения. Системный поток: 1. Система сохраняет переданные данные в базу данных. 2. Система возвращает результат добавления данных.
Постусловие: Данные о персоне добавлены

Рисунок 15 – Спецификация варианта использования «Добавить данные о человеке»

Наименование: Добавить родственное отношение
ID: 2
Краткое описание: Пользователь вводит данные о родственной связи, система сохраняет их в базу данных
Действующие лица: Пользователь, система
Предусловие: Введены данные о минимум двух людях
Пользовательский поток: 1. Пользователь вводит данные о родственной связи и отправляет их на сохранение. 2. Пользователь получает результат добавления. Системный поток: 1. Система проверяет возможность добавления родственного отношения. 2. Если проверка пройдена успешно, система сохраняет переданные данные в базу данных. 3. Система возвращает результат добавления данных.
Постусловие: Данные о родственной связи добавлены

Рисунок 16 – Спецификация варианта использования «Добавить родственное отношение»

Наименование: Выполнить поиск человека
ID: 3
Краткое описание: Пользователь вводит данные о человеке, система ищет по ним человека в базе данных
Действующие лица: Пользователь, система
Предусловие:
Пользовательский поток: 1. Пользователь вводит данные о человеке. 2. Пользователь получает результат поиска. Системный поток: 1. Система ищет переданные данные в базе данных. 2. Система возвращает результат поиска.
Постусловие: Поиск человека произведен

Рисунок 17 – Спецификация варианта использования «Выполнить поиск человека»

Помимо этого, для варианта «Выполнить поиск человека» была разработана диаграмма последовательности, изображенная на рисунке 18.

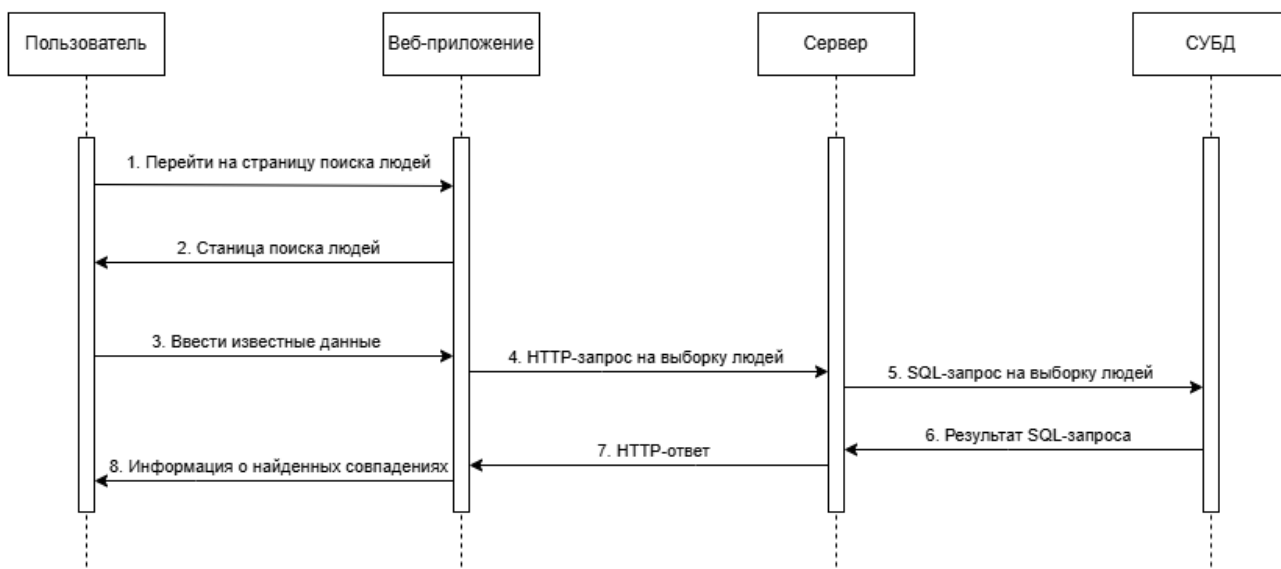


Рисунок 18 – Диаграмма последовательности варианта использования «Выполнить поиск человека»

Анализ функциональных требований позволил определить необходимые роли пользователей и основные действия, которые доступны им в системе. Это стало основой для проектирования интерфейса, архитектуры, базы данных и дальнейшей реализации приложения.

2.2 Анализ архитектурных требований

Система разрабатывается на основе сервис-ориентированной архитектуры. Она является распределённой и содержит клиентскую и серверную части, а также слой базы данных.

Клиентская часть представляет собой веб-интерфейс, обеспечивающий удобный доступ к функциональности приложения.

Серверная часть обеспечивает обработку запросов от клиента. В данной части реализована основная бизнес-логика приложения.

Помимо этого, сервер взаимодействует с базой данных, в которой хранятся данные обо всех сущностях системы.

Сервис-ориентированная архитектура строится на принципе автономности. Он предполагает разделение функциональности на независимые модули. Каждый модуль выполняет строго определенные задачи. Сервисы способны работать независимо друг от друга. Данный подход упрощает добавление новых функций или изменение уже существующих. Изменение происходит без влияния на другие компоненты системы. Кроме этого, данный подход улучшает масштабируемость приложения.

Взаимодействие между клиентом и сервером реализуется с помощью REST API.

Обмен данными между клиентской и серверной частями происходит в формате JSON.

Таким образом, выбранный подход обеспечивает простоту обслуживания и масштабируемости [20, 21]. Диаграмма обобщённой архитектуры разрабатываемого приложения изображена на рисунке 19.

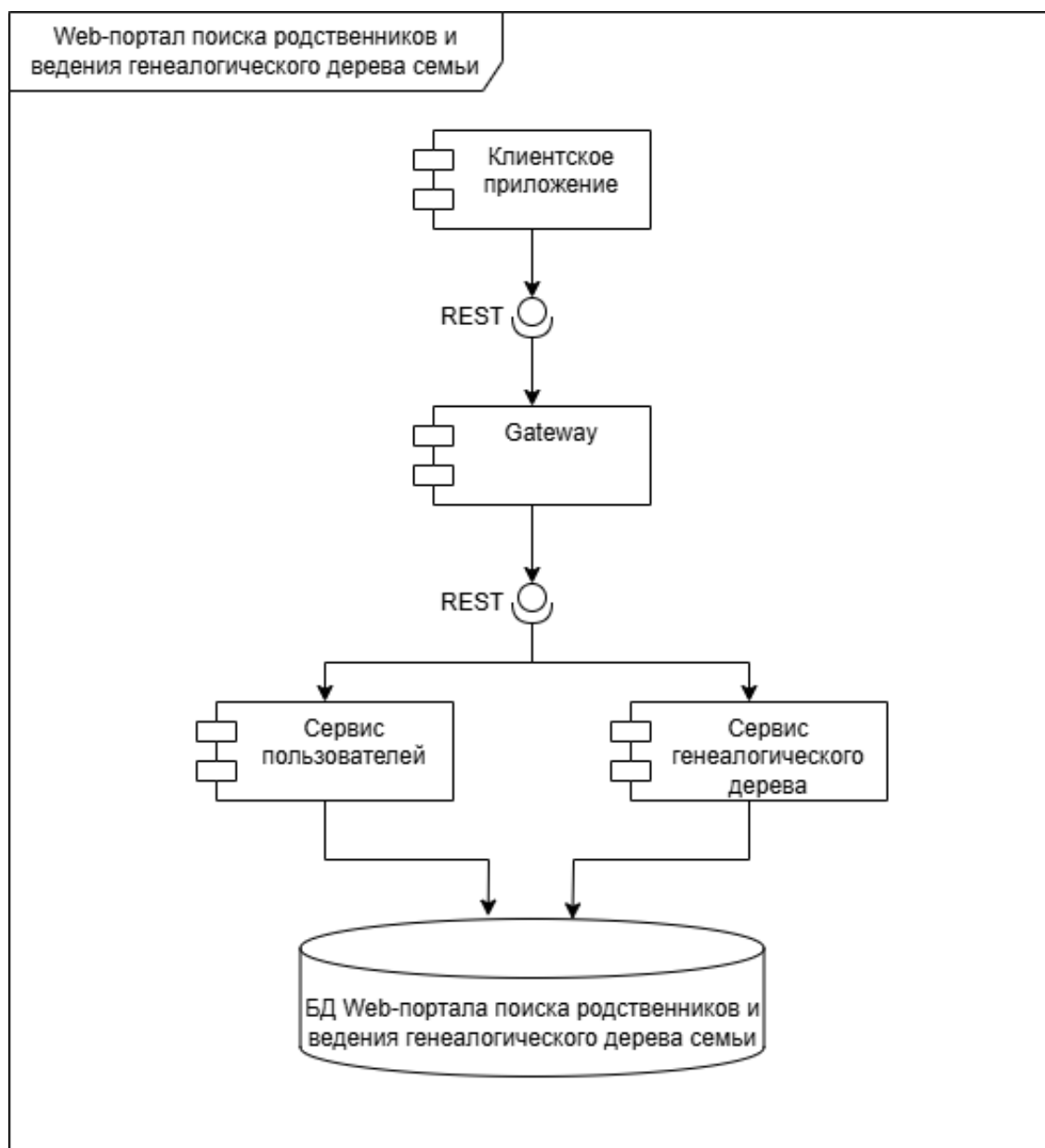


Рисунок 19 – Обобщенная архитектура web-портала

При создании приложения была выбрана архитектура Model-View-Controller (MVC). Она реализует принцип разделения ответственности, т. е. позволяет чётко разграничить области.

Model (Модель) отвечает за представление данных и бизнес-логику приложения. Она включает работу с базой данных и выполнение операций обработки данных.

View (Представление) отвечает за отображение данных пользователю. Оно получает данные от модели и формирует ответ.

Controller (Контроллер) является посредником между моделью и представлением. Он обрабатывает пользовательские запросы, обращается к модели за необходимыми данными и выбирает соответствующее представление для отображения результата [22].

Преимуществами использования MVC являются:

- облегчение сопровождения и развития системы за счёт разделения логики;
- повторное использование компонентов;
- упрощает модульное тестирование.

Таким образом, использование архитектуры MVC способствует созданию гибкой и масштабируемой системы.

2.3 Анализ нефункциональных требований

После формулирования функциональных требований были определены нефункциональные требования [23 – 25], которые включают в себя следующие положения:

- эксплуатационные свойства, в которые входят: доступность, производительность, масштабируемость и надёжность;
- структурные свойства, содержащие сопровождаемость, расширяемость и переносимость;
- сквозные свойства, включающие безопасность, удобство и простоту использования и поддерживаемость.

2.3.1 Эксплуатационные характеристики

Доступность – программное обеспечение должно обеспечивать круглосуточный режим работы (24 часа в сутки, 7 дней в неделю) с допустимым временем недоступности не более 0,1% в расчёте на календарный месяц. Для обеспечения требуемой доступности используется Gateway API, выполняющий функции централизованного управления маршрутизацией запросов.

Производительность – время отклика системы на пользовательский запрос не должно превышать 3 секунд при стандартной нагрузке. Gateway API должен обеспечивать маршрутизацию запросов к необходимым компонентам системы с минимальной задержкой. Обращение к базе данных MySQL должно осуществляться с использованием индексируемых и оптимизированных SQL-запросов.

Масштабируемость – программное обеспечение должно поддерживать горизонтальное масштабирование сервисных компонентов без изменения базовой архитектуры. То есть обработка увеличивающегося объема пользовательских данных должна обеспечиваться за счёт масштабирования отдельных сервисов. И также архитектура системы должна предусматривать возможность добавления новых функций путём подключения дополнительных сервисов через Gateway API.

Надёжность – программное средство должно обеспечивать устойчивую работу в течение длительного времени без сбоев. Для сохранения целостности данных при выполнении операций должна использоваться поддержка транзакций на уровне СУБД MySQL.

2.3.2 Структурные характеристики

Сопровождаемость – программное средство должно иметь модульную архитектуру с чётким разграничением ответственности между компонентами. Gateway API должен обеспечивать централизованное управление маршрутами и взаимодействием между сервисами, что упрощает сопровождение и развитие системы.

Расширяемость – архитектура должна обеспечивать возможность расширения путём добавления новых сервисов без необходимости внесения изменений в существующий код. Интеграция новых компонентов должна осуществляться через Gateway API.

Переносимость – программное средство должно быть независимым от конкретной программно-аппаратной платформы и допускать развёртывание на любой инфраструктуре, поддерживающей Java, MySQL и Node JS.

2.3.3 Сквозные характеристики

Безопасность – должна быть обеспечена централизованная аутентификация и авторизация всех входящих запросов. Для управления пользовательскими сессиями должны применяться токены доступа формата JWT (JSON Web Token), соответствующие современным стандартам безопасности.

Удобство и простота использования – пользовательский интерфейс должен быть реализован с использованием библиотеки React и обеспечивать интуитивно понятное взаимодействие. Gateway API должен скрывать внутреннюю структуру сервисов от конечного пользователя.

Поддерживаемость – необходимо предоставить централизованный механизм для мониторинга и диагностики запросов, что ускоряет устранение неполадок. Данная возможность реализуется посредством Gateway API.

Все перечисленные требования направлены на обеспечение стабильной и удобной работы системы.

2.4 Анализ требований к данным

2.4.1 Определение концептуальной модели базы данных

На основе анализа предметной области и определения функциональных требований была построена концептуальная модель базы данных.

Целью определения концептуальной модели является первоначальное структурирование информации в предметной области без учёта деталей возможной реализации базы данных [26]. Концептуальная модель необходима для исключения ошибок в дальнейшей разработке базы данных.

Диаграмма концептуальной модели изображена на рисунке 20.

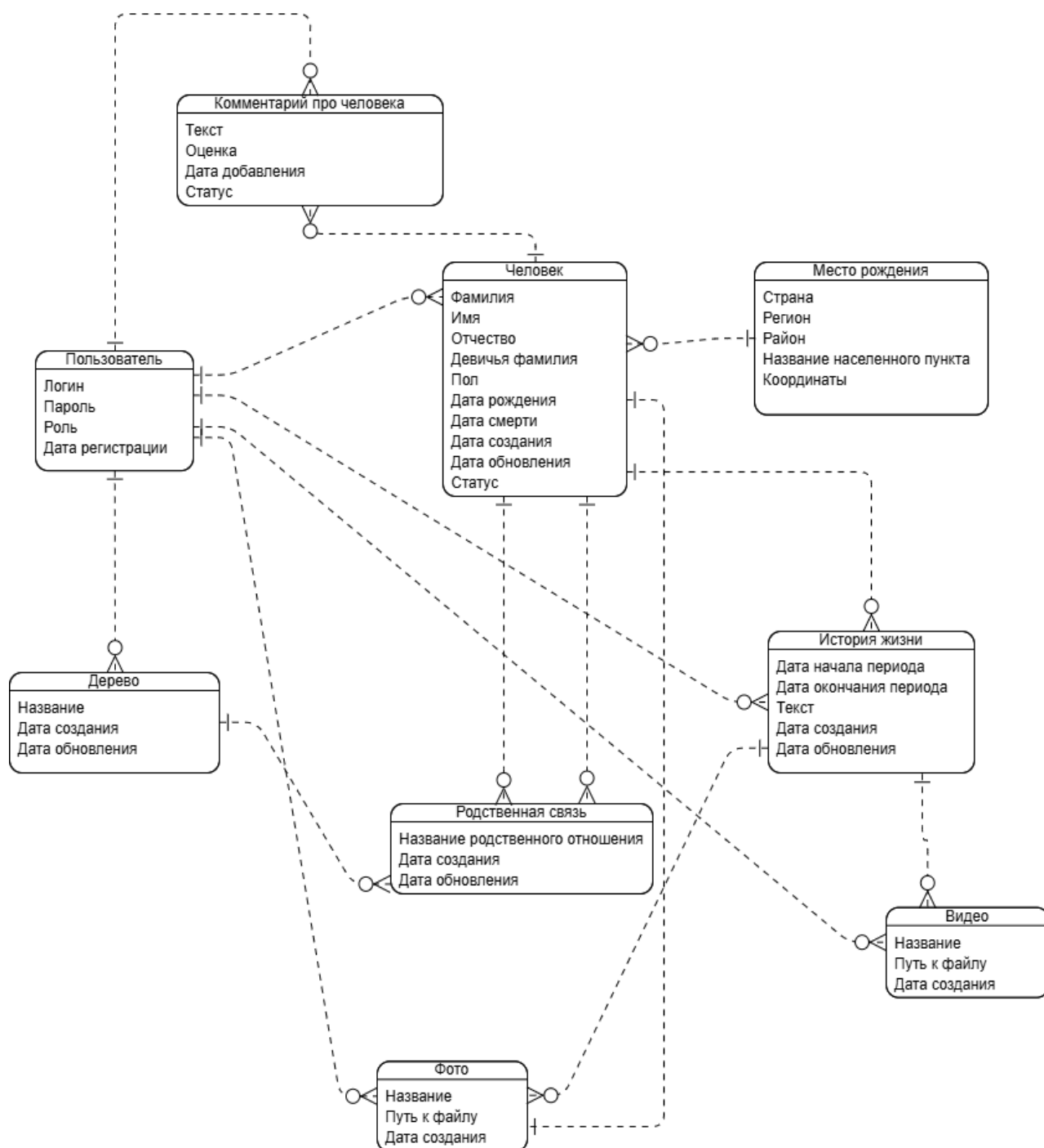


Рисунок 20 – Концептуальная модель базы данных web-портала поиска родственников и ведения генеалогического дерева семьи

Для разработанной концептуальной модели базы данных было создано табличное описание сущностей, представленное в таблице 5. Табличное описание сущностей является связующим звеном между концептуальной моделью и физической реализацией базы данных.

Таблица 5 – Табличное описание сущностей для концептуальной модели базы данных

Сущность	Атрибуты	Характеристики
Пользователь	ID пользователя, Логин, Пароль, Почта, Роль, Дата регистрации	Идентификатор, личные данные, связь с деревом, человеком, родственной связью, редактором, историей жизни, фото, видео, комментарием про человека, комментарием про родственную связь
Человек	ID человека, Фамилия, Имя, Отчество, Девичья фамилия, Пол, Дата рождения, Дата смерти, Статус	Идентификатор, личные данные, связь с пользователем, родственной связью, местом рождения, историей жизни, комментарием про человека, статус(жив/мертв/неизвестно)
Родственная связь	ID связи, Название родственного отношения, Дата создания, Дата обновления	Идентификатор, информация о родственном отношении, связь между двумя людьми, пользователем, деревом, комментарием про родственную связь
Дерево	ID дерева, Название, Дата создания, Дата обновления	Идентификатор дерева, информация о дереве, связь с пользователем, редактором, родственной связью
История жизни	ID истории жизни, Дата начала периода, Дата окончания периода, Текст, Дата добавления, Дата обновления	Идентификатор, событие в жизни человека с указанием деталей, связь с пользователем, человеком, фото, видео
Место рождения	ID места, Страна, Регион, Район, Название населенного пункта, Почтовый индекс, Координаты	Идентификатор, информация о населенном пункте, связь с человеком
Фото	ID фото, Название, Путь к файлу, Дата создания	Идентификатор, информация о фото, связь с пользователем, историей жизни
Видео	ID видео, Название, Путь к файлу, Дата создания	Идентификатор, информация о видео, связь с пользователем, историей жизни
Комментарий про человека	ID комментария про человека, Текст, Оценка, Статус, Дата добавления	Идентификатор, заметки о человеке, статус, связь с пользователем, человеком

Для разработанной концептуальной модели базы данных также было составлено табличное описание связей между сущностями, представленное в таблице 6.

Таблица 6 – Спецификации связей между сущностями для концептуальной модели базы данных

Связь сущностей	Описание	Тип связи
Пользователь – Комментарий про человека	Пользователь является автором комментария	Один ко многим
Пользователь – Дерево	Пользователь является владельцем дерева	Один ко многим
Пользователь – Человек	Пользователь является создателем человека	Один ко многим
Пользователь – Родственная связь	Пользователь является создателем родственной связи	Один ко многим
Пользователь – История жизни	Пользователь является создателем истории жизни	Один ко многим
Пользователь – Фото	Пользователь является владельцем фото	Один ко многим
Пользователь – Видео	Пользователь является владельцем видео	Один ко многим
Комментарий про человека – Человек	Комментарий пишется для конкретного человека	Многие к одному
Человек – Фото	Профилю человека можно добавлять аватар	Один к одному
Дерево – Родственная связь	Родственная связь строится в конкретном дереве	Один ко многим
Человек – Родственная связь	Родственная связь соединяет двух людей родственным отношением	Один ко многим
Человек – Место рождения	Человек имеет своё место рождения	Многие к одному
Человек – История жизни	У человека есть своя история жизни	Один ко многим
История жизни – Фото	К истории жизни можно прикреплять фото	Один ко многим
История жизни – Видео	К истории жизни можно прикреплять видео	Один ко многим

Полученные результаты являются исходными данными для дальнейшего анализа требований к данным.

2.4.2 Определение логической модели базы данных

На основе полученной концептуальной модели была построена логическая модель базы данных. Логическая модель данных – это расширение концептуальной модели данных. Она включает в себя все сущности, атрибуты, ключи и взаимосвязи.

Диаграмма представлена на рисунке 21. На данной модели были определены первичные и внешние ключи для каждой из сущностей.

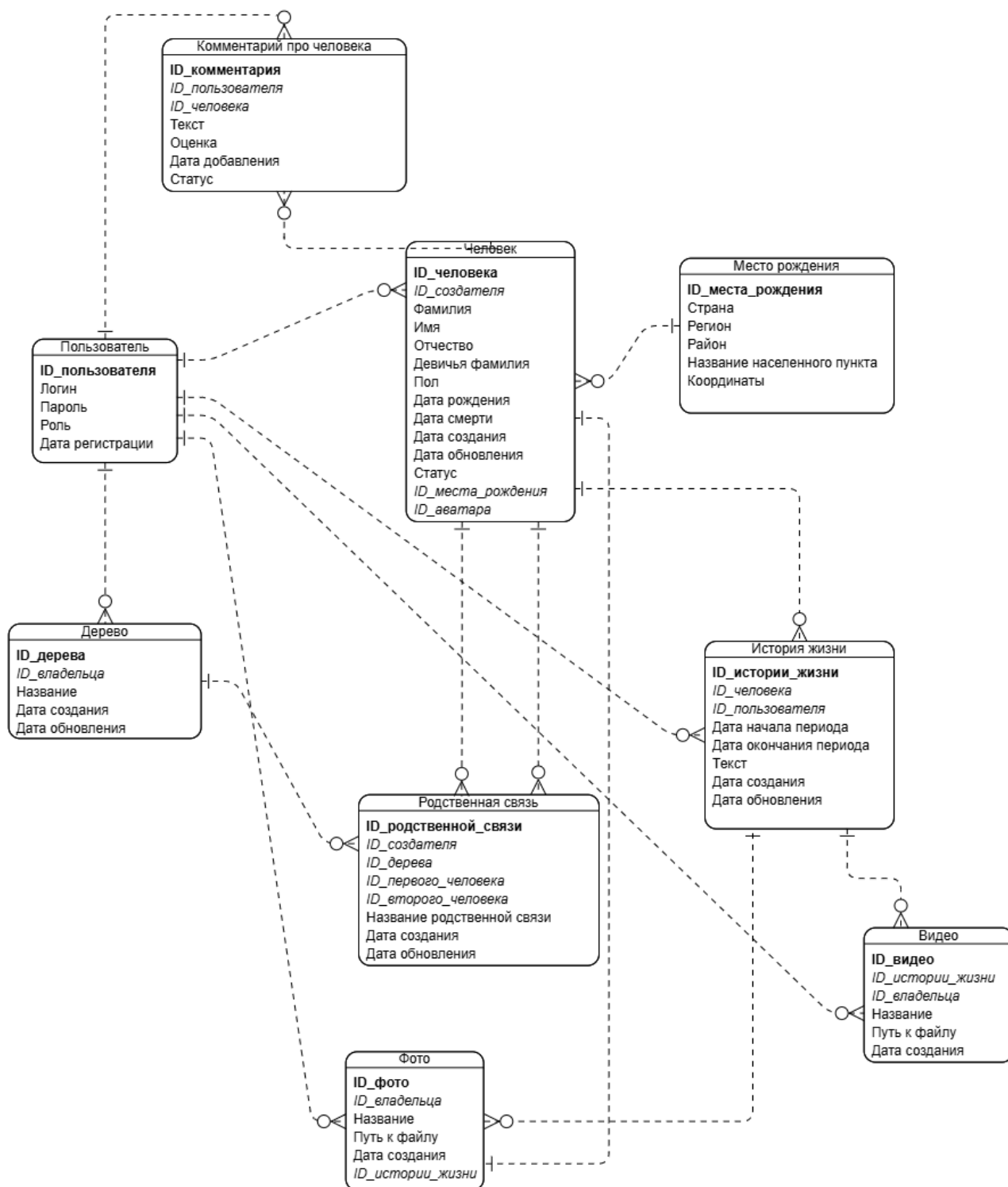


Рисунок 21 – Логическая модель базы данных web-портала поиска
 родственников и ведения генеалогического дерева семьи

На основе логической модели базы данных было составлено описание сущностей, представленное в таблице 7.

