МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное автономное образовательное учреждение
высшего образования
**«Южно-Уральский государственный университет
(национальный исследовательский университет)»**
Высшая школа электроники и компьютерных наук
Кафедра системного программирования

ДОПУСТИТЬ К ЗАЩИТЕ

Заведующий кафедрой, д.ф.-м.н., профессор

__________ Л.Б. Соколинский

«___»___________ 2024 г.

Разработка API модуля управления задачами корпоративной системы ООО «ЭнергоИнжиниринг»

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА
ЮУрГУ – 09.03.04.2024.308-563.ВКР

Научный руководитель,
доцент кафедры СП, к.ф.-м.н.
__________ А.Т. Латипова

Автор работы,
студент группы КЭ-403
__________ М.Р. Ларионов

Ученый секретарь
(нормоконтролер)
_____________ И.Д. Володченко
«___»___________ 2024 г.

Челябинск, 2024 г.

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное автономное образовательное учреждение
высшего образования
**«Южно-Уральский государственный университет
(национальный исследовательский университет)»**
Высшая школа электроники и компьютерных наук
Кафедра системного программирования

УТВЕРЖДАЮ
Зав. кафедрой СП
__________ Л.Б. Соколинский
29.01.2024 г.

ЗАДАНИЕ
на выполнение выпускной квалификационной работы бакалавра
студенту группы КЭ-403
Ларионову Михаилу Романовичу,
обучающемуся по направлению
09.03.04 «Программная инженерия»

1. **Тема работы** (утверждена приказом ректора от 22.04.2024 г. № 764-13/12)
Разработка API модуля управления задачами корпоративной системы ООО «ЭнергоИнжиниринг».
2. **Срок сдачи студентом законченной работы:** 03.06.2024 г.
3. **Исходные данные к работе**

3.1. Microsoft Learn. [Электронный ресурс] URL: https://learn.microsoft.com/en-us/docs (дата обращения: 04.06.2024 г.).
3.2. ASP.NET Core MVC для .NET 5. Первая часть. [Электронный ресурс] URL: https://udemy.com/course/aspnet-core-mvc-1-net-5/ (дата обращения: 04.06.2024 г.).
3.3. ASP.NET Core MVC для .NET 5. Вторая часть. [Электронный ресурс] URL: https://udemy.com/course/aspnet-core-mvc-net-5-2/ (дата обращения: 04.06.2024 г.).
3.4. C# Programming Language. [Электронный ресурс] URL: https://dotnet.microsoft.com/en-us/languages/csharp (дата обращения: 04.06.2024 г.).

4. **Перечень подлежащих разработке вопросов**

4.1. Выбрать СУБД и ORM для работы с данными в базе данных.
4.2. Выбрать фреймворк для создания API приложения.

4.3. Разработать необходимые модели и сущности, связанные с предметной областью.
4.4. Разработать логику по работе над CRUD операциями в базе данных.
4.5. Разработать логику по обработке запросов на аутентификацию и авторизацию пользователей в системе.

5. Дата выдачи задания: 29.01.2024 г.

Научный руководитель,
доцент кафедры СП, к.ф.-м.н. А.Т. Латипова

Задание принял к исполнению М.Р. Ларионов

ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ 5
1. АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ 7
2. ПРОЕКТИРОВАНИЕ 9
    2.1. Диаграмма вариантов использования 10
    2.2. Компоненты системы 11
    2.3. База данных 13
3. РЕАЛИЗАЦИЯ 15
    3.1. Инструменты разработки 15
    3.2. Создание моделей 15
    3.3. Разработка функционала задач 26
    3.4. Разработка функционала комментариев 30
    3.5. Разработка API 32
4. ТЕСТИРОВАНИЕ 38
ЗАКЛЮЧЕНИЕ 41
ЛИТЕРАТУРА 42
ПРИЛОЖЕНИЯ 44
    Приложение А. Спецификации вариантов использования 44
    Приложение Б. Класс AppDbContext и Repository 48
    Приложение В. Результаты тестирования 50

ВВЕДЕНИЕ

Актуальность

В современном мире для любого бизнеса одной из основных задач является автоматизация бизнес-процессов, то есть, цифровизация и снижение ручного труда сотрудников предприятия. Для этого на рынке существуют ERP-системы, которые выполняют данную задачу. С помощью ERP-систем можно автоматизировать многие рутинные задачи, такие как учет товаров, управление складом, финансовый учет, распределение задач сотрудникам и т.д., что позволяет сосредоточиться на более стратегических задачах для предприятия. Кроме того, ERP-системы обеспечивают централизованное хранение данных, что улучшает качество принимаемых решений и обеспечивает более точный анализ бизнес-процессов. В целом, ERP-системы являются необходимым инструментом для предприятий любого размера, которые стремятся к оптимизации своей деятельности и повышению эффективности работы.

Постановка задачи

Целью выпускной квалификационной работы является разработка API модуля управления задачами корпоративной системы ООО «Энерго-Инжиниринг». Для достижения поставленной цели необходимо решить следующие задачи:

1) провести анализ предметной области;
2) разработать архитектуру системы;
3) создать функционал системы;
4) провести тестирование разработанного функционала.

Структура и содержание работы

Работа состоит из введения, четырех глав, заключения и списка литературы. Объем работы составляет 56 страниц, объем списка литературы – 17 источников.

В первой главе описывается анализ предметной области разрабатываемого в ходе выполнения выпускной квалификационной работы прило-

жения, а также обзор аналогичных существующих на данный момент решений.

Вторая глава посвящена проектированию разрабатываемой системы. В ней приведено описание функциональных и нефункциональных требований к разрабатываемому программному продукту, диаграмма вариантов использования системы, а также описана архитектура разрабатываемого приложения.

В третьей главе приведено описание реализации разрабатываемой программы. Данная глава описывает: используемые в ходе разработки программные продукты и технологии, создание необходимых классов и сущностей, разработку конфигурации приложения для работы с базой данных, логику для механизма аутентификации и авторизации в системе, основной функционал для работы с задачами и API для взаимодействия с клиентским приложением.

В четвертой главе приведены протоколы и результаты тестирования разрабатываемой системы.

В приложении А содержится спецификация вариантов использования, разрабатываемого в ходе выполнения выпускной квалификационной работы бакалавра программного продукта.

В приложении Б представлен код класса `AppDbContext` и `Repository`.

В приложении В представлены результаты тестирования системы.

1. АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ

Предметная область проекта

Предметная область проекта связана с автоматизацией бизнес-процессов предприятия, таких как, документооборот, учет сотрудников, распределение задач между сотрудниками.

Анализ аналогичных проектов

Для успешной разработки приложения в ходе выпускной квалификационной работы необходимо провести тщательный анализ уже существующих и широко используемых на рынке аналогичных программных продуктов. Данная необходимость обусловлена тем, что так или иначе в процессе разработки продукта необходимо и важно иметь представление о том, как может выглядеть и осуществлять свою работу конечное приложение. Кроме того, подобного рода анализ может помочь выявить как преимущества, так и недостатки подобных программных продуктов, что позволит более осознанно подходить к проектированию и разработке.

SAP ERP – комплексная система планирования ресурсов предприятия (ERP) со встроенными интеллектуальными технологиями, среди которых присутствую: ИИ, машинное обучение и расширенная аналитика. Оно помогает компаниям внедрять новые бизнес-модели, оперативно управлять изменениями в бизнесе, координировать внутренние и внешние ресурсы и использовать прогнозные возможности ИИ.

Microsoft Dynamics 365 – CRM/ERPсистема, имеющая полную интеграцию с платформой и приложениями Microsoft. Содержит инструменты для управления продажами, маркетингом, сервисом и бизнес-процессами. Возможна работа с системой прямо из Outlook.

1С: ERP – решение на платформе 1С: Предприятие для построения комплексных информационных систем управления деятельностью многопрофильных предприятий с учетом лучших мировых и отечественных практик автоматизации крупного и среднего бизнеса.

Odoo – SaaScервис на основе open-source системы OpenERP. Является интегрированной платформой для управления бизнес-процессами, которая включает в себя широкий спектр модулей, таких как управление продажами, управление складом, управление производством, управление проектами, управление человеческими ресурсами. Предоставляет комплексное решение для автоматизации бизнес-процессов и управления ресурсами компании.

Oracle ERP – Программное обеспечение для управления предприятием, которое включает в себя различные модули, такие как финансы, управление человеческими ресурсами, управление производством, логистикой и другие. Предоставляет комплексный набор инструментов для автоматизации бизнес-процессов, управления ресурсами и анализа данных.

Выводы по первой главе

Проведя анализ предметной области разрабатываемого в ходе выпускной квалификационной работы бакалавра приложения и сравнительного обзора аналогичных программных продуктов, можно заключить, что указанные выше приложения имеют схожий набор функций и возможностей. Однако, из-за сложностей и затрат на приобретение и доработку этих систем, предприятие не может воспользоваться ими, так как эти процессы требуют не менее усилий и ресурсов, чем создание новой системы с нуля.

2. ПРОЕКТИРОВАНИЕ

Необходимо разработать API модуль веб-приложения, реализующий функционал системы для управления задачами, необходимый конечному заказчику (директору предприятия). На данный момент система продолжает находиться в стадии разработки, и одним из основных направлений в разработке системы является такой компонент, как задачник для сотрудников предприятия. Данное направление обусловлено требованием заказчика.

В ходе проектирования были разработаны функциональные требования, представленные ниже.

1. Создание, получение, редактирование, удаление задач.
2. Назначение и удаление ответственных за задачу.
3. Поиск задач по присущим им параметрам.
4. Добавление, редактирование, удаление, чтение комментариев к задачам.
5. Добавление, редактирование, удаление, чтение ответов на комментарии к задачам.
6. Отправка выполненных задач на проверку создателям ответственными за них пользователями.
7. Добавление подзадач.

Также в ходе проектирования были сформулированы нефункциональные требования, представленные ниже.

1. Система должна быть написана с помощью использования фреймворка ASP.NET Core WEB API [1].
2. Система должна быть написана на языке программирования C# [2].
3. В качестве ORM должна использоваться библиотека Entity Framework Core [3].
4. В качестве СУБД должна использоваться PostgreSQL [4].

5. Для создания сущностей пользователя и роли должна использоваться библиотека ASP.NET Core Identity [5].

6. Для настройки работы с JSONдолжна использоваться библиотека Newtonsoft.Json [6].

2.1. Диаграмма вариантов использования

На основе подробно сформулированных функциональных и нефункциональных требований была разработана детальная диаграмма вариантов использования. В этой диаграмме актером является клиентская часть веб-приложения (например, веб-приложение в браузере или мобильное приложение) которая была разработана другими разработчиками, работающими над созданием данной корпоративной системы. Подробное изображение диаграммы вариантов использования можно увидеть на рисунке 1.

Клиентское программное обеспечение (например, мобильное или веб-приложение в браузере) взаимодействует с разрабатываемым в ходе выполнения выпускной квалификационной работы бакалавра программным продуктом, обеспечивая работу с функционалом задач в системе. Для этого клиентское приложение отправляет REST запросы на сервер с использованием таких методов, как: GET, POST, PUT, DELETE. Это позволяет пользователю эффективно управлять данными и взаимодействовать с различными функциями системы через интерфейс веб-приложения. В результате пользователь получает актуальную информацию и может осуществлять необходимые действия для отслеживания, управления, и выполнения своих задач. Пользователь может создавать, редактировать, удалять, получать задачи, отправлять их на проверку, а также отклонять или принимать выполненные задачи от других ответственных. Также пользователь может добавлять к задачам комментарии и работать с ними. Для данных действий пользователю также необходимо быть аутентифицированным в системе.

