

МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РФ

Федеральное государственное бюджетное образовательное учреждение
высшего образования

ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

ПОЛИТЕХНИЧЕСКИЙ ИНСТИТУТ

Факультет

вычислительной техники

Кафедра

Вычислительная техника

Направление подготовки 09.04.01 «Информатика и вычислительная техника»

Магистерская программа Вычислительные машины, комплексы, системы и сети

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ

РАБОТА МАГИСТРА

на тему

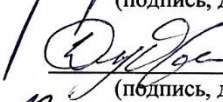
АВТОМАТИЗИРОВАННАЯ СИСТЕМА ПОДБОРА
ПЕРСОНАЛА ДЛЯ ФОРМИРОВАНИЯ ПРОЕКТНЫХ ГРУПП

Студент

 6.06.18
(подпись, дата)

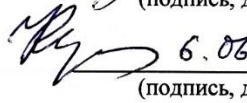
Миронова Елена Евгеньевна
(ФИО полностью)

Руководитель

 6.06.18
(подпись, дата)

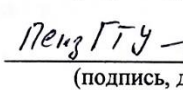
А.В. Дубравин
(фамилия, инициалы)

Нормоконтролёр

 6.06.2018
(подпись, дата)

А.В. Кучин
(фамилия, инициалы)

Рецензент

Доцент каф. ВМиС ПензГУ  8.06.18
(должность, место работы) (подпись, дата)

С.В. Печерский
(фамилия, инициалы)

Работа допущена к защите (протокол зас. кафедры от 25.05.2018г. № 38-09-12)

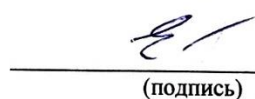
Заведующий кафедрой


(подпись)

Д.В. Пащенко
(фамилия, инициалы)

Работа защищена с отметкой  (протокол заседания ГЭК от 14.06.18г. № 1)

Секретарь ГЭК


(подпись)

Е.И. Гурин
(фамилия, инициалы)

Пенза, 2018

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

Факультет

Вычислительной техники

Кафедра

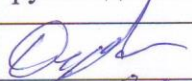
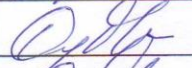
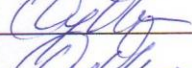
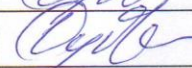
Вычислительная техника

«Утверждаю»
Заведующий кафедрой
Д.В. Пашенко
« » 20 г

**ЗАДАНИЕ
НА ВЫПУСКНУЮ КВАЛИФИКАЦИОННУЮ РАБОТУ
МАГИСТРА**

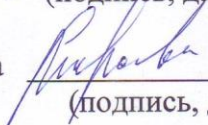
1. Студент Миронова Елена Евгеньевна гр. 16ВВм1 факультета вычислительной техники.
2. Тема работы «Автоматизированная система подбора персонала для формирования проектных групп».
Тема утверждена приказом ПГУ № 979 а/о от "3" октября 2016 г.
3. Руководитель работы к.т.н., доцент каф. ВТ ПГУ Дубравин В.А.
4. Задание на работу
Назначение разработки: формирование проектных групп на основе профессиональных качеств сотрудников организации.
Разработать автоматизированную систему подбора персонала в виде сервера авторизации и REST-ful веб-сервиса. Взаимодействие с системой должно быть доступно как администраторам системы, так и обычным пользователям.
5. Перечень подлежащих разработке вопросов.
Требуется разработать две базы данных: для учетных данных и для хранения информации о проектах и сотрудниках. Для обеспечения целостности информации необходимо ограничить доступ к некоторым компонентам системы для пользователей различных категорий. Со стороны клиента должен отправляться HTTP-запрос, сервер в свою очередь должен обрабатывать поступающие запросы и выдавать соответствующую информацию.

6. Календарный график выполнения работы

№ п/п	Наименование этапов работы	Объем работы	Срок выполнения	Подпись руководителя
1.	Анализ предметной области	10	21.05.18	
2.	Выбор средств для реализации	5	23.05.18	
3.	Проектирование баз данных	5	24.05.18	
4.	Разработка компонента авторизации	10	25.05.18	
5.	Разработка REST-ful веб-сервиса	30	27.05.18	
6.	Отладка приложения на ЭВМ	10	03.06.18	
7.	Оформление выпускной работы	30	05.06.18	

Дата выдачи задания "21" мая 2018 г.

Руководитель магистерской работы  Дубравин А.В.
(подпись, дата)

Задание к исполнению приняла студентка  21.05.18 Миронова Е.Е.
(подпись, дата)

Работу к защите допустить
Декан факультета  09.06.2018 Фионова Л.Р.
(подпись, дата)

Заведующему кафедрой
"Вычислительная техника"
Д. В. Пащенко
студентки 16ВВм1
Мироновой Елены Евгеньевны

ЗАЯВЛЕНИЕ

Прошу утвердить в качестве руководителя ВКР к.т.н, доцента кафедры
«Вычислительная техника» Дубравина Алексея Викторовича.

Руководитель – тема ВКР: «Автоматизированная система подбора
персонала для формирования проектных групп».

Дата:

14.09.2016

Руководитель



Студентка

Миронова Е.Е. 

Ректору Пензенского
государственного университета
А. Д. Гулякову
студентки 2 курса ФВТ
очной формы обучения
Мироновой Елены Евгеньевны

ЗАЯВЛЕНИЕ

Я, Миронова Елена Евгеньевна, магистрант 2 курса очной формы обучения направления подготовки 09.04.01 Информатика и вычислительная техника, магистерская программа «Вычислительные машины, комплексы, системы и сети», прошу провести проверку с использованием системы «Антиплагиат» выпускной квалификационной работы магистра на тему «Автоматизированная система подбора персонала для формирования проектных групп», выполненной мной самостоятельно, на содержание элементов плагиата.

Все прямые заимствования из печатных и электронных источников, а также из защищенных ранее выпускных квалификационных работ, кандидатских и докторских диссертаций имеют соответствующие ссылки.

Я ознакомлена с действующим Стандартом университета «Выпускная квалификационная работа обучающихся по образовательным программам высшего образования – программам бакалавриата, программам специалитета и программам магистратуры», согласно которому обнаружение плагиата является основанием для недопуска ВКР к защите и отчисления из университета.

ВКР представлена на проверку «24» мая 2018 г.

Руководитель ВКР Дубравин А.В.

Протокол

проверки на оригинальность в системе «Антиплагиат.ВУЗ» выпускной квалификационной работы на тему «Автоматизированная система подбора персонала для формирования проектных групп» магистранта 2 курса Мироновой Е.Е. направления подготовки 09.04.01 Информатика и вычислительная техника, магистерская программа «Вычислительные машины, комплексы, системы и сети».

Руководитель ВКР к.т.н, доцент каф. ВТ Дубравин А.В.



Отчет о проверке на заимствования №1

Автор: miroнова3939@yandex.ru / ID: 5700339

Проверяющий: miroнова3939@yandex.ru / ID: 5700339

Отчет предоставлен сервисом «Антиплагиат» - <http://www.antiplagiat.ru>

ИНФОРМАЦИЯ О ДОКУМЕНТЕ

№ документа: 1
Начало загрузки: 24.05.2018 13:01:24
Длительность загрузки: 00:00:03
Имя исходного файла: Диплом
Размер текста: 3274 кБ
Символов в тексте: 152064
Слов в тексте: 16690
Число предложений: 1600

ИНФОРМАЦИЯ ОБ ОТЧЕТЕ

Последний готовый отчет (ред.)
Начало проверки: 24.05.2018 13:01:28
Длительность проверки: 00:00:04
Комментарии: не указано
Модули поиска:

ЗАИМСТВОВАНИЯ	ЦИТИРОВАНИЯ	ОРИГИНАЛЬНОСТЬ
9,83%	0%	90,17%

№	Доля в отчете	Доля в тексте	Источник	Ссылка	Актуален на	Модуль поиска	Блоков в отчете	Блоков в тексте
[01]	1,28%	1,28%	http://www.rusnauka.com/21_AND_2017...	http://rusnauka.com	03 Авг 2017	Модуль поиска Интернет	8	8
[02]	1,25%	1,25%	TYPE CAST EXCEPTION ASP.Net (2/2)	http://typecastexception.com	06 Янв 2016	Модуль поиска Интернет	57	57
[03]	1,17%	1,17%	Введение в JSON «Alexandr's Blog»	http://alexandr.logdown.com	05 Июл 2017	Модуль поиска Интернет	8	8

Еще источников: 17

Еще заимствований: 6,15%

«24» мая 2018 г.

 А.В. Дубравин

ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

Факультет вычислительной техники

Кафедра «Вычислительная техника»

ОТЗЫВ

на выпускную квалификационную работу магистра

Мироновой Елены Евгеньевны

Тема работы: Автоматизированная система подбора персонала для формирования проектных групп.

Цель работы: Разработка автоматизированной системы, позволяющей формировать проектные группы на основе профессиональных данных сотрудников организации.

Направление: 09.04.01 «Информатика и вычислительная техника»

Профиль: «Вычислительные машины, комплексы, системы и сети»

Выпускная квалификационная работа магистра, выполненная магистрантом Мироновой Е. Е., относится к области разработки веб-сервисов на платформе ASP.NET.

Благодаря подробному анализу предметной области были выделены основные способы, методы и технологии для реализации поставленной цели. Были разработаны компонент авторизации пользователей в системе и компонент обработки запросов пользователей, детально спроектированы и реализованы базы данных для них. Разработанные компоненты были объединены в единую систему и размещены на сервере.

В ходе работы была разработана универсальная автоматизированная система подбора персонала, позволяющая формировать проектные группы, основываясь на соотношении требований проекта и профессиональных навыках сотрудников.

Поставленные в работе исследовательские и практические задачи магистрантом Мироновой Е.Е. выполнены в полном объеме. Пояснительная записка оформлена в соответствии с действующими стандартами.

В процессе выполнения выпускной квалификационной работы Миронова Е.Е. проявила себя, как квалифицированный специалист с высоким уровнем подготовки, обладающий хорошими знаниями в сфере программирования, продемонстрировала высокий уровень теоретических и практических знаний в предметной сфере.

За время выполнения выпускной квалификационной работы был продемонстрирован значительный уровень теоретической подготовки, умение применить и использовать знания, полученные в ВУЗе, для решения практических задач.

С поставленными задачами Миронова Е.Е. справилась успешно. Степень оригинальности работы 90,17%.

Считаю, что выпускная квалификационная работа магистра выполнена на должном уровне и заслуживает оценки «отлично», а магистрант Миронова Е.Е. заслуживает присвоения ей квалификации магистра по направлению подготовки 09.04.01

Руководитель выпускной квалификационной работы магистранта

к.т.н., доцент каф. ВТ



А. В. Дубравин

РЕЦЕНЗИЯ

на выпускную квалификационную работу студента группы 16ВВм1
Пензенского государственного университета
Мироновой Елены Евгеньевны
на тему: "Автоматизированная система подбора персонала для
формирования проектных групп"

Выпускная квалификационная работа (ВКР) Е.Е. Мироновой посвящена решению актуальной для многих организаций задачи разработки автоматизированной системы подбора персонала.

Разработанная система позволяет формировать списки сотрудников для участия в проектах посредством сопоставления требований имеющихся проектов с характеристиками сотрудников предприятия. На основе совпадений подбираются подходящие для проекта сотрудники или проекты, доступные для сотрудника. Доступ к системе могут иметь как ответственные за подбор персонала, так и обычные сотрудники.

Автором были разработаны сервисы для авторизации пользователей в системе и обработки запросов пользователей. Помимо этого, для каждого сервиса были спроектированы и смоделированы базы данных для хранения соответствующей информации. Было предусмотрено разделение ролей пользователей с разграничением уровня доступа к функциям системы.

В процессе работы Е.Е. Миронова проявила себя грамотным специалистом, способным самостоятельно решать сложные технические задачи. Работа выполнена качественно и аккуратно, соответствует заданию, отвечает требованиям ЕСПД. Пояснительная записка составлена грамотно и раскрывает заданную тему в полном объеме.

Достоинством разработки можно считать возможность её использования в различных промышленных сферах.

К недостатку работы можно отнести незначительные неточности в оформлении пояснительной записки.

Компетенции освоены в полном объёме.

Результаты ВКР могут быть с успехом использованы в организациях с большим штатом работников для автоматизации процесса подбора персонала в проектные группы.

Считаю, что ВКР Е.Е. Мироновой может быть допущена к защите, заслуживает оценки "отлично", а ее автор присвоения степени магистра по направлению 09.04.01 – "Информатика и вычислительная техника".

Доцент кафедры "ВМиС"
Пензенского государственного
технологического университета, к.т.н.

С.В. Печерский

08.06.2018 г.



Реферат

Магистерская работа содержит 79 листов, 41 рисунок, 29 источников, 3 приложения.

СЕРВЕР, БАЗА ДАННЫХ, СЕТЕВОЕ ПРИЛОЖЕНИЕ, REST-FUL, АВТОРИЗАЦИЯ, HTTP-ЗАПРОС, КЛИЕНТ-СЕРВЕР, ASP.NET, ТОКЕН

Цель работы – разработать автоматизированную систему, позволяющую формировать проектные группы на основе профессиональных данных сотрудников организации.

Объектами разработки являются сервера для авторизации и обработки запросов пользователей. Сервер авторизации предоставляет пользователю токен доступа. Сервер обработки запросов обменивается с пользователем информацией на основе HTTP-запросов.

В результате проведенной работы создан проект, удовлетворяющий поставленной задаче. Была разработана автоматизированная система, которая, основываясь на сопоставлении данных между проектами и сотрудниками, формирует списки пользователей, подходящих под соответствующий проект. Система позволяет пополнять и обновлять информацию в базах данных. Взаимодействие с системой доступно пользователям различных категорий.

Данный проект рекомендуется использовать в организациях для автоматизации процесса подбора персонала для выполнения задач, соответствующих специфике предприятия.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ	6
1 ПОСТАНОВКА ЗАДАЧИ	9
2 ОБЗОР СРЕДСТВ ДЛЯ РЕШЕНИЯ ЗАДАЧИ	11
2.1 Выбор языка разработки	11
2.2 Платформа ASP.NET	12
2.3 OWIN	13
2.4 Entity Framework	15
2.4.1 Подход Code-First	17
2.4.2 Библиотека AutoMapper	17
2.5 Microsoft SQL Server	18
2.6 Авторизация на основе токенов	19
2.7 Формат JSON	23
2.8 Среда разработки Microsoft Visual Studio	24
2.9 Internet Information Services	25
2.10 Advanced REST Client	26
2.11 Выбранные средства для решения задачи	27
3 РАЗРАБОТКА БАЗЫ ДАННЫХ	29
3.1 Анализ предметной области	29
3.2 Проектирование базы данных учетных данных сотрудников	30
3.3 Проектирование базы данных хранения информации о проектах и профессиональных данных сотрудников	31
3.3.1 Сущность «Сотрудники»	31
3.3.2 Сущность «Специализации»	32
3.3.3 Сущность «Должности»	32

3.3.4	Сущность «Проекты».....	33
3.3.5	Сущность «Участники проекта».....	33
3.3.6	Сущность «Этапы проекта»	34
3.3.7	Сущность «Заявки»	34
3.3.8	Диаграмма «сущность-связь»	35
3.4	Разработка базы данных	35
3.4.1	Класс DbSpecialization	36
3.4.2	Класс DbProject.....	37
3.4.3	Класс DbJobPosition	38
3.4.4	Класс DbEmployeeJobPosition	39
3.4.5	Класс DbEmployeeSpecialization	39
3.4.6	Класс DbProjectMember	40
3.4.7	Класс DbProjectStage	41
3.4.8	Класс DbProjectApplications	41
3.4.9	Диаграмма классов базы данных	42
4	РАЗРАБОТКА СИСТЕМЫ.....	43
4.1	Разработка компонентов системы	44
4.1.1	Компонент управления учетными записями пользователей	44
4.1.2	Компонент обработки запросов пользователей	45
5	ВЗАИМОДЕЙСТВИЕ С БАЗАМИ ДАННЫХ.....	48
5.1	Подключение к базам данных	48
5.2	Работа с базами данных	49
6	ПУБЛИКАЦИЯ И ТЕСТИРОВАНИЕ СИСТЕМЫ	51
6.1	Размещение системы на сервере	51
6.2	Проверка реакции системы на HTTP-запросы пользователей	53

6.2.1	Регистрация нового пользователя	53
6.2.2	Вход в систему.....	54
6.2.3	Общедоступный запрос	55
6.2.4	Запрос с ограничением прав доступа.....	57
6.2.5	Выход пользователя из системы.....	57
6.3	Проверка работоспособности системы	58
6.3.1	Тестирование относительно сотрудника	58
6.3.2	Тестирование относительно администратора	64
ЗАКЛЮЧЕНИЕ		76
СПИСОК ИСПОЛЬЗУЕМЫХ ИСТОЧНИКОВ		77
ПРИЛОЖЕНИЕ А ЛИСТИНГИ ПРОГРАММ (обязательное)		80
Приложение А.1 Листинг сервиса TeamHunterIdentity.....		80
Приложение А.2 Листинг сервиса TeamHunter		88
ПРИЛОЖЕНИЕ Б ДИАГРАММЫ UML (рекомендуемое).....		134
Приложение Б.1 Диаграмма развертывания		134
Приложение Б.2 Диаграмма сущность-связь.....		135
Приложение Б.3 Диаграмма классов базы данных		136
Приложение Б.4 Диаграмма прецедентов относительно администратора		137
Приложение Б.5 Диаграмма прецедентов относительно пользователя.....		138
ПРИЛОЖЕНИЕ В ПРЕЗЕНТАЦИЯ (рекомендуемое)		139

ВВЕДЕНИЕ

Эффективная система подбора персонала играет большую роль в деятельности компании. В ходе подбора кадров осуществляется качественный отбор трудовых ресурсов, используемых предприятиями, что особенно важно для компаний, штат которых может превышать сотни человек. Однако, многие организации не уделяют достаточно внимания этому процессу, что, как правило, приводит к тому, что подбор персонала в них организован хаотично. Любая система отбора кадров предназначена для того, чтобы дать возможность компании получить в свой штат лучших из лучших. Кроме того, такая система позволяет распределить текущие задачи между действующими сотрудниками в соответствии с их профессиональными качествами для оптимизации производственного процесса.

Выбор кандидата может основываться на различных критериях, определяющих уровень его компетенции. Использование того или иного критерия зависит от размера предприятия, отрасли, специфики выполняемых работ, уровня ответственности и многих других критериях.

Для правильного подбора персонала должно быть четкое понимание внутри самой фирмы: кто, когда, зачем и на каких условиях нужен. В кадровой службе, отделе по персоналу компании должны быть собственные сильные HR-менеджеры, которые должны хорошо знать специфику, внутреннюю «кухню» фирмы, слабые и сильные места коллектива. Они должны уметь вести правильную информационную и учетную политику, грамотно оценивать сотрудников, понимать специфику поиска и привлечения кандидатов, владеть современными HRM-технологиями. Все это дает возможность эффективно подбирать персонал, не прибегая к помощи специализированных агентств [1].

При реализации проекта по поиску и привлечению персонала возникает масса других проблем и вопросов: какие каналы поиска задействовать, обращаться ли в компании, подбирающие кадры, какую рекламную стратегию

выбрать, сколько менеджеров по персоналу подключить, сколько времени этому уделить, как правильно рассчитать бюджет и другие ресурсы. А еще: какую базу для учета кандидатов и резюме использовать, где хранить и как обрабатывать огромный объем информации, как не потерять ни одно резюме и не упустить ни одного соискателя, как оценить эффективность подбора персонала. Наконец, как подобрать нужного специалиста – того, кто будет максимально эффективен в работе, какие методики и технологии использовать, из каких критериев исходить [1].

Эффективно решить все упомянутые проблемы и задачи возможно лишь с помощью автоматизации процессов поиска, привлечения, подбора и учета кадров. Внедрение автоматизированной системы позволит усовершенствовать работу с подбором кандидатов, с ведением базы вакансий, соискателей и резюме, поможет повысить управляемость и результативность всей работы с персоналом [1].

При внедрении автоматизированной системы подбора персонала на предприятии появляются следующие выгоды и преимущества:

- Сокращаются трудозатраты, связанные с планированием и реализацией подбора кадров.
- Минимизируются расходы на подбор кадров.
- Ускоряется поиск и привлечение кандидатов.
- Постоянный рост базы.
- Ведется грамотный учет, исключаются потери данных.
- Качественно и полноценно прорабатываются все обращения и резюме.
- Происходит рост конверсии потенциальных кандидатов, соискателей, сотрудников и работников.
- Увеличивается эффективность рекламных и PR-мероприятий, направленных на поиск и привлечение кандидатов.
- Оптимизируется управление отдела по работе с персоналом.

Данная работа разделена на два компонента: один из которых должен отвечать за управление учетными записями пользователей системы, другой – за обработку запросов, поступающих на сервер.

Актуальность данной работы обусловлена тем, что в связи с развитием рыночной экономики в России, все большую значимость приобретает рынок трудовых ресурсов, поэтому политика предприятия по отбору и набору кадров является главенствующей и играет важнейшую роль для достижения миссии и целей организации. На современном рынке программных продуктов не существует аналогов вышеописанной системы с двусторонним взаимодействием сотрудника предприятия и ответственными за подбор персонала.

1 ПОСТАНОВКА ЗАДАЧИ

Целью работы является разработка автоматизированной системы подбора персонала для формирования проектных групп.

Система должна состоять из двух модулей, для каждого будет создана своя база данных. База данных первого компонента должна хранить идентифицирующую информацию пользователей системы, второго – данные, по которым ведется отбор. Задача баз данных заключается в хранении необходимой информации в нескольких таблицах, связанных между собой или независимых друг от друга. Создание нескольких баз данных преследует две основные цели: понизить избыточность данных и повысить их надежность.

Система должна быть универсальна – архитектура позволит использовать ее в различных промышленных областях. Кроме того, не составит большого труда разработать под эту систему клиентское, браузерное или мобильное приложение, что обусловлено особенностью передачи данных между клиентским и серверным модулями.

Возможность пользоваться разрабатываемой системой должна предоставляться как ответственным за подбор персонала, так и обычным пользователям системы – сотрудникам предприятия. Для этого необходимо предусмотреть разделение ролей на администраторов и обычных пользователей системы, ограничив при этом некоторые права последних. Сотрудники предприятия смогут указывать свои профессиональные качества, видеть список доступных проектов. Администраторы системы – отвечать за обновление базы данных пользователей и базы, связанной с деятельностью предприятия, а также формировать списки участников проектов.

Для достижения поставленной цели требуется выполнить ряд подзадач:

- 1) Выбрать наиболее подходящие средства и методы для решения поставленной задачи.

- 2) Спроектировать базу данных для хранения учетных данных пользователей системы.
- 3) Спроектировать базу данных для хранения информации о проектах и профессиональных данных сотрудников.
- 4) Программно реализовать структуры разработанных баз данных.
- 5) Разработать сервис авторизации пользователей в системе.
- 6) Разработать сервисы, обрабатывающие запросы, связанные с проектной и профессиональной деятельностью.
- 7) Определить роли пользователей системы.
- 8) Разграничить возможности пользователей.
- 9) Обеспечить корректность выполняемых запросов.
- 10) Протестировать готовую систему.

Выполнение перечисленных подзадач поэтапно и в указанной последовательности позволит максимально точно и за достаточно короткий период времени достигнуть поставленной цели.

2 ОБЗОР СРЕДСТВ ДЛЯ РЕШЕНИЯ ЗАДАЧИ

2.1 Выбор языка разработки

Сегодня существует огромное количество языков программирования. Каждый язык был придуман и создан для решения определенного типа задач. Большая часть языков пересекается в функционале, поэтому одну и ту же задачу можно решать различными инструментами, но при этом, у каждого языка программирования есть свои преимущества, даже при решении задач одинакового типа. Например, на одном языке программирования написание программы займет довольно много времени, но в итоге получится компактное по весу и быстро работающее приложение, что очень важно при решении сложных вычислительных задач или при большом объеме данных. На другом языке написать программу получится во много раз быстрее, но она, возможно, будет дольше выполняться, больше будет размер исполняемого файла кода [2].

Для выполнения работы в качестве языка программирования был выбран язык C#. C# – это современный объектно-ориентированный язык программирования, разработанный для создания множества приложений, работающих в среде .NET Framework. Язык C# прост, типобезопасен и объектно-ориентирован. Благодаря множеству нововведений C# обеспечивает возможность быстрой разработки приложений, но при этом сохраняет выразительность и элегантность, присущую C-подобным языкам.

Язык C# был выпущен в середине 2000 года группой программистов компании Microsoft под руководством Аннерса Хейлсберга (Anders Hejlsberg) – создателя Turbo Pascal и Delphi.

Основой языка является C++. Однако, некоторые детали были позаимствованы из языка Java (например, пространства имен). Еще одно перекочевавшее из Java достоинство, это система автоматического управления памятью. На систему возложены функции выделения памяти для различных классов и ее освобождения после прекращения использования классов. Из

старых языков программирования (например, из Паскаля) были взяты две следующие возможности. Первая – четкая типизация переменных. При описании переменной необходимо указать ее тип, чего не было в C++. Вторая – автоматическая инициализация переменных [3].

