МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РФ

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

Факультет	*Кафедра*
вычислительной техники	МОиПЭВМ

Направление подготовки 09.03.04 «Программная инженерия»

Направленность Программная инженерия

БАКАЛАВРСКАЯ РАБОТА

на тему

Серверное приложение электронной контактной книги с возможностью ведения заметок

Студент _____ Мартьянова Анастасия Романовна
(подпись, дата) (ФИО полностью)

Руководитель _____ Козлов А. Ю.
(подпись, дата) (фамилия, инициалы)

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

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

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

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

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

Пенза, 2024

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

Факультет	*Кафедра*
Вычислительной техники	МО и ПЭВМ

«Утверждаю»
Заведующий кафедрой
_____А.Ю. Козлов «25»
декабря 2023 г

ЗАДАНИЕ
НА ВЫПУСКНУЮ КВАЛИФИКАЦИОННУЮ РАБОТУ
БАКАЛАВРА

1. Студент__Мартьянова Анастасия Романовна___гр._20ВП1_факультета___ВТ___
<Фамилия, имя, отчество полностью>

2.Тема работы_____Серверное приложение электронной контактной книги с возможностью ведения заметок_____
_____
_____

Тема утверждена приказом ПГУ №_____от "_____"_____20_____

2. Руководитель работы_д. т. н., профессор, зав. Кафедры «МОиПЭВМ» Козлов А. Ю._

3. Задание на работу (назначение разработки, исходные данные и т.п.)
1) Назначение: Серверное приложение электронной контактной книги с возможностью ведения заметок
2) Среда выполнения: Windows 10.
Языковые средства разработки: C#, PostgreSQL.
Инструментальные средства: Visual Studio 2022, pgAdmin 4, Visual Paradigm
Технологии разработки: ООП.
3) Основные функции разрабатываемого программного обеспечения:
а) регистрация и авторизация пользователей в личном кабинете;
б) управление личным кабинетом;
в) создание, просмотр, редактирование, удаление заметок;
г) создание, просмотр, редактирование и удаление контактов;
д) создание, просмотр, редактирование, и удаление тегов;
е) создание, просмотр, редактирование и удаление напоминаний;
ж) сохранение контактов и заметок в файл;
з) сортировка и фильтрация контактов, заметок и тегов;
и) поиск по контактам, заметкам и тегам.

4. Перечень подлежащих разработке вопросов _____

1) Анализ предметной области и постановка задачи.
2) Разработка требований и проектирование.
3) Реализация.
4) Контроль качества программного обеспечения.
5) Оформление пояснительной записки.

5. Календарный график выполнения работы

№ п/п	Наименование этапов работы	Объем работы	Срок выполнения	Подпись руководителя
1	Анализ предметной области и требований к системе	10%	08.04.2024 – 13.04.2024	
2	Проектирование программных средств	20%	15.04.2024 – 25.04.2024	
3	Реализация программных средств	30%	26.04.2024 – 12.05.2024	
4	Тестирование программных средств	20%	13.05.2024 – 22.05.2024	
5	Оформление пояснительной записки	20%	23.05.2024 – 03.06.2024	

Дата выдачи задания «21» декабря 2023г.

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 150 страниц, 40 рисунков, 49 таблиц, 26 использованных источников и 5 приложений.

СЕРВЕРНОЕ ПРИЛОЖЕНИЕ, КЛИЕНТ-СЕРВЕРНАЯ АРХИТЕКТУРА, ASP .NET CORE MVC, КОНТРОЛЛЕРЫ, МОДЕЛИ ДАННЫХ, МОДУЛЬНОЕ ТЕСТИРОВАНИЕ, ФУНКЦИОНАЛЬНОЕ ТЕСТИРОВАНИЕ, МЕТРИКИ КОДА, POSTGRESQL, VISUAL PARADIGM, КОНТАКТЫ, ЗАМЕТКИ

Объект разработки – серверное приложение электронной контактной книги с возможностью ведения заметок.

Цель разработки – изучить предметную область и провести её анализ, выделить бизнес-процессы, разработать концептуальную и физическую модель базы данных в среде Visual Paradigm, а также реализовать и протестировать серверное приложение электронной контактной книги с возможностью ведения заметок.

Результаты разработки – исследована и проанализирована предметная область, выделены основные бизнес-процессы, разработана база данных и серверное приложение электронной контактной книги с возможностью ведения заметок.

Средства разработки: СУБД PostgreSQL 16, case-средство Visual Paradigm 17.1, язык программирования C#, среда разработки Visual Studio 2022.

Содержание

Введение 7
1 Анализ предметной области и постановка задачи на разработку серверного приложения 8
1.1 Анализ предметной области приложения 8
1.1.1 Анализ структурной организации предметной области 8
1.1.2 Анализ свойств информационных объектов предметной области 9
1.2 Сравнительный анализ аналогичных разработок 11
1.2.1 Цели и задачи сравнительного анализа 11
1.2.2 Выбор приложений-аналогов для анализа 12
1.2.3 Обзор приложений-аналогов 12
1.2.4 Результаты сравнительного анализа 17
1.3 Постановка задачи на разработку серверного приложения 18
1.3.1 Цели разработки и назначение приложения 18
1.3.2 Перечень автоматизируемых функций 18
1.3.3 Требования к организации серверного приложения 19
1.4 Планирование и оценка процесса разработки серверного приложения 19
1.4.1 Выбор нотации и программных средств планирования и оценки процесса разработки 19
1.4.2 Планирование процесса разработки приложения 20
1.4.3 Оценка временных и финансовых затрат на разработку 24
2 Проектирование серверного приложения 29
2.1 Моделирование функциональных возможностей приложения 29
2.2 Разработка концептуальной модели данных 38
2.3 Обощенная архитектура системы электронной контактной книги с возможностью ведения заметок 45
2.4 Требования к аппаратно-программному обеспечению серверного приложения 45
2.5 Обоснование выбора технологий и средств разработки серверного приложения 46

3 Разработка программного обеспечения серверного приложения 48
3.1 Разработка архитектуры серверного приложения 48
3.2 Разработка физической модели базы данных 51
3.3 Разработка программного обеспечения доступа к данным 60
3.4 Разработка программного интерфейса с клиентским приложением 64
4 Контроль качества программного обеспечения 68
4.1 Планирование процесса контроля качества программного обеспечения 68
4.1.1 Обоснование выбора модели тестирования 68
4.1.2 Обоснование выбора метрик программного кода 70
4.2 Тестирование программных средств 70
4.2.1 Функциональное тестирование 70
4.2.2 Модульное тестирование 74
4.3 Оценка качества программного кода на основе метрик 76
4.3.1 Метрики Холстеда 76
Заключение 80
Список использованных источников 81
Приложение А (справочное) Глоссарий 84
Приложение Б (рекомендуемое) Скрипты для создания объектов базы данных на PostgreSQL 87
Приложение В (рекомендуемое) Листинг серверного приложения 92
Приложение Г (рекомендуемое) Скриншоты результатов функционального тестирования 132
Приложение Д (рекомендуемое) Листинг модульного тестирования 146

Введение

В настоящее время существует множество систем для комфортного планирования и записи своих мыслей, идей и данных. Такие системы позволяют пользователям хранить важную информацию в удобном месте и быстро получать к ней доступ. Кроме того, подобные приложения привлекательны тем, что с их помощью можно формировать новые привычки, а также хранить списки дел и прочую важную информацию.

Однако обычно такие системы не универсальны: для хранения информации из разных сфер приходится использовать разные сервисы. Списки дел, заметки, контакты – все хранится в разных местах, что причиняет множество неудобств при переключении между различными приложениями. Именно поэтому появляется необходимость в разработке сервиса, который позволил бы хранить большее количество различных данных в одной системе.

Таким образом, целью выпускной квалификационной работы является разработка универсального приложения электронной контактной книги с возможностью ведения заметок. Работа включает в себя анализ и разработку требований, проектирование, реализацию и контроль качества программного обеспечения.

1 Анализ предметной области и постановка задачи на разработку серверного приложения

1.1 Анализ предметной области приложения

1.1.1 Анализ структурной организации предметной области

Электронная контактная книга с возможностью ведения заметок представляет собой приложение с клиент-серверной архитектурой [1]. Это подразумевает под собой описание взаимодействия между пользователями и данными, а также способ хранения данных.

В клиент-серверных архитектурах используется три компонента:

– клиент – устройства или приложения, которые запрашивают информацию у сервера;

– сервер – компьютер, который представляет запрашиваемую информацию клиенту;

– базы данных – хранилища информации, которые используются на серверной стороне и отвечают за хранение и управление данными.

Клиент-серверная архитектура может быть двухуровневой и трехуровневой. При использовании двухуровневой архитектуры приложения сервер выполняет функции приложения и базы данных. Главный минус такого подхода заключается в том, что он обладает небольшой надежностью, так как при взломе сервера происходит и утечка пользовательских данных. Для решения данной проблемы используют базу данных, хранящуюся отдельно от сервера. Именно это и является трехуровневой архитектурой клиент-серверного приложения. При таком подходе сервер выполняет роль логической машины, которая обрабатывает данные, но не хранит их. Кроме того, в серверной части приложения происходит проверка доступности запрашиваемых клиентом данных.

При разработке электронной контактной книги будет использоваться трехуровневая архитектура клиент-серверного приложения [2]. Кроме того, в рамках выпускной квалификационной работы будет разрабатываться именно

серверная часть приложения, то есть, непосредственно, сам сервер, а также база данных.

В рамках исследования структурной организации предметной области были выделены основные компоненты, которые влияют на функционирование серверного приложения. В соответствии с этим, в разрабатываемой электронной контактной книге с возможностью ведения заметок главным действующим лицом будет выступать пользователь. Он сможет создавать и хранить там личные записи, такие как заметки и контакты. Для удобства организации и использования данных пользователь сможет объединять созданные элементы в категории, которые будут идентифицироваться не только при помощи названий, но и с применением цветовых инструментов для наглядности. Кроме того, пользователь будет иметь возможность создавать уведомления о важных событиях, описанных в заметках, а также датах, связанных с контактами. Кроме того, пользователь сможет экспортировать созданные контакты и заметки.

Полученные результаты анализа позволяют определить ключевые аспекты при разработке и внедрении серверной части приложения.

1.1.2 Анализ свойств информационных объектов предметной области

В процессе анализа свойств объектов управления и информационных объектов предметной области было обращено внимание на специфические характеристики, определяющие взаимодействие компонентов в рамках серверного приложения.

Таким образом, каждый пользователь обязательно должен иметь уникальную электронную почту и пароль. При помощи этого будет происходить идентификация пользователя и, соответственно, получение его имеющихся данных. К ним относятся контакты, заметки, категории и уведомления.

Каждый контакт будет иметь следующие характеристики: фамилию, имя, отчество, дату рождения, номера телефонов, ссылки на социальные сети, адреса электронных почт, место работы, адрес проживания и раздел «другое», который может содержать прочую важную информацию о контакте. Каждая заметка

будет иметь название, содержание, а также дату создания. В заметках пользователь сможет хранить различную информацию, например, события, встречи, обсуждения, планы и т. д.

Настраиваемые пользователем уведомления для контактов будут опираться, в первую очередь, на дату рождения. При необходимости можно произвести настройку напоминаний для других важных событий, таких, например, как годовщина свадьбы. Для заметок можно настроить напоминания через необходимый промежуток времени: сутки, неделю и т. д. Кроме того, напоминания будут содержать и другую важную информацию: категорию события, описание события, а также уникальную для каждого типа напоминания иконку. Помимо этого, пользователь сможет настроить тип каждого уведомления: со звуком или без.

В разрабатываемом приложении будут реализованы категории контактов и заметок. Категории представляют собой папки, в которых созданные пользователем элементы сгруппированы по общим тематикам, например, «Работа», «Учеба» и т. д. Для удобства использования каждая группа будет выделена отдельным цветом, настраиваемым пользователем.

В разрабатываемом приложении также будут представлены полезные для пользователей функции: поиск по записям, сортировка, фильтрация, а также экспорт созданных элементов. Поиск будет осуществляться по удобным для пользователя критериям: в случае контактов – по ФИО и дате рождения, в случае заметок – по названию и дате создания, в случае категорий – по названию. При наличии большого объема информации это позволит пользователю быстро находить нужную запись. Сортировка контактов и заметок будет осуществляться по тем же критериям. Эта функция позволит определить, например, ближайшую дату рождения человека из списка созданных контактов. Фильтрация контактов и заметок будет происходить по датам. Благодаря данной функции пользователь сможет просмотреть, например, список контактов, чьи даты рождения в октябре, а также список заметок, созданных в марте.

Полученные при анализе свойств объектов управления и информационных объектов предметной области результаты позволяют определить наиболее значимые свойства объектов, которые следует учитывать при разработке функционала приложения.

1.2 Сравнительный анализ аналогичных разработок

1.2.1 Цели и задачи сравнительного анализа

В ходе выполнения выпускной квалификационной работы необходимо провести сравнительный анализ аналогичных разработок [3]. Для его выполнения были поставлены следующие цели:

– изучить различные подходы к разработке серверных приложений;

– определить основные особенности и преимущества других разработок;

– выявить возможные трудности и ограничения при использовании определенных технологий;

– оценить эффективность и надежность других разработок с целью их усовершенствования в ходе выполнения выпускной квалификационной работы.

В соответствии с определенными выше целями был поставлен ряд задач на выполнение сравнительного анализа:

– провести сравнительный анализ различных подходов к разработке серверных приложений электронный контактных книг с возможностью ведения заметок;

– изучить основные технологии и инструменты, используемые для разработки таких приложений;

– оценить возможности и ограничения, связанные с использованием выбранных технологий, а также их влияние на качество и производительность серверной части приложения.

После выполнения поставленных целей и задач будет определен перечень обязательных функций, которые необходимо реализовать в разрабатываемом

приложении. Кроме того, на основе этого будут определены уникальные функции, позволяющие выделить разработку среди конкурентов на рынке.

1.2.2 Выбор приложений-аналогов для анализа

В качестве приложений для сравнительного анализа были выбраны не только электронные контактные книги с возможностью ведения заметок, но и отдельные приложения для хранения контактов и заметок.

Далее в выпускной квалификационной работе будут описываться следующие электронные сервисы:

– Google Контакты;
– Evernote;
– Notion;
– Trello;
– Remember the Milk.

Данные системы представляют собой приложения для выполнения различных целей, поэтому по результатам сравнения удастся получить исчерпывающую характеристику функционала как части приложения, связанной с контактами, так и части приложения, связанной с заметками.

1.2.3 Обзор приложений-аналогов

При выполнении сравнительного анализа аналогов, в первую очередь, следует обратить внимание на базовые функции, такие, как доступность системы и наличие онлайн-версии. Кроме того, немаловажным фактором является фирма-производитель. Сравнительный анализ по данным характеристикам содержится в таблице 1.

Таким образом, большинство рассматриваемых приложений являются платными, при этом все перечисленные аналоги имеют онлайн-версии. Сервис Google Контакты предназначен для работы с контактами, а остальные приложения предоставляют функционал для работы с заметками.

Таблица 1 – Базовая информация об аналогичных системах

Система	Доступность	Наличие онлайн-версии	Фирма-производитель
Google Контакты	Бесплатная	+	Google
Evernote	Платная, но есть бесплатная пробная версия	+	Evernote Corporation
Notion	Платная, но есть бесплатная пробная версия	+	Notion Labs Inc.
Trello	Бесплатная	+	Atlassian и Glitch, Inc.
Remember the Milk	Платная, но есть бесплатная пробная версия	+	Remember The Milk Inc.

Далее стоит рассмотреть каждый сервис более подробно.

Google Контакты – это веб-сервис от корпорации Google, предоставляющий инструменты для работы с контактной книгой. Работает полностью на русском языке. На рисунке 1 представлен интерфейс страницы для добавления нового контакта.

Сервис предоставляет пользователю возможность вводить личные данные контактов: имя, фамилию, место работы и т. д. Кроме того, в приложении также есть возможность редактировать, удалять, экспортировать контакты. Google Контакты позволяют пользователю вводить несколько номеров телефонов для контакта, а также хранить адрес места жительства контакта. Помимо этого, для наглядности к карточке контакта можно добавить фотографию, которая затем будет отображаться в основном списке контактов. Кроме того, в Google Контактах ведется статистика того, с кем из списка знакомых пользователь чаще всего общается. При желании, такие контакты можно помечать как избранные.

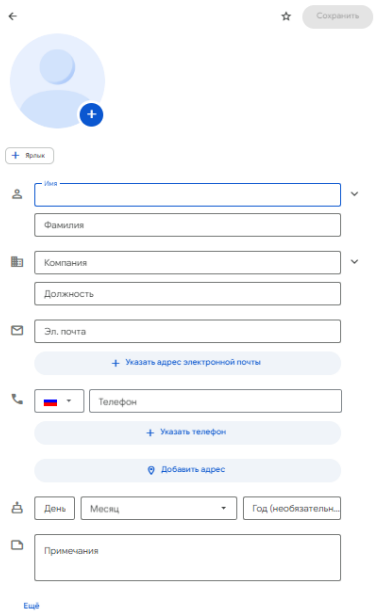


Рисунок 1 – Страница добавления нового контакта в Google Контакты

В сервисе Google Контакты также доступна функция группировки контактов.

Evernote – это веб-сервис для создания и хранения заметок. Представлен полностью на английском языке. Интерфейс приложения представлен на рисунке 2.

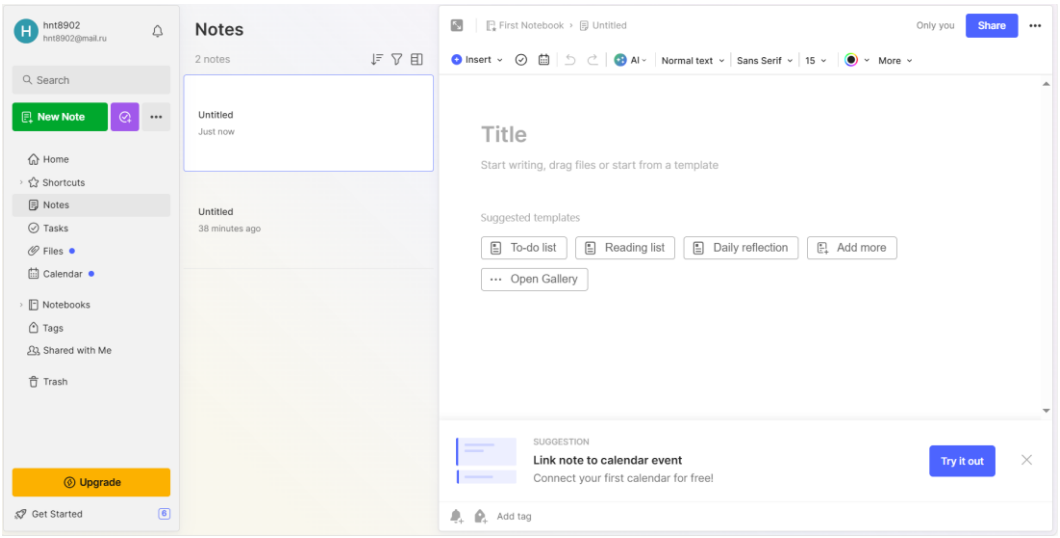


Рисунок 2 – Интерфейс веб-приложения Evernote для создания и хранения заметок

Пользователь может редактировать тип, цвет и размер шрифта, а также прикреплять к заметке различные файлы. В приложении в отдельном разделе присутствуют задачи и календарь, служащий для напоминаний о запланированных мероприятиях. Кроме того, в приложении есть «дневники», представляющие собой группы заметок, а также теги, которые можно присваивать заметкам внутри «дневников».

Notion – веб –приложение, позволяющее пользователям создавать системы управления знаниями, вести заметки, управлять данными и проектами. Представлено только на английском языке. На рисунке 3 изображен интерфейс страницы добавления задач на неделю.

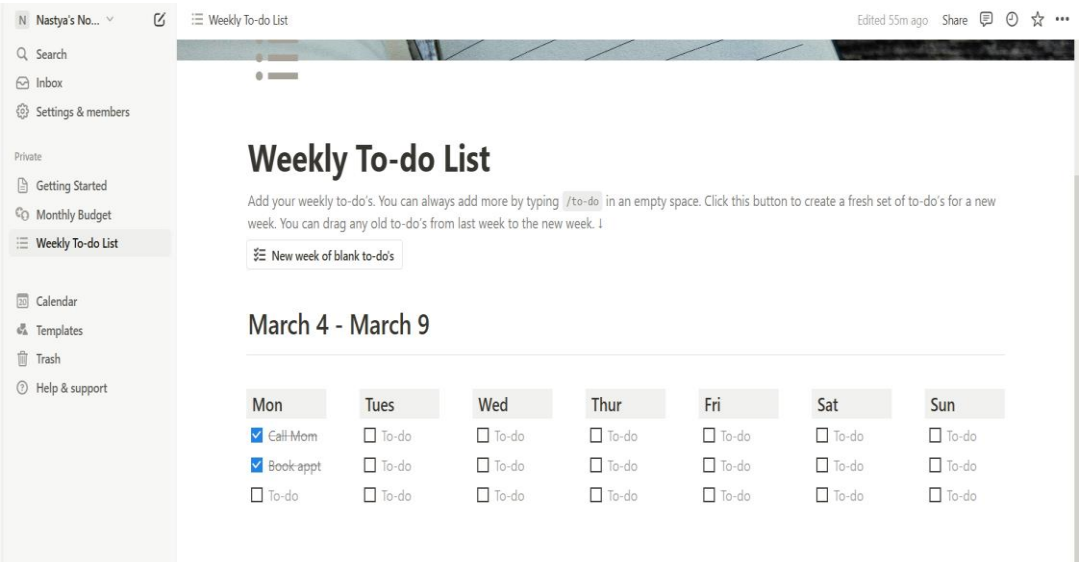


Рисунок 3 – Страница веб-приложения Notion для добавления задач на неделю

Кроме того, приложение Notion позволяет пользователям формировать и отслеживать ежемесячный бюджет.

Trello – онлайн-система для управления проектами, основанная на японской системе планирования с помощью кабан-досок. Поддерживает русский язык. На рисунке 4 представлен интерфейс главной страницы веб-приложения.

Данный сервис предоставляет пользователям возможность просмотра созданных задач не только на досках, но и в табличном виде и на календаре. В приложение также можно добавлять пользователей для совместной работы над

проектами. При этом они отобразятся во вкладке «Участники». Также в сервисе присутствуют функции сортировки и фильтрации по созданным записям.

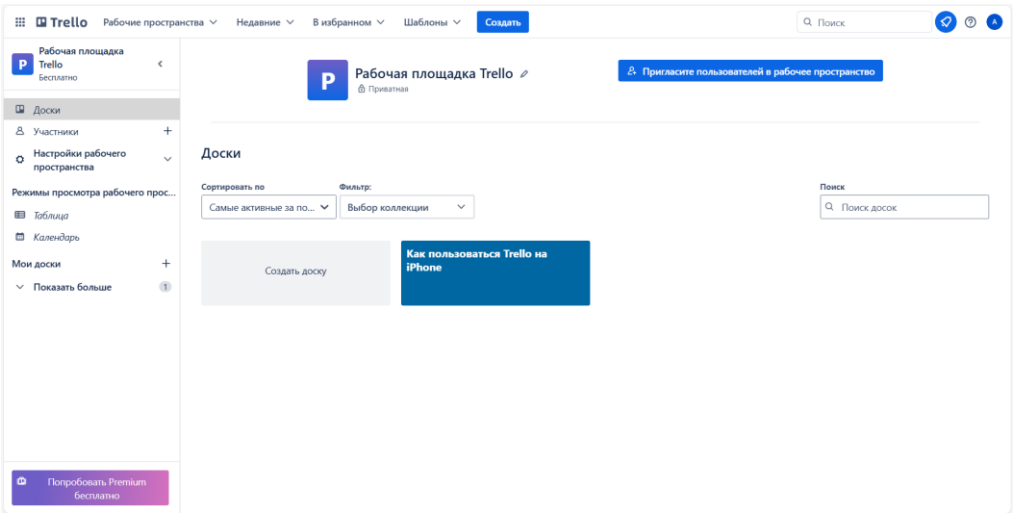


Рисунок 4 – Главная страница веб-приложения Trello

Remember the Milk – сервис для хранения списка дел. Поддерживает русский язык. Интерфейс главной страницы веб-приложения представлен на рисунке 5.

Приложение позволяет пользователям создавать задачи, а также группировать их по спискам, которые, в свою очередь, тоже создаются пользователем.

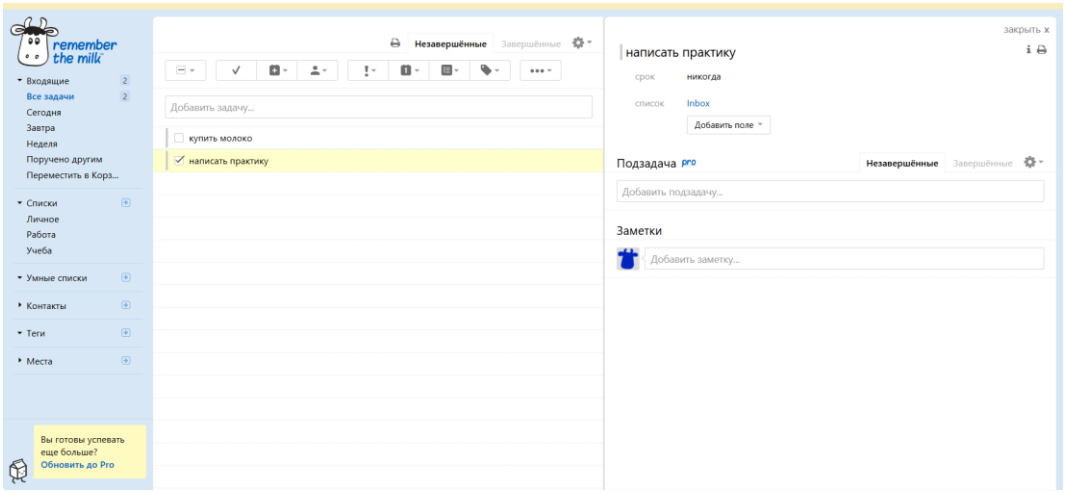


Рисунок 5 – Интерфейс главной страницы веб-приложения Remember the Milk

Помимо этого, в приложении есть возможность помечать задачи тегами, отмечать завершенные дела, а также создавать умные списки.

1.2.4 Результаты сравнительного анализа

В результате обзора аналогичных разработок были разработаны следующие критерии для сравнительного анализа:

– поддержка русского языка;

– возможность группировки созданных элементов (наличие категорий);

– наличие напоминаний;

– наличие функционала для работы и с контактами, и с заметками;

– возможность сохранения данных в файл.

Данные критерии позволяют оценить преимущества и недостатки функциональных возможностей рассматриваемых приложений. Результаты сравнительного анализа в соответствии с определенными критериями представлены в таблице 2.

Таблица 2 – Результаты сравнительного анализа приложений-аналогов

Название	Поддержка русского языка	Возможность группировки созданных элементов	Наличие напоминаний	Возможность работы с контактами и заметками	Сохранение данных в файл
Google Контакты	+	+	-	Только контакты	+
Evernote	-	+	+	Только заметки	-
Notion	-	-	+	Только заметки	-
Trello	+	+	+	Только заметки	-
Remember the Milk	+	+	+	Только заметки	-

На основе таблицы можно сделать вывод о том, что приложения-аналоги предоставляют пользователям возможность работы либо с заметками, либо с контактами. Таким образом, главным преимуществом разрабатываемой электронной контактной книги с возможностью ведения заметок станет именно совмещенный функционал для работы с контактами и заметками. Кроме того, большинство приложений не поддерживают сохранение созданных данных в

файлы, поэтому это также станет весомым преимуществом в разрабатываемом приложении.

1.3 Постановка задачи на разработку серверного приложения

1.3.1 Цели разработки и назначение приложения

Разрабатываемое приложение электронной контактной книги с возможностью ведения заметок должно обеспечивать пользователю удобный набор инструментов для хранения и управления данными. В соответствии с этим, были выделены основные цели разработки:

– обеспечить хранение и управление пользовательскими данными: заметками, контактами, напоминаниями и т. д.;

– предоставить API для работы с данными: создания, чтения, редактирования и удаления;

– обеспечить надежную и масштабируемую архитектуру.

Основным назначением на разработку серверной части приложения является обработка запросов от клиентской части [4]. Помимо этого, требуется также сохранять данные в базе данных и предоставлять их по запросу со стороны клиентской части. Стоит также отметить, что после окончания основного процесса разработки необходимо осуществлять поддержку функционала приложения и, помимо этого, при необходимости внедрять новые возможности для упрощения работы приложения.

1.3.2 Перечень автоматизируемых функций

Программные средства приложения электронной контактной книги с возможностью ведения заметок должны обладать следующим перечнем автоматизированных функций:

– создание, чтение, изменение и удаление данных;

– автоматическая проверка корректности данных пользователя при авторизации;

– автоматическая проверка полноты и правильности данных, вводимых пользователем при создании контактов, заметок и т. д.;

– автоматическая отправка пользовательских уведомлений.

Помимо перечисленных функций, для корректной работы приложения должна происходить корректная доставка и обработка данных от клиентской части к серверной, а также в обратном направлении. Это включает в себя, в том числе, сортировку и фильтрацию данных.

1.3.3 Требования к организации серверного приложения

Разрабатываемое приложение электронной контактной книги с возможностью ведения заметок будет иметь клиент-серверную архитектуру. Серверная часть должна состоять из двух уровней: реализации бизнес-логики и доступа к данным. При этом данные уровни должны быть слабосвязанными, а также развернутыми независимо друг от друга.

Для обеспечения безопасности должна осуществляться авторизация пользователей. Благодаря данной функции каждый пользователь будет получать только свои данные и, кроме того, доступ к ним будет защищен уникальным для системы адресом электронной почты и паролем.

В разрабатываемой серверной части приложения должно быть учтено требование о производительности приложения. В данном случае оно будет достигаться за счет базового механизма оптимизации производительности – индексации базы данных [5].

1.4 Планирование и оценка процесса разработки серверного приложения

1.4.1 Выбор нотации и программных средств планирования и оценки процесса разработки

Для реализации планирования и оценки процесса разработки серверной части электронной контактной книги с возможностью ведения заметок было принято решение использовать Microsoft Project [6]. Данное программное

обеспечение предназначено для управления проектами, распределения ресурсов по задачам. Помимо этого, Microsoft Project предоставляет возможность отслеживать прогресс выполнения проекта, а также составлять отчеты о стоимости и трудозатратах процесса разработки.

При работе с Microsoft Project выделают ряд существенных преимуществ этого программного обеспечения:

– организованный обзор задач и сроков проекта;

– разнообразные инструменты для отслеживания прогресса при работе над проектом;

– возможность настраивать проект с учетом пожеланий пользователя;

– интеграция с другими приложениями Microsoft Office, в том числе, с Excel.

Таким образом, Microsoft Project предоставляет полный спектр функциональных возможностей для планирования проекта и оценки затрат на его выполнение.

1.4.2 Планирование процесса разработки приложения

Для формирования правильного процесса разработки приложения необходимо провести планирование. Основной целью планирования является составление графика выполнения работ по проекту и приблизительный расчет трудовых и финансовых затрат.

В таблице 3 описаны задачи, которые необходимо выполнить в рамках разработки приложения. Кроме того, для каждой из них определена длительность и предшественники – задачи, которые необходимо выполнить до начала заданной.

Таблица 3 – Список задач проекта

№	Наименование задачи	Длительность	Предшественник
1	2	3	4
1	Начало работы над проектом	-	-
2	Анализ предметной области и постановка задачи на разработку серверного приложения	-	-
3	Анализ предметной области приложения	1 день	1
4	Анализ структурной организации предметной области	1 день	3

Продолжение таблицы 3

1	2	3	4
5	Анализ свойств объектов управления и информационных объектов предметной области	1 день	4
6	Анализ процессов управления и ведения данных в предметной области	1 день	5
7	Анализ предметной области приложения завершен	-	6
8	Сравнительный анализ аналогичных разработок	-	2
9	Определение целей и задач сравнительного анализа	1 день	7
10	Выбор приложений-аналогов для анализа	1 день	9
11	Обзор приложений-аналогов	1 день	10
12	Анализ результатов сравнительного анализа	1 день	11
13	Сравнительный анализ аналогичных разработок завершен	-	12
14	Постановка задачи на разработку серверного приложения	-	8
15	Определение целей разработки и назначения приложения	1 день	13
16	Формирование перечня автоматизируемых функций	1 день	15
17	Формирование требований к организации серверного приложения	1 день	16
18	Постановка задачи на разработку серверного приложения завершена	-	17
19	Планирование и оценка процесса разработки серверного приложения	-	14
20	Выбор нотации и программных средств планирования и оценки процесса разработки	1 день	18
21	Планирование процесса разработки приложения	1 день	20
22	Оценка временных и финансовых затрат на разработку	1 день	21
23	Планирование и оценка процесса разработки серверного приложения завершено	-	22
24	Проектирование серверного приложения	-	19
25	Моделирование функциональных возможностей приложения	2 дня	23
26	Разработка концептуальной модели данных	1 день	25
27	Формирование требований к аппаратно-программному обеспечению серверного приложения	1 день	26
28	Обоснование выбора технологий и средств разработки серверного приложения	1 день	27
29	Проектирование серверного приложения завершено	-	28
30	Разработка программного обеспечения серверного приложения	-	24
31	Разработка архитектуры серверного приложения	3 дня	29
32	Разработка физической модели базы данных	3 дня	31
33	Разработка программного обеспечения доступа к данным	2 дня	32
34	Разработка программного интерфейса с клиентским приложением	2 дня	33
35	Разработка программного обеспечения бизнес-логики серверного приложения	3 дня	34
36	Разработка программного обеспечения серверного приложения завершена	-	35
37	Контроль качества программного обеспечения	-	30
38	Планирование процесса контроля качества программного обеспечения	1 день	36
39	Проведение мероприятий по контролю качества программного обеспечения	3 дня	38
40	Тестирование программных средств	3 дня	39
41	Оценка качества программного кода на основе метрик	1 день	40
42	Контроль качества программных средств завершен	-	41
43	Сдача проекта	-	37
44	Оформление и сбор документации	3 дня	42
45	Ввод в эксплуатацию	1 день	44
46	Сдача проекта окончена	-	45
47	Конец работы над проектом	-	43

После определения задач проекта и их длительности необходимо установить требуемые для выполнения проекта ресурсы. Результат формирования данного списка представлен в таблице 4.