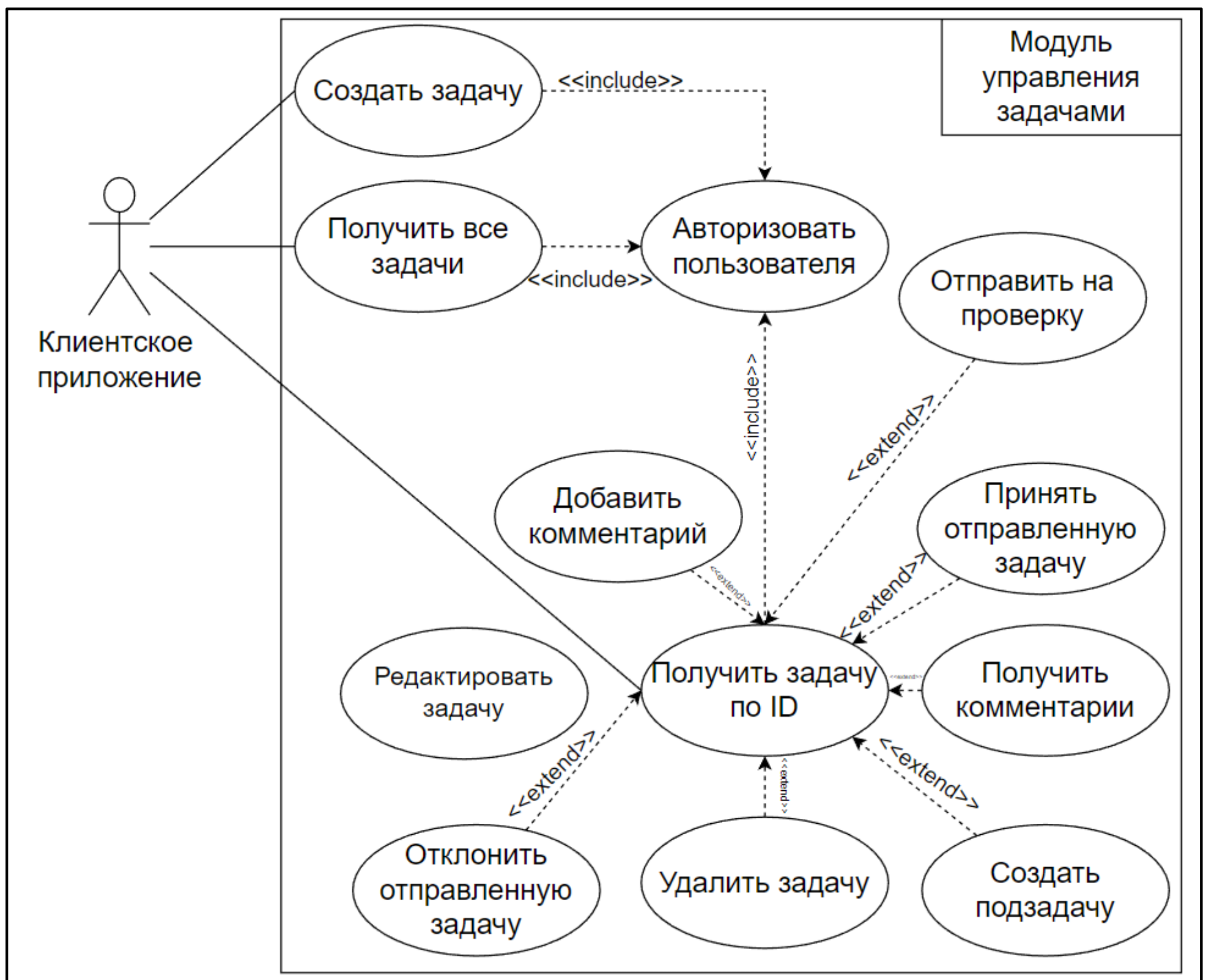

Рисунок 1 – Диаграмма вариантов использования

2.2. Компоненты системы

На рисунке 2 изображена диаграмма компонентов системы модуля управления задачами. В ходе разработки модуля API для работы с задачами были задействованы такие компоненты системы, представленные ниже.

1. Models – включает в себя классы, которые определяют сущности предметной области в приложении и в базе данных, а также DTO для отправки данных на сервер со стороны клиента и для их получения в нужном виде.

2. DataAccess – включает в себя классы сервисов-репозиториев (интерфейсы и классы) для инкапсуляции работы с базой данных и ее конфигурации из программного кода приложения.

3. Services – включает в себя классы и интерфейсы сервисов, которые содержат основную и вспомогательную бизнес-логику приложения.

4. Controllers – включает в себя классы – контроллеры, методы которых содержат логику обработки HTTP-запросов.

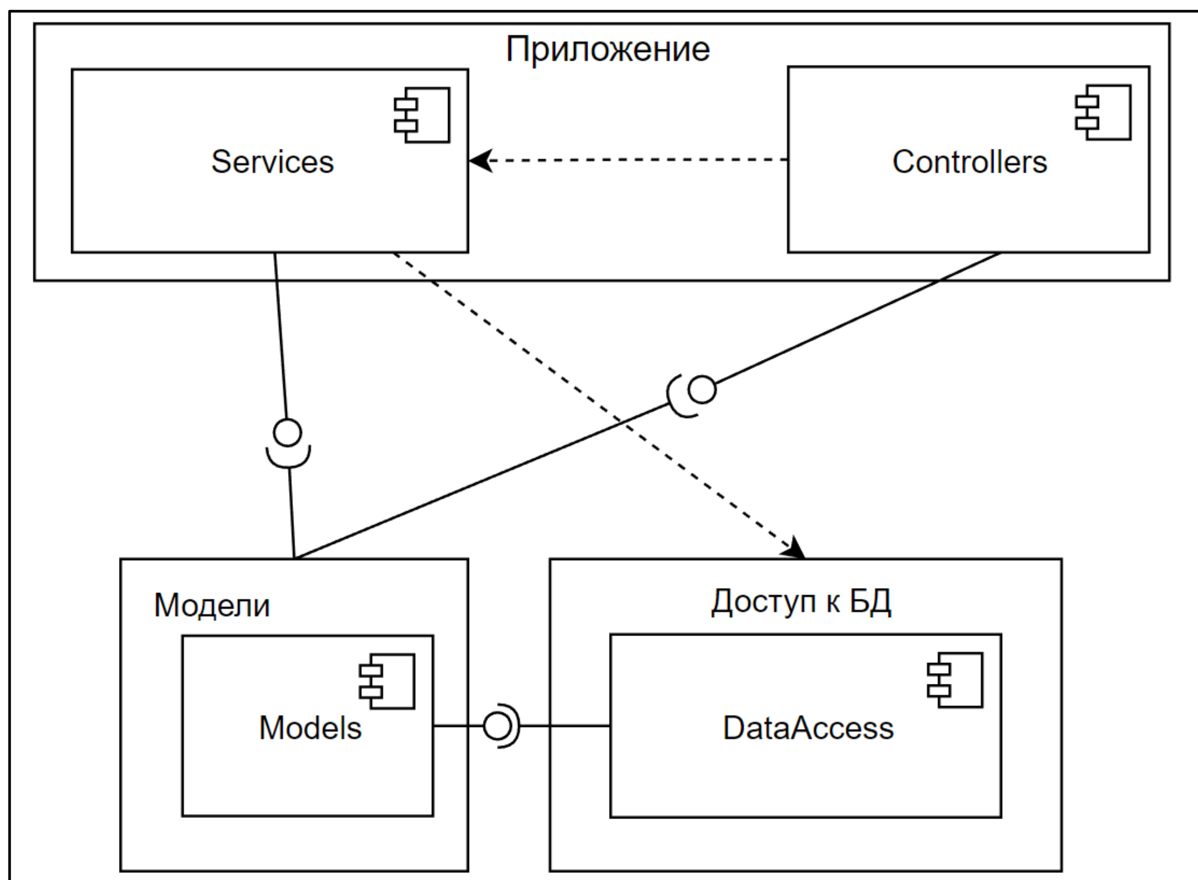

Рисунок 2 – Диаграммы компонентов системы

Объекты сервисов (DataAccess и Services) добавляются в приложения в качестве scoped-сервисов и затем посредством механизма внедрения зависимостей ASP.NET Core создаются в классах-контроллерах. Также, сервисы (Services) могут использовать логику друг друга в рамках одного компонента.

Компонент слоя доступа к базе данных DataAccess использует классы сущностей из компонента моделей Models для работы над конфигурацией базы данных, разработки таблиц по этим сущностям, а также для разработки функционала по работе сущностей и базы данных. Компонент Services использует объекты компонента Models и сервисы-репозитории из DataAccess в основных и вспомогательных алгоритмах бизнес-логики системы. Классы-контроллеры, в свою очередь, используют объекты из ком-

понента Services в методах, которые обрабатывают соответствующие REST запросы со стороны клиентского программного обеспечения. Стоит отметить, что компоненты DataAccess и Models являются отдельными слоями приложения, так как были созданы в качестве отдельных библиотек DLL, подключаемых к основному проекту, который хранит Services и Controllers и является слоем приложения. В слое приложения также располагается функционал для преобразования объектов из компонента Models к соответствующим DTO.

2.3. База данных

На рисунке 3 изображена схема базы данных. В рамках данной выпускной квалификационной работы было задействовано 7 таблиц, которые перечислены ниже.

1. `AspNetUsers` – таблица для хранения данных о пользователях.
2. `WorkTasks` – таблица для хранения данных о задачах.
3. `WorkTasksLabels` – таблица для хранения данных о метках задач.
4. `ResponsibleUsers` – промежуточная таблица между пользователем и задачей, используется для хранения данных о сущности пользователя, который является ответственным за задачу.
5. `WorkTaskExecutionHistory` – таблица для хранения данных об истории выполнения задачи ответственными.
6. `CommentsSections` – промежуточная таблица для связи между задачей и комментарием (комментарий не нужно привязывать к задаче напрямую).
7. `Comments` – таблица для хранения данных о комментариях.

Таблица `AspNetUsers` имеет связь с таблицей `WorkTasks` напрямую через отношение «один-ко-многим» - данное отношение характеризует пользователя как создателя задачи. Также пользователь связан с задачей через промежуточную таблицу `ResponsibleUsers`: `AspNetUsers` имеет отношение «один-ко-многим» к `ResponsibleUsers`, а также `WorkTask` свя-

зана с `ResponsibleUsers` через данный вид связи. Таким образом, пользователь может быть создателем разных задач, а также быть ответственным за разные задачи. К сущности ответственного за задачу привязана сущность `WorkTaskExecutionHistory`, которая хранит записи о конкретном объекте ответственного за конкретную задачу.

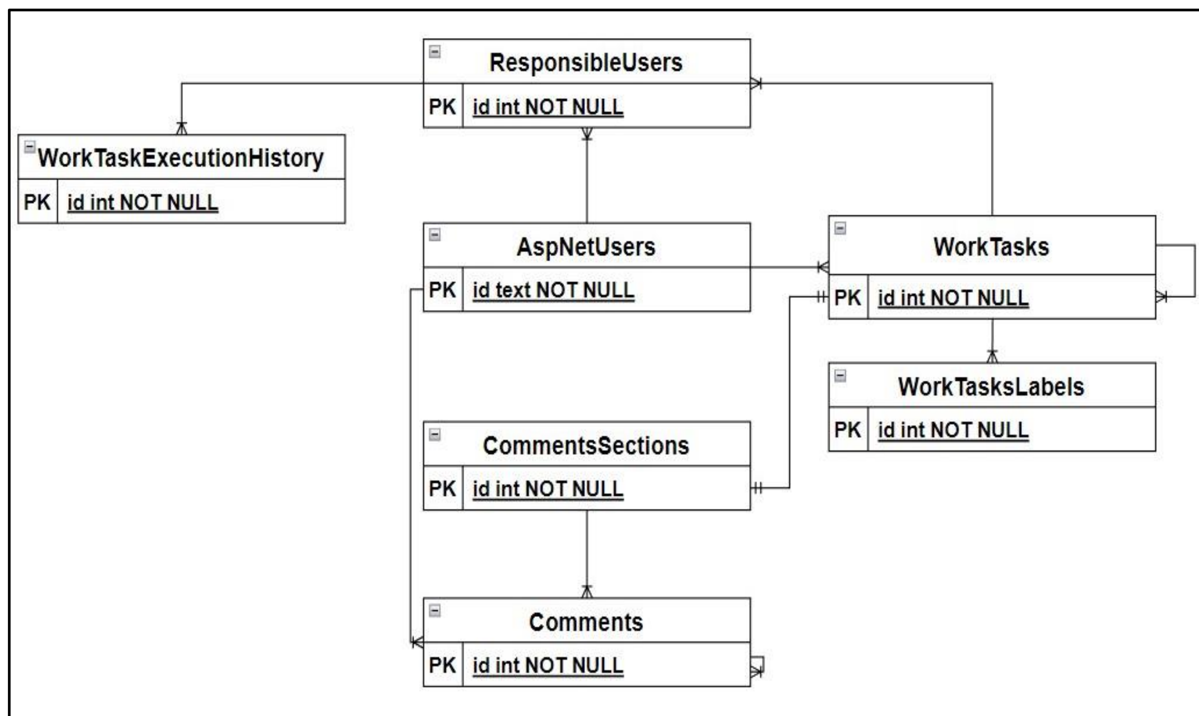

Рисунок 3 – Схема базы данных

Выводы по второй главе

В данной главе были сформулированы основные функциональные и нефункциональные требования к системе, разрабатываемой в ходе выполнения выпускной квалификационной работы бакалавра. Также были выделены основные варианты использования API модуля управления задачами корпоративной системы ООО «ЭнергоИнжиниринг», разработана и подробно описана его архитектура, а также была спроектирована база данных со всеми необходимыми сущностями и связями. Разработанные варианты использования, архитектура приложения и схема базы данных также были представлены на соответствующих диаграммах.