Таблица 7 – Спецификация сущностей логической модели базы данных

№ п/п	Наименование	Описание	Категория
1	Пользователь	Содержит сведения о пользователях системы	Стержневая
2	Дерево	Содержит сведения о семейных деревьях	Стержневая
3	Человек	Содержит сведения о людях, связанных с деревом	Стержневая
4	Родственная связь	Содержит сведения о связях между людьми	Иерархическая
5	Место рождения	Содержит сведения о местах рождения людей	Справочная
6	Комментарий о человеке	Содержит сведения о комментариях к человеку	Иерархическая
7	История жизни	Содержит сведения о событиях из жизни человека	Стержневая
8	Фото	Содержит сведения о фотографиях, связанных с жизнью	Иерархическая
9	Видео	Содержит сведения о видео, связанных с жизнью	Иерархическая

Также было составлено табличное описание связей между сущностями, представленное в таблице 8. В данной таблице указана мощность для каждой связи, её обязательность и ограничения.

Таблица 8 – Табличное описание связей между сущностями логической модели базы данных

№	Тип сущности 1	Тип сущности 2	Семантика связи	Мощность	Обязательность	Ограничения
1	Пользователь	Дерево	Один пользователь может владеть несколькими деревьями	1:M	Да	—
2	Пользователь	Комментарий про человека	Один пользователь может писать множество комментариев	1:M	Да	—
3	Пользователь	Человек	Один пользователь может создавать несколько людей	1:M	Да	—

Продолжение таблицы 8

№	Тип сущности 1	Тип сущности 2	Семантика связи	Мощность	Обязательность	Ограничения
4	Пользователь	Родственная связь	Один пользователь может создавать несколько родственных связей	1:M	Да	—
5	Пользователь	История жизни	Один пользователь может создавать несколько историй жизни	1:M	Да	—
6	Пользователь	Фото	Один человек может добавлять несколько фото	1:M	Да	—
7	Пользователь	Видео	Один человек может добавлять несколько видео	1:M	Да	—
8	Человек	Родственная связь	Один человек может участвовать в нескольких родственных связях	1:M	Да	Невозможно создание более одной связи между одними и теми же людьми или одним и тем же человеком
9	Родственная связь	Человек	Родственная связь включает двух людей	1:2	Да	Указаны оба человека
10	Человек	Место рождения	У человека есть одно место рождения	1:1	Да	—
11	Человек	История жизни	У человека может быть несколько событий из жизни	1:M	Да	—
12	Человек	Фото	У человека может быть только один аватар	1:1	Да	—

Окончание таблицы 8

№	Тип сущности 1	Тип сущности 2	Семантика связи	Мощность	Обязательность	Ограничения
13	Дерево	Родственная связь	Одно дерево может содержать много родственных связей	1:M	Да	—
14	История жизни	Фото	Одно событие может иметь несколько фотографий	1:M	Да	—
15	История жизни	Видео	Одно событие может иметь несколько видео	1:M	Да	—
16	Человек	Комментарий о человеке	У человека может быть несколько комментариев	1:M	Да	—

Полученные результаты являются исходными данными для дальнейшего моделирования и реализации базы данных приложения.

2.5 Анализ требований к пользовательскому интерфейсу

Пользовательский интерфейс является ключевым компонентом информационной системы. Именно он обеспечивает взаимодействие пользователя с функциональностью приложения. От удобства и понятности интерфейса напрямую зависят удовлетворённость пользователя, его эффективность работы с системой. С целью реализации удобного интерфейса применяется архитектура одностраничного приложения (Single Page Application, SPA).

Использование SPA необходимо для того, чтобы интерфейс работал быстро и динамично. Данная технология помогает минимизировать время отклика, а также избежать полной перезагрузки страниц. В результате её использования взаимодействие с интерфейсом становится плавным, а навигация по разделам происходит без задержек.

Вся бизнес-логика реализуется на серверной части приложения, с которой клиентская часть взаимодействует через REST API. Это позволяет четко разделять задачи и масштабировать интерфейс отдельно от серверной части [27].

Необходимо разработать следующие компоненты интерфейса:

- панель навигации по проекту;
- окно регистрации;
- окно входа;
- страница поиска родственников;
- страница просмотра профиля человека;
- окно просмотра комментариев;
- страница просмотра истории жизни человека;
- страница просмотра всех деревьев;
- страница просмотра дерева;
- окно добавления нового родственника в дерево;
- окно редактирования данных о человеке;
- окно создания/редактирования события в жизни человека;
- окно создания нового дерева;
- окно уведомления;
- окно для просмотра фото и видео.

Интерфейс должен обладать свойством отзывчивости, что подразумевает мгновенную реакцию системы на манипуляции пользователя. При каждом взаимодействии, например, нажатии кнопки, пользователь должен получать визуальные сигналы, подтверждающие успешность или неудачу операции. Эти сигналы могут быть представлены в виде сообщений об успешном выполнении действия, индикаторов загрузки, всплывающих уведомлений или индикаторов ошибок. Наличие таких сигналов способствует повышению ясности и предсказуемости функционирования системы, а также предотвращает повторение действий из-за неопределённости результата.

2.6 Обоснование выбора технологий и средств разработки

В качестве системы управления базами данных (СУБД) выбрана MySQL. Её преимуществом является поддержка транзакций и внешних ключей. Несмотря на то, что PostgreSQL обладает более широкими возможностями в обработке сложных запросов, MySQL превосходит её в скорости выполнения простых операций чтения.

В качестве языка программирования для реализации серверной части выбран Java и фреймворк Spring Boot. Фреймворк Spring Data JPA обеспечивает легкое создание и удобное взаимодействие с базой данных. Для реализации механизма аутентификации и авторизации используется модуль Spring Security.

Для реализации клиентской части приложения был выбран язык JavaScript и фреймворк React. Он обеспечивает высокую производительность за счёт виртуального DOM. Также позволяет реализовать одностраничное приложение (SPA). Компонентный подход в React обеспечивает повторное использование кода, улучшает структуру проекта и упрощает его поддержку.

Node.js используется для сборки клиентской части проекта и её запуска на сервере.

npm (Node Package Manager) – это менеджер пакетов для Node.js. Он используется для установки, обновления и управления зависимостями проекта.

В качестве инструмента для выполнения HTTP-запросов на клиентской стороне приложения используется библиотека Axios. Она обеспечивает удобный способ взаимодействия с REST API. Axios автоматически сериализует и десериализует данные, упрощая обмен информацией между клиентом и сервером.

Для отображения всплывающих в клиентской части приложения используется библиотека react-toastify. Это необходимо для реализации отзывчивого пользовательского интерфейса, т. е. предоставления мгновенной обратной связи пользователю при выполнении действий.

Для визуализации родословных деревьев выбран инструмент React Flow. Библиотека позволяет реализовывать сложные графовые структуры, поддерживает перетаскивания узлов и изменение масштабирования.

Для работы с формами выбран React Hook Form – библиотека для управления состоянием и валидацией форм в React-приложениях. Она минимизирует количество повторных рендеров компонентов, поддерживает валидацию, включая асинхронные проверки.

Для управления состоянием выбрана библиотека MobX. Благодаря механизму отслеживания зависимостей данная библиотека обновляет только те компоненты, которые зависят от изменившихся данных. Это существенно снижает нагрузку на рендеринг.

Для работы с токенами авторизации в проекте используется библиотека jwt-decode. Она позволяет на основе содержимого токена отобразить нужный пользовательский интерфейс.

Tailwind CSS выбран для стилизации интерфейса. Он позволяет быстро создавать адаптивные и современные пользовательские интерфейсы.

React Yandex Maps используется для интеграции интерактивных карт Яндекса в интерфейс приложения. Это позволит отображать географические данные, например, место рождения человека.

Разработка велась в среде Windows 10. Клиентская часть разрабатывалась в WebStorm, серверная – в IntelliJ IDEA. Для отладки и тестирования использовался браузер Google Chrome. Разработка велась на операционной системе Windows 10.

3 Разработка программного обеспечения web-портала поиска родственников и ведения генеалогического дерева семьёй

3.1 Разработка физической модели базы данных

На основе разработанной логической модели базы данных с учётом специфики системы управления базами данных MySQL Server была автоматически сформирована физическая структура базы данных с использованием фреймворка Spring Data JPA. Для визуализации данной структуры программой MySQL Workbench была сгенерирована ER-диаграмма (рисунок 22).

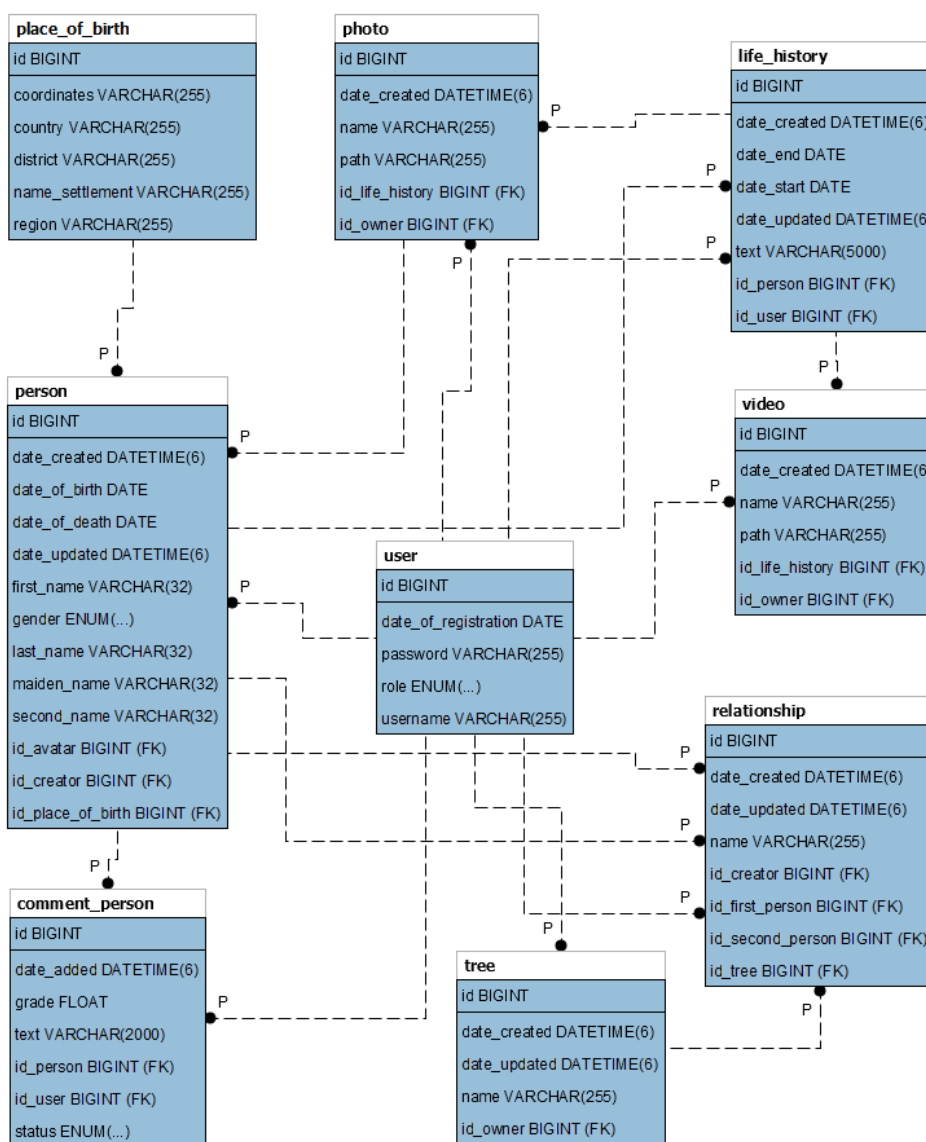


Рисунок 22 – Сгенерированная диаграмма базы данных

Для разработанной физической модели базы данных было составлено табличное описание атрибутов сущностей, приведённое в приложении Д.

В таблице 9 приведено описание атрибутов для сущности «Человек».

Таблица 9 – Атрибуты сущности «Человек»

№ п/п	Наименование	Описание	Тип данных	Размер	Область допустимых значений	Возможность значения NULL	Пример
1	ID	Уникальный идентификатор человека	Числовой	10	Уникальные значения	нет	1
2	Last_name	Фамилия человека	Строковый	32	A-я, A-z	да	Иванов
3	First_name	Имя человека	Строковый	32	A-я, A-z	да	Иван
4	Second_name	Отчество человека	Строковый	32	A-я, A-z	да	Иванович
5	Maiden_name	Девичья фамилия (если применимо)	Строковый	32	A-я, A-z	да	Петрова
6	Gender	Пол человека	Строковый	10	MALE, FEMALE	нет	MALE
7	Date_of_birth	Дата рождения	Дата		Временной диапазон	да	2000 – 01 – 01
8	Date_of_death	Дата смерти	Дата		Временной диапазон	Да	2070 – 12 – 31
9	ID_creator	Уникальный идентификатор создателя	Числовой		Уникальные значения	нет	1
10	ID_place_of_birth	Уникальный идентификатор места рождения	Числовой		Уникальные значения	нет	1
11	ID_avatar	Уникальный идентификатор аватара	Числовой		Уникальные значения	нет	1

В таблице 10 приведено описание атрибутов для сущности «Родственное отношение».

Таблица 10 – Атрибуты сущности «Родственное отношение»

№ п/п	Наименование	Описание	Тип данных	Размер	Область допустимых значений	Возможность значения NULL	Пример
1	ID	Уникальный идентификатор связи	Числовой	10	Уникальные значения	нет	1

Продолжение таблицы 10

№ п/п	Наименование	Описание	Тип данных	Размер	Область допустимых значений	Возможность значения NULL	Пример
2	Name	Тип родственного отношения	Строковый	255	А-я, А-з	нет	отец
3	Date_created	Дата создания родственной связи	Дата/время	–	Временной диапазон	нет	2024 – 12 – 01
4	Date_updated	Дата последнего изменения связи	Дата/время	–	Временной диапазон	да	2024 – 12 – 05
5	ID_creator	Уникальный идентификатор создателя	Числовой	10	Уникальные значения	нет	1
6	ID_tree	Уникальный идентификатор дерева	Числовой	10	Уникальные значения	нет	1
7	ID_first_person	Уникальный идентификатор первого человека	Числовой	10	Уникальные значения	нет	1
8	ID_second_person	Уникальный идентификатор второго человека	Числовой	10	Уникальные значения	нет	1

На данном этапе разработана физическая структура базы данных, которая отражает основные сущности предметной области и их взаимосвязи. Диаграмма базы данных и табличное описание атрибутов детально описывают состав и характеристики каждой сущности, что необходимо для корректной реализации логики взаимодействия компонентов системы.

3.2 Разработка пользовательского интерфейса

Интерфейс пользователя приложения для ведения генеалогического дерева состоит из нескольких компонентов, каждый из которых предназначен для выполнения определенных задач:

- форма входа;
- форма регистрации;
- страница навигации;
- меню;

- страница таблицы деревьев;
- окно создания дерева;
- страница поиска родственников;
- таблица родственников;
- окно уведомления;
- страница комментариев;
- страница полезных ссылок;
- страница дерева;
- окно создания человека;
- окно добавления места жительства;
- страница профиля человека;
- страница истории жизни;
- окно добавления истории жизни;
- страница дерева семьи;
- окно просмотра данных человека в дереве;
- окно просмотра ближайших родственников человека;
- окно добавления комментария;
- окно просмотра фото и видео.

Интерфейсы всех перечисленных компонентов представлены в приложении Е. Для некоторых из них ниже приведено описание элементов.

На рисунке 23 представлен интерфейс формы авторизации. При старте приложения открывается именно эта страница.

Поля, которые пользователь должен ввести, являются обязательными, а при регистрации логин должен быть уникальным.

При нажатии на кнопку «Войти», пользователь переходит к окну навигации. При нажатии на кнопку «Регистрация», пользователь переходит к окну регистрации.

Если у пользователя ещё нет аккаунта, то он может его зарегистрировать на форме регистрации.

Вход

Логин

Пароль

ВОЙТИ

[Регистрация](#)

Рисунок 23 – Форма для авторизации пользователя

На форме цифрами отмечены следующие обозначения:

- 1 – поле для ввода логина;
- 2 – поле для ввода пароля;
- 3 – кнопка просмотра пароля;
- 4 – кнопка для входа в аккаунт;
- 5 – кнопка для регистрации пользователя.

На рисунке 24 изображена страница поиска людей.

Поиск людей

1 — [Назад]

2 — Фамилия:

6 — Имя:

3 — Отчество:

7 — Девичья фамилия:

4 — Дата рождения:

8 — Дата смерти:

5 — Пол:

9 —

#	ФИО
1	Иванов Алексей
2	Иванова Мария

11 — [Правка]

Рисунок 24 – Страница поиска людей

На рисунке цифрами отмечены следующие элементы формы:

- 1 – кнопка открытия/скрытия фильтра;
- 2 – поле ввода фамилии;
- 3 – поле ввода отчества;
- 4 – поле ввода даты рождения;
- 5 – поле выбора пола;
- 6 – поле ввода имени;
- 7 – поле ввода девичьей фамилии;
- 8 – поле ввода даты смерти
- 9 – кнопка поиска;
- 10 – кнопка для сброса фильтрации;
- 11 – кнопка перехода в профиль человека.

На этой странице пользователь может посмотреть всех людей и произвести поиск человека по конкретным данным, а также перейти в профиль нужного человека.

На рисунке 25 изображено окно дерева.



Рисунок 25 – Окно дерева

На этой странице пользователь может посмотреть дерево иерархии, а также нажать на любой узел для просмотра информации о человеке.

На рисунке 26 приведена страница истории жизни человека.



Рисунок 26 – Страница истории жизни

На рисунке цифрами отмечены следующие элементы формы:

- 1 – кнопка сортировки по дате события;
- 2 – кнопка для добавления нового события;
- 3 – кнопка для редактирования события;
- 4 – кнопка для удаления события.

В ходе разработки пользовательского интерфейса уделено внимание удобству взаимодействия, визуальной адаптивности для устройств с разной шириной экрана. Разработанный интерфейс обеспечивает интуитивную навигацию и быстрое выполнение действий.

3.3 Разработка программного обеспечения

3.3.1 Разработка программного обеспечения серверной части

Разработка серверной части приложения производилась в среде программирования IntelliJ IDEA на языке Java с использованием фреймворка

Spring. Для взаимодействия с базой данных использовалась технология Spring Data JPA (Java Persistence API). Доступ к сервисам реализован через API шлюз. Основной функционал доступен после авторизации пользователя и зависит от его роли.

На рисунке 27 представлена диаграмма классов слоя сервисов и контроллеров для сервиса пользователей.

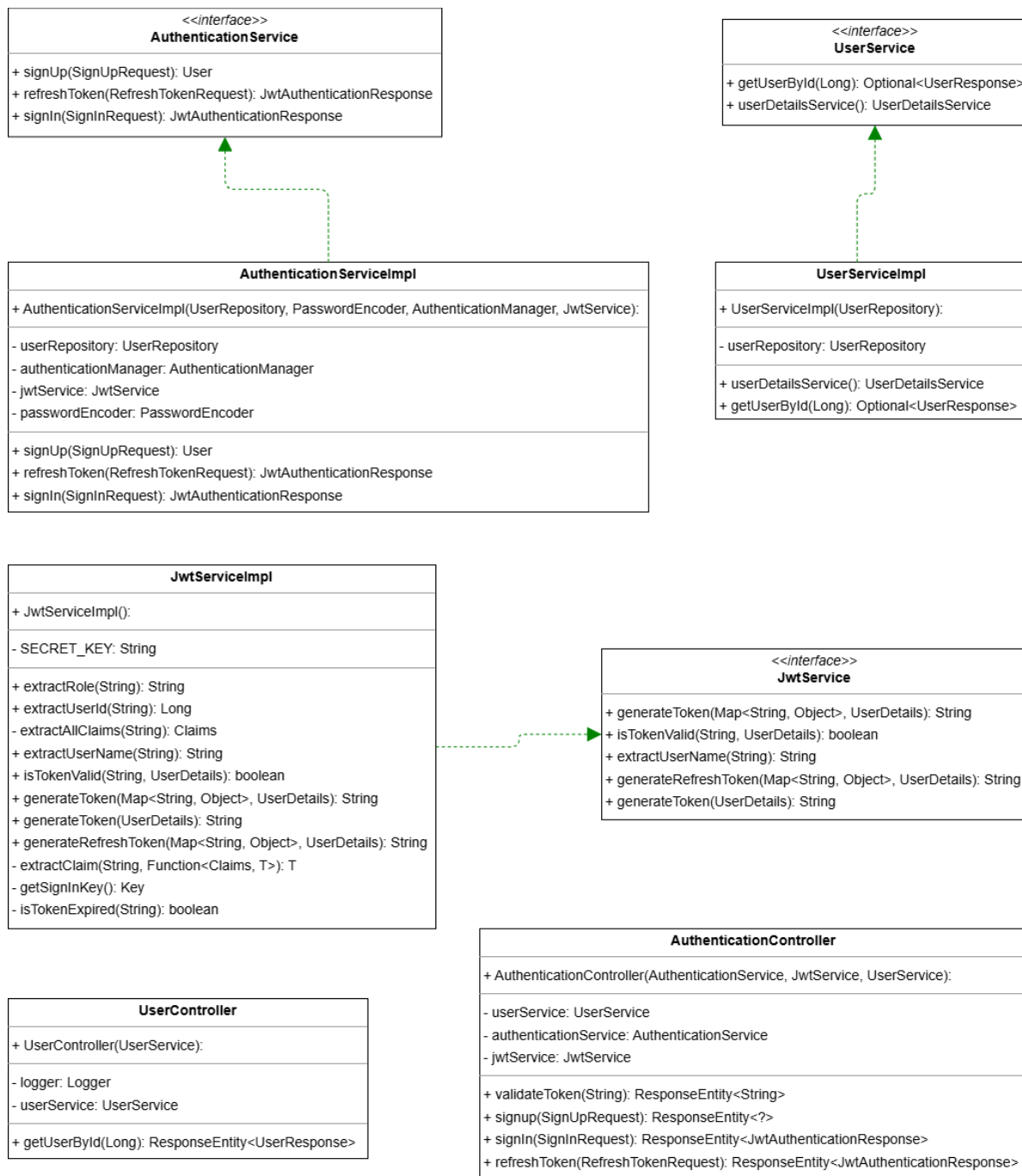


Рисунок 27 – Диаграмма классов слоя сервисов и контроллеров для сервиса пользователей

Данный сервис отвечает за предоставление данных о пользователях, а также реализует авторизацию и аутентификацию.

В таблице 11 представлено описание основных API методов сервиса пользователей.

Таблица 11 – Основные методы API авторизации

URL запроса	Http метод	Описание
api/auth/signup	POST	Регистрация пользователя
api/auth/signin	POST	Авторизация пользователя
api/auth/refresh	POST	Обновление токена
api/auth/validate	POST	Проверка токена на валидность

Сервис генеалогического дерева отвечает за предоставление данных, необходимых для построения структуры дерева, а также за предоставление всей информации, вносимой пользователем в родословные.

В таблице 12 представлено описание основных API методов сервиса генеалогического дерева.

Таблица 12 – Основные методы API генеалогического дерева

URL запроса	Http метод	Описание
/people/	GET	Получение всех людей
/people/with-parents/{treeId}	POST	Создание человека и его родителей
/api/tree/{treeId}/structure	GET	Получение дерева в виде структуры для отрисовки в клиентском приложении
/api/tree/search	GET	Получение найденных людей по переданным параметрам
/api/tree/byUserId	GET	Получение всех деревьев пользователя
/api/life	POST	Создание истории жизни человека с сохранением фото и видео
/api/comments/person/{personId}	GET	Получение всех комментариев для определенного человека

На рисунке 28 изображена диаграмма классов слоя контроллеров для сервиса генеалогического дерева.