В качестве ресурсов для выполнения проекта выступают аналитик, программист, дизайнер, тестировщик, интернет и компьютер. При этом данные ресурсы подразделяются на три категории в зависимости от их назначения: трудовые, затраты, материальные.

Таблица 4 – Перечень ресурсов проекта

Название ресурса	Тип ресурса	Стандартная ставка	Сверхурочная ставка
Аналитик	Трудовой	200,00 ₽/ч	300,00 ₽/ч
Программист	Трудовой	200,00 ₽/ч	300,00 ₽/ч
Дизайнер	Трудовой	150,00 ₽/ч	200,00 ₽/ч
Тестировщик	Трудовой	150,00 ₽/ч	200,00 ₽/ч
Интернет	Затраты	-	-
Компьютер	Материальный	50 000,00 ₽	-

Далее необходимо назначить ресурсы для каждой задачи. В дальнейшем благодаря этим данным будет происходить формирование бюджета проекта. Данные о задачах и требуемых для их исполнения ресурсах представлены в таблице 5.

Таблица 5 – Назначение ресурсов для задач проекта

№	Наименование задачи	Ресурсы	Единицы
1	2	3	4
1	Начало работы над проектом		
2	Анализ предметной области и постановка задачи на разработку серверного приложения		
3	Анализ предметной области приложения	Аналитик	100%
4	Анализ структурной организации предметной области	Аналитик	100%
5	Анализ свойств объектов управления и информационных объектов предметной области	Аналитик	100%
6	Анализ процессов управления и ведения данных в предметной области	Аналитик	100%
7	Анализ предметной области приложения завершен		

Продолжение таблицы 5

1	2	3	4
8	Сравнительный анализ аналогичных разработок		
9	Определение целей и задач сравнительного анализа	Аналитик	100%
10	Выбор приложений-аналогов для анализа	Аналитик	100%
11	Обзор приложений-аналогов	Аналитик	100%
12	Анализ результатов сравнительного анализа	Аналитик	100%
13	Сравнительный анализ аналогичных разработок завершен		
14	Постановка задачи на разработку серверного приложения		
15	Определение целей разработки и назначения приложения	Программист	100%
16	Формирование перечня автоматизируемых функций	Программист	100%
17	Формирование требований к организации серверного приложения	Программист	100%
18	Постановка задачи на разработку серверного приложения завершена	Программист	100%
19	Планирование и оценка процесса разработки серверного приложения		
20	Выбор нотации и программных средств планирования и оценки процесса разработки	Программист	100%
21	Планирование процесса разработки приложения	Программист	100%
22	Оценка временных и финансовых затрат на разработку	Аналитик	47%
23	Планирование и оценка процесса разработки серверного приложения завершено		
24	Проектирование серверного приложения		
25	Моделирование функциональных возможностей приложения	Программист	100%
26	Разработка концептуальной модели данных	Программист	100%
27	Формирование требований к аппаратно-программному обеспечению серверного приложения	Программист	100%
28	Обоснование выбора технологий и средств разработки серверного приложения	Программист	100%
29	Проектирование серверного приложения завершено		
30	Разработка программного обеспечения серверного приложения		
31	Разработка архитектуры серверного приложения	Программист Интернет	100% 200 руб.
32	Разработка физической модели базы данных	Программист Интернет	100% 200 руб.
33	Разработка программного обеспечения доступа к данным	Программист Интернет	100% 200 руб.
34	Разработка программного интерфейса с клиентским приложением	Дизайнер Интернет	100% 200 руб.
35	Разработка программного обеспечения бизнес-логики серверного приложения	Программист Интернет	100% 200 руб.
36	Разработка программного обеспечения серверного приложения завершена		
37	Контроль качества программного обеспечения		
38	Планирование процесса контроля качества программного обеспечения	Тестировщик	100%

Окончание таблицы 5

39	Проведение мероприятий по контролю качества программного обеспечения	Тестировщик	100%
40	Тестирование программных средств	Тестировщик Интернет	100% 500 руб.
41	Оценка качества программного кода на основе метрик	Тестировщик	100%
42	Контроль качества программных средств завершен		
43	Сдача проекта		
44	Оформление и сбор документации	Программист	100%
45	Ввод в эксплуатацию	Программист Интернет	100% 400 руб.
46	Сдача проекта окончена		
47	Конец работы над проектом		

На основе полученных данных можно провести оценку временных и финансовых затрат на разработку приложения.

1.4.3 Оценка временных и финансовых затрат на разработку

На основе данных, полученных в предыдущем разделе, был создан проект в Microsoft Project. В этой среде при помощи автоматизированных функций была сформирована диаграмма Ганта, описывающая задачи и распределение ресурсов, а также временные затраты на разработку (рисунок 6).

На диаграмме Ганта созданные задачи расположены на временном графике, что позволяет удобно отслеживать время и сроки выполнения каждой задачи. Кроме того, на диаграмме также отображается последовательность выполняемых задач: какая из них является предшественником и что за ней следует.

Сбоку от представления задачи располагается наименование ресурсов, необходимых для ее выполнения. Для затрат рядом также располагается стоимость ресурса, необходимая для выполнения данной задачи.

Группы задач, вехи, отображаются в виде черных полос над задачами, в них входящими. Для вех также отображается точный срок начала и окончания выполнения работы. Кроме того, на диаграмме при помощи связей отображена последовательность выполнения задач.

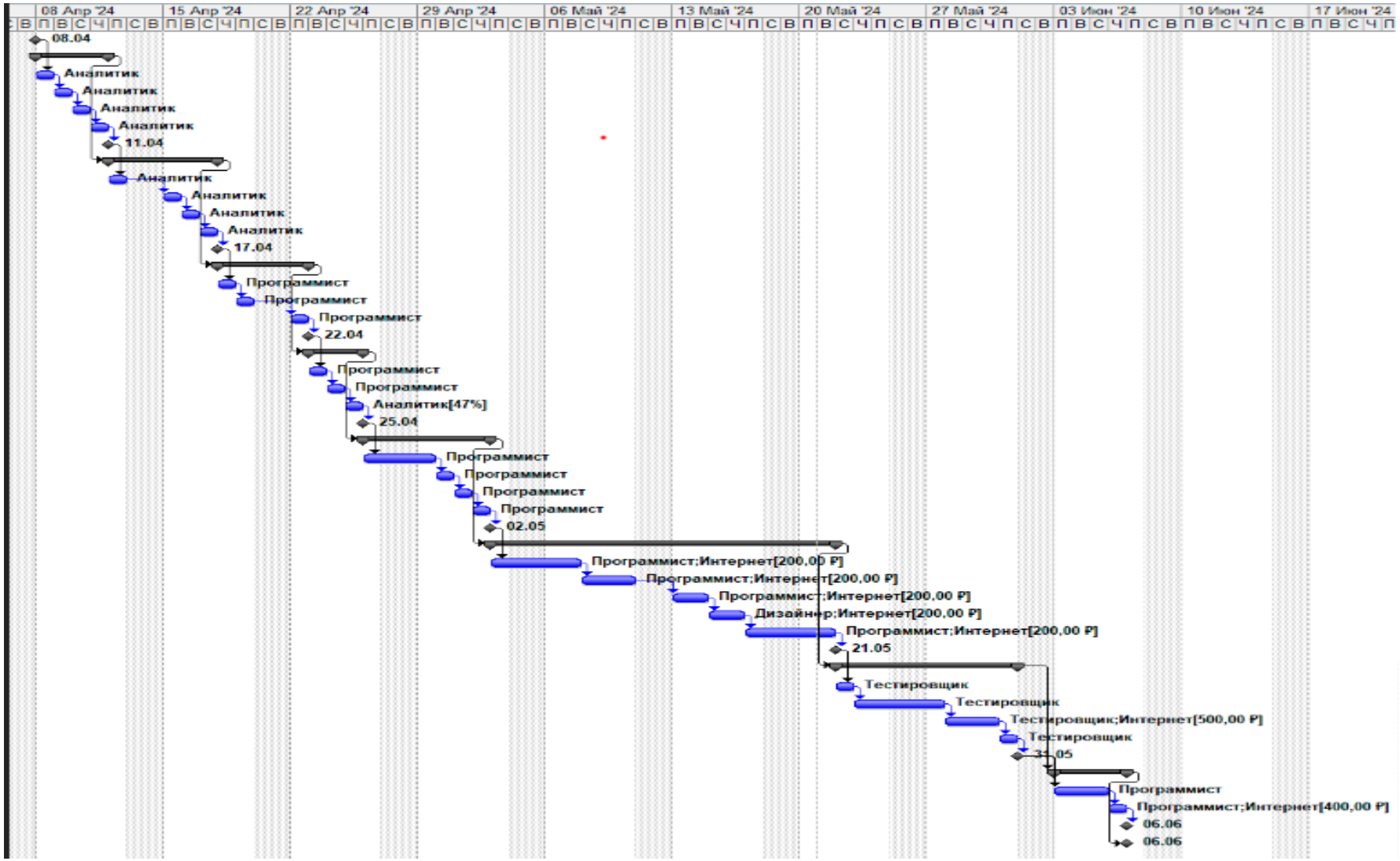


Рисунок 6 – Диаграмма Ганта задач с распределением ресурсов

На основе диаграммы видно, что в качестве начальной даты проекта было взято 08.04.2024, а дата окончания проекта, 06.06.2024, была подсчитана автоматически на основе данных о длительности выполнения задач с учетом пятидневной рабочей недели.

Microsoft Project также предоставляет полезную функцию для получения отчета по созданному проекту. Таким образом был сформирован отчет по по общим затратам на выполнение проекта, представленный на рисунке 7.

Ид.	Название задачи	Фиксированные затраты	Начисление фикс. затрат	Общие затраты
31	Разработка архитектуры серверного	0,00 ₽	Пропорциональное	5 000,00 ₽
32	Разработка физической модели базы	0,00 ₽	Пропорциональное	5 000,00 ₽
35	Разработка программного обеспечен	0,00 ₽	Пропорциональное	5 000,00 ₽
44	Оформление и сбор документации	0,00 ₽	Пропорциональное	4 800,00 ₽
40	Тестирование программных средств	0,00 ₽	Пропорциональное	4 100,00 ₽
39	Проведение мероприятий по контрол	0,00 ₽	Пропорциональное	3 600,00 ₽
33	Разработка программного обеспече	0,00 ₽	Пропорциональное	3 400,00 ₽
25	Моделирование функциональных во	0,00 ₽	Пропорциональное	3 200,00 ₽
34	Разработка программного интерфей	0,00 ₽	Пропорциональное	2 600,00 ₽
45	Ввод в эксплуатацию	0,00 ₽	Пропорциональное	2 000,00 ₽
3	Анализ предметной области прилож	0,00 ₽	Пропорциональное	1 600,00 ₽
4	Анализ структурной организации пр	0,00 ₽	Пропорциональное	1 600,00 ₽
5	Анализ свойств объектов управлени	0,00 ₽	Пропорциональное	1 600,00 ₽
6	Анализ процессов управления и вед	0,00 ₽	Пропорциональное	1 600,00 ₽
9	Определение целей и задач сравни	0,00 ₽	Пропорциональное	1 600,00 ₽
10	Выбор приложений-аналогов для ан	0,00 ₽	Пропорциональное	1 600,00 ₽
11	Обзор приложений-аналогов	0,00 ₽	Пропорциональное	1 600,00 ₽
12	Анализ результатов сравнительного	0,00 ₽	Пропорциональное	1 600,00 ₽
15	Определение целей разработки и на	0,00 ₽	Пропорциональное	1 600,00 ₽
16	Формирование перечня автоматизир	0,00 ₽	Пропорциональное	1 600,00 ₽
17	Формирование требований к организ	0,00 ₽	Пропорциональное	1 600,00 ₽
20	Выбор нотации и программных сред	0,00 ₽	Пропорциональное	1 600,00 ₽
21	Планирование процесса разработки	0,00 ₽	Пропорциональное	1 600,00 ₽
26	Разработка концептуальной модели	0,00 ₽	Пропорциональное	1 600,00 ₽
27	Формирование требований к аппара	0,00 ₽	Пропорциональное	1 600,00 ₽
28	Обоснование выбора технологий и с	0,00 ₽	Пропорциональное	1 600,00 ₽
38	Планирование процесса контроля ка	0,00 ₽	Пропорциональное	1 200,00 ₽
41	Оценка качества программного кода	0,00 ₽	Пропорциональное	1 200,00 ₽
22	Оценка временных и финансовых за	0,00 ₽	Пропорциональное	746,67 ₽
1	Начало работы над проектом	0,00 ₽	Пропорциональное	0,00 ₽
7	Анализ предметной области прилож	0,00 ₽	Пропорциональное	0,00 ₽
13	Сравнительный анализ аналогичных	0,00 ₽	Пропорциональное	0,00 ₽
18	Постановка задачи на разработку се	0,00 ₽	Пропорциональное	0,00 ₽
23	Планирование и оценка процесса ра	0,00 ₽	Пропорциональное	0,00 ₽
29	Проектирование серверного прилож	0,00 ₽	Пропорциональное	0,00 ₽
36	Разработка программного обеспече	0,00 ₽	Пропорциональное	0,00 ₽
42	Контроль качества программных сре	0,00 ₽	Пропорциональное	0,00 ₽
46	Сдача проекта окончена	0,00 ₽	Пропорциональное	0,00 ₽
47	Конец работы над проектом	0,00 ₽	Пропорциональное	0,00 ₽
		0,00 ₽		**67 446,67 ₽**

Рисунок 7 – Отчет об общих затратах проекта

На сформированном отчете все задачи распределяются по сумме затрат: наиболее затратные располагаются вверху документа, а наименее затратные – внизу. При этом стоит отметить, что все затраты на выполнение проекта начислялись пропорционально в течение всего периода работы.

Кроме того, был также сформирован отчет по использованию ресурсов, представленный на рисунке 8.

	Апрель	Май	Июнь	Июль	Итого
Аналитик	67,73 ч				67,73 ч
Анализ предметной области приложения	8 ч				8 ч
Анализ структурной организации предметной области	8 ч				8 ч
Анализ свойств объектов управления и информационных объек-тов предметной области	8 ч				8 ч
Анализ процессов управления и ведения данных в предметной области	8 ч				8 ч
Определение целей и задач сравнительного анализа	8 ч				8 ч
Выбор приложений-аналогов для анализа	8 ч				8 ч
Обзор приложений-аналогов	8 ч				8 ч
Анализ результатов сравнительного анализа	8 ч				8 ч
Оценка временных и финансовых затрат на разработку	3,73 ч				3,73 ч
Программист	64 ч	104 ч	32 ч		200 ч
Определение целей разработки и назначения приложения	8 ч				8 ч
Формирование перечня автоматизируемых функций	8 ч				8 ч
Формирование требований к организации серверного приложения	8 ч				8 ч
Постановка задачи на разработку серверного приложения завершена					
Выбор нотации и программных средств планирования и оценки процесса раз-работки	8 ч				8 ч
Планирование процесса разработки приложения	8 ч				8 ч
Моделирование функциональных возможностей приложения	16 ч				16 ч
Разработка концептуальной модели данных	8 ч				8 ч
Формирование требований к аппаратно-программному обеспечению серверного приложения		8 ч			8 ч
Обоснование выбора технологий и средств разработки серверного приложения		8 ч			8 ч
Разработка архитектуры серверного приложения		24 ч			24 ч
Разработка физической модели базы данных		24 ч			24 ч
Разработка программного обеспечения доступа к данным		16 ч			16 ч
Разработка программного обеспечения бизнес-логики серверного приложения		24 ч			24 ч
Оформление и сбор документации			24 ч		24 ч
Ввод в эксплуатацию			8 ч		8 ч
Дизайнер		16 ч			16 ч
Разработка программного интерфейса с клиентским приложением		16 ч			16 ч
Тестировщик		64 ч			64 ч
Планирование процесса контроля качества программного обеспечения		8 ч			8 ч
Проведение мероприятий по контролю качества программного обеспечения		24 ч			24 ч
Тестирование программных средств		24 ч			24 ч
Оценка качества программного кода на основе метрик		8 ч			8 ч
Интернет					
Разработка архитектуры серверного приложения					
Разработка физической модели базы данных					
Разработка программного обеспечения доступа к данным					
Разработка программного интерфейса с клиентским приложением					
Разработка программного обеспечения бизнес-логики серверного приложения					
Тестирование программных средств					
Ввод в эксплуатацию					
Компьютер					
Итого	131,73 ч	184 ч	32 ч		347,73 ч

Рисунок 8 – Отчет по использованию ресурсов проект

По результатам отчета по использованию ресурсов можно сделать вывод, что наибольшая нагрузка приходится на программиста – 200 рабочих часов. При этом этот же ресурс имеем и наибольшее количество задач.

По результатам отчета также видно, что наибольшая часть работ была выполнена в мае – 104 часа. При этом часть проекта была выполнена в мае – 67,73 часа, а наименьшая часть работ была проведена в июне – 32 часа.

По результатам отчета по общим затратам наибольшее количество средств было потрачено на разработку архитектуры серверного приложения, разработку физической модели базы данных и разработку программного обеспечения серверного приложения. Помимо этого, достаточно затратными задачами также стали оформление и сбор документации и тестирование программных средств. Наименее затратной задачей стала оценка временных и финансовых затрат на разработку серверного приложения.

По результатам оценки временных затрат длительность выполнения проекта составит 347,73 часа, при этом необходимый бюджет для разработки приложения – 67446,67 рублей.

2 Проектирование серверного приложения

2.1 Моделирование функциональных возможностей приложения

В ходе анализа установленных информационных задач была разработана диаграмма вариантов использования UML, представленная на рисунке 9 [7].

На диаграмме видно, что пользователю доступны различные действия с контактами, заметками и тегами. Немаловажным доступным для пользователя функционалом является просмотр и редактирование информации в личном кабинете, а также удаление и выход из аккаунта.

На диаграмме вариантов использования действующим лицом (актером) является клиентская часть приложения. Именно с ней происходит взаимодействие со стороны пользователя, а она, в свою очередь, осуществляет отправку запросов в серверное приложение, а также получает ответы от серверной части.

На разработанной диаграмме вариантов использования все бизнес-процессы взаимодействуют, непосредственно, с системой управления базами данных (СУБД), в которой хранятся все требуемые данные [8].

Стоит отметить, что на диаграмме вариантов использования используется два типа отношений: отношения ассоциации и отношения включения. Первый тип служит для обозначения связи между актером и отдельным вариантом использования, отображенным на диаграмме. Второй же тип отношений указывает на то, что каждый экземпляр первого варианта использования включает в себя выполнение второго варианта использования, в сторону которого направлена связь. Так, например, при добавлении новой заметки можно добавить данные заметки, определить ее категорию, а также сразу настроить уведомления. После выполнения данных действий их результаты будут переданы в СУБД.

Некоторые варианты использования не содержат отношений включения, так как для их выполнения не требуется присутствия каких-либо других действий.

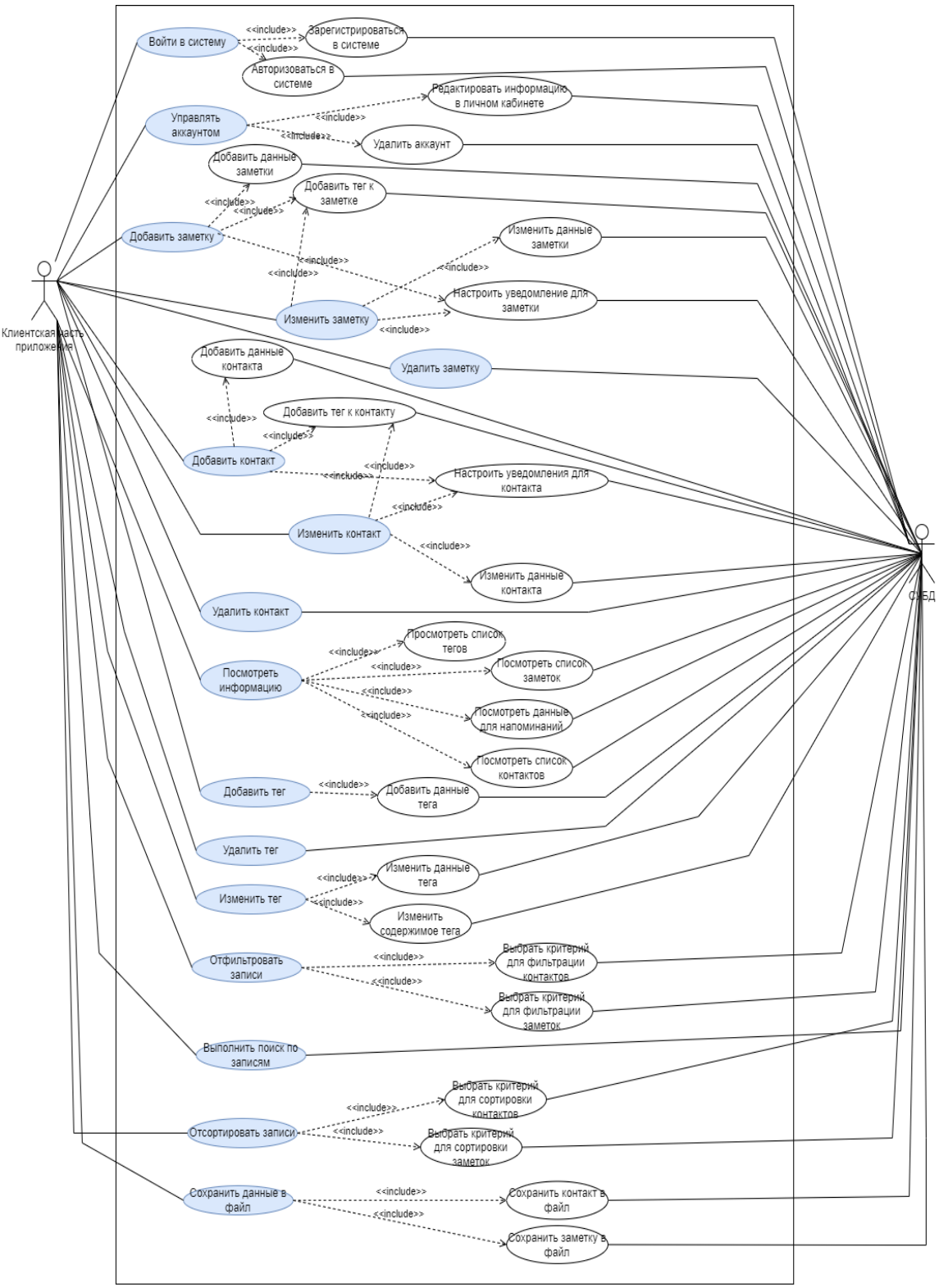


Рисунок 9 – Диаграмма вариантов использования приложения для ведения и учета контактов в электронной контактной книге

На основе диаграммы вариантов использования были составлены спецификации бизнес-процессов, представленные в таблице 6.

Таблица 6 – Спецификации бизнес-процессов серверного приложения электронной контактной книги с возможностью ведения заметок

№	Бизнес-процесс	Исполнитель	Входные данные		Выходные данные		Ограничения
			Поставщик	Содержание	Потребитель	Содержание	
1	2	3	4	5	6	7	8
1	Войти в систему	Клиентская часть приложения	Клиентская часть приложения	Логин и пароль	Клиентская часть приложения	Доступ к функционалу системы и данным аккаунта	
2	Управлять аккаунтом	Клиентская часть приложения	Клиентская часть приложения	Переход на страницу пользовательского аккаунта	Клиентская часть приложения	Измененные данные аккаунта	Пользователь должен быть авторизован
3	Добавить заметку	Клиентская часть приложения	Клиентская часть приложения	Переход на страницу создания заметки	Клиентская часть приложения	Страница созданной заметки	Пользователь должен быть авторизован
4	Изменить заметку	Клиентская часть приложения	Клиентская часть приложения	Страница конкретной заметки	Клиентская часть приложения	Страница измененной заметки	Заметка должна существовать
5	Удалить заметку	Клиентская часть приложения	Клиентская часть приложения	Страница конкретной заметки	Клиентская часть приложения	Обновленный список заметок	Заметка должна существовать до удаления
6	Добавить контакт	Клиентская часть приложения	Клиентская часть приложения	Переход на страницу создания контакта	Клиентская часть приложения	Страница созданного контакта	Пользователь должен быть авторизован
7	Изменить контакт	Клиентская часть приложения	Клиентская часть приложения	Страница конкретного контакта	Клиентская часть приложения	Страница контакта с измененными данными	Контакт должен существовать
8	Удалить контакт	Клиентская часть приложения	Клиентская часть приложения	Страница конкретного контакта	Клиентская часть приложения	Обновленный список контактов	Контакт должен существовать до удаления

Продолжение таблицы 6

1	2	3	4	5	6	7	8
9	Просмотреть информацию	Клиентская часть приложения	Клиентская часть приложения	Переход на страницу контактов/ заметок/ тегов	Клиентская часть приложения	Страница контактов/ заметок/ тегов	Пользователь должен быть авторизован
10	Добавить тег	Клиентская часть приложения	Клиентская часть приложения	Переход на страницу создания тега	Клиентская часть приложения	Страница созданного тега	Пользователь должен быть авторизован
11	Удалить тег	Клиентская часть приложения	Клиентская часть приложения	Переход на страницу конкретного тега	Клиентская часть приложения	Обновленный список тегов	Тег должен существовать до удаления
12	Изменить тег	Клиентская часть приложения	Клиентская часть приложения	Страница конкретного тега	Клиентская часть приложения	Страница измененного тега	Тег должен существовать
13	Отфильтровать записи	Клиентская часть приложения	Клиентская часть приложения	Список контактов/ заметок/ тегов	Клиентская часть приложения	Список контактов/ заметок/ тегов в соответствии с установленным фильтром	Хотя бы 1 контакт/ заметка/ тег должны существовать
14	Выполнить поиск по записям	Клиентская часть приложения	Клиентская часть приложения	Список контактов/ заметок/ тегов	Клиентская часть приложения	Список контактов/ заметок/ тегов, удовлетворяющих критериям поиска	Хотя бы 1 контакт/ заметка/ тег должны существовать
15	Отсортировать записи	Клиентская часть приложения	Клиентская часть приложения	Список контактов/ заметок/ тегов	Клиентская часть приложения	Отсортированный список контактов/ заметок/ тегов	Хотя бы 1 контакт/ заметка/ тег должны существовать
16	Сохранить данные в файл	Клиентская часть приложения	Клиентская часть приложения	Страница конкретного контакта/ заметки	Клиентская часть приложения	Файл, содержащий данные контакта/ заметки	Хотя бы 1 контакт/ заметка должны существовать

Кроме того, были разработаны сценарии наиболее важных бизнес-процессов. Сценарии выполнения для бизнес-процессов «Войти в систему» и «Управлять аккаунтом» представлены в таблицах 7 и 8 соответственно. Важно заметить, что для варианта использования «Войти в систему» предусловием служит то, что пользователь еще не был зарегистрирован или авторизован в системе. Для управления аккаунтом, наоборот, предварительная авторизация обязательна.

Таблица 7 – Сценарий выполнения варианта использования «Войти в систему»

Наименование пункта	Значение
Наименование	Войти/авторизоваться в системе
ID	1
Краткое описание	Пользователь регистрируется или авторизуется в приложении
Действующие лица	Клиентская часть приложения
Предусловие	Пользователь не был зарегистрирован/авторизован
Основной поток	1. Пользователь переходит на страницу входа в приложении; 2. Пользователь вводит данные для входа в существующий аккаунт (логин и пароль); 3. Пользователь получает доступ к функционалу приложения. Альтернативные потоки: 2.а. Пользователь вводит данные для регистрации (логин и пароль).
Постусловие	Пользователь вошел в приложение, ему доступен функционал

Таблица 8 – Сценарий выполнения варианта использования «Управлять аккаунтом»

Наименование пункта	Значение
Наименование	Управлять аккаунтом
ID	2
Краткое описание	Пользователь редактирует информацию в личном кабинете и удаляет аккаунт
Действующие лица	Клиентская часть приложения
Предусловие	Пользователь прошел авторизацию
Основной поток	1. Пользователь редактирует информацию в личном кабинете Альтернативные потоки: 1а. Пользователь удаляет аккаунт
Постусловие	Данные аккаунта отредактированы или аккаунт удален

Варианты использования «Войти в систему» и «Управлять аккаунтом», позволяют пользователю совершать различные действия с личным аккаунтом. Для начала работы в системе необходимо зарегистрироваться/авторизоваться в системе. Для этого следует ввести уникальный для системы адрес электронной почты и пароль. Далее пользователю становится доступен различный функционал с контактами, заметками и категориями, а также совершение операций с личным кабинетом. Кроме того, также предусматривается возможность выхода из системы, либо, при необходимости, удаления аккаунта. Важным аспектом является то, что действующим лицом для всех отображенных вариантов использования является клиентская часть приложения, из которой приходят соответствующие запросы.

Сценарии выполнения для бизнес-процессов «Добавить контакт», «Изменить контакт» и «Удалить контакт» представлены в таблицах 9, 10 и 11 соответственно.

Таблица 9 – Сценарий выполнения варианта использования «Добавить контакт»

Наименование пункта	Значение
Наименование	Добавить контакт
ID	6
Краткое описание	Создание нового контакта
Действующие лица	Клиентская часть приложения
Предусловие	Пользователь прошел авторизацию и перешел во вкладку «Контакты»
Основной поток	1. Добавить данные контакта 2. Добавить тег к контакту 3. Настроить уведомления для контакта
Постусловие	Контакт создан

Таблица 10 – Сценарий выполнения варианта использования «Изменить контакт»

Наименование пункта	Значение
Наименование	Изменить контакт
ID	7
Краткое описание	Пользователь редактирует данные контакта
Действующие лица	Клиентская часть приложения
Предусловие	Пользователь прошел авторизацию и выбрал контакт для редактирования
Основной поток	1. Добавить тег к контакту 2. Изменить данные контакта 3. Настроить уведомления для контакта
Постусловие	Контакт отредактирован

Таблица 11 – Сценарий выполнения варианта использования «Удалить контакт»

Наименование пункта	Значение
Наименование	Удалить контакт
ID	8
Краткое описание	Пользователь удаляет контакт
Действующие лица	Клиентская часть приложения
Предусловие	Пользователь прошел авторизацию и выбрал контакт для удаления
Основной поток	1. Удалить контакт
Постусловие	Контакт удален

Для вариантов использования «Добавить контакт», «Изменить контакт», «Удалить контакт» предусловием является авторизация в системе. Кроме того, для выполнения удаления и редактирования также необходимо выбрать нужный контакт.

Сценарии выполнения для бизнес-процессов «Просмотреть информацию», «Добавить заметку» и «Изменить заметку» представлены в таблицах 12, 13 и 14 соответственно.

Таблица 12 – Сценарий выполнения варианта использования «Просмотреть информацию»

Наименование пункта	Значение
1	2
Наименование	Просмотреть информацию
ID	9
Краткое описание	Пользователь просматривает информацию об имеющихся заметках, контактах и напоминаниях
Действующие лица	Клиентская часть приложения
Предусловие	Пользователь прошел авторизацию

Продолжение таблицы 12

1	2
Основной поток	1. Просмотреть список заметок. Альтернативный поток: 1а. Просмотреть список контактов; 1б. Просмотреть данные для напоминаний; 1в. Просмотреть данные для заметок.
Постусловие	Данные о заметках/контактах/напоминаниях просмотрены.

Таблица 13 – Сценарий выполнения варианта использования «Добавить заметку»

Наименование пункта	Значение
Наименование	Добавить заметку
ID	6
Краткое описание	Создание новой заметки
Действующие лица	Клиентская часть приложения
Предусловие	Пользователь прошел авторизацию и перешел во вкладку «Заметки»
Основной поток	1. Добавить данные заметки 2. Добавить тег к заметке 3. Настроить уведомления для заметки
Постусловие	Заметка создана

Таблица 14 – Сценарий выполнения варианта использования «Изменить заметку»

Наименование пункта	Значение
1	2
Наименование	Изменить заметку
ID	7

Продолжение таблицы 14

1	2
Краткое описание	Пользователь редактирует данные заметки
Действующие лица	Клиентская часть приложения
Предусловие	Пользователь прошел авторизацию и выбрал заметку для редактирования
Основной поток	1. Добавить тег к заметке 2. Изменить данные заметки 3. Настроить уведомления для заметки
Постусловие	Заметка отредактирована

Полученные данные будут использоваться при разработке концептуальной модели базы данных.

2.2 Разработка концептуальной модели данных

На основе анализа предметной области и результатов моделирования бизнес-процессов была разработана концептуальная модель базы данных, представленная на рисунке 10 [9].

Цель разработки исходной концептуальной модели – первоначальное структурирование информации в предметной области без учета деталей возможной реализации базы данных.

На концептуальной модели данных отображаются сущности и типы связей между ними. Так, например, связь «один-ко-многим» предполагает, что некоторый первый объект может быть связан с несколькими вторыми объектами. Между элементом и контактом и заметкой используется связь, обозначающая наследование. Это подразумевает под собой то, что контакты и заметки наследуются от элемента и содержат в себе его характеристики.