3. РЕАЛИЗАЦИЯ

3.1. Инструменты разработки

Основным инструментом для разработки приложения является текстовый редактор исходного кода Visual Studio Code [8] (VSCode). Основными достоинствами данного инструмента являются: его доступность – VSCode распространяется бесплатно и разрабатывается как программное обеспечение с открытым исходным кодом (open-source), наличие большого количества бесплатных расширений, упрощающих работу со многими существующими технологиями для разработки программного обеспечения (в их числе так же и язык программирования C#) и его легковесность.

Для обеспечения возможности просмотра и, при крайней необходимости, редактирования данных в базе данных используются такие инструменты как pgAdmin [9] и DBeaver [12]. Данные программные продукты являются широко распространенными приложениями, предназначенными для работы с различными СУБД (в случае DBeaver), в том числе и с PostgreSQL.

Компиляция и запуск приложения, работа с git, добавление библиотек в проект, создание новых проектов, создание миграций для БД и т.д. осуществляются посредством ввода соответствующих команд в командной строке операционной системы Windows 10, а также в терминале Unix-подобных операционных систем, таких как Linux и MacOS.

3.2. Создание моделей

Сущностью задачи в системе является класс `WorkTask`. Он определяет такие данные, как название, описание, дата начала задачи, крайний срок выполнения задачи, ссылка на создателя задачи, ответственные за задачу пользователи и т.д. Класс `WorkTask` и его свойства приведены в листинге 1.

Листинг 1 – Класс `WorkTask`

```
public class WorkTask
{
    public int Id { get; set; }
    public string Name { get; set; } = null!;
```

```
    public string? Description { get; set; }
    public DateTimeOffset Deadline { get; set; }
    public bool IsUrgent { get; set; }
    public string? CreatedByUserId { get; set; }
    public ApplicationUser? CreatedBy { get; set; }
    public List<WorkTaskResponsibleUser> ResponsibleUsers { get; set; }
    public List<WorkTaskLabel> WorkTaskLabels { get; set; }
    public WorkTaskStatus TaskStatus { get; set; }
    public DateTimeOffset DateComplete { get; set; }
    public WorkTask? ParentWorkTaskId { get; set; }
    public List<WorkTask> SubWorkTasks { get; set; }
    public DateTimeOffset DateAdded { get; set; }
    public CommentsSection? CommentsSection { get; set; }
}
```

В классе `WorkTask` определены свойства, представленные ниже.

1. `Id` – свойство типа `int`, используется в качестве первичного ключа сущности в базе данных.

2. `Name` – свойство типа `string`, представляет собой название задачи.

3. `Description` – свойство типа `string`, представляет собой описание задачи, является необязательным свойством (может не иметь значения).

4. `Deadline` – свойство типа `DateTimeOffset`, представляет собой дедлайн задачи, крайний срок ее выполнения.

5. `IsUrgent` – свойство типа `bool`, значение которого устанавливает, является ли задача срочной или нет.

6. `CreatedByUserId` – свойство типа `string`, используется в качестве внешнего ключа на сущность пользователя в базе данных.

7. `CreatedBy` – объект класса `ApplicationUser`, представляет собой навигационное свойство для обращения к объекту пользователя, на который объект задачи ссылается с помощью внешнего ключа.

8. `DateComplete` – свойство типа `DateTimeOffset`, представляет собой дату выполнения задачи.

9. `ParentWorkTaskId` – свойство типа `int`, представляет собой внешний ключ, с помощью которого объект подзадачи связан с основной задачей, которой он принадлежит. Является опциональным, так как задача не обязательно является подзадачей.

10. SubWorkTasks – навигационное свойство типа List<WorkTask>, представляет собой коллекцию ссылок на объекты подзадач.

11. DateAdded – свойство типа DateTimeOffset, представляет собой дату создания задачи.

12. ResponsibleUsers – навигационное свойство типа List<WorkTaskResponsibleUser>, представляет собой коллекцию ссылок на объекты ответственных за задачу пользователей.

13. WorkTaskLabels – навигационное свойство типа List<WorkTaskLabel>, представляет собой коллекцию ссылок на объекты меток задачи.

14. TaskStatus – свойство, типом которого является перечисление WorkTaskStatus, представляет собой статус задачи.

15. CommentsSection – объект класса CommentsSection, представляет собой ссылку на объект секции комментариев задачи.

Задача создается пользователем системы. Сущность пользователя системы определяется классом ApplicationUser, который наследует класс IdentityUser, являющийся частью библиотеки Identity. ApplicationUser и его свойства приведены в листинге 2.

Листинг 2 – Класс ApplicationUser

```
public class ApplicationUser : IdentityUser
{
    public string FirstName { get; set; }
    public string LastName { get; set }
    public string MiddleName { get; set; }
    public List<WorkTask> WorkTasks { get; set; }
    public List<WorkTaskResponsibleUser> TaskResponsibilities { get; set; }
    public List<Comment> Comments { get; set; }
}
```

Свойства класса ApplicationUser представлены ниже.

1. FirstName – свойство типа string, представляет собой имя пользователя.

2. LastName – свойство типа string, представляет собой фамилию пользователя.

3. MiddleName – свойство типа string, представляет собой отчество пользователя.

4. WorkTasks – навигационной свойство, представляющее собой коллекцию ссылок на объекты задач, создателем которых пользователь является.

5. TaskResponsibilities – навигационное свойство, представляющее собой коллекцию ссылок на объекты класса WorkTaskResponsibleUser, с помощью которого пользователь связан с задачей в качестве ответственного.

6. Comments – навигационное свойство, представляющее собой коллекцию ссылок на объекты класса Comment, являющихся комментариями, которые были созданы пользователем.

Пользователь может быть не только создателем задачи, но и ответственным за ее выполнение. Для этого в системе используется класс WorkTaskResponsibleUser. Данный класс используется в качестве промежуточной сущности между задачей и пользователем системы (объект класса ApplicationUser), который отвечает за задачу. WorkTaskResponsibleUser приведен в листинге 3. Создатель и ответственные пользователи добавляются для задачи во время ее создания. Ответственных также можно редактировать.

Листинг 3 – Класс WorkTaskResponsibleUser

```
public class WorkTaskResponsibleUser : ResponsibleUser
{
    public int WorkTaskId { get; set; }
    public WorkTask? WorkTask { get; set; }
    public WorkTaskStatus ImplementationStatus { get; set; }
    public DateTime DateAdded { get; set; }
    public List<ExecutionHistory> History { get; set; }
}
```

Свойства класса WorkTaskResponsibleUser представлены ниже.

1. WorkTaskId – свойство типа int, представляет собой внешний ключ для связи с задачей, которой принадлежит объект ответственного пользователя.

2. WorkTask – навигационное свойство типа WorkTask, представляет собой объект задачи, на которую объект ответственного ссылается с помощью внешнего ключа, используется для работы с данными задачи.

3. ImplementationStatus – свойство, типом которого является перечисление WorkTaskStatus, представляет собой статус выполнения задачи конкретным относительно конкретного ответственного.

4. DateAdded – свойство типа DateTime, представляет собой дату создания объекта.

5. History – навигационное свойство, представляет собой массив ссылок на объекты класса ExecutionHistory, которые, в свою очередь, хранят данные об истории выполнения задачи ответственным.

Код перечисления WorkTaskStatus представлен в листинге 4.

Листинг 4 – Перечисление WorkTaskStatus

```
public enum WorkTaskStatus
{
    Implementation,
    UnderConsideration,
    Completed
}
```

Элементы перечисления WorkTaskStatus описаны ниже.

1. Implementation – статус задачи «В работе», задача получает его при создании, ответственный – при добавлении к задаче.

2. UnderConsideration – статус задачи «На проверке», ответственный получает его при отправке задачи на проверку создателю, задача – когда все ее ответственные получат данный статус.

3. Completed – статус задачи «Завершена», ответственный получает его, когда отправленная им задача была принята создателем, задача – когда все ее ответственные получат данный статус.

К задаче могут добавляться комментарии. Для связи задачи и ее комментариев используется класс CommentsSection, который приведен в листинге 5.

Листинг 5 – Класс `CommentsSection`

```
public class CommentsSection
{
    public long Id { get; set; }
    public int WorkTaskId { get; set; }
    public WorkTask? WorkTask { get; set; }
    public List<Comment> Comments { get; set; }
}
```

Свойства класса `CommentsSection` представлены ниже.

1. `Id` – свойство типа int, представляет собой первичный ключ сущности в базе данных.

2. `WorkTaskId` – свойство типа int, представляет собой внешний ключ на сущность `WorkTask`.

3. `WorkTask` – навигационное свойство типа `WorkTask`, представляет собой объект задачи, на который ссылается секция комментариев с помощью соответствующего внешнего ключа.

4. `Comments` – навигационное свойство, представляющее собой коллекцию ссылок на объекты класса `Comment`, которые являются комментариями задачи.

В листинге 6 представлен класс `Comment`, описывающий сущность комментария.

Листинг 6 – Класс `Comment`

```
public class Comment : BaseComment
{
    public long? CommentsSectionId { get; set; }
    public CommentsSection? CommentsSection { get; set; }
    public string? UserId { get; set; }
    public ApplicationUser? User { get; set; }
    public int? ParentId { get; set; }
    public Comment? Parent { get; set; }
    public List<Comment> Children { get; set; }
}
```

Свойства класса `Comment` представлены ниже.

1. `CommentSectionId` – свойство типа `long`, представляет собой внешний ключ на секцию комментариев в базе данных.

2. `CommentSection` – навигационное свойство типа `CommentSection`, используется для получения данных о секции комментариев, на которую комментарий ссылается с помощью внешнего ключа.

3. `UserId` – свойство типа string, представляет собой внешний ключ на пользователя, являющегося автором комментария.

4. `User` – навигационное свойство типа `ApplicationUser`, которое используется для получение данных об авторе комментария, на который комментарий ссылается с помощью внешнего ключа.

5. `ParentId` – свойство типа `int`, представляет собой внешний ключ для связи с родительским комментарием.

6. `Parent` – навигационное свойство типа `Comment`, представляет собой объект родительского комментария.

7. `Children` – навигационное свойство, представляющее собой коллекцию объектов `Comment`, которые являются дочерними объектами комментария (ответами на комментарий).

Класс `Comment` наследует класс `BaseComment`, который приведен в листинге 7.

Листинг 7 – Класс `BaseComment`

```
public class BaseComment
{
    public int Id { get; set; }
    public string Text { get; set; }
    public DateTimeOffset DateAdded { get; set; }
}
```

Свойства класса `BaseComment` представлены ниже.

1. `Id` – свойство типа `int`, представляет собой первичный ключ сущности в базе данных.

2. `Text` – свойство типа `string`, представляет собой текст комментария.

3. `DateAdded` – свойство типа `DateTimeOffset`, представляет собой дату добавление комментария.

Сущность метки задачи описывает класс `WorkTaskLabel`, код которого приведен в листинге 8.

Листинг 8 – Класс `WorkTaskLabel`

```
public class WorkTaskLabel
{
    public long Id { get; set; }
    public string? Name { get; set; }
    public int WorkTaskId { get; set; }
    public WorkTask? WorkTask { get; set; }
    public DateTimeOffset DateAdded { get; set; }
}
```

Свойства класса `WorkTaskLabel` представлены ниже.

1. `Id` – свойство типа `long`, представляет собой первичный ключ сущности в базе данных.
2. `Name` – свойство типа `string`, представляет собой название метки.
3. `WorkTaskId` – свойство типа `int`, представляющее собой внешний ключ для связи с сущностью задачи в базе данных.
4. `WorkTask` – навигационное свойство типа `WorkTask`, используемое для получения данных объекта задачи, на который ссылается метка с помощью соответствующего внешнего ключа.
5. `DateAdded` – свойство типа `DateTimeOffset`, представляет собой дату добавление метки.