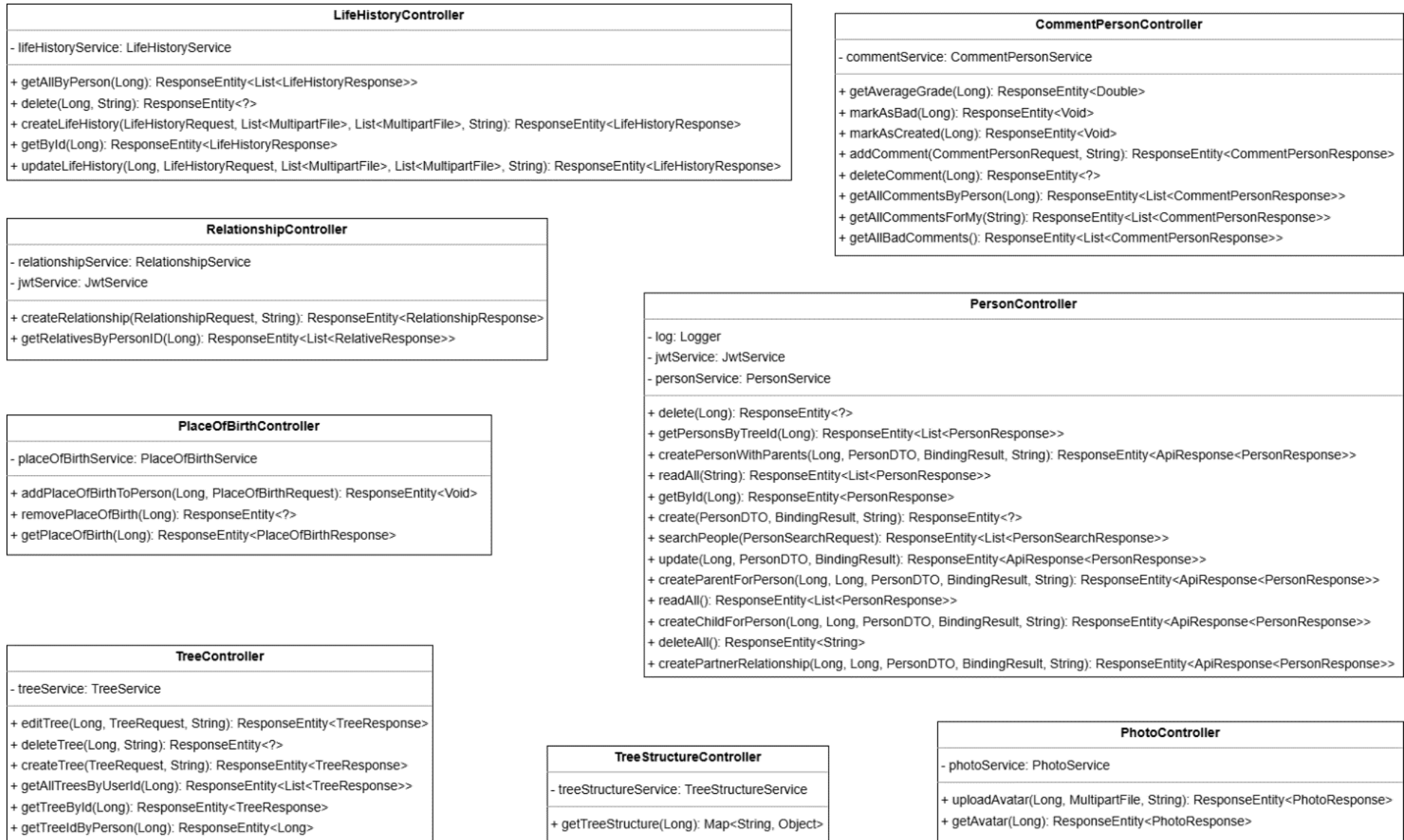


Рисунок 28 – Диаграмма классов слоя контроллеров для сервиса генеалогического дерева

На рисунке 29 изображена диаграмма классов для сервиса API шлюза, который отвечает за маршрутизацию запросов от клиентской части web-портала.

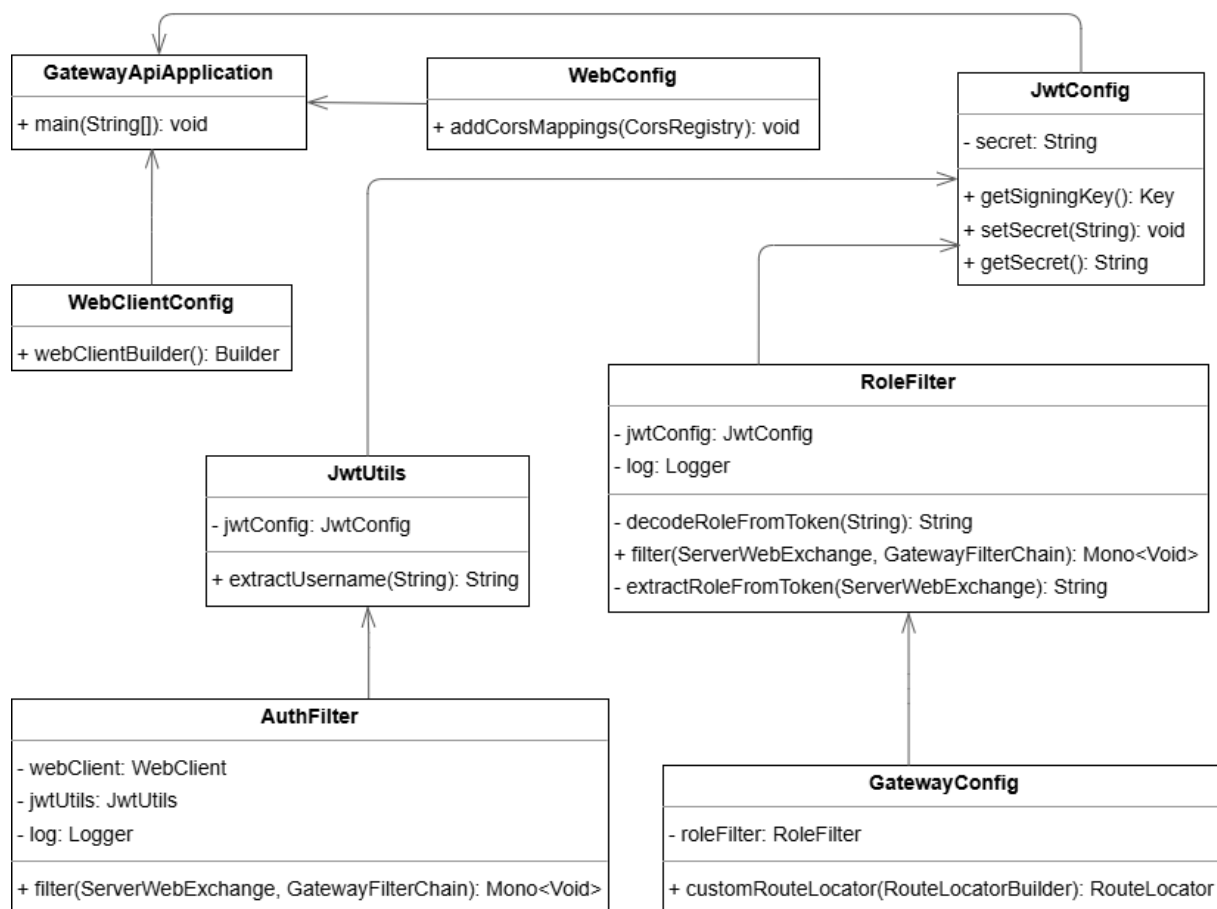


Рисунок 29 – Диаграмма классов для сервиса API шлюза

В таблице 13 приведено описание назначения каждого класса сервиса API шлюза.

Таблица 13 – Описание назначения классов сервиса API шлюза

Класс	Назначение
GatewayApiApplication	Главный класс приложения (точка входа)
WebConfig	Конфигурация CORS – настройка разрешений для кросс-доменных запросов
JwtConfig	Хранит и управляет токеном JWT и предоставляет ключи подписи токена
WebClientConfig	Создает и настраивает WebClient для отправки HTTP-запросов
JwtUtils	Утилита для извлечения данных из JWT
AuthFilter	Фильтр авторизации: проверяет токен
RoleFilter	Фильтр ролей: извлекает и проверяет роль пользователя из токена
GatewayConfig	Настройка маршрутов для API шлюза

Фрагменты исходного кода серверной части приложения представлены в приложении Б.

3.3.2 Реализация клиентской части приложения

В рамках выпускной квалификационной работы клиентская часть проекта была разработана с использованием языка программирования JavaScript и фреймворка React JS. Было построено компонентное приложение с возможностью масштабирования и повторного использования кода. Для управления состоянием и обработки пользовательского ввода использованы библиотеки MobX и React Hook Form. Для взаимодействия с REST API применена библиотека Axios.

Визуализация генеалогического дерева реализована с использованием React Flow, уведомления с помощью react-toastify. Также использована интеграция с внешними сервисами, такими как Яндекс.Карты.

Разработанная диаграмма компонентов клиентского приложения представлена на рисунке 30. На диаграмме представлены следующие компоненты:

- app.jsx – главный компонент программы;
- treesTablePage.jsx – компонент для страницы деревьев;
- personsTablePage.jsx – компонент для страницы поиска людей;
- mainPage.jsx – компонент для страницы навигации;
- relativesTablePage.jsx – компонент для страницы родственников;
- personPage.jsx – компонент для страницы профиля человека;
- treePage.jsx – компонент для страницы просмотра дерева;
- authPage.jsx – компонент для страницы авторизации пользователей;
- commentsPage.jsx – компонент для страницы просмотра комментариев;
- linksPage.jsx – компонент для страницы полезных ссылок;
- lifePage.jsx – компонент для страницы истории жизни человека.

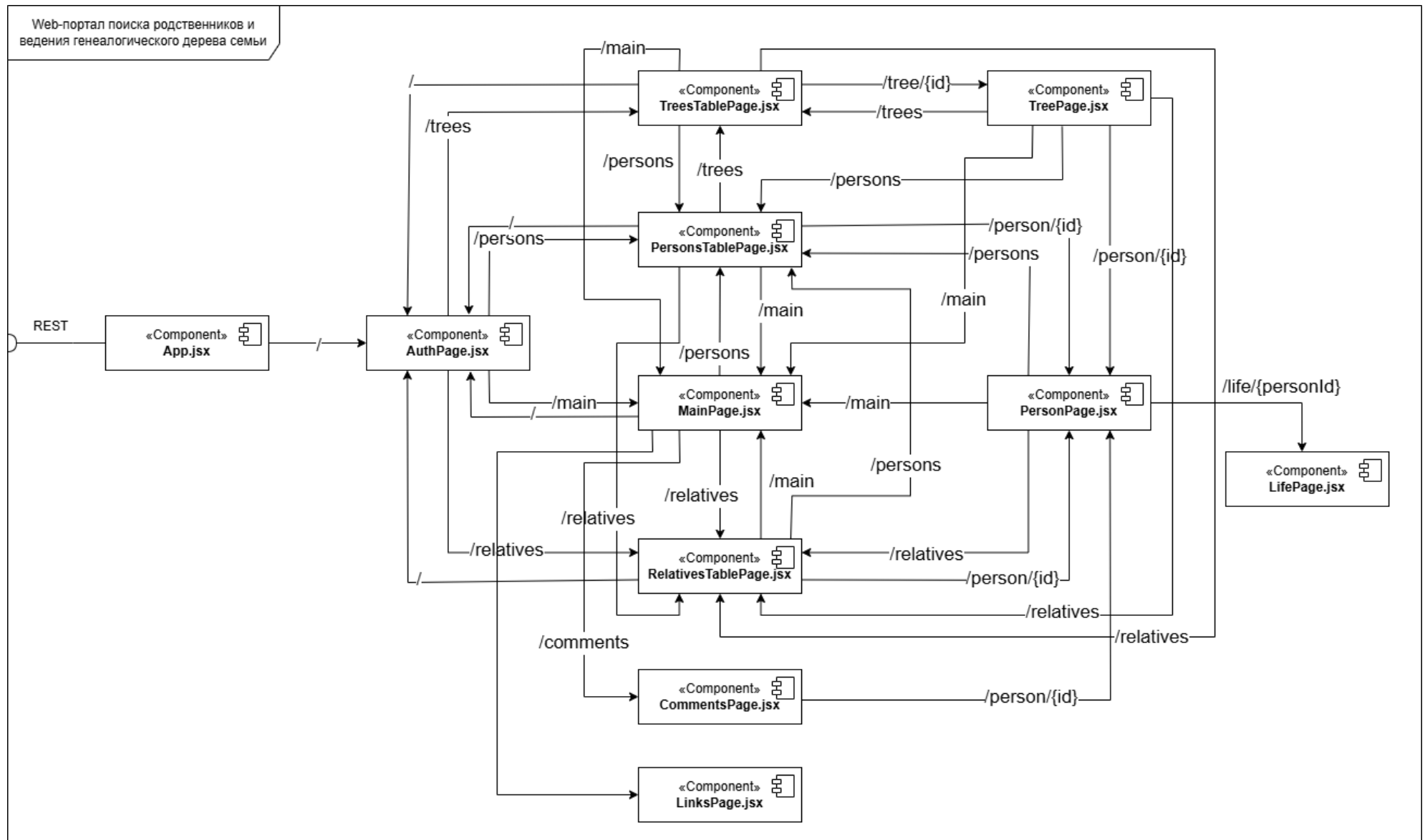


Рисунок 30 – Диаграмма компонентов клиентского приложения

Листинг фрагментов программы клиентской части приложения приведен в приложении В.

4 Контроль качества программного продукта

4.1 Обоснование выбора модели тестирования

Существует несколько основных видов тестирования программного обеспечения, каждый из которых решает определённые задачи и применяется на разных этапах разработки:

- функциональное – проверяет, соответствует ли поведение системы заявленным требованиям;
- нефункциональное – оценивает характеристики работы системы;
- интеграционное – проверяет взаимодействие между модулями или компонентами системы;
- модульное – оценивает отдельные компоненты в изоляции;
- регрессионное – проводится после внесения изменений, чтобы убедиться, что существующий функционал не нарушен;
- нагрузочное – оценивает поведение системы при большой нагрузке;
- стресс-тестирование – проверяет, как система работает за пределами допустимых нагрузок, и как она восстанавливается после сбоев;
- тестирование безопасности – проверка защиты данных, авторизации, уязвимостей к атакам;
- кроссбраузерное и кроссплатформенное – необходимо, чтобы убедиться, что система работает на разных устройствах, ОС и браузерах;
- тестирование удобства – оценивает, насколько интерфейс системы интуитивно понятен, прост и эффективен для пользователя;
- санитарное – проверка, что исправленные баги действительно исправлены и всё критично работает.

В выпускной квалификационной работе использованы следующие модели тестирования: функциональное, интеграционное и нагрузочное. Выбор этих

моделей обусловлен необходимостью обеспечения надежности, стабильности и высокой производительности системы на всех уровнях её работы.

4.2 Тестирование web-портала поиска родственников и ведения генеалогического дерева семьи

Тестирование программного обеспечения необходимо по нескольким важным причинам:

- предоставляет разработчикам информацию, которая помогает выявлять и устранять ошибки на ранних этапах разработки;
- менеджеры получают данные для объективной оценки рисков, связанных с эксплуатацией системы;
- тестирование направлено на создание программного продукта с минимальным количеством дефектов, что повышает его надежность;
- система проектируется с учетом требований к тестируемости, что облегчает её анализ, проверку соответствия спецификациям и сопровождение;
- тестирование подтверждает соответствие системы заданным требованиям, доказывая её функциональность и корректную работу [28].

4.2.1 Функциональное тестирование. Тестирование чёрным ящиком

Функциональное тестирование, также известное как тестирование методом «черного ящика» (black-box testing), представляет собой подход к проверке программного обеспечения, при котором не требуется доступ к его внутренней структуре или коду. Основное внимание уделяется взаимодействию с внешним интерфейсом системы [29].

На данном этапе было проведено тестирование граничными условиями.

Тестирование граничными условиями – это метод тестирования программного обеспечения, при котором проверяется поведение системы на границах допустимых входных значений. Этот подход основан на том, что ошибки чаще всего проявляются именно в крайних точках диапазона, а не в его середине. Для тестирования выбираются минимальные, максимальные и

предельные значения параметров, а также значения, находящиеся чуть выше или чуть ниже границы.

Ниже в таблице 14 определены границы для поля имени человека.

Таблица 14 – Границы тестирования поля имени человека

Границы	
Физические	Отрицательная длина имени (Ф1)
Логические	Имя содержит запрещенные символы (Л1)
Технологические	Имя длиннее максимально возможного (переполнение памяти) (Т1), Имя содержит невалидную кодировку (Т2)
Произвольные	Минимально допустимое значение (П1), Максимально допустимое значение (П2), На 1 больше максимума (П3)

Результаты функционального тестирования на примере поля имени при создании человека в дереве представлены в таблице 15 и в приложении Ж.

Таблица 15 – Результаты функционального тестирования ввода поля имени

№	Границы	Входные данные	Выходные данные
1	Л1	«Мария678»	Ошибка: запрещенные символы (рисунок Ж.1)
2	Т1	«Анна-Юлия-Полина-Екатерина-Ульяна-Валерия»	Ошибка: превышение 32 символов (рисунок Ж.2)
3	Т2	«  »	Ошибка: недопустимые символы (неподдерживаемая кодировка) (рисунок Ж.3)
4	П1	«»	Успешно (рисунок Ж.4)
5	П2	«Maximiliana-Alexandrietta-Josephina»	Успешно (рисунок Ж.5)
6	П3	«Maximiliana-Alexandrietta-Josephinaaaaa»	Ошибка: имя больше 32 символов (рисунок Ж.6)

Все тест-кейсы были пройдены успешно, что говорит о корректной работе программы.

4.2.2 Интеграционное тестирование

В дополнение к функциональному тестированию было проведено интеграционное тестирование.

Интеграционное тестирование – это процесс проверки взаимодействия между модулями системы, при котором отдельные компоненты объединяются и тестируются как единое целое. Оно выявляет ошибки, возникающие на этапе интеграции, которые не обнаруживаются при модульном тестировании [30].

В рамках интеграционного тестирования особое внимание уделялось взаимодействию компонентов, связанных базой данных, через gateway-api. Были проведены API-запросы, корректность возврата данных, а также отклики при ошибках или некорректных данных.

Тестирование было проведено на примере запроса создания человека в дереве, который требует авторизации в роли Пользователь. Исходный код тестов приведен в приложении 3, а его результаты – на рисунках 31 – 33.

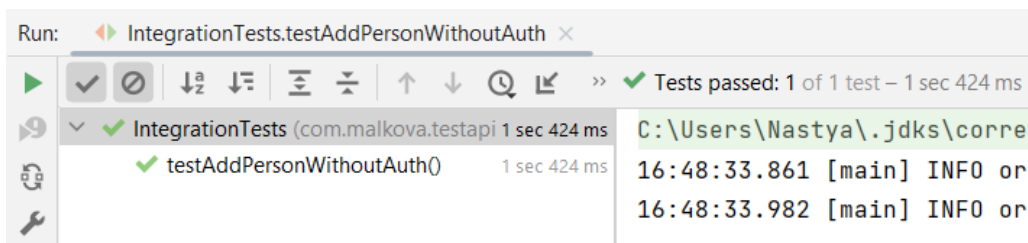


Рисунок 31 – Результат интеграционного теста на проверку попытки отправки запроса, требующего авторизацию, без авторизации

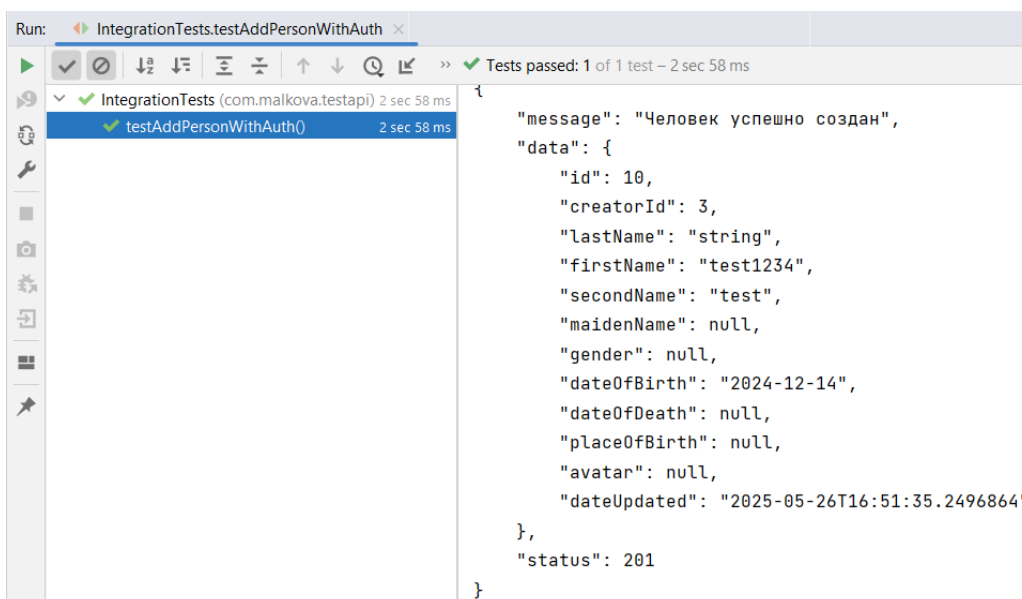


Рисунок 32 – Результат интеграционного теста на проверку отправки запроса на создание человека, с полученным токеном роли пользователя

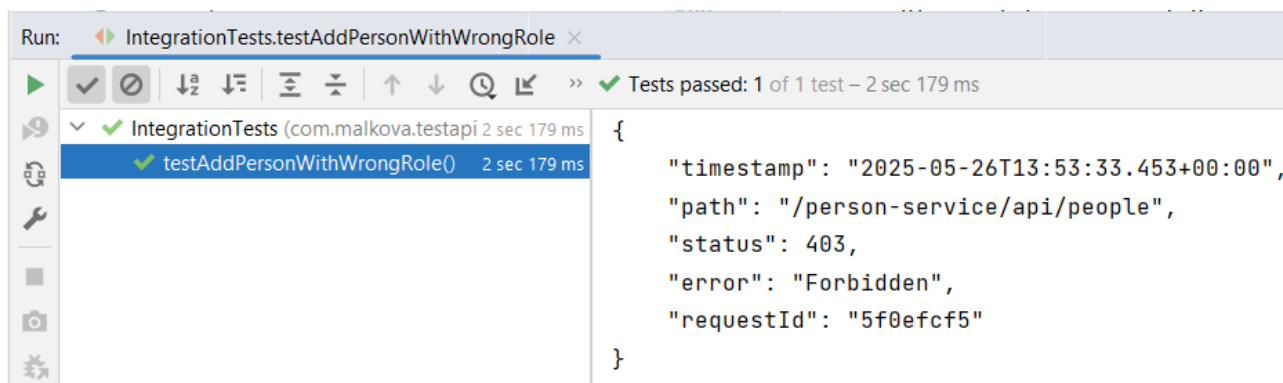


Рисунок 33 – Результат интеграционного теста на проверку попытки отправки запроса на создание человека, с недопустимой ролью

Все тесты интеграционного тестирования прошли успешно. Следовательно, можно говорить о корректной работе данного функционала.

4.2.3 Нагрузочное тестирование

Нагрузочное тестирование – это метод оценки производительности программного обеспечения или системы под определёнными нагрузками для определения её поведения в условиях интенсивного использования. Его цель – выявить, как приложение реагирует на различные объёмы данных или запросов, а также определить его способность эффективно функционировать в условиях высокой нагрузки.

Для проведения нагрузочного тестирования клиентской части приложения был выбран Lighthouse. Он позволяет анализировать различные показатели качества веб-сайта. Основными показателями являются:

- производительность (performance) – скорость загрузки и отклика страницы;
- доступность (accessibility) – насколько сайт доступен для людей с ограниченными возможностями;
- лучшие практики (best practices) – оценка соблюдения стандартов веб-разработки, таких как безопасное подключение, использование https, защита от уязвимостей;

– поисковая оптимизация (SEO) – наличие мета-тегов, адаптивность для мобильных устройств и другие параметры.

Полученные результаты нагрузочного тестирования приведены на рисунке 34.