В языке C# появилась возможность использования библиотек других систем программирования, разработанных фирмой Microsoft, как, например, Visual Basic.

Главной особенностью языка C# является его ориентированность на платформу Microsoft .NET. Программист, использующий C# может лучше сконцентрироваться на содержательной части задачи, благодаря сокрытию от него большого количества незначительных технических деталей, включая операции по упаковке/распаковке типов, инициализации переменных и сборке мусора.

Программа на языке C# выполняется в среде .NET Framework – интегрированном компоненте Windows, содержащем виртуальную систему выполнения (среда CLR) и унифицированный набор библиотек классов [4].

2.2 Платформа ASP.NET

ASP.NET является платформой для разработки веб-приложений от компании Microsoft, входит в состав платформы .NET Framework и использует все возможности, предоставляемые этой платформой [5].

Так как основой для ASP.NET выступает Common Language Runtime (CLR), которая, в свою очередь, является основой всех приложений Microsoft .NET, разработчики могут писать код для ASP.NET на языках программирования, входящих в комплект .NET Framework: C#, Visual Basic .NET, J# и Jscript .NET [6]. Использование этих языков позволяет применять непосредственно в веб-приложениях возможности объектно-ориентированного программирования. Благодаря применению серверных объектов, автоматически генерирующих код элементов управления HTML, существенно

сокращается объем кода. По сравнению со скриптовыми технологиями ASP.NET имеет преимущество в скорости, так как при первом обращении код компилируется и помещается в специальный кеш, который в дальнейшем только исполняется, не требуя затрат времени на парсинг, оптимизацию и т.п. [7].

2.3 OWIN

Характерной чертой классического ASP.NET веб-приложения является зависимость от System.Web. Эта сборка входит в .NET Framework и, по сути, представляет функциональность IIS (Internet Information Services) (Рисунок 2.1) [8].

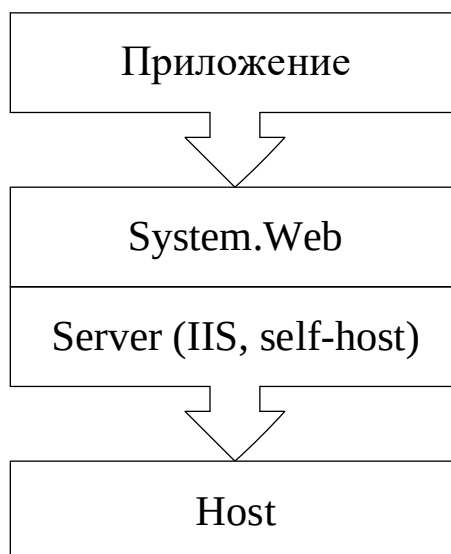


Рисунок 2.1 – Структура классического ASP.NET веб-приложения

Недостаток такой структуры – сильная связь веб-приложения с System.Web. Это означает, что перенести его на другой тип без изменений невозможно. System.Web – это часть .NET Framework, а значит ее новые версии появляются только с очередным обновлением самой платформы. С учетом текущих темпов развития веб-технологий это достаточно медленно.

Стало ясно, что нужна некая абстракция, представляющая собой сервер и позволяющая строить свой конвейер обработки запросов. Так появились OWIN и Katana [9].

OWIN (Open Web Interface for .NET) – это спецификация, определяющая интерфейс, который устраняет сильную связанность веб-приложения с конкретной реализацией сервера. Схематически это выглядит так (Рисунок 2.2):

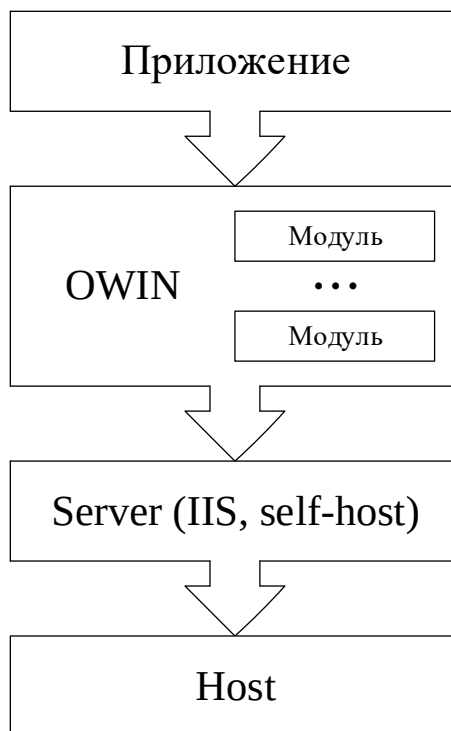


Рисунок 2.2 – Структура веб-приложения с использованием OWIN

Данная структура позволяет запускать приложение на любой платформе, поддерживающей OWIN, без изменений. Для работы оно использует готовые или созданные самим разработчиком модули. В свою очередь, спецификация задает интерфейс взаимодействия последних с сервером.

В момент запуска веб-приложения, разработчик конфигурирует нужные модули в конвейер обработки запросов. Таким образом используется только необходимая функциональность.

Модули и приложения, написанные для OWIN, способны работать на различных типах серверов. Так достигается гибкость решения. Это хорошо заметно на примере библиотек WebAPI и signalR, которые могут работать как под IIS, так и в отдельном приложении [9].

2.4 Entity Framework

Entity Framework представляет специальную объектно-ориентированную технологию на базе фреймворка .NET для работы с данными. Традиционные средства ADO.NET (набор классов, предоставляющих службы доступа к данным программистам, которые используют платформу .NET Framework [10]) позволяют создавать подключения, команды и прочие объекты для взаимодействия с базами данных. Entity Framework представляет собой более высокий уровень абстракции, позволяющий работать с данными независимо от типа хранилища и независимый от базы данных. На физическом уровне разработчик работает с таблицами, индексами, первичными и внешними ключами, на концептуальном уровне, который предлагает Entity Framework, разработчик оперирует с объектами [11].

Основной концепцией Entity Framework является понятие сущности (entity). Сущность представляет набор данных, ассоциированных с определенным объектом, поэтому данная технология предполагает работу не с таблицами, а с объектами и их наборами [11].

Отличительной чертой Entity Framework является использование запросов LINQ для выборки данных из базы. С помощью них разработчик может не только извлекать определенные строки, хранящие объекты, из базы данных, но и получать объекты, связанные различными ассоциативными связями.

Другим ключевым понятием является Entity Data Model. Эта модель сопоставляет классы сущностей с реальными таблицами в БД. Entity Data

Model состоит из трех уровней: концептуального, уровень хранилища и уровень сопоставления (маппинга). На концептуальном уровне происходит определение классов сущностей, используемых в приложении. Уровень хранилища определяет таблицы, столбцы, отношения между таблицами и типы данных, с которыми сопоставляется используемая база данных. Уровень сопоставления служит посредником между предыдущими двумя, определяя сопоставление между свойствами класса сущности и столбцами таблиц. Таким образом, разработчик может через классы, определенные в приложении, взаимодействовать с таблицами из базы данных.

Entity Framework предполагает три возможных способа взаимодействия с базой данных:

- Database first: Entity Framework создает набор классов, которые отражают модель конкретной базы данных.
- Model first: сначала разработчик создает модель базы данных, по которой затем Entity Framework создает реальную базу данных на сервере.
- Code first: разработчик создает класс модели данных, которые будут храниться в БД, а затем Entity Framework по этой модели генерирует базу данных и ее таблицы [12].

Для того, чтобы получить доступ к базе данных с помощью Entity Framework, нужно использовать поставщики данных, совместимые с Entity Framework. Поставщики данных (data providers) организуют взаимодействие между сущностными классами и системой управления базами данных (СУБД). Visual Studio по умолчанию поставляется поставщиком данных SqlClient, что позволяет получить доступ к большинству версий СУБД SQL Server: 2005, 2008, 2010, 2012, 2014. В настоящее время поставщики данных Entity Framework поддерживаются следующими СУБД: MySQL, Oracle, Progress, VistaDB, Devart, OpenLink, SQL Anywhere, Sybase, SQLite, Synergex, Firebird и PostgreSQL [13].

2.4.1 Подход Code-First

Code-First впервые появился в Entity Framework 4.1. Данный позволяет описать модель данных с использованием классов C# и взаимодействиями между классами.

Code-First автоматически определяет, что поле является первичным ключом, если свойство класса с моделью данных называется "ID" либо оканчивается на "ID" (регистр не учитывается). Если тип свойства первичного ключа является числовым или имеет тип GUID, поле будет настроено как столбец идентификаторов в таблице базы данных.

В Entity Framework связи обеспечивают отношения между двумя типами сущностей. Каждый объект может иметь свойство навигации для каждой связи, в которой он участвует. Свойства навигации позволяют перемещаться и управлять отношениями в обоих направлениях, возвращая либо ссылочный объект, либо коллекцию.

Если внешний ключ зависимого объекта не является нулевым, то Code First устанавливает каскадное удаление в отношении. Если внешний ключ на зависимом объекте является нулевым, Code First не устанавливает каскадное удаление в отношении, и когда основной объект удаляется, внешний ключ будет иметь значение null [13].

2.4.2 Библиотека AutoMapper

Открытая библиотека AutoMapper представляет собой соответствующее соглашению средство объектно-объектного преобразования. Она работает путем преобразования входных объектов одного типа в выходной объект другого типа. Для установления соглашений между двумя типами объектов, которые требуется сопоставить, требуется почти нулевая конфигурация [14].

AutoMapper обеспечивает простую конфигурацию типов, а также простое тестирование сопоставлений. Automapper сам соотносит поля по имени и типу. Сопоставление может происходить во многих местах

приложения, но главным образом в границах между слоями, например, между уровнями пользовательского интерфейса и таблицей базы данных [14].

Возможности библиотеки:

- Flattening – преобразование сложной объектной модели в более простую структуру.
- Unflattening – преобразование простой модели в сложную.
- Проецирование – получение из исходного объекта его проекции на более сложную модель данных.
- Списки и массивы.
- Вложенные сопоставления.
- Условные сопоставления.

Помимо перечисленных, AutoMapper предоставляет обширный круг других, не менее значимых возможностей [15].

2.5 Microsoft SQL Server

Microsoft SQL Server является надежной базой данных для любых целей, может продолжать расширяться по мере наполнения информацией, без заметного уменьшения быстродействия операций с записями в многопользовательском режиме.

Обеспечивается максимальная безопасность. За счет интеграции сетевой безопасности с сервером безопасности данные пользователей защищены от несанкционированного доступа. Пользователи могут иметь ограниченный доступ к записи данных, тем самым защищая их от модификации или поиска, указав доступ на уровне пользовательских привилегий. Кроме того, с данными, хранящимися на отдельном сервере, сервер работает как шлюз, который ограничивает несанкционированный доступ.

Microsoft SQL Server обрабатывает запросы от пользователей и отправляет пользователю только результаты запроса – таким образом по сети передается минимальная информация.

Техническое обслуживание SQL Server очень простое и не требует больших знаний. Возможны изменения в структуре данных, а также резервное копирование во время работы сервера без остановки [16].

Важнейшие характеристики данной СУБД – это:

- Простота администрирования.
- Возможность подключения к Web.
- Быстродействие и функциональные возможности механизма сервера СУБД.
- Наличие средств удаленного доступа.

В комплект средств административного управления Microsoft SQL Server входит целый набор специальных компонентов и средств автоматической настройки параметров конфигурации. Также данная база данных оснащена средствами тиражирования, позволяющими синхронизировать данные ПК с информацией БД и наоборот [17].

2.6 Авторизация на основе токенов

Аутентификация и авторизация в Web API имеет некоторые особенности в отличие от ASP.NET MVC. Здесь фактически нам доступны три вида аутентификации: стандартная через куки, через внешние сервисы и аутентификация с помощью токена.

В то же время Web API и ASP.NET MVC имеют ряд общих моментов.

Так, для ограничения доступа можно применять встроенную реализацию атрибута авторизации – `AuthorizeAttribute`, который находится в пространстве имен `System.Web.Http`. Он одновременно проверяет, аутентифицирован ли пользователь в системе, и также проверяет права пользователя на доступ к данному ресурсу (если данный ресурс подразумевает доступ только для определенных ролей или пользователей). Если пользователь неаутентифицирован, то система посылает в ответ клиенту статусный код 401 (Unauthorized) [18].

Авторизация на основе токенов состоит из нескольких компонентов:

- Клиент, который обращается к веб-сервису. Может представлять собой веб-браузер, мобильное приложение, десктопное приложение.
- Веб-сервис, к ресурсу которого обращается клиент.
- Токен доступа (access token), наличие которого дает доступ к ресурсам веб-сервиса.
- Bearer-токен - специальный вид токена доступа.
- Сервер авторизации, который выдает токены доступа клиенту.

Весь процесс аутентификации выглядит следующим образом (Рисунок 2.3):

- 1) Пользователь вводит данные авторизации (логин, пароль) и нажимает на кнопку отправки.
- 2) Клиент (веб-браузер) отправляет данные серверу авторизации.
- 3) Сервер авторизации аутентифицирует пользователя и возвращает токен доступа.
- 4) Для доступа к ресурсу веб-сервиса клиент добавляет в запрос ранее полученный токен доступа.

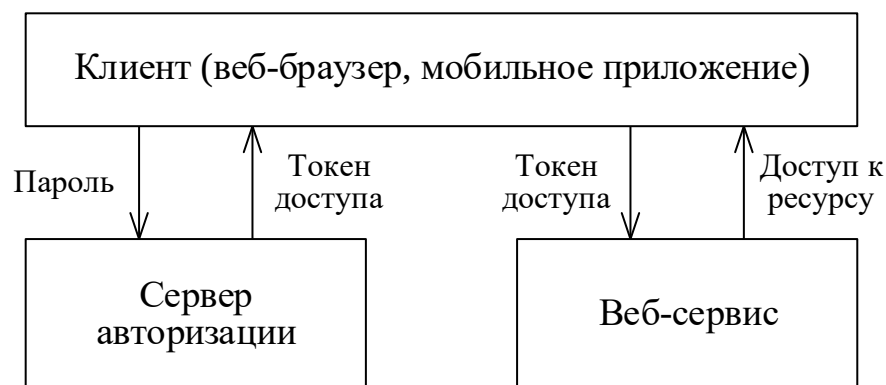


Рисунок 2.3 – Получение токена доступа в OAuth2

Хотя выше сервер авторизации и веб-сервис, но в реальности они могут быть объединены, как это и происходит в приложении Web API (Рисунок 2.4):

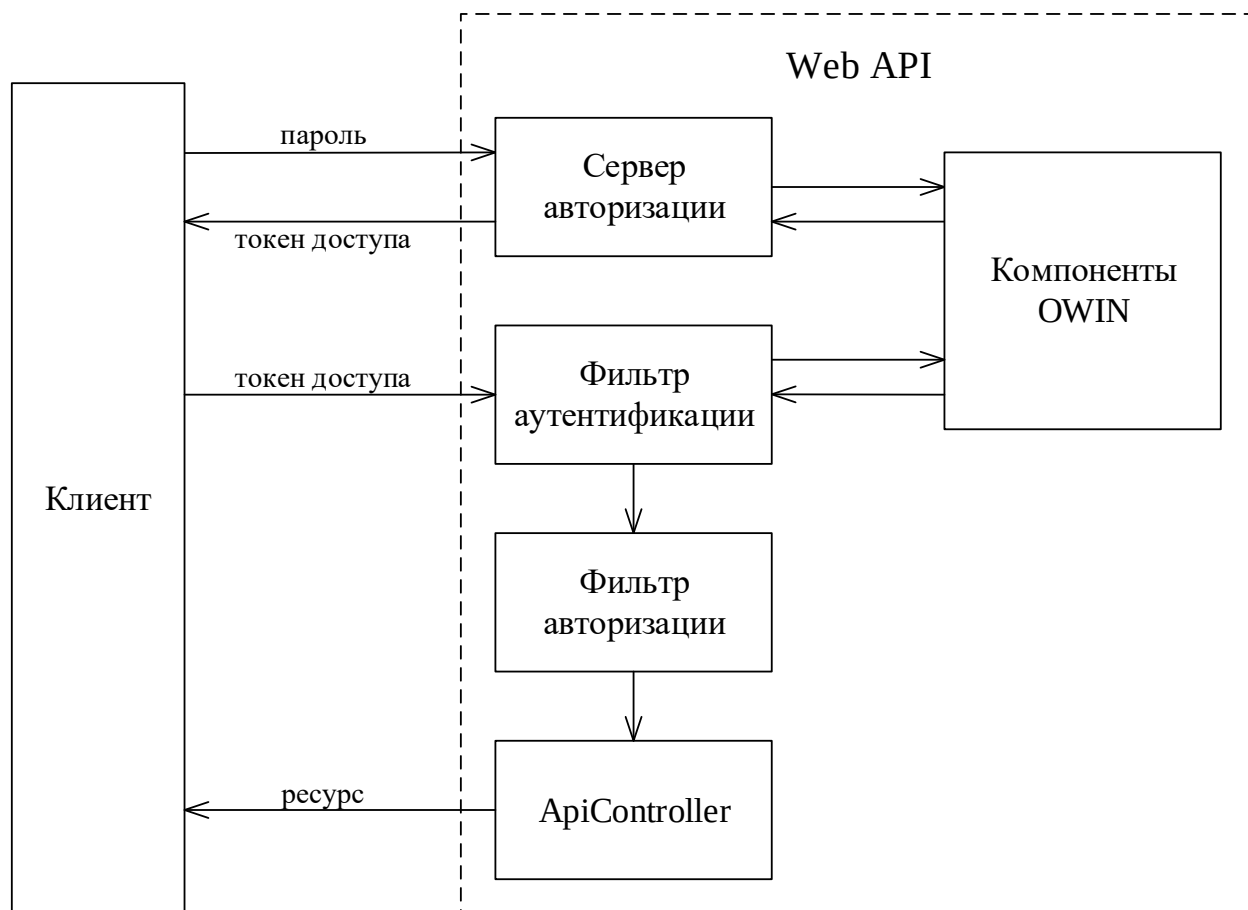


Рисунок 2.4 – Получение токена доступа в ASP.NET Web API

Web API применяет спецификацию OAuth2, которая описывает механизм аутентификации на основе токенов. Конкретную реализацию этой спецификации представляют компоненты OWIN, благодаря которым и производит генерация токенов, их выдача клиенту и их дальнейшая валидация. Поэтому при создании приложения Web API нам не надо самим реализовывать сервер авторизации, все эти детали, а достаточно взять готовый механизм, который предоставляет ASP.NET [18].

OAuth – открытый протокол авторизации. Данный протокол, не передавая логин и пароль пользователя, позволяет предоставить ограниченный доступ к защищенным данным пользователя стороннему серверу (веб-сайту, приложению и т.п.) [19]. Протокол начал разрабатываться в ноябре 2006 года, а последняя версия OAuth 1.0 была утверждена 4 декабря

2007 года. Последователем стал протокол OAuth 2.0, который появился в 2010 году, его последняя версия была опубликована в октябре 2012 года.

Одна из проблем аутентификации и информационной безопасности заключается в том, что любой пользователь может иметь несколько разных учетных записей на различных независимых сервисах (например, на Google, Facebook и др.). С помощью технологии OAuth пользователь может авторизоваться на других сервисах, и при этом ему не потребуется раскрывать свой основной пароль – в этом состоит главная особенность применения OAuth. Например, пользователь, который хочет предоставить социальной сети доступ к контактам своей почты, не должен передавать сервису свой пароль от электронного ящика. Вместо этого он проходит авторизацию непосредственно в почтовом сервисе, который предоставляет сервису социальной сети права доступа к адресной книге [20].

При использовании OAuth-авторизации данные пользователя не передаются напрямую в приложение, а значит:

- Несанкционированный доступ к личным данным пользователя невозможен: приложение не владеет ни логином, ни паролем пользователя и может выполнять только те действия с данными, которые разрешил пользователь, следовательно, пользователь может доверять приложению [20].
- При разработке приложения с использованием протокола OAuth не нужно заботиться об обеспечении конфиденциальности логина и пароля пользователя. Логин и пароль не передаются приложению и не могут попасть в руки злоумышленников [21].

В OAuth 2.0 используются полномочия трех видов: учетные данные клиента, временные учетные данные и токены:

- Для проверки подлинности клиента используются его учетные данные.
- Вместо имени и пароля владельца ресурса используется токен. Владелец ресурса, не раскрывая своих учетных данных, разрешает

серверу выдавать клиенту токен, который является специальным классом учетных данных, предоставляющим клиенту некоторые ограниченные возможности. Токен состоит из цифровой подписи, уникального и криптографически-стойкого набора букв и цифр и из ключа для защиты токена от использования третьими лицами. Токен ограничен по полномочиям и продолжительности и в любой момент может быть отозван владельцем ресурса, не затрагивая при этом токены других клиентов.

- Временные учетные данные используются на стадии запроса авторизации. В целях удовлетворения различного рода клиентов (веб-интерфейсных, мобильных и т.п.), временные полномочия предлагают дополнительную гибкость и безопасность [22].

2.7 Формат JSON

JSON (JavaScript Object Notation) – простой формат обмена данными, удобный для чтения и написания как человеком, так и компьютером. Он основан на подмножестве языка программирования JavaScript, определенного в стандарте ECMA-262 3rd Edition – December 1999. JSON – текстовый формат, полностью независимый от языка реализации, но он использует соглашения, знакомые программистам С-подобных языков, таких как С, С++, С#, Java, JavaScript, Perl, Python и многих других. Эти свойства делают JSON идеальным языком обмена данными [23].

За счет своей краткости по сравнению с XML, формат JSON может быть более подходящим для сериализации сложных структур. В веб-приложениях он уместен в задачах обмена данными как между браузером и сервером, так и между самими серверами.

JSON-текст представляет собой (в закодированном виде) одну из двух структур:

- Коллекция пар ключ/значение. В разных языках, эта концепция реализована как объект, структура, словарь, именованный список или ассоциативный массив.
- Упорядоченный список значений. В большинстве языков это реализовано как массив, список или последовательность.

Это универсальные структуры данных. Почти все современные языки программирования поддерживают их в какой-либо форме. Логично предположить, что формат данных, независимый от языка программирования, должен быть основан на этих структурах [23].

В качестве значений в JSON могут быть использованы:

- Объект – неупорядоченный набор пар ключ/значение. Объект начинается с открывающей фигурной скобки и заканчивается закрывающей фигурной скобкой. Каждое имя сопровождается двоеточием, пары ключ/значение разделяются запятой.
- Массив – упорядоченная коллекция значений. Массив начинается с открывающей квадратной скобки и заканчивается закрывающей квадратной скобкой, значения разделены запятой.
- Строка – коллекция нуля или больше символов Unicode, заключенная в двойные кавычки, символы могут быть указаны с использованием ескапе-последовательностей, начинающихся с обратной косой черты в качестве символа экранирования. Символ представляется как односимвольная строка. Похожий синтаксис используется в C и Java.

Число представляется так же, как в C или Java, кроме того, что используется только десятичная система счисления. Пробелы могут использоваться между любыми лексемами [23].

2.8 Среда разработки Microsoft Visual Studio

Microsoft Visual Studio – линейка продуктов компании Майкрософт, включающих интегрированную среду разработки программного обеспечения

и ряд других инструментальных средств. Данные продукты позволяют разрабатывать как консольные приложения, так и приложения с графическим интерфейсом.

Visual Studio включает в себя редактор исходного кода с поддержкой технологии IntelliSense и возможностью простейшего рефакторинга кода. Встроенный отладчик может работать как отладчик уровня исходного кода, так и как отладчик машинного уровня. Остальные встраиваемые инструменты включают в себя редактор форм для упрощения создания графического интерфейса приложения, веб-редактор, дизайнер классов и дизайнер схемы базы данных. Visual Studio позволяет создавать и подключать сторонние дополнения (плагины) для расширения функциональности практически на каждом уровне, включая добавление поддержки систем контроля версий исходного кода, добавление новых наборов инструментов [24].

2.9 Internet Information Services

IIS (Internet Information Services) для Windows – гибкий, безопасный и легкоуправляемый веб-сервер для публикации любого приложения в сети. IIS может все, начиная от трансляции потокового видео до публикации бизнес-приложений. Он легко масштабируется, и, благодаря открытой архитектуре, готов для реализации большинства задач [25].

Основным компонентом IIS является веб-сервер, который позволяет размещать в Интернете сайты. IIS поддерживает протоколы HTTP, HTTPS, FTP, POP3, SMTP, NNTP [8].

Основным компонентом IIS является веб-сервер – служба WWW, которая предоставляет клиентам доступ к сайтам по протоколам HTTP и, если произведена настройка, HTTPS.

Один сервер IIS может обслуживать несколько сайтов. Каждый сайт имеет следующие атрибуты:

- IP-адрес сайта.