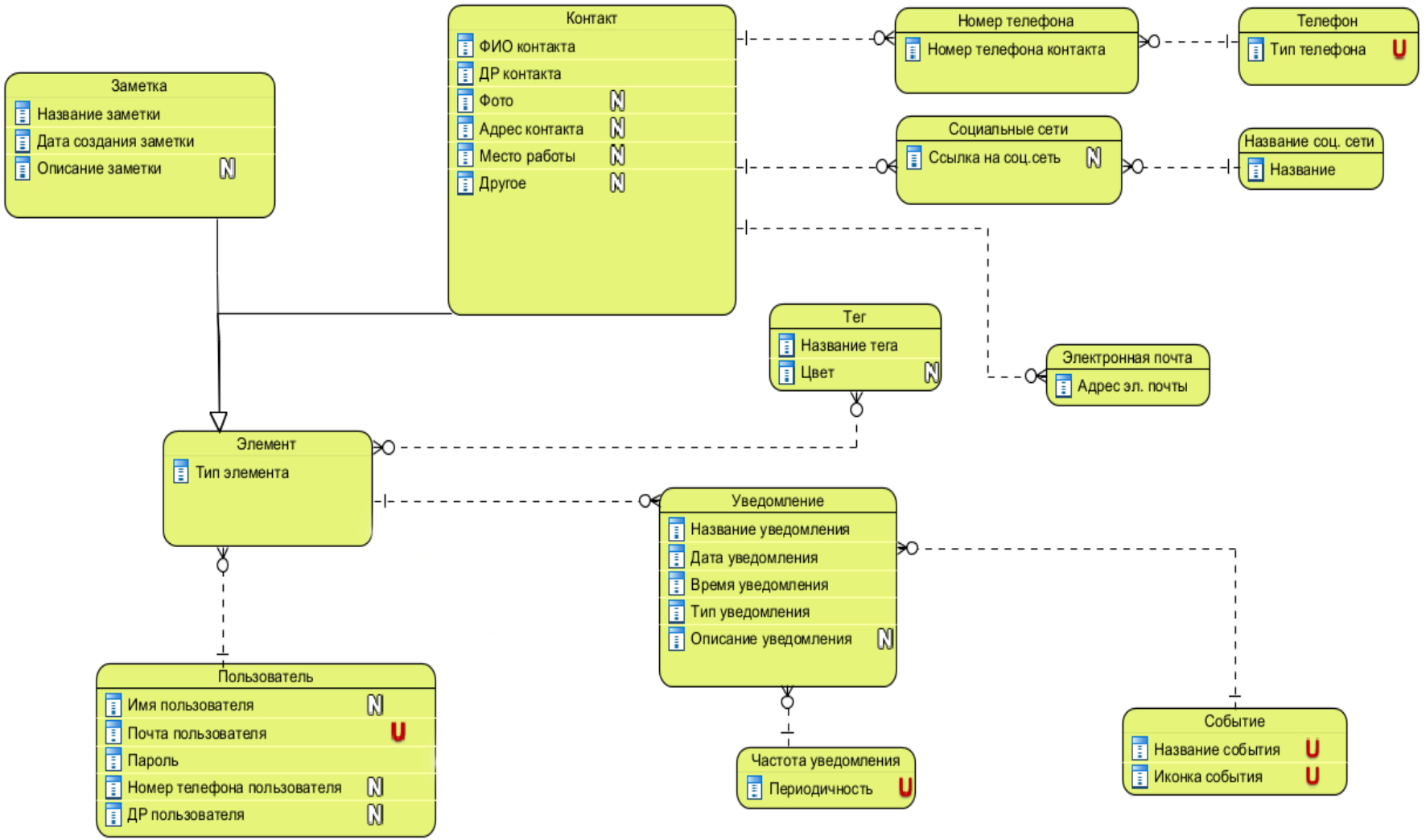


Рисунок 10 – Концептуальная модель базы данных

Спецификации атрибутов сущностей «Заметка», «Контакт», «Номер телефона» представлены в таблицах 15, 16, 17 соответственно. При этом номер телефона связан с контактом, то есть номера телефонов пользовательских контактов сохраняются в отдельных сущностях.

Таблица 15 – Атрибуты сущности «Заметка»

№	Название	Описание	Тип данных	Размер	Область допустимых значений	Возможность значения Null	Пример
1	Дата создания заметки	Дата создания заметки	Дата	10	«0 – 9», «.»	Нет	10.10.2023
2	Название заметки	Название заметки	Строковый			Нет	Новая заметка
3	Описание заметки	Содержание заметки	Строковый			Да	Пример содержания заметки

Таблица 16 – Атрибуты сущности «Контакт»

№	Название	Описание	Тип данных	Размер	Область допустимых значений	Возможность значения Null	Пример
1	Адрес контакта	Адрес проживания контакта	Строковый	50		Да	г. Пенза, ул. Кирова, 176
2	ДР контакта	День рождения контакта	Дата	10	01.01.1950 – 01.01.2023	Нет	13.06.2004
3	Другое	Доп. информация о контакте	Строковый	200		Да	Любит сливочное мороженое
4	Место работы	Место работы контакта	Строковый	50		Да	ООО «ИнтерПрайс»
5	ФИО контакта	ФИО контакта	Строковый	100	А-я	Нет	Федоров Сергей Сергеевич
6	Фото	Фотография	Двоичный объект	-		Да	

Таблица 17 – Атрибуты сущности «Номер телефона»

№	Название	Описание	Тип данных	Размер	Область допустимых значений	Возможность значения Null	Пример
1	Номер телефона контакта	Номер телефона контакта	Строковый	11	0 – 9	Нет	89476547382

Спецификации атрибутов сущностей «Пользователь», «Событие», «Социальная сеть» представлены в таблицах 18, 19, 20 соответственно.

Таблица 18 – Атрибуты сущности «Пользователь»

№	Название	Описание	Тип данных	Размер	Область допустимых значений	Возможность значения Null	Пример
1	ДР пользователя	День рождения пользователя	Дата	10	01.01.1950 – 01.01.2023	Да	12.01.2000
2	Имя пользователя	Имя пользователя	Строковый	100	«А-я»	Да	Иванов Иван
3	Номер телефона	Номер телефона пользователя	Строковый	11	0 – 9	Да	89479280883
4	Пароль	Пароль для входа	Строковый	20	«A-z», «0 – 9», «_», «@»	Нет	nastya123
5	Почта	Адрес электронной почты пользователя	Строковый	30	«A-z», «@», «.», «_», «-», «0 – 9»	Нет	mailmail@mail.ru

Таблица 19 – Атрибуты сущности «Событие»

№	Название	Описание	Тип данных	Размер	Область допустимых значений	Возможность значения Null	Пример
1	Иконка события	Символичное изображение события	Двоичный объект	100Кб		Нет	
2	Название события	Название события (день рождения, встреча, новый год)	Строковый	50	«А-я»	Нет	День рождения

Таблица 20 – Атрибуты сущности «Социальная сеть»

№	Название	Описание	Тип данных	Размер	Область допустимых значений	Возможность значения Null	Пример
1	Название соц. сети	Название социальной сети	Строковый	15	«А-я», «A-z»	Да	Telegram
2	Ссылка на соц. сеть	Ссылка на профиль контакта в социальной сети	Строковый	100	«А-я», «A-z»	Да	@ElenaForms

Пользователь является главным действующим лицом в разрабатываемом приложении электронной контактной книги с возможностью ведения заметок. Социальная сеть, как и номер телефона, привязана к контакту и хранит ссылки на социальные сети контакта. Аналогичным образом, события привязаны к уведомлениям.

Спецификации атрибутов сущностей «Тег», «Тип телефона», «Уведомление» представлены в таблицах 21, 22, 23 соответственно.

Таблица 21 – Атрибуты сущности «Тег»

№	Название	Описание	Тип данных	Размер	Область допустимых значений	Возможность значения Null	Пример
1	Название тега	Категория объектов в системе	Строковый	30	«А-я»	Нет	Работа
2	Цвет	Цвет	Строковый	20		Да	Красный

Таблица 22 – Атрибуты сущности «Тип телефона»

№	Название	Описание	Тип данных	Размер	Область допустимых значений	Возможность значения Null	Пример
1	Тип телефона	Тип телефона (домашний, рабочий, личный)	Строковый	15	«А-я»	Нет	Рабочий

Таблица 23 – Атрибуты сущности «Уведомление»

№	Название	Описание	Тип данных	Размер	Область допустимых значений	Возможность значения Null	Пример
1	Время уведомления	Время (часы и минуты) уведомления	Время	7	«0 – 9», «:»	Нет	15:00
2	Дата уведомления	Дата уведомления	Дата	10	«0 – 9», «.»	Нет	12.12.2023
3	Название уведомления	Название события для уведомления	Строковый	30		Нет	Встреча с Воландом
4	Описание уведомления	Описание уведомления (события)	Строковый	100		Нет	Патриаршие пруды, лавочка у белого здания
5	Тип уведомления	Тип уведомления системы (со звуком/без звука)	Булевый	1	0, 1	Нет	1 (со звуком)

Тег представляет собой некоторую группу контактов и заметок, которые связаны между собой по какому-либо признаку, заданному пользователем. Например, пользователь может создать тег «Работа» и поместить туда контакты коллег и заметки, связанные с рабочими проектами и встречами.

Спецификации атрибутов сущностей «Частота уведомления», «Электронная почта», «Элемент» представлены в таблицах 24, 25, 26 соответственно.

Таблица 24 – Атрибуты сущности «Частота уведомления»

№	Название	Описание	Тип данных	Размер	Область допустимых значений	Возможность значения Null	Пример
1	Периодичность	Частота отправки уведомления	Строковый	20	«А-я»	Нет	Не повторять

Таблица 25 – Атрибуты сущности «Электронная почта»

№	Название	Описание	Тип данных	Размер	Область допустимых значений	Возможность значения Null	Пример
1	Адрес эл. почты	Адрес электронной почты	Строковый	30	«A-z», «@», «.», «_», «-», «0 – 9»	Нет	markova@gmail.com

Таблица 26 – Атрибуты сущности «Элемент»

№	Название	Описание	Тип данных	Размер	Область допустимых значений	Возможность значения Null	Пример
1	Тип элемента	Элемент системы (заметка или контакт)	Булевый	1	0, 1	Нет	1 (заметка)

Описание связей между сущностями представлено в таблице 27.

Таблица 27 – Спецификации связей между сущностями

№	Тип сущности 1	Тип сущности 2	Семантика связи	Мощность	Обязательность
1	2	3	4	5	6
1	Заметка	Элемент	Каждый элемент является заметкой/ заметка может быть элементом	1:1 (наследование)	Обязательная
2	Контакт	Элемент	Каждый элемент является заметкой/ заметка может быть элементом	1:1 (наследование)	Обязательная
3	Пользователь	Элемент	У пользователя есть элемент/ Элемент имеет пользователя	1:M	Обязательная
4	Пользователь	Общий доступ	У пользователя есть доступ к информации/ Доступ к информации обязательно имеет владельца (пользователя)	1:M	Необязательная, так как пользователь может не иметь чужих элементов, к которым имеет доступ
5	Элемент	Общий доступ	Элемент может быть предоставлен для общего доступа/Общий доступ относится к определенному элементу	1:1	Необязательная, т. к. элемент может быть недоступен для общего доступа

Продолжение таблицы 27

1	2	3	4	5	6
6	Тег	Элемент	Тег есть у элемента/ элемент имеет тег	M:M	Необязательная, т. к. к тегу может быть не прикреплено ни одной заметки/контакта; контакты и заметки могут не иметь тега
7	Элемент	Уведомление	Элемент имеет уведомление/ Уведомление относится к элементу	1:M	Необязательная, т. к. элемент может не иметь уведомлений
8	Уведомление	Частота уведомления	Уведомление имеет частоту/ Частота связана с уведомлением	1:1	Обязательная
9	Уведомление	Событие	Уведомление связано с событием/ Событие прикреплено к уведомлению	1:1	Необязательная, т. к. уведомление может не относиться к событию
10	Контакт	Номер телефона	Контакт имеет номер телефона/ номер телефона принадлежит определенному контакту	1:M	Обязательная
11	Номер телефона	Телефон	Номер телефона относится к определенному типу телефона/Тип телефона связан с номером телефона	M:1	Необязательная, т. к. номер телефона может не относиться к какому-то определенному типу
12	Контакт	Социальная сеть	Контакт имеет социальную сеть/ Социальная сеть относится к определенному контакту	1:M	Необязательная, т. к. контакт может не иметь социальных сетей
13	Контакт	Электронная почта	Контакт имеет электронную почту/ Электронная почта принадлежит определенному контакту	1:M	Обязательная

Полученные результаты являются исходными данными для моделирования базы данных и разработки приложения информационной системы.

2.3 Обощенная архитектура системы электронной контактной книги с возможностью ведения заметок

Для разработки приложения электронной контактной книги с возможностью ведения заметок была выбрана клиент-серверная архитектура. Она обладает рядом преимуществ:

– масштабируемость;
– централизованное управление;
– надежность;
– безопасность.

При необходимости, клиент-серверное приложение можно масштабировать, что позволяет значительно улучшить производительность системы, а также реализовать обработку большого количества запросов от клиентов. При этом сервер является центральным узлом, не только контролирующим всю систему, но и обеспечивающим безопасность и управление доступом к данным. Благодаря такому подходу обеспечивается контроль доступа, а также защита пользовательских данных. Также важно отметить, что работа компонентов клиент-серверной архитектуры не связана между собой: отказ одного из модулей не затронет работоспособность другого [10].

2.4 Требования к аппаратно-программному обеспечению серверного приложения

Доступ к разрабатываемому серверному приложению обеспечивается через просмотр в веб-браузере. Согласно документациям используемых технологий разработки, корректное отображение обеспечивают следующие браузеры

– Google Chrome;
– Mozilla Firefox;
– Microsoft Edge;
– Safari (для MacOS);

– Opera.

Один из самых популярных и универсальных браузеров, который поддерживает большинство современных веб-технологий, является Google Chrome.

Кроме того, доступ к результатам разработки серверной части приложения электронной контактной книги с возможностью ведения заметок может быть осуществлен при помощи Postman [11]. Данное приложение представляет собой сервис для создания, тестирования, документирования, публикации и обслуживания API. Для его использования необходимо выполнить минимальные системные требования:

– операционная система Windows 7;
– 4 GB оперативной памяти;
– 200 МБ свободного места на жестком диске;
– Интернет-соединение.

Выполнение перечисленных минимальных условий для работы с Postman позволит достичь оптимальной производительности для доступа к результатам разработки серверной части электронной контактной книги с возможностью ведения заметок.

2.5 Обоснование выбора технологий и средств разработки серверного приложения

Разработка серверного приложения производится в среде Visual Studio [12]. Данная программа представляет собой комплексную интегрированную среду разработки для записи, редактирования, отладки и сборки кода. Кроме того, Visual Studio также позволяет развертывать разрабатываемые приложения. Немаловажно отметить, что данная среда разработки включает в себя компиляторы, средства завершения кода, управление версиями и расширения. Также Visual Studio обладает рядом преимуществ:

– простота освоения, в том числе, интуитивно понятный пользователю интерфейс;

– функциональность и возможность установки дополнительных расширений;

– кроссплатформенность;

– поддержка разных языков программирования;

– интеграция управления версиями;

– разработка с поддержкой искусственного интеллекта.

Также при разработке приложения была выбрана технология ASP .NET Core [13]. Данная среда позволяет выполняет следующие задачи:

– создавать веб-приложения и службы, серверные части мобильных приложений;

– выполнять развертывание в облаке или локальной среде;

– использовать избранные средства разработки в Windows, macOS и Linux.

Помимо перечисленных функций, ASP .NET Core обладает рядом преимуществ по сравнению с аналогами. В первую очередь, выделяется возможность создания единого решения для реализации пользовательского веб-интерфейса и веб-API. Кроме того, ASP .NET Core позволяет управлять параллельными версиями разрабатываемых приложений, а также обладает инструментарием, упрощающим процесс веб-разработки. Среди преимуществ немаловажными являются открытый код, ориентация на сообщество и возможность разработки и запуска в операционных системах Windows, macOS и Linux.

3 Разработка программного обеспечения серверного приложения

3.1 Разработка архитектуры серверного приложения

Для описания архитектуры разрабатываемой электронной контактной книги с возможностью ведения заметок была составлена диаграмма развертывания, представленная на рисунке 11 [14].

Диаграмма развертывания предназначена для визуализации архитектуры системы, а также отображения развертывания программных компонентов на аппаратном обеспечении. Диаграмма развертывания состоит из следующих частей:

– узел;
– компонент;
– артефакт;
– интерфейс.

Главным предназначением узла является развертывание артефактов для выполнения, при этом узел является некоторым физическим объектом. Кроме того, данный компонент также описывает связи между различными объектами в системе.

Артефакт – это спецификация конкретного реального объекта разрабатываемого программного обеспечения. В качестве таких объектов могут выступать исходные файлы, исполняемые файлы, файлы базы данных и т. п. Важно заметить, что артефакты должны располагаться на узлах.

Компоненты предназначены для представления программных модулей или артефактов. При этом основное отличие компонентов от артефактов состоит в том, что компоненты – это логические составляющие разрабатываемой системы, а артефакты – физические.

На представленной диаграмме все компоненты отображены как артефакты, так как они представляют собой конкретные реальные объекты, реализованные в системе.

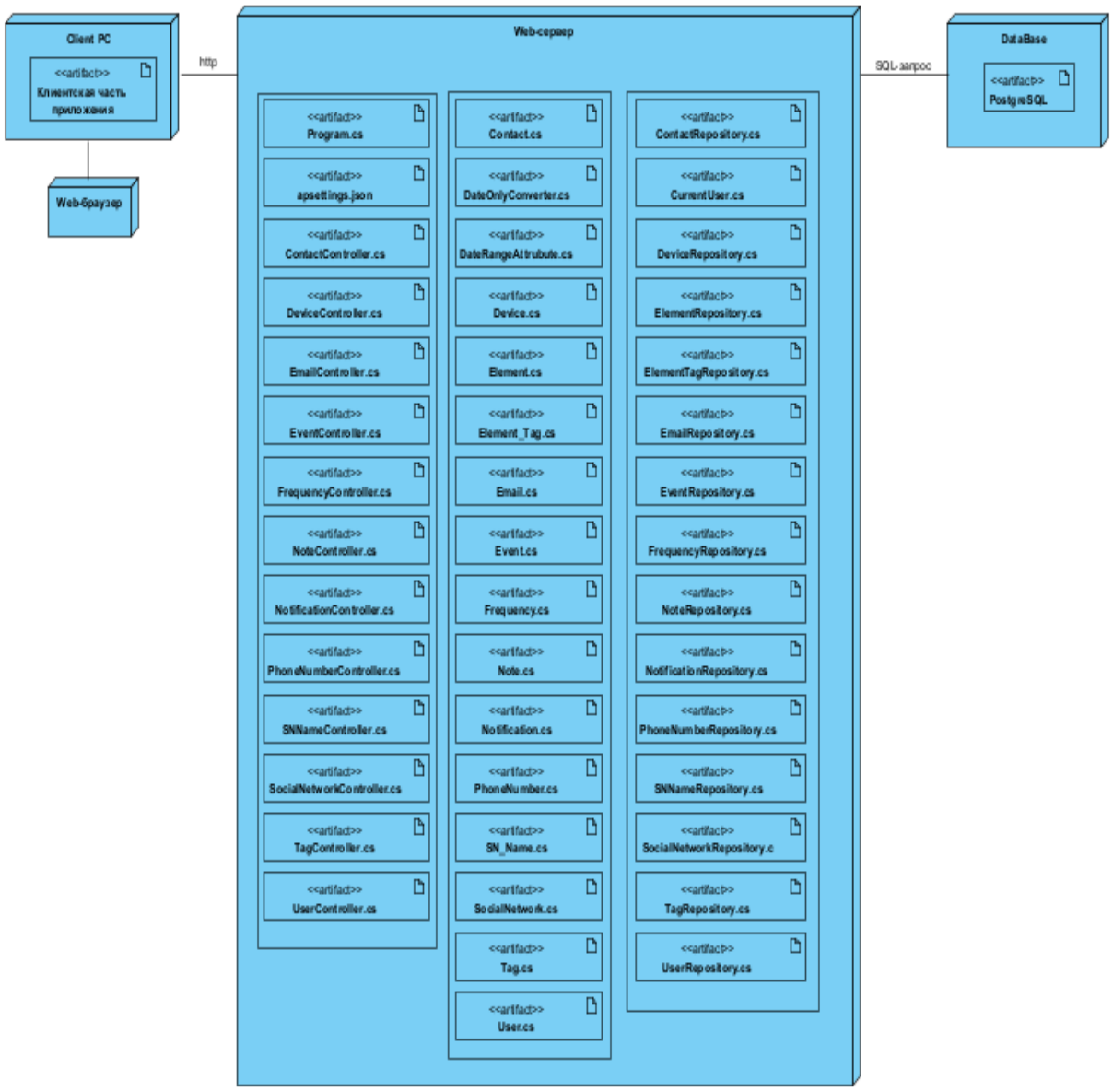


Рисунок 11 – Архитектура электронной контактной книги с возможностью ведения заметок

На разработанной диаграмме клиент-серверная архитектура приложения отображена при помощи трех узлов:

– клиентская часть – узел Client PC;

– серверная часть – узел Web-сервер;

– сервер базы данных – узел DataBase.

Каждый узел на диаграмме развертывания содержит ряд артефактов – физических файлов, используемых для функционирования разрабатываемой

системы. Описания элементов диаграммы развертывания представлены в таблице 28.

Таблица 28 – Спецификации элементов диаграммы развертывания

№	Название	Описание
1	2	3
1	Клиентская часть приложения	Артефакт, представляющий клиентскую часть приложения
2	Program.cs	Артефакт, содержащий основные настройки серверной части приложения
3	appsettings.json	Артефакт, содержащий настройки приложения, в т. ч. строку подключения к базе данных
4	ContactController.cs	Артефакт, содержащий контроллер для работы с контактами
5	DeviceController.cs	Артефакт, содержащий контроллер для работы с устройствами
6	EmailController.cs	Артефакт, содержащий контроллер для работы с адресами электронных почт контактов
7	EventController.cs	Артефакт, содержащий контроллер для работы с основными типами событий для уведомлений
8	FrequencyController.cs	Артефакт, содержащий контроллер для работы с периодами напоминаний
9	NoteController.cs	Артефакт, содержащий контроллер для работы с заметками
10	NotificationController.cs	Артефакт, содержащий контроллер для работы с уведомлениями
11	PhoneNumberController.cs	Артефакт, содержащий контроллер для работы с номерами телефонов контактов
12	SNNameController.cs	Артефакт, содержащий контроллер для работы с названиями социальных сетей
13	SocialNetworkController.cs	Артефакт, содержащий контроллер для работы с ссылками на социальные сети контактов
14	TagController.cs	Артефакт, содержащий контроллер для работы с тегами
15	UserController.cs	Артефакт, содержащий контроллер для работы с пользовательскими данными
16	Contact.cs	Артефакт, содержащий описание модели контакта
17	DateOnlyConverter.cs	Артефакт, содержащий настройки для конвертации даты после обработки в контроллере
18	DateRangeAttribute.cs	Артефакт, содержащий описание для проверки необходимого диапазона дат
19	Device.cs	Артефакт, содержащий описание модели устройства
20	Element.cs	Артефакт, содержащий описание модели элемента
21	Element_Tag.cs	Артефакт, содержащий описание модели связи между элементом и тегом
22	Email.cs	Артефакт, содержащий описание модели адреса электронной почты контакта
23	Event.cs	Артефакт, содержащий описание модели мероприятия
24	Frequency.cs	Артефакт, содержащий описание модели периодичности
25	Note.cs	Артефакт, содержащий описание модели заметки
26	Notification.cs	Артефакт, содержащий описание модели уведомления
27	PhoneNumber.cs	Артефакт, содержащий описание модели номера телефона контакта
28	SN_Name.cs	Артефакт, содержащий описание модели названия социальной сети
29	SocialNetwork.cs	Артефакт, содержащий описание модели ссылки на социальную сеть
30	Tag.cs	Артефакт, содержащий описание модели тега
31	User.cs	Артефакт, содержащий описание модели пользователя
32	ContactRepository.cs	Артефакт, содержащий репозиторий контакта для связи с базой данных
33	CurrentUser.cs	Артефакт, содержащий данные о текущем пользователе системы
34	DeviceRepository.cs	Артефакт, содержащий репозиторий устройств для связи с базой данных

Продолжение таблицы 28

1	2	3
35	ElementRepository.cs	Артефакт, содержащий репозиторий элемента для связи с базой данных
36	ElementTagRepository.cs	Артефакт, содержащий репозиторий связи элемента с тегом для связи с базой данных
37	EmailRepository.cs	Артефакт, содержащий репозиторий электронных почт контакта для связи с базой данных
38	EventRepository.cs	Артефакт, содержащий репозиторий типов событий для связи с базой данных
39	FrequencyRepository.cs	Артефакт, содержащий репозиторий периодичности напоминаний для связи с базой данных
40	NoteRepository.cs	Артефакт, содержащий репозиторий заметок для связи с базой данных
41	NotificationRepository.cs	Артефакт, содержащий репозиторий уведомлений для связи с базой данных
42	PhoneNumberRepository.cs	Артефакт, содержащий репозиторий номеров телефонов контактов для связи с базой данных
43	SNNameRepository.cs	Артефакт, содержащий репозиторий названий социальных сетей для связи с базой данных
44	SocialNetworkRepository.cs	Артефакт, содержащий репозиторий ссылок на социальные сети контактов для связи с базой данных
45	TagRepository.cs	Артефакт, содержащий репозиторий тегов для связи с базой данных
46	UserRepository.cs	Артефакт, содержащий репозиторий пользователей для связи с базой данных
47	PostgreSQL	Артефакт, представляющий собой базу данных для разрабатываемого приложения электронной контактной книги с возможностью ведения заметок

В качестве связей между клиентской и серверной частью выступают HTTP-запросы, а для соединения сервера и базы данных используются SQL-запросы.

3.2 Разработка физической модели базы данных

Физическая модель базы данных представляет собой модель базы данных, которая определяет способ и параметры результирующего представления данных, обеспечивающие дальнейшее корректное создание базы данных [15]. Физическая модель базы данных приложения электронной контактной книги с возможностью ведения заметок представлена на рисунке 12.

В отношениях были заданы уникальные идентификаторы, а также типы данных, которые впоследствии будут использоваться для реализации базы данных. Кроме того, на физической модели базы данных отображена допустимая размерность каждого атрибута таблицы.

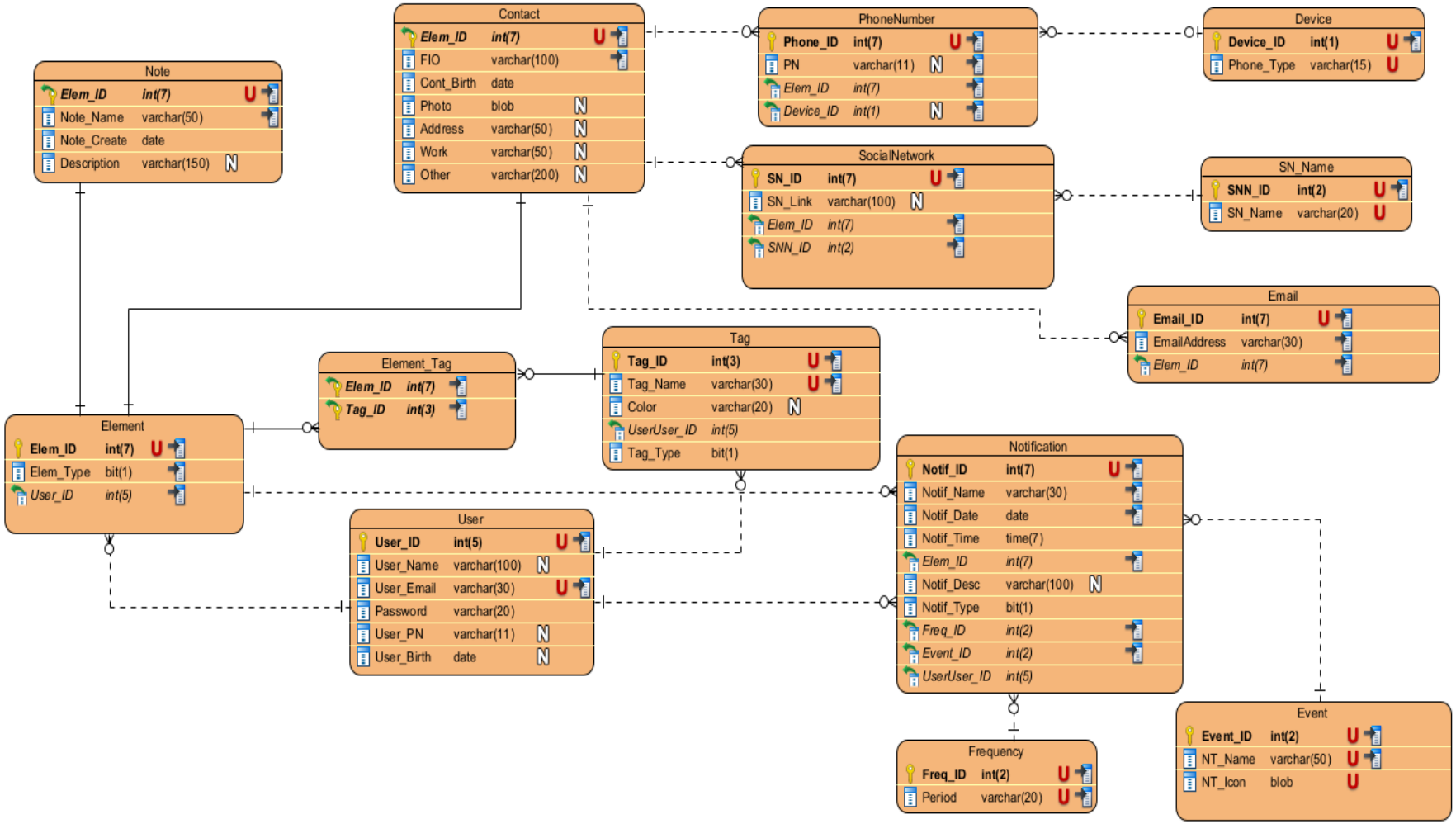


Рисунок 12 – Физическая модель базы данных

Описание отношений представлено в таблице 29.

Таблица 29 – Спецификация отношений физической модели базы данных

№ п/п	Наименование	Описание	Категория
1	Заметка	Содержит сведения о заметках (название, описание, дата создания)	Стержневая Характеристическая
2	Контакт	Содержит сведения о контактах (ФИО, дата рождения, адрес и др.)	Стержневая Характеристическая
3	Элемент	Содержит информацию о типах элементов в системе (контакт/заметка)	Стержневая
4	Пользователь	Содержит сведения о пользователе (имя, почта, номер телефона и др.)	Стержневая
6	Уведомление	Содержит сведения о напоминаниях в системе	Ассоциативная
7	Тег	Содержит сведения о категориях элементов	Ассоциативная
8	Частота уведомления	Содержит информацию о периодичности отправки уведомления конкретного события	Характеристическая Справочная
9	Событие	Содержит краткую информацию о событии	Характеристическая Справочная
10	Номер телефона	Содержит номера телефонов	Стержневая
11	Телефон	Содержит типы телефонов	Характеристическая Справочная
12	Социальная сеть	Содержит информацию о социальных сетях контакта	Стержневая
13	Электронная почта	Содержит информацию об электронных почтах контакта	Стержневая

Спецификации атрибутов отношений «Контакт», «Устройство», «Элемент» представлены в таблицах 30, 31 и 32 соответственно.

Таблица 30 – Атрибуты отношения «Контакт»

Идентификатор столбца	Содержательное описание	Тип данных	Размерность	Область допустимых значений	Значение Null	Роль	Пример
1	2	3	4	5	6	7	8
Address	Адрес жительства контакта	varchar	50		Да		г. Пенза, ул. Кирова, 176
Cont_Birth	День рождения контакта	date	10	01.01.1950 – 01.01.2023	Нет		13.06.2004
Elem_ID	Уникальный идентификатор контакта	int	7	1 – 9999999	Нет	PK	367

Продолжение таблицы 30

1	2	3	4	5	6	7	8
FIO	ФИО контакта	varchar	100	А-я	Нет		Федоров Сергей Сергеевич
Other	Раздел «Другое» для хранения прочей информации	varchar	200		Да		Любит сливочное мороженое
Photo	Фото контакта	blob	0		Да		
Work	Место работы контакта	varchar	50		Да		ООО «ИнтерПрай»

Таблица 31 – Атрибуты отношения «Устройство»

Идентификатор столбца	Содержательное описание	Тип данных	Размерность	Область допустимых значений	Значение Null	Роль	Пример
Device_ID	Уникальный идентификатор устройства	int	2	1 – 5	Нет	PK	37
Phone_Type	Тип устройства	varchar	15	«А-я»	Да		Рабочий

Таблица 32 – Атрибуты отношения «Элемент»

Идентификатор столбца	Содержательное описание	Тип данных	Размерность	Область допустимых значений	Значение Null	Роль	Пример
Elem_ID	Уникальный идентификатор элемента	int	7	1 – 9999999	Нет	PK	3838
Elem_Type	Тип элемента (заметка или контакт)	bit	1	0, 1	Нет		1
User_ID	Уникальный идентификатор создателя элемента	int	5	1 – 99999	Нет	FK	3

В отличие от концептуальной модели базы данных, в разработанной физической модели были добавлены такие атрибуты, как Elem_ID для контакта, Device_ID для типа номера телефона и Elem_ID для элемента. Эти атрибуты являются уникальными идентификаторами для элементов.

Спецификации атрибутов отношений «Электронная почта», «Событие» представлены в таблицах 33 и 34 соответственно.

Таблица 33 – Атрибуты отношения «Электронная почта»

Идентификатор столбца	Содержательное описание	Тип данных	Размерность	Область допустимых значений	Значение Null	Роль	Пример
Elem_ID	Идентификатор элемента, к которому привязан адрес электронной почты	int	7	1 – 9999999	Нет	FK	299
Email_ID	Уникальный идентификатор адреса электронной почты	int	7	1 – 9999999	Нет	PK	3738
EmailAddress	Адрес электронной почты	varchar	30	«A-z», «@», «.», «_», «-», «0 – 9»	Нет		markova@gmail.com

Таблица 34 – Атрибуты отношения «Событие»

Идентификатор столбца	Содержательное описание	Тип данных	Размерность	Область допустимых значений	Значение Null	Роль	Пример
Event_ID	Уникальный идентификатор события	int	2	1 – 20	Нет	PK	9
NT_Name	Название события	varchar	50	«А-я»	Нет		День рождения

Для отношения электронной почты уникальным идентификатором выступает Email_ID, а для события – Event_ID. Все адреса электронных почт контактов хранятся в отдельном отношении для удобства использования и хранения. Для обеспечения данной особенности необходимо внести в отношение электронной почты атрибут Elem_ID, отвечающий за хранение уникального идентификатора контакта, которому принадлежит заданная электронная почта.

Спецификации атрибутов отношений «Заметка», «Уведомление» представлены в таблицах 35 и 36 соответственно.