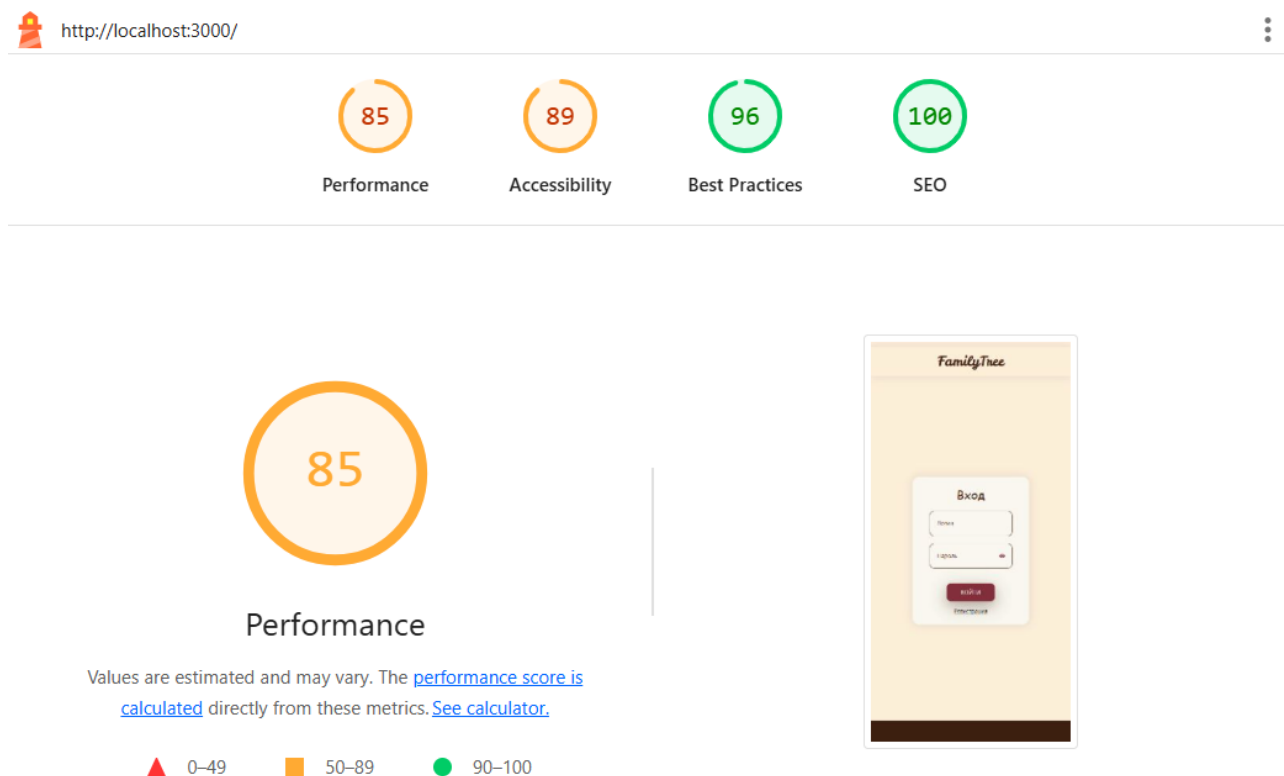


Рисунок 34 – Результаты нагрузочного тестирования

Полученные результаты нагрузочного тестирования показывают высокое качество разработанного приложения.

Показатель производительности составил 85 баллов. Данный показатель демонстрирует хороший уровень оптимизации рендеринга.

Оценка доступности составила 89 баллов и подтверждает доступность приложения для большинства категорий пользователей.

Показатели «Лучшие практики» и «Поисковая оптимизация» составили 96 и 100 баллов соответственно. Они подтверждают, что разработанное приложение обеспечивает безопасное подключение и полностью оптимизировано для поисковых систем.

После внесения правок было выполнено регрессионное тестирование. Его целью было подтверждение устранения ранее обнаруженных ошибок.

4.3 Оценка метрик кода

Метрика кода – это мера, позволяющая получить численное значение некоторого свойства программного обеспечения.

Одна из самых простых и распространенных – метрика количества строк кода (SLOC), позволяющая оценить объем программного продукта. Еще одним популярным видом метрик являются метрики Холстеда, основанные на подсчете определенных элементов кода и предназначенные для анализа его сложности.

На примере функции создания ребенка для человека вычислим метрики кода. В листинге 1 представлен код исходной функции.

Листинг 1 – Функция createChildForPerson

```
public PersonResponse createChildForPerson(PersonDTO dto, Long
creatorId, Long treeId, Long parentId) {
    // Проверка: все ли поля null
    if (isAllFieldsNull(dto)) {
        throw new IllegalArgumentException("At least one field
must be provided in PersonDTO");
    }

    // Проверка: извлечение дерева
    Tree tree = treeRepository.findById(treeId)
        .orElseThrow(() -> new NoSuchElementException("Tree
not found"));

    // Извлечение создателя
    User creator = userRepository.findById(creatorId)
        .orElseThrow(() -> new NoSuchElementException("User
not found"));

    // Извлечение родителя
    Person parent = personRepository.findById(parentId)
        .orElseThrow(() -> new NoSuchElementException("Parent
not found"));

    // Создание нового человека
    Person person = Person.builder()
        .firstName(dto.getFirstName())
        .secondName(dto.getSecondName())
        .lastName(dto.getLastName())
        .maidenName(dto.getMaidenName())
```

```

        .dateOfBirth(dto.getDateOfBirth())
        .dateOfDeath(dto.getDateOfDeath())
        .gender(dto.getGender())
        .creator(creator)
        .build();
person = personRepository.save(person);

// Создание отношения "ребенок-родитель"
Relationship relationship = Relationship.builder()
    .creator(creator)
    .firstPerson(parent)
    .secondPerson(person)
    .name(RelationshipType.PARENT_CHILD.name())
    .tree(tree) // Привязка к дереву
    .build();
relationshipRepository.save(relationship);
// Возврат результата
return mapToPersonResponse(person);
}

```

4.3.1 Метрика SLOC

Метрика SLOC (Source Lines of Code) используется для измерения размера программного обеспечения путём подсчёта количества строк в исходном коде программы. Она часто применяется для прогнозирования объёма усилий, необходимых для разработки программы, а также для оценки производительности программирования или сопровождаемости после создания программного обеспечения [31].

Вычислим метрики размера программы на примере кода, приведенного в листинге 1. Результаты вычислений представлены в таблице 16.

Таблица 16 – Метрики размера программы (SLOC)

Метрика	Количество
Количество строк кода	34
Количество строк комментариев	8

Нормой для процента комментариев является:

- 10 – 25% – хорошо документированный код;
- >25% – избыточная документация;
- <10% – код недостаточно документирован.

Вычислим процент комментариев для выбранного кода. Он равен $(8/34)*100\%=23,5\%$. Это говорит о том, что код содержит достаточное количество пояснений для его понимания, но при этом не перегружен избыточными комментариями.

4.3.2 Метрика Холстеда

Метрика Холстеда рассчитывается путем анализа количества строк и синтаксических элементов исходного кода программы, включая такие характеристики, как количество уникальных операторов, количество уникальных операндов, общее количество операторов и общее количество операндов [32 – 33]. Для измерения данных характеристик был составлен словарь операторов (таблица 17).

Таблица 17 – Словарь операторов

Оператор	Число вхождений
=	8
.	13
if	1
throw	2
findById	3
orElseThrow	3
builder	2
get	7
save	2
return	1

Также был составлен словарь операндов (таблица 18).

Таблица 18 – Словарь операндов

Операнд	Число вхождений
dto	1
creatorId	1
treeId	1
parentId	1
creator	2
tree	1
parent	1
person	3
relationship	1

Продолжение таблицы 18

Операнд	Число вхождений
firstName	1
secondName	1
lastName	1
maidenName	1
dateOfBirth	1
dateOfDeath	1
gender	1

Из вышеописанных таблиц получаем следующие данные: число уникальных операторов программы $n1 = 10$, число уникальных операндов программы $n2 = 16$, общее число операторов в программе $N1 = 42$, общее число операндов в программе $N2 = 19$.

Вычислим некоторые из существующих характеристик на основании полученных данных:

- длина программы (N): $N = N_1 + N_2 = 61$;
- словарь программы (n): $n = n_1 + n_2 = 26$;
- теоретический объем программы (V): $V = N \cdot \log_2 N \approx 361,77$;
- сложность программы (D): $D = \left(\frac{n_1}{2}\right) \cdot \left(\frac{N_2}{n_2}\right) \approx 5,94$;
- оценка интеллектуальных усилий (E): $E = D \cdot V \approx 2149$.

Если бы сложность программы (D) была больше 15 и интеллектуальные усилия (E) были больше 10000, то это означало бы, что поддержка кода была бы затруднительна.

На основе вычисленных характеристик, можно заключить, что программа хорошо разбита на функции, соответствует критериям читаемости и проста для дальнейшей поддержки.

Заключение

В ходе выполнения выпускной квалификационной работы был разработан web-портал поиска родственников и ведения генеалогического дерева семьи.

На этапе анализа предметной области был произведен сравнительный анализ аналогичных разработок, в следствии чего была поставлена задача на разработку. Также было осуществлено планирование процесса разработки и произведен расчёт затрачиваемого бюджета. Общий срок разработки составил 56 дней. Общая сумма необходимого бюджета составила 215800 рублей. Показатели подтверждают, что разработка является финансово оправданной.

На этапе анализа требований были определены функциональные, нефункциональные и архитектурные требования, а также требования к пользовательскому интерфейсу. Была спроектирована модель базы данных. Также был осуществлен выбор технологий и средств разработки.

На этапе разработки было реализовано приложение, позволяющее создавать генеалогические деревья, заполнять данные родственников и осуществлять поиск новых родственников. Реализована возможность добавления истории жизни для человека с прикреплением медиафайлов, а также осуществлена возможность указания места рождения человека на Яндекс.Картах. Каждый пользователь может просматривать любые данные деревьев и людей, а также оставлять комментарии о их достоверности. Для разработки были использованы следующие технологии, библиотеки и языки программирования: Java, Spring, Spring Security, JPA, JWT, MySQL, JavaScript, React, Axios, MobX, React Flow, React Hook Form, react-toastify, tailwindcss. Использованные среды разработки: MS Server, WebStorm, IntelliJ IDEA.

На этапе тестирования была осуществлена проверка работоспособности продукта и корректность его функционала. Все обнаруженные проблемы были успешно устранены. Также были подсчитаны и проанализированы метрики кода.

В будущем планируется интеграция приложения с сервисами генетического анализа, которые позволят пользователям загружать данные ДНК-

тестов и находить потенциальных родственников на основе генетических совпадений. Также планируется реализация возможности совместного редактирования в реальном времени с отображением действий других пользователей и сохранением истории изменений. Для повышения совместимости с другими системами планируется реализация функциональности импорта и экспорта данных в стандартные форматы для генеалогических деревьев, такие как GEDCOM. Это позволит пользователям легко переносить свои деревья из сторонних сервисов, а также делиться результатами исследований в общепринятом виде.

Итоги выполненной выпускной квалификационной работы свидетельствуют о полном достижении всех поставленных целей.

Список использованных источников

1. Правительство Российской Федерации. Программа Цифровая экономика Российской Федерации // Национальные проекты [Электронный ресурс]. URL: <http://static.government.ru/media/files/9gFM4FHj4PsB79I5v7yLVuPgu4bvR7M0.pdf> (дата обращения: 15.04.2025)
2. Национальный проект Культура // Национальные проекты [Электронный ресурс]. URL: <https://culture.gov.ru/about/national-project/about-project/> (дата обращения: 15.04.2025)
3. ФСБ России рассекретила архивы к 80-летию Победы // Newsler [Электронный ресурс]. URL: <https://www.newsler.ru/society/2025/03/18/fsb-rossii-rassekretila-arhivy-k-80-letiyu-pobedy> (дата обращения: 15.04.2025)
4. Вигерс К. Разработка требований к программному обеспечению. 3-е изд., дополненное: Пер. с англ. / К. Вигерс, Дж. Битти. – М.: Издательство «Русская редакция»; СПб.: БХВ-Петербург, 2014 – 736 стр.
5. MyHeritage // Официальный сайт сервиса MyHeritage [Электронный ресурс]. URL: <https://www.myheritage.com.tr/home/> (дата обращения: 22.04.2025).
6. Кратко о сервисе Famiry // Официальный сайт сервиса Famiry [Электронный ресурс]. URL: <https://famiry.notion.site/Famiry-d90c3e47e5064c5784bfa7177d3085f2> (дата обращения: 22.04.2025)
7. Бесплатное генеалогическое дерево онлайн – сервис Famiry // Официальный сайт сервиса Famiry [Электронный ресурс]. URL: <https://famiry.ru/> (дата обращения: 22.04.2025)
8. Дерево Жизни – программа для построения генеалогических древ // Официальный сайт сервиса Дерево Жизни [Электронный ресурс]. URL: <https://genery.com/ru/> (дата обращения: 22.04.2025)
9. FamilySearch.org // Официальный сайт сервиса FamilySearch [Электронный ресурс]. URL: <https://www.familysearch.org/ru/?&> (дата обращения: 22.04.2025)

10. Генеалогический форум – поиск людей в прошлом // Официальный сайт форума ВГД [Электронный ресурс]. URL: <https://forum.vgd.ru/> (дата обращения: 22.04.2025)
11. Буч Г., Рамбо Дж., Якобсон И. UML. Руководство пользователя / Пер. с англ. 2-е изд. – М.: ДМК Пресс, 2016. – 640 с.
12. Ехлаков, Ю.П. Управление программными проектами. Стандарты, модели: Учебное пособие / Ю.П. Ехлаков. – СПб.: Лань, 2019. – 244 с.
13. Беркун, С. Искусство управления IT-проектами / С. Беркун. – СПб.: Питер, 2007. – 400 с.
14. Гонтарева, И.В. Управление проектами / И.В. Гонтарева, Р.М. Нижегородцев, Д.А. Новиков. – М.: КД Либроком, 2018. – 384 с.
15. Скороход, С.В. Управление проектами средствами Microsoft Project. Курс лекций. / С.В. Скороход. – М.: Интуит НОУ, 2016. – 319 стр.
16. ProjectLibre Desktop // Официальный сайт программного обеспечения ProjectLibre [Электронный ресурс]. URL: <https://www.projectlibre.com/projectlibre-desktop/> (дата обращения: 25.04.2025)
17. Primavera P6 Enterprise Project Portfolio Management // Официальный сайт компании Oracle [Электронный ресурс]. URL: <https://www.oracle.com/construction-engineering/primavera-p6/> (дата обращения: 25.04.2025)
18. GanttPRO: Онлайн-Диаграмма Ганта // Официальный сайт программного обеспечения GanttPRO [Электронный ресурс]. URL: <https://ganttpro.com/ru/> (дата обращения: 25.04.2025)
19. НОУ ИНТУИТ. Нотация и семантика языка UML. Лекция 3: Элементы графической нотации диаграммы вариантов использования // Официальный сайт НОУ ИНТУИТ [Электронный ресурс]. URL: <https://intuit.ru/studies/courses/32/32/lecture/1004> (дата обращения: 03.05.2025)
20. Фаулер, М. Архитектура корпоративных приложений. [пер. с англ.] / М. Фаулер. – М.: Издательский дом «Вильяме», 2006. – 544 с.

21. Курс JSP & Servlets – Лекция: Трехуровневая архитектура // JavaRush [Электронный ресурс]. – URL: <https://javarush.com/quests/lectures/questservlets.level14.lecture01> (дата обращения: 29.04.2025)
22. Кирютенко, Ю.А. Объектно-ориентированное программирование. Архитектура MVC: учебное пособие. / Ю.А. Кирютенко. – Ростов-на-Дону, 2008. – 153 с.
23. Вигерс, К. Разработка требований к программному обеспечению. [пер. с англ.] / К. Вигерс, Д. Битти. – 3-е изд., доп. – СПб.: БХВ-Петербург, 2014. – 736 с.
24. Маглинец, Ю.А. Анализ требований к информационным системам: конспект лекций // Федеральное агентство по образованию, Сибирский федеральный университет [Электронный ресурс]. URL: <https://ivan-shamaev.ru/wp-content/uploads/2013/06/Information-systems-analysis-and-requirements-analysis.pdf> (дата обращения: 29.04.2025)
25. Желнова, Н. Нефункциональные требования к программному обеспечению // Habr [Электронный ресурс]. URL: <https://habr.com/ru/articles/231961/> (дата обращения: 29.04.2025)
26. Войтюк, Т.Е. Основы проектирования реляционных баз данных средствами инструментальной среды / Т.Е. Войтюк, И.С. Осетрова – СПб: Университет ИТМО, 2020 – 70 с.
27. Миковски, М.С. Разработка одностраничных приложений / М.С. Миковски, Дж.К. Пауэлл; пер. с англ. – М.: ДМК-Пресс, 2018. – 512 с.
28. Бейзер, Б. Тестирование черного ящика. Технологии функционального тестирования программного обеспечения и систем / Б. Бейзер // СПб.: Питер, 2004. – 318 с.
29. Пероцкая, В. Н. Основы тестирования программного обеспечения: учеб. пособие / В. Н. Пероцкая, Д. А. Градусов // Владимир: Изд-во ВлГУ, 2017 – 100 с.

30. Литвинов, А.А. Тестирование информационных систем: модульное, интеграционное, системное: учебное пособие. / А.А. Литвинов, Н.В. Карпенко // Д.: Лира, 2016. – 283 с.
31. Бхатт, К. Анализ метрики строк исходного кода (SLOC) / К. Бхатт, В. Тарей, П. Патель // International Journal of Emerging Technology and Advanced Engineering [Электронный ресурс]. URL: https://www.researchgate.net/publication/281840565_Analysis_Of_Source_Lines_Of_CodeSLOC_Metric (дата обращения: 29.05.2025).
32. Аверьянов, А.В. Применение метрик Холстеда для количественного оценивания характеристик программ ЭВМ / А.В. Аверьянов, И.Н. Кошель, В.В. Кузнецов // Известия высших учебных заведений. Приборостроение. – 2019. – №11. – С. 970 – 975.
33. Полякова, Э.Р. Исследование программного обеспечения встраиваемой вычислительной системы метриками Холстеда / Э.Р. Полякова, А.А. Поляков, А.Л. Шустов, Д.Г. Придачкин // Cloud of Science. – 2020. – №4. – С. 950 – 959.

Приложение А
(справочное)
Глоссарий

MySQL – система управления базами данных (СУБД). Хранит данные в виде таблиц, связанных между собой. Для взаимодействия с MySQL используется язык SQL (Structured Query Language), который позволяет выполнять запросы к базе данных.

Система управления базами данных (СУБД) – совокупность программных средств, обеспечивающих управление созданием и использованием баз данных.

Java – строго типизированный объектно-ориентированный язык программирования.

Spring – фреймворк с открытым исходным кодом для языка программирования Java. Создан для упрощения разработки и поддержки масштабируемых, слабосвязанных и повторно используемых приложений.

Spring Security это фреймворк для языка программирования Java, предоставляющий механизмы построения систем аутентификации и авторизации, а также другие возможности обеспечения безопасности для приложений, созданных с помощью Spring Framework.

React – JavaScript-библиотека с открытым исходным кодом для разработки пользовательских интерфейсов.

Single Page Application (SPA) – это веб-приложение, которое загружает только одну HTML-страницу и динамически обновляет ее содержимое без необходимости полной перезагрузки страницы при взаимодействии с пользователем.

Axios – JavaScript-библиотека для выполнения HTTP-запросов.

REST – архитектурный стиль взаимодействия компонентов распределённого приложения в сети. Обычно используется протокол HTTP.

React-toastify – библиотека для отображения уведомлений в React-приложениях в виде всплывающих уведомлений.

React Flow – библиотека для визуального отображения графов и деревьев.

React Hook Form – библиотека для управления формами и валидацией в React.

Mobx – библиотека для React приложений, дающая разработчикам инструмент для глобального использования переменных и методов между разными компонентами.

JSON Web Token – открытый стандарт для создания токенов доступа, основанный на формате JSON.

Tailwind CSS – CSS-фреймворк, предоставляющий утилитарные классы для разработки пользовательских интерфейсов.

React Yandex Maps – библиотека, обеспечивающая интеграцию карт Яндекса в React-приложения.

Frontend (клиентская часть) – часть приложения, с которой пользователь взаимодействует напрямую.

Backend (серверная часть) – часть приложения, которая обрабатывает бизнес-логику, осуществляет работу с базой данных и реализацию API.

API – набор правил, по которым приложения или части программы взаимодействуют друг с другом.

Приложение Б

(обязательное)

Фрагменты исходного кода серверной части web-портала

Листинг Б.1 – Файл CommentPersonService.java

```

@Service
@RequiredArgsConstructor
public class CommentPersonService {

    private final CommentPersonRepository commentRepository;
    private final UserRepository userRepository;
    private final PersonRepository personRepository;
    private final JwtService jwtService;

    @Transactional
    public CommentPersonResponse addComment(CommentPersonRequest
dto, String token) {
        // Извлекаем ID пользователя из токена
        Long userId = jwtService.extractUserId(token);

        User user = userRepository.findById(userId)
            .orElseThrow(() -> new
NoSuchElementException("Пользователь не найден"));

        Person person =
personRepository.findById(dto.getPersonId())
            .orElseThrow(() -> new
NoSuchElementException("Человек не найден"));

        CommentPerson comment = CommentPerson.builder()
            .user(user)
            .person(person)
            .text(dto.getText())
            .grade(dto.getGrade())
            .dateAdded(LocalDateDateTime.now())
            .status(CommentStatus.CREATED)
            .build();

        CommentPerson saved = commentRepository.save(comment);
        return mapToResponse(saved);
    }

    public void deleteComment(Long commentId) throws
NoSuchElementException {
        if (!commentRepository.existsById(commentId)) {
            throw new NoSuchElementException("Комментарий не
найден");
        }
        commentRepository.deleteById(commentId);
    }

    public List<CommentPersonResponse>
getAllCommentsByPersonId(Long personId) {
        return
commentRepository.findAllByPersonId(personId).stream()

```

```

        .map(this::mapToResponse)
        .collect(Collectors.toList());
    }

    public List<CommentPersonResponse> getAllCommentsForMy(String
token) {
        // Извлекаем ID пользователя из токена
        Long userId = jwtService.extractUserId(token);

        userRepository.findById(userId).orElseThrow(() -> new
NoSuchElementException("Пользователь не найден"));

        // 2. Получаем всех людей, созданных этим пользователем
        List<Person> personsCreatedByUser =
userRepository.findAllByCreatorId(userId);

        // 3. Извлекаем их ID
        List<Long> personIds = personsCreatedByUser.stream()
            .map(Person::getId)
            .collect(Collectors.toList());

        // 4. Получаем все комментарии, написанные к этим людям
        List<CommentPerson> comments =
commentRepository.findAllByPersonIdIn(personIds);

        // 5. Преобразуем в DTO
        return comments.stream()
            .map(this::mapToResponse)
            .collect(Collectors.toList());
    }

    public List<CommentPersonResponse> getAllBadComments() {
        List<CommentPerson> badComments =
commentRepository.findAllByStatus(CommentStatus.BAD);

        return badComments.stream()
            .map(this::mapToResponse)
            .collect(Collectors.toList());
    }

    public void setCommentStatusCreated(Long commentId) {
        CommentPerson comment =
commentRepository.findById(commentId)
            .orElseThrow(() -> new
NoSuchElementException("Комментарий не найден"));
        comment.setStatus(CommentStatus.CREATED);
        commentRepository.save(comment);
    }

    public void setCommentStatusBad(Long commentId) {
        CommentPerson comment =
commentRepository.findById(commentId)

```

```

        .orElseThrow(() -> new
NoSuchElementException("Комментарий не найден"));
        comment.setStatus(CommentStatus.BAD);
        commentRepository.save(comment);
    }

    public Double getAverageGradeForPerson(Long personId) {
        return
commentRepository.findAverageGradeByPersonId(personId);
    }

    private CommentPersonResponse mapToResponse(CommentPerson
comment) {
        return CommentPersonResponse.builder()
            .id(comment.getId())
            .username(comment.getUser().getUsername())
            .personId(comment.getPerson().getId())
            .text(comment.getText())
            .grade(comment.getGrade())
            .dateAdded(comment.getDateAdded())
            .status(comment.getStatus())
            .build();
    }
}

```

Листинг Б.2 – Файл LifeHistoryService.java

```

@Service
@RequiredArgsConstructor
public class LifeHistoryService {
    private final LifeHistoryRepository lifeHistoryRepository;
    private final JwtService jwtService;
    private final UserRepository userRepository;
    private final PersonRepository personRepository;
    private final PhotoService photoService;
    private final VideoService videoService;

    public LifeHistoryResponse createLifeHistoryWithFiles(
        LifeHistoryRequest request,
        List<MultipartFile> photoFiles,
        List<MultipartFile> videoFiles,
        String token
    ) {
        Long userId = jwtService.extractUserId(token);
        User user = userRepository.findById(userId)
            .orElseThrow(() -> new
NoSuchElementException("Пользователь не найден"));

        Person person =
personRepository.findById(request.getPersonId())
            .orElseThrow(() -> new
NoSuchElementException("Человек не найден"));
    }
}

```

```

LifeHistory history = LifeHistory.builder()
    .user(user)
    .person(person)
    .text(request.getText())
    .dateStart(request.getDateStart())
    .dateEnd(request.getDateEnd())
    .dateCreated(LocalDateTime.now())
    .dateUpdated(LocalDateTime.now())
    .build();

LifeHistory saved = lifeHistoryRepository.save(history);

// Сохраняем файлы
photoService.saveFiles(photoFiles, saved, user);
videoService.saveFiles(videoFiles, saved, user);

return mapToResponse(saved);
}

public LifeHistoryResponse updateWithFiles(
    Long id,
    LifeHistoryRequest request,
    List<MultipartFile> photoFiles,
    List<MultipartFile> videoFiles,
    String token
) {
    Long userId = jwtService.extractUserId(token);
    User user = userRepository.findById(userId)
        .orElseThrow(() -> new
NoSuchElementException("Пользователь не найден"));

    LifeHistory history = lifeHistoryRepository.findById(id)
        .orElseThrow(() -> new
NoSuchElementException("История жизни не найдена"));

    // Обновляем текст и даты
    history.setText(request.getText());
    history.setDateStart(request.getDateStart());
    history.setDateEnd(request.getDateEnd());
    history.setDateUpdated(LocalDateTime.now());

    LifeHistory updated = lifeHistoryRepository.save(history);

    // Удаляем указанные фото и видео
    if (request.getPhotoIdsToDelete() != null &&
!request.getPhotoIdsToDelete().isEmpty()) {
photoService.deleteByIds(request.getPhotoIdsToDelete());
    }
    if (request.getVideoIdsToDelete() != null &&
!request.getVideoIdsToDelete().isEmpty()) {