- TCP-порт, на котором служба WWW ожидает подключений к данному сайту.
- Заголовок узла (Host header name) – значение заголовка Host запроса HTTP, указывающее обычно DNS-имя сайта.

Таким образом один сервер с одним IP-адресом может обслуживать на одном TCP-порту несколько сайтов. Для этого необходимо создать несколько DNS-записей, указывающих на IP-адрес сервера, и различать сайты по заголовкам узла.

Для каждого сайта указывается домашний каталог – каталог в файловой системе сервера, соответствующий «корню» сайта.

2.10 Advanced REST Client

Advanced REST client – это вспомогательная программа сетевых разработчиков для создания и тестирования пользовательских запросов HTTP [26]. Посредством этого инструмента, пользователь получает возможность вносить данные в формы и делать POST, PUT, GET, DELETE запросы, а также к RESTFUL сервисам [27].

Функциональные возможности:

- Интегрировано с Google Drive.
- Имеется собственный вторичный сервер для хранения и обмена данными между сотрудниками.
- Удобные заголовки HTTP и редактор Payload (полезная загрузка) благодаря CodeMirror.
- Составление HTTP запроса (посредством XmlHttpRequest 2-го уровня).
- Отладка сокета (посредством сетевого сокета API).
- Обозреватель ответов JSON.
- Обозреватель ответов XML.

- Установка пользовательских заголовков – даже без поддержки объекта XMLHttpRequest.
- Помощь с заполнением HTTP заголовков (подсказка + окончание кода).
- Добавление в список заголовков в виде необработанных данных или посредством формы.
- Создание тела POST или PUT посредством необработанного ввода или отправки файла (-ов) с запросом.
- Установка кодировки пользовательской формы.
- Запоминание последнего запроса (сохранение текущего состояния формы и восстановление при загрузке).
- Сохранение и открытие сохраненных форм запроса.
- Поддержка истории.
- Импорт/экспорт данных [26].

В Advanced Rest Client можно задавать множество параметров, как в стоковом режиме, так и в графическом режиме, по каждому параметру поля, а также выполнять большое количество различных запросов, к числу которых также относится работа с сокетами [27].

2.11 Выбранные средства для решения задачи

Исходя из вышеперечисленных рассуждений, проект может быть реализован с использованием следующих средств:

- Язык программирования C#.
- Платформа разработки веб-приложений ASP.NET.
- Спецификация OWIN.

- Entity Framework для разработки базы данных на основе подхода CodeFirst с использованием AutoMapper для связывания объектов базы данных с объектами приложения.
- Microsoft SQL Server в качестве СУБД.
- Авторизация на основе токенов с использованием протокола OAuth2.
- Формат JSON для передачи данных.
- Среда разработки – Microsoft Visual Studio.
- IIS для публикации сервиса.
- Advanced REST Client для тестирования приложения.

Указанные средства удовлетворяют всем требованиям и содержат весь набор функций, необходимый для решения задачи.

3 РАЗРАБОТКА БАЗЫ ДАННЫХ

3.1 Анализ предметной области

Главным условием успешной разработки системы подбора персонала можно считать ясное и однозначное понимание ее назначения и функций. Цель любой системы подбора персонала – отсев несоответствующих и отбор наиболее подходящих по определенным критериям сотрудников [28].

Если в организации существует потребность в полноценной и эффективной системе отбора персонала, к процессу ее создания необходимо подходить комплексно. Первым шагом после определения потребностей компании будет структурирование этапов создания системы персонала:

- **Постановка целей.** Основной целью системы подбора персонала является отбор лучших кандидатов для определенной должности и отсев неподходящих по каким-то параметрам сотрудников.
- **Структурирование должностей компании.** Решающими факторами при структурировании будут: сложность оценки требуемых для должности компетенций, навыков, умений, способностей.
- **Определение исходных данных.** Исходными данными, используемыми системой подбора персонала, являются: план потребностей в персонале, принципы отбора, требуемые компетенции, возможность приобретения необходимых знаний и умений, список основных заинтересованных в процессе отбора лиц и степень их влияния.
- **Разработка механизма отбора персонала.** Механизм отбора кадров включает приемы, методы и инструменты, применяющиеся для определения соответствия кандидата определенной должности (наличие необходимых компетенций, согласно профилю должности).
- **Адаптация системы подбора персонала.** Важно не только иметь представление о том, как создать систему подбора персонала, но и

понимать, каким образом она будет взаимодействовать с имеющимися подсистемами.

- Определение ответственных и исполнителей. Необходимо определить ответственных за действие системы сотрудников и организовать внутреннее или внешнее обучение исполнителей.
- Формулировка принципов обратной связи. Эффективная система подбора персонала должна обладать регулярными, обязательными и неформальными процедурами обратной связи.
- Корректировка и саморазвитие системы. Обработка и анализ данных обратной связи, определение необходимых для корректировки системы действий.

В процессе отбора кандидат получает более полную информацию об ожиданиях компании в отношении его компетенций. Предъявляемые компанией жесткие требования отбора способствуют общему росту профессионального уровня кандидатов [28].

3.2 Проектирование базы данных учетных данных сотрудников

В компоненте управления учетными записями пользователей используется система авторизации и аутентификации ASP.NET Identity с типом аутентификации Individual User Accounts. Данная настройка производится при создании приложения в среде Microsoft Visual Studio и уже по умолчанию имеет всю необходимую для авторизации инфраструктуру: модели, контроллеры и представления. В проект автоматически подключаются библиотеки, которые содержат классы Entity Framework, необходимые для авторизации и аутентификации.

3.3 Проектирование базы данных хранения информации о проектах и профессиональных данных сотрудников

Все будущие таблицы с характеристиками их полей и связей описываются непосредственно в коде программы в соответствующих классах. Для того, чтобы корректно составить структуру каждого класса, необходимо составить диаграмму базы данных.

Сначала необходимо выделить основные сущности для будущей базы данных:

- 1) сотрудники (Employee);
- 2) специализации (Specializations);
- 3) должности (JobPositions);
- 4) проекты (Projects),

а затем определить связи между ними и, при необходимости, добавить промежуточные таблицы и дополнительные сущности.

Каждая таблица представляет собой сущность, в которой хранится информация определенного типа. Для наглядного представления зависимостей между таблицами базы данных необходимо составить диаграмму «сущность-связь», определив отношения между сущностями.

3.3.1 Сущность «Сотрудники»

Информация о сотрудниках (фамилия, имя, номер телефона и т.д.) хранится в базе личных данных сотрудников. Для установления связей в проектируемой базе данных используется только GUID сотрудника, получаемый из данных авторизации либо по запросу администратора. Определим характеристики сущности «Сотрудники» (Employee).

- 1) Каждый сотрудник может иметь несколько специализаций, следовательно, одному сотруднику соответствует много специализаций.
- 2) Сотрудник может иметь различные должности. Получаем связь один ко многим.

- 3) Сотрудник может отправить заявку для участия в проекте администратору для дальнейшего рассмотрения. Для этого выделяется дополнительная сущность «Заявки» (ProjectApplications). Один сотрудник может отправить несколько заявок.

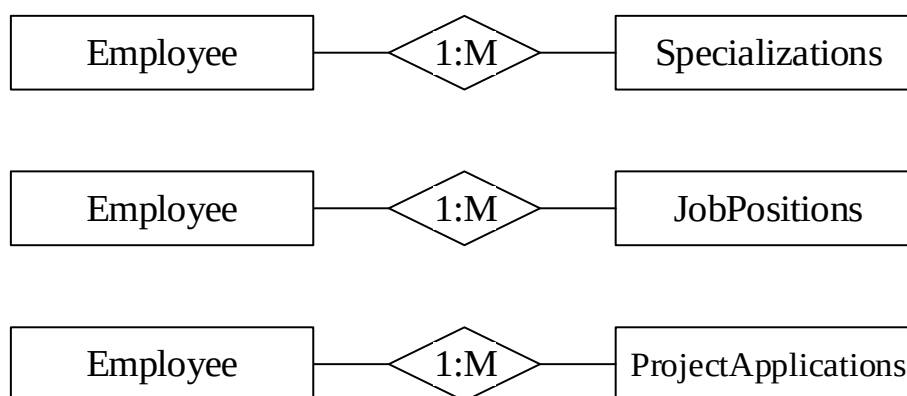


Рисунок 3.1 – Диаграммы связей для сущности «Сотрудник»

На рисунке 3.1 представлены полученные диаграммы связей для сущности «Сотрудник».

3.3.2 Сущность «Специализации»

Сущность «Специализации» (Specializations) представляет собой список компетенций (знаний, умений, навыков и т.д.), необходимых для характеристики сотрудника или требований проекта и является справочником.

3.3.3 Сущность «Должности»

Сущность «Должности» (JobPositions) по аналогии с сущностью «Специализации» является справочником и содержит список должностей.

3.3.4 Сущность «Проекты»

За описание проектов отвечает сущность «Проекты» (Projects).
Определим характеристики сущности.

- 1) Для реализации проекта требуется владение определенными специализациями.
- 2) Проект может быть разбит на этапы, для этого необходимо выделить еще одну сущность «Этапы проекта» (ProjectStages).
- 3) В проекте могут принимать участие несколько сотрудников. Для этого выделим дополнительную сущность «Участники проекта» (ProjectMembers).

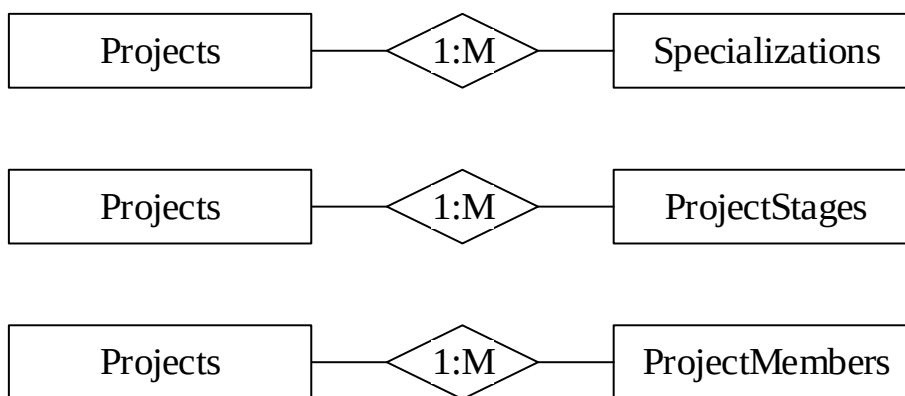


Рисунок 3.2 – Диаграммы связей для сущности «Проекты»

На рисунке 3.2 представлены полученные диаграммы связей для сущности «Проекты».

3.3.5 Сущность «Участники проекта»

Сущность «Участники проекта» (ProjectMembers) является составной сущностью. Определим ее характеристики:

- 1) Каждый из участников проекта – сотрудник организации. Одна запись в таблице соответствует одному сотруднику.
- 2) Каждый участник занимает определенную должность.
- 3) Каждый участник владеет определенной специализацией.

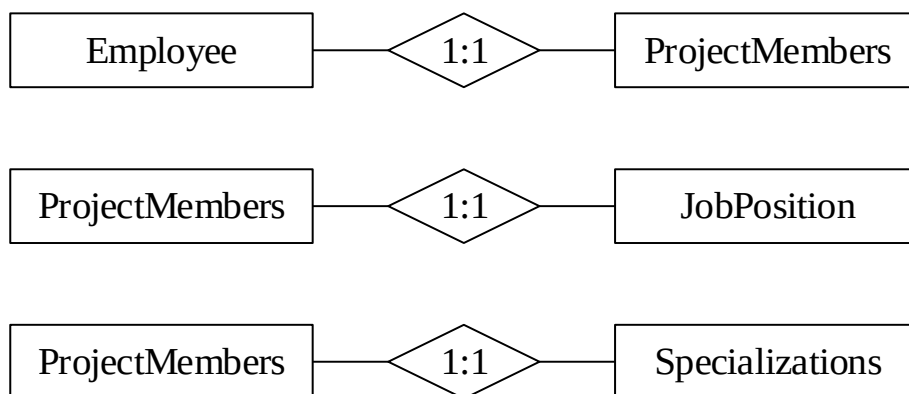


Рисунок 3.3 – Диаграммы связей для сущности «Участники проекта»

На рисунке 3.3 представлены полученные диаграммы связей для сущности «Участники проекта».

3.3.6 Сущность «Этапы проекта»

Каждый проект может быть разделен на этапы. Сущность «Этапы проекта» (ProjectStages) отражает эту информацию. Один проект содержит один или несколько этапов.

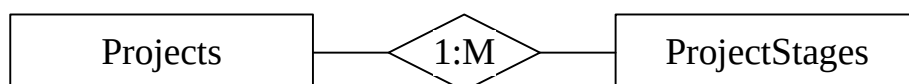


Рисунок 3.4 – Диаграмма связи между сущностями Projects – ProjectStages

На рисунке 3.4 представлена диаграмма связей для сущности «Этапы проекта».

3.3.7 Сущность «Заявки»

Сотрудник может отправить заявку для участия в проекте. За список заявок отвечает сущность «Заявки» (ProjectApplications).

- 1) Одна специализация из справочника специализаций может соответствовать нескольким заявкам.

- 2) Одна должность из справочника должностей может соответствовать нескольким заявкам.
- 3) Для одного проекта может быть составлено несколько заявок.

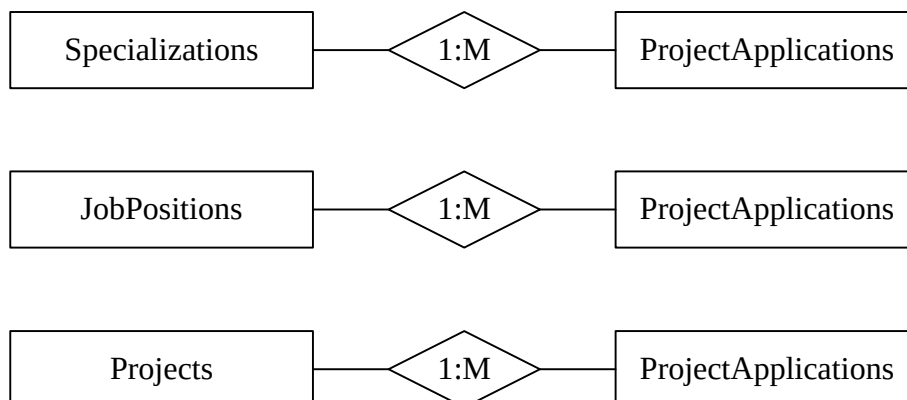


Рисунок 3.5 – Диаграммы связей для сущности «Заявки»

На рисунке 3.5 представлены полученные диаграммы связей для сущности «Заявки».

3.3.8 Диаграмма «сущность-связь»

На основе указанных отношений между сущностями можно составить диаграмму «сущность-связь» базы данных (Приложение Б.2).

Необходимо учитывать, что сущность Employee в данной базе данных состоит лишь из поля GUID, который приходит из базы личных данных пользователей. Поэтому для связей Employee-Specialization, Employee-JobPosition, Employee-Application и Employee-ProjectMembers достаточно составить промежуточные таблицы связей, где непосредственно указывать GUID сотрудника.

3.4 Разработка базы данных

При проектировании приложения используется подход Code First: сначала создаются классы моделей данных, а когда необходимо работать

непосредственно с базой данных, используются инструменты, которые проецируют структуру базы данных из созданной модели классов. К модели данных в коде можно вернуться в любое время и, как вариант, изменить ее. Выполненные изменения затем можно отразить в базе данных, используя все те же инструменты.

Так как компонент управления учетными записями пользователей использует систему ASP.NET Identity, никаких дополнительных классов в нем создавать не требуется.

Для второго компонента было принято решение разработать следующие классы моделей данных:

- DbEmployeeJobPosition.
- DbEmployeeSpecialization.
- DbJobPosition.
- DbProject.
- DbProjectApplication.
- DbProjectMember.
- DbProjectStage.
- DbSpecialization.

Каждый класс представляет собой сущность базы данных, при этом таблицы промежуточных связей создаются автоматически, при указании определенных зависимостей в разрабатываемых классах.

3.4.1 Класс DbSpecialization

Класс DbSpecialization является основой для таблицы базы данных, в которой хранятся сведения о списке специализаций (компетенций, знаний, умений), и содержит следующие поля:

- Id – уникальный идентификатор специализации.
- Name – наименование специализации.
- Description – описание специализации.

- Projects – связь со списком проектов, для решения которых необходима специализация.
- EmployeeSpecializations – связь со списком сотрудников, владеющих специализацией.
- ProjectApplication – связь с таблицей заявок, в которых указана данная специализация.

Структура класса DbSpecializations представлена в листинге 1.

Листинг 1. Структура класса DbSpecialization.cs

```
public class DbSpecialization
{
    public int Id { get; set; }
    public string Name { get; set; }
    public string Description { get; set; }
    public virtual ICollection<DbProject> Projects { get; set; }
    public virtual ICollection<DbEmployeeSpecialization>
    EmployeeSpecializations { get; set; }
    public virtual ICollection<DbProjectApplication>
    ProjectApplication { get; set; }
}
```

3.4.2 Класс DbProject

Класс DbProject отражает структуру для описания проекта (листинг 2).

Листинг 2. Структура класса DbProject.cs

```
public class DbProject
{
    public int Id { get; set; }
    public DateTime Created { get; set; }
    public string Title { get; set; }
    public string Description { get; set; }
    public string JiraLink { get; set; }
    public bool IsOpen { get; set; }
    public DateTime? StartDate { get; set; }
    public DateTime? EndDate { get; set; }
    public virtual ICollection<DbSpecialization> Specializations
    { get; set; }
    public virtual ICollection<DbProjectStage> ProjectStages {
    get; set; }
    public virtual ICollection<DbProjectMember> Members { get;
    set; }
}
```

Значение свойств класса:

- Id – уникальный идентификатор проекта.
- Created – дата создания записи о проекте.
- Title – название проекта.
- Description – описание проекта.
- JiraLink – ссылка на проект (например, в системе контроля версий).
- IsOpen – статус проекта (открыт/закрыт).
- StartDate – дата начала проекта.
- EndDate – дата окончания проекта.
- Specializations – связь со списком специализаций, необходимых для реализации проекта.
- ProjectStages – связь со списком этапов проекта.
- Members – связь со списком участников проекта.

3.4.3 Класс DbJobPosition

Класс DbJobPosition отражает структуру справочника для хранения должностей, определенных в организации (листинг 3).

Листинг 3. Структура класса DbJobPosition.cs

```
public class DbJobPosition
{
    public int Id { get; set; }
    public string Title { get; set; }
    public virtual ICollection<DbEmployeeJobPosition>
EmployeesJobPosition { get; set; }
    public virtual ICollection<DbProjectApplication>
ProjectApplication { get; set; }
}
```

Свойства класса обозначают следующее:

- Id – уникальный идентификатор должности.
- Title – наименование должности.
- EmployeesJobPosition – связь со списком сотрудников с данной должностью.

- ProjectApplication – связь с таблицей заявок, в которых указана данная должность.

3.4.4 Класс DbEmployeeJobPosition

Класс EmployeeJobPosition является структурой промежуточной таблицы базы данных, в которой хранятся связи сотрудников и их должностей:

- Id – уникальный идентификатор записи.
- Guid – статический уникальный 128-битный идентификатор сотрудника, приходит из в.
- JobPositionId – идентификатор должности.
- JobPosition – связь с должностью.

Листинг 4. Структура класса DbEmployeeJobPosition.cs

```
public class DbEmployeeJobPosition
{
    public int Id { get; set; }
    public string Guid { get; set; }
    public int JobPositionId { get; set; }
    public virtual DbJobPosition JobPosition { get; set; }
}
```

В листинге 4 представлена структура класса DbEmployeeJobPosition.

3.4.5 Класс DbEmployeeSpecialization

Класс DbEmployeeSpecialization описывает таблицу, связывающую сотрудников со специализациями.

Листинг 5. Структура класса DbEmployeeSpecialization.cs

```
public class DbEmployeeSpecialization
{
    public int Id { get; set; }
    public string Guid { get; set; }
    public int Level { get; set; }
    public int SpecializationId { get; set; }
    public virtual DbSpecialization Specialization { get; set; }
}
```

В листинге 5 отражена структура класса DbEmployeeSpecialization. Класс содержит свойства:

- Id – уникальный идентификатор записи.
- Guid – статический уникальный 128-битный идентификатор сотрудника.
- Level – уровень владения навыком.
- SpecializationId – идентификатор специализации.
- Specialization – связь со специализацией.

3.4.6 Класс DbProjectMember

Класс DbProjectMember отражает структуру таблицы, в которой хранится информация об участниках проекта (сам сотрудник, его специализация на проекте и должность). Содержит свойства:

- Id – уникальный идентификатор записи.
- ProjectId – идентификатор проекта.
- Project – связь с проектом, к которому относится участник.
- Guid – статический уникальный 128-битный идентификатор сотрудника.
- Specializations – связь со списком специализаций специализаций.
- JobPositions – связь со списком должностей.

Более подробно структура данного класса описана в листинге 6.

Листинг 6. Структура класса DbProjectMember.cs

```
public class DbProjectMember
{
    public int Id { get; set; }
    public int ProjectId { get; set; }
    public virtual DbProject Project { get; set; }
    public string Guid { get; set; }
    public virtual ICollection<DbSpecialization> Specializations
{ get; set; }
    public virtual ICollection<DbJobPosition> JobPositions {get;
set; }
}
```

3.4.7 Класс DbProjectStage

Класс DbProjectStage отражает модель данных для описания этапов проекта (листинг 7). Содержит свойства:

- Id – уникальный идентификатор записи.
- ProjectId – идентификатор проекта.
- Title – название этапа.
- Description – описание.
- Project – связь с проектом.

Листинг 7. Структура класса DbProjectStage.cs

```
public class DbProjectStage
{
    public int Id { get; set; }
    public int ProjectId { get; set; }
    public string Title { get; set; }
    public string Description { get; set; }
    public virtual DbProject Project { get; set; }
}
```

3.4.8 Класс DbProjectApplications

Структура класса, необходимого для описания заявок проекта, показана в листинге 8.

Листинг 8. Структура класса DbProjectApplications.cs

```
public class DbProjectApplication
{
    public int Id { get; set; }
    public string GuId { get; set; }
    public int ProjectId { get; set; }
    public int JobPositionId { get; set; }
    public int SpecializationId { get; set; }
    public virtual DbJobPosition JobPosition { get; set; }
    public virtual DbSpecialization Specialization { get; set; }
}
```

Класс DbProjectApplications содержит свойства:

- Id – уникальный идентификатор заявки.

- Guid – статический уникальный 128-битный идентификатор сотрудника.
- ProjectId – идентификатор проекта, к которому относится заявка.
- JobPositionId – идентификатор должности, указанной в заявке.
- SpecializationId – идентификатор специализации, указанной в заявке.
- JobPosition – связь со списком должностей.
- Specialization – связь со списком специализаций.

3.4.9 Диаграмма классов базы данных

На основе указанных отношений между сущностями можно составить диаграмму классов базы данных.

Диаграмма, показанная в Приложении Б.3, описывает все связи между таблицами, на основе полей соответствующих таблиц.

4 РАЗРАБОТКА СИСТЕМЫ

На данном этапе выполнения работы требуется разработать два компонента системы – компонент управления учетными записями пользователей системы (TeamHunterIdentity) и компонент обработки запросов пользователей (TeamHunter).

Компонент управления учетными записями пользователей связан со своей собственной базой данных. С помощью механизма авторизации и аутентификации он идентифицирует пользователя и предоставляет ему те или иные полномочия для работы с другим компонентом системы.

Второй компонент является RESTful веб-сервисом и, по аналогии с первым, имеет свою базу данных. RESTful Web Service – это веб-сервис, написанный на основании структуры REST. REST (REpresentational State Transfer) – это стиль архитектуры программного обеспечения для распределенных систем, таких как World Wide Web, который, как правило, используется для построения веб-служб.

Управление информацией веб-сервиса – целиком и полностью основывается на протоколе передачи данных HTTP. Для HTTP действие над данными задается с помощью методов: GET (получить), PUT (добавить, заменить), POST (добавить, изменить, удалить), DELETE (удалить). Самое главное достоинство веб-сервиса в том, что работать с ним может любая система, будь то веб-сайт, десктопное или мобильное приложение, и др. Связано это с тем, что методы выполнения запросов HTTP присутствуют почти везде.

Оба компонента изолированы друг от друга и имеют разные домены. Это необходимо, чтобы гарантировать, что код, выполняемый одним приложением, не сможет каким-либо образом нарушить работу другого. Для обмена данными между компонентами используются ключи шифрования и расшифровки, что гарантирует целостность системы и подлинность

обмениваемой информацией. На рисунке 4.1 приведена диаграмма развертывания.