Таблица 35 – Атрибуты отношения «Заметка»

Идентификатор столбца	Содержательное описание	Тип данных	Размерность	Область допустимых значений	Значение Null	Роль	Пример
1	2	3	4	5	6	7	8
Description	Описание заметки	varchar	150		Да		Пример для заметки

Продолжение таблицы 35

1	2	3	4	5	6	7	8
Elem_ID	Уникальный идентификатор заметки	int	7	1 – 9999999	Нет	PK	6
Note_Create	Дата создания заметки	date	10	«0 – 9», «.»	Нет		10.10.2023
Note_Name	Название заметки	varchar	50		Нет		Новая заметка

Таблица 36 – Атрибуты отношения «Уведомление»

Идентификатор столбца	Содержательное описание	Тип данных	Размерность	Область допустимых значений	Значение Null	Роль	Пример
Elem_ID	Идентификатор элемента	int	7	1 – 9999999	Нет	FK	39834
Event_ID	Идентификатор события	int	2	1 – 20	Нет	FK	98
Freq_ID	Идентификатор частоты уведомления	int	2	1 – 10	Нет	FK	34
Notif_Date	Дата уведомления	date	10	«0 – 9», «.»	Нет		12.12.2023
Notif_Desc	Описание уведомления	varchar	100		Нет		Лавочка у белого здания
Notif_ID	Уникальный идентификатор уведомления	int	7	1 – 9999999	Нет	PK	47382
Notif_Name	Название уведомления	varchar	30		Нет		Встреча с Воландом
Notif_Time	Время уведомления	time	7	«0 – 9», «:»	Нет		15:00
Notif_Type	Тип уведомления (со звуком или без)	bit	1	0, 1	Нет		1 (со звуком)
User_ID	Идентификатор пользователя	int	5	1 – 99999	Нет	FK	32038

Пользовательская заметка также в качестве атрибута уникального идентификатора содержит Elem_ID. Отношение уведомления содержит ссылки сразу на несколько других отношений. Так, например, внешний ключ User_ID является уникальным идентификатором пользователя, который создал данное напоминание. Freq_ID служит внешним ключом для определения частоты напоминания, при этом ссылаясь на справочную сущность.

Спецификации атрибутов отношений «Номер телефона», «Социальная сеть» и «Тег» представлены в таблицах 37, 38 и 39 соответственно.

Таблица 37 – Атрибуты отношения «Номер телефона»

Идентификатор столбца	Содержательное описание	Тип данных	Размерность	Область допустимых значений	Значение Null	Роль	Пример
Device_ID	Идентификатор типа устройства	int	2	1 – 99	Да	FK	98
Elem_ID	Идентификатор элемента	int	7	1 – 9999999	Нет	FK	28
Phone_ID	Уникальный идентификатор номера телефона	int	7	1 – 9999999	Нет	PK	6
PN	Номер телефона	varchar	11	«0 – 9»	Да		89476547382

Таблица 38 – Атрибуты отношения «Социальная сеть»

Идентификатор столбца	Содержательное описание	Тип данных	Размерность	Область допустимых значений	Значение Null	Роль	Пример
Elem_ID	Идентификатор элемента	int	7	1 – 9999999	Да	FK	2837
SN_ID	Уникальный идентификатор социальной сети	int	7	1 – 9999999	Нет	PK	3738
SN_Link	Ссылка на аккаунт в социальной сети	varchar	100	«А-я», «A-z»	Нет		@ElenaForms
SN_Name	Название социальной сети	varchar	15	«А-я», «A-z»	Да		Telegram

Таблица 39 – Атрибуты отношения «Тег»

Идентификатор столбца	Содержательное описание	Тип данных	Размерность	Область допустимых значений	Значение Null	Роль	Пример
Color	Цвет тега	varchar	20		Да		красный
Tag_ID	Уникальный идентификатор тега	int	3	1 – 999	Нет	PK	45
Tag_Name	Название тега	varchar	30	«А-я»	Нет		Работа
User_ID	Идентификатор пользователя	int	5	1 – 99999	Нет	FK	32038

Отношение номера телефона хранит данные о всех номерах телефонов контактов, поэтому для связи между данными отношениями введен атрибут Elem_ID, выступающий в роли внешнего ключа. Для каждого номера телефона можно определить его тип: домашний, рабочий и т. д. Данная особенность поддерживается за счет введения атрибута Device_ID, как и в предыдущем

случае, в роли внешнего ключа. Социальная сеть строится по схожему принципу: в отношении содержатся атрибуты для связи с отношениями контакта и названия социальной сети, выступающими в роли внешний ключей.

Спецификации атрибутов отношений «Частота», «Пользователь» представлены в таблицах 40 и 41 соответственно.

Таблица 40 – Атрибуты отношения «Частота»

Идентификатор столбца	Содержательное описание	Тип данных	Размерность	Область допустимых значений	Значение Null	Роль	Пример
Freq_ID	Уникальный идентификатор частоты уведомления	Int	2	1 – 20	Нет	PK	27
Period	Периодичность уведомления	varchar	20	«А-я»	Нет		Не повторять

Таблица 41 – Атрибуты отношения «Пользователь»

Идентификатор столбца	Содержательное описание	Тип данных	Размерность	Область допустимых значений	Значение Null	Роль	Пример
Password	Пароль	varchar	20	«A-z», «0 – 9», «_», «@»	Нет		Nastya123
User_Birth	Дата рождения пользователя	date	10	01.01.1950 – 01.01.2023	Да		12.01.2000
User_Email	Адрес электронной почты пользователя	varchar	30	«A-z», «@», «.», «_», «-», «0 – 9»	Нет		mailmail@mail.ru
User_ID	Уникальный идентификатор пользователя	int	5	1 – 99999	Нет	PK	32038
User_Name	Имя пользователя	varchar	100	«А-я»	Да		Иванов Иван
User_PN	Номер телефона пользователя	varchar	11	0 – 9	Да		89479280883

Описание связей между сущностями представлено в таблице 42.

Таблица 42 – Спецификации связей между сущностями

№	Тип сущности 1	Тип сущности 2	Семантика связи	Мощность	Обязательность
1	2	3	4	5	6
1	Заметка	Элемент	Каждый элемент является заметкой/ заметка может быть элементом	1:1 (идентификационная связь)	Обязательная
2	Контакт	Элемент	Каждый элемент является заметкой/ заметка может быть элементом	1:1 (идентификационная связь)	Обязательная

Продолжение таблицы 42

1	2	3	4	5	6
3	Пользователь	Элемент	У пользователя есть элемент/ Элемент имеет пользователя	1:M	Обязательная
4	Пользователь	Напоминание	У пользователя есть напоминания/ Напоминание имеет пользователя	1:M	Обязательная, т. к. каждое напоминание должно иметь пользователя
5	Пользователь	Тег	У пользователя есть теги/ Тег имеет пользователя	1:M	Обязательная, т. к. каждый тег должен иметь пользователя
6	Тег	Элемент	Тег есть у элемента/ элемент имеет тег	M:M	Необязательная, т. к. к тегу может быть не прикреплено ни одной заметки/контакта; контакты и заметки могут не иметь тега
7	Элемент	Уведомление	Элемент имеет уведомление/ Уведомление относится к элементу	1:M	Необязательная, т. к. элемент может не иметь уведомлений
8	Уведомление	Частота уведомления	Уведомление имеет частоту/ Частота связана с уведомлением	1:1	Обязательная
9	Уведомление	Событие	Уведомление связано с событием/ Событие прикреплено к уведомлению	1:1	Необязательная, т. к. уведомление может не относиться к событию
10	Контакт	Номер телефона	Контакт имеет номер телефона/ номер телефона принадлежит определенному контакту	1:M	Обязательная
11	Номер телефона	Телефон	Номер телефона относится к определенному типу телефона/Тип телефона связан с номером телефона	M:1	Необязательная, т. к. номер телефона может не относиться к какому-то определенному типу
12	Контакт	Социальная сеть	Контакт имеет социальную сеть/ Социальная сеть относится к определенному контакту	1:M	Необязательная, т. к. контакт может не иметь социальных сетей
13	Контакт	Электронная почта	Контакт имеет электронную почту/ Электронная почта принадлежит определенному контакту	1:M	Обязательная

Кроме того, на основе разработанной физической модели базы данных был сгенерирован скрипт на PostgreSQL для создания объектов базы данных [16]. Фрагмент скрипта для создания таблиц «Пользователь» и «Контакт» базы данных приведен в листинге 1. Полный скрипт для создания базы данных содержится в приложении Б.

Листинг 1 – Создание таблиц «Пользователь» и «Контакт» в базе данных PostgreSQL

```
CREATE TABLE User (
User_ID SERIAL NOT NULL,
User_Name varchar(100),
User_Email varchar(30) NOT NULL,
Password varchar(20) NOT NULL,
User_PN varchar(12),
User_Birth date,
PRIMARY KEY (User_ID));

CREATE TABLE Contact (
Elem_ID int4 NOT NULL,
FIO varchar(100) NOT NULL,
Cont_Birth date NOT NULL,
Photo bytea,
Address varchar(50),
Work varchar(50),
Other varchar(200),
PRIMARY KEY (Elem_ID));
```

В приведенном выше фрагменте кода указано создание таблиц «Пользователь» и «Контакт». При создании базы данных обязательно определяются поля, которые выполняют роль уникальных идентификаторов сущностей. Кроме того, также сразу устанавливается тип данных для каждого поля, определяются ограничения на максимальную длину, а также возможность нулевого значения при занесении новых данных в таблицу и их обновления.

3.3 Разработка программного обеспечения доступа к данным

Для доступа к базе данных PostgreSQL в разрабатываемой серверной части приложения электронной контактной книги с возможностью ведения заметок используется Entity Framework Core – самое популярное средство взаимодействия с реляционными базами данных на платформе ASP .NET Core [17]. Кроме того, Entity Framework Core представляет собой object-relational

mapper (ORM) – инструмент, необходимый для установления соотношения между классами, используемыми в коде программы, и реляционной моделью базы данных. Например, связь класса «PhoneNumber» с таблицей базы данных «PhoneNumber» представлена в листинге 2. Полный листинг кода серверного приложения содержится в приложении В.

Листинг 2 – Связь класса «PhoneNumber» с таблицей базы данных «PhoneNumber»

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

     public DbSet<PhoneNumber> PhoneNumbers { get; set; }
}
```

В данном фрагменте кода класс PhoneNumberRepository наследуется от DbContext. DbContext используется в приложении для взаимодействия с базой данных и выполняет такие задачи, как представление набора объектов, отслеживание изменений и выполнение различных операций с базой данных. После получения требуемого набора данных из таблицы PhoneNumber происходит их запись в коллекцию DbSet. Данная коллекция используется для запроса, вставки, обновления и удаления элементов.

Для получения данных из базы данных в контроллерах используются LINQ-запросы [18]. С их помощью происходит формирование интуитивно понятных запросов, по форме близких к языку SQL. Далее при помощи средств Entity Framework Core происходит трансляция таких запросов на требуемый язык для используемого источника данных – в данном случае, для PostgreSQL. В листинге 3 приведен фрагмент кода из контроллера PhoneNumberController.cs для получения списка номеров телефонов заданного контакта.

Листинг 3 – Получение списка номеров телефонов заданного контакта в контроллере PhoneNumberController.cs

```
private readonly PhoneNumberRepository _phonenumbers;
```

```
public IActionResult GetListPhones([Required] int contactId)
{
     var phones = _phonenumbers.PhoneNumbers.Where(e => e.elem_id
== contactId).ToList();
     return new JsonResult(phones) { StatusCode = 200 };
}
```

В начале данного фрагмента кода происходит определение объекта класса PhoneNumberRepository, то есть, получение списка всех имеющихся в базе данных номеров телефонов. Далее определяется функция GetListPhones, получающая в качестве аргумента уникальный идентификатор контакта, номера телефонов которого требуется найти. Затем в переменную phones записывается результат LINQ-запроса, преобразованный к списку. Стоит обратить внимание на вызванный в запросе метод Where, в качестве параметров которому передается проверка соответствия идентификатора контакта в таблице номера телефона заданному. Таким образом, сначала произойдет трансляция и отправка запроса в базу данных PostgreSQL, затем – получение ответа от базы данных, и, как результат, материализация результатов запроса в C#-список.

Для реализации добавления данных в базу данных требуется сначала создать новый экземпляр требуемого класса, причем при выполнении данной операции происходят проверки на соответствие данных требуемым в базе данных. Пример проверки валидности данных представлен в листинге 4.

Листинг 4 – Проверка валидности данных для номера телефона контакта

```
[Table("phonenumber")]
public class PhoneNumber
{
     [Key]
     [DatabaseGenerated(DatabaseGeneratedOption.Identity)]
     public int phone_id { get; set; }
     [Required(ErrorMessage = "Номер телефона - обязательное
поле")]
     [RegularExpression(@"^((\+7|7|8)+([0 - 9]){10})$",
ErrorMessage = "Номер телефона введен некорректно")]
     [MaxLength(12, ErrorMessage = "максимальная длина номера
телефона - 12 символов")]
     public string pn { get; set; }
     [Required(ErrorMessage = "Идентификатор контакта для номера
телефона - обязательное поле")]
     public int elem_id { get; set; }
     public int? device_id { get; set; }
}
```

В таблице PhoneNumber разработанной базы данных PostgreSQL содержатся некоторые ограничения на вводимые поля: ограничение на ввод корректного номера телефона и его длины, а также ограничение на ввод пустых полей. В листинге класса PhoneNumber, приведенном выше, происходит проверка соответствия всех полей требуемым стандартам перед их отправкой в базу данных. В случае отсутствия таких проверок, при некорректном вводе каких-либо значений происходит ошибка при добавлении/изменении полей в базе данных, что может повлечь за собой некорректную работу всей серверной части приложения. Таким образом, проверка Required позволяет установить, были ли введены обязательные для базы данных поля, и при отрицательном результате происходит вывод соответствующего сообщения об ошибке. RegularExpression осуществляет проверку введенного номера телефона на соответствие заданному шаблону, допустимому для базы данных, а MaxLength – проверку на максимальную длину для введенного значения. Таким образом, при некорректном вводе выводятся ошибки с указанием того, что требуется исправить для корректного добавления нового значения в базу данных или его изменения.

Для добавления данных в базу данных используются строки, представленные в листинге 5.

Листинг 5 – Добавление номеров телефонов контакта в базу данных

```
_phonenumbers.PhoneNumbers.Add(phone);
_phonenumbers.SaveChanges();
```

При вызове SaveChanges в случае добавления новых данных происходит формирование соответствующего INSERT запроса в базу данных, а в случае обновления существующих данных – формирование UPDATE запроса.

При этом стоит отметить еще один важный аспект, строго необходимый для работы с базой данных PostgreSQL при разработке программного кода. Для работы с данными требуется обозначить строку подключения к базе данных, содержащую название базы данных, имя пользователя, пароль и прочие необходимые сведения. В разрабатываемом коде данная строка находится в файле appsettings.json и имеет синтаксис, представленный в листинге 6.

Листинг 6 – Строка подключения к базе данных

```
"UserConnection":
"Server=localhost;Port=5432;Database=bihd_labs;Username=postgres;P
assword=nastya123"
```

Далее, для доступа к данным таблиц базы данных необходимо связать созданные репозитории со строкой подключения. Это происходит в файле Program.cs при помощи вызова метода, описанного в листинге 7.

Листинг 7 – Создание связи между репозиторием и строкой подключения

```
builder.Services.AddDbContext<PhoneNumberRepository>(options =>
options.UseNpgsql(builder.Configuration.GetConnectionString("UserC
onnection")));
```

Таким образом происходит привязка всех созданных репозиториев со строкой подключения к базе данных, после чего открывается доступ для работы с данными.

3.4 Разработка программного интерфейса с клиентским приложением

В разрабатываемом приложении электронной контактной книги с возможностью ведения заметок для связи между серверной и клиентской частями приложения был использован REST API [19].

REST (Representational state transfer) представляет собой архитектурный стиль проектирования API с использованием протокола HTTP. Таким образом, взаимодействие REST API осуществляется при помощи HTTP-запросов, при этом выполняются операции создания, обновления, чтения и удаления требуемых данных. Для этого существуют четыре метода, определяющие порядок действий с требуемым ресурсом:

– POST – создание ресурса;
– GET – получение ресурса;
– PUT – обновление ресурса;
– DELETE – удаление ресурса.

REST API не имеет привязки к какому-либо языку программирования, платформе или технологии, однако для его использования необходимо учитывать некоторые архитектурные ограничения, предъявляемые к

разрабатываемому приложению. В первую очередь, приложение, использующее REST API, должно иметь клиент-серверную архитектуру. При таком подходе клиент и сервер являются независимыми друг от друга составляющими и, соответственно, могут развиваться отдельно, что делает REST API более гибким и масштабируемым. Помимо этого, одним из принципов REST API является использование единого способа взаимодействия с сервером, то есть, унифицированного интерфейса. Стоит также отметить, что сервер не содержит никакой информации о клиенте. Каждый запрос включает всю необходимую информацию для обработки, что позволяет полностью разделить клиентскую и серверную части приложения.

Добавление REST API в контроллеры проекта ASP .NET Core MVC начинается со строк, представленный в листинге 8 [20].

Листинг 8 – Добавление REST API в контроллеры проекта

```
[Route("[controller]")]
[ApiController]
```

Первая строка представляет собой атрибут маршрутизации, при помощи которого формируется итоговый URL-адрес. Так, например, для контроллера PhoneNumberController, содержащего методы для работы с номерами телефонов контакта, в результате будет автоматически сгенерирован URL-адрес «/PhoneNumber». Вторая строка предназначена для указания, что создаваемый контроллер является API-контроллером, использующим специальные правила для обработки запросов.

Далее внутри контроллера необходимо реализовать методы обработки требуемого ресурса: GET, POST, PUT, DELETE. Каждому из заданных методов требуется передать соответствующий атрибут, содержащий, в том числе, название метода и, в некоторых случаях, уникальный идентификатор требуемого элемента. Для контроллера, осуществляющего работу с номера телефонов контакта, данные атрибуты реализуются при помощи строк, представленных в листинге 9.

Листинг 9 – Реализация методов обработки требуемого ресурса

```
[HttpGet("GetListPhones")]
```

```
[HttpGet("GetOnePhone/{id}")]
[HttpPost("CreatePhoneNumber")]
[HttpPut("UpdatePhoneNumber/{id}")]
[HttpDelete("DeletePhoneNumber/{id}")]
```

В результате реализации данных методов пользователю становится доступен программный интерфейс, отображенный на рисунке 13.

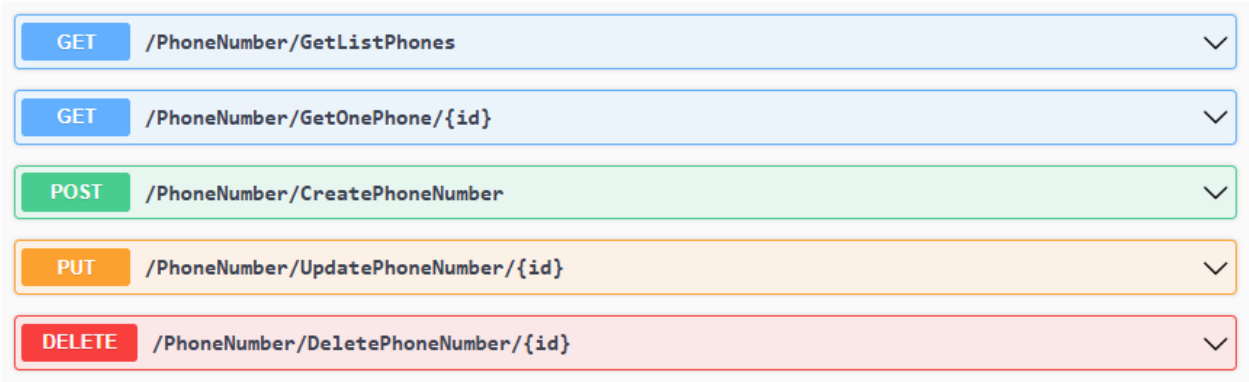


Рисунок 13 – Реализация программного интерфейса для обработки номеров телефонов контакта с клиентским приложением

Для метода GET было создано две реализации: в первом случае результатом является список всех номеров телефонов конкретного контакта, а во втором случае результатом работы метода является конкретный номер телефона, уникальный идентификатор которого выбирается клиентом и отображается, в том числе, в URL-адресе запроса. Для методов PUT и DELETE также необходимо получить через HTTP-запрос уникальный идентификатор конкретного номера телефона, данные которого впоследствии подвергнутся изменению или удалению.

Результатом работы каждого из методов контроллера является JSON-ответ, содержащий информацию о результате выполненного действия. Метод DELETE для контроллера номеров телефонов контакта представлен в листинге 10.

Листинг 10 – Метод DELETE для контроллера номеров телефонов контакта

```
[HttpDelete("DeletePhoneNumber/{id}")]
public IActionResult DeletePhoneNumber(int id)
{
     PhoneNumber phone =
_phonenumbers.PhoneNumbers.FirstOrDefault(e => e.phone_id == id);
     if (phone == null)
```

```
     {
          List<string> errors = new List<string> { "Номер телефона
     не найден" };
          return new JsonResult(errors)
          { StatusCode = 404 }; // NotFound
     }

     _phonenumbers.PhoneNumbers.Remove(phone);
     _phonenumbers.SaveChanges();
     return new JsonResult("Номер телефона успешно удален") {
StatusCode = 200 };
}
```

Результатом работы данного метода может служить либо успешное удаление требуемого элемента из базы данных, либо неуспешное.

В случае успешного удаления элемента в клиентскую часть приложения передается соответствующее сообщение в формате JSON. Кроме того, также передается анонимный тип, содержащий HTTP-код «200», который указывает на то, что выполнение операции прошло успешно.

В случае ошибок при удалении выбранного номера телефона контакта в качестве результата работы метода передается соответствующее сообщение в формате JSON. В отличие от успешного удаления, в данном случае в результате передается HTTP-код «404», который означает, что искомый номер телефона не найден.

Таким образом, в результате работы метода формируется и возвращается ответ, содержащий соответствующее сообщение об успешном или неуспешном удалении. Далее, в клиентской части разрабатываемого приложения данный ответ используется для отображения результата, выполняемого при удалении объекта.

4 Контроль качества программного обеспечения

4.1 Планирование процесса контроля качества программного обеспечения

4.1.1 Обоснование выбора модели тестирования

Тестирование – это проверка созданного программного продукта на соответствие заданным требованиям и на отсутствие дефектов [21].

Методы тестирования программный средств подразделяют на две группы: ручное и автоматическое. При ручном тестировании все тесты вводятся вручную и, в случае с REST API, с использованием специального инструмента для взаимодействия.

Существует большое количество различных видов тестирования программных средств. Основные типы тестирования: функциональное, интеграционное, модульное и тестирование производительности.

Функциональное тестирование – это тип тестирования, который заключается установке соответствия разработанного программного обеспечения (ПО) исходным функциональным требованиям компании клиента. Иными словами, в данном случае происходит проверка способности информационной системы в определенных условиях решать задачи, нужные пользователям. Основные преимущества функционального тестирования заключаются в имитации фактического использования системы, своевременном выявлении системных ошибок ПО и экономии за счёт избавления от ошибок на раннем этапе жизненного цикла ПО. Этот тип подразделяется на несколько этапов: подготовка, проведение и отчёт. В первом этапе происходит анализ исходных документов о системе. Кроме того, разрабатывается план тестирования, а затем он согласуется с заказчиком. Второй этап заключается в ручной проверке по продуманным сценариям и занесении всех найденных ошибок. В последней части происходит разработка и согласование отчетов о проведённом тестировании со списком обнаруженных отклонений. На этом этапе также выносятся рекомендации по улучшению системы [22].

Модульное тестирование представляет собой метод тестирования программных средств, при котором объектами тестирования выступают отдельные методы программного кода, для которых требуется осуществить проверку корректности работоспособности. Автоматизация тестирования модулей осуществляется посредством разработки отдельного программного модуля, однако это не требует больших временных и финансовых затрат [23].

Интеграционное тестирование, в большинстве случаев, предназначено для проверки корректности взаимодействия между клиентской частью приложения и серверной посредством API. Важной особенностью данного типа также является то, что проверке подвергается взаимодействие различных модулей и сервисов приложения.

Отличительной чертой тестирования производительности является то, что при его выполнении удается оценить надежность, скорость и отзывчивость разработанных программных средств, а также оценить стабильность и соответствие требованиям производительности.

Для выполнения ручного типа тестирования серверной части электронной контактной книги с возможностью ведения заметок было выбрано функциональное тестирование. Ручное функциональное тестирование имеет ряд преимуществ:

– ручное тестирование легко адаптируется под разные условия проекта;

– данный тип тестирования позволяет на ранних этапах выявить дефекты программных средств;

– большое разнообразие тестовых примеров по сравнению с автоматическими видами тестирования.

Для автоматического тестирования было выбрано модульное тестирование. В первую очередь, модульные тесты также помогают выявить ошибки в разрабатываемых программных средствах на раннем этапе. Кроме того, одним из значительных преимуществ модульного тестирования является то, что данный тип позволяет программисту выполнять рефакторинг кода.

Благодаря данной особенности можно отследить, какое изменение в исходном коде вызвало ошибку, и сразу же ликвидировать ее.

4.1.2 Обоснование выбора метрик программного кода

Метрики кода представляют собой набор оценок программного обеспечения, которые дают разработчикам более глубокое представление о программном коде [24]. Выделяют следующие наиболее распространенные метрики кода:

– метрики устойчивости и абстрактности компонентов приложения;

– комплексный показатель качества кода;

– метрика Холстеда – анализ строк и синтаксических параметров кода программы.

Для оценки контроля качества разработанного серверного приложения были выбраны метрики Холстеда. Они позволяют рассчитать показатели программы на основе количества операндов, операторов и строк программного кода. С помощью них можно определить сложность написанного программного кода.

4.2 Тестирование программных средств

4.2.1 Функциональное тестирование

Для серверного приложения электронной контактной книги с возможностью ведения заметок был проведен ряд функциональных тестов, результаты которых содержатся в Приложении Г. Тесты были разделены по группам в соответствии с тестируемой областью приложения.

План функционального тестирования опций для работы с личным кабинетом представлен в таблице 43. Данный модуль содержит 6 тестов, проверяющих все типы ошибок при регистрации и авторизации пользователя, а также при работе с личным кабинетом.

Таблица 43 – План функционального тестирования опций для работы с личным кабинетом

№	Назначение теста	Состав теста	Результаты
1	Тест предназначен для проверки варианта использования «Зарегистрироваться в системе» с некорректными пользовательскими данными	Электронная почта: newnewnewnewnewnewnewnewnewnewnewnewnewnewnewnewnewnewnewnewnewnewnewnewnewnewnewnewnewnewnewnewnewnewnewnewnewnewnewnewnew====@mail.ru Пароль: 123	Пользователь не прошел регистрацию. Выведены ошибки о превышении допустимой длины адреса электронной почты, содержании в электронной почте некорректных символов, минимальной длине пароля. Тест пройден успешно (рисунок Г.1)
2	Тест предназначен для проверки варианта использования «Зарегистрироваться в системе» с корректными пользовательскими данными	Электронная почта: newuser@mail.ru Пароль: user123	Пользователь прошел регистрацию. В результате выведены данные нового пользователя в JSON-формате. Тест пройден успешно (рисунок Г.2)
3	Тест предназначен для проверки варианта использования «Авторизоваться в системе» с некорректными пользовательскими данными	Электронная почта: newuser@mail.ru Пароль: wrong123	Пользователь не прошел авторизацию. Появилось сообщение о том, что пароль для аккаунта неверный. Тест пройден успешно (рисунок Г.3)
4	Тест предназначен для проверки варианта использования «Авторизоваться в системе» с корректными пользовательскими данными	Электронная почта: newuser@mail.ru Пароль: user123	Пользователь прошел авторизацию. В результате выведены данные авторизованного пользователя в JSON-формате. Тест пройден успешно (рисунок Г.4)
5	Тест предназначен для проверки варианта использования «Редактировать информацию в личном кабинете» с некорректными пользовательскими данными	Имя пользователя: ===new $$$user Электронная почта: newuser@mail.ru Пароль: user123 Номер телефона: ikyehdthsui Дата рождения: 01.01.1900	Пользователь не смог изменить данные в личном кабинете. В результате выведены сообщения о некорректности имени пользователя, номера телефоны и даты рождения. Тест пройден успешно (рисунок Г.5)
6	Тест предназначен для проверки варианта использования «Редактировать информацию в личном кабинете» с некорректными пользовательскими данными	Имя пользователя: Петров Петр Электронная почта: newuser@mail.ru Пароль: user123 Номер телефона: 89027368273 Дата рождения: 10.01.1990	Пользователь смог изменить данные в личном кабинете. В результате выведены обновленные данные пользователя в JSON-формате. Тест пройден успешно (рисунок Г.6)
7	Тест предназначен для проверки варианта использования «Удалить аккаунт» с корректным идентификатором пользователя	Идентификатор пользователя: 63	Пользователь смог удалить аккаунт. Тест пройден успешно (рисунок Г.7)
8	Тест предназначен для проверки варианта использования «Удалить аккаунт» с некорректным идентификатором пользователя	Идентификатор пользователя: -1	Пользователь не смог удалить аккаунт. Выведено сообщение о том, что пользователь не найден. Тест пройден успешно (рисунок Г.8)

План функционального тестирования опций для работы с пользовательскими заметками представлен в таблице 44. Проведенные тесты отображают ошибки, возможные при работе с заметками.

Таблица 44 – План функционального тестирования опций для работы с пользовательскими заметками

№	Назначение теста	Состав теста	Результаты
1	Тест предназначен для проверки варианта использования «Изменить данные заметки» с некорректными данными заметки	Название заметки: === Описание заметки: ===	Пользователь не смог изменить данные заметки. Выведено сообщение о том, что название и описание содержат некорректные символы. Тест пройден успешно (рисунок Г.9)
2	Тест предназначен для проверки варианта использования «Изменить данные заметки» с корректными данными заметки	Название заметки: Измененная заметка Описание заметки: Измененное описание	Пользователь смог изменить данные заметки. Выведены обновленные данные заметки в JSON-формате. Тест пройден успешно (рисунок Г.10)
3	Тест предназначен для проверки варианта использования «Удалить заметку» с некорректным уникальным идентификатором	Идентификатор заметки: 1000	Пользователь не смог удалить заметку. Выведено сообщение о том, что заметка не найдена. Тест пройден успешно (рисунок Г.11)
4	Тест предназначен для проверки варианта использования «Удалить заметку» с корректным уникальным идентификатором	Идентификатор заметки: 204	Пользователь смог удалить заметку. Выведено сообщение о том, что заметка успешно удалена. Тест пройден успешно (рисунок Г 12)

План функционального тестирования опций для работы с пользовательскими контактами представлен в таблице 45.

Таблица 45 – План функционального тестирования опций для работы с пользовательскими контактами

№	Назначение теста	Состав теста	Результаты
1	2	3	4
1	Тест предназначен для проверки варианта использования «Добавить данные контакта» с некорректными данными контакта	ФИО: === Дата рождения: 01.01.1900 Адрес: === Место работы: === Другое: некоторая информация	Пользователь не смог добавить контакт. Выведено сообщение о том, что данные контакта содержат некорректные символы. Тест пройден успешно (рисунок Г.13)
2	Тест предназначен для проверки варианта использования «Добавить данные контакта» с корректными данными контакта	ФИО: Иванов Иван Дата рождения: 05.10.1989 Адрес: Место работы: Другое: некоторая информация	Пользователь смог добавить контакт. Выведены данные нового контакта в JSON-формате. Тест пройден успешно (рисунок Г.14)
3	Тест предназначен для проверки варианта использования «Изменить данные контакта» с некорректным идентификатором контакта	Идентификатор контакта: 1000	Пользователь не смог изменить данные контакта. Выведено сообщение о том, что контакт не найден. Тест пройден успешно (рисунок Г.15)
4	Тест предназначен для проверки варианта использования «Удалить контакт» с некорректным идентификатором контакта	Идентификатор контакта: -1	Пользователь не смог удалить контакт. Выведено сообщение о том, что контакт не найден. Тест пройден успешно (рисунок Г.16)

Продолжение таблицы 45

1	2	3	4
5	Тест предназначен для проверки варианта использования «Удалить контакт» с корректным идентификатором контакта	Идентификатор контакта: 205	Пользователь смог удалить контакт. Выведено сообщение о том, что контакт удален успешно. Тест пройден успешно (рисунок Г.17)

План функционального тестирования опций для работы с пользовательскими тегами представлен в таблице 46.

Таблица 46 – План функционального тестирования опций для работы с пользовательскими тегами

№	Назначение теста	Состав теста	Результаты
1	Тест предназначен для проверки варианта использования «Добавить данные тега» с некорректными данными тега	Название тега: Новый тег Цвет: = Идентификаторы заметок: 206, 207, 208	Пользователь не смог добавить данные тега. Выведено сообщение о том, что цвет содержит некорректные символы. Тест пройден успешно (рисунок Г.18)
2	Тест предназначен для проверки варианта использования «Добавить данные тега» с корректными данными тега	Название тега: Новый тег Идентификаторы заметок: 206, 207, 208	Пользователь смог добавить данные тега. Выведены данные нового тега в JSON-формате. Тест пройден успешно (рисунок Г.19)
3	Тест предназначен для проверки варианта использования «Изменить данные тега» с некорректным идентификатором тега	Идентификатор тега: 1000	Пользователь не смог изменить данные тега. Выведено сообщение о том, что тег не найден. Тест пройден успешно (рисунок Г.20)
4	Тест предназначен для проверки варианта использования «Изменить данные тега» с корректными данными тега	Название тега: Измененный тег Идентификаторы заметок: 206, 207	Пользователь смог изменить данные тега. Выведены обновленные данные тега в JSON-формате. Тест пройден успешно (рисунок Г.21)
5	Тест предназначен для проверки варианта использования «Удалить тег» с некорректным идентификатором тега	Идентификатор тега: -1	Пользователь не смог удалить тег. Выведено сообщение о том, что тег не найден. Тест пройден успешно (рисунок Г.22)
6	Тест предназначен для проверки варианта использования «Удалить тег» с корректным идентификатором тега	Идентификатор тега: 81	Пользователь смог удалить тег. Выведено сообщение о том, что тег успешно удален. Тест пройден успешно (рисунок Г.23)

План функционального тестирования опций для различных отображений пользовательских данных представлен в таблице 47. В разработанном приложении такими отображениями служат списки, отсортированные по некоторому критерию, а также списки контактов и заметок, отфильтрованные по заданной дате. В случае контактов фильтром выступает месяц даты рождения, а в случае заметок – месяц и год создания. Сортировка списков пользовательских контактов и заметок происходит по ФИО для контактов и по названию для заметок.

Таблица 47 – План функционального тестирования опций для различных отображений пользовательских данных

№	Назначение теста	Состав теста	Результаты
1	Тест предназначен для проверки варианта использования «Выбрать критерий для фильтрации контактов» с корректным месяцем для фильтрации по дате	Критерий для фильтрации: 05	Пользователь смог просмотреть отфильтрованный список контактов. Выведен отфильтрованный список контактов в JSON-формате. Тест пройден успешно (рисунок Г.24)
2	Тест предназначен для проверки варианта использования «Выбрать критерий для фильтрации заметок» с корректным месяцем и годом для фильтрации по дате	Месяц: 06 Год: 2024	Пользователь смог просмотреть отфильтрованный список заметок. Выведен отфильтрованный список заметок в JSON-формате. Тест пройден успешно (рисунок Г.25)
3	Тест предназначен для проверки варианта использования «Выбрать критерий для сортировки контактов», сортировка осуществляется по ФИО контакта		Пользователь смог просмотреть отсортированный по ФИО список контактов. Выведен отфильтрованный список контактов в JSON-формате. Тест пройден успешно (рисунок Г.26)

В результате выполнения функционального тестирования все тесты были пройдены успешно.

4.2.2 Модульное тестирование

Модульное тестирование серверной части приложения электронной контактной книги с возможностью ведения заметок было выполнено при помощи платформы xUnit [25]. Для тестирования были выбраны методы GET и POST контроллера NoteController, отвечающего за работу с пользовательскими заметками.

Для идентификации метода модульного тестирования используется параметр [Fact], следом за которым идет описание самого метода. Тестовый метод для вывода пустого списка заметок приведен в листинге 11. Полный листинг кода модульных тестов представлен в приложении Д.

Листинг 11 – Тестовый метод для вывода пустого списка заметок