```

```

videoService.deleteByIds(request.getVideoIdsToDelete());
    }

    // Сохраняем новые файлы
    photoService.saveFiles(photoFiles, updated, user);
    videoService.saveFiles(videoFiles, updated, user);

    return mapToResponse(updated);
}

public void delete(Long id, String token) {
    LifeHistory history = lifeHistoryRepository.findById(id)
        .orElseThrow(() -> new
NoSuchElementException("История жизни не найдена"));

    lifeHistoryRepository.delete(history);
}

public List<LifeHistoryResponse> getAllByPersonId(Long
personId) {
    return
lifeHistoryRepository.findAllByPersonId(personId).stream()
        .map(this::mapToResponse)
        .collect(Collectors.toList());
}

public LifeHistoryResponse getById(Long id) {
    LifeHistory history = lifeHistoryRepository.findById(id)
        .orElseThrow(() -> new
NoSuchElementException("История жизни не найдена"));
    return mapToResponse(history);
}

private PhotoResponse toPhotoResponse(Photo photo) {
    PhotoResponse response = new PhotoResponse();
    response.setId(photo.getId());
    response.setOwnerUsername(photo.getOwner().getUsername());
    response.setOwnerId(photo.getOwner().getId());
    response.setPath(photo.getPath());
    return response;
}

private VideoResponse toVideoResponse(Video video) {
    VideoResponse response = new VideoResponse();
    response.setId(video.getId());
    response.setOwnerUsername(video.getOwner().getUsername());
    response.setOwnerId(video.getOwner().getId());
    response.setPath(video.getPath());
    return response;
}

```

```

        private LifeHistoryResponse mapToResponse(LifeHistory entity)
    {
        return LifeHistoryResponse.builder()
            .id(entity.getId())
            .personId(entity.getPerson().getId())
            .creatorId(entity.getUser().getId())
            .dateStart(entity.getDateStart())
            .dateEnd(entity.getDateEnd())
            .text(entity.getText())
            .dateCreated(entity.getDateCreated())
            .dateUpdated(entity.getDateUpdated())
            .photos(entity.getPhotos() != null ?
entity.getPhotos().stream()
                    .map(this::toPhotoResponse)
                    .collect(Collectors.toList()) : new
ArrayList<>())
            .videos(entity.getVideos() != null ?
entity.getVideos().stream()
                    .map(this::toVideoResponse)
                    .collect(Collectors.toList()) : new
ArrayList<>())
            .build();
    }
}

```

Листинг Б.3 – Файл PersonService.java

```

@Service
@AllArgsConstructor
public class PersonService {

    /**
     * Объект репозитория для работы с MySQL
     */
    private final PersonRepository personRepository;

    private final UserRepository userRepository;

    private final TreeRepository treeRepository;

    /**
     * Объект логгера
     */
    private static final Logger log =
LoggerFactory.getLogger(PersonService.class);

    private final RelationshipRepository relationshipRepository;

    /**
     * Создание нового человека
     * @param dto Объект человека для создания
     * @return Объект созданного человека
     */
}

```

```

@Transactional
public PersonResponse create(PersonDTO dto, Long creatorId) {
    // Проверка: все ли поля null
    if (isAllFieldsNull(dto)) {
        throw new IllegalArgumentException("At least one field
must be provided in PersonDTO");
    }

    // Извлечение создателя
    User creator = userRepository.findById(creatorId)
        .orElseThrow(() -> new
NoSuchElementException("User not found"));

    Person person = Person.builder()
        .firstName(dto.getFirstName())
        .secondName(dto.getSecondName())
        .lastName(dto.getLastName())
        .maidenName(dto.getMaidenName())
        .dateOfBirth(dto.getDateOfBirth())
        .dateOfDeath(dto.getDateOfDeath())
        .gender(dto.getGender())
        .creator(creator)
        .build();

    // Сохраняем в MySQL
    person = personRepository.save(person);

    return mapToPersonResponse(person);
}

// Метод для проверки всех полей на null
private boolean isAllFieldsNull(PersonDTO dto) {
    return dto.getFirstName() == null &&
        dto.getSecondName() == null &&
        dto.getLastName() == null &&
        dto.getMaidenName() == null &&
        dto.getDateOfBirth() == null &&
        dto.getDateOfDeath() == null &&
        (dto.getGender() == null || dto.getGender() ==
Gender.NONE);
//      && dto.getIdPlaceOfBirth() == null;
}

// Чтение всех объектов из MySQL

/**
 * Получение всех людей
 * @return Список всех людей из базы данных
 */
public List<PersonResponse> readAll() {
    List<Person> persons = personRepository.findAll();

```

```

        return persons.stream()
            .map(this::mapToPersonResponse)
            .toList();
    }

    /**
     * Обновление человека по ID
     * @param id Идентификатор человека
     * @param dto Объект человека для обновления
     * @return Обновленный объект человека
     */
    @Transactional
    public PersonResponse update(Long id, PersonDTO dto) {
        if (isAllFieldsNull(dto)) {
            throw new IllegalArgumentException("Хотя бы одно поле
должно быть заполнено в PersonDTO");
        }

        Person existingPerson = personRepository.findById(id)
            .orElseThrow(() -> new
NoSuchElementException("Человек с id " + id + " не найден."));

        existingPerson.setFirstName(dto.getFirstName());
        existingPerson.setSecondName(dto.getSecondName());
        existingPerson.setLastName(dto.getLastName());
        existingPerson.setDateOfBirth(dto.getDateOfBirth());
        existingPerson.setDateOfDeath(dto.getDateOfDeath());
        existingPerson.setGender(dto.getGender());
        existingPerson.setMaidenName(dto.getMaidenName());

        // Сохраняем обновлённого человека в MySQL
        personRepository.save(existingPerson);
        log.info("Обновлён объект Person: {}", existingPerson);

        return mapToPersonResponse(existingPerson);
    }

    /**
     * Удаление человека по ID
     * @param id Идентификатор человека
     */
    @Transactional
    public void delete(Long id) {
        // Удаляем из MySQL
        personRepository.deleteById(id);
        log.info("Удалён объект с ID: {}", id);
    }

    /**
     * Поиск человека по ID
     * @param id Идентификатор человека
     * @return Объект человека с указанным идентификатором
     */

```

```

    */
    public Optional<PersonResponse> findById(Long id) {
        log.info("Ищем человека в MySQL с ID: {}", id);
        return personRepository.findById(id)
            .map(this::mapToPersonResponse);
    }

    /**
     * Удаление всех людей
     */
    public void deleteAll() {
        personRepository.deleteAll();
        log.info("Удалены все объекты из MySQL");
    }

    /**
     * Подсчёт количества людей
     * @return Количество людей в базе данных
     */
    public long countPeople() {
        long count = personRepository.count();
        log.info("Количество людей: {}", count);
        return count;
    }

    private PersonResponse mapToPersonResponse(Person person) {
        return PersonResponse.builder()
            .id(person.getId())
            .firstName(person.getFirstName())
            .secondName(person.getSecondName())
            .lastName(person.getLastName())
            .maidenName(person.getMaidenName())
            .creatorId(person.getCreator().getId())
            .gender(person.getGender())
            .dateOfBirth(person.getDateOfBirth())
            .dateOfDeath(person.getDateOfDeath())
            .placeOfBirth(person.getPlaceOfBirth())
            .dateUpdated(person.getDateUpdated())
            .build();
    }

    public List<PersonSearchResponse> searchPersons(
        String lastName,
        String firstName,
        String secondName,
        LocalDate birthDate,
        String settlement,
        String maidenName,
        LocalDate deathDate,
        Gender gender
    ) {

```

```

        return personRepository.searchPersons(lastName, firstName,
secondName, birthDate, settlement, maidenName, deathDate, gender);
    }

    public List<PersonResponse> readAllByCreatorId(Long creatorId)
    {
        return personRepository.findAllByCreatorId(creatorId)
            .stream()
            .map(person -> PersonResponse.builder()
                .id(person.getId())
                .creatorId(person.getCreator().getId())
                .lastName(person.getLastName())
                .firstName(person.getFirstName())
                .secondName(person.getSecondName())
                .maidenName(person.getMaidenName())
                .gender(person.getGender())
                .dateOfBirth(person.getDateOfBirth())
                .dateOfDeath(person.getDateOfDeath())
                .placeOfBirth(person.getPlaceOfBirth())
                .dateUpdated(person.getDateUpdated())
                .build())
            .toList();
    }

    @Transactional
    public PersonResponse createPersonWithParents(PersonDTO dto,
Long creatorId, Long treeId) {
        // Проверка: все ли поля null
        if (isAllFieldsNull(dto)) {
            throw new IllegalArgumentException("At least one field
must be provided in PersonDTO");
        }

        // Проверка: извлечение дерева
        Tree tree = treeRepository.findById(treeId)
            .orElseThrow(() -> new
NoSuchElementException("Tree not found"));

        // Извлечение создателя
        User creator = userRepository.findById(creatorId)
            .orElseThrow(() -> new
NoSuchElementException("User not found"));

        // Создание человека
        Person person = Person.builder()
            .firstName(dto.getFirstName())
            .secondName(dto.getSecondName())
            .lastName(dto.getLastName())
            .maidenName(dto.getMaidenName())
            .dateOfBirth(dto.getDateOfBirth())
            .dateOfDeath(dto.getDateOfDeath())
            .gender(dto.getGender())

```

```

        .creator(creator)
        .build();
person = personRepository.save(person);

// Создание мамы
Person mother = Person.builder()
    .firstName("Неизвестно")
    .gender(Gender.FEMALE)
    .creator(creator)
    .build();
mother = personRepository.save(mother);

// Создание отца
Person father = Person.builder()
    .firstName("Неизвестно")
    .gender(Gender.MALE)
    .creator(creator)
    .build();
father = personRepository.save(father);

// Создание отношений "ребенок-родитель"
Relationship motherRelationship = Relationship.builder()
    .creator(creator)
    .firstPerson(mother)
    .secondPerson(person)
    .name(RelationshipType.PARENT_CHILD.name())
    .tree(tree) // Привязка к дереву
    .build();
relationshipRepository.save(motherRelationship);

Relationship fatherRelationship = Relationship.builder()
    .creator(creator)
    .firstPerson(father)
    .secondPerson(person)
    .name(RelationshipType.PARENT_CHILD.name())
    .tree(tree) // Привязка к дереву
    .build();
relationshipRepository.save(fatherRelationship);

// Возврат результата
return mapToPersonResponse(person);
}

@Transactional
public PersonResponse createChildForPerson(PersonDTO dto, Long
creatorId, Long treeId, Long parentId) {
    // Проверка: все ли поля null
    if (isAllFieldsNull(dto)) {
        throw new IllegalArgumentException("At least one field
must be provided in PersonDTO");
    }
}

```

```

        // Проверка: извлечение дерева
        Tree tree = treeRepository.findById(treeId)
            .orElseThrow(() -> new
NoSuchElementException("Tree not found"));

        // Извлечение создателя
        User creator = userRepository.findById(creatorId)
            .orElseThrow(() -> new
NoSuchElementException("User not found"));

        // Извлечение родителя
        Person parent = personRepository.findById(parentId)
            .orElseThrow(() -> new
NoSuchElementException("Parent not found"));

        // Создание нового человека
        Person person = Person.builder()
            .firstName(dto.getFirstName())
            .secondName(dto.getSecondName())
            .lastName(dto.getLastName())
            .maidenName(dto.getMaidenName())
            .dateOfBirth(dto.getDateOfBirth())
            .dateOfDeath(dto.getDateOfDeath())
            .gender(dto.getGender())
            .creator(creator)
            .build();
        person = personRepository.save(person);

        // Создание отношения "ребенок-родитель"
        Relationship relationship = Relationship.builder()
            .creator(creator)
            .firstPerson(parent)
            .secondPerson(person)
            .name(RelationshipType.PARENT_CHILD.name())
            .tree(tree) // Привязка к дереву
            .build();
        relationshipRepository.save(relationship);

        // Возврат результата
        return mapToPersonResponse(person);
    }

    @Transactional
    public PersonResponse createParentForPerson(PersonDTO dto,
Long creatorId, Long treeId, Long childId) {
        // Проверка: все ли поля null
        if (isAllFieldsNull(dto)) {
            throw new IllegalArgumentException("At least one field
must be provided in PersonDTO");
        }

        // Проверка: извлечение дерева

```

```

        Tree tree = treeRepository.findById(treeId)
            .orElseThrow(() -> new
NoSuchElementException("Tree not found"));

        // Извлечение создателя
        User creator = userRepository.findById(creatorId)
            .orElseThrow(() -> new
NoSuchElementException("User not found"));

        // Извлечение ребёнка
        Person child = personRepository.findById(childId)
            .orElseThrow(() -> new
NoSuchElementException("Child not found"));

        // Создание нового родителя
        Person parent = Person.builder()
            .firstName(dto.getFirstName())
            .secondName(dto.getSecondName())
            .lastName(dto.getLastName())
            .maidenName(dto.getMaidenName())
            .dateOfBirth(dto.getDateOfBirth())
            .dateOfDeath(dto.getDateOfDeath())
            .gender(dto.getGender())
            .creator(creator)
            .build();
        parent = personRepository.save(parent);

        // Создание отношения "родитель-ребенок"
        Relationship relationship = Relationship.builder()
            .creator(creator)
            .firstPerson(parent)
            .secondPerson(child)
            .name(RelationshipType.PARENT_CHILD.name())
            .tree(tree) // Привязка к дереву
            .build();
        relationshipRepository.save(relationship);

        // Возврат результата
        return mapToPersonResponse(parent);
    }

    @Transactional
    public PersonResponse createPartnerForPerson(PersonDTO dto,
Long creatorId, Long treeId, Long personId1) {
        // Проверка: все ли поля null
        if (isAllFieldsNull(dto)) {
            throw new IllegalArgumentException("At least one field
must be provided in PersonDTO");
        }

        // Проверка: извлечение дерева
        Tree tree = treeRepository.findById(treeId)

```

```

        .orElseThrow(() -> new
NoSuchElementException("Tree not found"));

        // Извлечение создателя
        User creator = userRepository.findById(creatorId)
            .orElseThrow(() -> new
NoSuchElementException("User not found"));

        // Извлечение первого человека
        Person person1 = personRepository.findById(personId1)
            .orElseThrow(() -> new
NoSuchElementException("Person 1 not found"));

        // Создание второго человека (партнёра)
        Person partner = Person.builder()
            .firstName(dto.getFirstName())
            .secondName(dto.getSecondName())
            .lastName(dto.getLastName())
            .maidenName(dto.getMaidenName())
            .dateOfBirth(dto.getDateOfBirth())
            .dateOfDeath(dto.getDateOfDeath())
            .gender(dto.getGender())
            .creator(creator)
            .build();
        partner = personRepository.save(partner);

        // Создание отношения "Партнёр"
        Relationship partnerRelationship = Relationship.builder()
            .creator(creator)
            .firstPerson(person1)
            .secondPerson(partner)
            .name(RelationshipType.PARTNER.name())
            .tree(tree) // Привязка к дереву
            .build();
        relationshipRepository.save(partnerRelationship);

        // Возврат результата
        return mapToPersonResponse(partner);
    }

    public List<PersonResponse> getAllPersonsByTreeId(Long treeId)
    {
        List<Person> firstPersons =
personRepository.findFirstPersonsByTreeId(treeId);
        List<Person> secondPersons =
personRepository.findSecondPersonsByTreeId(treeId);

        // Объединяем и убираем дубликаты по ID
        Map<Long, Person> uniquePersons = new HashMap<>();
        Stream.concat(firstPersons.stream(),
secondPersons.stream())

```

```

        .forEach(person ->
uniquePersons.putIfAbsent(person.getId(), person));

        return uniquePersons.values().stream()
            .map(this::mapToPersonResponse)
            .collect(Collectors.toList());
    }
}

```

Листинг Б.4 – Файл RelationshipService.java

```

@Service
@RequiredArgsConstructor
public class RelationshipService {

    private final PersonRepository personRepository;
    private final RelationshipRepository relationshipRepository;
    private final TreeRepository treeRepository;
    private final UserRepository userRepository;

    public RelationshipResponse createRelationship(Long userId,
RelationshipRequest request) {
        User creator = userRepository.findById(userId)
            .orElseThrow(() -> new
NoSuchElementException("Пользователь не найден"));

        Person firstPerson =
personRepository.findById(request.getFirstPersonId())
            .orElseThrow(() -> new
NoSuchElementException("Первый человек не найден"));

        Person secondPerson =
personRepository.findById(request.getSecondPersonId())
            .orElseThrow(() -> new
NoSuchElementException("Второй человек не найден"));

        Tree tree = treeRepository.findById(request.getTreeId())
            .orElseThrow(() -> new
NoSuchElementException("Дерево не найдено"));

        RelationshipType relationshipType;
        try {
            relationshipType = request.getName();
        } catch (IllegalArgumentException e) {
            throw new IllegalArgumentException("Неверный тип
отношения: " + request.getName());
        }

        // Проверка существующей связи (в обе стороны)
        boolean exists = relationshipRepository

.existsByTreeIdAndFirstPersonIdAndSecondPersonIdOrTreeIdAndFirstPe
rsonIdAndSecondPersonId(

```

```

        request.getTreeId(), firstPerson.getId(),
secondPerson.getId(),
        request.getTreeId(), secondPerson.getId(),
firstPerson.getId()
    );

    if (exists) {
        throw new IllegalArgumentException("Связь между этими
людьми в этом дереве уже существует");
    }

    Relationship relationship = relationshipRepository.save(
        Relationship.builder()
            .creator(creator)
            .tree(tree)
            .firstPerson(firstPerson)
            .secondPerson(secondPerson)
            .name(relationshipType.name())
            .build()
    );

    return toDto(relationship);
}

public RelationshipResponse toDto(Relationship relationship) {
    return RelationshipResponse.builder()
        .id(relationship.getId())
        .treeId(relationship.getTree().getId())

.firstPersonId(relationship.getFirstPerson().getId())

.secondPersonId(relationship.getSecondPerson().getId())

.name(RelationshipType.valueOf(relationship.getName()))
        .build();
}

public List<RelativeResponse> getRelatives(Long personID) {
    return relationshipRepository.fetchRelatives(personID);
}
}

```

Листинг Б.5 – Файл TreeStructureService.java

```

@Service
public class TreeStructureService {

    @Autowired
    private TreeRepository treeRepository;

    @Autowired
    private RelationshipRepository relationshipRepository;
}

```

```

    public Map<String, Object> getTreeStructure(Long treeId) {
        // Получаем дерево по ID
        Tree tree = treeRepository.findById(treeId)
            .orElseThrow(() -> new
NoSuchElementException("Tree not found"));

        // Получаем все отношения в дереве
        List<Relationship> relationships =
relationshipRepository.findById(treeId);

        // Извлекаем всех уникальных людей, участвующих в
отношениях
        Set<Person> personsSet = relationships.stream()
            .flatMap(relationship ->
Stream.of(relationship.getFirstPerson(),
relationship.getSecondPerson()))
            .collect(Collectors.toSet());

        // Преобразуем людей в узлы
        List<Map<String, Object>> nodes = personsSet.stream()
            .map(person -> {
                String label =
Optional.ofNullable(person.getFirstName())
                    .orElse("")
                    .concat(" ")

                .concat(Optional.ofNullable(person.getSecondName()).orElse(""))
                    .concat(" ")

                .concat(Optional.ofNullable(person.getLastName()).orElse(""))
                    .trim();

                if (label.isEmpty() || label.equals(" ")) {
                    label = "Неизвестно";
                }

                Map<String, Object> node = new HashMap<>();
                node.put("id",
String.valueOf(person.getId()));
                node.put("data", Map.of(
                    "label", label,
                    "gender", person.getGender() != null ?
person.getGender().toString().toLowerCase() : "unknown"
                ));
                node.put("position", Map.of("x", 400, "y",
400)); // Позицию можно настроить
                node.put("type", "customNode");
                return node;
            })
            .collect(Collectors.toList());

        // Преобразуем отношения в связи

```

```

        List<Map<String, Object>> edges = relationships.stream()
            .map(relationship -> {
                Map<String, Object> edge = new HashMap<>();
                edge.put("id",
String.valueOf(relationship.getId()));
                edge.put("source",
String.valueOf(relationship.getFirstPerson().getId()));
                edge.put("target",
String.valueOf(relationship.getSecondPerson().getId()));

                // Устанавливаем label в зависимости от типа
отношения

                String label = "";
                if (relationship.getName() != null) {
                    if
(rerelationship.getName().equals(RelationshipType.PARTNER.name())) {
                        label = "супруги";
                    } else if
(rerelationship.getName().equals(RelationshipType.PARENT_CHILD.name(
))) {
                        label = "родитель-ребенок";
                    } else {
                        label = relationship.getName();
                    }
                }

                edge.put("label", label);
                edge.put("type", "custom");
                return edge;
            })
            .collect(Collectors.toList());

        // Возвращаем структуру с узлами и связями
        return Map.of("nodes", nodes, "edges", edges);
    }
}

```

Приложение В
(обязательное)

Фрагменты исходного кода клиентской части web-портала

Листинг В.1 – Файл axios.js

```
import axios from "axios";
import { toast } from "react-toastify";

// Базовый URL API
const API_URL = 'http://localhost:8080'; // Укажите ваш серверный URL

// Создаем инстанс axios
const instance = axios.create({
  baseURL: API_URL,
});

// Функция для обновления токена
const refreshToken = async () => {
  try {
    const storedTokens =
JSON.parse(localStorage.getItem("tokens"));
    if (!storedTokens?.refreshToken) {
      throw new Error("Refresh token не найден");
    }

    // Отправляем запрос на обновление токена
    const response = await axios.post(`${API_URL}/auth-
service/api/auth/refresh`, {
      refreshToken: storedTokens.refreshToken,
    });

    // Обновляем токены в localStorage
    const updatedTokens = {
      ...storedTokens,
      accessToken: response.data.accessToken,
      refreshToken: response.data.refreshToken,
    };
    localStorage.setItem("tokens",
JSON.stringify(updatedTokens));

    return response.data.accessToken; // Возвращаем новый
accessToken
  } catch (error) {
    toast.error("Не удалось обновить токен. Пожалуйста,
войдите заново.");
    console.error("Ошибка обновления токена:", error);
    throw error;
  }
};

// Интерцептор для запросов
instance.interceptors.request.use(
  (config) => {
    const url = config.url;
```

```

        // Искключаем запросы, где токен не нужен
        if (url && (url.includes("signin") ||
url.includes("signup") || url.includes("refresh"))) {
            return config;
        }

        // Извлекаем токен из localStorage
        const token =
JSON.parse(localStorage.getItem("tokens"))?.accessToken;

        // Добавляем токен в заголовок Authorization
        if (token) {
            config.headers.Authorization = `Bearer ${token}`;
        }

        return config;
    },
    (error) => {
        toast.error("Ошибка в запросе: проверьте настройки.");
        return Promise.reject(error);
    }
);

// Интерцептор для ответов
instance.interceptors.response.use(
    (response) => {
        // Если ответ содержит сообщение в формате { message:
"...", ... }, можно показать его
        if (response.data?.message && !response.config?.silent) {
            toast.success(response.data.message);
        }
        return response;
    }, // Успешный ответ
    async (error) => {
        const originalRequest = error.config;

        // Если ошибка 401 и запрос не был на refresh или
signin/signup
        if (error.response?.status === 401 &&
!originalRequest._retry &&
!originalRequest.url.includes('/refresh') &&
!originalRequest.url.includes('/signin') &&
!originalRequest.url.includes('/signup')) {

            originalRequest._retry = true; // Помечаем, что это
повторный запрос

            try {
                // Получаем новый accessToken
                const newAccessToken = await refreshToken();