В листинге 9 представлен класс `ExecutionHistory`, сущность которого нужна для того, чтобы хранить данные о ходе выполнения задач ответственными.

Листинг 9 – Класс `ExecutionHistory`

```
public class ExecutionHistory
{
    public long Id { get; set; }
    public string? Text { get; set; }
    public ProgressStatus Status { get; set; }
    public DateTimeOffset DateAdded { get; set; }
    public int ResponsibleId { get; set; }
    public WorkTaskResponsibleUser? Responsible { get; set; }

    [NotMapped]
    public string? SenderId
    {
        get
        {
            return Status switch
```

```
            {
                ProgressStatus.Sent => Responsible?.UserId,
                ProgressStatus.Accepted or ProgressStatus.Rejected => Re-
sponsible?.WorkTask?.CreatedByUserId,
                _ => null,
            };
        }
    }

    [NotMapped]
    public string? SenderFullName
    {
        get
        {
            return Status switch
            {
                ProgressStatus.Sent => $"{Responsi-
ble?.ApplicationUser?.LastName} {Responsible?.ApplicationUser?.FirstName}
{Responsible?.ApplicationUser.MiddleName}",
                ProgressStatus.Accepted or ProgressStatus.Rejected =>
$"{Responsible?.WorkTask?.CreatedBy?.LastName} {Responsi-
ble?.WorkTask?.CreatedBy?.FirstName} {Responsi-
ble?.WorkTask?.CreatedBy?.MiddleName}",
                _ => null,
            };
        }
    }
}
```

Свойства класса `ExecutionHistory` представлены ниже.

1. `Id` – свойство типа `long`, представляет собой первичный ключ сущности в базе данных.

2. `Text` – свойство типа `string`, представляет собой текст, который ответственный за задачу вводит при отправке задачи на проверку, либо причину отклонения отправленной на проверку задачи.

3. `Status` – свойство, типом которого является перечисление `ProgressStatus`, представляет собой вид действия.

4. `DateAdded` – свойство типа `DateTimeOffset`, представляет собой дату добавления записи.

5. `ResponsibleId` – свойство типа `int`, представляет собой внешний ключ на объект ответственного в базе данных.

6. `Responsible` – навигационное свойство типа `WorkTaskResponsibleUser` для работы со связанным объектом ответственного.

7. SenderId – свойство типа string, которое хранит в себе идентификатор пользователя, чье действие было записано в историю, не хранится в базе данных.

8. SenderFullName – свойство типа string, которое хранит в себе ФИО пользователя, чье действие было записано в историю, не хранится базе данных.

Работа с базой данных

Для создания базы данных и конфигурации сущностей используется класс AppDbContext. В нем определены публичные свойства типа DbSet для создания соответствующих таблиц в базе данных. Данный класс также переопределяет метод OnModelCreating, в котором были разработаны необходимые связи между сущностями. Он приведен в листинге 1 приложения Б.

Чтобы воспроизводить основные операции над объектами в базе данных, была определена логика в соответствии с паттерном «репозиторий» [13]. В листинге 10 приведен обобщенный (generic) интерфейс IRepository<T>, определяющий основные CRUD-операции, и методы которого будут реализовываться уже конкретными классами-репозиториями для каждой необходимой сущности.

Листинг 10 – Интерфейс IRepository

```
public interface IRepository<T> where T : class
{
      Task<IEnumerable<T>> GetAllAsync(Expression<Func<T, bool>>? filter
=null, Func<IQueryable<T>,IOrderedQueryable<T>>? orderBy = null,
      string? includeProp = null,bool isTraking = true);
      Task<T?> FirstOrDefaultAsync(Expression<Func<T, bool>>? filter =
null,string? includeProp = null,bool isTraking = true);
        void Add(T entity);
        public void AddRange(IEnumerable<T> collection);
        void Remove(T entity);
        void RemoveRange(IEnumerable<T> collection);
        void Save();
        void UpdateRange(IEnumerable<T> collection);
}
```

Методы, определенные в интерфейсе IRepository, реализуются в классе Repository. Код класса приведен в листинге 2 приложения Б.

Класс `Repository<T>` используется в качестве обобщенного абстрактного базового класса для классов-репозиториев конкретных моделей. Примером одного из классов-наследников `Repository<T>` является класс `WorkTaskRepository`, который приведен в листинге 11.

Листинг 11 – Класс `WorkTaskRepository`

```
public class WorkTaskRepository(AppDbContext db) : Repository<WorkTask>(db)
{ }
```

В свою очередь, объект этого класса-репозитория будет внедрен в качестве зависимости в соответствующий сервис.

Разработка механизма аутентификации

Для аутентификации пользователей в системе используется аутентификация с помощью JWT токенов. [14] Для того, чтобы создать необходимый токен для пользователя используется сервис `ITokenService`, представленный в листинге 12. Он определяет метод `CreateToken`, реализация которого должна содержать бизнес-логику по созданию необходимых токенов для успешного входа пользователя в систему.

Листинг 12 – Интерфейс `ITokenService`

```
public interface ITokenService
{
    string CreateToken(ApplicationUser user);
}
```

Метод `CreateToken` интерфейса `ITokenService` реализован в классе `TokenService`, который представлен в листинге 13.

Листинг 13 – Класс `TokenService`

```
public class TokenService : ITokenService {
   public string CreateToken(ApplicationUser user) {
        var token = user.CreateClaims().CreateJwtToken(_configuration);
        var tokenHandler = new JwtSecurityTokenHandler();
        return tokenHandler.WriteToken(token);
    }
}
```

В реализации данного метода токен создается с помощью метода расширения для объекта `ApplicationUser` `CreateClaims` и метода расширения для коллекции объектов `Claim` `CreateJwtToken`. Далее, с помощью

объекта `JwtSecurityTokenHandler`, который является частью пространства имен `System.IdentityModel.Tokens.Jwt` [16] он проходит сериализацию в необходимый вид.

3.3. Разработка функционала задач

После создания необходимых моделей и сущностей, а также логики по работе с БД и аутентификацией пользователей можно приступить к тому, чтобы определить основной функционал данной работы – бизнес-логика по управлению задачами в системе. Операции по работе над задачами определены в интерфейсе `IWorkTaskService`, который приведен в листинге 14.

Листинг 14 – Интерфейс `IWorkTaskService`

```
public interface IWorkTaskService
{
    Task<IEnumerable<WorkTask>> GetAllAsync(
        Expression<Func<WorkTask, bool>>? filter = null,
        string? includeProp = null);
    Task<WorkTask> GetByIdAsync(int id, string? includeProp = null);
    Task CreateAsync(WorkTask workTask, ApplicationUser creator);
    Task EditAsync(
        WorkTask workTask,
        ApplicationUser user,
        IEnumerable<WorkTaskResponsibleUser> responsiblesCopy,
        IEnumerable<WorkTaskLabel> labelsCopy);
    void DeleteWorkTask(WorkTask workTask);
    Task CreateSubtask(WorkTask workTask, WorkTask subtask, ApplicationUser
user);
    Task DoTask(WorkTask workTask, WorkTaskResponsibleUser responsibleUser,
ExecutionHistory history);
    Task DismissTask(WorkTask workTask, WorkTaskResponsibleUser responsi-
bleUser, ExecutionHistory history);
    Task AcceptTask(WorkTask workTask, WorkTaskResponsibleUser responsi-
bleUser, ExecutionHistory history);
    Task EditSubtask(WorkTask subWorkTask,
        ApplicationUser user,
        IEnumerable<WorkTaskResponsibleUser> responsiblesCopy);
      }
```

Данный сервис использует логику, определенную в `IRepository`, а также определяет уже другой необходимый функционал. Помимо получения, редактирования, удаления и создания имеются также методы для работы над жизненным циклом задачи. Описание данных методов представлено ниже.

1. `CreateSubtask` – создание подзадач для основной задачи.

2. `DoTask` – отправка ответственным задачи в качестве выполненной на проверку создателю.

3. `DismissTask` – отклонение создателем выполненной задачи и возврат на доработку ответственному.

4. `AcceptTask` – создатель задачи принимает выполненную и отправленную ответственным задачу, тем самым делая её завершенной.

Методы, определенные в интерфейсе `IWorkTaskService`, реализованы в классе `WorkTaskService`, представленном в листинге 15.

Листинг 15 – Класс `WorkTaskService`

```
public class WorkTaskService(
        WorkTaskRepository workTaskRepo,
        TaskResponsibleUserRepository responsiblesRepo,
        WorkTaskLabelRepository labelRepo,
        ExecutionHistoryRepository historyRepo,
    ) : IWorkTaskService
{
    private readonly WorkTaskRepository _workTaskRepo = workTaskRepo;
    private readonly TaskResponsibleUserRepository _responsiblesRepo = re-
sponsiblesRepo;
    private readonly WorkTaskLabelRepository _labelRepo = labelRepo;
    private readonly ExecutionHistoryRepository _historyRepo = historyRepo;
}
```

Класс `WorkTaskService` использует в своем функционале объекты классов-репозиториев, которые создаются в нем в качестве scoped-сервисов с помощью механизма внедрения зависимостей [15].

В листинге 16 представлена реализация метода создания задачи.

Листинг 16 – Создание задачи

```
public async Task CreateAsync(WorkTask workTask, ApplicationUser creator)
{
    workTask.CreatedBy = creator;
    workTask.TaskStatus = WorkTaskStatus.Implementation;
    workTask.DateAdded = TimeProvider.System.GetUtcNow();
    _workTaskRepo.Add(workTask);
    _workTaskRepo.Save();
}
```

В листинге 17 представлена реализация метода редактирования задачи.

Листинг 17 – Редактирование задачи

```
public async Task EditAsync(WorkTask editWorkTask,
    ApplicationUser user, IEnumerable<WorkTaskResponsibleUser> responsi-
blesCopy, IEnumerable<WorkTaskLabel> labelsCopy)
{
    var existingLabels = labelsCopy.ToList();
    var removedLabels = existingLabels.ExceptBy(
        editWorkTask.WorkTaskLabels.Select(l => l.Name), l => l.Name
    ).ToList();

    if (removedLabels.Count != 0)
        _labelRepo.RemoveRange(removedLabels);

    var existingResponsibles = responsiblesCopy.ToList();
    var addedResponsibles = editWorkTask.ResponsibleUsers.ExceptBy(
        existingResponsibles.Select(r => r.UserId), r => r.UserId
    ).ToList();

    var removedResponsibles = existingResponsibles.ExceptBy(
        editWorkTask.ResponsibleUsers.Select(r => r.UserId), r => r.UserId
    ).ToList();

    if (removedResponsibles.Count != 0)
    {
        foreach (var r in removedResponsibles)
            _responsiblesRepo.RemoveRange(removedResponsibles);
    }

    if (editWorkTask.TaskStatus != WorkTaskStatus.Implementation && adde-
dResponsibles.Count != 0)
        editWorkTask.TaskStatus--;

    _workTaskRepo.Update(editWorkTask);
    _workTaskRepo.Save();
}
```

В листинге 18 представлена реализация метода получения задачи по `Id`.

Листинг 18 – Получение задачи по `Id`

```
public async Task<WorkTask> GetByIdAsync(int id, string? includeProp =
null) => await _workTaskRepo.FirstOrDefaultAsync(u => u.Id == id, includ-
eProp);
```

В листинге 19 представлена реализация метода получения всех задач.

Листинг 19 – Получение всех задач

```
public async Task<IEnumerable<WorkTask>> GetAllAsync
(Expression<Func<WorkTask, bool>>? filter = null, string? includeProp =
null) =>  await _workTaskRepo.GetAllAsync(filter: filter, includeProp: in-
cludeProp);
```

В листинге 20 представлена реализация метода удаления задачи.