```
[Fact]
public async Task GetNotes_WhenNoNotes_ReturnsEmptyList()
{
     // Arrange
     var client = _factory.CreateClient();
     client.DefaultRequestHeaders.Add("UserId", "61");
     CurrentUser.UserId = 61;

     // Act
```

```
        var response = await client.GetAsync("/Note/GetNotes");

        // Assert
        response.StatusCode.Should().Be(HttpStatusCode.OK);
        var options = new JsonSerializerOptions
        {
            Converters = { new DateOnlyConverter() }
        };
        var notes = await
response.Content.ReadFromJsonAsync<List<Note>>(options);
        notes.Should().BeEmpty();
}
```

В данном фрагменте кода для получения списка заметок происходит обращение по URL-адресу «/Note/GetNotes» к соответствующему методу GET в контроллере заметок. После этого происходит проверка результата выполненного действия. Так как у заданного по уникальному идентификатору пользователя нет ни одной заметки, то список должен быть пустым. При этом данный результат является корректным для метода в контроллере, поэтому ожидаемый статус HTTP-запроса – «200».

Полный состав методов тестирования представлен на рисунке 14.

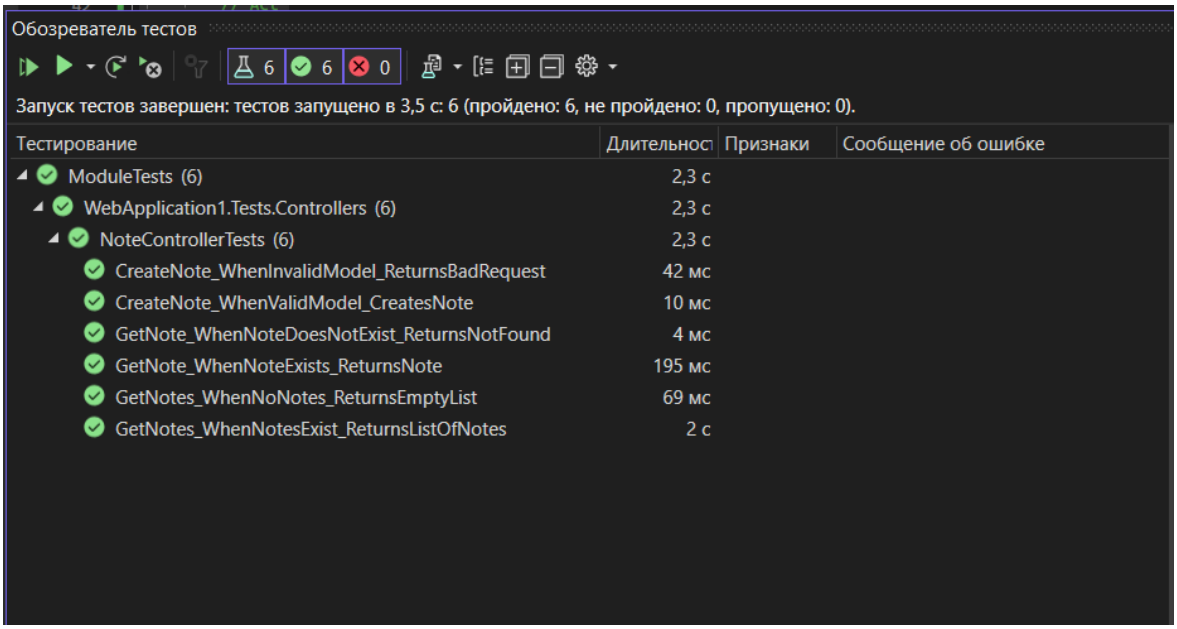


Рисунок 14 – Состав модульных тестов для контроллера заметок

Для метода создания новой заметки POST было выполнено два модульных теста: на случай успешного создания и на случай неуспешного. Для метода получения списка всех заметок пользователя GET также было выполнено два

теста: в первом случае, когда у пользователя нет заметок, а во втором – когда у пользователя есть заметки и выводится их список. Для метода получения конкретной пользовательской заметки GET были выполнены следующие тесты: в первом случае, когда заметка не найдена, а во втором – когда найдена и выведена.

Результаты тестирования показывают, что данные методы контроллера заметок работают исправно и без ошибок.

4.3 Оценка качества программного кода на основе метрик

4.3.1 Метрики Холстеда

Для оценки сложности программного кода были использованы метрики Холстеда. С их помощью можно рассчитать длину, словарь, а также сложность кода на основе количества общих и уникальных операторов и операндов [26].

Для оценки качества программного кода был взят метод CreatePhoneNumber, предназначенный для добавления нового номера телефона контакта. Код данного метода представлен в листинге 12.

Листинг 12 – Метод CreatePhoneNumber для оценки качества кода

```
public IActionResult CreatePhoneNumber(int contactId, [FromForm]
PhoneNumberCreateModel model)
{
     var elemsId = _elements.Elements.Where(e => e.user_id ==
CurrentUser.UserId && e.elem_type == true).Select(e =>
e.elem_id).ToList();
     var contacts = _contacts.Contacts.Where(n =>
elemsId.Contains(n.elem_id));
     var contact = contacts.FirstOrDefault(n => n.elem_id ==
contactId);
     if (contact == null)
     {
          List<string> errors = new List<string> { "Контакт не
найден" };
          return new JsonResult(errors)
          { StatusCode = 404 }; // NotFound
     }
     else
     {
          if (ModelState.IsValid)
          {
               var phone = new PhoneNumber();
```

```
                phone.elem_id = contactId;
                phone.pn = model.pn;
                phone.device_id = model.device_id;
                _phonenumbers.PhoneNumbers.Add(phone);
                _phonenumbers.SaveChanges();
                return new JsonResult(phone) { StatusCode = 200 };
           }
           else
           {
                List<string> errors = new List<string>();
                foreach (var item in ModelState)
                {
                     if (item.Value.ValidationState ==
ModelValidationState.Invalid)
                     {
                          foreach (var error in item.Value.Errors)
                          {
                               errors.Add(error.ErrorMessage);
                          }
                     }
                }
                return new JsonResult(errors)
                { StatusCode = 400 }; // BadRequest
           }
      }
}
```

Для оценки сложности программного кода метриками Холстеда требуется подсчитать количество общих и уникальных операторов и операндов. Операторы представлены в таблице 48.

Таблица 48 – Операторы метода CreatePhoneNumber

Оператор	Число вхождений
==	2
!=	1
&&	2
||	1
+	1

Операнды данного метода представлены в таблице 49. Общее количество операндов составило 31, а количество уникальных – 26. Общее количество операторов составило 7, а количество уникальных – 5.

Таблица 49 – Операнды метода CreatePhoneNumber

Операнд	Число вхождений
contactId	4
_elements.Elements	1
e.user_id	1
CurrentUser.UserId	1
e.elem_type	1
true	1
_contacts.Contacts	1
n.elem_id	2
contact	2
null	1
«Контакт не найден»	1
ModelState.IsValid	1
model.pn	1
model.device_id	1
_phonenumbers.PhoneNumbers	1
_phonenumbers.SaveChanges()	1
404	1
200	1
item	1
item.Value.ValidationState	1
ModelValidationState.Invalid	1
error	1
item.Value.Errors	1
error.ErrorMessage	1
errors	1
400	1

Длина программы рассчитывается по формуле:

$$N = N1 + N2 = 7 + 31 = 38,$$

где *N1* – общее число операторов;

N2 – общее число операндов.

Словарь программы рассчитывается по формуле:

$$n = n1 + n2 = 5 + 26 = 31,$$

где *n1* – число уникальных операторов;

n2 – число уникальных операндов.

Объем программы рассчитывается по формуле:

$$HV = N \cdot \log_2 n = 38 \cdot 4{,}95 = 188{,}1,$$

где *N* – длина программы;

n – словарь программы.

Сложность программы рассчитывается по формуле:

$$HD = \frac{n1}{2} \cdot \frac{N2}{n2} = \frac{5}{2} \cdot \frac{31}{26} = 2{,}98.$$

Сложность метода приблизительно равна 3, поэтому можно считать метод достаточно простым для понимания.

Заключение

Целью данной выпускной квалификационной работы была разработка серверного приложения электронной контактной книги с возможностью ведения заметок. Приложение позволяет пользователю выполнять следующие действия:

– регистрироваться и авторизоваться в системе;

– управлять своими контактами и заметками;

– группировать контакты и заметки в теги и осуществлять контроль над ними;

– создавать напоминания о важных событиях.

В дальнейшем приложение электронной контактной книги с возможностью ведения заметок будет развиваться и расширять свой функционал. Планируется добавить возможность общего доступа для совместной работы над контактами и заметками, а также приглашение новых пользователей и иное совместное взаимодействие в приложении.

Список использованных источников

1. Клиент-серверная архитектура в картинках // Хабр [Электронный ресурс]. URL: https://habr.com/ru/articles/495698/ (дата обращения: 25.12.2023)

2. Архитектура клиент-сервер // ITELON [Электронный ресурс]. URL: https://itelon.ru/blog/arkhitektura-klient-server/ (дата обращения: 25.12.2023)

3. Сравнительный анализ: Что это такое и как его проводить // HR-Portal [Электронный ресурс]. URL: https://hr-portal.ru/story/sravnitelnyy-analiz-chto-eto-takoe-i-kak-ego-provodit (дата обращения: 05.01.2024)

4. API от А до Я (теория и практика) // Хабр [Электронный ресурс]. URL: https://habr.com/ru/articles/768752/ (дата обращения: 10.01.2024)

5. Как устроено индексирование баз данных // Хабр [Электронный ресурс]. URL: https://habr.com/ru/companies/ruvds/articles/724066/ (дата обращения: 11.01.2024)

6. Зубрицкий, А. А. Эффективное управление проектами с использованием Microsoft Project 2016 / А. А. Зубрицкий. – М.: Наука, 2016. – 123 с.

7. Использование диаграммы вариантов использования UML при проектировании программного обеспечения // Хабр [Электронный ресурс]. URL: https://habr.com/ru/articles/566218/ (дата обращения: 23.01.2024)

8. Система управления базами данных: что это такое и зачем она нужна // Skillbox Media [Электронный ресурс]. URL: https://skillbox.ru/media/code/sistema-upravleniya-bazami-dannykh-chto-eto-takoe-i-zachem-ona-nuzhna/ (дата обращения: 23.01.2024)

9. Моделирование данных в СУБД: от концептуального проектирования к реализации // AppMaster [Электронный ресурс]. URL: https://appmaster.io/ru/blog/modelirovanie-dannykh-v-rdbms (дата обращения: 01.02.2024)

10. Лобзенко, П.В. Проектирование клиент-серверных приложений / П. В. Лобзенко, И. В. Щербань. – М.: СКФ МТУСИ, 2018. – 55 с.

11. Как пользоваться Postman // losst [Электронный ресурс]. URL: https://losst.pro/kak-polzovatsya-postman (дата обращения: 06.02.2024)

12. Платформа .NET и C# от А до Я // csharp [Электронный ресурс]. URL: https://csharp.webdelphi.ru/ (дата обращения: 09.02.2024)

13. Введение в ASP.NET CORE // ITVDN [Электронный ресурс]. URL: https://itvdn.com/ru/blog/article/aspnet_core#:~:text=ASP.NET%20Core%20–%20это%20новый,на%20полной%20платформе%20.NET%20Framework (дата обращения: 09.02.2024)

14. Простое руководство к диаграммам развертывания UML // creately [Электронный ресурс]. URL: https://creately.com/blog/ru/uncategorized-ru/учебное-пособие-по-диаграмме-развёрт/ (дата обращения: 15.02.2024)

15. Физические модели баз данных // НОУ ИНТУИТ [Электронный ресурс]. URL: https://intuit.ru/studies/courses/1001/297/lecture/7415 (дата обращения: 21.02.2024)

16. Стоунз, Р. Основы PostgreSQL / Р. Стоунз, Н. Меттью. – М.: Символ-Плюс, 2002. – 640 с.

17. Руководство по Entity Framework Core 8 // metanit.com [Электронный ресурс]. URL: https://metanit.com/sharp/efcore/ (дата обращения: 01.03.2024)

18. Синтаксис запросов LINQ – едооцененный инструмент для разработчиков C# // Хабр [Электронный ресурс]. URL: https://habr.com/ru/companies/otus/articles/723438/ (дата обращения: 01.03.2024)

19. Как создать простое Rest API на .NET Core // Хабр [Электронный ресурс]. URL: https://habr.com/ru/articles/531106/ (дата обращения: 06.03.2024)

20. ASP.NET Core MVC // metanit.com [Электронный ресурс]. URL: https://metanit.com/sharp/aspnet5/3.1.php (дата обращения: 08.03.2024)

21. Тестирование и 7 основных этапов его проведения // Neiros [Электронный ресурс]. URL: https://neiros.ru/blog/code/testirovanie-i-7-osnovnykh-etapov-ego-provedeniya/ (дата обращения: 20.03.2024)

22. Функциональное тестирование: что это, этапы, виды и инструменты использования // Tproger [Электронный ресурс]. URL:

https://tproger.ru/articles/funkcionalnoe-testirovanie-chto-eto-etapy-vidy-i-instrumenty-ispolzovaniya (дата обращения: 20.03.2024)

23. Модульное тестирование: что это? Типы, инструменты // logrocon [Электронный ресурс]. URL: https://logrocon.ru/news/unit_testing (дата обращения: 20.03.2024)

24. Программный код и его метрики // Хабр [Электронный ресурс]. URL: https://habr.com/ru/companies/intel/articles/106082/ (дата обращения: 27.03.2024)

25. Введение в написание тестов и знакомство с xUnit // Хабр [Электронный ресурс]. URL: https://habr.com/ru/articles/357648/ (дата обращения: 01.04.2024)

26. Метрика холстеда. // studfile [Электронный ресурс]. URL: https://studfile.net/preview/1568115/page:2/ (дата обращения: 10.04.2024)

Приложение А

(справочное)

Глоссарий

API (Application Programming Interface) – это набор правил и протоколов, который позволяет разным программам взаимодействовать друг с другом.

UML (Unified Modeling Language) – язык графического описания для объектного моделирования в области разработки программного обеспечения, для моделирования бизнес-процессов, системного проектирования и отображения организационных структур.

СУБД (Система управления базами данных) – это совокупность программных и лингвистических средств общего или специального назначения, обеспечивающих управление созданием и использованием баз данных.

ASP .NET Core – расширение платформы .NET инструментами и библиотеками, специально предназначенными для создания веб-приложений.

MVC – это схема разделения данных приложения и управляющей логики на три отдельных компонента: модель, представление и контроллер.

HTTP –это протокол, позволяющий получать различные ресурсы

HTTP-запросы – это сообщения, отправляемые клиентом, чтобы инициировать реакцию со стороны сервера.

SQL – это язык запросов для управления реляционными базами данных.

PostgreSQL – это объектно-реляционная система управления базами данных, наиболее развитая из открытых СУБД в мире

ORM (Object-Relational Mapping) – технология программирования, которая связывает базы данных с концепциями объектно-ориентированных языков программирования, создавая «виртуальную объектную базу данных».

LINQ (Language Integrated Query) – проект Microsoft по добавлению синтаксиса языка запросов, напоминающего SQL, в языки программирования платформы .NET.

REST API – это архитектурный подход, который устанавливает ограничения для API: как они должны быть устроены и какие функции должны поддерживать.

URL – это адрес страницы в интернете, который позволяет получить доступ к этой странице через веб-браузер

JSON (JavaScript Object Notation) – стандартный текстовый формат для хранения и передачи структурированных данных.

ПО (программное обеспечение) – набор программ, баз данных, файлов, а также описывающих их документов, составляющих систему, для решения группы связанных задач на одном или нескольких взаимодействующих компьютерах.

Приложение Б

(рекомендуемое)

Скрипты для создания объектов базы данных на PostgreSQL

```
CREATE TABLE fullUser (

     user_ID SERIAL PRIMARY KEY,

     user_name  VARCHAR(100) CHECK(user_name ~ '[а-яА-Я_ ]'),

     user_email  VARCHAR (35) CHECK(user_email ~ '^[A-Za-z0-9._%-
]+@[A-Za-z0-9.-]+[.][A-Za-z]+$') NOT NULL UNIQUE,

     user_password VARCHAR(20) NOT NULL,

     user_PN     VARCHAR(12)  CHECK(user_PN  ~  '^((\+7|7|8)+([0-
9]){10})$'),

     user_birth    DATE    CHECK    (user_birth>'1950-01-01'    AND
user_birth<'2023-01-01')

);

CREATE TABLE elem(

     elem_ID SERIAL PRIMARY KEY,

     elem_type BOOL NOT NULL,

     user_ID INTEGER REFERENCES fullUser (user_ID) NOT NULL

);

CREATE TABLE contact(

     elem_ID INTEGER REFERENCES elem (elem_ID) NOT NULL,

     FIO  VARCHAR(100) NOT NULL CHECK(FIO ~ '^[а-яА-ЯёЁa-zA-Z_ ]+
[а-яА-ЯёЁa-zA-Z_ ]+ ?[а-яА-ЯёЁa-zA-Z_ ]+$'),

     cont_birth  DATE  NOT  NULL  CHECK  (cont_birth>'1950-01-01'  AND
cont_birth<'2023-01-01'),

     address VARCHAR(100) CHECK(address ~ '^[а-яА-ЯёЁa-zA-Z0-9\.\-
\,\s]+$'),

     cont_work  VARCHAR(100) CHECK(cont_work ~ '^[а-яА-ЯёЁa-zA-Z0-
9_ ]+$'),

     other VARCHAR(200) CHECK(other ~ '^[а-яА-ЯёЁa-zA-Z0-9\.\-\,\s
]+$'),

     PRIMARY KEY (elem_ID)

);

CREATE TABLE note(

     elem_id INTEGER REFERENCES elem (elem_ID) NOT NULL,
```

```
    note_name varchar(50) CHECK(note_name ~ '^[a-яА-ЯёЁa-zA-Z0-9\.\-\,\s ]+$') NOT NULL,

    note_create DATE NOT NULL CHECK (note_create>'2023-01-01' AND note_create<'2024-01-01'),

    description varchar(150) CHECK(description ~ '^[a-яА-ЯёЁa-zA-Z0-9\.\-\,\s ]+$'),

    PRIMARY KEY (elem_ID)

);

CREATE TABLE tag(

    tag_ID SERIAL PRIMARY KEY,

    tag_name VARCHAR (30) CHECK(tag_name ~ '^[a-яА-ЯёЁa-zA-Z0-9\.\-\,\s ]+$') NOT NULL UNIQUE,

    color VARCHAR(20) CHECK(color ~ '^[a-яА-ЯёЁ_ ]')

);

CREATE TABLE phoneNumber(

    phone_ID SERIAL PRIMARY KEY,

    PN  VARCHAR(11) CHECK(PN ~ '^((\+7|7|8)+([0-9]){10})$') NOT NULL,

    elem_ID INTEGER REFERENCES elem (elem_ID) NOT NULL

);


--ТАБЛИЦА СОЦСЕТЕЙ БЕЗ ВНЕШНЕГО КЛЮЧА НА НАЗВАНИЕ СОЦСЕТИ

CREATE TABLE socialNetwork(

    sn_ID SERIAL PRIMARY KEY,

    sn_link VARCHAR(100) NOT NULL,

    elem_id INTEGER REFERENCES elem (elem_ID) NOT NULL

);

CREATE TABLE email(

    email_ID SERIAL PRIMARY KEY,

    email_address  VARCHAR(30) CHECK(email_address ~ '^[A-Za-z0-9._%-]+@[A-Za-z0-9.-]+[.][A-Za-z]+$') NOT NULL,
```

```
    elem_ID INTEGER REFERENCES elem (elem_ID) NOT NULL
);
CREATE TABLE generalAccess(
    GA_ID SERIAL PRIMARY KEY,
    ga_link VARCHAR(100) NOT NULL UNIQUE,
    user_id INTEGER REFERENCES fullUser (user_ID) NOT NULL,
    elem_id INTEGER REFERENCES elem (elem_ID) NOT NULL
);

CREATE TABLE notification (
    notif_ID SERIAL PRIMARY KEY,
    notif_name VARCHAR(30) CHECK(notif_name ~ '^[a-яА-ЯёЁa-zA-Z0-9\.\-\,\s ]+$') NOT NULL,
    notif_date DATE NOT NULL CHECK (notif_date>'2023-01-01' AND notif_date<'2024-01-01'),
    notif_time TIME(6) NOT NULL,
    elem_ID INTEGER REFERENCES elem (elem_ID) NOT NULL,
    notif_desc VARCHAR(100) CHECK(notif_desc ~ '^[a-яА-ЯёЁa-zA-Z0-9\.\-\,\s ]+$'),
    notif_type BOOL NOT NULL
);

CREATE TABLE element_tag (
    elem_ID INTEGER NOT NULL,
    tag_ID INTEGER NOT NULL
);

CREATE UNIQUE INDEX user_user_ID ON fullUser (user_ID);
CREATE UNIQUE INDEX user_user_email ON fullUser (user_Email);
CREATE UNIQUE INDEX contact_elem_ID ON contact (elem_ID);
```

```
CREATE INDEX contact_FIO ON contact (FIO);

CREATE UNIQUE INDEX note_elem_ID ON note (elem_ID);

CREATE UNIQUE INDEX tag_tag_ID ON tag (tag_ID);

CREATE UNIQUE INDEX tag_tag_name ON tag (tag_Name);

CREATE UNIQUE INDEX phoneNumber_phone_ID ON phoneNumber (phone_ID);

CREATE UNIQUE INDEX socialNetwork_SN_ID ON socialNetwork (SN_ID);

CREATE UNIQUE INDEX email_email_ID ON email (email_ID);

CREATE INDEX email_emailAddress ON email (email_address);

CREATE UNIQUE INDEX generalAccess_GA_ID ON generalAccess (GA_ID);

CREATE UNIQUE INDEX notification_notif_ID ON notification
(notif_ID);

CREATE INDEX notification_notif_name ON notification (notif_Name);

CREATE INDEX notification_notif_date ON notification (notif_Date);

CREATE INDEX element_tag_elem_ID ON element_tag (elem_ID);

CREATE INDEX element_tag_tag_ID ON element_tag (tag_ID);

CREATE UNIQUE INDEX element_elem_ID ON elem (elem_ID);
```

Приложение В

(рекомендуемое)

Листинг серверного приложения

Файл ContactController.cs

```
using Microsoft.AspNetCore.Http;
using Microsoft.AspNetCore.Mvc;
using Microsoft.AspNetCore.Mvc.ModelBinding;
using System.ComponentModel.DataAnnotations;
using System.Text.Json;
using WebApplication1.Entities;
using WebApplication1.JsonConverter;

namespace WebApplication1.Controllers
{
     [Route("[controller]")]
     [ApiController]
     public class ContactController : Controller
     {
          private readonly ContactRepository _contacts;
          private readonly ElementRepository _elements;
          private readonly PhoneNumberRepository _phoneNumbers;
          private readonly SocialNetworkRepository _socialNetworks;
          private readonly EmailRepository _emails;

          public ContactController(ContactRepository contacts,
ElementRepository elements, PhoneNumberRepository phoneNumbers,
SocialNetworkRepository socialNetworks, EmailRepository emails)
          {
               _contacts = contacts;
               _elements = elements;
               _phoneNumbers = phoneNumbers;
               _socialNetworks = socialNetworks;
               _emails = emails;
          }

          //только для получения всех контактов, чтобы использовать
данный код в других методах!!!
          [HttpGet("GetFullContact")]
          public IActionResult GetFullContact()
          {
               var elemsId = _elements.Elements.Where(e =>
e.user_id == CurrentUser.UserId && e.elem_type == true).Select(e
=> e.elem_id).ToList();
               var contacts = _contacts.Contacts.Where(n =>
elemsId.Contains(n.elem_id));
               var phones = new PhoneNumberController(_contacts,
_elements, _phoneNumbers);
               var emails = new EmailController(_contacts,
_elements, _emails);
               var networks = new
SocialNetworkController(_contacts, _elements, _socialNetworks);
               List<ContactAndInformation> contactAndInformation =
new List<ContactAndInformation>();
               foreach (var contact in contacts)
               {
```

```
                    var currentPhones =
phones.GetListPhonesForContact(contact.elem_id).ToList();
                    var currentEmails =
emails.GetListEmailsForContact(contact.elem_id).ToList();
                    var currentNetworks =
networks.GetListSocialNetworksForContact(contact.elem_id).ToList()
;
                    var fullContact = new ContactAndInformation();
                    fullContact.contact = contact;
                    fullContact.phoneNumbers = currentPhones;
                    fullContact.emails = currentEmails;
                    fullContact.socialNetworks = currentNetworks;
                    contactAndInformation.Add(fullContact);
               }
               return Ok(contactAndInformation);
          }

          // Получение всех контактов пользователя
          [HttpGet("GetAllContacts")]
          public IActionResult GetContacts()
          {
               var options = new JsonSerializerOptions
               {
                    Converters = { new DateOnlyConverter() }
               };
               return new JsonResult(GetFullContact(), options) {
StatusCode = 200 };
          }

          // Получение контакта по ID (с проверкой принадлежности
пользователю)
          [HttpGet("GetContact/{id}")]
          public IActionResult GetContact(int id)
          {
               var elemsId = _elements.Elements.Where(e =>
e.user_id == CurrentUser.UserId && e.elem_type == true).Select(e
=> e.elem_id).ToList();
               var contacts = _contacts.Contacts.Where(n =>
elemsId.Contains(n.elem_id));
               var contact = contacts.FirstOrDefault(n => n.elem_id
== id);
               var phones = new PhoneNumberController(_contacts,
_elements, _phoneNumbers);
               var emails = new EmailController(_contacts,
_elements, _emails);
               var networks = new
SocialNetworkController(_contacts, _elements, _socialNetworks);
               if (contact == null)
               {
                    List<string> errors = new List<string> {
"Контакт не найден" };
                    return new JsonResult(errors)
                    { StatusCode = 404 }; // NotFound
```

```
                }
                var fullContact = new ContactAndInformation();
                fullContact.contact = contact;
                fullContact.phoneNumbers =
phones.GetListPhonesForContact(contact.elem_id).ToList();
                fullContact.emails =
emails.GetListEmailsForContact(contact.elem_id).ToList();
                fullContact.socialNetworks =
networks.GetListSocialNetworksForContact(contact.elem_id).ToList()
;

                var options = new JsonSerializerOptions
                {
                     Converters = { new DateOnlyConverter() }
                };

                return new JsonResult(fullContact, options) {
StatusCode = 200 };
          }

          // Создание нового контакта
          [HttpPost("CreateContact")]
          public IActionResult CreateContact([FromForm]
ContactModel contactModel)
          {
                if (ModelState.IsValid)
                {
                     Element newElem = new Element();
                     newElem.user_id = CurrentUser.UserId;
                     newElem.elem_type = true;
                     _elements.Elements.Add(newElem);
                     _elements.SaveChanges();
                     int newElemId = newElem.elem_id;

                     var contact = new Contact();
                     contact.elem_id = newElemId;
                     contact.fio = contactModel.fio;
                     DateOnly newBirth;
                     if (DateOnly.TryParse(contactModel.cont_birth,
out newBirth))
                     {
                          contact.cont_birth = newBirth;
                     }
                     else
                     {
                          List<string> errors = new List<string> {
"Неверный формат ввода даты рождения" };
                          return new JsonResult(errors)
                          { StatusCode = 400 }; // BadRequest
                     }
                     contact.address = contactModel.address;
                     contact.cont_work = contactModel.cont_work;
                     contact.other = contactModel.other;
```

```
                    _contacts.Contacts.Add(contact);

                    var phones = new
PhoneNumberController(_contacts, _elements, _phoneNumbers);
                    var emails = new EmailController(_contacts,
_elements, _emails);
                    var networks = new
SocialNetworkController(_contacts, _elements, _socialNetworks);
                    _contacts.SaveChanges();

                    var options = new JsonSerializerOptions
                    {
                         Converters = { new DateOnlyConverter() }
                    };

                    var fullContact = new ContactAndInformation();
                    fullContact.contact = contact;
                    fullContact.phoneNumbers =
phones.GetListPhonesForContact(contact.elem_id).ToList();
                    fullContact.emails =
emails.GetListEmailsForContact(contact.elem_id).ToList();
                    fullContact.socialNetworks =
networks.GetListSocialNetworksForContact(contact.elem_id).ToList()
;

                    return new JsonResult(fullContact, options) {
StatusCode = 200 };
               }
               else
               {
                    List<string> errors = new List<string>();
                    foreach (var item in ModelState)
                    {
                         if (item.Value.ValidationState ==
ModelValidationState.Invalid)
                         {
                              foreach (var error in
item.Value.Errors)
                              {
                                   errors.Add(error.ErrorMessage);
                              }
                         }
                    }
                    return new JsonResult(errors)
                    { StatusCode = 400 }; // BadRequest
               }
          }

          // Изменение контакта (с проверкой принадлежности
пользователю)
          [HttpPut("UpdateContact/{id}")]
          public IActionResult UpdateContact(int id, [FromForm]
ContactModel contactModel)
```

```
          {
               if (ModelState.IsValid)
               {
                    var elemsId = _elements.Elements.Where(e =>
e.user_id == CurrentUser.UserId && e.elem_type == true).Select(e
=> e.elem_id).ToList();
                    var contacts = _contacts.Contacts.Where(n =>
elemsId.Contains(n.elem_id));
                    var contact = contacts.FirstOrDefault(n =>
n.elem_id == id);
                    var phones = new
PhoneNumberController(_contacts, _elements, _phoneNumbers);
                    var emails = new EmailController(_contacts,
_elements, _emails);
                    var networks = new
SocialNetworkController(_contacts, _elements, _socialNetworks);
                    if (contact == null)
                    {
                         List<string> errors = new List<string> {
"Контакт не найден" };
                         return new JsonResult(errors)
                         { StatusCode = 404 }; // NotFound
                    }

                    contact.fio = contactModel.fio;
                    DateOnly newBirth;
                    if (DateOnly.TryParse(contactModel.cont_birth,
out newBirth))
                    {
                         contact.cont_birth = newBirth;
                    }
                    else
                    {
                         List<string> errors = new List<string> {
"Неверный формат ввода даты рождения" };
                         return new JsonResult(errors)
                         { StatusCode = 400 }; // BadRequest
                    }
                    contact.address = contactModel.address;
                    contact.cont_work = contactModel.cont_work;
                    contact.other = contactModel.other;
                    _contacts.SaveChanges();

                    var fullContact = new ContactAndInformation();
                    fullContact.contact = contact;
                    fullContact.phoneNumbers =
phones.GetListPhonesForContact(contact.elem_id).ToList();
                    fullContact.emails =
emails.GetListEmailsForContact(contact.elem_id).ToList();
                    fullContact.socialNetworks =
networks.GetListSocialNetworksForContact(contact.elem_id).ToList()
;
```

```
                    var options = new JsonSerializerOptions
                    {
                         Converters = { new DateOnlyConverter() }
                    };
                    return new JsonResult(fullContact, options) {
StatusCode = 200 };
               }
               else
               {
                    List<string> errors = new List<string>();
                    foreach (var item in ModelState)
                    {
                         if (item.Value.ValidationState ==
ModelValidationState.Invalid)
                         {
                              foreach (var error in
item.Value.Errors)
                              {
                                   errors.Add(error.ErrorMessage);
                              }
                         }
                    }
                    return new JsonResult(errors)
                    { StatusCode = 400 }; // BadRequest
               }
          }

          // Удаление контакта (с проверкой принадлежности
пользователю)
          [HttpDelete("DeleteContact/{id}")]
          public IActionResult DeleteContact(int id)
          {
               var elemsId = _elements.Elements.Where(e =>
e.user_id == CurrentUser.UserId && e.elem_type == true).Select(e
=> e.elem_id).ToList();
               var contacts = _contacts.Contacts.Where(n =>
elemsId.Contains(n.elem_id));
               var contact = contacts.FirstOrDefault(n => n.elem_id
== id);
               if (contact == null)
               {
                    List<string> errors = new List<string> {
"Контакт не найден" };
                    return new JsonResult(errors)
                    { StatusCode = 404 }; // NotFound
               }
               _contacts.Contacts.Remove(contact);
               _contacts.SaveChanges();
               return new JsonResult("Контакт успешно удален") {
StatusCode = 200 };
          }