```

```

        // Обновляем заголовок Authorization в
оригинальном запросе
        originalRequest.headers.Authorization = `Bearer
${newAccessToken}`;

        // Повторяем оригинальный запрос
        return instance(originalRequest);
    } catch (refreshError) {
        console.error("Не удалось обновить токен:",
refreshError);

        // Если refresh token истек, перенаправляем на
страницу входа
        localStorage.removeItem("tokens");
        toast.error("Сессия истекла. Пожалуйста, войдите
заново.");
        window.location.href = '/'; // Укажите URL вашей
страницы входа

        return Promise.reject(refreshError);
    }
}

// Обработка других ошибок
if (error.response) {
    const status = error.response.status;
    let message = '';

    // Пытаемся извлечь сообщение об ошибке из разных мест
    if (error.response.data?.message) {
        message = error.response.data.message;
    } else if (error.response.data?.error) {
        message = error.response.data.error;
    } else if (error.response.statusText) {
        message = `Ошибка ${status}:
${error.response.statusText}`;
    } else {
        message = `Произошла ошибка (${status})`;
    }

    // Показываем уведомление, если запрос не помечен как
"тихий"
    if (!originalRequest?.silent && message !== "Аватар не
установлен") {
        toast.error(message);
    }
} else if (error.request) {
    if (!originalRequest?.silent) {
        toast.error("Сервер не отвечает. Проверьте
соединение.");
    }
} else {

```

```

        if (!originalRequest?.silent) {
            toast.error("Произошла ошибка. Попробуйте еще
раз.");
        }
    }

    return Promise.reject(error);
}
);

export default instance;

```

Листинг В.2 – Файл PersonStore.js

```

import { makeAutoObservable } from "mobx";
import axios from "../utils/axios";
import treeStore from "../TreeStore";

class PersonStore {
    persons = []; // Состояние для хранения всех
    person = {};

    constructor() {
        makeAutoObservable(this); // Делаем все свойства и методы
        наблюдаемыми
    }

    // Метод для загрузки всех
    async loadMyPersons() {
        try {
            const response = await axios.get(`/person-
service/api/people/my`);
            // console.log("Данные родственников:",
response.data);
            this.persons = response.data;
        } catch (error) {
            console.error("Ошибка при загрузке родственников:",
error);
        }
    }

    async loadByTree(treeId) {
        try {
            const response = await axios.get(`/person-
service/api/people/byTree/${treeId}`);
            // console.log("Данные родственников:",
response.data);
            this.persons = response.data;
        } catch (error) {
            console.error("Ошибка при загрузке родственников:",
error);
        }
    }
}

```

```

    }

    // Метод для загрузки всех
    async loadPersonById(id) {
        try {
            const response = await axios.get(`/person-
service/api/people/${id}`);
            console.log("Данные человека:", response.data);
            this.person = response.data;
        } catch (error) {
            console.error("Ошибка при загрузке человека:", error);
        }
    }

    addPerson(person, treeId) {
        axios.post(`/person-service/api/people/with-
parents/${treeId}`, person)
            .then(() => {
                this.persons.push(person); // Локально обновляем
состояние
                // Уведомляем компонент, что данные обновились
                treeStore.loadTreeVisualization(treeId).then(() =>
{
                    console.log("Дерево обновлено после добавления
человека");
                });
            })
            .catch((err) => {
                console.error("Ошибка при добавлении человека:",
err);
            });
    }

    addChild(person, treeId, parentId) {
        axios.post(`/person-
service/api/people/create/child/${treeId}/${parentId}`, person)
            .then(() => {
                this.persons.push(person); // Локально обновляем
состояние
                // Уведомляем компонент, что данные обновились
                treeStore.loadTreeVisualization(treeId).then(() =>
{
                    console.log("Дерево обновлено после добавления
человека");
                });
            })
            .catch((err) => {
                console.error("Ошибка при добавлении человека:",
err);
            });
    }

```

```

    addParent(person, treeId, childId) {
      axios.post(`/person-
service/api/people/create/parent/${treeId}/${childId}`, person)
        .then(() => {
          this.persons.push(person); // Локально обновляем
состояние
          // Уведомляем компонент, что данные обновились
          treeStore.loadTreeVisualization(treeId).then(() =>
{
          console.log("Дерево обновлено после добавления
человека");
        });
      });
    }.catch((err) => {
      console.error("Ошибка при добавлении человека:",
err);
    });
  }

  addPartner(person, treeId, personId1) {
    axios.post(`/person-
service/api/people/create/partner/${treeId}/${personId1}`, person)
      .then(() => {
        this.persons.push(person); // Локально обновляем
состояние
        // Уведомляем компонент, что данные обновились
        treeStore.loadTreeVisualization(treeId).then(() =>
{
        console.log("Дерево обновлено после добавления
человека");
      });
    });
    }.catch((err) => {
      console.error("Ошибка при добавлении человека:",
err);
    });
  }

  // Метод для обновления
  updatePerson(updatedPerson) {
    axios.put(`/person-
service/api/people/${updatedPerson.id}`, updatedPerson)
      .then(() => {
        const index = this.persons.findIndex(item =>
item.id === updatedPerson.id);
        if (index !== -1) {
          this.persons[index] = updatedPerson; //
Обновляем в состоянии
        }
        if (this.person.id === updatedPerson.id) {
          this.person = updatedPerson;
        }
      });
  }

```

```

        }
    })
    .catch((err) => {
        console.error("Ошибка при обновлении человека:",
err);
    });
}

// Метод для удаления дерева
removePerson(id) {
    axios.delete(`/person-service/api/people/${id}`)
        .then(() => {
            this.persons = this.persons.filter(person =>
person.id !== id); // Удаляем из состояния
        })
        .catch((err) => {
            console.error("Ошибка при удалении человека:",
err);
        });
}

async searchPeople(criteria) {
    try {
        const { lastName, firstName, secondName, birthDate,
settlement, maidenName, deathDate, gender } = criteria;

        // Формируем строку параметров для запроса
        const params = new URLSearchParams();
        if (lastName) params.append('lastName', lastName);
        if (firstName) params.append('firstName', firstName);
        if (secondName) params.append('secondName',
secondName);
        if (birthDate) params.append('birthDate', birthDate);
        if (settlement) params.append('settlement',
settlement);
        if (maidenName) params.append('maidenName',
maidenName);
        if (deathDate) params.append('deathDate', deathDate);
        if (gender) params.append('gender', gender);

        const response = await axios.get(`/person-
service/api/people/search?${params.toString()}`);

        console.log("Результаты поиска:", response.data);
        this.persons = response.data; // Обновляем список
людей в состоянии
    } catch (error) {
        console.error("Ошибка при поиске людей:", error);
    }
}
}

```

```
const personStore = new PersonStore(); // Создаем экземпляр стор
export default personStore;
```

Листинг В.3 – Файл FamilyTreeFlow.jsx

```
const FamilyTreeFlow = observer(({ treeId }) => {
  const [nodes, setNodes] = useState([]);
  const [edges, setEdges] = useState([]);
  const [errorLoad, setErrorLoad] = useState("Ничего не
найдено");

  const [isOpenPersonModal, setIsOpenPersonModal] =
useState(false); // Состояние для открытия PersonModal
  const [isOpenProfileModal, setIsOpenProfileModal] =
useState(false); // Состояние для открытия PersonProfileModal
  const [isOpenAddRelativeModal, setIsOpenAddRelativeModal] =
useState(false); // State for AddRelativeModal
  const [isOpenCreateRelationshipModal,
setIsOpenCreateRelationshipModal] = useState(false); // Состояние
для открытия CreateRelationshipModal
  const [selectedPersonId, setSelectedPersonId] =
useState(null); // ID выбранного человека
  const [personModalData, setPersonModalData] = useState(null);
// Данные человека для редактирования

  const handleAdd = (item) => {
    personStore.addPerson({ ...item }, treeId);
    treeStore.loadTreeVisualization(treeId).then(() => {
      updateVisualization();
    });
  };

  const handleUpdate = (updatedItem) => {
    personStore.updatePerson(updatedItem);
    treeStore.loadTreeVisualization(treeId).then(() => {
      updateVisualization();
    });
  };

  const handleAddRelative = (relativeData) => {
    console.log(relativeData)
    if (relativeData.relative === "CHILD"){
      personStore.addChild(relativeData, treeId,
selectedPersonId)
    } else if (relativeData.relative === "PARENT"){
      personStore.addParent(relativeData, treeId,
selectedPersonId)
    } else if (relativeData.relative === "PARTNER"){
      personStore.addPartner(relativeData, treeId,
selectedPersonId)
    }
    treeStore.loadTreeVisualization(treeId).then(() => {
```

```

        updateVisualization();
    });
    // Implement logic for adding a relative
    // personStore.addRelative(selectedPersonId,
relativeData); // Assuming a method in personStore
    setIsOpenAddRelativeModal(false); // Close the modal after
adding

};

const handleDeletePerson = () => {
    personStore.removePerson(selectedPersonId)
    treeStore.loadTreeVisualization(treeId).then(() => {
        updateVisualization();
    });
};

const onNodesChange = (changes) => {
    setNodes((currentNodes) =>
        applyNodeChanges(changes, currentNodes).map((node) =>
({
            ...node,
            type: 'customNode',
        })))
    );
};

const handleNodeClick = (event, node) => {
    setSelectedPersonId(node.id);
    setIsOpenProfileModal(true);
};

const updateVisualization = () => {
    if (treeId) {
        treeStore.loadTreeById(treeId);
        treeStore.loadTreeVisualization(treeId).then(() => {
            if (treeStore.structure.nodes?.length > 0 &&
treeStore.structure.edges?.length > 0) {
                const dataNodes = treeStore.structure.nodes;
                const dataEdges = treeStore.structure.edges;

                setEdges(dataEdges);

                const newNodes = layoutNodes(dataNodes,
dataEdges);

                setNodes(newNodes);
            } else if (treeStore.structure.nodes?.length ===
0) {
                setErrorLoad("Вы ещё не добавили людей");
            }
        });
    }
};

```

```

    }
  };

  useEffect(() => {
    updateVisualization();
  }, [treeId]);

  return (
    <div className="w-full ">
      <div className="text-[var(--brown)] text-center flex
w-full justify-center pb-2 text-[18px] sm:text-[24px] md:text-
[30px] lg:text-[36px]"
        style={{ fontSize: 22 }}>
        <div className="text-[var(--brown)] text-center
flex w-full justify-center pb-2 text-[15px] sm:text-[18px]
md:text-[24px] lg:text-[26px]">
          Депо: {treeStore.selectedTree &&
treeStore.selectedTree.name}
        </div>
      </div>
      {treeStore.structure.nodes?.length > 0 ? (
        <ReactFlowProvider>
          <div className="w-full h-[calc(85vh-1.5rem)]
lg:h-[calc(85vh-3rem)]">
            <ReactFlow
              nodes={nodes}
              edges={edges}
              onNodesChange={onNodesChange}
              nodeTypes={nodeTypes}
              edgeTypes={edgeTypes}
              onNodeClick={handleNodeClick}
              fitView
            />
            <MiniMap />
            <Controls />
          </div>
        </ReactFlowProvider>
      ) : (
        <div style={{
          width: '100%',
          height: '80vh',
          display: 'flex',
          justifyContent: 'center',
          alignItems: 'center'
        }}>
          <div style={{
            marginTop: -150, display: 'flex',
            alignItems: 'center'
          }}>
            {errorLoad}
            <div style={{ marginLeft: 10 }}>
              <div className="tooltipWrapper">

```

```

        <Tooltip position="top"
content="Добавить" style={{ width: "80px", textAlign: "center" }}>
            <button className="button-
style button-table"
                                onClick={() =>
setIsOpenPersonModal(true)}>
                +
            </button>
        </Tooltip>
    </div>
</div>
</div>
</div>
) }

{/* Модальное окно для создания нового человека */}
<PersonModal
    isOpen={isOpenPersonModal}
    setIsOpen={setIsOpenPersonModal}
    onSubmit={handleAdd}
    initialData={personModalData}
    setInitialData={setPersonModalData}
    treeId={treeId}
/>

{/* Модальное окно для отображения профиля */}
{isOpenProfileModal && (
    <PersonProfileModal
        isOpen={isOpenProfileModal}
        setIsOpen={setIsOpenProfileModal}
        personId={selectedPersonId}
        setIsOpenPersonModal={setIsOpenPersonModal}
        setPersonModalData={setPersonModalData}

setIsOpenAddRelativeModal={setIsOpenAddRelativeModal}

setIsOpenCreateRelationshipModal={setIsOpenCreateRelationshipModal}
    }

    onDelete={handleDeletePerson}
/>
) }

{/* Модальное окно для добавления родственника */}
{isOpenAddRelativeModal && selectedPersonId && (
    <AddRelativeModal
        isOpen={isOpenAddRelativeModal}
        setIsOpen={setIsOpenAddRelativeModal}
        onSubmit={handleAddRelative}
        personId={selectedPersonId}
/>
) }

```

```

        { /* Модальное окно для редактирования информации о
человеке */ }
        { isOpenPersonModal && personModalData && (
            <PersonModal
                isOpen={isOpenPersonModal}
                setIsOpen={setIsOpenPersonModal}
                personData={personModalData}
                onSubmit={handleUpdate}
                initialData={personModalData}
                setInitialData={setPersonModalData}
                treeId={treeId}
            />
        ) }

        { /* Модальное окно для создания нового отношения */ }
        { isOpenCreateRelationshipModal && selectedPersonId &&
(
            <CreateRelationshipModal
                isOpen={isOpenCreateRelationshipModal}
                setIsOpen={setIsOpenCreateRelationshipModal}
                onSubmit={ (data) => console.log(data) }
                targetPersonId={selectedPersonId}
                targetPersonName={ 'человек' }
                treeId={treeId}
                peopleList={personStore.persons}
            />
        ) }
    </div>
);
});

export default FamilyTreeFlow;

```

Листинг В.4 – Файл PersonPage.jsx

```

const PersonPage = observer(() => {
    const {id} = useParams();
    const navigate = useNavigate();

    const [isOpen, setIsOpen] = useState(false);
    const [isOpenRelatives, setIsOpenRelatives] = useState(false);
    const [isCommentModalOpen, setIsCommentModalOpen] =
useState(false);
    const [dataItem, setDataItem] = useState(undefined);
    const [isPlaceModalOpen, setIsPlaceModalOpen] =
useState(false);

    useEffect(() => {
        personStore.loadPersonById(id);
        commentStore.loadCommentsByPerson(id);
        commentStore.getAverageGrade(id);
        avatarStore.fetchAvatar(id);
        treeStore.getSingleTreeIdByPersonId(id);
    });

```

```

    placeOfBirthStore.getPlaceOfBirth(id);
  }, [id]);

function handleUpdate(updatedItem) {
  personStore.updatePerson(updatedItem);
}

function handleEditClick(item) {
  setDataItem(item);
  setIsOpen(true);
}

function handleCommentSend(comment) {
  commentStore.addComment({...comment, personId: id});
}

const getGender = (data) => {
  switch (data) {
    case "FEMALE":
      return "Женский";
    case "MALE":
      return "Мужской";
    default:
      return "-";
  }
};

return <MainTemplate>
  <div className="container !p-[5px]">
    <div className="md:w-[900px]">
      <div
        className="text-[var(--brown)] text-center
flex w-full justify-center pb-2 text-[18px] sm:text-[24px]
md:text-[30px] lg:text-[36px]">
        <div>
          Профиль
        </div>
      </div>
      <div
        className="grid sm:grid-cols-2 justify-center
gap-2.5 w-full mt-[10px] border-[2px] border-dashed p-[10px] md:p-[30px]
text-[15px] md:text-[18px] lg:text-[22px]
text-[var(--brown)]">
        <div className="max-w-[370px]">
          <ImageUploader
savedImage={avatarStore.avatar?.path}/>
        </div>
        <div
          className="grid h-fit grid-cols-
[120px_minmax(100px,_1fr)] md:grid-cols-[170px_1fr] gap-2.5 w-
full">

```

```

        <div className="font-light">Фамилия:</div>
        <div className="font-
normal">{personStore.person.lastName || "-"}</div>

        <div className="font-light">Имя:</div>
        <div className="font-
normal">{personStore.person.firstName || "-"}</div>

        <div className="font-
light">Отчество:</div>
        <div className="font-
normal">{personStore.person.secondName || "-"}</div>

        <div className="font-light">Пол:</div>
        <div className="font-
normal">{getGender(personStore.person.gender) }</div>

        {personStore.person.gender === "FEMALE" &&
(
    <>
        <div className="font-
light">Девичья фамилия:</div>
        <div className="font-
normal">{personStore.person.maidenName || "-"}</div>
    </>
)}

        <div className="font-light">Дата
рождения:</div>
        <div className="font-
normal">{personStore.person.dateOfBirth || "-"}</div>

        <div className="font-light">Дата
смерти:</div>
        <div className="font-
normal">{personStore.person.dateOfDeath || "-"}</div>
    </div>
    <div className="mt-2">
        <MapComponent
location={placeOfBirthStore.currentPlace}/>
        {authStore.user.userId ===
personStore.person.creatorId &&
            <div className="!flex gap-2 md:gap-4">
                <button
                    className="button-style !bg-[var(--cherry-lite)] hover:!bg-[var(--cherry)] !m-0 !w-full"
                    onClick={() =>
setIsPlaceModalOpen(true) }>
                    {placeOfBirthStore.currentPlace ?
"Изменить место рождения" : "Добавить место рождения"}
                </button>
            </div>
        }
    </div>

```

```

                                {placeOfBirthStore.currentPlace &&
<button
                                className="button-style !bg-[var(--cherry-lite)] hover:!bg-[var(--cherry)] !m-0 !w-full"
                                onClick={() =>
placeOfBirthStore.removePlaceOfBirth(id)}>
                                Удалить место рождения
                                </button>
                                </div>
                                }
                                </div>
                                <div className="border-t border-dashed border-[var(--brown)] mt-[30px]"></div>
                                {authStore.user.userId ===
personStore.person.creatorId &&
                                <div className="grid grid-cols-[1fr_1fr] gap-4 lg:gap-6 w-full mt-[10px]">
                                <button
                                className="button-style !bg-[var(--cherry-lite)] hover:!bg-[var(--cherry)] !m-0 !md:mx-[20px]"
                                onClick={() =>
setIsOpenRelatives(true)}>
                                Посмотреть родственников
                                </button>
                                <button
                                className="button-style !bg-[var(--cherry-lite)] hover:!bg-[var(--cherry)] !m-0 !md:mx-[20px]"
                                onClick={() =>
handleEditClick(personStore.person)}>
                                Редактировать
                                </button>
                                </div>
                                }
                                <div className="grid grid-cols-[1fr_1fr] gap-4 lg:gap-6 w-full mt-[10px] mb-[20px]">
                                <div className="grid grid-cols-[1fr_1fr] gap-2 lg:gap-4 w-full">
                                <button
                                className="button-style !bg-[var(--cherry-lite)] hover:!bg-[var(--cherry)] !m-0 !w-full !px-[5px]"
                                onClick={() =>
navigate(`/tree/${treeStore.singleTreeId}`)}>
                                В дерево
                                </button>
                                <button
                                className="button-style !bg-[var(--cherry-lite)] hover:!bg-[var(--cherry)] !m-0 !w-full !px-[5px]"
                                onClick={() =>
navigate(`/life/${personStore.person.id}`)}>
                                К истории жизни
                                </button>
                                </div>
                                </div>

```

```

        <button className="button-style !bg-[var(--cherry-lite)] hover:!bg-[var(--cherry)] !m-0"
            onClick={() =>
setIsCommentModalOpen(true)}>
            Оставить комментарий
        </button>
    </div>
    <div className="border-t border-dashed border-[var(--brown)] mt-[30px]"></div>
    <div className="mb-[20px]">
        <Comments isMy={false}
data={commentStore.comments}
avaregeGrade={commentStore.averageGrade}/>
    </div>
</div>
<PersonModal
    isOpen={isOpen}
    setIsOpen={setIsOpen}
    initialData={dataItem}
    setInitialData={setDataItem}
    onSubmit={handleUpdate}
/>
<RelativesModal
    isOpen={isOpenRelatives}
    setIsOpen={setIsOpenRelatives}
    id={id}
/>
<AddCommentModal
    isOpen={isCommentModalOpen}
    setIsOpen={setIsCommentModalOpen}
    onSubmit={handleCommentSend}
/>
<PlaceOfBirthModal
    isOpen={isPlaceModalOpen}
    setIsOpen={setIsPlaceModalOpen}
    personId={id}
/>
</div>
</MainTemplate>

));

export default PersonPage;

```

Приложение Г
(рекомендуемое)
Диаграммы вариантов использования



Рисунок Г.2 – Диаграмма вариантов использования подсистемы управления родственными связями

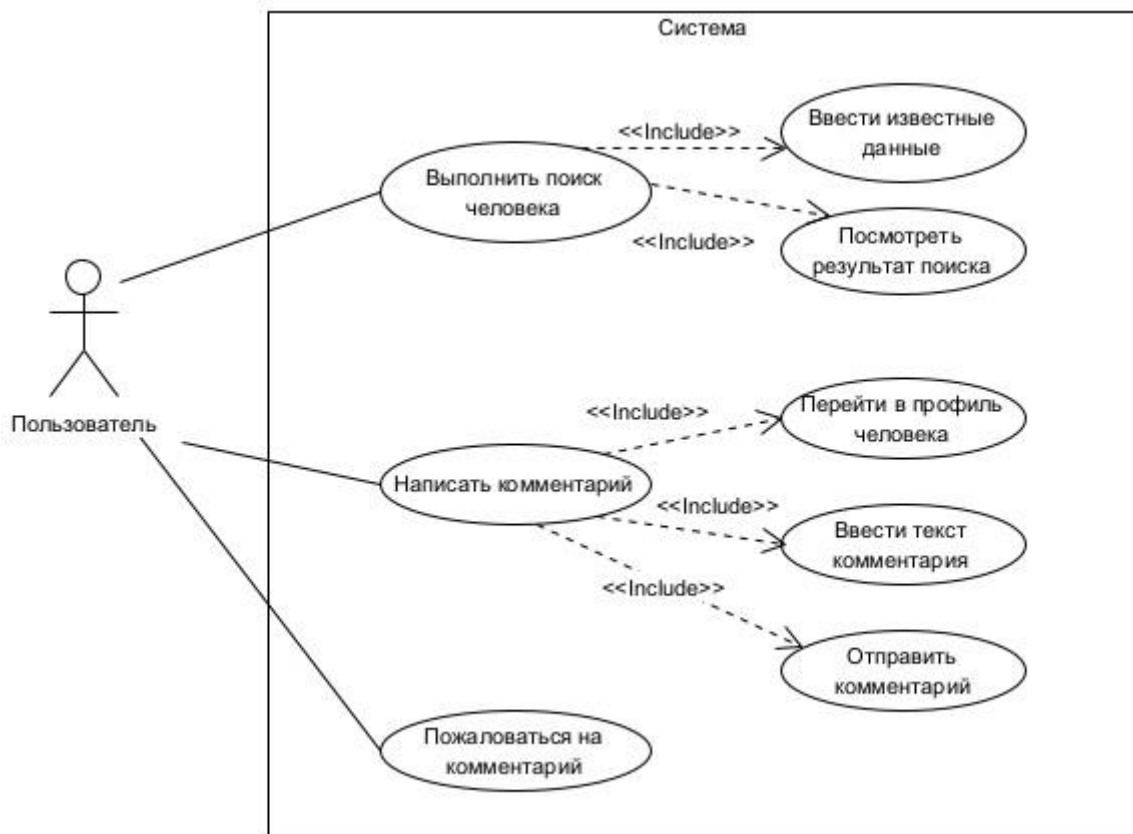


Рисунок Г.3 – Диаграмма вариантов использования подсистемы поиска персон и оценки достоверности данных