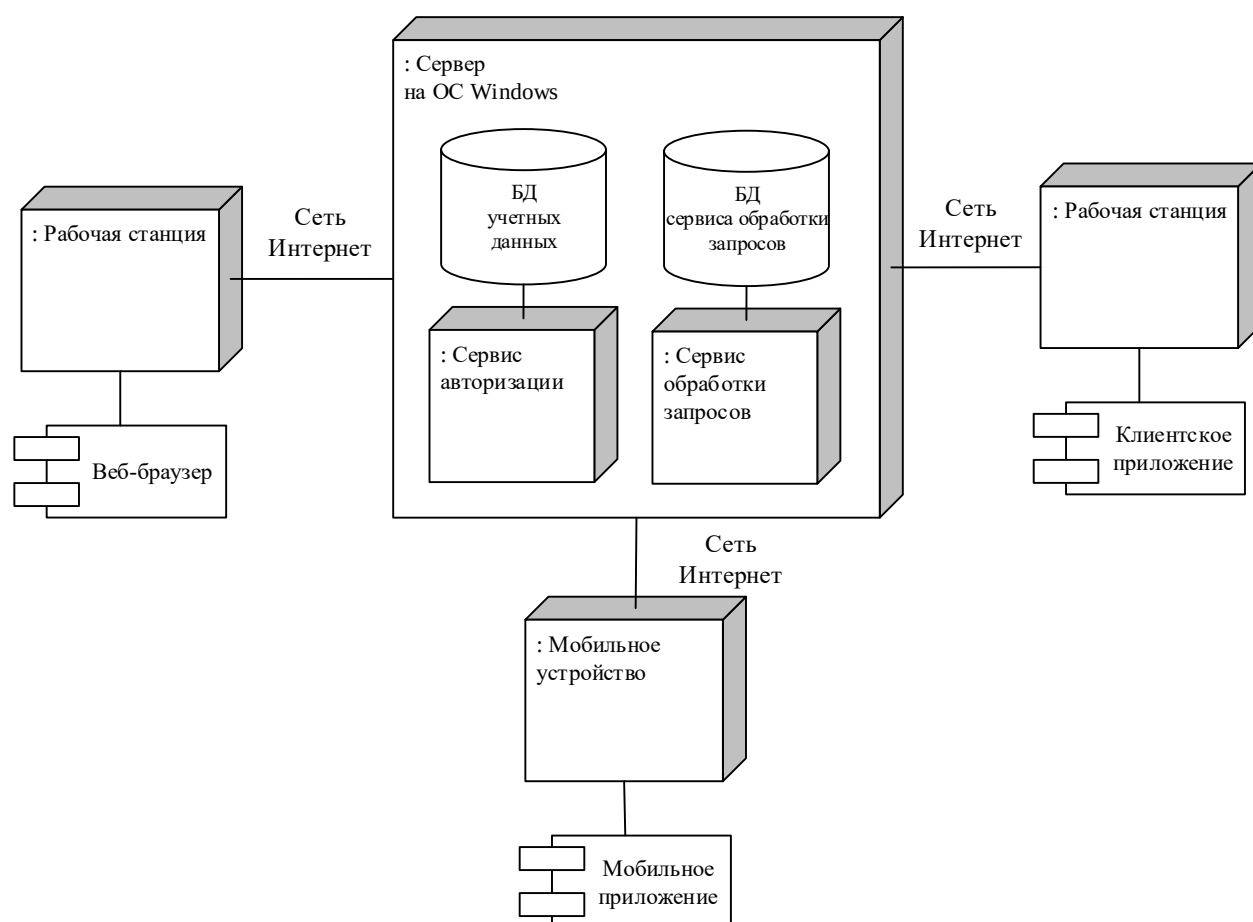


Рисунок 4.1 – Диаграмма развертывания

4.1 Разработка компонентов системы

Система состоит из двух компонентов – компонента управления учетными записями пользователей системы и компонента обработки запросов пользователей.

4.1.1 Компонент управления учетными записями пользователей

Данный компонент позволяет зарегистрироваться в системе, авторизоваться или выйти из нее. Помимо этих возможностей,

администратор может удалить сотрудника из системы (например, при увольнении сотрудника из компании).

При регистрации пользователь указывает следующую информацию:

- e-mail;
- логин;
- фамилия;
- имя;
- номер телефона;
- пароль;
- подтверждение пароля.

После успешной регистрации данные добавляются в соответствующую базу, а пользователю присваивается роль.

Для входа в систему пользователю необходимо пройти процедуру аутентификации и авторизации: если введенные логи и пароль соответствуют данным, хранящимся в базе, в ответ приходит cookie с токеном, который используется для доступа к определенным сервисам (зависит от роли).

При выходе из системы с устройства пользователя удаляется cookie, следовательно, отсутствует токен доступа и дальнейшие действия с системой невозможны.

4.1.2 Компонент обработки запросов пользователей

Другой компонент системы представляет собой совокупность нескольких сервисов, сгруппированных по своему назначению. К ним относятся:

- Сервис для работы с персоналом.
- Сервис для работы со списком должностей.
- Сервис для работы с проектами.
- Сервис для работы с участниками проекта.
- Сервис для работы с этапами проекта.
- Сервис для работы со списком специализаций.

Все перечисленные сервисы обращаются к своей базе данных. Функции, которые они предоставляют распределены по уровням доступа: для администраторов и обычных пользователей.

Доступ к каждому сервису осуществляется путем отправки HTTP-запроса на соответствующий адрес. Все запросы пользователей и ответы на них отправляются в формате JSON.

Отправленные пользователем данные подвергаются процедуре валидации на сервере. Если данные прошли проверку, то выполняется запрос, если нет – пользователю сообщается о некорректном формате данных.

Возможности пользователей системы более подробно представлены на диаграммах прецедентов (Приложение Б.4, Б.5).

Объекты для взаимодействия с базой данных и объекты, которые передаются между слоями приложения и используются для отображения пользователю информации, принадлежат разным классам.

Рассмотрим на примере классов `DbJobPosition` и `JobPositionDto`: класс `DbJobPosition` связан с базой данных, класс `JobPositionDto` используется для обмена информацией в приложении. Структура классов представлена в листинге 9.

Листинг 9. Структура классов `DbJobPosition` и `JobPositionDto`

```
public class DbJobPosition
{
    public int Id { get; set; }
    public string Title { get; set; }
    public virtual ICollection<DbEmployeeJobPosition>
EmployeesJobPosition { get; set; }
    public virtual ICollection<DbProjectApplication>
ProjectApplication { get; set; }
}

public class JobPositionDto
{
    public int Id { get; set; }
    public string Title { get; set; }
}
```

Оба класса описывают объект сущности «Должность». Класс `JobPositionDto` содержит описание объекта и доступен для просмотра или редактирования пользователю. При взаимодействии с базой данных (запись или получение информации) используется класс `DbJobPosition`. Как видно из листинга 9, в нем описывается структура соответствующей таблицы базы данных для объекта, а также связи с другими таблицами базы. Данный класс используется только для обращения к базе данных и никак не связан с клиентом.

Пользователь не может напрямую взаимодействовать с базой данных – ему доступны только объекты класса `JobPositionDto`. При внесении изменений пользователем, объект отправляется на сервер, где с помощью библиотеки `AutoMapper` данные одного класса сопоставляются с данными другого (что гарантирует идентичность соответствующих полей) и, если введенная информация корректна, объект записывается в базу.

Данная схема обмена информацией между клиентом и базой данных гарантирует надежность системы и позволяет снизить нагрузку на базу данных.

5 ВЗАИМОДЕЙСТВИЕ С БАЗАМИ ДАННЫХ

5.1 Подключение к базам данных

Для подключения к базе данных через приложения используется объект `System.Data.SqlClient`. Объект `System.Data.SqlClient` предоставляет доступ к версиям SQL Server и инкапсулирует протоколы для конкретных баз данных, а также включает средство синтаксического анализа потока табличных данных, предназначенное для непосредственного взаимодействия с SQL Server [29].

Связь с базой данных в приложении осуществляется через строку подключения, которая указывается в файле `Web.config`. Компоненты системы используют разные базы данных, а значит будут иметь различные строки подключения.

Строка подключения для компонента управления учетными записями пользователей выглядит следующим образом (листинг 10):

Листинг 10. Строка подключения к базе данных

```
<connectionStrings>
  <add name = "TeamHunterIdentityDb" connectionString = "Data
Source      =      DESKTOP-E15QV10\DB;      Initial      Catalog      =
TeamHunterIdentityDb; Integrated Security = True; Connection
Timeout = 30" providerName="System.Data.SqlClient" />
</connectionStrings>
```

Здесь `name` – имя подключения, `Data Source` – имя сервера базы данных, `Initial Catalog` – имя базы данных.

Строка подключения другого компонента представлена в листинге 11.

Листинг 11. Строка подключения к облачному серверу

```
<connectionStrings>
  <add name = "DefaultConnection" connectionString = "Data
Source = DESKTOP-E15QV10\DB; Initial Catalog = teamhunterDB;
Integrated Security = True; Connection Timeout = 30" providerName
= "System.Data.SqlClient"/>
```

```

<add name= "TeamHunterIdentityDb" connectionString = "Data
Source      =      DESKTOP-E15QV10\DB;      Initial      Catalog      =
teamhunteridentityDB; Integrated Security = True; Connection
Timeout = 30" providerName = "System.Data.SqlClient"/>
</connectionStrings>

```

Значения полей аналогичны строке подключения первого компонента.

В данном случае, помимо основной базы данных, подключается база данных первого компонента. Это необходимо для верификации прав доступа пользователя.

5.2 Работа с базами данных

В связи с тем, что при разработке используется подход Code First, создавать базы данных и описывать их таблицы через среду Microsoft SQL Server Management Studio не требуется. При первом запуске системы на указанном в строке подключения сервере создаются базы данных с соответствующими именами и генерируются таблицы, описанные в коде программы. Результат создания баз данных представлен на рисунке 5.1.

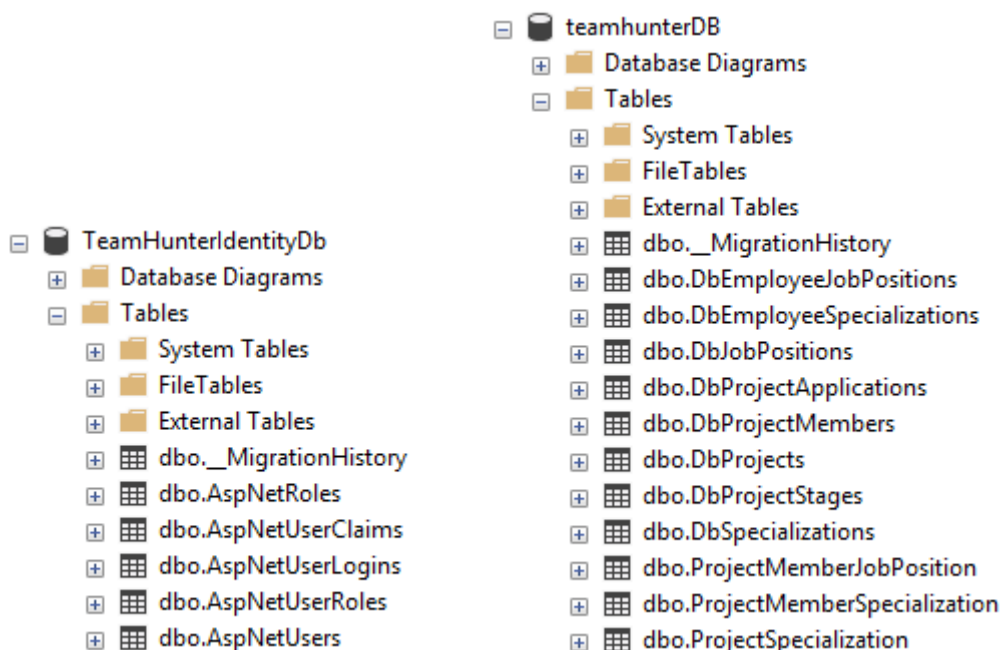


Рисунок 5.1 – Результат создания базы данных

Если при разработке системы поменялась структура таблиц, то после перезапуска системы таблицы базы данных примут соответствующий вид. Такое поведение обусловлено использованием Entity Framework. Результат модификации таблицы DbProjects базы данных teamHunterDB представлен на рисунке 5.2 (из структуры класса было удалено поле JiraLink)

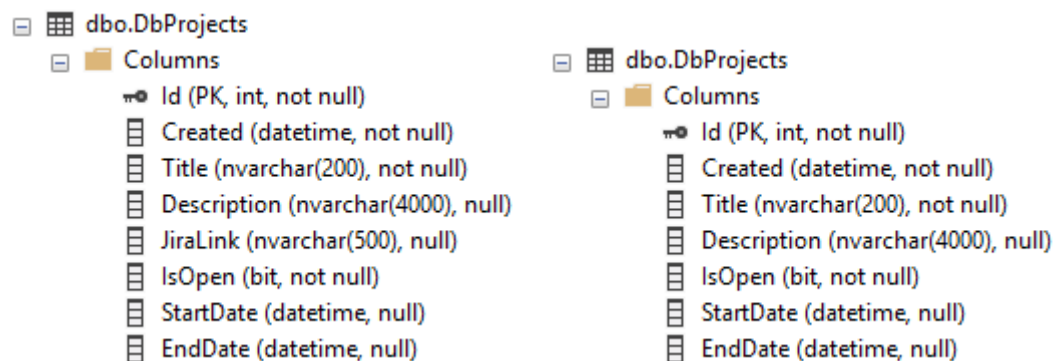


Рисунок 5.2 – Результат модификации базы данных. До (слева) и после (справа)

Для каждого класса, отвечающего за определенную таблицу в базе данных, создается свой класс конфигурации, в котором указываются необходимые свойства столбцов таблицы. Например, для таблицы специализаций класс конфигурации выглядит следующим образом (листинг 12):

Листинг 12. Класс конфигурации SpecializationConfig.cs

```
public class SpecializationConfig :
EntityTypeConfiguration<DbSpecialization>
{
    public SpecializationConfig()
    {
        HasKey(c => c.Id);
        Property(c => c.Name).HasMaxLength(400).IsRequired();
    }
}
```

Здесь указывается первичный ключ (свойство HasKey) и для поля Name указывается максимальная длина (400 символов) и то, что поле не может быть пустым.

6 ПУБЛИКАЦИЯ И ТЕСТИРОВАНИЕ СИСТЕМЫ

6.1 Размещение системы на сервере

Для публикации системы используется Internet Information Services. Каждый компонент публикуется поотдельности. Для этого в диспетчере конфигураций IIS на сервере, на котором будет размещена система, необходимо выбрать необходимый веб-сайт или создать новый веб-сайт. В данном случае будем использовать Default Web Site (Рисунок 6.1).

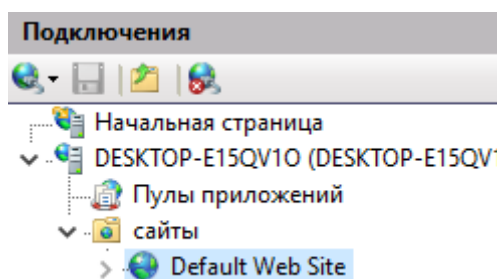


Рисунок 6.1 – Окно подключений в диспетчере конфигураций IIS

После этого, в обозревателе решений в среде разработки Microsoft Visual Studio необходимо создать профиль публикации для интересующего нас компонента. Требуется указать способ публикации, сервер, на котором будет развернут сайт, имя сайта и его адрес в сети.

Опубликовать

Опубликовать

Подключение

Параметры

CustomProfile1 *

Способ публикации: Веб-развертывание

Сервер: localhost

Имя сайта: Default Web Site/thi

Имя пользователя:

Пароль:

☐ Сохранить пароль

Целевой URL-адрес: http://localhost/thi

Проверить подключение ✓

< Назад Далее > **Сохранить** Отмена

Рисунок 6.2 – Настройки профиля публикации для компонента TeamHunterIdentity

Опубликовать

Опубликовать

Подключение

Параметры

CustomProfile *

Способ публикации: Веб-развертывание

Сервер: localhost

Имя сайта: Default Web Site/th

Имя пользователя:

Пароль:

☐ Сохранить пароль

Целевой URL-адрес: http://localhost/th

Проверить подключение ✓

< Назад Далее > **Сохранить** Отмена

Рисунок 6.3 – Настройки профиля публикации для компонента TeamHunter

После публикации компонент TeamHunterIdentity будет доступен по адресу <http://localhost/thi>, TeamHunter – <http://localhost/th>.

6.2 Проверка реакции системы на HTTP-запросы пользователей

Для того, чтобы убедиться, что система работает правильно, необходимо провести ряд HTTP-запросов:

- регистрация нового пользователя;
- вход зарегистрированного пользователя в систему;
- запрос, доступный всем пользователям системы (от имени администратора и обычного пользователя);
- запрос с ограничением прав доступа (от имени администратора и обычного пользователя);
- выход пользователя из системы.

Для проведения тестирования использовался инструмент Advanced REST Client.

6.2.1 Регистрация нового пользователя

Для регистрации нового пользователя на адрес `http://localhost/thi/Registration` необходимо отправить POST-запрос с параметрами, указанными на рисунке 6.4.

Headers	Authorization	Body	Variables	Actions
Body content type application.. ▾	Editor view Form data (www-url-form-encoded) ▾			
ENCODE PAYLOAD		DECODE PAYLOAD		
Email		user1@mail.com		×
Username		user1		×
FirstName		Аркадий		×
LastName		Аркадиев		×
Phone		880005553535		×
Password		qwerty123		×
PasswordConfirm		qwerty123		×

Рисунок 6.4 – Параметры запроса

При успешной регистрации получаем HTTP Response code 200 и ответ (Листинг 13).

Листинг 13. Ответ на успешный запрос регистрации в формате JSON.

```
{
  "Value": "91ef6a4e-a12f-483d-bcfb-75abadd2b3af",
  "Success": true,
  "Error": ""
}
```

В ответ приходит ID зарегистрировавшегося пользователя и сообщение «Success: true», что означает, что регистрация прошла успешно.

6.2.2 Вход в систему

Для входа пользователя в систему необходимо отправить POST-запрос на адрес `http://localhost/thi/OAuth/Token`. Параметры для тела запроса указаны на рисунке 6.5.

Headers	Authorization	Body	Variables	Actions
Body content type application/x-www-form... ▼	Editor view Form data (www-url-form-encoded) ▼			
ENCODE PAYLOAD DECODE PAYLOAD				
grant_type		password		×
Username		user1		×
Password		qwerty123		×

Рисунок 6.5 – Параметры для запроса на вход в систему



Рисунок 6.6 – Ответ на успешный вход в систему

На рисунке 6.6 приведен пример ответа на успешный вход в систему. Если при отправке запроса на вход в систему были введены верные учетные данные, пользователь получает токен (“access_token”) с определенным временем жизни (“expires_in”).

6.2.3 Общедоступный запрос

На рисунке 6.7 приведен пример выполнения запроса – просмотр списка всех специализаций.

☒ GET
 ☐ POST
 ☐ PUT
 ☐ DELETE
 ☐ PATCH
 Other methods ▾

Request URL
 http://localhost/th/Specializations ▾
 SEND
⋮

Parameters ▾

200 OK
302.00 ms
DETAILS ▾

📄
📥
⏏
☰

```

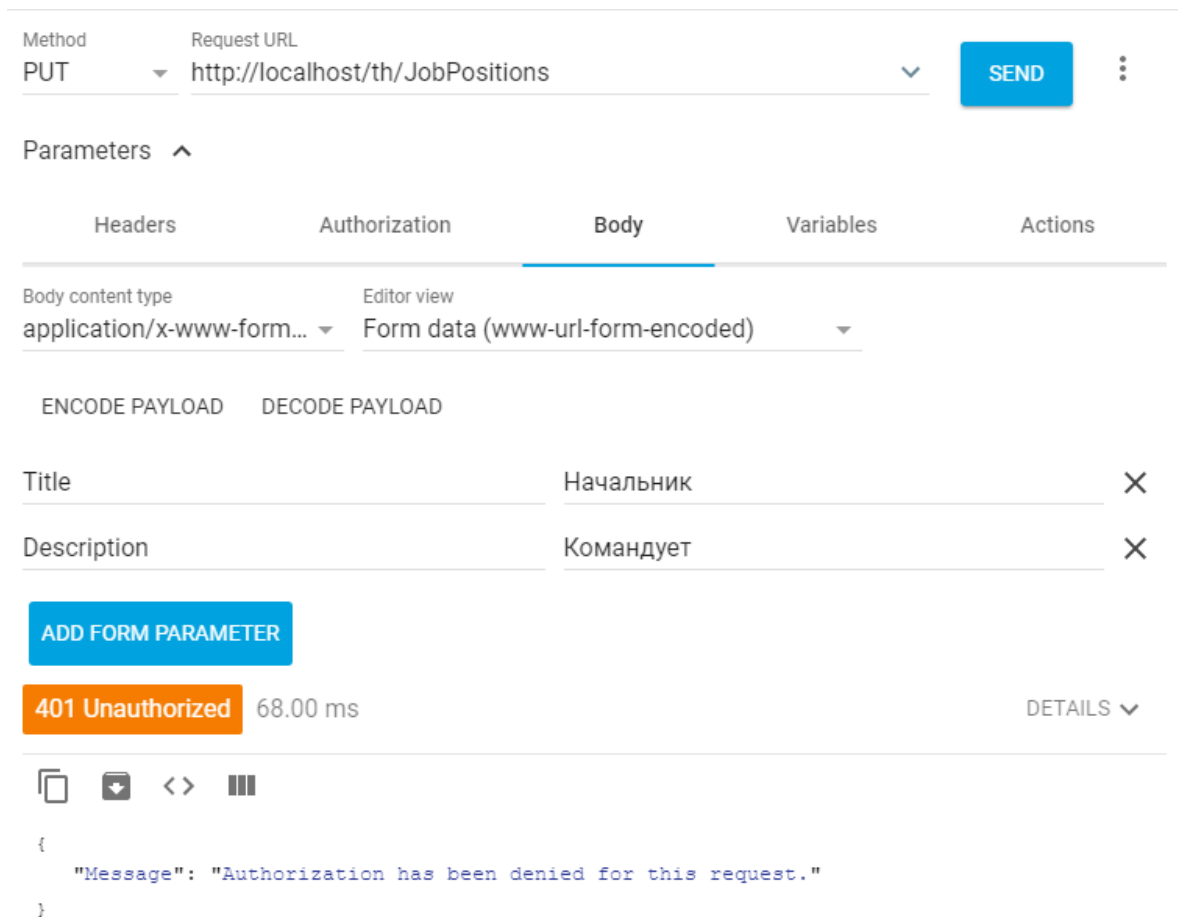
{
  -"Value": {
    "Total": 4,
    -"Data": [Array[4]
      -0: {
        "Id": 1,
        "Name": "Дизайнер",
        "Description": "Разработка макета сайтов"
      },
      -1: {
        "Id": 2,
        "Name": "Программист",
        "Description": "Разработка приложений"
      },
      -2: {
        "Id": 3,
        "Name": "Мобильный разработчик iOS",
        "Description": "Разработка мобильных приложений на базе iOS"
      },
      -3: {
        "Id": 4,
        "Name": "Мобильный разработчик Android",
        "Description": "Разработка мобильных приложений на базе ОС Android"
      }
    ],
    "Success": true,
    "Error": ""
  }
}
  
```

Рисунок 6.7 – результат выполнения общедоступного запроса.

Запрос доступен для выполнения как администраторам, так и обычным пользователям.

6.2.4 Запрос с ограничением прав доступа

На рисунке 6.8 показан пример выполнения запроса на добавление должности.



Method PUT Request URL http://localhost/th/JobPositions

Parameters

Headers Authorization Body Variables Actions

Body content type application/x-www-form... Editor view Form data (www-url-form-encoded)

ENCODE PAYLOAD DECODE PAYLOAD

Title Начальник

Description Командует

ADD FORM PARAMETER

401 Unauthorized 68.00 ms DETAILS

```
{
  "Message": "Authorization has been denied for this request."
}
```

Рисунок 6.8 – Результат запроса

Данный запрос доступен для выполнения только администраторам системы. Запрос был отправлен с обычного пользователя, следовательно, был получен HTTP Response code 401 и соответствующее сообщение.

6.2.5 Выход пользователя из системы

Выход пользователя из системы осуществляется посредством отправки GET-запроса на адрес http://localhost/thi/Logout.



Рисунок 6.9 – Ответ на успешный выход пользователя из системы

На рисунке 6.9 представлен ответ на успешный выход пользователя из системы.

6.3 Проверка работоспособности системы

Разработанная система должна удовлетворять поставленным требованиям.

Сотрудник может:

- добавлять себе специализацию и указывать уровень владения ей;
- видеть список подходящих проектов.

Администратор системы может:

- добавлять и редактировать списки специализаций и должностей;
- добавлять в систему проекты и указывать требования к ним;
- видеть список подходящих сотрудников для проекта;
- сформировать список участников проекта;
- удалить пользователя из системы.

6.3.1 Тестирование относительно сотрудника

Проведем ряд тестов относительно обычного пользователя системы.

1) Добавление специализации пользователю

Для того, чтобы указать себе специализацию, пользователю необходимо отправить POST-запрос на адрес вида

<http://localhost/th/Employee/Specializatio/{id}>, где {id} – идентификатор добавляемой специализации.