          //поиск контактов по фио
```

```
          [HttpPost("FindContactByFIO")]
          public IActionResult FindContactsByFIO([Required] string
request)
          {
               var elemsId = _elements.Elements.Where(e =>
e.user_id == CurrentUser.UserId && e.elem_type == true).Select(e
=> e.elem_id).ToList();
               var contacts = _contacts.Contacts.Where(n =>
elemsId.Contains(n.elem_id)).Where(c =>
c.fio.Contains(request)).ToList();
               var phones = new PhoneNumberController(_contacts,
_elements, _phoneNumbers);
               var emails = new EmailController(_contacts,
_elements, _emails);
               var networks = new
SocialNetworkController(_contacts, _elements, _socialNetworks);
               List<ContactAndInformation> contactAndInformation =
new List<ContactAndInformation>();
               foreach (var contact in contacts)
               {
                    var fullContact = new ContactAndInformation();
                    fullContact.contact = contact;
                    fullContact.phoneNumbers =
phones.GetListPhonesForContact(contact.elem_id).ToList();
                    fullContact.emails =
emails.GetListEmailsForContact(contact.elem_id).ToList();
                    fullContact.socialNetworks =
networks.GetListSocialNetworksForContact(contact.elem_id).ToList()
;
                    contactAndInformation.Add(fullContact);
               }

               if (contactAndInformation.Count > 0)
               {
                    var options = new JsonSerializerOptions
                    {
                         Converters = { new DateOnlyConverter() }
                    };
                    return new JsonResult(contactAndInformation,
options) { StatusCode = 200 };
               }
               else
               {
                    return new JsonResult("Контакты не найдены") {
StatusCode = 200 };
               }
          }

          //сортировка контактов по ФИО
          [HttpGet("SortContactsByFIO")]
          public IActionResult SortContactsByFIO()
          {
```

```
                var elemsId = _elements.Elements.Where(e =>
e.user_id == CurrentUser.UserId && e.elem_type == true).Select(e
=> e.elem_id).ToList();
                var contacts = _contacts.Contacts.Where(n =>
elemsId.Contains(n.elem_id)).OrderBy(c => c.fio);
                var phones = new PhoneNumberController(_contacts,
_elements, _phoneNumbers);
                var emails = new EmailController(_contacts,
_elements, _emails);
                var networks = new
SocialNetworkController(_contacts, _elements, _socialNetworks);
                List<ContactAndInformation> contactAndInformation =
new List<ContactAndInformation>();
                foreach (var contact in contacts)
                {
                     var fullContact = new ContactAndInformation();
                     fullContact.contact = contact;
                     fullContact.phoneNumbers =
phones.GetListPhonesForContact(contact.elem_id).ToList();
                     fullContact.emails =
emails.GetListEmailsForContact(contact.elem_id).ToList();
                     fullContact.socialNetworks =
networks.GetListSocialNetworksForContact(contact.elem_id).ToList()
;
                     contactAndInformation.Add(fullContact);
                }
                if (contactAndInformation.Count > 0)
                {
                     var options = new JsonSerializerOptions
                     {
                          Converters = { new DateOnlyConverter() }
                     };
                     return new JsonResult(contactAndInformation,
options) { StatusCode = 200 };
                }
                else
                {
                     return new JsonResult("Контакты не найдены") {
StatusCode = 200 };
                }
           }

           //сортировка контактов по дате рождения
           [HttpGet("SortContactsByDate")]
           public IActionResult SortContactsByDate()
           {
                var elemsId = _elements.Elements.Where(e =>
e.user_id == CurrentUser.UserId && e.elem_type == true).Select(e
=> e.elem_id).ToList();
                var contacts = _contacts.Contacts.Where(n =>
elemsId.Contains(n.elem_id)).OrderBy(c => c.cont_birth);
                var phones = new PhoneNumberController(_contacts,
_elements, _phoneNumbers);
```

```
                var emails = new EmailController(_contacts,
_elements, _emails);
                var networks = new
SocialNetworkController(_contacts, _elements, _socialNetworks);
                List<ContactAndInformation> contactAndInformation =
new List<ContactAndInformation>();
                foreach (var contact in contacts)
                {
                     var fullContact = new ContactAndInformation();
                     fullContact.contact = contact;
                     fullContact.phoneNumbers =
phones.GetListPhonesForContact(contact.elem_id).ToList();
                     fullContact.emails =
emails.GetListEmailsForContact(contact.elem_id).ToList();
                     fullContact.socialNetworks =
networks.GetListSocialNetworksForContact(contact.elem_id).ToList()
;
                     contactAndInformation.Add(fullContact);
                }
                if (contactAndInformation.Count > 0)
                {
                     var options = new JsonSerializerOptions
                     {
                          Converters = { new DateOnlyConverter() }
                     };
                     return new JsonResult(contactAndInformation,
options) { StatusCode = 200 };
                }
                else
                {
                     return new JsonResult("Контакты не найдены") {
StatusCode = 200 };
                }
           }

           //фильтрация контактов по месяцу рождения
           //месяц считывается числом
           [HttpPost("FilterByBirth")]
           public IActionResult FilterByBirth([FromForm][Required]
int request)
           {
                var elemsId = _elements.Elements.Where(e =>
e.user_id == CurrentUser.UserId && e.elem_type == true).Select(e
=> e.elem_id).ToList();
                var contacts = _contacts.Contacts.Where(n =>
elemsId.Contains(n.elem_id)).Where(c =>
c.cont_birth.Month.Equals(request));
                var phones = new PhoneNumberController(_contacts,
_elements, _phoneNumbers);
                var emails = new EmailController(_contacts,
_elements, _emails);
                var networks = new
SocialNetworkController(_contacts, _elements, _socialNetworks);
```

```
                List<ContactAndInformation> contactAndInformation =
new List<ContactAndInformation>();
                foreach (var contact in contacts)
                {
                     var fullContact = new ContactAndInformation();
                     fullContact.contact = contact;
                     fullContact.phoneNumbers =
phones.GetListPhonesForContact(contact.elem_id).ToList();
                     fullContact.emails =
emails.GetListEmailsForContact(contact.elem_id).ToList();
                     fullContact.socialNetworks =
networks.GetListSocialNetworksForContact(contact.elem_id).ToList()
;
                     contactAndInformation.Add(fullContact);
                }
                if (contactAndInformation.Count > 0)
                {
                     var options = new JsonSerializerOptions
                     {
                          Converters = { new DateOnlyConverter() }
                     };
                     return new JsonResult(contactAndInformation,
options) { StatusCode = 200 };
                }
                else
                {
                     return new JsonResult("Контакты не найдены") {
StatusCode = 200 };
                }
           }
     }
}
```

Файл DeviceController.cs

```
using Microsoft.AspNetCore.Http;
using Microsoft.AspNetCore.Mvc;
using WebApplication1.Entities;

namespace WebApplication1.Controllers
{

     [Route("[controller]")]
     [ApiController]
     public class DeviceController : Controller
     {
          private readonly DeviceRepository _devices;

          public DeviceController(DeviceRepository devices)
          {
               _devices = devices;
          }

          // Получение всех типов телефонов
```

```
          [HttpGet("GetAllDevices")]
          public IActionResult GetDevices()
          {
               var devices = _devices.Devices.ToList();
               return new JsonResult(devices) { StatusCode = 200 };
          }

          //получение конкретного типа телефона
          [HttpGet("GetOneDevice/{id}")]
          public IActionResult GetOneDevice(int id)
          {
               Device device = _devices.Devices.FirstOrDefault(e =>
e.device_id == id);
               if (device == null)
               {
                    List<string> errors = new List<string> { "Тип
номера телефона не найден" };
                    return new JsonResult(errors)
                    { StatusCode = 404 }; // NotFound
               }
               return new JsonResult(device) { StatusCode = 200 };
          }
     }
}
```

Файл EmailController.cs

```
using Microsoft.AspNetCore.Http;
using Microsoft.AspNetCore.Mvc;
using Microsoft.AspNetCore.Mvc.ModelBinding;
using System.ComponentModel.DataAnnotations;
using System.Xml.Linq;
using WebApplication1.Entities;

namespace WebApplication1.Controllers
{
     [Route("[controller]")]
     [ApiController]
     public class EmailController : Controller
     {
          private readonly EmailRepository _emails;
          private readonly ContactRepository _contacts;
          private readonly ElementRepository _elements;
          public EmailController(ContactRepository contacts,
ElementRepository elements, EmailRepository emails)
          {
               _contacts = contacts;
               _elements = elements;
               _emails = emails ?? throw new
ArgumentNullException(nameof(emails));
          }

          //только для использования в контроллере контакта
          [HttpGet("GetListEmailsForContact")]
```

```
          public List<Email> GetListEmailsForContact([Required] int
contactId)
          {
               var emails = _emails.Emails.Where(e => e.elem_id ==
contactId).ToList();
               return emails;
          }

          //получить все электронные почты конкретного контакта
          [HttpGet("GetListEmails")]
          public IActionResult GetListEmails([Required] int
contactId)
          {
               var emails = _emails.Emails.Where(e => e.elem_id ==
contactId).ToList();
               return new JsonResult(emails) { StatusCode = 200 };
          }

          //получить конкретную электронную почту
          [HttpGet("GetOneEmail/{id}")]
          public IActionResult GetOneEmail([Required] int id)
          {
               Email email = _emails.Emails.FirstOrDefault(e =>
e.email_id == id);
               if (email == null)
               {
                    List<string> errors = new List<string> { "Адрес
электронной почты не найден" };
                    return new JsonResult(errors)
                    { StatusCode = 404 }; // NotFound
               }
               return new JsonResult(email) { StatusCode = 200 };
          }

          // Создание новой почты
          [HttpPost("CreateEmail")]
          public IActionResult CreateEmail(int contactId,
[FromForm] EmailModel model)
          {
               var elemsId = _elements.Elements.Where(e =>
e.user_id == CurrentUser.UserId && e.elem_type == true).Select(e
=> e.elem_id).ToList();
               var contacts = _contacts.Contacts.Where(n =>
elemsId.Contains(n.elem_id));
               var contact = contacts.FirstOrDefault(n => n.elem_id
== contactId);
               if (contact == null)
               {
                    List<string> errors = new List<string> {
"Контакт не найден" };
                    return new JsonResult(errors)
                    { StatusCode = 404 }; // NotFound
               }
```

```
                else
                {
                    if (ModelState.IsValid)
                    {
                        var email = new Email();
                        email.elem_id = contactId;
                        email.email_address = model.email_address;
                        _emails.Emails.Add(email);
                        _emails.SaveChanges();
                        return new JsonResult(email) { StatusCode
= 200 };
                    }
                    else
                    {
                        List<string> errors = new List<string>();
                        foreach (var item in ModelState)
                        {
                            if (item.Value.ValidationState ==
ModelValidationState.Invalid)
                            {
                                foreach (var error in
item.Value.Errors)
                                {

     errors.Add(error.ErrorMessage);
                                }
                            }
                        }
                        return new JsonResult(errors)
                        { StatusCode = 400 }; // BadRequest
                    }
                }
            }

            // Изменение почты
            [HttpPut("UpdateEmail/{id}")]
            public IActionResult UpdateEmail(int id, [FromForm]
EmailModel model)
            {
                if (ModelState.IsValid)
                {
                    Email email = _emails.Emails.FirstOrDefault(e
=> e.email_id == id);
                    if (email == null)
                    {
                        List<string> errors = new List<string> {
"Адрес электронной почты не найден" };
                        return new JsonResult(errors)
                        { StatusCode = 404 }; // NotFound
                    }
                    email.email_address = model.email_address;
                    _emails.SaveChanges();
```

```
                    return new JsonResult(email) { StatusCode = 200
};
               }
               else
               {
                    List<string> errors = new List<string>();
                    foreach (var item in ModelState)
                    {
                         if (item.Value.ValidationState ==
ModelValidationState.Invalid)
                         {
                              foreach (var error in
item.Value.Errors)
                              {
                                   errors.Add(error.ErrorMessage);
                              }
                         }
                    }
                    return new JsonResult(errors)
                    { StatusCode = 400 }; // BadRequest
               }
          }

          // Удаление электронной почты
          [HttpDelete("DeleteEmail/{id}")]
          public IActionResult DeleteEmail(int id)
          {
               Email email = _emails.Emails.FirstOrDefault(e =>
e.email_id == id);
               if (email == null)
               {
                    List<string> errors = new List<string> { "Адрес
электронной почты не найден" };
                    return new JsonResult(errors)
                    { StatusCode = 404 }; // NotFound
               }
               _emails.Emails.Remove(email);
               _emails.SaveChanges();
               return new JsonResult("Электронная почта успешно
удалена") { StatusCode = 200 };
          }
     }
}
```

Файл EventController.cs

```
using Microsoft.AspNetCore.Http;
using Microsoft.AspNetCore.Mvc;
using WebApplication1.Entities;

namespace WebApplication1.Controllers
{
     [Route("[controller]")]
     [ApiController]
```

```
     public class EventController : Controller
     {
          private readonly EventRepository _events;

          public EventController(EventRepository events)
          {
               _events = events;
          }

          // Получение всех событий
          [HttpGet("GetEvents")]
          public IActionResult GetEvents()
          {
               var events = _events.Events.ToList();
               return Ok(events);
          }

          // Получение конкретного события
          [HttpGet("GetOneEvent/{id}")]
          public IActionResult GetOneEvent(int id)
          {
               var ev = _events.Events.FirstOrDefault(e =>
e.event_id == id);
               if (ev == null)
               {
                    List<string> errors = new List<string> { "Тип
события для уведомления не найден" };
                    return new JsonResult(errors)
                    { StatusCode = 404 }; // NotFound
               }
               return new JsonResult(ev) { StatusCode = 200 };
          }
     }
}
```

Файл FrequencyController.cs

```
using Microsoft.AspNetCore.Http;
using Microsoft.AspNetCore.Mvc;
using WebApplication1.Entities;
using WebApplication1;

namespace ContactBook.Controllers
{
     [Route("[controller]")]
     [ApiController]
     public class FrequencyController : Controller
     {
          private readonly FrequencyRepository _frequency;

          public FrequencyController(FrequencyRepository frequency)
          {
               _frequency = frequency;
          }
```

```
          // Получение всех частот напоминаний
          [HttpGet("GetAllFrequencies")]
          public IActionResult GetAllFrequencies()
          {
               var freq = _frequency.Frequencies.ToList();
               return new JsonResult(freq) { StatusCode = 200 };
          }

          //получение конкретной частоты напоминания
          [HttpGet("GetOneFrequency/{id}")]
          public IActionResult GetOneFrequency(int id)
          {
               var freq = _frequency.Frequencies.FirstOrDefault(e
=> e.freq_id == id);
               if (freq == null)
               {
                    List<string> errors = new List<string> {
"Переодичность для уведомлений некорректна" };
                    return new JsonResult(errors)
                    { StatusCode = 404 }; // NotFound
               }
               return new JsonResult(freq) { StatusCode = 200 };
          }
     }
}
```

Файл NoteController.cs

```
using Microsoft.AspNetCore.Http;
using Microsoft.AspNetCore.Mvc;
using Microsoft.AspNetCore.Mvc.ModelBinding;
using Microsoft.EntityFrameworkCore;
using Microsoft.EntityFrameworkCore.Query.SqlExpressions;
using System.ComponentModel.DataAnnotations;
using System.Data.Entity.Validation;
using System.Text.Json;
using System.Xml.Linq;
using WebApplication1.Entities;
using WebApplication1.JsonConverter;

namespace WebApplication1.Controllers
{
     [Route("[controller]")]
     [ApiController]
     public class NoteController : Controller
     {
          private readonly NoteRepository _notes;
          private readonly ElementRepository _elements;

          public NoteController(NoteRepository notes,
ElementRepository elements)
          {
               _notes = notes;
```

```
            _elements = elements;
        }

        // Получение всех заметок пользователя
        [HttpGet("GetNotes")]
        public IActionResult GetNotes()
        {
            var elemsId = _elements.Elements.Where(e =>
e.user_id == CurrentUser.UserId && e.elem_type == false).Select(e
=> e.elem_id).ToList();
            var notes = _notes.Notes.Where(n =>
elemsId.Contains(n.elem_id));
            var options = new JsonSerializerOptions
            {
                Converters = { new DateOnlyConverter() }
            };
            return new JsonResult(notes,options) { StatusCode =
200 };
        }

        // Получение заметки по ID (с проверкой принадлежности
пользователю)
        [HttpGet("GetNote/{id}")]
        public IActionResult GetNote(int id)
        {
            var elemsId = _elements.Elements.Where(e =>
e.user_id == CurrentUser.UserId && e.elem_type == false).Select(e
=> e.elem_id).ToList();
            var notes = _notes.Notes.Where(n =>
elemsId.Contains(n.elem_id));
            var note = notes.FirstOrDefault(n => n.elem_id ==
id);
            if (note == null)
            {
                List<string> errors = new List<string> {
"Заметка не найдена" };
                return new JsonResult(errors)
                { StatusCode = 404 }; // NotFound
            }

            var options = new JsonSerializerOptions
            {
                Converters = { new DateOnlyConverter() }
            };
            return new JsonResult(note, options) { StatusCode =
200 };
        }

        // Создание новой заметки
        [HttpPost("CreateNote")]
        public IActionResult CreateNote([FromForm]
NoteCreateModel newNote)
        {
```

```
                if (ModelState.IsValid)
                {
                     Element newElem = new Element();
                     newElem.user_id = CurrentUser.UserId;
                     newElem.elem_type = false;
                     _elements.Elements.Add(newElem);
                     _elements.SaveChanges();
                     int newElemId = newElem.elem_id;

                     var note = new Note();
                     note.elem_id = newElemId;
                     note.note_name = newNote.note_name;
                     note.note_create =
DateOnly.FromDateTime(DateTime.Now);
                     note.description = newNote.description;
                     _notes.Notes.Add(note);
                     _notes.SaveChanges();

                     var options = new JsonSerializerOptions
                     {
                          Converters = { new DateOnlyConverter() }
                     };
                     return new JsonResult(note,options) {
StatusCode = 200 };
                }
                else
                {
                     List<string> errors = new List<string>();
                     foreach (var item in ModelState)
                     {
                          if (item.Value.ValidationState ==
ModelValidationState.Invalid)
                          {
                               foreach (var error in
item.Value.Errors)
                               {
                                    errors.Add(error.ErrorMessage);
                               }
                          }
                     }
                     return new JsonResult(errors)
                     { StatusCode = 400 }; // BadRequest
                }
           }

           // Изменение заметки (с проверкой принадлежности
пользователю)
           [HttpPut("UpdateNote/{id}")]
           public IActionResult UpdateNote(int id, [FromForm]
NoteCreateModel newNote)
           {
                if (ModelState.IsValid)
                {
```

```
                    var elemsId = _elements.Elements.Where(e =>
e.user_id == CurrentUser.UserId && e.elem_type == false).Select(e
=> e.elem_id).ToList();
                    var notes = _notes.Notes.Where(n =>
elemsId.Contains(n.elem_id));
                    var note = notes.FirstOrDefault(n => n.elem_id
== id);
                    if (note == null)
                    {
                         List<string> errors = new List<string> {
"Заметка не найдена" };
                         return new JsonResult(errors)
                         { StatusCode = 404 }; // NotFound
                    }
                    note.note_name = newNote.note_name;
                    note.note_create =
DateOnly.FromDateTime(DateTime.Now);
                    note.description = newNote.description;
                    _notes.SaveChanges();

                    var options = new JsonSerializerOptions
                    {
                         Converters = { new DateOnlyConverter() }
                    };
                    return new JsonResult(note, options) {
StatusCode = 200 };
               }
               else
               {
                    List<string> errors = new List<string>();
                    foreach (var item in ModelState)
                    {
                         if (item.Value.ValidationState ==
ModelValidationState.Invalid)
                         {
                              foreach (var error in
item.Value.Errors)
                              {
                                   errors.Add(error.ErrorMessage);
                              }
                         }
                    }
                    return new JsonResult(errors)
                    { StatusCode = 400 }; // BadRequest
               }
          }

          // Удаление заметки (с проверкой принадлежности
пользователю)
          [HttpDelete("DeleteNote/{id}")]
          public IActionResult DeleteNote(int id)
          {
```

```
                var elemsId = _elements.Elements.Where(e =>
e.user_id == CurrentUser.UserId && e.elem_type == false).Select(e
=> e.elem_id).ToList();
                var notes = _notes.Notes.Where(n =>
elemsId.Contains(n.elem_id));
                var note = notes.FirstOrDefault(n => n.elem_id ==
id);
                if (note == null)
                {
                     List<string> errors = new List<string> {
"Заметка не найдена" };
                     return new JsonResult(errors)
                     { StatusCode = 404 }; // NotFound
                }
                _notes.Notes.Remove(note);
                _notes.SaveChanges();
                return new JsonResult("Заметка успешно удалена") {
StatusCode = 200 };
          }

          [HttpPost("FindNoteByName")]
          public IActionResult FindNotesByName([Required] string
request)
          {
                var elemsId = _elements.Elements.Where(e =>
e.user_id == CurrentUser.UserId && e.elem_type == false).Select(e
=> e.elem_id).ToList();
                var notes = _notes.Notes.Where(n =>
elemsId.Contains(n.elem_id)).Where(c =>
c.note_name.Contains(request)).ToList();
                if (notes.Count != 0)
                {
                     var options = new JsonSerializerOptions
                     {
                          Converters = { new DateOnlyConverter() }
                     };
                     return new JsonResult(notes,options) {
StatusCode = 200 };
                }
                return new JsonResult("Заметки не найдены") {
StatusCode = 200 };
          }

          [HttpGet("SortNotesByName")]
          public IActionResult SortNotesByName()
          {
                var elemsId = _elements.Elements.Where(e =>
e.user_id == CurrentUser.UserId && e.elem_type == false).Select(e
=> e.elem_id).ToList();
                var notes = _notes.Notes.Where(n =>
elemsId.Contains(n.elem_id)).OrderBy(c => c.note_name).ToList();
                if (notes.Count != 0)
                {
```

```
                    var options = new JsonSerializerOptions
                    {
                         Converters = { new DateOnlyConverter() }
                    };
                    return new JsonResult(notes, options) {
StatusCode = 200 };
               }
               return new JsonResult("Заметки не найдены") {
StatusCode = 200 };
          }

          //месяц считывается числом
          [HttpPost("FilterByDate")]
          public IActionResult FilterByDate([FromForm][Required]
int month, [FromForm][Required] int year)
          {
               if (month <= 12 && month > 0)
               {
                    if (year >= 2020 && year <= 2024)
                    {
                         var elemsId = _elements.Elements.Where(e
=> e.user_id == CurrentUser.UserId && e.elem_type ==
false).Select(e => e.elem_id).ToList();
                         var notes = _notes.Notes.Where(n =>
elemsId.Contains(n.elem_id)).Where(c =>
c.note_create.Month.Equals(month) &&
c.note_create.Year.Equals(year)).ToList();
                         if (notes.Count != 0)
                         {
                              var options = new
JsonSerializerOptions
                              {
                                   Converters = { new
DateOnlyConverter() }
                              };
                              return new JsonResult(notes, options)
{ StatusCode = 200 };
                         }
                    }
                    else {
                         List<string> errors = new List<string> {
"Некорректно введен год" };
                         return new JsonResult(errors)
                         { StatusCode = 400 }; // BadRequest
                    }
               }
               else {
                    List<string> errors = new List<string> {
"Некорректно введен месяц" };
                    return new JsonResult(errors)
                    { StatusCode = 400 }; // BadRequest
               }
```

```
                return new JsonResult("Заметки не найдены") {
StatusCode = 200 };
            }
        }
}
```

Файл NotificationController.cs

```
using Microsoft.AspNetCore.Http;
using Microsoft.AspNetCore.Mvc;
using Microsoft.AspNetCore.Mvc.ModelBinding;
using System.ComponentModel.DataAnnotations;
using WebApplication1.Entities;
using WebApplication1;
using WebApplication1.Controllers;
using System.Xml.Linq;
using System.Text.Json;
using WebApplication1.JsonConverter;

namespace ContactBook.Controllers
{
    [Route("[controller]")]
    [ApiController]
    public class NotificationController : Controller
    {
        private readonly NotificationRepository _notifications;
        private readonly EventRepository _events;
        private readonly FrequencyRepository _frequencies;
        private readonly ContactRepository _contacts;
        private readonly ElementRepository _elements;
        private readonly NoteRepository _notes;

        public NotificationController(NoteRepository notes,
NotificationRepository notifications, EventRepository events,
FrequencyRepository frequencies,
                                                    ContactRepository
contacts, ElementRepository elements)
        {
            _notes = notes;
            _notifications = notifications;
            _events = events;
            _frequencies = frequencies;
            _contacts = contacts;
            _elements = elements;
        }

        // Получение всех напоминаний пользователя
        [HttpGet("GetNotifications")]
        public IActionResult GetNotifications()
        {
            var notifications =
_notifications.Notifications.Where(e => e.user_id ==
CurrentUser.UserId).ToList();
```

```
                var allNotifications = new
List<NotificationAndInformation>();
                var events = new EventController(_events);
                var frequencies = new
FrequencyController(_frequencies);
                foreach (var notification in notifications)
                {
                     var notificationWithInformation = new
NotificationAndInformation();
                     notificationWithInformation.notification =
notification;
                     notificationWithInformation.ev =
_events.Events.FirstOrDefault(n => n.event_id ==
notification.ev_id);
                     notificationWithInformation.frequency =
_frequencies.Frequencies.FirstOrDefault(n => n.freq_id ==
notification.freq_id);

     allNotifications.Add(notificationWithInformation);
                }

                var options = new JsonSerializerOptions
                {
                     Converters = { new DateOnlyConverter(), new
TimeOnlyConverter() }
                };
                return new JsonResult(allNotifications, options) {
StatusCode = 200 };
          }

          // Получение одного напоминания
          [HttpGet("GetOneNotification/{id}")]
          public IActionResult GetOneNotification(int id)
          {
                var notification =
_notifications.Notifications.FirstOrDefault(e => e.user_id ==
CurrentUser.UserId && e.notif_id == id);
                if (notification != null)
                {
                     var events = new EventController(_events);
                     var frequencies = new
FrequencyController(_frequencies);
                     var notificationWithInformation = new
NotificationAndInformation();
                     notificationWithInformation.notification =
notification;
                     notificationWithInformation.ev =
_events.Events.FirstOrDefault(n => n.event_id ==
notification.ev_id);
                     notificationWithInformation.frequency =
_frequencies.Frequencies.FirstOrDefault(n => n.freq_id ==
notification.freq_id);
```

```
                    var options = new JsonSerializerOptions
                    {
                         Converters = { new DateOnlyConverter(),
new TimeOnlyConverter() }
                    };
                    return new JsonResult(notification, options) {
StatusCode = 200 };
               }
               else
               {
                    List<string> errors = new List<string> {
"Уведомление не найдено" };
                    return new JsonResult(errors)
                    { StatusCode = 404 }; // NotFound
               }
          }

          // Создание нового напоминания для контакта
          [HttpPost("CreateNotificationForContact")]
          public IActionResult CreateNotificationForContact(int
elem_id)
          {
               //получаем сам контакт
               var elemsId = _elements.Elements.Where(e =>
e.user_id == CurrentUser.UserId && e.elem_type == true).Select(e
=> e.elem_id).ToList();
               var contacts = _contacts.Contacts.Where(n =>
elemsId.Contains(n.elem_id));
               var contact = contacts.FirstOrDefault(n => n.elem_id
== elem_id);
               if (contact == null)
               {
                    List<string> errors = new List<string> {
"Выберите контакт для создания напоминания" };
                    return new JsonResult(errors)
                    { StatusCode = 404 }; // NotFound
               }

               Notification newNotification = new Notification();
               newNotification.elem_id = elem_id;
               newNotification.user_id = CurrentUser.UserId;
               newNotification.notif_name = "День рождения";
               newNotification.notif_desc = contact.fio + "
празднует день рождения. Не забудьте поздравить";
               newNotification.notif_type = true;
               newNotification.freq_id = 6;
               newNotification.ev_id = 1;

               var oldDate = contact.cont_birth;
               var newDate = oldDate.AddYears(DateTime.Today.Year -
oldDate.Year + 1);
               newNotification.notif_date = newDate;
               newNotification.notif_time = new TimeOnly(12, 0, 0);
```

```
                _notifications.Notifications.Add(newNotification);
                _notifications.SaveChanges();

                var fullNotification = new
NotificationAndInformation();
                fullNotification.notification = newNotification;

                var events = new EventController(_events);
                var frequencies = new
FrequencyController(_frequencies);

                fullNotification.ev =
_events.Events.FirstOrDefault(n => n.event_id ==
newNotification.ev_id);
                fullNotification.frequency =
_frequencies.Frequencies.FirstOrDefault(n => n.freq_id ==
newNotification.freq_id);

                var options = new JsonSerializerOptions
                {
                     Converters = { new DateOnlyConverter(), new
TimeOnlyConverter() }
                };

                return new JsonResult(fullNotification, options) {
StatusCode = 200 };
          }

          //создание напоминания для заметки
          [HttpPost("CreateNotificationForNote")]
          public IActionResult CreateNotificationForNote([FromForm]
NotificationModel notification, int elem_id)
          {
                var elemsId = _elements.Elements.Where(e =>
e.user_id == CurrentUser.UserId && e.elem_type == false).Select(e
=> e.elem_id).ToList();
                var notes = _notes.Notes.Where(n =>
elemsId.Contains(n.elem_id));
                var note = notes.FirstOrDefault(n => n.elem_id ==
elem_id);
                if (note != null)
                {
                     if (ModelState.IsValid)
                     {
                          Notification newNotification = new
Notification();
                          newNotification.elem_id = elem_id;
                          newNotification.user_id =
CurrentUser.UserId;
                          newNotification.notif_name =
notification.notif_name;
                          newNotification.notif_desc =
notification.notif_desc;
```

```
                        newNotification.notif_type = true;
                        newNotification.freq_id =
notification.freq_id;
                        newNotification.ev_id =
notification.ev_id;
                        if (notification.freq_id == 1)
                        {
                             List<string> answer = new
List<string> { "О данном событии не будет напоминаний" };
                             return new JsonResult(answer) {
StatusCode = 200 };
                        }
                        else if (notification.freq_id == 2)
                        {
                             int currentYear =
newNotification.notif_date.Year;
                             newNotification.notif_date =
DateOnly.FromDateTime(DateTime.Now).AddDays(1);
                             newNotification.notif_time = new
TimeOnly(12, 0, 0);
                        }
                        else if (notification.freq_id == 3)
                        {
                             newNotification.notif_date =
DateOnly.FromDateTime(DateTime.Now).AddDays(7);
                             newNotification.notif_time = new
TimeOnly(12, 0, 0);
                        }
                        else if (notification.freq_id == 4)
                        {
                             newNotification.notif_date =
DateOnly.FromDateTime(DateTime.Now).AddMonths(1);
                             newNotification.notif_time = new
TimeOnly(12, 0, 0);
                        }
                        else if (notification.freq_id == 5)
                        {
                             newNotification.notif_date =
DateOnly.FromDateTime(DateTime.Now).AddYears(1);
                             newNotification.notif_time = new
TimeOnly(12, 0, 0);
                        }
                        else if (notification.freq_id == 6)
                        {
                             if (notification.notif_date == null)
                             {
                                  List<string> errors = new
List<string> { "Введите дату уведомления" };
                                  return new JsonResult(errors)
                                  { StatusCode = 400 };
                             }
                             DateOnly newDate;
```

```
                                if
(DateOnly.TryParse(notification.notif_date, out newDate))
                                {
                                     newNotification.notif_date =
newDate;
                                }
                                else
                                {
                                     List<string> errors = new
List<string> { "Неверный формат ввода даты напоминания" };
                                     return new JsonResult(errors)
                                     { StatusCode = 400 }; //
BadRequest
                                }
                                newNotification.notif_time = new
TimeOnly(12, 0, 0);
                                newNotification.notif_time = new
TimeOnly(12, 0, 0);
                           }
                           else
                           {
                                List<string> errors = new
List<string> { "Выберите переодичность напоминания" };
                                return new JsonResult(errors)
                                { StatusCode = 400 };
                           }

     _notifications.Notifications.Add(newNotification);
                           _notifications.SaveChanges();

                           var fullNotification = new
NotificationAndInformation();
                           fullNotification.notification =
newNotification;

                           fullNotification.ev =
_events.Events.FirstOrDefault(n => n.event_id ==
newNotification.ev_id);
                           fullNotification.frequency =
_frequencies.Frequencies.FirstOrDefault(n => n.freq_id ==
newNotification.freq_id);

                           var options = new JsonSerializerOptions
                           {
                                Converters = { new
DateOnlyConverter(), new TimeOnlyConverter() }
                           };
                           return new JsonResult(fullNotification,
options) { StatusCode = 200 };
                      }
                      else
                      {
                           List<string> errors = new List<string>();
```

```
                    foreach (var item in ModelState)
                    {
                         if (item.Value.ValidationState ==
ModelValidationState.Invalid)
                         {
                              foreach (var error in
item.Value.Errors)
                              {

     errors.Add(error.ErrorMessage);
                              }
                         }
                    }
                    return new JsonResult(errors)
                    { StatusCode = 400 }; // BadRequest
               }
          }
          else
          {
               List<string> errors = new List<string> {
"Выберите заметку для создания напоминания" };
               return new JsonResult(errors)
               { StatusCode = 404 }; // NotFound
          }
     }

     // Изменение напоминания для заметки
     [HttpPut("UpdateNotificationForNote/{id}")]
     public IActionResult UpdateNotificationForNote(int
idNote, int id, [FromForm] NotificationModel updateNotification)
     {
          var elemsId = _elements.Elements.Where(e =>
e.user_id == CurrentUser.UserId && e.elem_type == false).Select(e
=> e.elem_id).ToList();
          var notes = _notes.Notes.Where(n =>
elemsId.Contains(n.elem_id));
          var note = notes.FirstOrDefault(n => n.elem_id ==
idNote);
          if (note != null)
          {
               if (ModelState.IsValid)
               {
                    Notification newNotification =
_notifications.Notifications.FirstOrDefault(n => n.notif_id ==
id);
                    if (newNotification == null)
                    {
                         List<string> errors = new
List<string> { "Напоминание не найдено" };
                         return new JsonResult(errors)
                         { StatusCode = 404 }; // NotFound
                    }
                    newNotification.elem_id = idNote;
```

```
                        newNotification.notif_name =
updateNotification.notif_name;
                        newNotification.notif_desc =
updateNotification.notif_desc;
                        newNotification.notif_type = true;
                        newNotification.freq_id =
updateNotification.freq_id;
                        newNotification.ev_id =
updateNotification.ev_id;

                        if (updateNotification.freq_id == 1)
                        {
                            List<string> answer = new
List<string> { "О данном событии не будет напоминаний" };
                            return new JsonResult(answer) {
StatusCode = 200 };
                        }
                        else if (updateNotification.freq_id == 2)
                        {
                            int currentYear =
newNotification.notif_date.Year;
                            newNotification.notif_date =
DateOnly.FromDateTime(DateTime.Now).AddDays(1);
                            newNotification.notif_time = new
TimeOnly(12, 0, 0);