Листинг 30 – Метод `DoTask` для отправки задачи на проверку

```
public async Task<ActionResult<WorkTask>> DoWorkTask(int id, [FromBody] Ex-
ecutionHistoryCreateDto request)
{
        var workTask = await _workTaskService.GetByIdAsync(id, "Creat-
edBy,ResponsibleUsers.ApplicationUser,SubWorkTasks");

        if (workTask == null)
            return NotFound(new ResponseMessage() { Message = "Задача не
найдена." });

        var userId = User.FindFirstValue(ClaimTypes.NameIdentifier);
        var responsibleUser = workTask.ResponsibleUsers.FirstOrDefault(u =>
u.ApplicationUser!.Id == userId);

        if (responsibleUser == null)
            return StatusCode(403, new ResponseMessage() { Message = "Вы не
являетесь ответственным за данную задачу." });

        if (workTask.TaskStatus != WorkTaskStatus.Implementation
            || responsibleUser.ImplementationStatus !=
WorkTaskStatus.Implementation)
            return BadRequest(new ResponseMessage() { Message =
"Некорректный статус задачи." });
        var history = _mapper.Map<ExecutionHistory>(request);
        await _workTaskService.DoTask(workTask, responsibleUser, history);

        var response = _mapper.Map<ExecutionHistoryReadDto>(history);

        return Ok(response);
}
```

В листинге 31 представлен код метода `Accept` для утверждения задачи.

Листинг 31 – Метод `Accept` для утверждения задачи

```
public async Task<ActionResult<WorkTaskResponsibleUser>> Accept(int
workTaskId, int respId, [FromBody] ExecutionHistoryCreateDto request)
{
        var workTask = await _workTaskService.GetByIdAsync(workTaskId, "Re-
sponsibleUsers,SubWorkTasks");
        if (workTask == null) return NotFound(new ResponseMessage() { Mes-
sage = "Задача не найдена" });

        var userId = User.FindFirstValue(ClaimTypes.NameIdentifier);
        var workTaskCreator = workTask.CreatedBy;

        if (userId != workTaskCreator!.Id)
            return StatusCode(403, new ResponseMessage() { Message = "Вы не
являетесь создателем этой задачи." });

        var responsibleUser = await
_respUserService.GetResponsibleUserById<WorkTaskResponsibleUser>(respId,
"ApplicationUser");
        if (responsibleUser is null) return NotFound(new ResponseMessage {
Message = "Ответственный за задачу не найден" });
```

```
        if (responsibleUser.ImplementationStatus != 
WorkTaskStatus.UnderConsideration)
            return BadRequest(new ResponseMessage() { Message = 
"Некорректный статус задачи." });

        var history = _mapper.Map<ExecutionHistory>(request);
        await _workTaskService.AcceptTask(workTask, responsibleUser, histo-
ry);

        var response = _mapper.Map<ExecutionHistoryReadDto>(history);
        return Ok(response);
}
```

В листинге 32 представлен код метода `Dismiss` для отклонения задачи.

Листинг 32 – Метод `Dismiss` для отклонения задачи

```
public async Task<ActionResult<WorkTaskResponsibleUser>> Dismiss(int 
workTaskId, int respId, [FromBody] ExecutionHistoryCreateDto request) {
    var workTask = await _workTaskService.GetByIdAsync(workTaskId, "Sub-
WorkTasks");
    if (workTask == null)
        return NotFound(new ResponseMessage() { Message = "Задача не найде-
на." });

    var userId = User.FindFirstValue(ClaimTypes.NameIdentifier);
    var workTaskCreator = workTask.CreatedBy
    if (userId != workTaskCreator!.Id)
        return StatusCode(403, new ResponseMessage() { Message = "Вы не яв-
ляетесь создателем данной задачи." });

    var responsibleUser = await 
_respUserService.GetResponsibleUserById<WorkTaskResponsibleUser>(respId, 
"ApplicationUser");
    if (responsibleUser is null)
        return NotFound(new ResponseMessage { Message = "Ответственный за 
задачу не найден" });

    if (responsibleUser.ImplementationStatus != 
WorkTaskStatus.UnderConsideration)
        return BadRequest(new ResponseMessage() { Message = "Некорректный 
статус задачи." });

    var history = _mapper.Map<ExecutionHistory>(request);
    await _workTaskService.DismissTask(workTask, responsibleUser, history);
    var response = _mapper.Map<ExecutionHistoryReadDto>(history);
    return Ok(response);
}
```

Как можно увидеть, в листингах 29–32 методы контроллеров используют в качестве объекта передачи данных такие классы, как, например, `ExecutionHistoryCreateDto` или `ExecutionHistoryReadDto` (листинги 33 и 34). Данные классы нужны, чтобы получать или передавать клиенту данные, в том виде, в котором это необходимо для корректной работы. Для

работы с DTO [17], а именно для преобразования объектов сущностей из базы в DTO (маппинга) используется библиотека AutoMapper [11].

Листинг 33 – Класс `ExecutionHistoryCreateDto`

```
public class ExecutionHistoryCreateDto {
    public string? Text { get; set; }
}
```

Листинг 34 – Класс `ExecutionHistoryReadDto`

```
public class ExecutionHistoryReadDto {
    public long Id { get; set; }
    public string? Text { get; set; }
    public string Status { get; set; };
    public DateTimeOffset DateAdded { get; set; }
    public string? SenderId { get; set; }
    public string? SenderFullName { get; set; }
}
```

Конфигурация маппинга приведена в классе `AutoMapperProfile`, в листинге 35.

Листинг 35 – Класс `AutoMapperProfile`

```
class AutoMapperProfile : Profile {
    public AutoMapperProfile() {
        CreateMap<ApplicationUser, ApplicationUserInfo>();
        CreateMap<WorkTask, WorkTaskGeneralInfo>()
            .ForMember(u => u.TaskStatus, x => x.MapFrom(s =>
s.TaskStatus.ToString()));
        CreateMap<WorkTask, WorkTaskDetails>()
            .ForMember(u => u.TaskStatus, x => x.MapFrom(s =>
s.TaskStatus.ToString()));
            CreateMap<WorkTaskBody, WorkTask>()
            .BeforeMap((s, d) => d.TaskStatus =
WorkTaskStatus.Implementation)
            .BeforeMap((s, d) => d.IsActual = true)
            .BeforeMap((s, d) => d.IsClosed = false)
            .ForMember(src => src.ResponsibleUsers, opt =>
            {
                opt.MapFrom((source, target, member, ctx) => {
                    return ctx.Mapper.ResolveCollection(
                        source.ResponsibleUsers,
                        target.ResponsibleUsers,
                        (targetCollection, sourceObject) => targetCollec-
tion.SingleOrDefault(t => t.UserId == sourceObject.UserId)!
                    );
                });
            })
            .ForMember(src => src.WorkTaskLabels, opt =>
            {
                opt.MapFrom((source, target, member, ctx) =>
                {
                    return ctx.Mapper.ResolveCollection(
                        source.WorkTaskLabels,
                        target.WorkTaskLabels,
```

```
                    (targetCollection, sourceObject) => targetCollec-
tion.SingleOrDefault(t => t.GetHashCode() == sourceObject.GetHashCode())!
                    );
                });
            });
        CreateMap<ExecutionHistory, ExecutionHistoryReadDto>()
            .ForMember(dest => dest.Status, opt => opt.MapFrom(src =>
src.Status.ToString()));
        CreateMap<ExecutionHistoryCreateDto, ExecutionHistory>();
    }
}
```

Для корректного изменения списка ответственных и меток в задаче был разработан соответствующий метод расширения, который приведен в классе `AutoMapperExtensions` в листинге 36.

Листинг 36 – Класс `AutoMapperExtensions`

```
public static class AutoMapperExtensions {
    public static ICollection<TTargetType> ResolveCollection<TSourceType,
TTargetType>(this IRuntimeMapper mapper,
        ICollection<TSourceType> sourceCollection,
        ICollection<TTargetType> targetCollection,
        Func<ICollection<TTargetType>, TSourceType, TTargetType> getMap-
pingTargetFromTargetCollectionOrNull) {
        var existing = targetCollection.ToList();
        targetCollection.Clear();
        return ResolveCollection(mapper, sourceCollection, s => getMapping-
TargetFromTargetCollectionOrNull(existing, s), t => t);
    }
    private static ICollection<TTargetType> ResolveCollection<TSourceType,
TTargetType>(
        IRuntimeMapper mapper,
        ICollection<TSourceType> sourceCollection,
        Func<TSourceType, TTargetType> getMappingTargetFromTargetCollec-
tionOrNull,
        Func<IList<TTargetType>, ICollection<TTargetType>> updateTargetCol-
lection) {
        var updatedTargetObjects = new List<TTargetType>();
        foreach (var sourceObject in sourceCollection ?? Enumera-
ble.Empty<TSourceType>()) {
            TTargetType existingTargetObject = getMappingTargetFromTar-
getCollectionOrNull(sourceObject);
            updatedTargetObjects.Add(existingTargetObject == null
                ? mapper.Map<TTargetType>(sourceObject)
                : mapper.Map(sourceObject, existingTargetObject));
        }
        return updateTargetCollection(updatedTargetObjects);
    }
}
```

Выводы по третьей главе

В данной главе были подробно описаны и реализованы все необходимые компоненты разрабатываемого в ходе выпускной квалификационной работы бакалавра API модуля управления задачами корпоративной си-

стемы ООО «ЭнергоИнжиниринг». Были созданы все необходимые классы, описывающие сущности предметной области, разработана логика для конфигурации и взаимодействия с базой данных приложения, был разработан функционал для работы механизма аутентификации и авторизации, реализованы основные алгоритмы, связанные с операциями над задачами и их жизненным циклом, а также создан функционал для работы с комментариями для задач. Также в этом разделе был реализован функционал взаимодействия между объектами сущностей и их DTO, а именно соответствующие преобразования и необходимые методы расширения. Стоит отметить, что также были разработаны необходимые классы-контроллеры и их методы, которые отвечают за то, чтобы обрабатывать REST запросы типа GET, POST, PUT, DELETE со стороны клиентского программного обеспечения, и использовать разработанные сервисы и модели в качестве зависимостей. Что касается архитектуры исходного кода, а именно того, как разработанные компоненты программного кода взаимодействуют между собой, можно сделать вывод о том, что реализованный функционал соответствует ранее приведенному описанию архитектуры разрабатываемого в ходе выпускной квалификационной работы бакалавра API модулю управления задачами корпоративной системы ООО «ЭнергоИнжиниринг».

4. ТЕСТИРОВАНИЕ

Инструменты тестирования

В качестве основного инструмента тестирования разработанного API используется приложение Postman [10].

Функциональное тестирование

В ходе разработки API модуля управления задачами было проведено функциональное тестирование для проверки работоспособности разработанного функционала. Входными данными для тестов являются параметры маршрутов и тела запросов в формате JSON. В таблицах 10 – 12 приведены протоколы тестирования.

В таблице 1 приведены протоколы тестирования `AccountController`.

Таблица 1 – Протоколы тестирования `AccountController`

№	Название теста	Шаги	Ожидаемый результат	Тест пройден?
1	Аутентификация	Отправить POST запрос, передав данные в формате JSON на сервер	Код состояния 200, тело ответа имеет JWT-токен	Да

В таблице 2 приведены протоколы тестирования `WorkTaskController`.

Таблица 2 – Протоколы тестирования `WorkTaskController`

№	Название теста	Шаги	Ожидаемый результат	Тест пройден?
1	Получение всех задач	Отправить GET запрос на сервер	Код состояния 200, тело ответа имеет список задач	Да
2	Получение задачи по ID	Отправить GET запрос на сервер, передав ID задачи	Код состояния 200, тело ответа имеет задачу	Да
3	Создание задачи	Отправить POST запрос на сервер, передав данные в формате JSON	Код состояния 200, новая задача добавляется в базу данных	Да
4	Редактирование задачи	Отправить PUT запрос с телом в формате JSON и ID задачи	Код состояния 200, тело ответа имеет данные измененной задачи, данные задачи изменены в БД	Да

Окончание таблицы 2

№	Название теста	Шаги	Ожидаемый результат	Тест пройден?
5	Отправка задачи на проверку	Отправить PUT запрос на сервер, передав данные в формате JSON и ID задачи на сервер	Код состояния 200, добавление записи об отправке в базу данных, изменение статуса ответственного и/или изменение статуса задачи	Да
6	Отклонение задачи	Отправить PUT запрос на сервер, передав данные в формате JSON и ID задачи на сервер	Код состояния 200, добавление записи об отклонении в базу данных, изменение статуса ответственного и/или изменение статуса задачи	Да
7	Принятие задачи	Отправить PUT запрос на сервер, передав данные в формате JSON и ID задачи на сервер	Код состояния 200, тело ответа имеет данные измененной задачи	Да
8	Удаление задачи	Отправить DELETE запрос, передав ID задачи на сервер	Код состояния 200, задача удалена из базы данных	Да
9	Добавить подзадачу	Отправить POST запрос, передав ID задачи, и данные в формате JSON на сервер	Код состояния 200, добавление подзадачи в базу данных	Да

В таблице 3 приведены протоколы тестирования `CommentController`.