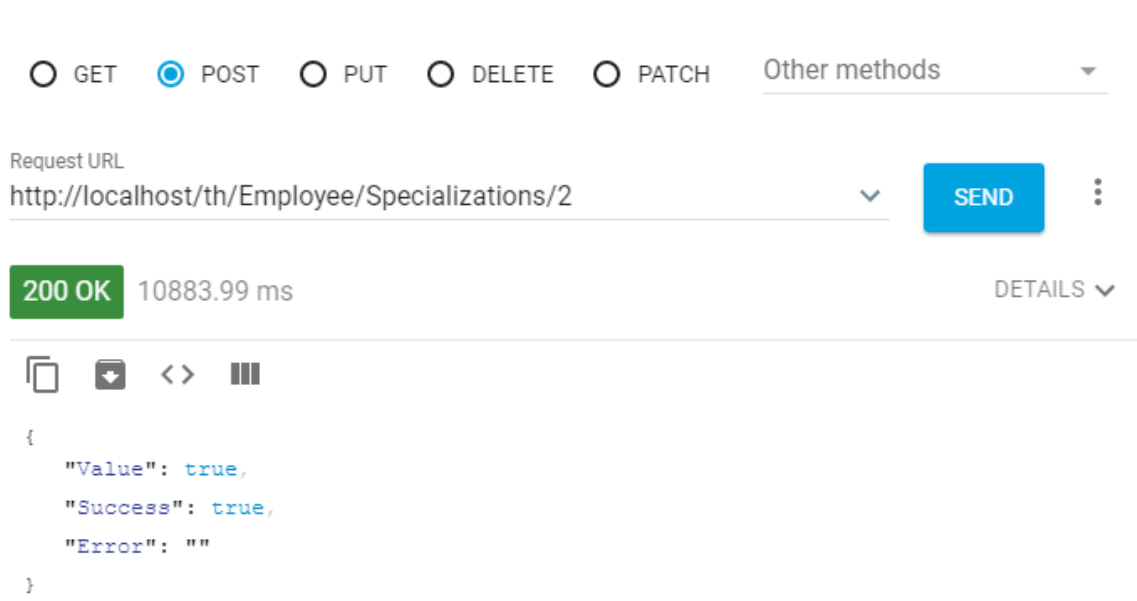


Рисунок 6.10 – Успешный запрос на добавление специализации сотрудником

На рисунке 6.10 приведен пример успешного запроса – пользователь добавил себе специализацию с ID = 2. Как видно из рисунка 6.7, данному ID соответствует специализация «Программист». Аналогичным образом была добавлена специализация «Мобильный разработчик Android» (ID = 4).

Информацию о сотруднике вместе со списком его специализаций можно посмотреть по адресу <http://localhost/th/Employees/{id}>, где {id} – идентификатор сотрудника.

```

{
  -"Value": {
    "Id": "91ef6a4e-a12f-483d-bcfc-75abadd2b3af",
    "Username": "user1",
    "FirstName": "Аркадий",
    "LastName": "Аркадиев",
    "Email": "user1@mail.com",
    "Phone": null,
    "JobPositions": [Array[0]],
    -"Specializations": [Array[2]
      -0: {
        "Id": 2,
        "Name": "Программист",
        "Description": "Разработка приложений"
      },
      -1: {
        "Id": 4,
        "Name": "Мобильный разработчик Android",
        "Description": "Разработка мобильных приложений на базе ОС Android"
      }
    ],
    "ProjectApplications": [Array[0]],
    "CurrentProject": null,
    "ProjectHistory": [Array[0]],
  },
  "Success": true,
  "Error": ""
}

```

Рисунок 6.11 – Просмотр информации о сотруднике

На рисунке 6.11 показана информация о сотруднике, добавившем себе специализации. В поле «Specializations» приведен список добавленных специализаций.

2) Добавление уровня владения специализацией.

Добавление уровня владения специализацией показано на рисунке 6.12.

GET

POST

PUT

DELETE

PATCH

Other methods

Request URL

http://localhost/th/Employee/AddSpecialization

SEND

Guid

91ef6a4e-a12f-483d-bcfb-75abadd2b3af

×

SpecId

4

×

Level

4

×

ADD FORM PARAMETER

200 OK7191.01 ms

DETAILS

<>

|||

{

"Value": true,

"Success": true,

"Error": ""

}

Рисунок 6.12 - Добавление уровня владения специализацией

На указанный на рисунке 6.12 адрес отправляется запрос, содержащий ID пользователя, ID специализации и уровень в числовом формате.

```

- "Specializations": [Array[2]
  -0: {
    "Id": 2,
    "Name": "Программист",
    "Description": "Разработка приложений"
  },
  -1: {
    "Id": 4,
    "Name": "Мобильный разработчик Android",
    "Description": "Разработка мобильных приложений на базе ОС Android"
  }
],
"ProjectApplications": [Array[0]],
"CurrentProject": null,
"ProjectHistory": [Array[0]],
- "employeeSpecializations": [Array[2]
  -0: {
    "GuId": "91ef6a4e-a12f-483d-bcfb-75abadd2b3af",
    "SpecId": 2,
    "Level": 0
  },
  -1: {
    "GuId": "91ef6a4e-a12f-483d-bcfb-75abadd2b3af",
    "SpecId": 4,
    "Level": 4
  }
],
},
"Success": true,
"Error": ""
}

```

Рисунок 6.13 – Просмотр информации о пользователе

После добавления специализаций, они указываются в поле «employeeSpecializations» в информации о пользователе (Рисунок 6.13).

3) Просмотр списка подходящих проектов

Список подходящих проектов составляется относительно специализаций пользователя: если в списке специализаций проекта указаны специализации, соответствующие пользователю, проект добавляется в список (Рисунок 6.14).

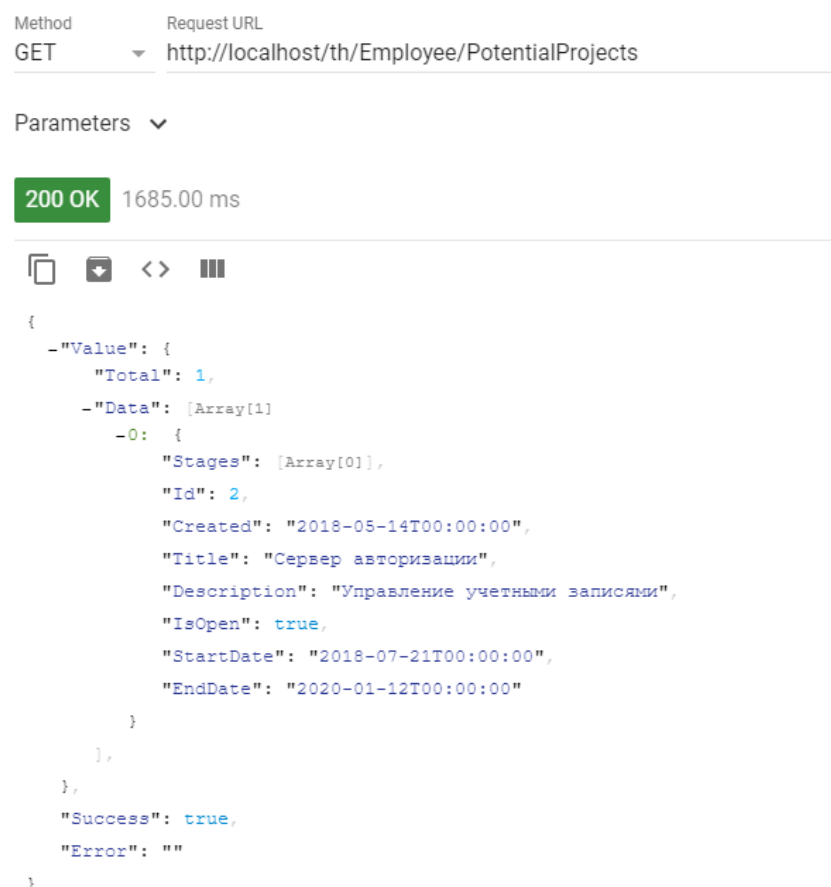


Рисунок 6.14 – Список доступных проектов

Если проект подходит пользователю и тот в нем заинтересован, можно оставить заявку для участия в проекте.

4) Создание заявки на участие в проекте

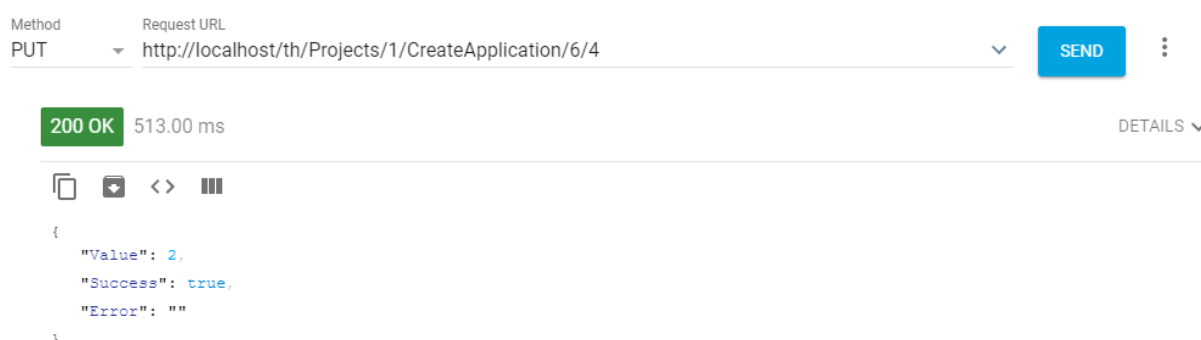


Рисунок 6.15 – Результат отправки заявки на участие в проекте

Пользователь может оставить заявку для участия в проекте (Рисунок 6.15). Для этого ему необходимо отправить запрос на адрес

http://localhost/th/Projects/{ProjectID}/CreateApplication/{JobPositionID}/{SpecializationID}, где {ProjectID}, {JobPositionID}, {SpecializationID} – ID проекта, ID должности, на которую претендует пользователь в этом проекте, ID специализации сотрудника соответственно.

6.3.2 Тестирование относительно администратора

Проведем ряд тестов относительно обычного пользователя системы.

1) Добавление записей в списки специализаций и должностей.

Администратору предоставлена возможность добавления специализации (Рисунок 6.16).

The screenshot shows a REST client interface with the following components:

- Request URL:** `http://localhost/th/Specialization`
- Parameters:** A tabbed interface with **Headers**, **Authorization**, **Body** (selected), **Variables**, and **Actions**.
- Body content type:** `application/x-www-form...`
- Editor view:** `Form data (www-url-form-encoded)`
- Buttons:** `ENCODE PAYLOAD` and `DECODE PAYLOAD`.
- Form Data Table:**

Name	Value	Action
Name	Дизайнер	✕
Description	Разработка макета сайтов	✕
- Buttons:** `ADD FORM PARAMETER`
- Status Bar:** `200 OK` and `895.00 ms`. A `DETAILS` dropdown is also present.
- Response Body:**

```
{
  "Value": 1,
  "Success": true,
  "Error": ""
}
```

Рисунок 6.16 – Добавление специализации

В ответ на запрос добавления специализации приходит сообщение, что информация добавлена успешно и ID добавленной специализации. Добавление должности в список происходит аналогичным способом.

2) Редактирование списков специализаций и должностей.

На рисунке 6.17 показан список должностей в формате JSON.

```
{
  -"Value": {
    "Total": 7,
    -"Data": [Array[7]
      -0: {
        "Id": 1,
        "Title": "PHP-программист"
      },
      -1: {
        "Id": 2,
        "Title": "PHP-программист"
      },
      -2: {
        "Id": 3,
        "Title": "Начальник"
      },
      -3: {
        "Id": 4,
        "Title": "Начальник"
      },
      -4: {
        "Id": 5,
        "Title": "Менеджер проекта"
      },
      -5: {
        "Id": 6,
        "Title": "Разработчик"
      },
      -6: {
        "Id": 7,
        "Title": "Разработчик"
      }
    ],
  },
  "Success": true,
  "Error": ""
}
```

Рисунок 6.17 – Список должностей

Удалим повторяющиеся (4, 7) и не относящиеся к должностям (1, 2) записи из списка. Для этого на адрес `http://localhost/th/JobPositions/{id}`, где {id} – ID должности, которую требуется удалить, отправим DELETE-запрос (Рисунок 6.18).

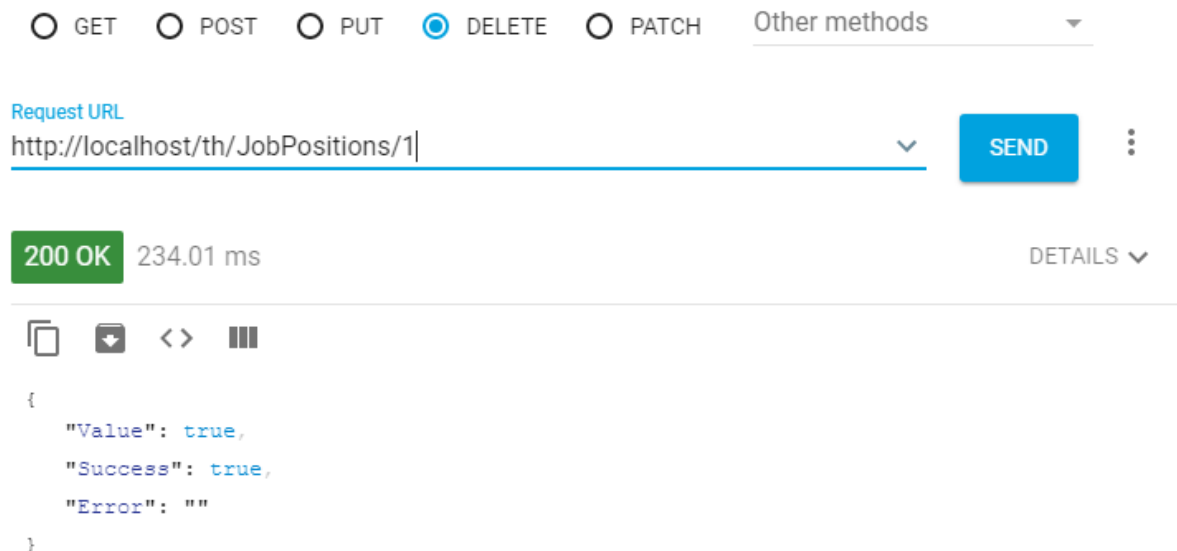


Рисунок 6.18 – Удаление из списка должности с ID = 1

```
{
  -"Value": {
    "Total": 3,
    -"Data": [Array[3]
      -0: {
        "Id": 3,
        "Title": "Начальник"
      },
      -1: {
        "Id": 5,
        "Title": "Менеджер проекта"
      },
      -2: {
        "Id": 6,
        "Title": "Разработчик"
      }
    ],
  },
  "Success": true,
  "Error": ""
}
```

Рисунок 6.19 – Результат удаления повторяющихся записей

После удаления всех повторяющихся записей, получим список, представленный на рисунке 6.19.

3) Добавление проекта.

Добавим несколько проектов в список. Для этого на адрес <http://localhost/th/Project> необходимо отправить PUT-запрос, в теле которого необходимо указать поля: Title (название), Description (описание), JiraLink (ссылка, необязательное поле, по умолчанию null), isOpen, StartDate (дата начала проекта, необязательное поле, по умолчанию указывается текущая дата), EndDate (дата окончания проекта, необязательное поле, по умолчанию указывается текущая дата).

200 OK 483.00 ms



```
{
  -"Value": {
    "Total": 2,
    -"Data": [Array[2]
      -0: {
        "Stages": [Array[0]],
        "Id": 1,
        "Created": "2018-05-14T00:00:00",
        "Title": "Система подбора персонала",
        "Description": "Подбирает персонал",
        "IsOpen": true,
        "StartDate": "2018-05-14T00:00:00",
        "EndDate": "2018-05-14T00:00:00"
      },
      -1: {
        "Stages": [Array[0]],
        "Id": 2,
        "Created": "2018-05-14T00:00:00",
        "Title": "Сервер авторизации",
        "Description": "Управление учетными записями",
        "IsOpen": true,
        "StartDate": "2018-05-14T00:00:00",
        "EndDate": "2018-05-14T00:00:00"
      }
    ],
    "Success": true,
    "Error": ""
  }
}
```

Рисунок 6.20 – Список проектов

На рисунке 6.20 представлен список проектов, добавленных в базу.

4) Редактирование информации о проекте.

Отредактируем информацию о проекте с ID = 2 (Рисунок 6.21), изменив дату начала и окончания на 21 июля 2018 года и 12 января 2020 года соответственно (даты вводятся в формате ММ.ДД.ГГГ).

☐ GET
 ☒ POST
 ☐ PUT
 ☐ DELETE
 ☐ PATCH
 Other methods ▾

Request URL
 http://localhost/th/Project ▾
 SEND
 ⋮

Parameters ^

Headers	Authorization	Body	Variables	Actions
Body content type application/x-www-form... ▾		Editor view Form data (www-url-form-encoded) ▾		
ENCODE PAYLOAD		DECODE PAYLOAD		
Title	Сервер авторизации			✕
Description	Управление учетными записями			✕
JiraLink	www.auth.net			✕
IsOpen	true			✕
Id	2			✕
StartDate	07.21.2018			✕
EndDate	1.12.2020			✕

Рисунок 6.21 – Редактирование информации о проекте

После редактирования, видим список проектов, представленный на рисунке 6.22.

```

{
  -"Value": {
    "Total": 2,
    -"Data": [Array[2]
      -0: {
        "Stages": [Array[0]],
        "Id": 1,
        "Created": "2018-05-14T00:00:00",
        "Title": "Система подбора персонала",
        "Description": "Подбирает персонал",
        "IsOpen": true,
        "StartDate": "2018-05-14T00:00:00",
        "EndDate": "2018-05-14T00:00:00"
      },
      -1: {
        "Stages": [Array[0]],
        "Id": 2,
        "Created": "2018-05-14T00:00:00",
        "Title": "Сервер авторизации",
        "Description": "Управление учетными записями",
        "IsOpen": true,
        "StartDate": "2018-07-21T00:00:00",
        "EndDate": "2020-01-12T00:00:00"
      }
    ],
    "Success": true,
    "Error": ""
  }
}

```

Рисунок 6.22 – Список проектов после изменения

Как видно из рисунка 6.22, даты в проекте с изменились на указанные в запросе (Рисунок 6.21).

5) Добавление специализации для проекта

The screenshot shows a REST client interface. At the top, the 'Request URL' is set to 'http://localhost/th/Project/Specializations'. A blue 'SEND' button is to the right. Below the URL, there are two input fields: 'ProjectId' with the value '2' and 'SpecializationId' with the value '4'. Each field has a small 'X' icon to its right. Below these fields is a blue button labeled 'ADD FORM PARAMETER'. At the bottom left, a green status bar shows '200 OK' and '6115.00 ms'. To the right of this is a 'DETAILS' link with a dropdown arrow. Below the status bar is a code editor showing the JSON response:

```
{
  "Value": true,
  "Success": true,
  "Error": ""
}
```

Рисунок 6.23 – Добавление специализации для проекта

При добавлении специализации необходимо на соответствующий адрес отправить запрос с указанием ID проекта и ID специализации (Рисунок 6.23). После чего, запись добавится в соответствующую таблицу базы данных. На рисунке НОМЕР проекту с ID = 2 (“Сервер авторизации”) была добавлена специализация с ID = 4 (“Мобильный разработчик Android”).

6) Просмотр списка потенциальных сотрудников для проекта.

Список потенциальных сотрудников формируется на основе их специализаций: если в списке специализаций проекта числится хотя бы одна специализация сотрудника, сотрудник добавляется в список. Для наглядности в базу данных сотрудников был добавлен еще один пользователь со специализацией «Дизайнер» (ID = 1).

Для просмотра списка потенциальных сотрудников необходимо отправить GET-запрос на адрес

<http://localhost/th/Projects/{id}/PotentialMembers>, где {id} – идентификатор проекта.

```
{
  -"Value": {
    "Total": 1,
    -"Data": [Array[1]
      -0: {
        "Id": "91ef6a4e-a12f-483d-bcfb-75abadd2b3af",
        "Username": "user1",
        "FirstName": "Аркадий",
        "LastName": "Аркадиев",
        "Email": "user1@mail.com",
        "Phone": null,
        "JobPositions": [Array[0]],
      -"Specializations": [Array[2]
        -0: {
          "Id": 2,
          "Name": "Программист",
          "Description": "Разработка приложений"
        },
        -1: {
          "Id": 4,
          "Name": "Мобильный разработчик Android",
          "Description": "Разработка мобильных приложений на базе ОС Android"
        }
      ]
    ]
  },
}
```

Рисунок 6.24 – Просмотр списка потенциальных сотрудников

На рисунке 6.24 показан список потенциальных сотрудников для проекта с ID = 2 (“Сервер авторизации”). Данному проекту была присвоена специализация с ID = 4 (Рисунок 6.23). Следовательно, в списке отображаются все сотрудники с этой специализацией.

7) Добавление сотрудника в список участников проекта.

По оставленной заявке на вступление в проект (Рисунок 6.25) администратор может добавить сотрудника в список участников. Для этого необходимо по адресу <http://localhost/th/Projects/AddMember/{id}>, где {id} – идентификатор заявки, отправить запрос (Рисунок 6.26).

Method GET Request URL http://localhost/th/Project/1/Applications

Parameters

200 OK 599.00 ms

Value (Object)

Total 1

Data (Array 1)

Id	Username	FirstName	LastName	Email	Phone	JobPositions	Specializations								
91ef6a4e-a12f-483d-bcfb-75abadd2b3af	user1	Аркадий	Аркадиев	user1@mail.com		hide array <table border="1"> <thead> <tr> <th>Id</th> <th>Title</th> </tr> </thead> <tbody> <tr> <td>6</td> <td>Разработчик</td> </tr> </tbody> </table>	Id	Title	6	Разработчик	hide array <table border="1"> <thead> <tr> <th>Id</th> <th>Name</th> </tr> </thead> <tbody> <tr> <td>4</td> <td>Мобильный разработчик Android</td> </tr> </tbody> </table>	Id	Name	4	Мобильный разработчик Android
Id	Title														
6	Разработчик														
Id	Name														
4	Мобильный разработчик Android														

Рисунок 6.25 – Список заявок для участия в проекте

Method PUT Request URL http://localhost/th/Projects/AddMember/2

Parameters

200 OK 1020.00 ms

```
{
  "Value": 1,
  "Success": true,
  "Error": ""
}
```

Рисунок 6.26 – Добавление участника проекта

После добавления, соответствующий проект будет отображен в данных сотрудника в поле CurrentProject, что видно из рисунка 6.27.

```

{
  -"Value": {
    "Id": "91ef6a4e-a12f-483d-bcfb-75abadd2b3af",
    "Username": "user1",
    "FirstName": "Аркадий",
    "LastName": "Аркадиев",
    "Email": "user1@mail.com",
    "Phone": null,
    "JobPositions": [Array[0]],
  -"Specializations": [Array[2]
    -0: {
      "Id": 2,
      "Name": "Программист",
      "Description": "Разработка приложений"
    },
    -1: {
      "Id": 4,
      "Name": "Мобильный разработчик Android",
      "Description": "Разработка мобильных приложений на базе ОС Android"
    }
  ],
  -"ProjectApplications": [Array[1]
    -0: {
      "Id": 2,
      "GuId": "91ef6a4e-a12f-483d-bcfb-75abadd2b3af",
      "ProjectId": 1,
      "JobPositionId": 6,
      "SpecializationId": 4
    }
  ],
  -"CurrentProject": {
    "Stages": [Array[0]],
    "Id": 1,
    "Created": "2018-05-14T00:00:00",
    "Title": "Система подбора персонала",
    "Description": "Подбирает персонал",
    "IsOpen": true,
    "StartDate": "2018-05-14T00:00:00",
    "EndDate": "2018-05-14T00:00:00"
  }
}

```

Рисунок 6.27 – Просмотр информации о сотруднике

8) Удаление сотрудника

Администратору предоставлена возможность удаления учетных данных сотрудника (например, при увольнении из компании). Для этого необходимо отправить запрос с учетными данными удаляемого пользователя на соответствующий адрес.

Method: DELETE Request URL: http://localhost/thi/User

Parameters

Headers Authorization Body Variables Actions

Body content type: application/x-www-form-urlencoded Editor view: Form data (www-url-form-encoded)

ENCODE PAYLOAD DECODE PAYLOAD

UserName: user2

Password: qwerty123

ADD FORM PARAMETER

200 OK 1567.99 ms

```
{
  "Value": true,
  "Success": true,
  "Error": ""
}
```

Рисунок 6.28 – Удаление сотрудника

После успешного запроса (Рисунок 6.28), запись о сотруднике удаляется из базы данных.