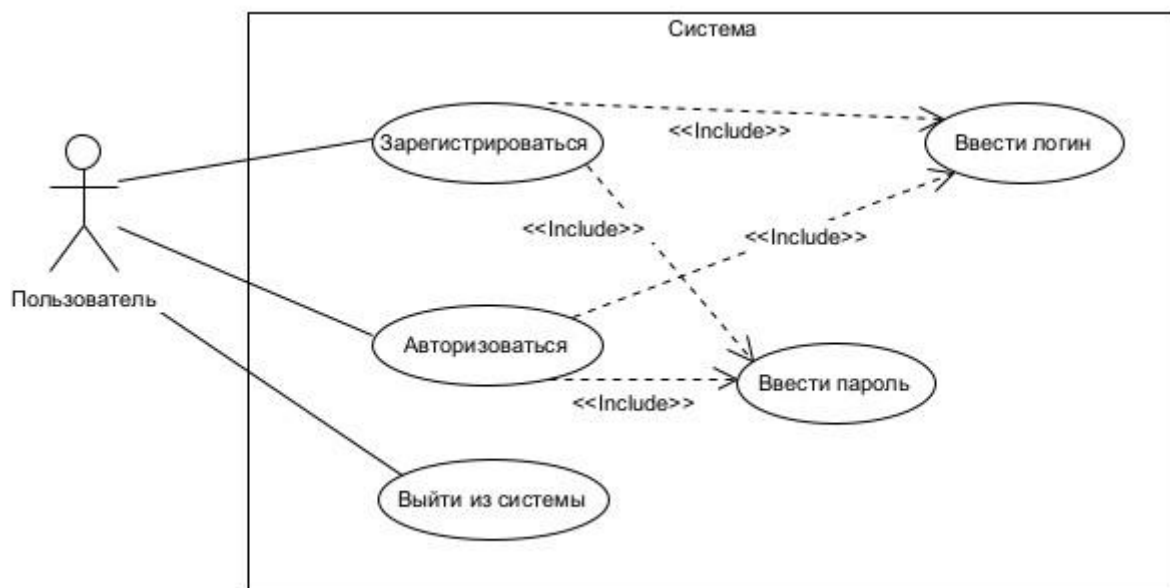


Рисунок Г.4 – Диаграмма вариантов использования подсистемы авторизации

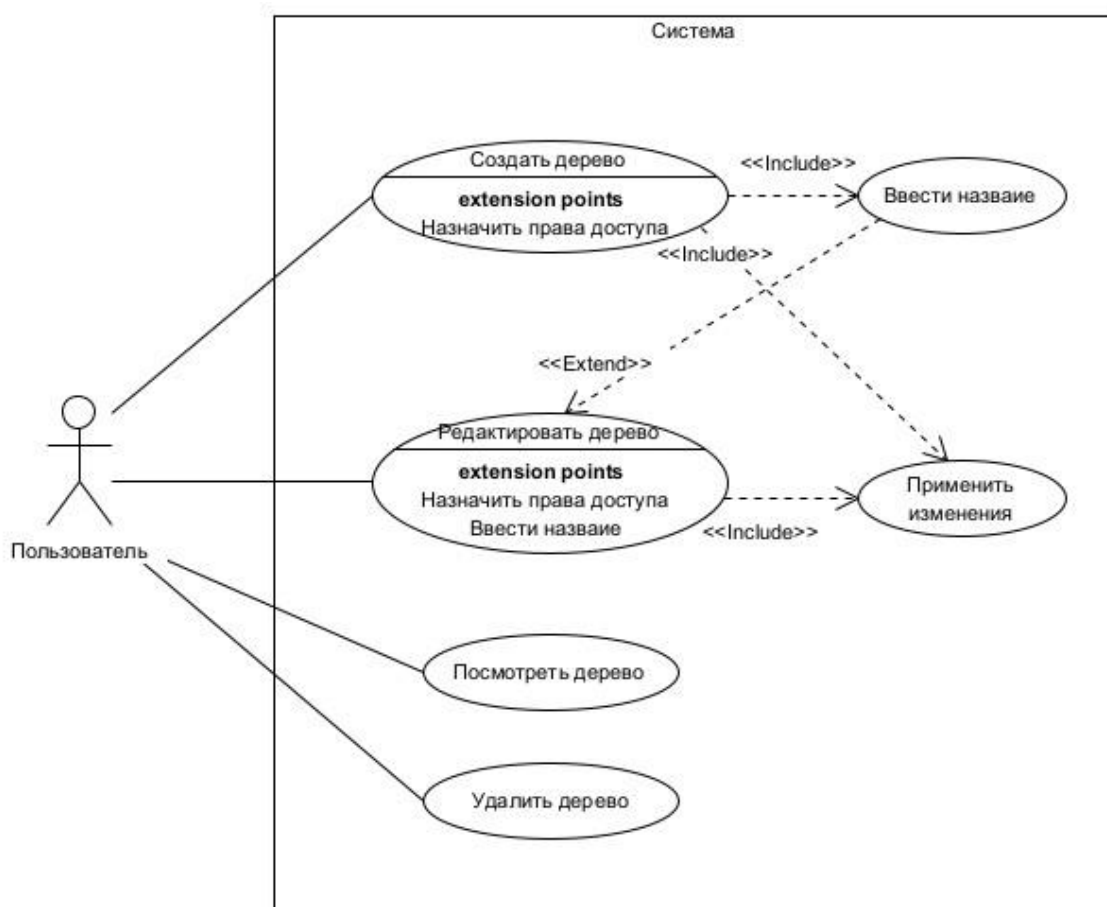


Рисунок Г.5 – Диаграмма вариантов использования подсистемы управления деревьями

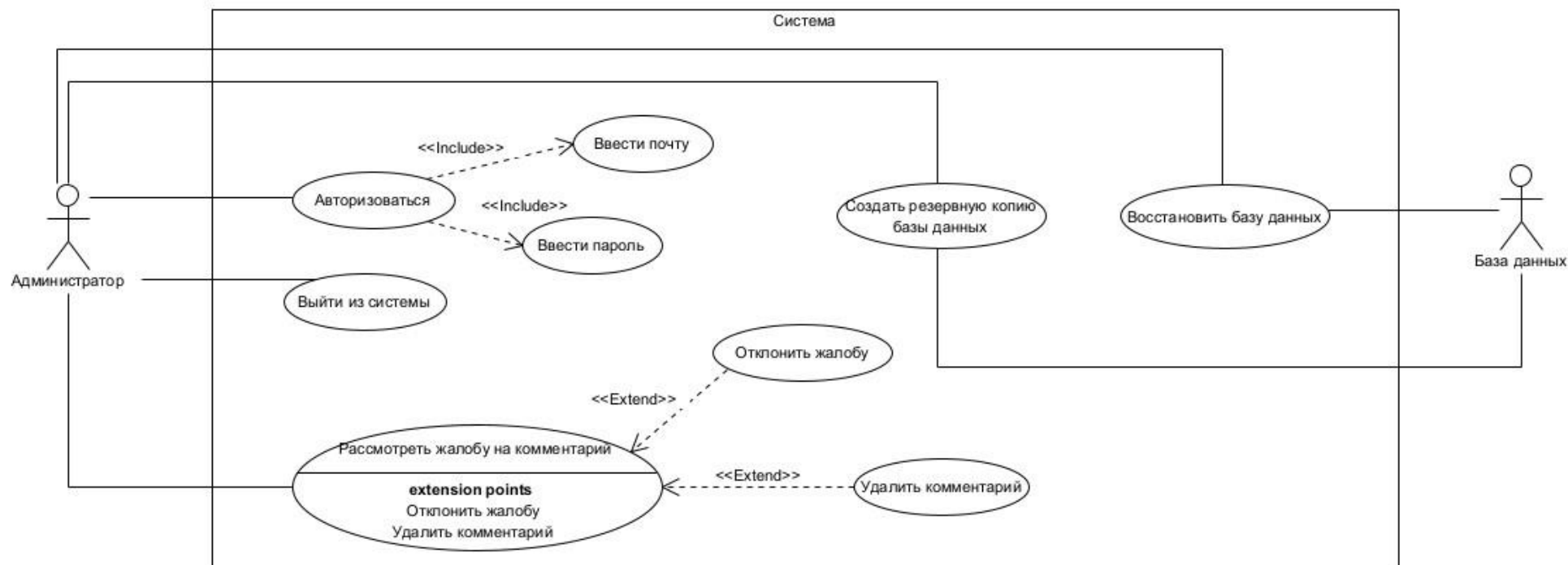


Рисунок Г.6 – Диаграмма вариантов использования подсистемы администрирования

Приложение Д
(рекомендуемое)

Табличное описание атрибутов сущностей системы

Таблица Д.1 – Атрибуты сущности «Пользователь»

№ п/п	Наименование	Описание	Тип данных	Размер	Область допустимых значений	Возможность значения NULL	Пример
1	ID	Уникальный идентификатор пользователя	Числовой	10	Уникальные значения	нет	1
2	Username	Логин пользователя	Строковый	255	A-я, A-z, 0 – 9	нет	User1
3	Role	Роль пользователя	Строковый	10	USER, ADMIN	нет	USER
4	Date_of_registration	Дата регистрации пользователя	Дата/время		Временной диапазон	нет	202 – 12 – 01
5	Password	Пароль пользователя	Строковый	255	-	нет	password

Таблица Д.2 – Атрибуты сущности «Человек»

№ п/п	Наименование	Описание	Тип данных	Размер	Область допустимых значений	Возможность значения NULL	Пример
1	ID	Уникальный идентификатор человека	Числовой	10	Уникальные значения	нет	1
2	Last_name	Фамилия человека	Строковый	32	A-я, A-z	да	Иванов
3	First_name	Имя человека	Строковый	32	A-я, A-z	да	Иван
4	Second_name	Отчество человека	Строковый	32	A-я, A-z	да	Иванович
5	Maiden_name	Девичья фамилия (если применимо)	Строковый	32	A-я, A-z	да	Петрова
6	Gender	Пол человека	Строковый	10	MALE, FEMALE	нет	MALE
7	Date_of_birth	Дата рождения	Дата		Временной диапазон	да	2000 – 01 – 01
8	Date_of_death	Дата смерти	Дата		Временной диапазон	Да	2070 – 12 – 31
9	ID_creator	Уникальный идентификатор создателя	Числовой		Уникальные значения	нет	1
10	ID_place_of_birth	Уникальный идентификатор места рождения	Числовой		Уникальные значения	нет	1

Продолжение таблицы Д.2

№ п/п	Наименование	Описание	Тип данных	Размер	Область допустимых значений	Возможность значения NULL	Пример
11	ID_avatar	Уникальный идентификатор аватара	Числовой		Уникальные значения	нет	1

Таблица Д.3 – Атрибуты сущности «Родственное отношение»

№ п/п	Наименование	Описание	Тип данных	Размер	Область допустимых значений	Возможность значения NULL	Пример
1	ID	Уникальный идентификатор связи	Числовой	10	Уникальные значения	нет	1
2	Name	Тип родственного отношения	Строковый	255	A-я, A-z	нет	отец
3	Date_created	Дата создания родственной связи	Дата/время	–	Временной диапазон	нет	2024 – 12 – 01
4	Date_updated	Дата последнего изменения связи	Дата/время	–	Временной диапазон	да	2024 – 12 – 05
5	ID_creator	Уникальный идентификатор создателя	Числовой	10	Уникальные значения	нет	1
6	ID_tree	Уникальный идентификатор дерева	Числовой	10	Уникальные значения	нет	1
7	ID_first_person	Уникальный идентификатор первого человека	Числовой	10	Уникальные значения	нет	1
8	ID_second_person	Уникальный идентификатор второго человека	Числовой	10	Уникальные значения	нет	1

Таблица Д.4 – Атрибуты сущности «Дерево»

№ п/п	Наименование	Описание	Тип данных	Размер	Область допустимых значений	Возможность значения NULL	Пример
1	ID	Уникальный идентификатор дерева	Числовой	10	Уникальные значения	нет	1
2	Name	Название дерева	Строковый	255	A-я, A-z	нет	Семья Ивановых
3	Date_created	Дата создания дерева	Дата/время	–	Временной диапазон	нет	2024 – 01 – 01
4	Date_updated	Дата последнего изменения дерева	Дата/время	–	Временной диапазон	да	2024 – 12 – 01
5	ID_owner	Уникальный идентификатор владельца	Числовой	10	Уникальные значения	нет	1

Таблица Д.5 – Атрибуты сущности «История жизни»

№ п/п	Наименование	Описание	Тип данных	Размер	Область допустимых значений	Возможность значения NULL	Пример
1	ID	Уникальный идентификатор истории	Числовой	10	Уникальные значения	нет	1
2	Date_start	Начало временного периода	Дата/время	–	Временной диапазон	нет	1990 – 01 – 01
3	Date_end	Конец временного периода	Дата/время	–	Временной диапазон	да	2000 – 12 – 31
4	Text	Описание события	Строковый	1000	Любой текст	нет	Родился в Москве
5	Date_created	Дата добавления записи	Дата/время	–	Временной диапазон	нет	2024 – 12 – 01
6	Date_updated	Дата последнего изменения	Дата/время	–	Временной диапазон	да	2024 – 12 – 15
7	ID_user	Уникальный идентификатор пользователя	Числовой	10	Уникальные значения	Нет	1

Продолжение таблицы Д.5

№ п/п	Наименование	Описание	Тип данных	Размер	Область допустимых значений	Возможность значения NULL	Пример
8	ID_person	Уникальный идентификатор человека	Числовой	10	Уникальные значения	нет	1

Таблица Д.6 – Атрибуты сущности «Комментарий про человека»

№ п/п	Наименование	Описание	Тип данных	Размер	Область допустимых значений	Возможность значения NULL	Пример
1	ID	Уникальный идентификатор комментария	Числовой	10	Уникальные значения	нет	1
2	Text	Текст комментария	Строковый	1000	Любой текст	нет	Был замечательным учёным
3	Date_added	Дата создания комментария	Дата/время	–	Временной диапазон	нет	2024 – 12 – 01
4	Grade	Оценка достоверности данных про человека	Числовой	–	Любое значение	да	4,8
5	ID_user	Уникальный идентификатор пользователя	Числовой	10	Уникальные значения	нет	1
6	ID_person	Уникальный идентификатор человека	Числовой	10	Уникальные значения	нет	1
7	Status	Статус комментария	Строковый	–	Любое значение	нет	CREATED

Таблица Д.7 – Атрибуты сущности «Место рождения»

№ п/п	Наименование	Описание	Тип данных	Размер	Область допустимых значений	Возможность значения NULL	Пример
1	ID	Идентификатор места рождения	Числовой	10	Уникальные значения	нет	1
2	Country	Название страны	Строковый	255	A-я, A-z	да	Россия

Продолжение таблицы Д.7

3	Region	Название региона	Строковый	255	А-я, А-z	да	Московская область
4	District	Название района	Строковый	255	А-я, А-z	да	Одинцовский
5	Name_settlement	Название города или деревни	Строковый	255	А-я, А-z	да	Одинцово
6	Coordinates	Географические координаты (широта, долгота)	Строковый	255	0 – 9 . ,	да	55.8417, 37.3115

Таблица Д.8 – Атрибуты сущности «Фото»

№ п/п	Наименование	Описание	Тип данных	Размер	Область допустимых значений	Возможность значения NULL	Пример
1	ID	Идентификатор фото	Числовой	10	Уникальные значения	нет	1
2	Name	Название фото	Строковый	255	А-я, А-z, 09	нет	Семейное фото
3	Path	Путь к файлу фото	Строковый	255	Валидный путь к файлу	нет	/images/photo1.jpg
4	Date_created	Дата создания фото	Дата	-	yyyy-MM-dd	да	2024 – 12 – 27
5	ID_owner	Уникальный идентификатор владельца	Числовой	10	Уникальные значения	нет	1
6	ID_life_history	Уникальный идентификатор истории жизни	Числовой	10	Уникальные значения	нет	1

Таблица Д.9 – Атрибуты сущности «Видео»

№ п/п	Наименование	Описание	Тип данных	Размер	Область допустимых значений	Возможность значения NULL	Пример
1	ID	Идентификатор видео	Числовой	10	Уникальные значения	нет	1
2	Name	Название видео	Строковый	255	А-я, А-z, 0 – 9	нет	Интервью с бабушкой

Продолжение таблицы Д.9

№ п/п	Наименование	Описание	Тип данных	Размер	Область допустимых значений	Возможность значения NULL	Пример
3	Path	Путь к файлу видео	Строковый	255	Валидный путь к файлу	нет	/videos/video1.mp4
4	Date_created	Дата создания видео	Дата	-	уууу-ММ-дд	да	2024 – 12 – 27
5	ID_owner	Уникальный идентификатор владельца	Числовой	10	Уникальные значения	нет	1
6	ID_life_history	Уникальный идентификатор истории жизни	Числовой	10	Уникальные значения	нет	1

Приложение Е
(рекомендуемое)
Пользовательский интерфейс приложения

The login form is titled "Вход" (Login) in a bold, dark font. It features two input fields: "Логин" (Login) and "Пароль" (Password). The password field includes a small eye icon for toggling visibility. Below the fields is a prominent dark red button labeled "ВОЙТИ" (Login). At the bottom, there is a link labeled "Регистрация" (Registration) in a smaller, underlined font.

Рисунок Е.1 – Форма входа в приложение

The registration form is titled "Регистрация" (Registration) in a bold, dark font. It contains three input fields: "Логин" (Login), "Пароль" (Password), and "Повторите пароль" (Repeat password). Both password fields have an eye icon for visibility control. A dark red button labeled "ЗАРЕГИСТРИРОВАТЬСЯ" (Register) is positioned below the fields. At the bottom, there is a link labeled "Вход" (Login) in a smaller, underlined font.

Рисунок Е.2 – Форма регистрации в приложении

The navigation screen consists of three stacked, dark red rectangular buttons with white text. The top button is labeled "МОИ ДЕРЕВЬЯ" (My Trees), the middle button is labeled "РОДСТВЕННИКИ" (Relatives), and the bottom button is labeled "ПОИСК ЛЮДЕЙ" (Search People).

Рисунок Е.3 – Страница навигации пользователя

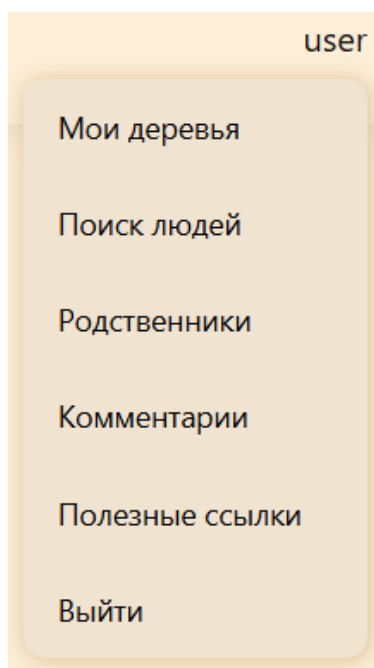


Рисунок Е.4 – Меню пользователя



Рисунок Е.5 – Страница таблицы деревьев

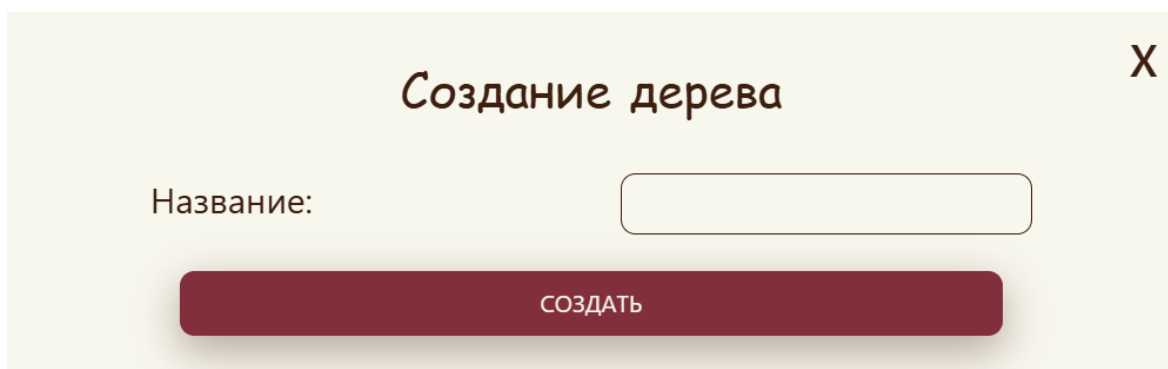


Рисунок Е.6 – Окно создания дерева

Поиск людей

Фамилия:

Дата рождения:

Имя:

Дата смерти:

Отчество:

Пол:

Девичья фамилия:

НАЙТИ

СБРОСИТЬ

#	ФИО	
1	Малькова Анастасия Сергеевна	👁

Рисунок Е.7 – Страница поиска родственников

Родственники

#	ФИО	
1	Малькова Анастасия Сергеевна	👁
2	Неизвестно	👁

Рисунок Е.8 – Таблица родственников

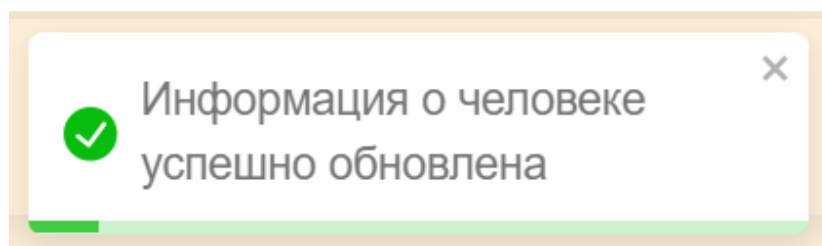


Рисунок Е.9 – Пример уведомления системы



Рисунок Е.10 – Страница комментариев, написанных к созданным пользователем людям

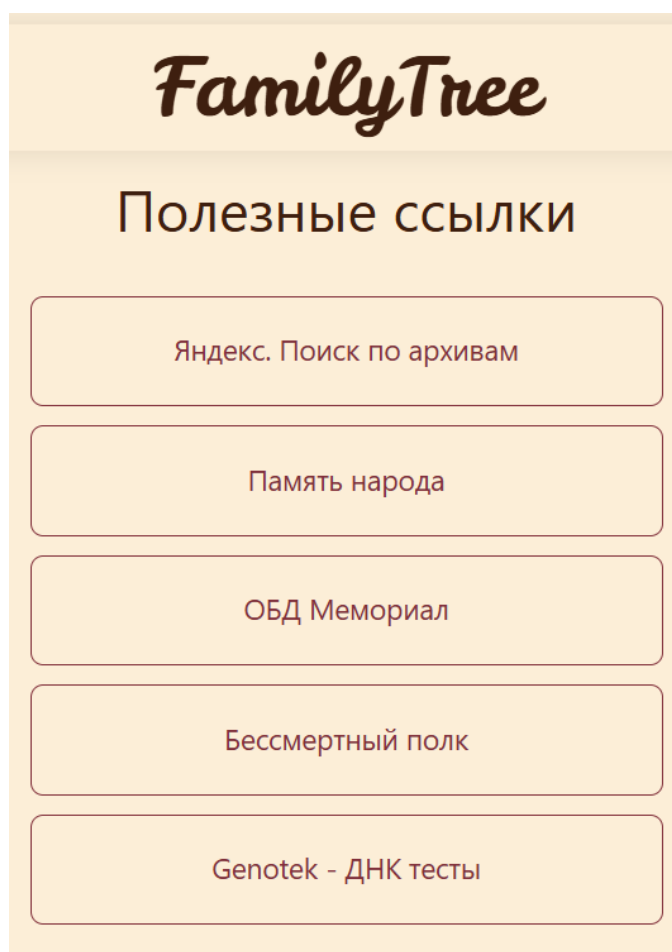


Рисунок Е.11 – Страница полезных ссылок

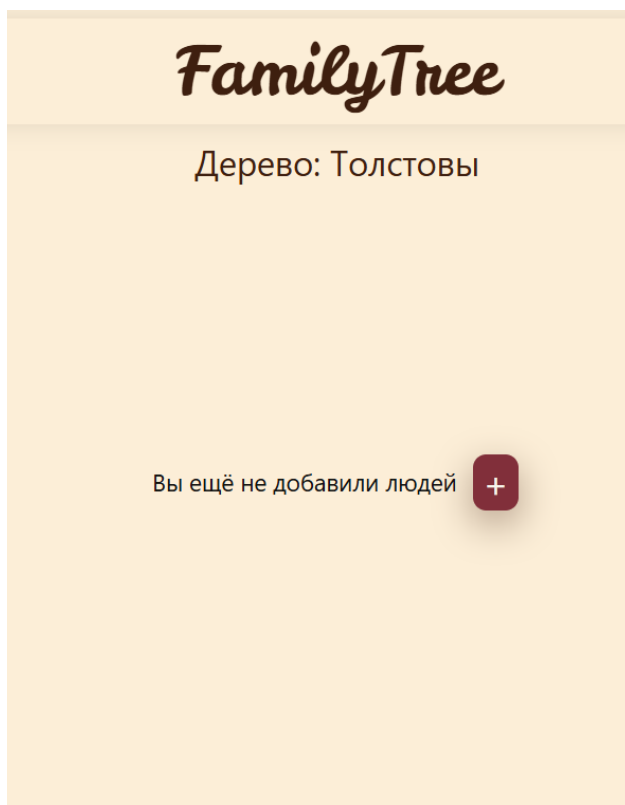


Рисунок Е.12 – Страница дерева, в которое ещё не добавили людей

The image displays a 'Создание' (Creation) window for adding a new person to the family tree. The window has a title bar with the word 'Создание' and a close button 'X'. The form consists of several labeled input fields: 'Фамилия:' (Family) with the value 'Толстой', 'Имя:' (Name) with 'Лев', 'Отчество:' (Patronymic) with 'Николаевич', 'Пол:' (Gender) with a dropdown menu showing 'Мужской' (Male), 'Девичья фамилия:' (Maiden name) which is empty, 'Дата рождения:' (Date of birth) with '28.08.1828' and a calendar icon, and 'Дата смерти:' (Date of death) with '07.11.1910' and a calendar icon. At the bottom of the form is a large red button labeled 'СОХРАНИТЬ' (Save).