Таблица 3 – Протоколы тестирования `CommentController`

№	Название теста	Шаги	Ожидаемый результат	Тест пройден?
1	Удаление комментария	Отправить DELETE запрос	Код 200, комментарий удаляется из БД и у задачи	Да
2	Создание комментария	Отправить POST запрос передав данные в формате JSON	Код состояния 201, комментарий добавляется в БД, тело ответа имеет данные созданного комментария	Да
3	Редактирование Комментария	Отправить PUT запрос с телом в формате JSON и ID комментария	Код состояния 200, комментарий обновляется в БД, тело ответа имеет данные обновленного комментария	Да
4	Ответ на комментарий	Отправить POST запрос, передав тело и ID комментария	Код состояния 201, комментарий добавляется в БД, тело ответа имеет данные комментария	Да

На рисунке 1 приложения В приведены результаты тестирования класса `AccountController`.

На рисунках 2–10 приложения В изображены результаты тестирования класса `WorkTaskController`.

На рисунках 11–14 приложения В изображены результаты тестирования класса `CommentController`.

Выводы по четвертой главе

В данной главе были приведены протоколы тестирования разработанного в ходе выпускной квалификационной работы бакалавра API модуля управления задачами корпоративной системы ООО «ЭнергоИнжиниринг» функционал. Исходя из приведенных в данной главе протоколов тестирования, а также из приведенных в приложении В рисунков, на которых изображены результаты тестов, можно сделать вывод о том, что разработанный функционал работает корректно и соответствует необходимым требованиям.

Был протестирован весь необходимый функционал таких разрабатываемых и описанных выше компонентов системы, как аутентификация пользователя, работа с задачами и работа с комментариями.

ЗАКЛЮЧЕНИЕ

В рамках выпускной квалификационной работы бакалавра был разработан API модуль управления задачами в корпоративной системе ООО «ЭнергоИнжиниринг». Поставленные и решенные задачи приведены ниже.

1. Проведен анализ предметной области и обзор существующих аналогов.
2. Разработана архитектура приложения – были выявлены и описаны основные варианты использования системой, определена архитектура приложения, а также спроектирована схема базы данных системы.
3. Разработан необходимый функционал и API – были написаны необходимые классы сущностей, сервисы по работе с базой данных, сервисы для работы с основной логикой, а также классы-контроллеры с методами для обработки запросов со стороны клиента.
4. Проведено функциональное тестирование системы, в ходе которого было выяснено, что разрабатываемые компоненты модуля работают корректно.

Разработанный в ходе выпускной квалификационной работы бакалавра API модуль управления задачами корпоративной системы ООО «ЭнергоИнжиниринг» также в будущем будет расширяться. В дальнейшем планируется разработать функционал по работе с бессрочными задачами, а также функционал по работе с уточнениями задачи.

ЛИТЕРАТУРА

1. Create Data Transfer Objects (DTOs). [Электронный ресурс] URL: https://learn.microsoft.com/en-us/aspnet/web-api/overview/data/using-web-api-with-entity-framework/part-5 (дата обращения: 06.06.2024 г.).

2. System.IdentityModel.Tokens.Jwt Namespace. [Электронный ресурс] URL: https://learn.microsoft.com/enus/dotnet/api/system.identitymodel .tokens.jwt?view=msal-web-dotnet-latest (дата обращения: 06.06.2024 г.).

3. Авторизация с помощью JWT-токенов. [Электронный ресурс] URL: https://metanit.com/sharp/aspnet5/23.7.php. (дата обращения: 06.06.2024 г.).

4. Документация по ASP.NET Core. [Электронный ресурс] URL: https://learn.microsoft.com/en-us/aspnet/core (дата обращения: 04.06.2024 г.).

5. Документация по AutoMapper. [Электронный ресурс] URL: https://docs.automapper.org/en/stable/ (дата обращения: 05.06.2024 г.).

6. Документация по Entity Framework Core. [Электронный ресурс] URL: https://learn.microsoft.com/en-us/ef/core (дата обращения: 04.06.2024 г.).

7. Документация по библиотеке Identity. [Электронный ресурс] URL: https://learn.microsoft.com/en-us/aspnet/core/security/authentication/ identity/ (дата обращения: 04.06.2024 г.)

8. Документация по созданию Web API на ASP.NET Core. [Электронный ресурс] URL:https://learn.microsoft.com/en-us/aspnet/core/web-api/ (дата обращения: 04.06.2024 г.).

9. Документация по языку программирования C#. [Электронный ресурс]URL: https://learn.microsoft.com/en-us/dotnet/csharp (дата обращения: 04.06.2024 г.).

10. Официальный сайт DBeaver. [Электронный ресурс] URL: https://dbeaver.io/ (дата обращения: 06.06.2024 г.).

11. Официальный сайт pgAdmin. [Электронный ресурс]URL: https://www.pgadmin.com (дата обращения: 04.06.2024 г.).

12. Официальный сайт Postman. [Электронный ресурс] URL: https://www.postman.com (дата обращения: 04.06.2024 г.).

13. Официальный сайт Visual Studio Code. [Электронный ресурс] URL: https://code.visualstudio.com (дата обращения: 04.06.2024 г.).

14. Официальный сайт библиотеки Newtonsoft.Json. [Электронный ресурс] URL: https://www.newtonsoft.com/json (дата обращения: 04.06.2024 г.).

15. Официальный сайт СУБД PostgreSQL. [Электронный ресурс]URL: https://www.postgresql.org (дата обращения: 04.06.2024 г.).

16. Паттерн «Репозиторий» в ASP.NET. [Электронный ресурс] URL: https://metanit.com/sharp/articles/mvc/11.php. (дата обращения: 06.06.2024 г.).

17. Создание сервисов. [Электронный ресурс] URL: https://metanit.com/sharp/aspnet6/4.2.php. (дата обращения: 06.06.2024 г.).

ПРИЛОЖЕНИЯ

Приложение А. Спецификации вариантов использования

Спецификация вариантов использования (ВИ) разрабатываемого в ходе выпускной квалификационной работы бакалавра API модуля управления задачами корпоративной системы ООО «ЭнергоИнжиниринг» приведена в таблицах 1–9.

Таблица 1 – Спецификация прецедента «Получить задачу по ID»

Прецедент: Получить задачу по ID
ID: 1
Краткое описание: Клиентское ПО отправляет запрос на получение задачи по ID
Главные актеры: Клиентское приложение
Второстепенные актеры: Нет
Предусловия: Приложение запущено на сервере, пользователь аутентифицирован, задача существует
Основной поток. Клиент посылает запрос на сервер и получает ответ
Постусловия. Происходит получение задачи по ID

Таблица 2 – Спецификация прецедента «Получить все задачи»

Прецедент: Получить все задачи
ID: 2
Краткое описание: Клиентское ПО отправляет запрос на получение всех задач
Главные актеры: Клиентское приложение
Второстепенные актеры: Нет
Предусловия: Приложение запущено на сервере, пользователь аутентифицирован
Основной поток. Клиент посылает запрос на сервер и получает ответ
Постусловия: Происходит создание/редактирование/удаление комментария в зависимости от запроса

Таблица 3 – Спецификация прецедента «Аутентифицировать пользователя»

Прецедент: Аутентифицировать пользователя
ID: 3
Краткое описание: Клиентское ПО отправляет запрос на аутентификацию пользователя
Главные актеры: Клиентское приложение
Второстепенные актеры: Нет
Предусловия: Приложение запущено на сервере, пользователь существует в БД
Основной поток. Клиент посылает запрос на сервер и получает ответ, использует полученный токен для аутентификации
Постусловия: Происходит аутентификация пользователя в системе

Таблица 4 – Спецификация прецедента «Отправить задачу на проверку»

Прецедент: Отправить задачу на проверку
ID: 4
Краткое описание: Клиентское ПО отправляет запрос на отправку задачи на проверку
Главные актеры: Клиентское приложение
Второстепенные актеры: Нет
Предусловия: Приложение запущено на сервере, пользователь аутентифицирован, пользователь является ответственным за задачу, задача существует
Основной поток. Клиент посылает запрос на сервер и получает ответ
Постусловия: Происходит отправка задачи на проверку создателю

Таблица 5 – Спецификация прецедента «Принять отправленную задачу»

Прецедент: Принять отправленную задачу
ID: 5
Краткое описание: Клиентское ПО отправляет запрос на утверждение отправленной задачи
Главные актеры: Клиентское приложение
Второстепенные актеры: Нет
Предусловия: Приложение запущено на сервере, пользователь аутентифицирован, и является создателем задачи, задача существует

Окончание таблицы 5 приложения А

Основной поток. Клиент посылает запрос на сервер и получает ответ
Постусловия: Происходит вход зарегистрированного в системе пользователя в приложение
Альтернативный поток: Происходит утверждение задачи

Таблица 6 – Спецификация прецедента «Отклонить отправленную задачу»

Прецедент: Отклонить отправленную задачу
ID: 6
Краткое описание: Клиентское ПО отправляет запрос на отклонение отправленной задачи
Главные актеры: Клиентское приложение
Второстепенные актеры: Нет
Предусловия: Приложение запущено на сервере, пользователь аутентифицирован, пользователь является создателем задачи, задача существует
Основной поток. Клиент посылает запрос на сервер и получает ответ
Постусловия: Происходит отклонение задачи

Таблица 7 – Спецификация прецедента «Редактировать задачу»

Прецедент: Редактировать задачу
ID: 7
Краткое описание: Клиентское ПО отправляет запрос на изменение данных задачи
Главные актеры: Клиентское приложение
Второстепенные актеры: Нет
Предусловия: Приложение запущено на сервере, задача существует, пользователь аутентифицирован
Основной поток. Клиент посылает запрос на сервер и получает ответ
Постусловия: Происходит изменение задачи

Таблица 8 – Спецификация прецедента «Удалить задачу»

Прецедент: Удалить задачу
ID: 8
Краткое описание: Клиентское ПО отправляет запрос на удаление задачи

Окончание таблицы 8 приложения А

Главные актеры: Клиентское приложение
Второстепенные актеры: Нет
Предусловия: Приложение запущено, задача существует, пользователь аутентифицирован
Основной поток: Клиент посылает запрос на сервер и получает ответ
Постусловия: Происходит удаление задачи

Таблица 9 – Спецификация прецедента «Создать задачу»

Прецедент: Создать задачу
ID: 9
Краткое описание: Клиентское ПО отправляет запрос на создание задачи
Главные актеры: Клиентское приложение
Второстепенные актеры: Нет
Предусловия: Приложение запущено на сервере, пользователь аутентифицирован
Основной поток. Клиент посылает запрос на сервер и получает ответ
Постусловия: Происходит создание задачи

Приложение Б. Класс AppDbContext и Repository

В листинге 1 представлен класс `AppDbContext`.