Value (Object)

Total 2

Data (Array 2)

Id	Username	FirstName	LastName	Email	Phone	JobPositions	Specializations	ProjectApplications	CurrentStatus
91ef6a4e-a12f-483d-bcfb-75abadd2b3af	user1	Аркадий	Аркадиев	user1@mail.com		Array (0) show array	Array (2) show array		Staf
e68443ad-5430-4f12-973f-d04aa5664633	superuser					Array (0) show array	Array (0) show array		

Рисунок 6.29 – Просмотр списка сотрудников

Как видно из рисунка 6.29, после удаления сотрудника с логином user2, в списке сотрудников осталось два пользователя.

ЗАКЛЮЧЕНИЕ

В процессе работы были определены и сформулированы требования к решению поставленной задачи, методика решения и инструменты реализации.

На базе платформы ASP.NET была разработана автоматизированная система подбора персонала для формирования проектных групп.

Были изучены особенности приложений на платформе ASP.NET, получены навыки разработки приложений с использованием Entity Framework и спецификацией OWIN, освоен подход Code First. Изучены основы авторизации по протоколу OAuth2 с помощью токенов.

Клиентская часть, ввиду универсальности системы, может быть разработана в виде любого приложения под любую операционную систему будь то веб-сайт или мобильное приложение под iOS. Система может быть применена в любой предметной области – таблицы подходят под описание задач различных категорий. При дальнейшей разработке системы (в зависимости от требований той или иной организации) возможно добавление дополнительных критериев, например, пола и возраста сотрудников, расширение описания проектов: как вариант, установление приоритета проекта, оценка работы участников проектов пользователями системы с целью повышения их знаний, умений и навыков.

Выполненная работа полностью соответствует полученному заданию. Разработанная система может быть применена в качестве поиска новых сотрудников для организации, если для проекта потребуется специалист с определенным уровнем знаний в какой-либо сфере или определенной должности, а в штате такой отсутствует. Для этого необходимо произвести минимальные модификации: добавить еще одну таблицу базы данных, в которой будет указана информация о человеке, отправившем свое резюме.

СПИСОК ИСПОЛЬЗУЕМЫХ ИСТОЧНИКОВ

1. Повышение эффективности подбора персонала за счет автоматизации процесса. URL: <http://efsol.ru/articles/recruiting-automation.html>
2. Выбор языка программирования. URL: <http://informatikum.ru/programmirovanie/2-vybor-yazyka-programmirovaniya.html>
3. C++, C#, C? URL: <http://www.codenet.ru/progr/cpp/ccc.php>
4. Введение в язык C# и .NET Framework. URL: <https://msdn.microsoft.com/ru-ru/library/z1zx9t92.aspx>
5. Платт Д. С. Знакомство с Microsoft .NET. — М.: И. Д. Русская редакция, 2001. — 240 с.
6. Дино Эспозито. Microsoft ASP.NET 2.0. Базовый курс. — СПб: И. Д. Питер, 2007. — 688 с.
7. Архитектура веб-приложений ASP.NET. Разработка веб-приложений на платформе .NET. URL: <https://www.intuit.ru/studies/courses/485/341/lecture/8198>
8. The Official Microsoft IIS Site. URL: <https://www.iis.net/>
9. OWIN и Katana. URL: <http://andrey.moveax.ru/post/owin-introduction-part1>
10. ADO.NET. URL: <https://docs.microsoft.com/ru-ru/dotnet/framework/data/adonet/>
11. Введение в Entity Framework. URL: <https://metanit.com/sharp/entityframework/1.1.php>
12. Создание приложений для СУБД Firebird с использованием различных компонент и драйверов: ADO.NET Entity Framework 6. URL: <https://itnan.ru/post.php?c=1&p=278405>
13. Подходы для работы с Entity Framework. URL: https://professorweb.ru/my/entity-framework/6/level1/1_3.php

14. Getting Started Guide – AutoMapper Documentation.
URL: <http://docs.automapper.org/en/stable/Getting-started.html>
15. AutoMapper. URL: <http://docs.automapper.org/en/stable/index.html>
16. Сравнение баз данных: Microsoft Sql Server и Microsoft Visual Foxpro. URL: <http://softclipper.net/foxpro-i-sql/sravnenie-baz-dannykh-microsoft-sql-server-i-microsoft-visual-foxpro.html>
17. Сравнительные характеристики СУБД.
URL: http://orastack.narod.ru/index_file/onebd.html
18. Аутентификация и авторизация.
URL: https://metanit.com/sharp/aspnet_webapi/5.1.php
19. D. Hardt, Ed. The OAuth 2.0 Authorization Framework. Internet Engineering Task Force.
20. Gibbons K., Raw J. O., Curran K. Security Evaluation of the OAuth 2.0 Framework // Information Management and Computer Security – 2014. – Vol. 22.
21. Chen E. Y., Pei Y., Chen S. et al. OAuth Demystified for Mobile Application Developers // Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security – New York City: ACM, 2014. – P. 892–903.
22. E. Hammer-Lahav, Ed. The OAuth 1.0 Protocol. Internet Engineering Task Force.
23. Введение в JSON. URL: <http://json.org/json-ru.html>
24. Федоров, А.Г. Microsoft Visual Studio 2010. Первое знакомство. – М.: Изд. Дом «Вильямс», 2009. – 42 с.
25. Технический центр по IIS. URL: <https://technet.microsoft.com/ru-ru/bb466129.aspx>
26. Advanced REST Client. URL: <https://www.hosoft.ru/plugins/google-chrome/advanced-rest-client-3.1.5>
27. Полезные расширения Google Chrome для разработчиков.
URL: <https://fructcode.com/ru/blog/useful-google-chrome-extensions-for-web-programmers/>

28. Система подбора персонала. URL: <https://www.hr-director.ru/article/65959-qqq-16-m3-sistema-podbora-personala>

29. SQL Server и ADO.NET. URL: [https://msdn.microsoft.com/ru-ru/library/kb9s9ks0\(v=vs.100\).aspx](https://msdn.microsoft.com/ru-ru/library/kb9s9ks0(v=vs.100).aspx)

ПРИЛОЖЕНИЕ А

ЛИСТИНГИ ПРОГРАММ

(обязательное)

Приложение А.1

Листинг сервиса TeamHunterIdentity

Приложение А.1.1

Файл «LoginModelDto.cs»

```
namespace TeamHunterIdentity.DataContracts
{
    public class LoginModelDto
    {
        public string Username { get; set; }

        public string Password { get; set; }
    }
}
```

Приложение А.1.2

Файл «RegisterModelDtoModelDto.cs»

```
namespace TeamHunterIdentity.DataContracts
{
    public class RegisterModelDto
    {
        public string Email { get; set; }
        public string Username { get; set; }
        public string FirstName { get; set; }
        public string LastName { get; set; }
        public string Phone { get; set; }
        public string Password { get; set; }
        public string PasswordConfirm { get; set; }
    }
}
```

Приложение А.1.3

Файл «ResultDto.cs»

```
namespace TeamHunterIdentity.DataContracts
{
    public class ResultDto
    {
        public bool Success { get; private set; }
    }
}
```

```

        public string Error { get; private set; }

        protected ResultDto(bool success, string error)
        {
            Success = success;
            Error = error;
        }

        public static ResultDto Fail(string message)
        {
            return new ResultDto(false, message);
        }

        public static ResultDto<T> Ok<T>(T value)
        {
            return new ResultDto<T>(value, true, string.Empty);
        }
    }

    public class ResultDto<T> : ResultDto
    {
        public T Value { get; set; }

        protected internal ResultDto(T value, bool success,
string error)
            : base(success, error)
        {
            Value = value;
        }
    }
}

```

Приложение А.1.4

Файл «RegisterModelDtoModelDto.cs»

```

namespace TeamHunterIdentity.DataContracts
{
    public class RegisterModelDto
    {
        public string Email { get; set; }
        public string Username { get; set; }
        public string FirstName { get; set; }
        public string LastName { get; set; }
        public string Phone { get; set; }
        public string Password { get; set; }
        public string PasswordConfirm { get; set; }
    }
}

```

Приложение А.1.5

Файл «ApplicationContext.cs»

```
namespace TeamHunterIdentity.DataAccess
{
    public class ApplicationDbContext :
    IdentityDbContext<ApplicationUser>
    {
        public ApplicationDbContext()
            : base("TeamHunterIdentityDb", throwIfV1Schema:
false)
        {
            Configuration.ProxyCreationEnabled = false;
            Configuration.LazyLoadingEnabled = false;
        }

        public static ApplicationDbContext Create()
        {
            var dbContext = new ApplicationDbContext();
            return dbContext;
        }
    }
}
```

Приложение А.1.6

Файл «AccountController.cs»

```
namespace TeamHunterIdentity.Controllers
{
    public class AccountController : ApiController
    {
        private ApplicationUserManager UserManager
        {
            get
            {
                return
HttpContext.Current.GetOwinContext().GetUserManager<ApplicationU
serManager>();
            }
        }
        [Route("Registration")]
        [HttpPost]
        public async Task<IHttpActionResult>
Registration(RegisterModelDto model)
        {
            ResultDto response;
            var validator = new RegisterValidator();
            var validate = validator.Validate(model);

            if (!validate.IsValid)
            {
                var errors = string.Join(", ",
validate.Errors.Select(a => a.ErrorMessage));
            }
        }
    }
}
```

```

        response =
ResultDto.Fail(string.Format("Ошибка, {0}", errors));
        return Ok(response);
    }
    ApplicationUser user = new ApplicationUser()
    {
        UserName = model.Username,
        Email = model.Email,
        FirstName = model.FirstName,
        LastName = model.LastName,
        PhoneNumber = model.Phone
    };
    IdentityResult result = await
userManager.CreateAsync(user, model.Password);

    if (result.Succeeded)
    {
        userManager.AddToRole(user.Id, "user");
        response = ResultDto.Ok(user.Id);
    }
    else
    {
        string message = "Ошибка,";

        foreach (string error in result.Errors)
        {
            message += " " + error;
        }

        response = ResultDto.Fail(message);
    }

    return Ok(response);
}

[Route("Logout")].
[HttpGet].
public IHttpActionResult Logout()
{
    ResultDto response;
    var context = HttpContext.Current.GetOwinContext();
    var cookies = context.Request.Cookies;
    var cookie = cookies.FirstOrDefault(c => c.Key ==
IdentityPaths.TokenName);

    if (!string.IsNullOrEmpty(cookie.Key) &&
!string.IsNullOrEmpty(cookie.Value))
    {
        context.Response.Cookies.Delete(IdentityPaths.TokenName);
        response = ResultDto.Ok(true);
        return Ok(response);
    }
}

```

```

        response = ResultDto.Fail("Токен отсутствует");
        return Ok(response);
    }

    [Route("User")]
    [HttpDelete]
    public async Task<IHttpActionResult>
DeleteUser(LoginModelDto model)
    {
        ResultDto response;
        var validator = new LoginValidator();
        var validate = validator.Validate(model);

        if (!validate.IsValid)
        {
            var errors = string.Join(", ",
validate.Errors.Select(a => a.ErrorMessage));
            response = ResultDto.Fail(string.Format("Ошибка,
{0}", errors));
            return Ok(response);
        }

        var user = await
UserManager.FindAsync(model.Username, model.Password);
        IdentityResult result = user != null ? await
UserManager.DeleteAsync(user) : new IdentityResult("User not
found");

        if (result.Succeeded)
        {
            response = ResultDto.Ok(true);
            return Ok(response);
        }
        else
        {
            string message = "Ошибка, ";

            foreach (string error in result.Errors)
            {
                message += " " + error;
            }

            response = ResultDto.Fail(message);
        }

        return Ok(response);
    }
}

```

Приложение А.1.7

Файл «SuperuserMiddleware.cs»

```
namespace TeamHunterIdentity.Middleware
{
    public class SuperuserMiddleware : OwinMiddleware
    {
        public SuperuserMiddleware(OwinMiddleware next) :
base(next) { }
        public override async Task Invoke(IOwinContext context)
        {
            var userManager =
context.GetUserManager<ApplicationUserManager>();
            var roleManager =
context.GetUserManager<ApplicationRoleManager>();
            var userRole = roleManager.FindByName("user");
            var adminRole = roleManager.FindByName("admin");
            var superUser = userManager.Find("superuser",
"superuser");
            if (userRole == null)
            {
                userRole = new IdentityRole { Name = "user" };
                roleManager.Create(userRole);
            }
            if (adminRole == null)
            {
                adminRole = new IdentityRole { Name = "admin" };
                roleManager.Create(adminRole);
            }
            if (superUser == null)
            {
                superUser = new ApplicationUser { UserName =
"superuser" };
                userManager.Create(superUser, "superuser");
                userManager.AddToRole(superUser.Id, "admin");
                userManager.AddToRole(superUser.Id, "user");
            }
            await Next.Invoke(context);
        }
    }
}
```

Приложение А.1.8

Файл «ApplicationOAuthProvider.cs»

```
namespace TeamHunterIdentity.Providers
{
    public class ApplicationOAuthProvider :
OAuthAuthorizationServerProvider
    {
        private readonly string _publicClientId;
        public ApplicationOAuthProvider(string publicClientId)
        {

```

```

        if (publicClientId == null)
        {
            throw new
ArgumentNullException("publicClientId");
        }
        _publicClientId = publicClientId;
    }
    public override async Task ValidateClientAuthentication(
        OAuthValidateClientAuthenticationContext context)
    {
        await Task.FromResult(context.Validated());
    }
    public override async Task
GrantResourceOwnerCredentials(OAuthGrantResourceOwnerCredentials
Context context)
    {
        var userManager =
context.OwinContext.GetUserManager<ApplicationUserManager>();
        var user = await
userManager.FindAsync(context.UserName, context.Password);
        if (user == null)
        {
            context.SetError("invalid_grant", "The user
name or password is incorrect.");

            context.Response.Headers.Add(IdentityPaths.OwinChallengeFlag
, new[] { ((int)HttpStatusCode.Unauthorized).ToString() });
            return;
        }
        ClaimsIdentity oAuthIdentity = await
user.GenerateUserIdentityAsync(userManager,
        OAuthDefaults.AuthenticationType);

        var roles = userManager.GetRoles(user.Id);
        var properties =
CreateProperties(context.UserName,
Newtonsoft.Json.JsonConvert.SerializeObject(roles.Select(x =>
x)));
        var ticket = new
AuthenticationTicket(oAuthIdentity, properties);
        context.Validated(ticket);
        context.Response.Cookies.Append(IdentityPaths.TokenName,
context.Options.AccessTokenFormat.Protect(ticket));
    }
    private static AuthenticationProperties
CreateProperties(string userName, string roles)
    {
        IDictionary<string, string> data = new
Dictionary<string, string>
        {
            { "userName", userName },
            { "roles", roles }
        };
    }

```

```

        return new AuthenticationProperties(data);
    }
}

```

Приложение А.1.9

Файл «LoginValidator.cs»

```

namespace TeamHunterIdentity.Validators
{
    public class LoginValidator :
    AbstractValidator<LoginModelDto>
    {
        public LoginValidator()
        {
            RuleFor(a =>
a.Username).NotEmpty().WithMessage("Вы не указали логин");
            RuleFor(a =>
a.Password).NotEmpty().WithMessage("Вы не указали пароль");
        }

        public override ValidationResult
Validate(ValidationContext<LoginModelDto> context)
        {
            return context.InstanceToValidate == null
? new ValidationResult(new[] { new
ValidationFailure("", "объект не задан", "Error") })
: base.Validate(context);
        }
    }
}

```

Приложение А.1.10

Файл «RegisterValidator.cs»

```

namespace TeamHunterIdentity.Validators
{
    public class RegisterValidator :
    AbstractValidator<RegisterModelDto>
    {
        public RegisterValidator()
        {
            RuleFor(a =>
a.Username).NotEmpty().WithMessage("Необходимо указать логин");
            RuleFor(a =>
a.Email).NotEmpty().WithMessage("Необходимо указать адрес
электронной почты");
            RuleFor(a => a.Email).Matches("^([a-z0-9_-
]+\.\.)*[a-z0-9_-]+@[a-z0-9_-]+(\.\. [a-
z]{2,6})$").WithMessage("Некорректный адрес электронной почты");
            RuleFor(a =>
a.FirstName).NotEmpty().WithMessage("Необходимо указать имя");
            RuleFor(a =>
a.LastName).NotEmpty().WithMessage("Необходимо указать фамилию");
        }
    }
}

```

```

        RuleFor(a => a.PasswordConfirm).Equal(a =>
a.Password).WithMessage("Пароли не совпадают");
    }

    public override ValidationResult
Validate(ValidationContext<RegisterModelDto> context)
    {
        return context.InstanceToValidate == null
        ? new ValidationResult(new[] { new
ValidationFailure("", "объект не задан", "Error") })
        : base.Validate(context);
    }
}

```

Приложение A.2

Листинг сервиса TeamHunter

Приложение A.2.1

Файл «EmployeeService.cs»

```

namespace EmployeeService
{
    public class EmployeeService : IEmployeeService
    {
        private readonly IMapper _mapper;
        private readonly IDataContext _identitycontext;
        private readonly IDataContext _context;
        public EmployeeService(IMapper mapper,
[Dependency("IdentityContext")]IDataContext identitycontext,
[Dependency("DataContext")]IDataContext context)
        {
            _mapper = mapper;
            _context = context;
            _identitycontext = identitycontext;
        }
        public EmployeeDto GetById(string id)
        {
            var dbModel =
_identitycontext.GetQueryable<DbUser>().FirstOrDefault(a =>
a.Id.Equals(id));
            if (dbModel != null)
            {
                var employee = _mapper.Map<DbUser,
EmployeeDto>(dbModel);
                employee.Specializations =
UserSpecializations(employee.Id) ;
                employee.JobPositions =
UserJobPositions(employee.Id);
            }
        }
    }
}

```

```

        employee.CurrentProject =
UserCurrentProject (employee.Id);
        employee.ProjectHistory =
UserProjectHistory (employee.Id);
        employee.ProjectApplications =
UserProjectApplications (employee.Id);
        employee.employeeSpecializations =
UserEmployeeSpecializations (employee.Id);
        return employee;
    }
    return null;
}
public SpecializationDto[] UserSpecializations(string id)
{
    var dbspecializations =
_context.GetQueryable<DbEmployeeSpecialization>().Where(a =>
a.GuId.Equals(id)).Select(p => p.Specialization);
    var specializations = new
SpecializationDto[dbspecializations.Count()];
    int i = 0;
    foreach (DbSpecialization s in dbspecializations)
    {
        specializations[i] =
_mapper.Map<DbSpecialization, SpecializationDto>(s);
        i++;
    }
    return specializations;
}
public EmployeeSpecializationDto[]
UserEmployeeSpecializations(string id)
{
    var dbspecializations =
_context.GetQueryable<DbEmployeeSpecialization>().Where(a =>
a.GuId.Equals(id));
    var specializations = new
EmployeeSpecializationDto[dbspecializations.Count()];
    int i = 0;
    foreach (DbEmployeeSpecialization s in
dbspecializations)
    {
        specializations[i] = new
EmployeeSpecializationDto { GuId = s.GuId, SpecId =
s.SpecializationId, Level = (s.Level==null)?0:(int)s.Level };
        i++;
    }
    return specializations;
}
public JobPositionDto[] UserJobPositions(string id)
{
    var dbJobPosition =
_context.GetQueryable<DbEmployeeJobPosition>().Where(a =>
a.GuId.Equals(id)).Select(p => p.JobPosition);

```

```

        var JobPosition = new
        JobPositionDto[dbJobPosition.Count()];
        int i = 0;
        foreach (DbJobPosition s in dbJobPosition)
        {
            JobPosition[i] = _mapper.Map<DbJobPosition,
            JobPositionDto>(s);
            i++;
        }
        return JobPosition;
    }
    public ProjectApplicationDto[]
    UserProjectApplications(string id)
    {
        var dbProjectApplication =
        _context.GetQueryable<DbProjectApplication>().Where(a => a.GuId
        == id);
        var ProjectApplication = new
        ProjectApplicationDto[dbProjectApplication.Count()];
        int i = 0;
        foreach (DbProjectApplication s in
        dbProjectApplication)
        {
            ProjectApplication[i] =
            _mapper.Map<DbProjectApplication, ProjectApplicationDto>(s);
            i++;
        }
        return ProjectApplication;
    }
    public ProjectOutputDto UserCurrentProject(string id)
    {
        var dbProject =
        _context.GetQueryable<DbProjectMember>().Where(a =>
        a.GuId.Equals(id)).Select(p => p.Project).FirstOrDefault(p =>
        p.IsOpen == true);
        var current = _mapper.Map<DbProject,
        ProjectOutputDto>(dbProject);
        return current;
    }
    public ProjectOutputDto[] UserProjectHistory(string id)
    {
        var dbProject =
        _context.GetQueryable<DbProjectMember>().Where(a =>
        a.GuId.Equals(id)).Select(p => p.Project).Where(p => p.IsOpen ==
        false);
        var ProjectHistory = new
        ProjectOutputDto[dbProject.Count()];
        int i = 0;
        foreach (DbProject s in dbProject)
        {
            ProjectHistory[i] = _mapper.Map<DbProject,
            ProjectOutputDto>(s);
            i++;
        }
    }

```

```

        }
        return ProjectHistory;
    }
    public IPagedResult<EmployeeDto> List(IDataRequest
request, EmployeeFilter filter)
    {
        var list = _identitycontext.GetQueryable<DbUser>();
        List<EmployeeDto> result = new List<EmployeeDto>();
        foreach (DbUser u in list)
        {
            var employee = _mapper.Map<DbUser,
EmployeeDto>(u);
            employee.Specializations =
UserSpecializations(employee.Id);
            employee.JobPositions =
UserJobPositions(employee.Id);
            employee.CurrentProject =
UserCurrentProject(employee.Id);
            employee.ProjectHistory =
UserProjectHistory(employee.Id);
            result.Add(employee);
        }
        return result.MapToPagedResult<EmployeeDto,
EmployeeDto>(request, _mapper);
    }
    public IPagedResult<EmployeeDto>
ListPotentialMembers(IDataRequest request, int projectId)
    {
        var dbProject =
_context.GetQueryable<DbProject>().FirstOrDefault(a => a.Id ==
projectId);
        var projectSpecializations =
dbProject.Specializations.Select(s => s.Id);
        List<EmployeeDto> result = new List<EmployeeDto>();
        if (projectSpecializations.Count() > 0)
        {
            var es =
_context.GetQueryable<DbEmployeeSpecialization>().Where(s =>
projectSpecializations.Contains(s.SpecializationId)).Select(e =>
e.GuId).Distinct();
            foreach (string l in es)
            {
                var dbModel =
_identitycontext.GetQueryable<DbUser>().FirstOrDefault(a =>
a.Id.Equals(l));
                if (dbModel != null)
                {
                    var employee = _mapper.Map<DbUser,
EmployeeDto>(dbModel);
                    result.Add(employee);
                }
            }
        }
    }
}

```

```

        else
        {
            var dbModel =
            _identitycontext.GetQueryable<DbUser>();
            if (dbModel != null)
            {
                foreach (DbUser u in dbModel)
                {
                    var employee = _mapper.Map<DbUser,
EmployeeDto>(u);
                    result.Add(employee);
                }
            }
            foreach (EmployeeDto e in result)
            {
                e.Specializations = UserSpecializations(e.Id);
                e.JobPositions = UserJobPositions(e.Id);
            }
            return result.MapToPagedResult<EmployeeDto,
EmployeeDto>(request, _mapper);
        }
        public IPagedResult<EmployeeDto>
WatchProjectApplications(IDataRequest request, int projectId)
        {
            var projectApplications =
            _context.GetQueryable<DbProjectApplication>().Where(a =>
a.ProjectId == projectId);
            List<DbUser> list= new List<DbUser>();
            foreach (var tmp in projectApplications)
            {
                var user =
                _identitycontext.GetQueryable<DbUser>().FirstOrDefault(a => a.Id
== tmp.GuId);
                list.Add(user);
            }
            List<EmployeeDto> result = new List<EmployeeDto>();
            foreach (DbUser u in list)
            {
                var employee = _mapper.Map<DbUser,
EmployeeDto>(u);
                employee.Specializations = new
SpecializationDto[1];
                employee.Specializations[0] =
_mapper.Map<DbSpecialization, SpecializationDto>(
_context.GetQueryable<DbSpecialization>().FirstOrDefault(s =>
s.Id == projectApplications.FirstOrDefault(a => a.GuId ==
employee.Id).SpecializationId));
                employee.JobPositions = new JobPositionDto[1];
                employee.JobPositions[0] =
_mapper.Map<DbJobPosition,
JobPositionDto>(_context.GetQueryable<DbJobPosition>().FirstOrDe

```

```

        fault(s => s.Id == projectApplications.FirstOrDefault(a => a.Guid
        == employee.Id).JobPositionId));
        result.Add(employee);
    }
    return
result.AsQueryable<EmployeeDto>().MapToPagedResult<EmployeeDto,
EmployeeDto>(request, _mapper);
    }
}
}