    Console.WriteLine(newNotification.notif_date);
                        }
                        else if (updateNotification.freq_id == 3)
                        {
                            newNotification.notif_date =
DateOnly.FromDateTime(DateTime.Now).AddDays(7);
                            newNotification.notif_time = new
TimeOnly(12, 0, 0);
                        }
                        else if (updateNotification.freq_id == 4)
                        {
                            newNotification.notif_date =
DateOnly.FromDateTime(DateTime.Now).AddMonths(1);
                            newNotification.notif_time = new
TimeOnly(12, 0, 0);
                        }
                        else if (updateNotification.freq_id == 5)
                        {
                            newNotification.notif_date =
DateOnly.FromDateTime(DateTime.Now).AddYears(1);
                            newNotification.notif_time = new
TimeOnly(12, 0, 0);
                        }
                        else if (updateNotification.freq_id == 6)
                        {
                            if (updateNotification.notif_date ==
null)
```

```
                                {
                                    List<string> errors = new
List<string> { "Введите дату уведомления" };
                                    return new JsonResult(errors)
                                    { StatusCode = 400 };
                                }
                                DateOnly newDate;
                                if
(DateOnly.TryParse(updateNotification.notif_date, out newDate))
                                {
                                    newNotification.notif_date =
newDate;
                                }
                                else
                                {
                                    List<string> errors = new
List<string> { "Неверный формат ввода даты напоминания" };
                                    return new JsonResult(errors)
                                    { StatusCode = 400 }; //
BadRequest
                                }
                                newNotification.notif_time = new
TimeOnly(12, 0, 0);
                            }
                            else
                            {
                                List<string> errors = new
List<string> { "Выберите переодичность напоминания" };
                                return new JsonResult(errors)
                                { StatusCode = 400 };
                            }
                            _notifications.SaveChanges();

                            var fullNotification = new
NotificationAndInformation();
                            fullNotification.notification =
newNotification;

                            fullNotification.ev =
_events.Events.FirstOrDefault(n => n.event_id ==
newNotification.ev_id);
                            fullNotification.frequency =
_frequencies.Frequencies.FirstOrDefault(n => n.freq_id ==
newNotification.freq_id);

                            var options = new JsonSerializerOptions
                            {
                                Converters = { new
DateOnlyConverter(), new TimeOnlyConverter() }
                            };

                            return new
JsonResult(fullNotification,options) { StatusCode = 200 };
```

```
                    }
                    else
                    {
                         List<string> errors = new List<string>();
                         foreach (var item in ModelState)
                         {
                              if (item.Value.ValidationState ==
ModelValidationState.Invalid)
                              {
                                   foreach (var error in
item.Value.Errors)
                                   {

     errors.Add(error.ErrorMessage);
                                   }
                              }
                         }
                         return new JsonResult(errors)
                         { StatusCode = 400 }; // BadRequest
                    }
               }
               else
               {
                    List<string> errors = new List<string> {
"Выберите заметку для изменения напоминания" };
                    return new JsonResult(errors)
                    { StatusCode = 404 }; // NotFound
               }
          }

          //изменение напоминания для контакта
          [HttpPut("UpdateNotificationForContact/{id}")]
          public IActionResult UpdateNotificationForContact(int
idContact, int id)
          {
               var elemsId = _elements.Elements.Where(e =>
e.user_id == CurrentUser.UserId && e.elem_type == true).Select(e
=> e.elem_id).ToList();
               var contacts = _contacts.Contacts.Where(n =>
elemsId.Contains(n.elem_id));
               var contact = contacts.FirstOrDefault(n => n.elem_id
== idContact);
               if (contact != null)
               {
                    if (ModelState.IsValid)
                    {
                         Notification newNotification =
_notifications.Notifications.FirstOrDefault(n => n.notif_id ==
id);
                         if (newNotification == null)
                         {
                              List<string> errors = new
List<string> { "Напоминание не найдено" };
```

```
                              return new JsonResult(errors)
                              { StatusCode = 404 }; // NotFound
                         }
                         var oldDate = contact.cont_birth;
                         var newDate =
oldDate.AddYears(DateTime.Today.Year - oldDate.Year + 1);
                         newNotification.notif_date = newDate;
                         newNotification.notif_time = new
TimeOnly(12, 0, 0);

                         _notifications.SaveChanges();

                         var fullNotification = new
NotificationAndInformation();
                         fullNotification.notification =
newNotification;

                         fullNotification.ev =
_events.Events.FirstOrDefault(n => n.event_id ==
newNotification.ev_id);
                         fullNotification.frequency =
_frequencies.Frequencies.FirstOrDefault(n => n.freq_id ==
newNotification.freq_id);

                         var options = new JsonSerializerOptions
                         {
                              Converters = { new
DateOnlyConverter(), new TimeOnlyConverter() }
                         };

                         return new JsonResult(fullNotification,
options) { StatusCode = 200 };
                    }
                    else
                    {
                         List<string> errors = new List<string>();
                         foreach (var item in ModelState)
                         {
                              if (item.Value.ValidationState ==
ModelValidationState.Invalid)
                              {
                                   foreach (var error in
item.Value.Errors)
                                   {

     errors.Add(error.ErrorMessage);
                                   }
                              }
                         }
                         return new JsonResult(errors)
                         { StatusCode = 400 }; // BadRequest
                    }
               }
```

```
                else
                {
                    List<string> errors = new List<string> {
"Выберите контакт для изменения напоминания" };
                    return new JsonResult(errors)
                    { StatusCode = 404 }; // NotFound
                }
            }

            // Удаление напоминания
            [HttpDelete("DeleteNotification/{id}")]
            public IActionResult DeleteNotification(int id)
            {
                Notification notification =
_notifications.Notifications.FirstOrDefault(n => n.notif_id ==
id);
                if (notification == null)
                {
                    List<string> errors = new List<string> {
"Уведомление не найдено" };
                    return new JsonResult(errors)
                    { StatusCode = 404 }; // NotFound
                }

                _notifications.Notifications.Remove(notification);
                _notifications.SaveChanges();
                return new JsonResult("Уведомление успешно удалено")
{ StatusCode = 200 };
            }

            //получение прошедших и текущих напоминаний
            [HttpGet("GetNotificationsBeforeToday")]
            public IActionResult GetNotificationsBeforeToday()
            {
                var notifications =
_notifications.Notifications.Where(e => e.user_id ==
CurrentUser.UserId).ToList();
                var allNotifications = new
List<NotificationAndInformation>();
                var events = new EventController(_events);
                var frequencies = new
FrequencyController(_frequencies);
                foreach (var notification in notifications)
                {
                    if (notification.notif_date <=
DateOnly.FromDateTime(DateTime.Now))
                    {
                        var notificationWithInformation = new
NotificationAndInformation();
                        notificationWithInformation.notification =
notification;
```

```
                        notificationWithInformation.ev =
_events.Events.FirstOrDefault(n => n.event_id ==
notification.ev_id);
                        notificationWithInformation.frequency =
_frequencies.Frequencies.FirstOrDefault(n => n.freq_id ==
notification.freq_id);

     allNotifications.Add(notificationWithInformation);
                    }
               }
               var options = new JsonSerializerOptions
               {
                    Converters = { new DateOnlyConverter(), new
TimeOnlyConverter() }
               };
               return new JsonResult(allNotifications,options) {
StatusCode = 200 };
          }
     }
}
```

Файл Contact.cs

```
using System.ComponentModel.DataAnnotations.Schema;
using System.ComponentModel.DataAnnotations;
using WebApplication1.CustomAttributes;

namespace WebApplication1.Entities
{
     [Table("contact")]
     public class Contact
     {
          [Key]
          [DatabaseGenerated(DatabaseGeneratedOption.Identity)]
          public int elem_id { get; set; }
          [Required]
          public string fio { get; set; }
          [Required]
          public DateOnly cont_birth { get; set; }
          public string? address { get; set; }
          public string? cont_work { get; set; }
          public string? other { get; set; }
     }

     public class ContactAndInformation
     {
          public Contact contact { get; set; }
          public List<PhoneNumber> phoneNumbers { get; set; }
          public List<Device> devices { get; set; }
          public List<Email> emails { get; set; }
          public List<SocialNetwork> socialNetworks { get; set; }
          public List<SN_Name> socialNetworkNames { get; set; }

     }
```

```
    public class ContactModel
    {
        [Required]
        [RegularExpression(@"^[а-яА-ЯёЁa-zA-Z_ ]+ [а-яА-ЯёЁa-zA-
Z_ ]+ ?[а-яА-ЯёЁa-zA-Z_ ]+$", ErrorMessage = "Некорректное ФИО для
контакта. Поле обязательно" +
            "должно содержать фамилию и имя русскими или
латинскими буквами")]
        [MinLength(1, ErrorMessage = "Минимальная длина ФИО - 1
символ")]
        [MaxLength(100, ErrorMessage = "Максимальная длина ФИО -
100 символов")]
        public string fio { get; set; }
        [Required]
        [DisplayFormat(DataFormatString = "{0:dd.MM.yyyy}")]
        [DateRange("1950-01-01", "2023-01-01", ErrorMessage =
"Дата рождения " +
            "должна быть в диапазоне от {0} до {1}")]
        [DataType(DataType.Date)]
        public string cont_birth { get; set; }
        [RegularExpression(@"^[а-яА-ЯёЁa-zA-Z0-9\.\-\,\s]+$",
ErrorMessage = "Некорректный адрес контакта. Адрес " +
            "может содержать буквы русского и латинского
алфавитов, цифры и знаки . , - , ,")]
        [MaxLength(100, ErrorMessage = "Максимальная длина адреса
- 100 символов")]
        public string? address { get; set; }
        [RegularExpression(@"^[а-яА-ЯёЁa-zA-Z0-9_ ]+$",
ErrorMessage = "Некорректное место работы контакта. Место работы "
+
            "может содержать буквы русского и латинского алфавитов
и цифры")]
        [MaxLength(100, ErrorMessage = "Максимальная длина места
работы - 100 символов")]
        public string? cont_work { get; set; }
        [RegularExpression(@"^[а-яА-ЯёЁa-zA-Z0-9\.\-\,\s ]+$",
ErrorMessage = "Некорректная дополнительная информация о контакте.
Дополнительная информация " +
            "может содержать буквы русского и латинского
алфавитов, цифры и знаки . , - , ,")]
        [MaxLength(200, ErrorMessage = "Максимальная длина
дополнительной информации - 200 символов")]
        public string? other { get; set; }
    }
}
```

Файл Device.cs

```
using System.ComponentModel.DataAnnotations.Schema;
using System.ComponentModel.DataAnnotations;

namespace WebApplication1.Entities
{
```

```
     [Table("device")]
     public class Device
     {
          [Key]
          [DatabaseGenerated(DatabaseGeneratedOption.Identity)]
          public int device_id { get; set; }
          [Required(ErrorMessage ="Название типа устройства -
обязательное поле")]
          public string phone_type { get; set; }
     }
}
```

Файл Element.cs

```
using System.ComponentModel.DataAnnotations;
using System.ComponentModel.DataAnnotations.Schema;

namespace WebApplication1.Entities
{
     [Table("elem")]
     public class Element
     {
          [Key]
          [DatabaseGenerated(DatabaseGeneratedOption.Identity)]
          public int elem_id { get; set; }
          [Required(ErrorMessage = "Тип элемента - обязательное
поле")]
          public bool elem_type { get; set; }
          [Required(ErrorMessage = "Идентификатор пользователя -
обязательное поле")]
          public int user_id { get; set; }
     }
}
```

Файл Note.cs

```
using Microsoft.AspNetCore.Identity;
using System.ComponentModel.DataAnnotations;
using System.ComponentModel.DataAnnotations.Schema;
namespace WebApplication1.Entities
{
     [Table("note")]
     public class Note
     {
          [Key]
          [DatabaseGenerated(DatabaseGeneratedOption.Identity)]
          public int elem_id { get; set; }
          public string note_name { get; set; }
          public DateOnly note_create { get; set; }
          public string? description { get; set; }
     }

     public class NoteCreateModel
     {
          [Required(ErrorMessage = "Название заметки - обязательное
поле")]
```

```
          [RegularExpression(@"^[a-яА-ЯёЁa-zA-Z0-9\.\-\,\s ]+$",
ErrorMessage = "Некорректное название заметки. Название " +
               "заметки может содержать буквы русского и латинского
алфавитов, цифры и знаки . , - , ,")]
          [MinLength(1, ErrorMessage = "Минимальная длина названия
заметки - 1 символ")]
          [MaxLength(50, ErrorMessage = "Максимальная длина
названия заметки - 50 символов")]
          public string note_name { get; set; }
          [RegularExpression(@"^[a-яА-ЯёЁa-zA-Z0-9\.\-\,\s ]+$",
ErrorMessage = "Некорректное описание заметки. Описание " +
               "заметки может содержать буквы русского и латинского
алфавитов, цифры и знаки . , - , ,")]
          [MaxLength(150, ErrorMessage = "Максимальная длина
описания заметки - 150 символов")]
          public string? description { get; set; }
     }
}
```

Файл Notification.cs

```
namespace WebApplication1.Entities
{
     [Table("notification")]
     public class Notification
     {
          [Key]
          [DatabaseGenerated(DatabaseGeneratedOption.Identity)]
          public int notif_id { get; set; }
          [Required]
          public string notif_name { get; set; }
          [Required]
          public DateOnly notif_date { get; set; }
          [Required]
          public int elem_id { get; set; }
          public string? notif_desc { get; set; }
          [Required]
          public bool notif_type { get; set; }
          [Required]
          public int freq_id { get; set; }
          public int? ev_id { get; set; }
          [Required]
          public int user_id { get; set; }
          [Required]
          public TimeOnly notif_time { get; set; }
     }

     public class NotificationModel
     {
          [Required(ErrorMessage ="Введите название уведомления")]
          [RegularExpression(@"^[a-яА-ЯёЁa-zA-Z0-9\.\-\,\s ]+$",
ErrorMessage = "Некорректное название уведомления. Название " +
             "уведомления может содержать буквы русского и
латинского алфавитов, цифры и знаки . , - , ,")]
```

```
          [MaxLength(100, ErrorMessage = "Максимальная длина адреса
- 100 символов")]
          public string notif_name { get; set; }
          [RegularExpression(@"^[a-яA-ЯёЁa-zA-Z0-9\.\-\,\s ]+$",
ErrorMessage = "Некорректное описание уведомления. Описание " +
              "может содержать буквы русского и латинского
алфавитов, цифры и знаки . , - , ,")]
          [MaxLength(100, ErrorMessage = "Максимальная длина адреса
- 100 символов")]
          public string? notif_desc { get; set; }
          [Required(ErrorMessage = "Выберите тип напоминания (со
звуком или без)")]
          public bool notif_type { get; set; }
          [Required(ErrorMessage = "Выберите частоту напоминания")]
          public int freq_id { get; set; }
          public int? ev_id { get; set; }
          [DisplayFormat(DataFormatString = "{0:dd.MM.yyyy}")]
          [DateRange("2023-01-01", "2026-01-01", ErrorMessage =
"Дата напоминания " +
               "должна быть в диапазоне от {0} до {1}")]
          [DataType(DataType.Date)]
          public string? notif_date{ get; set; }
     }

     public class NotificationAndInformation
     {
          [Required]
          public Notification notification { get; set; }
          public Event? ev { get; set; }
          [Required]
          public Frequency frequency { get; set; }
     }
}
```

Файл PhoneNumber.cs

```
namespace WebApplication1.Entities
{
     [Table("phonenumber")]
     public class PhoneNumber
     {
          [Key]
          [DatabaseGenerated(DatabaseGeneratedOption.Identity)]
          public int phone_id { get; set; }
          [Required]
          public string pn { get; set; }
          [Required(ErrorMessage = "Идентификатор контакта для
номера телефона - обязательное поле")]
          public int elem_id { get; set; }
          public int? device_id { get; set; }
     }

     public class PhoneNumberCreateModel
     {
```

```
        [Required(ErrorMessage = "Номер телефона - обязательное
поле")]
        [RegularExpression(@"^((\+7|7|8)+([0-9]){10})$",
ErrorMessage = "Номер телефона введен некорректно")]
        [MaxLength(12, ErrorMessage ="максимальная длина номера
телефона - 12 символов")]
        public string pn { get; set; }
        public int? device_id { get; set; }
    }
}
```

Приложение Г

(рекомендуемое)

Скриншоты результатов функционального тестирования

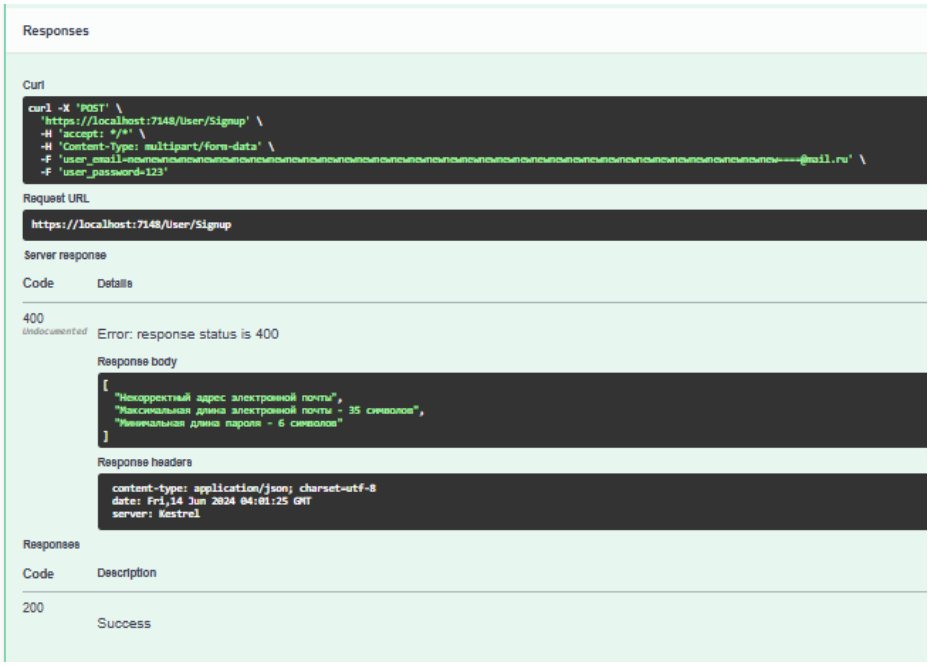


Рисунок Г.1 – Регистрация в системе с некорректными пользовательскими данными

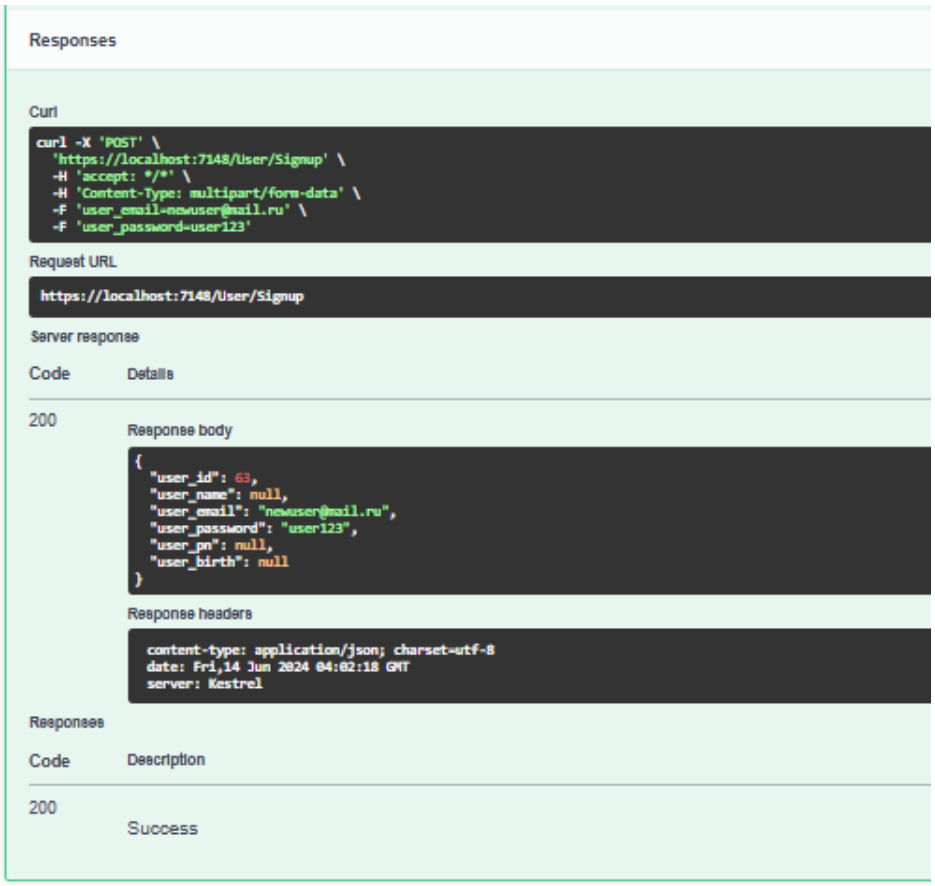


Рисунок Г.2 – Регистрация в системе с корректными пользовательскими данными

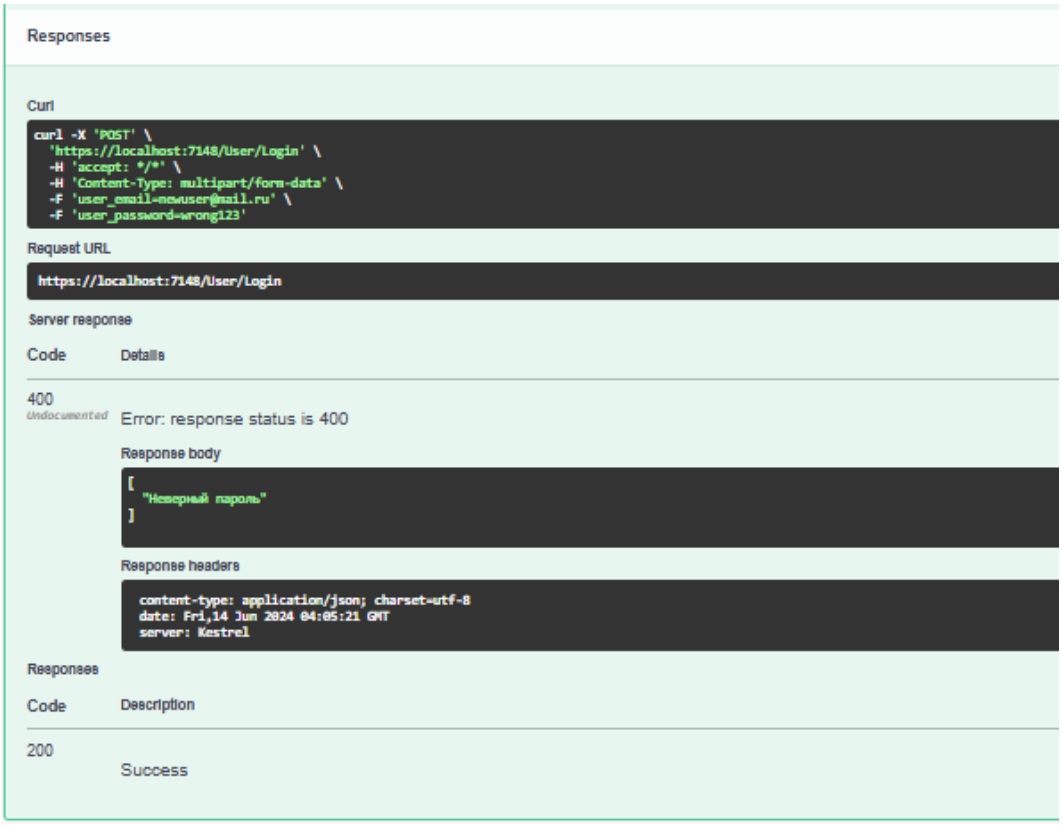


Рисунок Г.3 – Авторизация в системе с некорректными пользовательскими данными

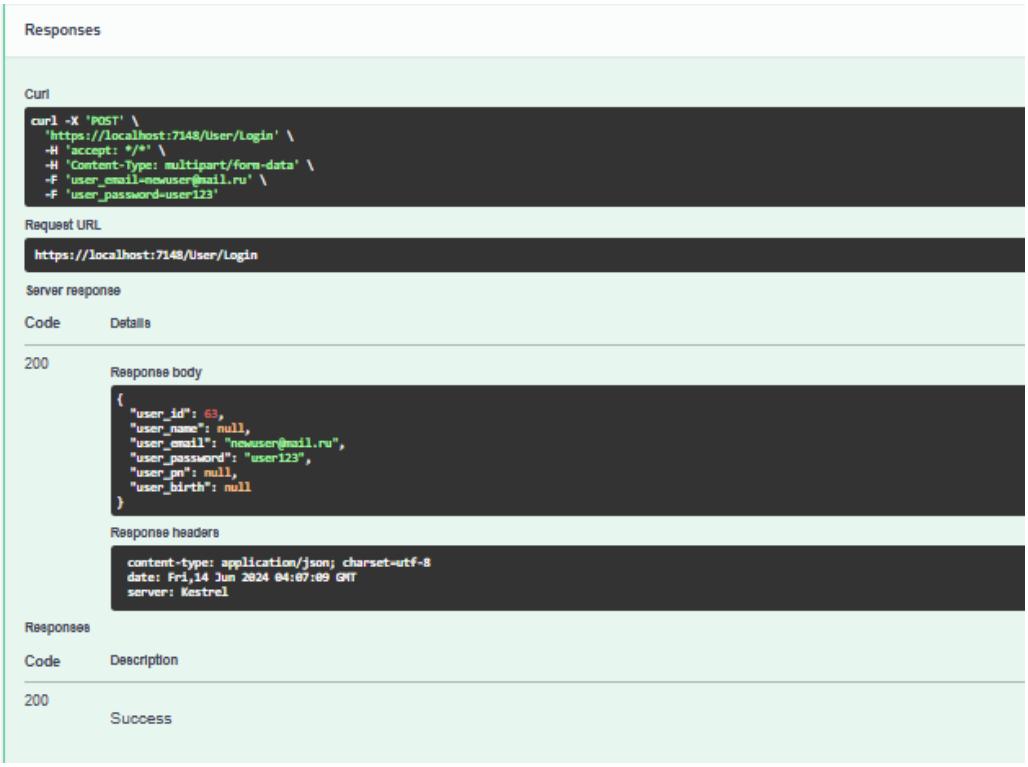


Рисунок Г.4 – Авторизация в системе с корректными пользовательскими данными

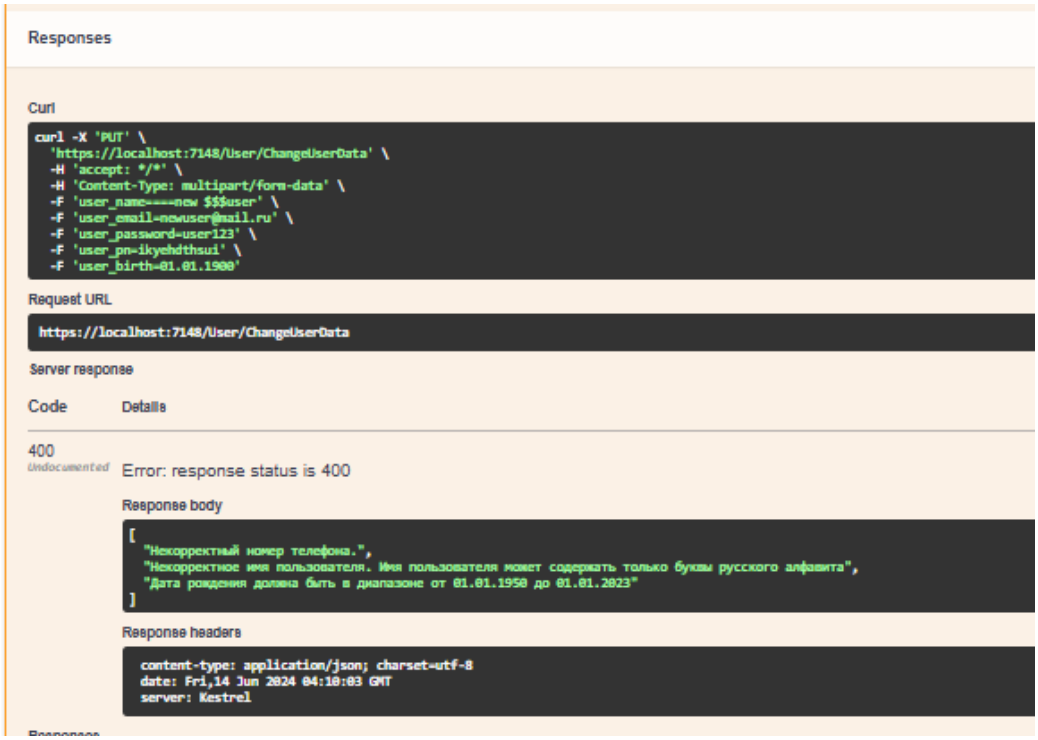


Рисунок Г.5 – Редактирование информации в личном кабинете с некорректными пользовательскими данными



Рисунок Г.6 – Редактирование информации в личном кабинете с корректными пользовательскими данными

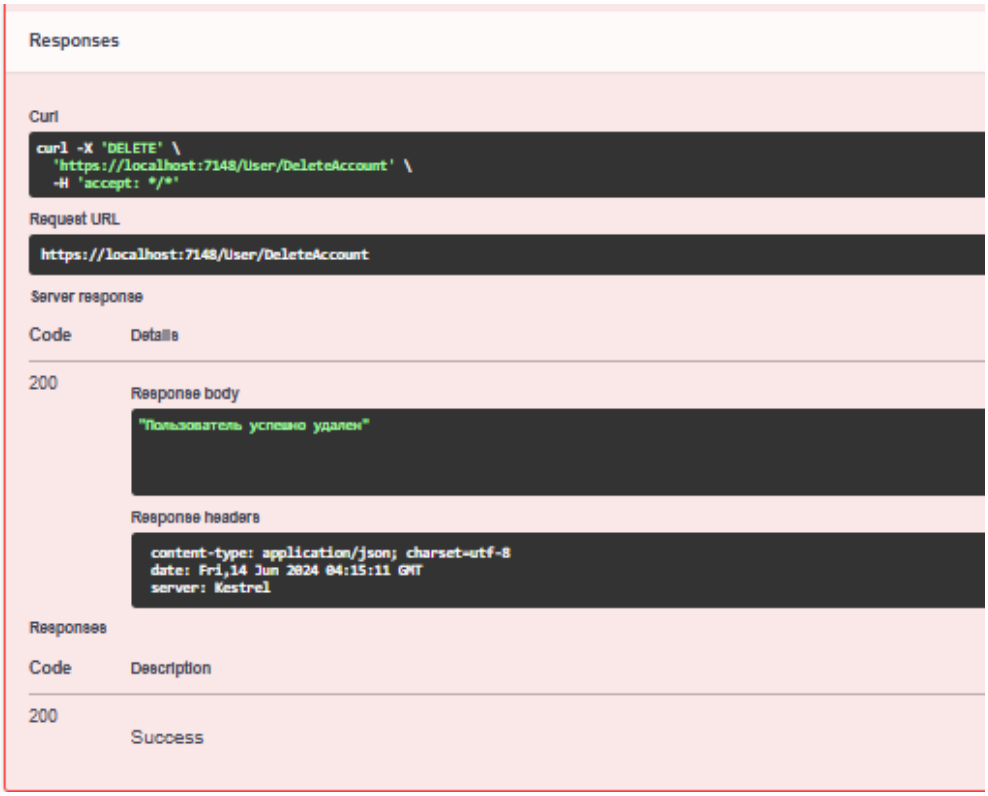


Рисунок Г.7 – Успешное удаление аккаунта

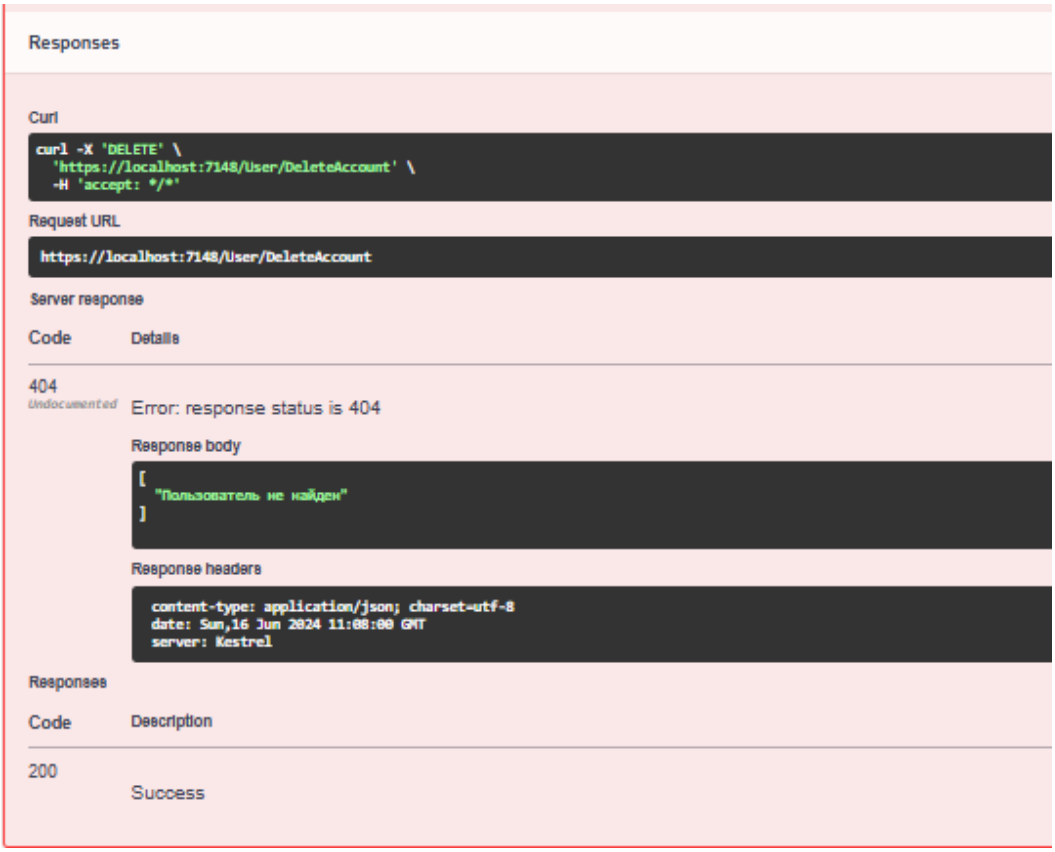


Рисунок Г.8 – Удаление аккаунта с некорректным уникальным идентификатором



Рисунок Г.9 – Изменение заметки с некорректными данными заметки



Рисунок Г.10 – Изменение заметки с корректными данными заметки

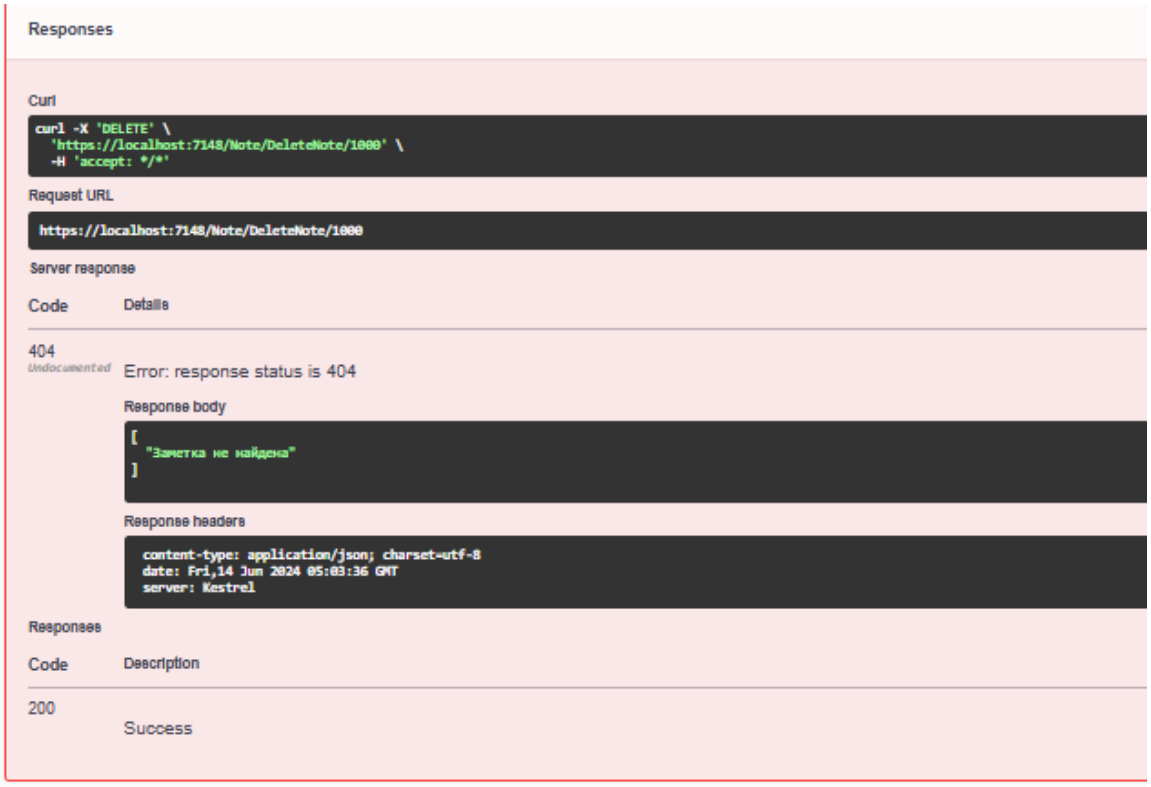


Рисунок Г.11 – Удаление заметки с некорректным уникальным идентификатором

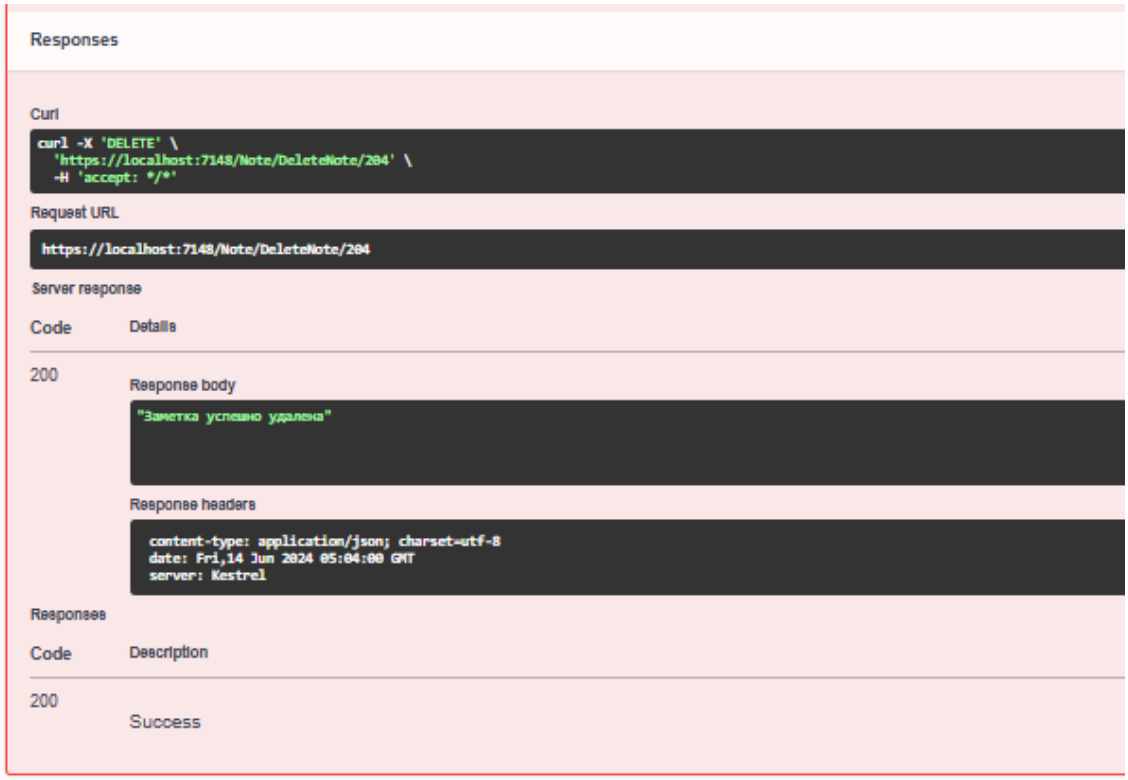


Рисунок Г.12 – Удаление заметки с корректным уникальным идентификатором

Responses

Curl