Листинг 1 – Класс `AppDbContext`

```
public class AppDbContext(DbContextOptions opts) : IdentityDbContext(opts)
{
    public DbSet<ApplicationUser> ApplicationUsers { get; set; }
    public DbSet<WorkTask> WorkTasks { get; set; }
    public DbSet<WorkTaskResponsibleUser> ResponsibleUsers { get; set; }
    public DbSet<CommentsSection> CommentsSections { get; set; }
    public DbSet<Comment> Comments { get; set; }
    public DbSet<WorkTaskLabel> WorkTasksLabels { get; set; }
    public DbSet<ExecutionHistory> WorkTasksExecutionHistory { get; set; }

    protected override void OnModelCreating(ModelBuilder builder)
    {
        base.OnModelCreating(builder);
        builder.Entity<WorkTask>()
            .HasOne(t => t.CreatedBy)
            .WithMany(u => u.WorkTasks)
            .OnDelete(DeleteBehavior.SetNull);
        builder.Entity<WorkTaskResponsibleUser>()
            .HasOne(u => u.ApplicationUser)
            .WithMany(x => x.TaskResponsibilities)
            .OnDelete(DeleteBehavior.Cascade);
        builder.Entity<WorkTaskResponsibleUser>()
            .HasOne(u => u.WorkTask)
            .WithMany(x => x.ResponsibleUsers)
            .OnDelete(DeleteBehavior.Cascade);
        builder.Entity<Comment>()
            .HasOne(u => u.User)
            .WithMany(x => x.Comments)
            .OnDelete(DeleteBehavior.Cascade);
        builder.Entity<WorkTask>()
            .HasOne(u => u.CommentsSection)
            .WithOne(x => x.WorkTask)
            .HasForeignKey<CommentsSection>(u => u.WorkTaskId)
            .OnDelete(DeleteBehavior.Cascade);
        builder.Entity<Comment>()
            .HasOne(u => u.CommentsSection)
            .WithMany(x => x.Comments)
            .OnDelete(DeleteBehavior.Cascade);
        builder.Entity<Comment>()
            .HasOne(u => u.Parent)
            .WithMany(x => x.Children)
            .OnDelete(DeleteBehavior.Cascade);
        builder.Entity<WorkTask>()
            .HasOne(u => u.ParentWorkTask)
            .WithMany(u => u.SubWorkTasks)
            .OnDelete(DeleteBehavior.Cascade);
        builder.Entity<ExecutionHistory>()
            .HasOne(u => u.Responsible)
            .WithMany(u => u.History)
            .OnDelete(DeleteBehavior.Cascade);
    }
}
```