```

Приложение A.2.2

Файл «EmployeeSpecializationService.cs»

```

namespace EmployeeSpecializationService
{
    public class EmployeeSpecializationService :
    IEmployeeSpecializationService
    {
        private readonly IDataContext _identityContext;
        private readonly IDataContext _dataContext;
        public
EmployeeSpecializationService([Dependency("IdentityContext")] IDa
taContext context, [Dependency("DataContext")] IDataContext
dataContext)
        {
            _identityContext = context;
            _dataContext = dataContext;
        }
        public bool AddSpecialization(string employeeGuid, int
specId)
        {
            var employee =
_identityContext.GetQueryable<DbUser>().FirstOrDefault(a => a.Id
== employeeGuid);
            var specialization =
_dataContext.GetQueryable<DbSpecialization>().FirstOrDefault(a =>
a.Id == specId);
            if (employee != null && specialization != null)
            {
                var e =
_dataContext.GetQueryable<DbEmployeeSpecialization>().FirstOrDef
ault(a => a.Guid == employee.Id && a.SpecializationId ==
specialization.Id);
                var exist =
_dataContext.GetQueryable<DbEmployeeSpecialization>().FirstOrDef
ault(a => a.Guid == employee.Id && a.SpecializationId ==
specialization.Id) != null;
                if (exist)
                {
                    return false;
                }
                var dbEmplSpec = new DbEmployeeSpecialization {
GuidId = employee.Id, SpecializationId = specialization.Id };
            }
        }
    }
}

```

```

        _dataContext.Insert(dbEmplSpec);
        _dataContext.Save();
        return true;
    }
    return false;
}

public bool DeleteSpecialization(string employeeGuid, int
specId)
{
    var employee =
_identityContext.GetQueryable<DbUser>().FirstOrDefault(a => a.Id
== employeeGuid);
    var specialization =
_dataContext.GetQueryable<DbSpecialization>().FirstOrDefault(a =>
a.Id == specId);
    if (employee != null && specialization != null)
    {
        var dbModel =
_dataContext.GetQueryable<DbEmployeeSpecialization>().FirstOrDef
ault(a => a.GuId == employee.Id && a.SpecializationId ==
specialization.Id);
        if (dbModel == null)
        {
            return false;
        }
        _dataContext.Delete(dbModel);
        _dataContext.Save();
        return true;
    }
    return false;
}

public bool UpdateSpecialization(string employeeGuid, int
specId, int level)
{
    var employee =
_identityContext.GetQueryable<DbUser>().FirstOrDefault(a => a.Id
== employeeGuid);
    var specialization =
_dataContext.GetQueryable<DbSpecialization>().FirstOrDefault(a =>
a.Id == specId);
    if (employee != null && specialization != null)
    {
        var dbModel =
_dataContext.GetQueryable<DbEmployeeSpecialization>().FirstOrDef
ault(a => a.GuId == employee.Id && a.SpecializationId ==
specialization.Id);
        if (dbModel == null)
        {
            return false;
        }
        dbModel.Level = level;
        _dataContext.Update(dbModel);
        _dataContext.Save();
    }
}

```

```

        return true;
    }
    return false;
}
}
}

```

Приложение А.2.3

Файл «JobPositionService.cs»

```

namespace JobPositionService
{
    public class JobPositionService : IJobPositionService
    {
        private readonly IMapper _mapper;
        private readonly IDataContext _context;
        public JobPositionService(IMapper mapper,
[Dependency("DataContext")]IDataContext context)
        {
            _mapper = mapper;
            _context = context;
        }
        public int Create(JobPositionDto model)
        {
            DbJobPosition dbjobposition =
_mapper.Map<JobPositionDto, DbJobPosition>(model);
            var result =
_context.Insert<DbJobPosition>(dbjobposition);
            _context.Save();
            return result.Id;
        }
        public bool Delete(int id)
        {
            DbJobPosition jobposition =
_context.GetQueryable<DbJobPosition>().ToList().Find(a => a.Id ==
id);
            if (jobposition != null)
            {
                _context.Delete<DbJobPosition>(jobposition);
                _context.Save();
                return true;
            }
            return false;
        }
        public JobPositionDto GetById(int id)
        {
            var dbResult =
_context.GetQueryable<DbJobPosition>().FirstOrDefault(a => a.Id
== id);
            JobPositionDto result;
            if (dbResult == null)
            {
                result = null;
            }
        }
    }
}

```

```

        else
        {
            result = _mapper.Map<DbJobPosition,
JobPositionDto>(dbResult);
        }
        return result;
    }

    public IPagedResult<JobPositionDto> List(IDataRequest
request, JobPositionFilter filter)
    {
        ISpecification<DbJobPosition> spec = new
TitleJobPositionSpecification(filter.Title);
        return
_context.GetQueryable<DbJobPosition>().MapToPagedResult<DbJobPos
ition, JobPositionDto>(request, _mapper, spec);
    }

    public bool Update(JobPositionDto model)
    {
        DbJobPosition dbModel = _mapper.Map<JobPositionDto,
DbJobPosition>(model);
        DbJobPosition dbJobPosition =
_context.GetQueryable<DbJobPosition>().FirstOrDefault(a => a.Id
== dbModel.Id);
        var dbResult =
_context.GetQueryable<DbJobPosition>().FirstOrDefault(a => a.Id
== model.Id);
        if (dbResult != null && dbJobPosition != null)
        {
            dbResult.Title = dbModel.Title;
            _context.Update(dbResult);
            _context.Save();
            return true;
        }
        else
        {
            return false;
        }
    }
}
}

```

Приложение A.2.4

Файл «ProjectApplicationService.cs»

```

namespace ProjectApplicationService
{
    public class ProjectApplicationService :
IProjectApplicationService
    {
        private readonly IMapper _mapper;
        private readonly IDataContext _context;
        public ProjectApplicationService(IMapper mapper,
[Dependency("DataContext")] IDataContext context)
        {

```

```

        _mapper = mapper;
        _context = context;
    }
}

```

Приложение А.2.5

Файл «ProjectMemberService.cs»

```

namespace ProjectMemberService
{
    public class ProjectMemberService : IProjectMemberService
    {
        private readonly IMapper _mapper;
        private readonly IDataContext _context;
        private readonly IDataContext _identityContext;
        public ProjectMemberService(IMapper mapper,
[Dependency("DataContext")]IDataContext context,
[Dependency("IdentityContext")]IDataContext identityContext)
        {
            _mapper = mapper;
            _context = context;
            _identityContext = identityContext;
        }
        public int AddMember(int projectApplicationId)
        {
            var projectApplication =
            _context.GetQueryable<DbProjectApplication>().FirstOrDefault(a =>
a.Id == projectApplicationId);
            var prj =
            _context.GetQueryable<DbProject>().FirstOrDefault(a => a.Id ==
projectApplication.ProjectId);
            if (!prj.IsOpen)
                return -1;
            ProjectMemberDto dbProjectMember = new
ProjectMemberDto();
            dbProjectMember.ProjectId =
projectApplication.ProjectId;
            dbProjectMember.GuId = projectApplication.GuId;
            var jobPositionDto = _mapper.Map<DbJobPosition,
JobPositionDto>(_context.GetQueryable<DbJobPosition>().FirstOrDe
fault(a => a.Id == projectApplication.JobPosition.Id));
            dbProjectMember.JobPositions = new JobPositionDto[1];
            dbProjectMember.JobPositions[0] = jobPositionDto;
            var specializationDto = _mapper.Map<DbSpecialization,
SpecializationDto>(_context.GetQueryable<DbSpecialization>().Fir
stOrDefault(a => a.Id == projectApplication.Specialization.Id));
            dbProjectMember.Specializations = new
SpecializationDto[1];
            dbProjectMember.Specializations[0]
=specializationDto;
            var result =
            _context.Insert<DbProjectMember>(_mapper.Map<ProjectMemberDto,
DbProjectMember>(dbProjectMember));

```

```

        _context.Save();
        return result.Id;
    }
    public IPagedResult<ProjectOutputDto>
    ListPotentialProjects(IDataRequest request, string employeeGuid)
    {
        var employeeSpecializations =
        _context.GetQueryable<DbEmployeeSpecialization>().ToList().Where
        (a => a.Guid.Equals(employeeGuid));
        List<DbSpecialization> Specializations = new
        List<DbSpecialization>();
        foreach (DbEmployeeSpecialization es in
        employeeSpecializations)
        {
            Specializations.Add(es.Specialization);
        }
        var projects =
        _context.GetQueryable<DbProject>().ToList().Where(a => a.IsOpen);
        List<DbProject> result = new List<DbProject>();
        foreach (DbProject p in projects)
        {
            if (p.Specializations.Count == 0)
            {
                result.Add(p);
            }
            else
            if (p.Specializations.Any(s =>
        Specializations.Contains(s)))
            {
                result.Add(p);
            }
        }
        return
        result.AsQueryable<DbProject>().MapToPagedResult<DbProject,
        ProjectOutputDto>(request, _mapper);
    }
    public int CreateApplication(string Guid, int projectId,
    int jobPosition, int specialization)
    {
        var projectApplication = new ProjectApplicationDto();
        projectApplication.Guid = Guid;
        projectApplication.ProjectId = projectId;
        projectApplication.JobPositionId=jobPosition;
        projectApplication.SpecializationId=specialization;
        var add =
        _context.Insert<DbProjectApplication>(_mapper.Map<ProjectApplica
        tionDto, DbProjectApplication>(projectApplication));
        _context.Save();
        return add.Id;
    }
}
}

```

Приложение А.2.6

Файл «ProjectService.cs»

```
namespace ProjectService
{
    public class ProjectService : IProjectService
    {
        private readonly IMapper _mapper;
        private readonly IDataContext _context;
        public ProjectService(IMapper mapper,
[Dependency("DataContext")]IDataContext context)
        {
            _mapper = mapper;
            _context = context;
        }
        public ProjectOutputDto GetById(int id)
        {
            var project =
            _context.GetQueryable<DbProject>().FirstOrDefault(a => a.Id ==
            id);
            if (project != null)
            {
                var dto = _mapper.Map<DbProject,
ProjectOutputDto>(project);
                return dto;
            }
            return null;
        }
        public IPagedResult<ProjectOutputDto> List(IDataRequest
request)
        {
            var list =
            _context.GetQueryable<DbProject>().MapToPagedResult<DbProject,
ProjectOutputDto>(request, _mapper);
            return list;
        }
        public int Create(ProjectInputDto model)
        {
            var project = _mapper.Map<ProjectInputDto,
DbProject>(model);
            var current = DateTime.Now;
            var date = new DateTime(current.Year, current.Month,
current.Day);
            project.Created = date;
            project.StartDate = model.StartDate == null ? date :
model.StartDate;
            project.EndDate = model.EndDate == null ? date :
model.EndDate;
            var result = _context.Insert(project);
            _context.Save();
            return result.Id;
        }
        public bool Edit(ProjectInputDto model)
```

```

        {
            var project =
            _context.GetQueryable<DbProject>().FirstOrDefault(a => a.Id ==
            model.Id);
            if (project != null)
            {
                var current = DateTime.Now;
                var date = new DateTime(current.Year,
            current.Month, current.Day);
                project.Title = model.Title;
                project.Description = model.Description;
                project.JiraLink = model.JiraLink;
                project.IsOpen = model.IsOpen;
                project.StartDate = model.StartDate == null ? date
            : model.StartDate;
                project.EndDate = model.EndDate == null ? date :
            model.EndDate;
                _context.Update(project);
                _context.Save();
                return true;
            }
            return false;
        }
        public bool Delete(int id)
        {
            var project =
            _context.GetQueryable<DbProject>().FirstOrDefault(a => a.Id ==
            id);
            if (project != null)
            {
                _context.Delete(project);
                _context.Save();
                return true;
            }
            return false;
        }
    }
}

```

Приложение А.2.7

Файл «ProjectSpecializationService.cs»

```

namespace ProjectSpecializationService
{
    public class ProjectSpecializationService :
    IProjectSpecializationService
    {
        private readonly IDataContext _context;
        public
        ProjectSpecializationService([Dependency("DataContext")] IDataCon
        text context)
        {
            _context = context;
        }
    }
}

```

```

        public bool Add(int ProjectId, int SpecializationId)
        {
            var project =
            _context.GetQueryable<DbProject>().FirstOrDefault(a => a.Id ==
            ProjectId);
            var specialization =
            _context.GetQueryable<DbSpecialization>().FirstOrDefault(a
            =>
            a.Id == SpecializationId);
            if (project != null && specialization != null)
            {
                var exist =
                project.Specializations.Contains(specialization);
                if (exist)
                {
                    return false;
                }
                project.Specializations.Add(specialization);
                _context.Save();
                return true;
            }
            return false;
        }
        public bool Delete(int ProjectId, int SpecializationId)
        {
            var project =
            _context.GetQueryable<DbProject>().FirstOrDefault(a => a.Id ==
            ProjectId);
            var specialization =
            _context.GetQueryable<DbSpecialization>().FirstOrDefault(a
            =>
            a.Id == SpecializationId);
            if (project != null && specialization != null)
            {
                var exist =
                project.Specializations.Contains(specialization);
                if (!exist)
                {
                    return false;
                }
                project.Specializations.Remove(specialization);
                _context.Save();
                return true;
            }
            return false;
        }
        public bool Update(int projectId, int specId, int level)
        {
            var project =
            _context.GetQueryable<DbProject>().FirstOrDefault(a => a.Id ==
            projectId);
            var specialization =
            _context.GetQueryable<DbSpecialization>().FirstOrDefault(a
            =>
            a.Id == specId);
            if (project != null && specialization != null)

```

```

        {
            var exist =
project.Specializations.Contains(specialization);
            if (!exist)
            {
                return false;
            }
            project.Specializations.Remove(specialization);
            _context.Save();
            return true;
        }
        return false;
    }
}
}

```

Приложение А.2.8

Файл «ProjectStagesService.cs»

```

namespace ProjectStagesService
{
    public class ProjectStagesService : IProjectStagesService
    {
        private readonly IMapper _mapper;
        private readonly IDataContext _context;
        public ProjectStagesService(IMapper mapper,
[Dependency("DataContext")]IDataContext context)
        {
            _mapper = mapper;
            _context = context;
        }
        public int Create(ProjectStagesDto model)
        {
            var dbModel = _mapper.Map<ProjectStagesDto,
DbProjectStage>(model);
            var dbProject =
_context.GetQueryable<DbProject>().FirstOrDefault(a => a.Id ==
dbModel.ProjectId);
            if (dbModel != null && dbProject != null)
            {
                _context.Insert(dbModel);
                _context.Save();
                return dbModel.Id;
            }
            return -1;
        }
        public bool Delete(int id)
        {
            var dbResult =
_context.GetQueryable<DbProjectStage>().FirstOrDefault(a => a.Id
== id);
            if (dbResult == null)
            {
                return false;
            }
        }
    }
}

```

```

        }
        _context.Delete(dbResult);
        _context.Save();
        return true;
    }
    public ProjectStagesDto GetById(int id)
    {
        var dbResult =
        _context.GetQueryable<DbProjectStage>().FirstOrDefault(a => a.Id
        == id);
        if (dbResult != null)
        {
            var dto = _mapper.Map<DbProjectStage,
            ProjectStagesDto>(dbResult);
            return dto;
        }
        return null;
    }
    public bool Update(ProjectStagesDto model)
    {
        var dbModel =
        _context.GetQueryable<DbProjectStage>().FirstOrDefault(a => a.Id
        == model.Id);
        if (dbModel != null)
        {
            dbModel.Title = model.Title;
            dbModel.Description = model.Description;
            _context.Update(dbModel);
            _context.Save();
            return true;
        }
        return false;
    }
    public IPagedResult<ProjectStagesDto> List(IDataRequest
    request, ProjectStagesFilter filter)
    {
        ISpecification<DbProjectStage> spec = new
        NameProjectStageSpecification(filter.Name).And(new
        DescriptionProjectStageSpecification(filter.Description));
        return
        _context.GetQueryable<DbProjectStage>().MapToPagedResult<DbProje
        ctStage, ProjectStagesDto>(request, _mapper, spec);
    }
}

```

Приложение А.2.9

Файл «SpecializationService.cs»

```

namespace SpecializationService
{
    public class SpecializationService : ISpecializationService
    {
        private readonly IMapper _mapper;
    }
}

```

```

        private readonly IDataContext _context;
        public SpecializationService(IMapper mapper,
[Dependency("DataContext")]IDataContext context)
        {
            _mapper = mapper;
            _context = context;
        }
        public int Create(SpecializationDto model)
        {
            DbSpecialization dbSpecialization =
            _mapper.Map<SpecializationDto, DbSpecialization>(model);
            var result =
            _context.Insert<DbSpecialization>(dbSpecialization);
            _context.Save();
            return result.Id;
        }
        public bool Delete(int id)
        {
            DbSpecialization dbSpecializations =
            _context.GetQueryable<DbSpecialization>().ToList().Find(a => a.Id
            == id);
            if (dbSpecializations != null)
            {
                _context.Delete<DbSpecialization>(dbSpecializations);
                _context.Save();
                return true;
            }
            return false;
        }
        public SpecializationDto GetById(int id)
        {
            if (id == 0)
                return null;
            var res =
            _context.GetQueryable<DbSpecialization>().ToList().Find(a => a.Id
            == id);
            if (res == null)
            {
                return null;
            }
            var result = _mapper.Map<DbSpecialization,
SpecializationDto>(res);
            return result;
        }
        public bool Update(SpecializationDto model)
        {
            DbSpecialization dbModel =
            _mapper.Map<SpecializationDto, DbSpecialization>(model);
            DbSpecialization dbSpecialization =
            _context.GetQueryable<DbSpecialization>().ToList().Find(a => a.Id
            == dbModel.Id);

```

```

        var list =
_context.GetQueryable<DbSpecialization>().ToList();
        var dbResult = list.Find(a => a.Id == model.Id);
        if (dbResult != null && dbSpecialization != null)
        {
            dbResult.Name = dbModel.Name;
            dbResult.Description = dbModel.Description;
            _context.Update(dbResult);
            _context.Save();
            return true;
        }
        else
        {
            return false;
        }
    }

    public IPagedResult<SpecializationDto> List(IDataRequest
request, SpecializationFilter filter)
    {
        ISpecification<DbSpecialization> spec = new
NameSpecializationSpecification(filter.Name);
        return
_context.GetQueryable<DbSpecialization>().MapToPagedResult<DbSpe
cialization, SpecializationDto>(request, _mapper, spec);
    }
}

```

Приложение A.2.10

Файл «EmployeeAddDto.cs»

```

namespace TeamHunter.DataContracts
{
    public class EmployeeAddDto
    {
        public string GuId { get; set; }
    }
}

```

Приложение A.2.11

Файл «EmployeeDto.cs»

```

namespace TeamHunter.DataContracts
{
    public class EmployeeDto
    {
        public string Id { get; set; }
        public string Username { get; set; }
        public string FirstName { get; set; }
        public string LastName { get; set; }
        public string Email { get; set; }
        public string Phone { get; set; }
        public JobPositionDto[] JobPositions { get; set; }
        public SpecializationDto[] Specializations { get; set; }
    }
}

```

```

        public ProjectApplicationDto[] ProjectApplications { get;
set; }
        public ProjectOutputDto CurrentProject { get; set; }
        public ProjectOutputDto[] ProjectHistory { get; set; }
        public
EmployeeSpecializationDto[]
employeeSpecializations { get; set; }
    }
}

```

Приложение A.2.12

Файл «EmployeeJobPositionDto.cs»

```

namespace TeamHunter.DataContracts
{
    public class EmployeeJobPositionDto
    {
        public string GuId { get; set; }
        public int JobPositionId { get; set; }
    }
}

```

Приложение A.2.13

Файл «EmployeeSpecializationDto.cs»

```

namespace TeamHunter.DataContracts
{
    public class EmployeeSpecializationDto
    {
        public string GuId { get; set; }
        public int SpecId { get; set; }
        public int Level { get; set; }
    }
}

```

Приложение A.2.14

Файл «JobPositionDto.cs»

```

namespace TeamHunter.DataContracts
{
    public class JobPositionDto
    {
        public int Id { get; set; }
        public string Title { get; set; }
    }
}

```

Приложение A.2.15

Файл «PageInfo.cs»

```

namespace TeamHunter.DataContracts
{
    public class PageInfo<T>
    {
        public int pageSize { get; set; }
        public int pageNumber { get; set; }
    }
}

```

```

        public IEnumerable<T> pageItems { get; set; }
    }
}

```

Приложение А.2.16

Файл «ProjectApplicationDto.cs»

```

namespace TeamHunter.DataContracts
{
    public class ProjectApplicationDto
    {
        public int Id { get; set; }
        public string GuId { get; set; }
        public int ProjectId { get; set; }
        public int JobPositionId { get; set; }
        public int SpecializationId { get; set; }
    }
}

```

Приложение А.2.17

Файл «ProjectDto.cs»

```

namespace TeamHunter.DataContracts
{
    public class ProjectDto
    {
        public int Id { get; set; }
        public DateTime Created { get; set; }
        public string Title { get; set; }
        public string Description { get; set; }
        public string JiraLink { get; set; }
        public bool IsOpen { get; set; }
        public DateTime? StartDate { get; set; }
        public DateTime? EndDate { get; set; }
    }
}

```

Приложение А.2.18

Файл «ProjectInputDto.cs»

```

namespace TeamHunter.DataContracts
{
    public class ProjectInputDto
    {
        public int Id { get; set; }
        public DateTime Created { get; set; }
        public string Title { get; set; }
        public string Description { get; set; }
        public string JiraLink { get; set; }
        public bool IsOpen { get; set; }
        public DateTime? StartDate { get; set; }
        public DateTime? EndDate { get; set; }
    }
}

```

Приложение А.2.19

Файл «ProjectMemberDto.cs»

```
namespace TeamHunter.DataContracts
{
    public class ProjectMemberDto
    {
        public int Id { get; set; }
        public int ProjectId { get; set; }
        public virtual DbProject Project { get; set; }
        public string GuId { get; set; }
        public JobPositionDto[] JobPositions { get; set; }
        public SpecializationDto[] Specializations { get; set; }
    }
}
```

Приложение А.2.20

Файл «ProjectOutputDto.cs»

```
namespace TeamHunter.DataContracts
{
    public class ProjectOutputDto
    {
        public int Id { get; set; }
        public DateTime Created { get; set; }
        public string Title { get; set; }
        public string Description { get; set; }
        public bool IsOpen { get; set; }
        public DateTime? StartDate { get; set; }
        public DateTime? EndDate { get; set; }
        public ProjectStagesDto[] Stages { get; set; }
    }
}
```

Приложение А.2.21

Файл «ProjectSpecializationDto.cs»

```
namespace TeamHunter.DataContracts
{
    public class ProjectSpecializationDto
    {
        public int ProjectId { get; set; }
        public int SpecializationId { get; set; }
    }
}
```

Приложение А.2.22

Файл «ProjectStagesDto.cs»

```
namespace TeamHunter.DataContracts
{
    public class ProjectStagesDto
    {
        public int Id { get; set; }
    }
}
```

```

        public int ProjectId { get; set; }
        public string Title { get; set; }
        public string Description { get; set; }
    }
}

```

Приложение A.2.23

Файл «SpecializationDto.cs»

```

namespace TeamHunter.DataContracts
{
    public class SpecializationDto
    {
        public int Id { get; set; }
        public string Name { get; set; }
        public string Description { get; set; }
    }
}

```

Приложение A.2.24

Файл «DbEmployee.cs»

```

namespace TeamHunter.Data
{
    public class DbEmployee
    {
        public int Id { get; set; }
        public string GuId { get; set; }
        public string FirstName { get; set; }
        public string LastName { get; set; }
        public string Email { get; set; }
        public string Phone { get; set; }
        public DbJobPosition JobPosition { get; set; }
        public virtual ICollection<DbSpecialization>
Specializations { get; set; }
        public virtual ICollection<DbProjectMember>
ProjectMembers { get; set; }
    }
}

```

Приложение A.2.25

Файл «DbEmployeeJobPosition.cs»

```

namespace TeamHunter.Data
{
    public class DbEmployeeJobPosition
    {
        public int Id { get; set; }
        public string GuId { get; set; }
        public int JobPositionId { get; set; }
        public virtual DbJobPosition JobPosition { get; set; }
    }
}

```

Приложение А.2.26

Файл «DbEmployeeSpecialization.cs»

```
namespace TeamHunter.Data
{
    public class DbEmployeeSpecialization
    {
        public int Id { get; set; }
        public string GuId { get; set; }
        public int SpecializationId { get; set; }
        public int? Level { get; set; }
        public virtual DbSpecialization Specialization { get; set; }
    }
}
```

Приложение А.2.27

Файл «DbJobPosition.cs»

```
namespace TeamHunter.Data
{
    public class DbJobPosition
    {
        public int Id { get; set; }
        public string Title { get; set; }
        public virtual ICollection<DbEmployeeJobPosition>
EmployeesJobPosition { get; set; }
        public virtual ICollection<DbProjectApplication>
ProjectApplication { get; set; }
    }
}
```