Рисунок Е.13 – Окно создания человека

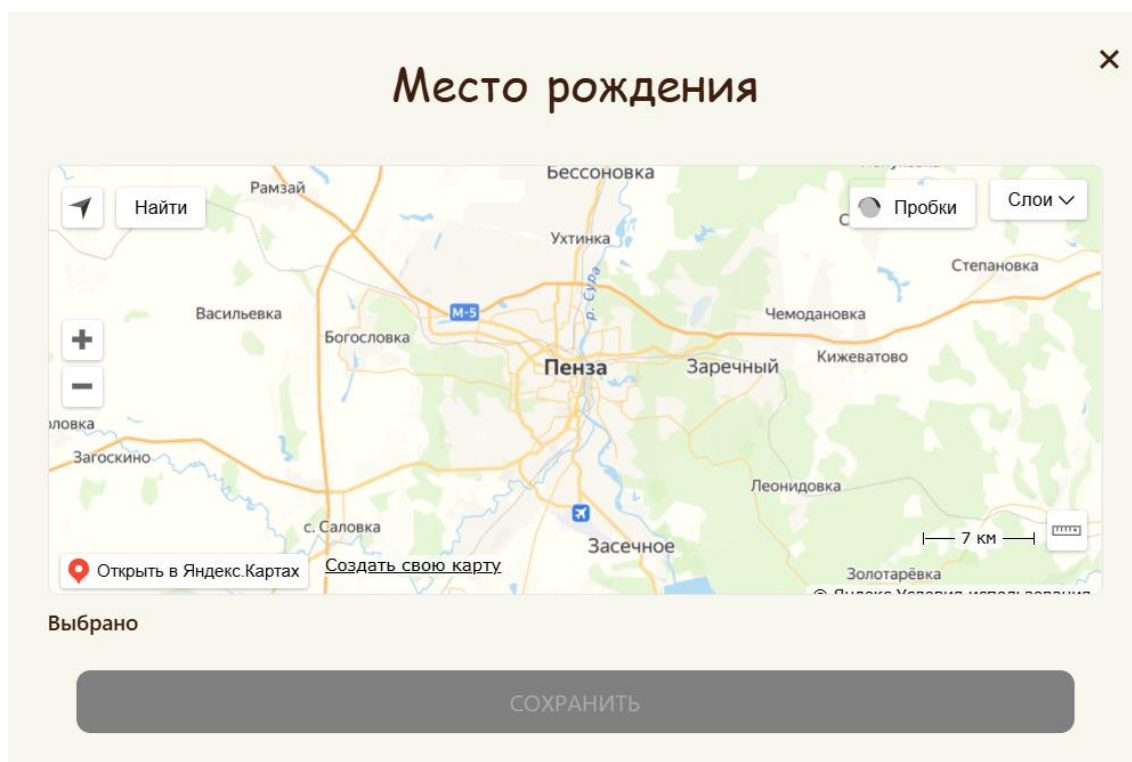


Рисунок Е.14 – Окно добавления места жительства

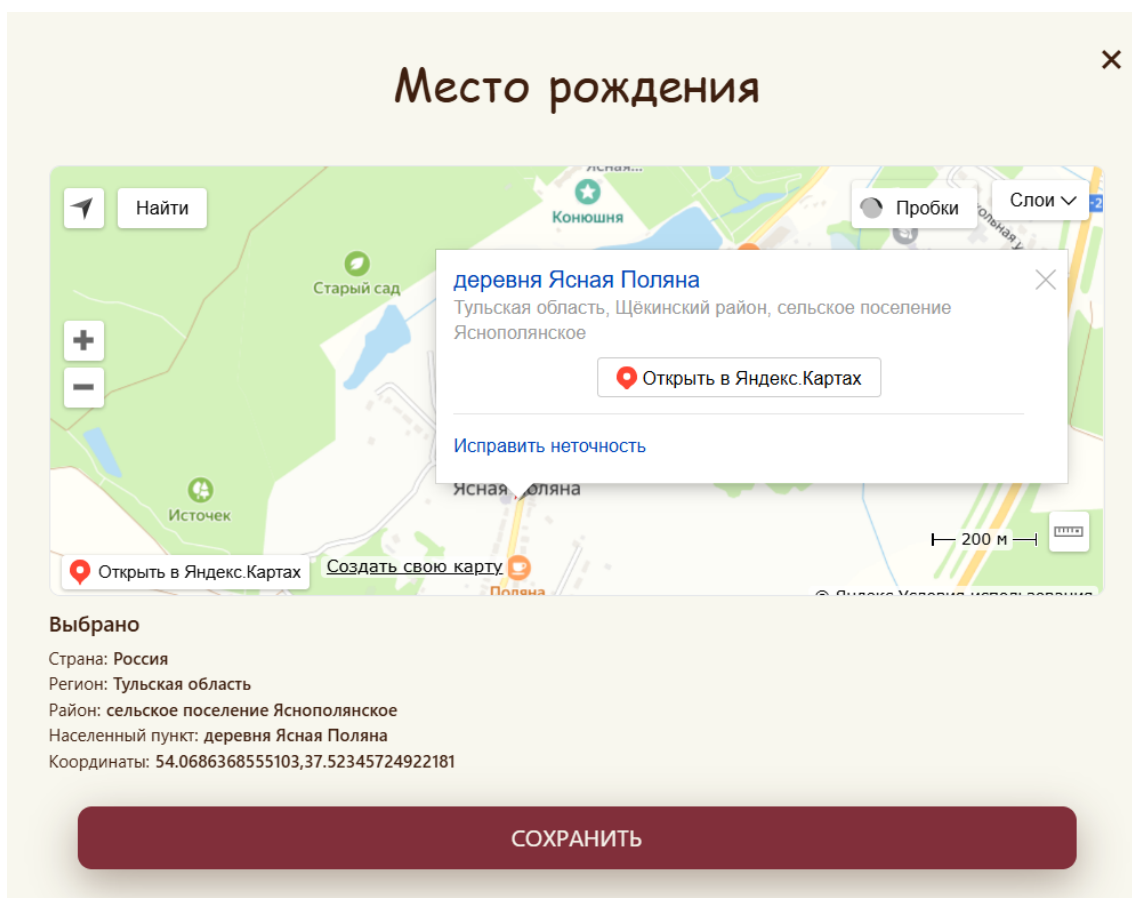
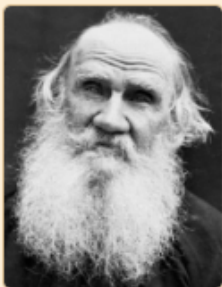


Рисунок Е.15 – Окно с выбранным местом жительства

Профиль

Фото



Выберите изображение (JPG/PNG, до 10 МБ)

Выберите файл толстой.jpg

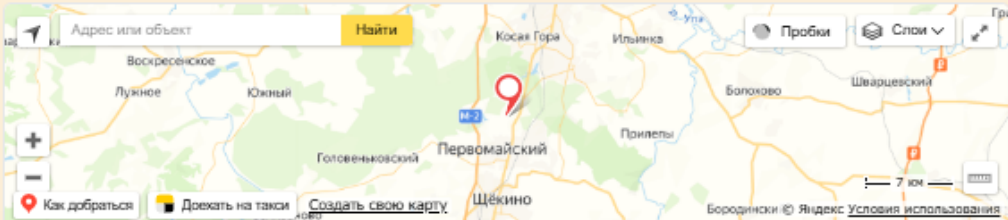
Сохранить

Сбросить

Фамилия:	Толстой
Имя:	Лев
Отчество:	Николаевич
Пол:	Мужской
Дата рождения:	1828-08-28
Дата смерти:	1910-11-07

Место рождения

Страна: Россия
 Регион: Тульская область
 Район: сельское поселение Яснополянское
 Населенный пункт: деревня Ясная Поляна
 Координаты: 54.068636855103,37.52345724922181



ИЗМЕНИТЬ МЕСТО РОЖДЕНИЯ

УДАЛИТЬ МЕСТО РОЖДЕНИЯ

ПОСМОТРЕТЬ РОДСТВЕННИКОВ

РЕДАКТИРОВАТЬ

В ДЕРЕВО

К ИСТОРИИ ЖИЗНИ

ОСТАВИТЬ КОММЕНТАРИЙ

Рисунок Е.16 – Страница профиля человека

Комментарии

Средняя оценка достоверности 5.0

user
★ 5

Всё верно!

3 июня 2025 г. в 11:23

[Пожаловаться](#)

Рисунок Е.17 – Раздел комментариев на странице профиля человека

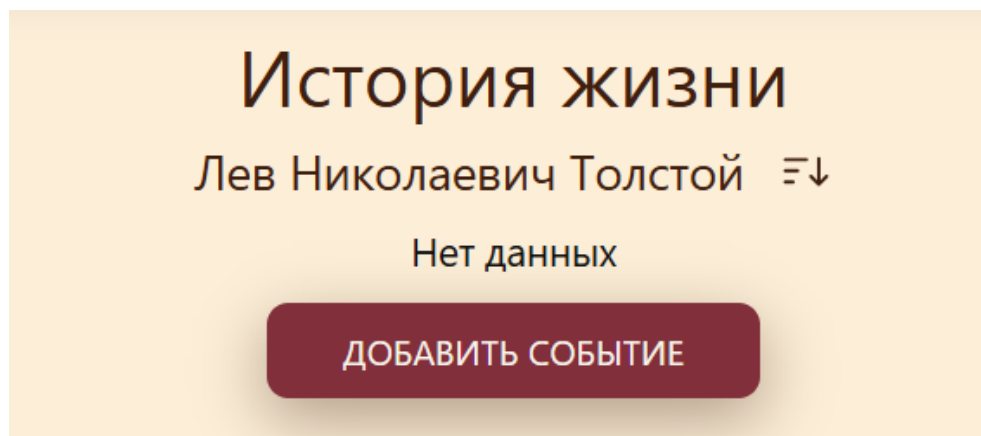


Рисунок Е.18 – Страница истории жизни, в которое ещё ничего не добавлено

Добавление жизненного события

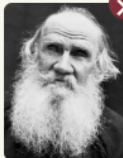
Дата начала01.01.1854

Дата окончания01.01.1913

ОписаниеДом, где родился Л.Н. Толстой. В 1854 году был продан и перевезен в село Долгое. Разобран в 1913 году

Добавить фото

Выбрать файлы толстой.jpg



Добавить видео

Выбрать файлы Файл не выбран

СОЗДАТЬ

Рисунок Е.19 – Окно добавления истории жизни

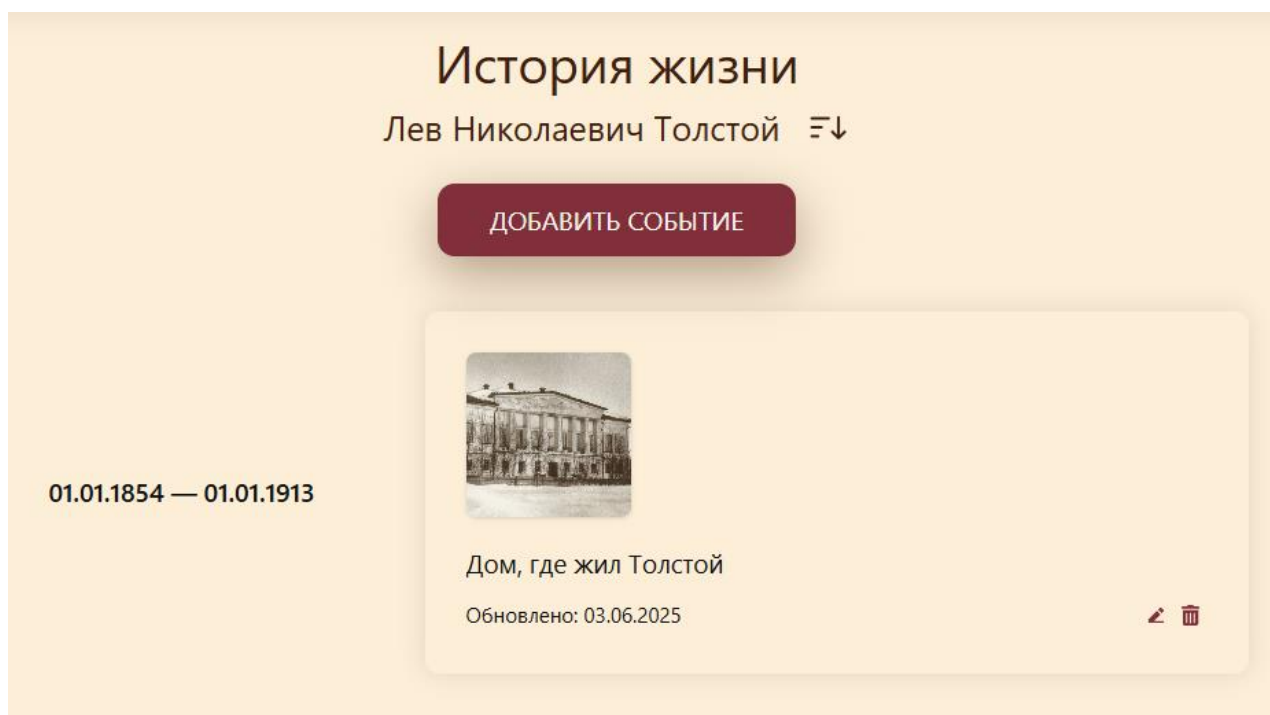


Рисунок Е.20 – Страница истории жизни человека

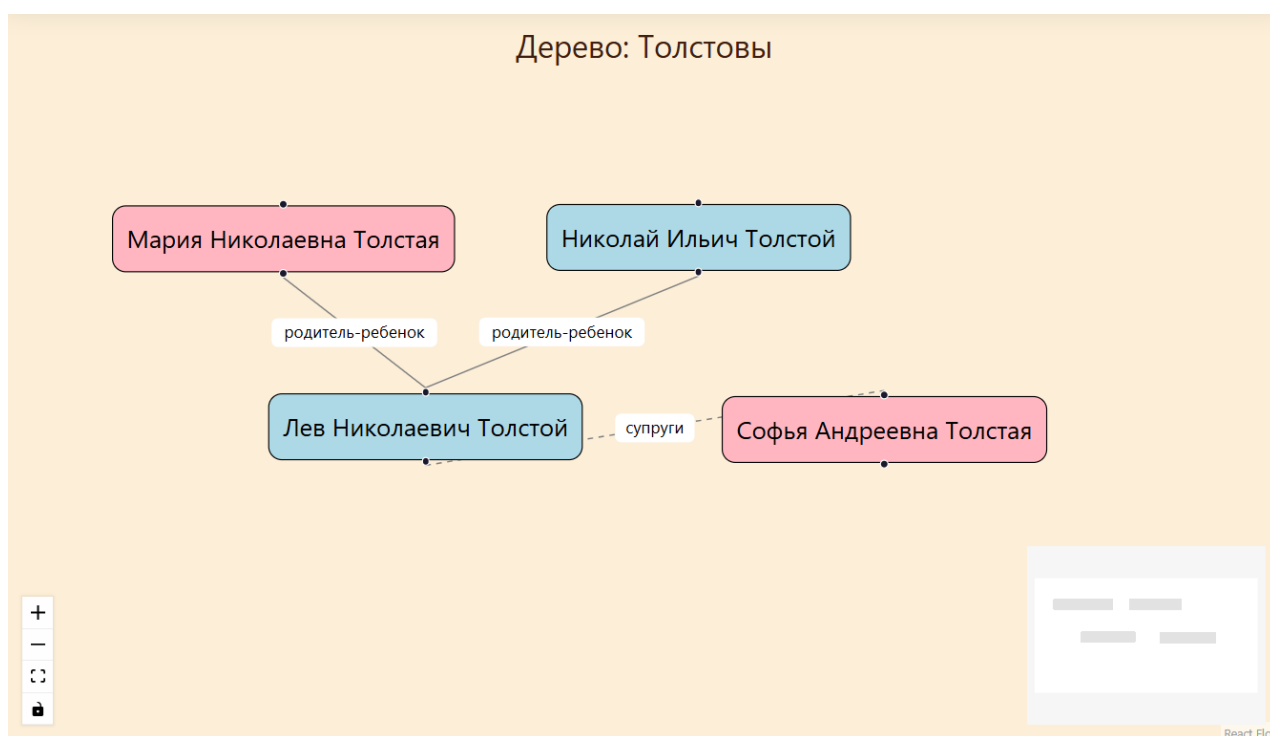


Рисунок Е.21 – Страница дерева семьи

Просмотр X

Фамилия:	Толстой
Имя:	Лев
Отчество:	Николаевич
Пол:	Мужской
Дата рождения:	1828-08-28
Дата смерти:	1910-11-07

В ПРОФИЛЬ РЕДАКТИРОВАТЬ

ДОБАВИТЬ НОВОГО РОДСТВЕННОГО ДОБАВИТЬ ОТНОШЕНИЕ УДАЛИТЬ

Рисунок Е.22 – Окно просмотра данных человека в дереве

Ближайшие родственники X

Толстая Мария Николаевна:	Мать
Толстой Николай Ильич:	Отец
Толстая Софья Андреевна:	Жена

Рисунок Е.23 – Окно просмотра ближайших родственников человека

Добавить комментарий X

Ваш комментарий:

Оценка: ☆ ☆ ☆ ☆ ☆

ОТПРАВИТЬ

Рисунок Е.24 – Окно добавления комментария

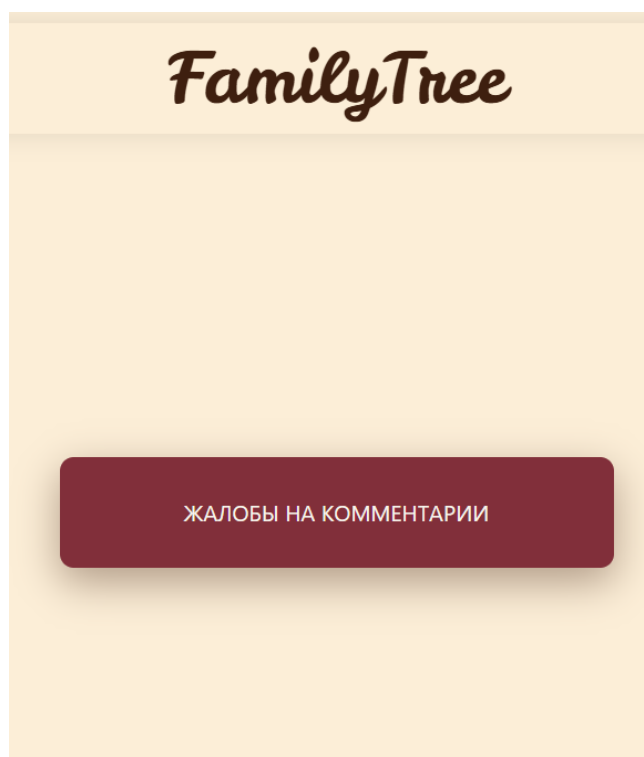


Рисунок Е.25 – Страница навигации администратора

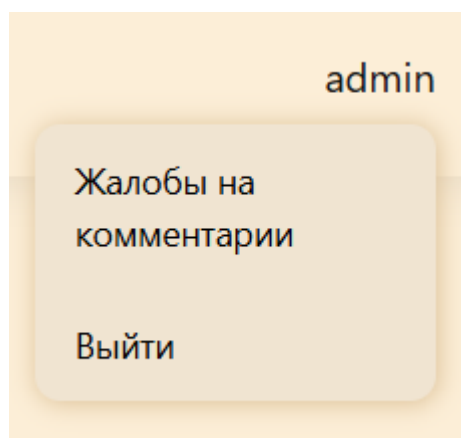


Рисунок Е.26 – Меню администратора



Рисунок Е.27 – Страница управления комментариями для администратора



Рисунок Е.28 – Окно просмотра фото и видео

Приложение Ж

(рекомендуемое)

Результаты функционального тестирования

Редактирование

X

Фамилия:

Неизвестно

Имя:

Мария678

Цифры не допускаются

Отчество:

Пол:

Женский ▾

Девичья фамилия:

Дата рождения:

дд.мм.гггг

Дата смерти:

дд.мм.гггг

СОХРАНИТЬ

Рисунок Ж.1 – Ввод запрещенных символов

Редактирование

X

Фамилия:

Неизвестно

Имя:

Анна-Юлия-Полина-Екатери

Допустимо не более 32 символов

Отчество:

Пол:

Женский ▾

Девичья фамилия:

Дата рождения:

дд.мм.гггг

Дата смерти:

дд.мм.гггг

СОХРАНИТЬ

Рисунок Ж.2 – Ввод символов, превышающий максимально допустимую длину

X

Редактирование

Фамилия:

Имя:
Смайлики не допускаются

Отчество:

Пол: Женский ▾

Девичья фамилия:

Дата рождения:

Дата смерти:

СОХРАНИТЬ

Рисунок Ж.3 – Ввод смайлов

X

Редактирование

Фамилия:

Имя:

Отчество:

Пол: Женский ▾

Девичья фамилия:

Дата рождения:

Дата смерти:

СОХРАНИТЬ

Рисунок Ж.4 – Ввод пустого значения имени

Редактирование

X

Фамилия:	<input type="text" value="Неизвестно"/>
Имя:	<input type="text" value="Maximiliana-Alexandrietta-Jos"/>
Отчество:	<input type="text"/>
Пол:	Женский ▾
Девичья фамилия:	<input type="text"/>
Дата рождения:	<input type="text" value="ДД.ММ.ГГГГ"/>
Дата смерти:	<input type="text" value="ДД.ММ.ГГГГ"/>

СОХРАНИТЬ

Рисунок Ж.5 – Ввод символов максимально допустимой длины

Редактирование

X

Фамилия:	<input type="text" value="Неизвестно"/>
Имя:	<div><input type="text" value="Maximiliana-Alexandrietta-Jos"/> <small>Допустимо не более 32 символов</small></div>
Отчество:	<input type="text"/>
Пол:	Женский ▾
Девичья фамилия:	<input type="text"/>
Дата рождения:	<input type="text" value="ДД.ММ.ГГГГ"/>
Дата смерти:	<input type="text" value="ДД.ММ.ГГГГ"/>

СОХРАНИТЬ

Рисунок Ж.6 – Ввод символов, превышающий максимально допустимую длину
на один символ

Приложение 3
(рекомендуемое)
Исходный код функций тестирования

Листинг 3.1 – Тест testAddPersonWithoutAuth

```
@Test
void testAddPersonWithoutAuth() {
    // Данные для добавления человека
    String personData = ""
        {
            "firstName": "John",
            "lastName": "Doe",
            "birthDate": "1980-01-01",
            "creatorId": 1
        }
        "";

    // Попытка добавления без авторизации
    given()
        .contentType("application/json")
        .body(personData)
        .when()
        .post("person-service/api/people")
        .then()
        .statusCode(401); // Ожидаем 401 Unauthorized
}
```

Листинг 3.2 – Тест testAddPersonWithAuth

```
@Test
void testAddPersonWithAuth() {
    // Регистрируем пользователя
    String signUpData = ""
        {
            "username": "user_test",
            "password": "password"
        }
        "";

    Response signUpResponse = given()
        .contentType("application/json")
        .body(signUpData)
        .when()
        .post("auth-service/api/auth/signup");

    signUpResponse.then()
        .statusCode(201);

    // Получаем токен через авторизацию
    String signInData = ""
        {
            "username": "user_test",
            "password": "password"
        }
        "";
```

```

        """;

Response signInResponse = given()
    .contentType("application/json")
    .body(signInData)
    .when()
    .post("auth-service/api/auth/signin");

signInResponse.then()
    .statusCode(200);

String token =
signInResponse.jsonPath().getString("token");

System.out.println("token: " + token);

// Данные для добавления человека
String personData = """
    {
        "firstName": "test1234",
        "secondName": "test",
        "lastName": "string",
        "dateOfBirth": "2024-12-14",
        "creatorId": 1
    }
""";

// Добавление человека с токеном
given()
    .header("Authorization", "Bearer " + token)
    .contentType("application/json")
    .body(personData)
    .when()
    .post("person-service/api/people")
    .then().log().all()
    .statusCode(201);
}

```

Листинг 3.3 – Тест testAddPersonWithWrongRole

```

@Test
void testAddPersonWithWrongRole() {
    // Получаем токен через авторизацию
    String signInData = """
        {
            "username": "admin",
            "password": "admin"
        }
        """;

    Response signInResponse = given()
        .contentType("application/json")
        .body(signInData)

```

```

        .when()
        .post("auth-service/api/auth/signin");

signInResponse.then()
    .statusCode(200);

String token =
signInResponse.jsonPath().getString("token");

System.out.println("token: " + token);

// Данные для добавления человека
String personData = ""
{
    "firstName": "test1234",
    "secondName": "test",
    "lastName": "string",
    "dateOfBirth": "2024-12-14",
    "creatorId": 1
}
"";

// Добавление человека с токеном
given()
    .header("Authorization", "Bearer " + token)
    .contentType("application/json")
    .body(personData)
    .when()
    .post("person-service/api/people")
    .then().log().all()
    .statusCode(403);
}

```