```
curl -X 'POST' \
  'https://localhost:7148/Contact/CreateContact' \
  -H 'accept: */*' \
  -H 'Content-Type: multipart/form-data' \
  -F 'fio=====' \
  -F 'cont_birth=01.01.1900' \
  -F 'address=====' \
  -F 'cont_work=====' \
  -F 'other=некоторая информация'
```

Request URL

```
https://localhost:7148/Contact/CreateContact
```

Server response

Code	Details
400 *Undocumented*	Error: response status is 400

Response body

```
[
  "Некорректное ФИО для контакта. Поле обязательнодолжно содержать фамилию и имя русскими или латинскими буквами",
  "Некорректный адрес контакта. Адрес может содержать буквы русского и латинского алфавитов, цифры и знаки . , - ' ,",
  "Некорректное место работы контакта. Место работы может содержать буквы русского и латинского алфавитов и цифры",
  "Дата рождения должна быть в диапазоне от 01.01.1950 до 01.01.2023"
]
```

Response headers

```
content-type: application/json; charset=utf-8
date: Fri,14 Jun 2024 05:05:02 GMT
server: Kestrel
```

Рисунок Г.13 – Добавление контакта с некорректными данными

Responses

Curl

```
curl -X 'POST' \
  'https://localhost:7148/Contact/CreateContact' \
  -H 'accept: */*' \
  -H 'Content-Type: multipart/form-data' \
  -F 'fio=Иванов Иван' \
  -F 'cont_birth=05.10.1989' \
  -F 'address=' \
  -F 'cont_work=' \
  -F 'other=некоторая информация'
```

Request URL

```
https://localhost:7148/Contact/CreateContact
```

Server response

Code	Details
200	Response body

```
{
  "contact": {
    "elem_id": 285,
    "fio": "Иванов Иван",
    "cont_birth": "1989-10-05",
    "address": null,
    "cont_work": null,
    "other": "некоторая информация"
  },
  "phoneNumbers": [],
  "devices": null,
  "emails": [],
  "socialNetworks": [],
  "socialNetworkNames": null
}
```

Рисунок Г.14 – Добавление контакта с корректными данными



Рисунок Г.15 – Изменение контакта с некорректным уникальным идентификатором



Рисунок Г.16 – Удаление контакта с некорректным уникальным идентификатором

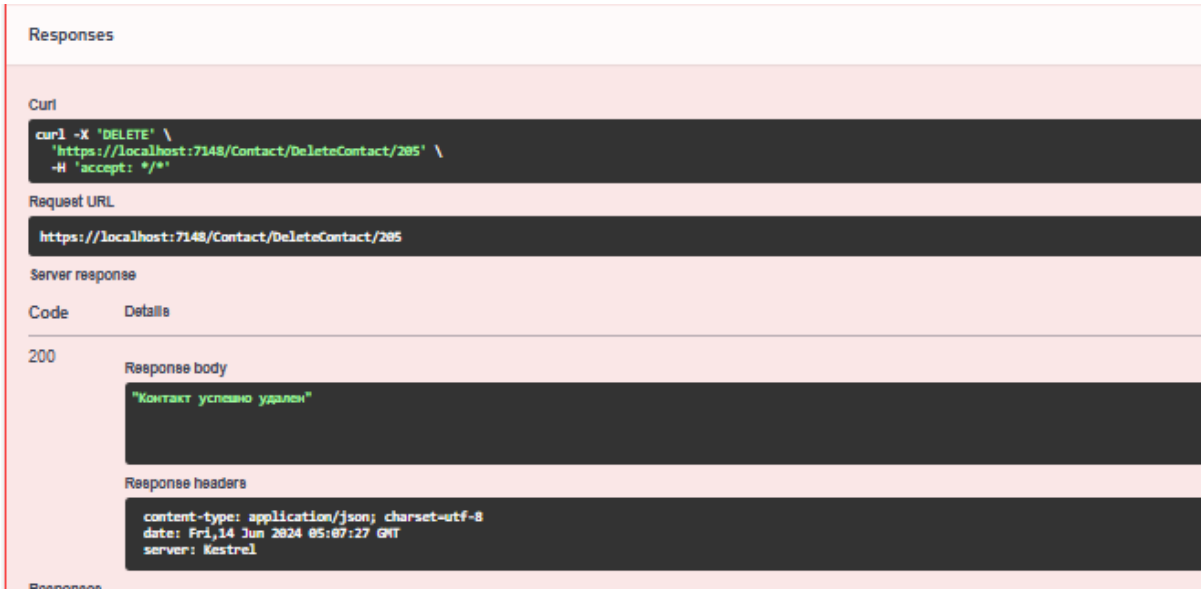


Рисунок Г.17 – Удаление контакта с корректным уникальным идентификатором

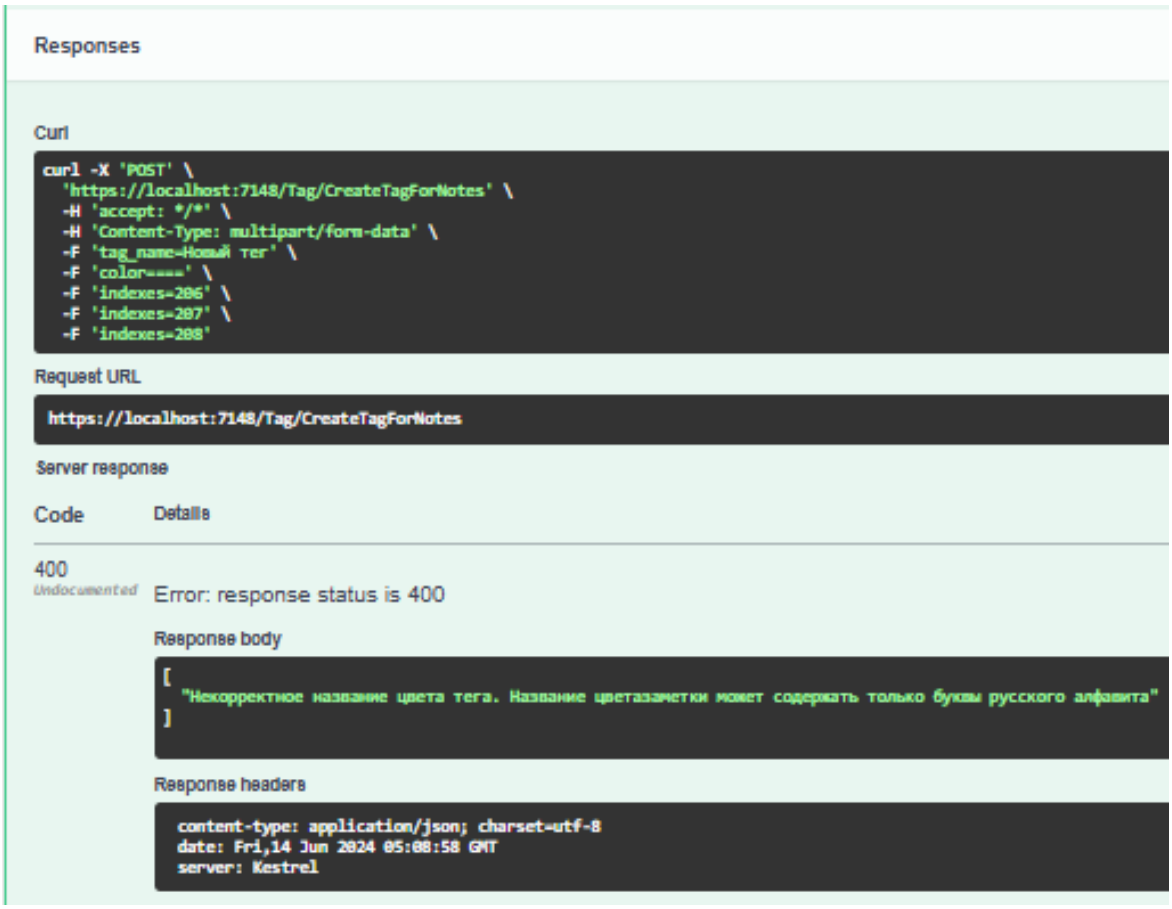


Рисунок Г.18 – Добавление тега с некорректными данными тега

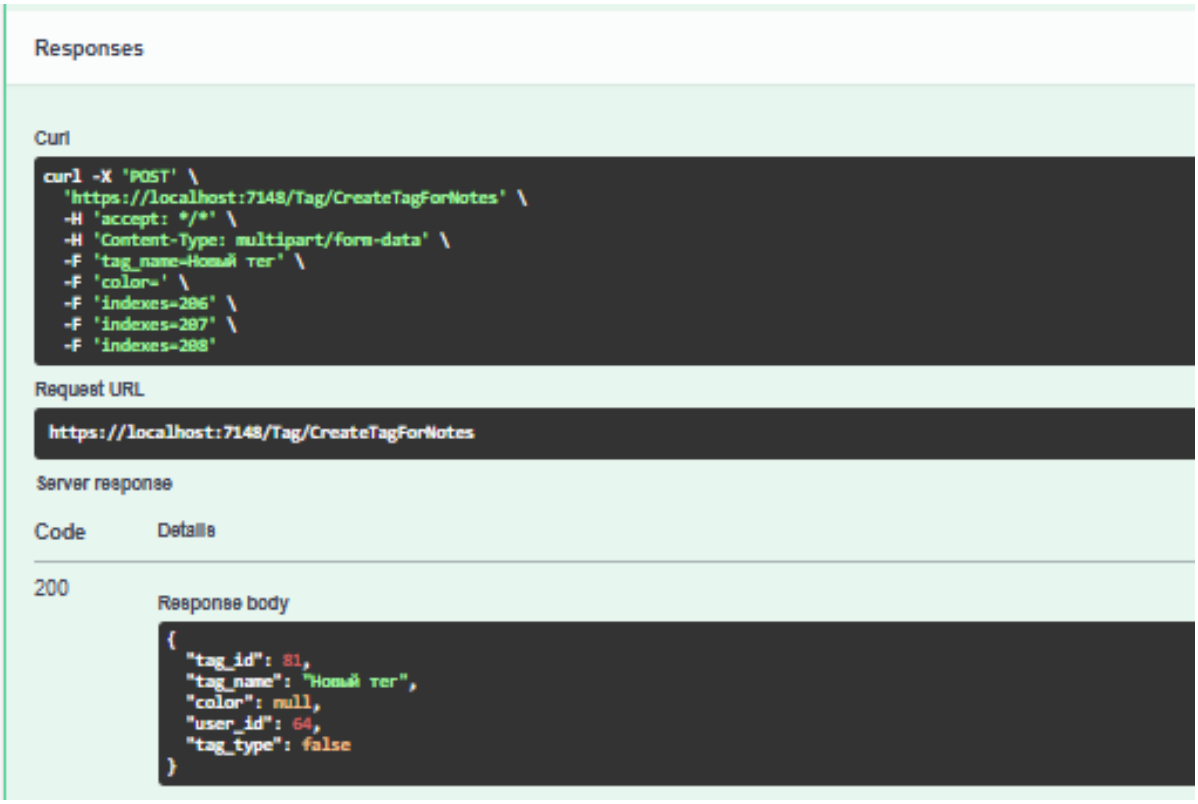


Рисунок Г.19 – Добавление тега с корректными данными тега



Рисунок Г.20 – Изменение тега с некорректным уникальным идентификатором

Responses

Curl

```
curl -X 'PUT' \
  'https://localhost:7148/Tag/UpdateTagForNotes/81' \
  -H 'accept: */*' \
  -H 'Content-Type: multipart/form-data' \
  -F 'tag_name=Измененный тег' \
  -F 'color=' \
  -F 'indexes=206' \
  -F 'indexes=207'
```

Request URL

```
https://localhost:7148/Tag/UpdateTagForNotes/81
```

Server response

Code	Details
200	Response body

```
{
  "tag": {
    "tag_id": 81,
    "tag_name": "Измененный тег",
    "color": null,
    "user_id": 64,
    "tag_type": false
  },
  "elements": [
    {
      "elem_id": 206,
      "note_name": "Новая заметка1",
      "note_create": "2024-06-14",
      "description": "Новое описание"
    },
    {
      "elem_id": 207,
      "note_name": "Новая заметка2",
      "note_create": "2024-06-14",
      "description": "Новое описание"
    }
  ]
}
```

Рисунок Г.21 – Изменение тега с корректными данными

Responses

Curl

```
curl -X 'DELETE' \
  'https://localhost:7148/Tag/DeleteTagForNotes/-1' \
  -H 'accept: */*'
```

Request URL

```
https://localhost:7148/Tag/DeleteTagForNotes/-1
```

Server response

Code	Details
404 *Undocumented*	Error: response status is 404

Response body

```
[
  "Тег не найден"
]
```

Response headers

```
content-type: application/json; charset=utf-8
date: Fri,14 Jun 2024 05:12:19 GMT
server: Kestrel
```

Рисунок Г.22 – Удаление тега с некорректным уникальным идентификатором

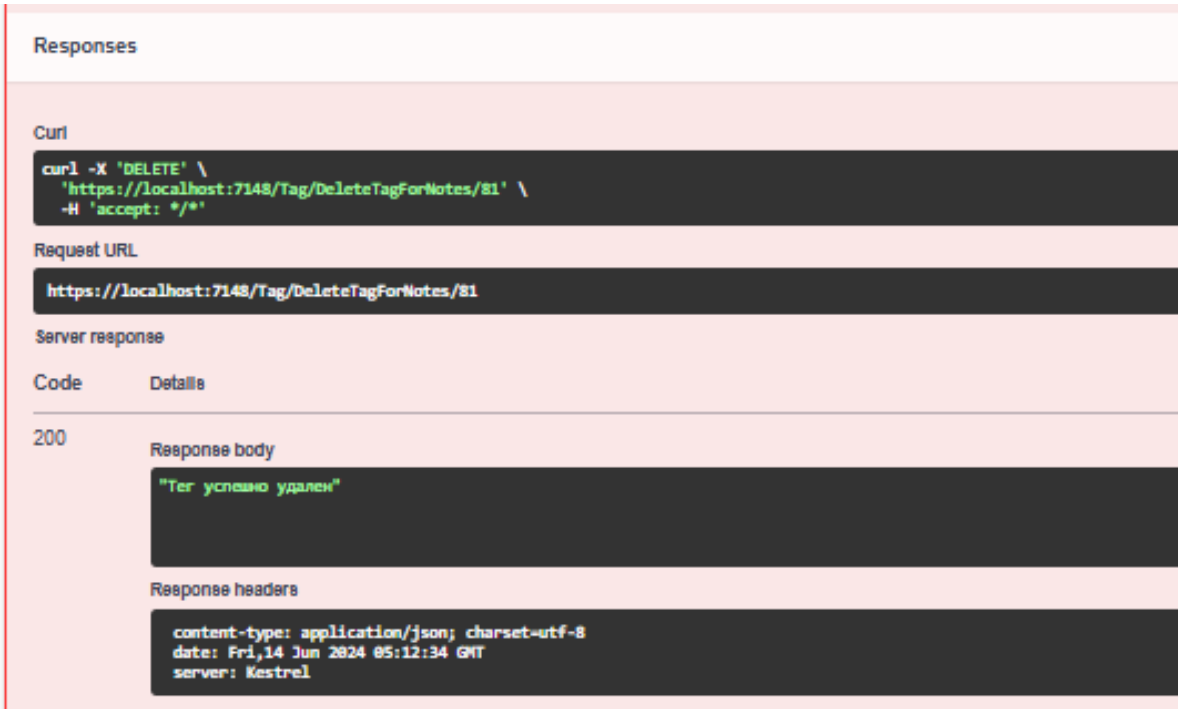


Рисунок Г.23 – Удаление тега с корректным уникальным идентификатором

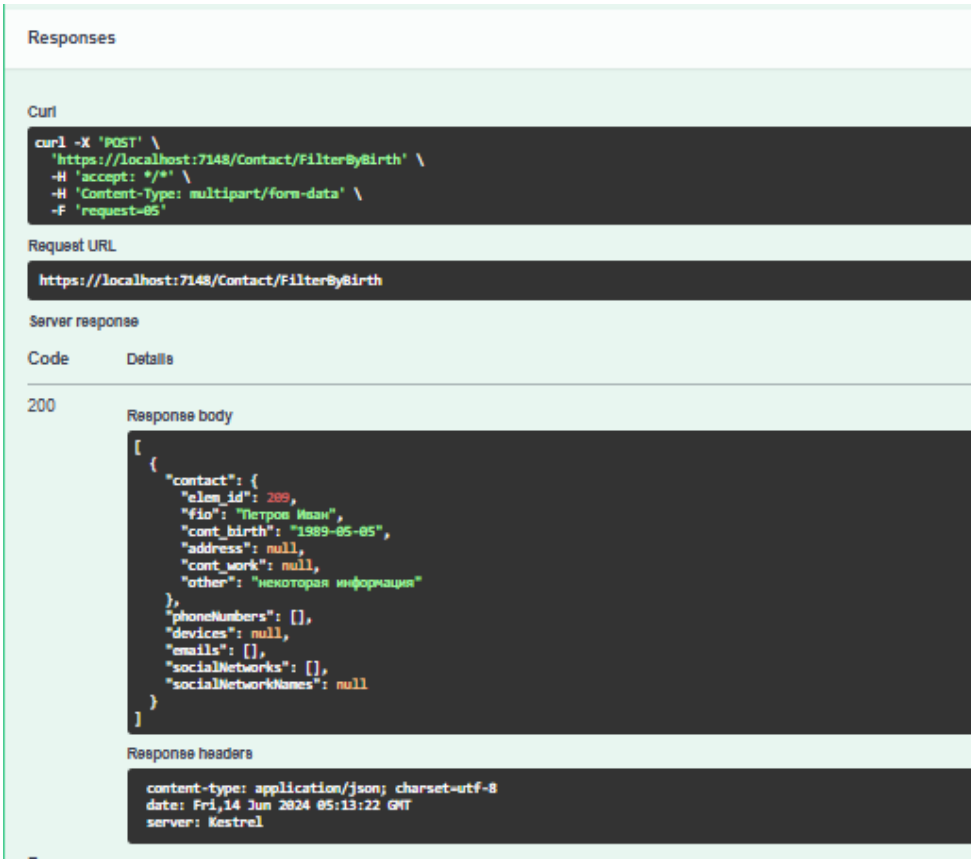


Рисунок Г.24 – Фильтрация контактов с корректным критерием для фильтрации

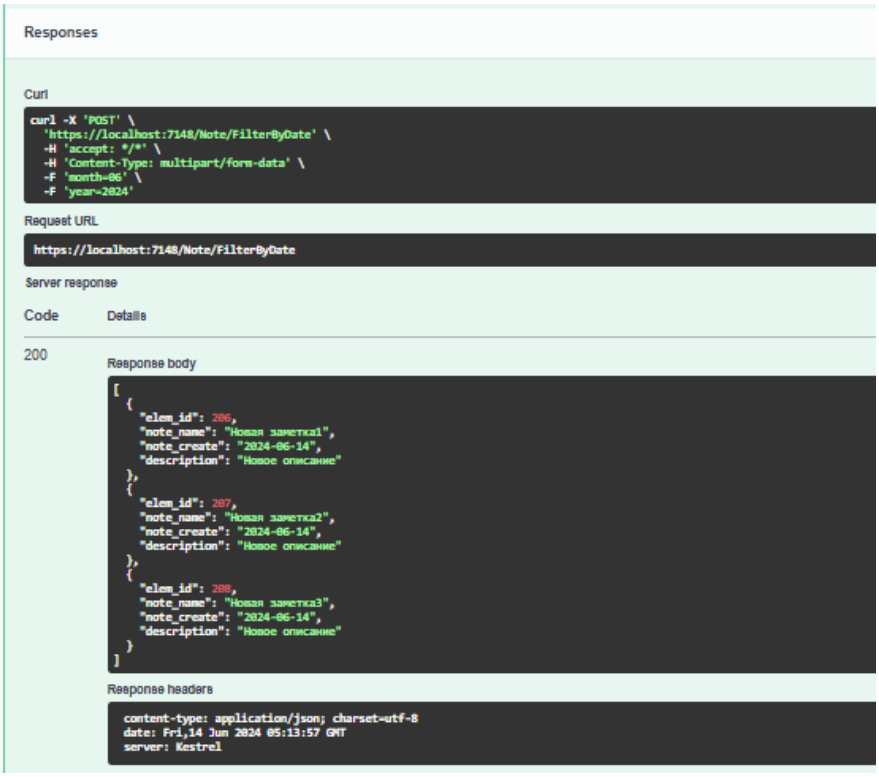


Рисунок Г.25 – Фильтрация заметок с корректным критерием для фильтрации

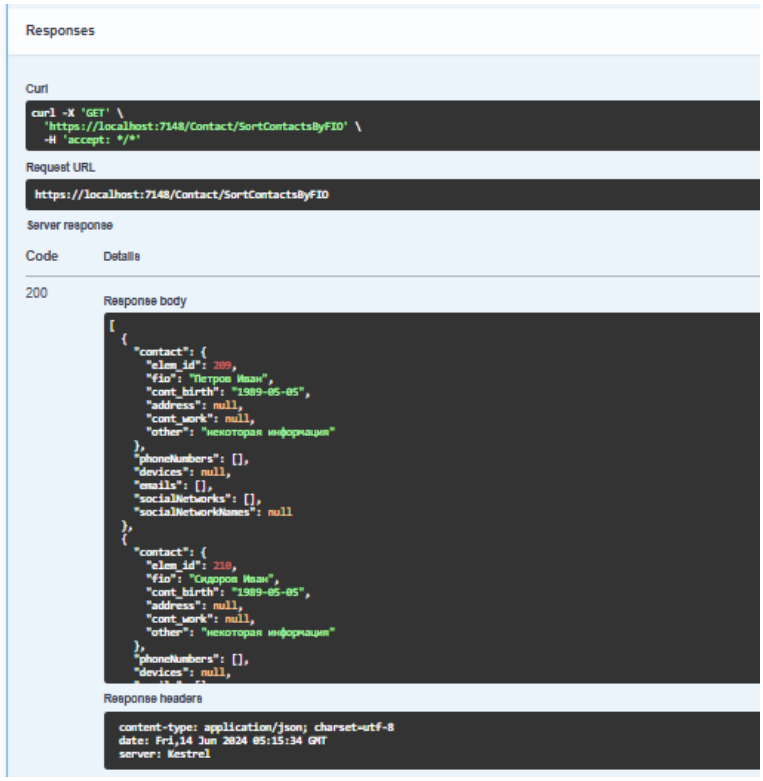


Рисунок Г.26 – Сортировка контактов по ФИО

Приложение Д

(рекомендуемое)

Листинг модульного тестирования

Файл UnitTest1.cs

```
using FluentAssertions;
using Microsoft.AspNetCore.Http;
using Microsoft.AspNetCore.Mvc;
using Microsoft.AspNetCore.Mvc.Testing;
using Microsoft.EntityFrameworkCore;
using Microsoft.Extensions.DependencyInjection;
using WebApplication1.Controllers;
using WebApplication1.Entities;
using WebApplication1;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Net;
using System.Net.Http;
using System.Net.Http.Json;
using System.Threading.Tasks;
using Xunit;
using Microsoft.VisualStudio.TestPlatform.TestHost;
using System.Xml.Linq;
using System.Text.Json;
using WebApplication1.JsonConverter;
using System.Diagnostics;

namespace WebApplication1.Tests.Controllers
{
     public class NoteControllerTests :
IClassFixture<WebApplicationFactory<Program>>
     {
          private readonly WebApplicationFactory<Program> _factory;

          public NoteControllerTests(WebApplicationFactory<Program>
factory)
          {
               _factory = factory;
          }

          [Fact]
          public async Task GetNotes_WhenNoNotes_ReturnsEmptyList()
          {
               // Arrange
               var client = _factory.CreateClient();
               client.DefaultRequestHeaders.Add("UserId", "61");
               CurrentUser.UserId = 61;

               // Act
               var response = await
client.GetAsync("/Note/GetNotes");

               // Assert
               response.StatusCode.Should().Be(HttpStatusCode.OK);
               var options = new JsonSerializerOptions
```

```
                {
                     Converters = { new DateOnlyConverter() }
                };
                var notes = await
response.Content.ReadFromJsonAsync<List<Note>>(options);
                notes.Should().BeEmpty();

           }

           [Fact]
           public async Task
GetNotes_WhenNotesExist_ReturnsListOfNotes()
           {
                // Arrange
                var client = _factory.CreateClient();
                client.DefaultRequestHeaders.Add("UserId", "55");
                CurrentUser.UserId = 55;

                using var scope = _factory.Services.CreateScope();
                var context =
scope.ServiceProvider.GetRequiredService<NoteRepository>();
                var context1 =
scope.ServiceProvider.GetRequiredService<ElementRepository>();

                // Act
                var response = await
client.GetAsync("/Note/GetNotes");

                // Assert
                response.StatusCode.Should().Be(HttpStatusCode.OK);
                var options = new JsonSerializerOptions
                {
                     Converters = { new DateOnlyConverter() }
                };

                var notes = await
response.Content.ReadFromJsonAsync<List<Note>>(options);
                notes.Count.Should().BeGreaterThanOrEqualTo(2);
           }

           [Fact]
           public async Task GetNote_WhenNoteExists_ReturnsNote()
           {
                // Arrange
                var client = _factory.CreateClient();
                client.DefaultRequestHeaders.Add("UserId", "55");
                CurrentUser.UserId = 55;

                using var scope = _factory.Services.CreateScope();
                var context =
scope.ServiceProvider.GetRequiredService<NoteRepository>();
                var context1 =
scope.ServiceProvider.GetRequiredService<ElementRepository>();
```

```
                Element newElem1 = new Element();
                newElem1.user_id = 55;
                newElem1.elem_type = false;
                context1.Elements.Add(newElem1);
                context1.SaveChanges();

                int newElemId = newElem1.elem_id;
                var note = new Note { elem_id = newElemId, note_name
= "Test Note 1", note_create =
DateOnly.FromDateTime(DateTime.Now), description = "This is a test
note 1." };
                context.Notes.Add(note);
                context.SaveChanges();

                // Act
                var response = await
client.GetAsync("/Note/GetNote/177");

                // Assert
                response.StatusCode.Should().Be(HttpStatusCode.OK);
                var options = new JsonSerializerOptions
                {
                     Converters = { new DateOnlyConverter() }
                };
                var returnedNote = await
response.Content.ReadFromJsonAsync<Note>(options);
                returnedNote.note_name.Should().Be("Test Note 1");
                returnedNote.description.Should().Be("This is a test
note 1.");
          }

          [Fact]
          public async Task
GetNote_WhenNoteDoesNotExist_ReturnsNotFound()
          {
                // Arrange
                var client = _factory.CreateClient();
                client.DefaultRequestHeaders.Add("UserId", "55");
                CurrentUser.UserId = 55;

                // Act
                var response = await
client.GetAsync("/Note/GetNote/1000");

                // Assert

     response.StatusCode.Should().Be(HttpStatusCode.NotFound);
          }

          [Fact]
          public async Task CreateNote_WhenValidModel_CreatesNote()
          {
                // Arrange
```

```
                var client = _factory.CreateClient();
                client.DefaultRequestHeaders.Add("UserId", "55");
                CurrentUser.UserId = 55;
                var newNote = new NoteCreateModel { note_name =
"Test Note 2", description = "This is a test note 2" };

                // Act
                var options = new JsonSerializerOptions
                {
                     Converters = { new DateOnlyConverter() }
                };
                var response = await
client.PostAsJsonAsync("/Note/CreateNote", newNote);

                // Assert
                response.StatusCode.Should().Be(HttpStatusCode.OK);

                var createdNote = await
response.Content.ReadFromJsonAsync<Note>(options);
                createdNote.note_name.Should().Be("Test Note 2");
                createdNote.description.Should().Be("This is a test
note 2");

          }

          [Fact]
          public async Task
CreateNote_WhenInvalidModel_ReturnsBadRequest()
          {
                // Arrange
                var client = _factory.CreateClient();
                client.DefaultRequestHeaders.Add("UserId", "55");
                CurrentUser.UserId = 55;
                var newNote = new NoteCreateModel();

                // Act
                var response = await
client.PostAsJsonAsync("/Note/CreateNote", newNote);

                // Assert

     response.StatusCode.Should().Be(HttpStatusCode.BadRequest);
          }
     }
}
```