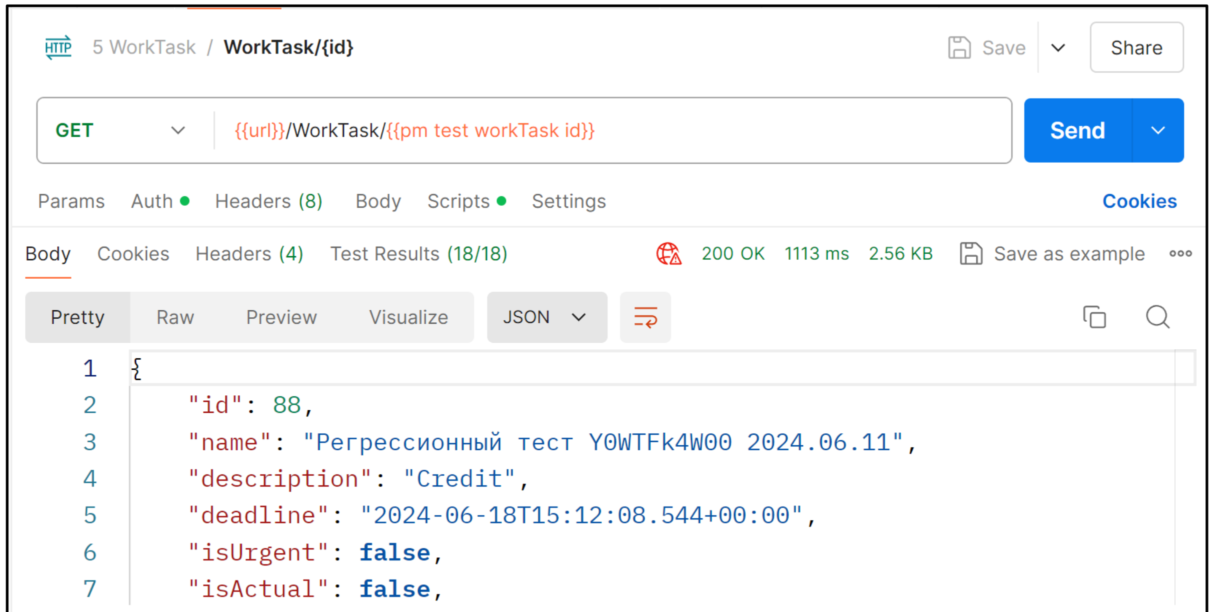

Рисунок 3 – Результаты тестирования получения задачи по ID

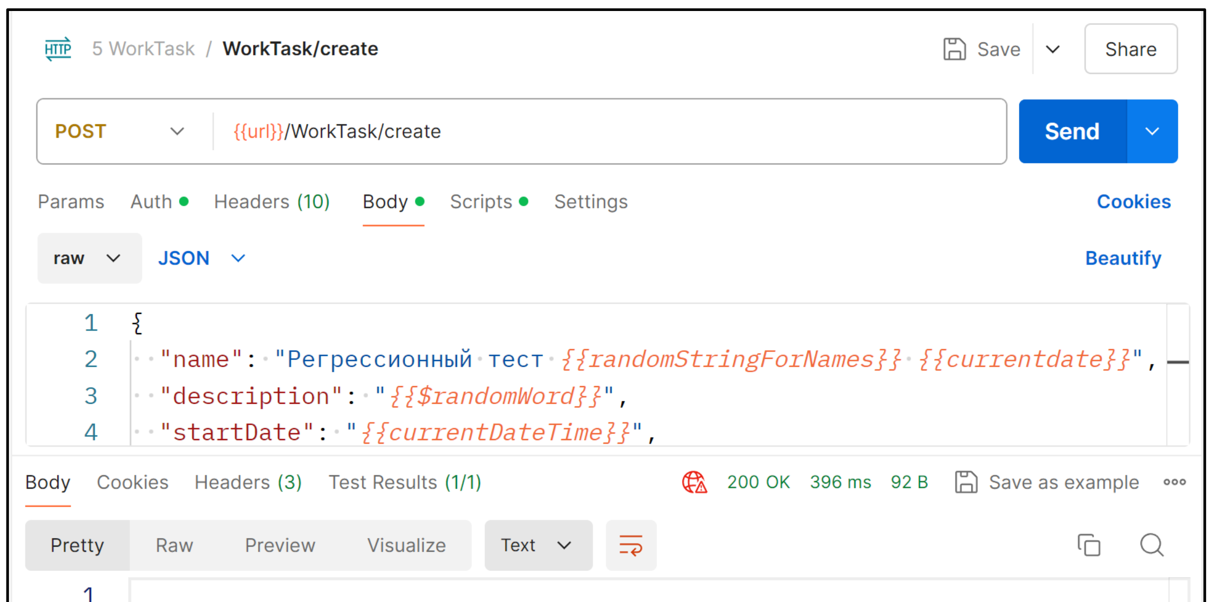

Рисунок 4 – Результаты тестирования создания задачи

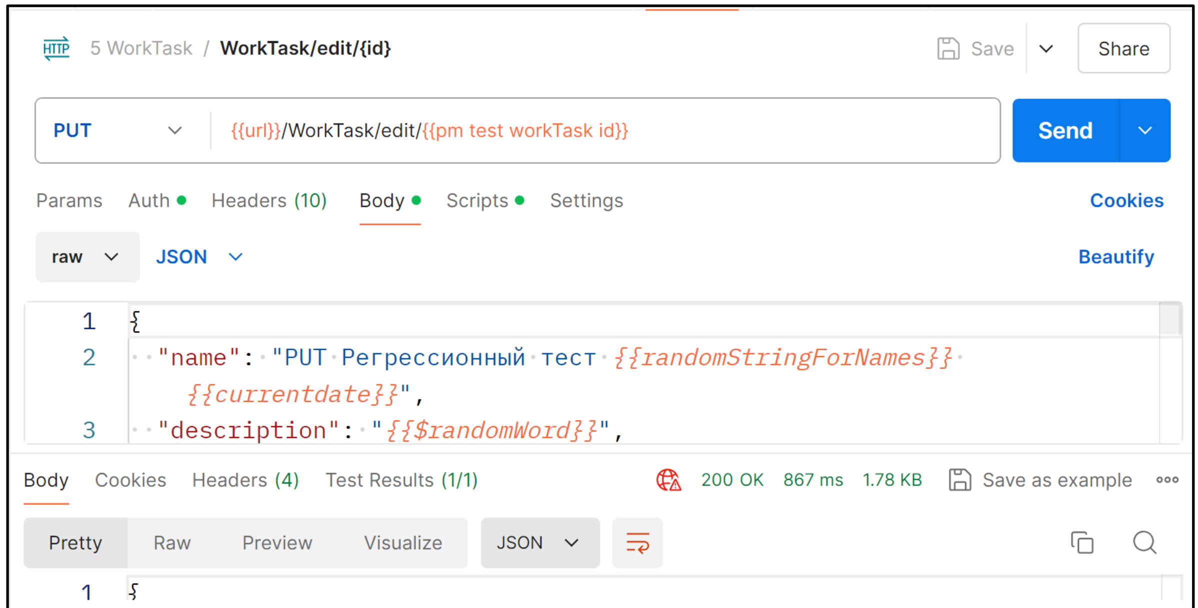

Рисунок 5 – Результаты тестирования редактирования задачи

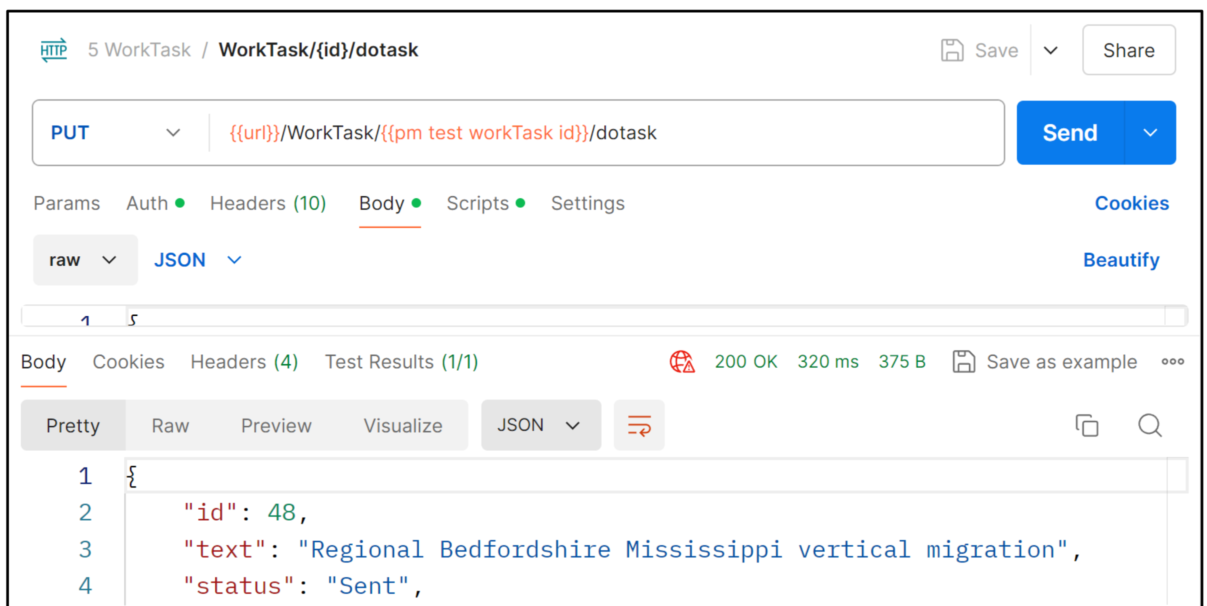

Рисунок 6 – Результаты тестирования отправки задачи на проверку

Рисунок 7 – Результаты тестирования отклонения задачи

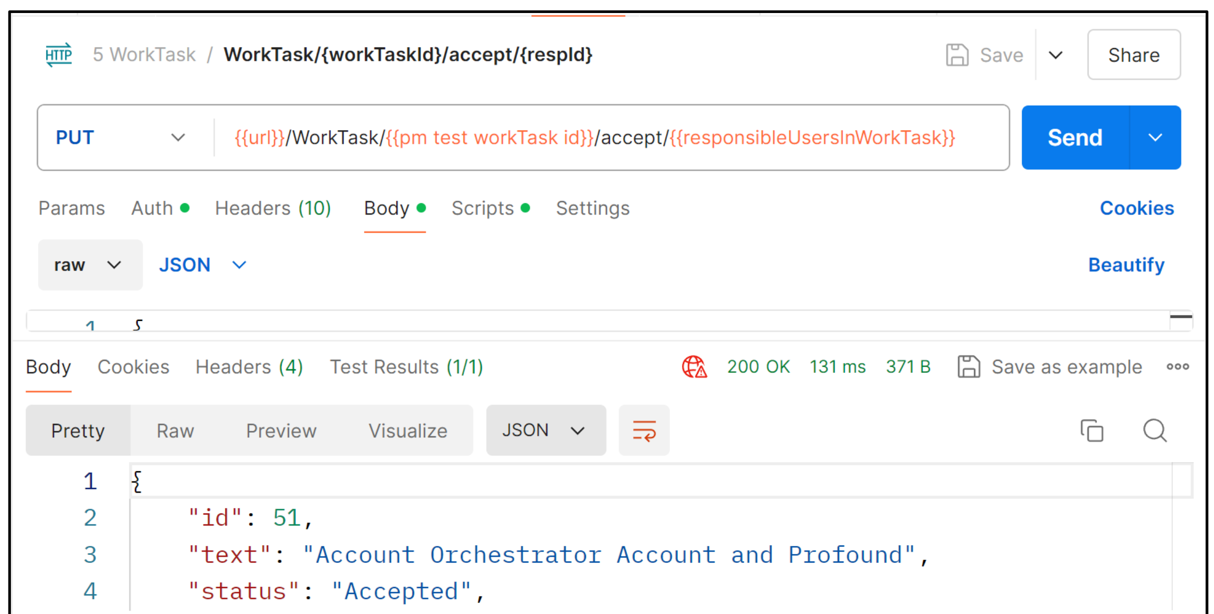

Рисунок 8 – Результаты тестирования принятия задачи

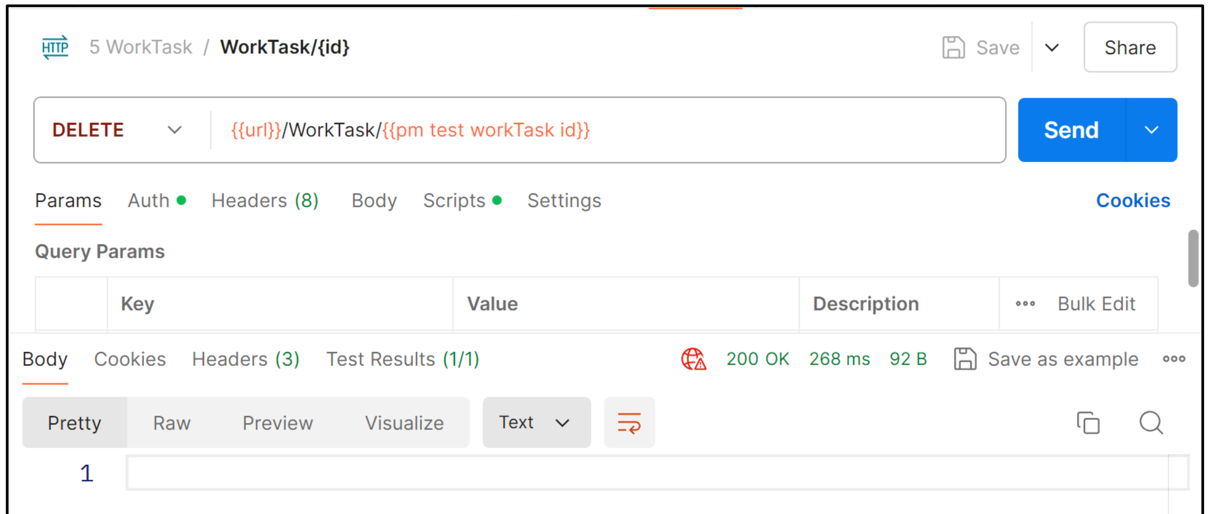

Рисунок 9 – Результаты тестирования удаления задачи

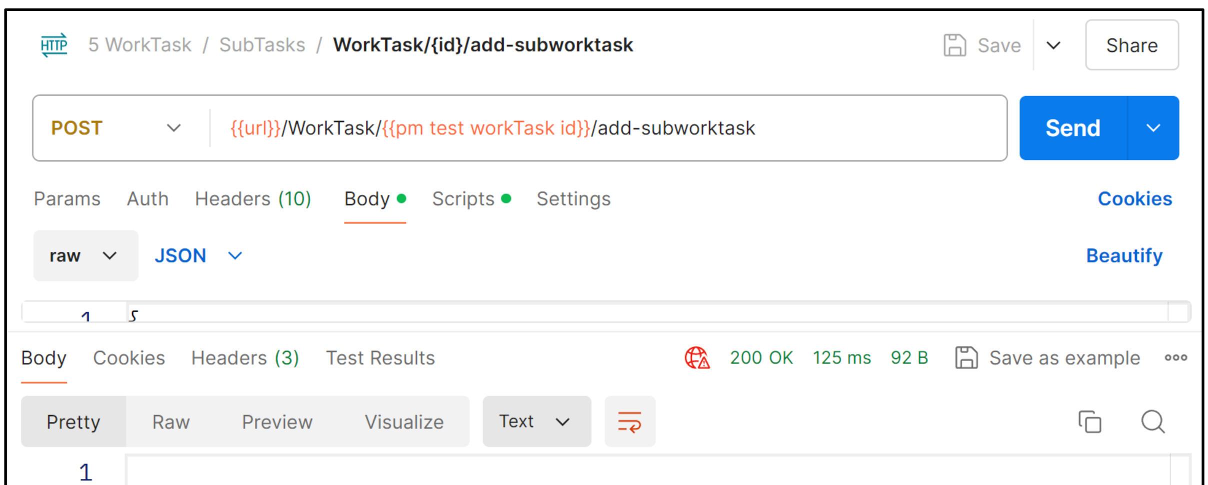

Рисунок 10 – Результаты тестирования добавления подзадачи

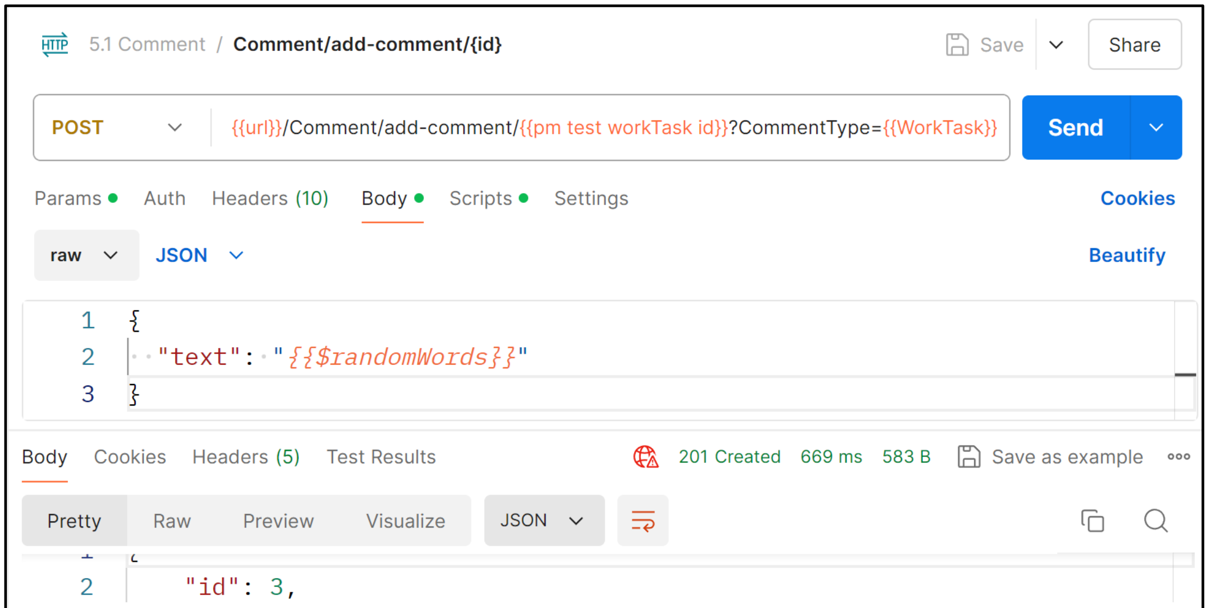

Рисунок 11 – Результаты тестирования добавления комментария

Рисунок 12 – Результаты тестирования редактирования комментария

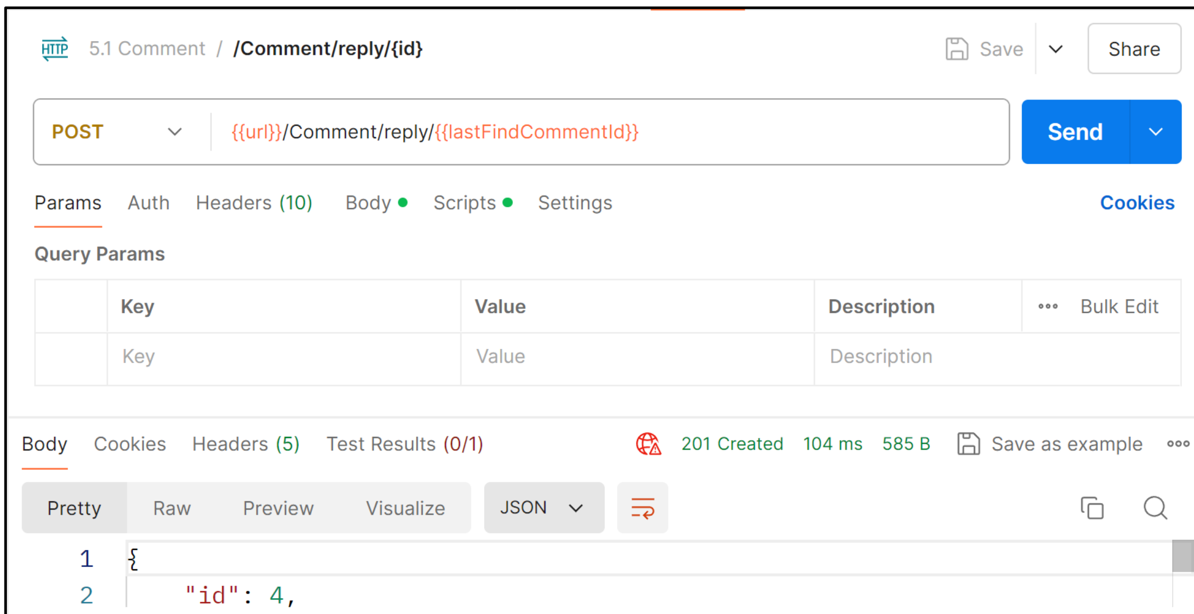

Рисунок 13 – Результаты тестирования добавления ответа на комментарий

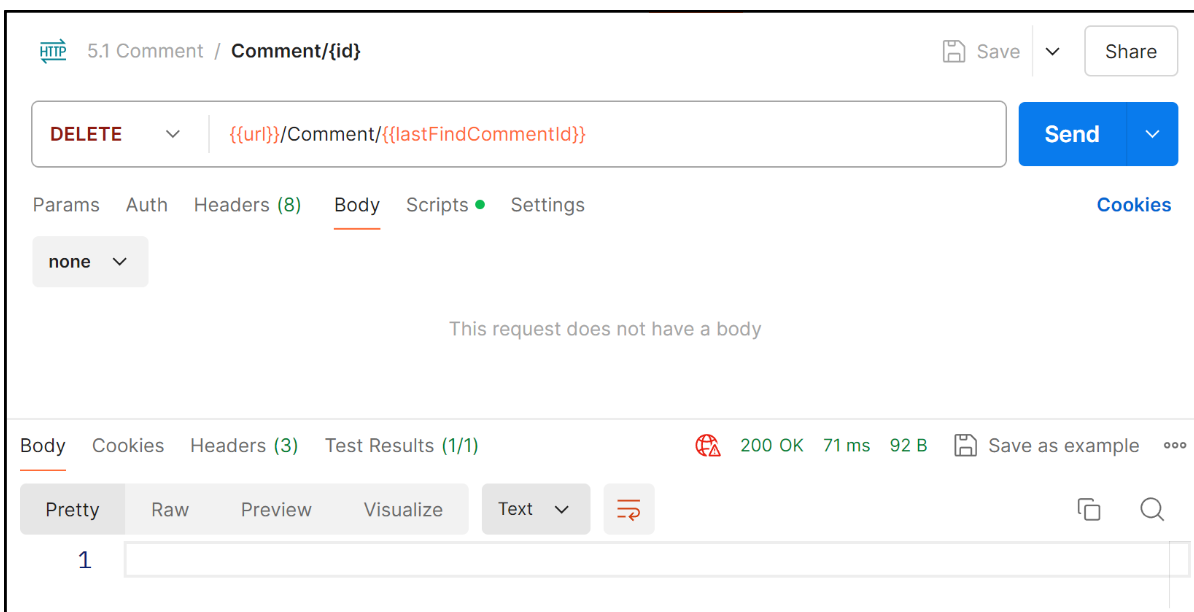

Рисунок 14 – Результаты тестирования удаления комментария