Приложение А.2.28

Файл «DbProject.cs»

```
namespace TeamHunter.Data
{
    public class DbProject
    {
        public int Id { get; set; }
        public DateTime Created { get; set; }
        public string Title { get; set; }
        public string Description { get; set; }
        public string JiraLink { get; set; }
        public bool IsOpen { get; set; }
        public DateTime? StartDate { get; set; }
        public DateTime? EndDate { get; set; }
        public virtual ICollection<DbSpecialization>
Specializations { get; set; }
        public virtual ICollection<DbProjectStage>
ProjectStages { get; set; }
        public virtual ICollection<DbProjectMember> Members {
get; set; }    }    }
```

Приложение А.2.29

Файл «DbProjectApplication.cs»

```
namespace TeamHunter.Data
{
    public class DbProjectApplication
    {
        public int Id { get; set; }
        public string GuId { get; set; }
        public int ProjectId { get; set; }
        public int JobPositionId { get; set; }
        public int SpecializationId { get; set; }
        public virtual DbJobPosition JobPosition { get; set; }
        public virtual DbSpecialization Specialization { get;
set; }
    }
}
```

Приложение А.2.30

Файл «DbProjectMember.cs»

```
namespace TeamHunter.Data
{
    public class DbProjectMember
    {
        public int Id { get; set; }
        public int ProjectId { get; set; }
        public virtual DbProject Project { get; set; }
        public string GuId { get; set; }
        public virtual ICollection<DbSpecialization>
Specializations { get; set; }
        public virtual ICollection<DbJobPosition> JobPositions
{ get; set; }
    }
}
```

Приложение А.2.31

Файл «DbProjectStages.cs»

```
namespace TeamHunter.Data
{
    public class DbProjectStages
    {
        public int Id { get; set; }
        public int ProjectId { get; set; }
        public string Title { get; set; }
        public string Description { get; set; }
        public virtual DbProject Project { get; set; }
    }
}
```

Приложение А.2.32

Файл «DbSpecialization.cs»

```
namespace TeamHunter.Data
{
    public class DbSpecialization
    {
        public int Id { get; set; }
        public string Name { get; set; }
        public string Description { get; set; }
        public virtual ICollection<DbProject> Projects { get;
set; }
        public virtual ICollection<DbEmployeeSpecialization>
EmployeeSpecializations { get; set; }
        public virtual ICollection<DbProjectApplication>
ProjectApplication { get; set; }
    }
}
```

Приложение А.2.33

Файл «Context.cs»

```
namespace TeamHunter.DataAccess
{
    public class Context : DbContext
    {
        public Context()
            : base("DefaultConnection")
        {
        }
        public Context(string nameOrConnectionString) :
base(nameOrConnectionString)
        {
        }
        public virtual IDbSet<DbSpecialization> Specializations
{ get; set; }
        public virtual IDbSet<DbEmployeeSpecialization>
EmployeeSpecializations { get; set; }
        public virtual IDbSet<DbJobPosition> JobPositions { get;
set; }
        public virtual IDbSet<DbProject> Projects { get; set; }
        public virtual IDbSet<DbProjectMember> ProjectMembers {
get; set; }
        public virtual IDbSet<DbProjectStage> ProjectStages { get;
set; }
        protected override void OnModelCreating(DbModelBuilder
modelBuilder)
        {
            modelBuilder.Configurations.Add(new
SpecializationConfig());
            modelBuilder.Configurations.Add(new
EmployeeSpecializationConfig());
        }
    }
}
```

```

        modelBuilder.Configurations.Add(new
EmployeeJobPositionConfig());
        modelBuilder.Configurations.Add(new
JobPositionConfig());
        modelBuilder.Configurations.Add(new
ProjectConfig());
        modelBuilder.Configurations.Add(new
ProjectMemberConfig());
        modelBuilder.Configurations.Add(new
ProjectStagesConfig());
    }
}
}

```

Приложение А.2.34

Файл «DataContext.cs»

```

namespace TeamHunter.DataAccess
{
    public class DataContext : Context, IDataContext
    {
        public DataContext()
        {
        }

        public DataContext(string nameOrConnectionString) :
base(nameOrConnectionString)
        {
        }

        public IQueryable<T> GetQueryable<T>(bool trackChanges
= true, bool disabled = false) where T : class, new()
        {
            return GetQueryable<T>(null, trackChanges,
disabled);
        }

        public IQueryable<T> GetQueryable<T>(Expression<Func<T,
bool>> expression, bool trackChanges = true, bool disabled = false,
bool deleted = false) where T : class, new()
        {
            var query = GetQueryableNonAudit(expression,
trackChanges, disabled, deleted);
            return query;
        }

        public                                IQueryable<T>
GetQueryableNonAudit<T>(Expression<Func<T, bool>> expression,
bool trackChanges = true, bool disabled = false, bool deleted =
false) where T : class, new()
        {
            var query = trackChanges
                ? Set<T>().AsQueryable()
                : Set<T>().AsNoTracking();
            if (expression != null)
            {
                query = query.Where(expression);
            }
        }
    }
}

```

```

        return query;
    }
    public T Delete<T>(T item) where T : class, new()
    {
        return Set<T>().Remove(item);
    }
    public T Insert<T>(T item) where T : class, new()
    {
        return Set<T>().Add(item);
    }
    public T Update<T>(T item) where T : class, new()
    {
        return PerformAction(item,
System.Data.Entity.EntityState.Modified);
    }
    public int Save(string userId = "")
    {
        int changes = base.SaveChanges();
        return changes;
    }
    public async Task SaveAsync()
    {
        await this.SaveChangesAsync();
    }
    protected virtual TItem PerformAction<TItem>(TItem item,
System.Data.Entity.EntityState entityState) where TItem : class,
new()
    {
        Entry(item).State = entityState;
        return item;
    }
}
}

```

Приложение A.2.35

Файл «EmployeeController.cs»

```

namespace TeamHunter.Controllers
{
    [Authorize]
    public class EmployeeController : ApiController
    {
        private readonly IEmployeeService _employeeService;
        private readonly IEmployeeSpecializationService
_employeeSpecializationService;
        private readonly IEmployeeJobPositionService
_employeeJobPositionService;
        private IAuthenticationManager AuthenticationManager
        {
            get
            {
                return
HttpContext.Current.GetOwinContext().Authentication;
            }
        }
    }
}

```

```

    }
    public EmployeeController(IEmployeeService service,
        IEmployeeSpecializationService empSpecService,
        IEmployeeJobPositionService employeeJobPositionService)
    {
        _employeeService = service;
        _employeeSpecializationService = empSpecService;
        _employeeJobPositionService =
employeeJobPositionService;
    }
    [Route("Employees")]
    [HttpGet]
    public IHttpActionResult List([FromUri]DataRequest
request, [FromUri]EmployeeFilter filter)
    {
        if (request == null)
        {
            request = new DataRequest();
        }
        if (filter == null)
        {
            filter = new EmployeeFilter();
        }
        var list = _employeeService.List(request, filter);
        ResultDto response = ResultDto.Ok(list);
        return Ok(response);
    }
    [Route("Employees/{id}")]
    [HttpGet]
    public IHttpActionResult GetEmployee(string id)
    {
        ResultDto response;
        var model = _employeeService.GetById(id);
        response = ResultDto.Ok(model);
        if (model == null)
            response = ResultDto.Fail("Не удалось получить
информацию о сотруднике");
        return Ok(response);
    }
    [Authorize(Roles = "user")]
    [Route("Employee/Specializations/{id}")]
    [HttpPost]
    public IHttpActionResult UserAddSpecialization(int id)
    {
        var claims = AuthenticationManager.User.Claims;
        var claim = claims.FirstOrDefault(a =>
a.Type.Contains("nameidentifier"));
        var guid = claim.Value;
        ResultDto response = ResultDto.Ok(claims);
        if (id < 1)
        {
            response = ResultDto.Fail("Объект не задан");
            return Ok(response);
        }
    }

```

```

        }
        var result =
_employeeSpecializationService.AddSpecialization(guid, id);
        if (result)
        {
            response = ResultDto.Ok(true);
        }
        else
        {
            response = ResultDto.Fail("Не удалось добавить
специализацию");
        }
        return Ok(response);
    }
    [Authorize(Roles = "user")]
    [Route("Employee/Specializations/{id}")]
    [HttpDelete]
    public IHttpActionResult UserDeleteSpecialization(int id)
    {
        var claims = AuthenticationManager.User.Claims;
        var claim = claims.FirstOrDefault(a =>
a.Type.Contains("nameidentifier"));
        var guid = claim.Value;
        ResultDto response = ResultDto.Ok(claims);
        if (id < 1)
        {
            response = ResultDto.Fail("Объект не задан");
            return Ok(response);
        }
        var result =
_employeeSpecializationService.DeleteSpecialization(guid, id);
        if (result)
        {
            response = ResultDto.Ok(true);
        }
        else
        {
            response = ResultDto.Fail("Не удалось удалить
специализацию");
        }
        return Ok(response);
    }
    [Authorize(Roles = "admin")]
    [Route("Employee/Specializations")]
    [HttpPost]
    public IHttpActionResult
AdminAddSpecialization(EmployeeSpecializationDto model)
    {
        ResultDto response;
        if (model == null ||
string.IsNullOrEmpty(model.GuId) || model.SpecId < 1)
        {
            response = ResultDto.Fail("Объект не задан");

```

```

        return Ok(response);
    }
    var result =
_employeeSpecializationService.AddSpecialization(model.GuId,
model.SpecId);
    if (result)
    {
        response = ResultDto.Ok(true);
    }
    else
    {
        response = ResultDto.Fail("Не удалось добавить
специализацию сотруднику");
    }
    return Ok(response);
}
[Authorize(Roles = "admin")]
[Route("Employee/Specializations")]
[HttpDelete]
public IHttpActionResult
AdminDeleteSpecialization(EmployeeSpecializationDto model)
{
    ResultDto response;
    if (model == null ||
string.IsNullOrEmpty(model.GuId) || model.SpecId < 1)
    {
        response = ResultDto.Fail("Объект не задан");
        return Ok(response);
    }
    var result =
_employeeSpecializationService.DeleteSpecialization(model.GuId,
model.SpecId);
    if (result)
    {
        response = ResultDto.Ok(true);
    }
    else
    {
        response = ResultDto.Fail("Не удалось удалить
специализацию у сотрудника");
    }
    return Ok(response);
}
[Authorize(Roles = "admin")]
[Route("Employee/JobPosition")]
[HttpPut]
public IHttpActionResult
AdminAddJobPosition(EmployeeJobPositionDto model)
{
    ResultDto response;
    var result =
_employeeJobPositionService.AddJobPosition(model.GuId,
model.JobPositionId);

```

```

        if (result)
        {
            response = ResultDto.Ok(true);
        }
        else
        {
            response = ResultDto.Fail("Не удалось добавить
должность сотруднику");
        }
        return Ok(response);
    }
    [Authorize(Roles = "admin")]
    [Route("Employee/JobPosition")]
    [HttpDelete]
    public IActionResult
AdminDeleteJobPosition(EmployeeJobPositionDto model)
    {
        ResultDto response;
        var result =
_employeeJobPositionService.DeleteJobPosition(model.GuId,
model.JobPositionId);
        if (result)
        {
            response = ResultDto.Ok(true);
        }
        else
        {
            response = ResultDto.Fail("Не удалось удалить
должность у сотрудника");
        }
        return Ok(response);
    }
    [Authorize(Roles = "user")]
    [Route("Employee/AddSpecialization")]
    [HttpPost]
    public IActionResult
EditSpecialization(EmployeeSpecializationDto model)
    {
        ResultDto response;
        var result =
_employeeSpecializationService.UpdateSpecialization(model.GuId,
model.SpecId, model.Level);
        if (!result)
            response = ResultDto.Fail("Не удалось обновить
данные");
        response = ResultDto.Ok(result);
        return Ok(response);
    }
}
}

```

Приложение А.2.36

Файл «JobPositionController.cs»

```
namespace TeamHunter.Controllers
{
    [Authorize]
    public class JobPositionController : ApiController
    {
        private readonly IJobPositionService _service;
        public JobPositionController(IJobPositionService service)
        {
            _service = service;
        }
        [Route("JobPositions")]
        [HttpGet]
        public IHttpActionResult List([FromUri]DataRequest request, [FromUri]JobPositionFilter filter)
        {
            if (request == null)
            {
                request = new DataRequest();
            }
            if (filter == null)
            {
                filter = new JobPositionFilter();
            }
            var list = _service.List(request, filter);
            ResultDto response = ResultDto.Ok(list);
            return Ok(response);
        }
        [Route("JobPositions/{id}")]
        [HttpGet]
        public IHttpActionResult GetJobPosition(int id)
        {
            ResultDto response;
            var model = _service.GetById(id);
            response = ResultDto.Ok(model);
            if (model == null)
                response = ResultDto.Fail("Не удалось получить информацию о должности");
            return Ok(response);
        }
        [Authorize(Roles = "admin")]
        [Route("JobPositions")]
        [HttpPut]
        public IHttpActionResult Create(JobPositionDto model)
        {
            ResultDto response;
            var validator = new JobPositionValidator();
            var validate = validator.Validate(model);
            if (!validate.IsValid)
            {

```

```

        var errors = string.Join(", ",
validate.Errors.Select(a => a.ErrorMessage));
        response = ResultDto.Fail(string.Format("Ошибка
валидации данных! {0}", errors));
        return Ok(response);
    }
    var result = _service.Create(model);
    response = ResultDto.Ok(result);
    if (result < 1)
        response = ResultDto.Fail("Не удалось добавить
специализацию");
    return Ok(response);
}
[Authorize(Roles = "admin")]
[Route("JobPosition")]
[HttpPost]
public IHttpActionResult EditJobPosition(JobPositionDto
model)
{
    ResultDto response;
    var validator = new JobPositionValidator();
    var validate = validator.Validate(model);
    if (!validate.IsValid)
    {
        var errors = string.Join(", ",
validate.Errors.Select(a => a.ErrorMessage));
        response = ResultDto.Fail(string.Format("Ошибка
валидации данных! {0}", errors));
        return Ok(response);
    }
    var result = _service.Update(model);
    if (!result)
        response = ResultDto.Fail("Не удалось обновить
данные");
    response = ResultDto.Ok(result);
    return Ok(response);
}
[Authorize(Roles = "admin")]
[Route("JobPositions/{id}")]
[HttpDelete]
public IHttpActionResult Delete(int id)
{
    ResultDto response;
    var result = _service.Delete(id);
    response = ResultDto.Ok(result);
    if (!result)
        response = ResultDto.Fail("Не удалось удалить
должность, т.к. она не существует");
    return Ok(response);
}
}
}

```

Приложение А.2.37

Файл «ProjectController.cs»

```
namespace TeamHunter.Controllers
{
    [Authorize]
    public class ProjectController : ApiController
    {
        private readonly IProjectService _projectService;
        private readonly IProjectSpecializationService
        _projectSpecializationService;
        private readonly IProjectMemberService
        _projectMemberService;
        private readonly IEmployeeService _employeeservice;
        public ProjectController(IProjectService projectService,
        IProjectSpecializationService projectSpecializationService,
        IProjectMemberService projectMemberService, IEmployeeService
        employeeservice)
        {
            _projectService = projectService;
            _projectSpecializationService =
        projectSpecializationService;
            _projectMemberService = projectMemberService;
            _employeeservice = employeeservice;
        }
        [Route("Projects")]
        [HttpGet]
        public IHttpActionResult List([FromUri]DataRequest
        request)
        {
            if (request == null)
            {
                request = new DataRequest();
            }
            var list = _projectService.List(request);
            var response = ResultDto.Ok(list);
            return Ok(response);
        }
        [Route("Projects/{id}")]
        [HttpGet]
        public IHttpActionResult GetById(int id)
        {
            ResultDto response;
            if (id < 1)
            {
                response = ResultDto.Fail("Id не может быть меньше
        1");
                return Ok(response);
            }
            var result = _projectService.GetById(id);
            response = ResultDto.Ok(result);
            return Ok(response);
        }
    }
}
```

```

[Authorize(Roles = "admin")]
[Route("Project")]
[HttpPut]
public IActionResult Create(ProjectInputDto model)
{
    ResultDto response;
    var validator = new ProjectValidator();
    var validate = validator.Validate(model);
    if (!validate.IsValid)
    {
        var errors = string.Join(", ",
validate.Errors.Select(a => a.ErrorMessage));
        response = ResultDto.Fail(string.Format("Ошибка
валидации данных! {0}", errors));
        return Ok(response);
    }
    var result = _projectService.Create(model);
    response = ResultDto.Ok(result);
    return Ok(response);
}
[Authorize(Roles = "admin")]
[Route("Project")]
[HttpPost]
public IActionResult Update(ProjectInputDto model)
{
    ResultDto response;
    var validator = new ProjectValidator();
    var validate = validator.Validate(model);
    if (!validate.IsValid)
    {
        var errors = string.Join(", ",
validate.Errors.Select(a => a.ErrorMessage));
        response = ResultDto.Fail(string.Format("Ошибка
валидации данных! {0}", errors));
        return Ok(response);
    }
    var result = _projectService.Edit(model);
    response = result ? ResultDto.Ok(result) :
ResultDto.Fail("Не удалось обновить проект");
    return Ok(response);
}
[Authorize(Roles = "admin")]
[Route("Project/Specializations")]
[HttpPost]
public IActionResult
AddSpecialization(ProjectSpecializationDto model)
{
    ResultDto response;
    if (model == null || model.ProjectId < 1 ||
model.SpecializationId < 1)
    {
        response = ResultDto.Fail("Одно из значений
указано некорректно");
    }
}

```

```

        return Ok(response);
    }
    var result =
        _projectSpecializationService.Add(model.ProjectId,
        model.SpecializationId);
    response = result ? ResultDto.Ok(result) :
    ResultDto.Fail("Не удалось добавить специализацию к проекту");
    return Ok(response);
}
[Authorize(Roles = "admin")]
[Route("Project/Specializations")]
[HttpDelete]
public IHttpActionResult DeleteSpecialization(ProjectSpecializationDto model)
{
    ResultDto response;
    if (model == null || model.ProjectId < 1 ||
    model.SpecializationId < 1)
    {
        response = ResultDto.Fail("Одно из значений
указано некорректно");
        return Ok(response);
    }
    var result =
        _projectSpecializationService.Delete(model.ProjectId,
        model.SpecializationId);
    response = result ? ResultDto.Ok(result) :
    ResultDto.Fail("Не удалось удалить специализацию у проекта");
    return Ok(response);
}
[Authorize(Roles = "admin")]
[Route("Projects/{id}")]
[HttpDelete]
public IHttpActionResult Delete(int id)
{
    ResultDto response;
    if (id < 1)
    {
        response = ResultDto.Fail("Id не может быть меньше
1");
        return Ok(response);
    }
    var result = _projectService.Delete(id);
    response = result ? ResultDto.Ok(result) :
    ResultDto.Fail("Не удалось удалить проект");
    return Ok(response);
}
[Authorize(Roles = "admin")]
[Route("Project/{id}/Applications")]
[HttpGet]
public IHttpActionResult WatchProjectApplications([FromUri]DataRequest request, int id)
{

```

```

        ResultDto response;
        if (id < 0)
        {
            response = ResultDto.Fail("Некорректный ввод
данных");
            return Ok(response);
        }
        var list =
_employeeService.WatchProjectApplications(request, id);
        response = ResultDto.Ok(list);
        return Ok(response);
    }
}

```

Приложение А.2.38

Файл «ProjectMembersContioller.cs»

```

namespace TeamHunter.Controllers
{
    [Authorize]
    public class ProjectMembersController : ApiController
    {
        private readonly IProjectMemberService
_projectMemberService;
        private readonly IEmployeeService _employeeService;
        public ProjectMembersController(IProjectMemberService
projectMemberService, IEmployeeService employeeService)
        {
            _projectMemberService = projectMemberService;
            _employeeService = employeeService;
        }
        private IAuthenticationManager AuthenticationManager
        {
            get
            {
                return
HttpContext.Current.GetOwinContext().Authentication;
            }
        }
        [Authorize(Roles = "admin")]
        [Route("Projects/{id}/PotentialMembers")]
        [HttpGet]
        public IHttpActionResult
ListPotentialMembers([FromUri]DataRequest request, int id)
        {
            if (request == null)
            {
                request = new DataRequest();
            }
            var e = _employeeService.List(request, null);
            var list =
_employeeService.ListPotentialMembers(request, id);
            var response = ResultDto.Ok(list);
        }
    }
}

```

```

        return Ok(response);
    }
    [Authorize(Roles = "user")]
    [Route("Employee/PotentialProjects")]
    [HttpGet]
    public IHttpActionResult ListPotentialProjects([FromUri]DataRequest request)
    {
        var claims = AuthenticationManager.User.Claims;
        var claim = claims.FirstOrDefault(a =>
a.Type.Contains("nameidentifier"));
        var guid = claim.Value;
        ResultDto response = ResultDto.Ok(claims);
        if (request == null)
        {
            request = new DataRequest();
        }
        var list =
_projectMemberService.ListPotentialProjects(request, guid);
        response = ResultDto.Ok(list);
        return Ok(response);
    }
    [Authorize(Roles = "user")]

    [Route("Projects/{id}/CreateApplication/{jobposition}/{specializ
ation}")]
    [HttpPut]
    public IHttpActionResult CreateApplication(int id, int
jobPosition, int specialization)
    {
        var claims = AuthenticationManager.User.Claims;
        var claim = claims.FirstOrDefault(a =>
a.Type.Contains("nameidentifier"));
        var guid = claim.Value;
        ResultDto response = ResultDto.Ok(claims);
        if (id < 0 || jobPosition < 0 || specialization<0)
        {
            response = ResultDto.Fail("Неверные параметры");
            return Ok(response);
        }
        var addId =
_projectMemberService.CreateApplication(guid, id, jobPosition,
specialization);
        response = ResultDto.Ok(addId);
        if (addId < 1)
            response = ResultDto.Fail("Не удалось добавить
участника проекта");
        return Ok(response);
    }
    [Authorize(Roles = "admin")]
    [Route("Projects/AddMember/{projectApplication}")]
    [HttpPut]

```

```

        public IHttpActionResult AddMember(int
projectApplication)
        {
            ResultDto response;
            if (projectApplication<0)
            {
                response = ResultDto.Fail("Неверные параметры");
                return Ok(response);
            }
            var addId =
            _projectMemberService.AddMember(projectApplication);
            response = ResultDto.Ok(addId);
            if (addId < 1)
                response = ResultDto.Fail("Не удалось добавить
участника проекта");
            return Ok(response);
        }
    }
}

```

Приложение А.2.39

Файл «ProjectStagesController.cs»

```

namespace TeamHunter.Controllers
{
    [Authorize]
    public class ProjectStagesController : ApiController
    {
        private readonly IProjectStagesService _service;

        public ProjectStagesController(IProjectStagesService
service)
        {
            _service = service;
        }

        [Route("Project/Stages")]
        [HttpGet]
        public IHttpActionResult List([FromUri]DataRequest
request, [FromUri]ProjectStagesFilter filter)
        {
            if (request == null)
            {
                request = new DataRequest();
            }

            if (filter == null)
            {
                filter = new ProjectStagesFilter();
            }

            var list = _service.List(request, filter);
            ResultDto response = ResultDto.Ok(list);
            return Ok(response);
        }
    }
}

```

```

    }

    [Route("Project/Stages/{id}")]
    [HttpGet]
    public IActionResult GetProjectStage(int id)
    {
        ResultDto response;

        if (id < 1)
        {
            response = ResultDto.Fail("ID может быть только
положительным числом");
        }

        else
        {
            var model = _service.GetById(id);
            response = ResultDto.Ok(model);
            if (model == null)
            {
                response = ResultDto.Fail("Не удалось
получить информацию о стадии проекта");
            }
        }

        return Ok(response);
    }
    [Authorize(Roles = "admin")]
    [Route("Project/Stage")]
    [HttpPut]
    public IActionResult Create(ProjectStagesDto model)
    {
        ResultDto response;
        var validator = new ProjectStagesValidator();
        var validate = validator.Validate(model);

        if (!validate.IsValid)
        {
            var errors = string.Join(", ",
validate.Errors.Select(a => a.ErrorMessage));
            response = ResultDto.Fail(string.Format("Ошибка
валидации данных! {0}", errors));
            return Ok(response);
        }

        var result = _service.Create(model);

        if (result > 0)
        {
            response = ResultDto.Ok(result);
        }

        else
    
```

```

        {
            response = ResultDto.Fail("Не удалось добавить
стадию проекта");
        }

        return Ok(response);
    }
    [Authorize(Roles = "admin")]
    [Route("Project/Stage")]
    [HttpPost]
    public IHttpActionResult Edit(ProjectStagesDto model)
    {
        ResultDto response;
        var validator = new ProjectStagesValidator();
        var validate = validator.Validate(model);

        if (!validate.IsValid)
        {
            var errors = string.Join(", ",
validate.Errors.Select(a => a.ErrorMessage));
            response = ResultDto.Fail(string.Format("Ошибка
валидации данных! {0}", errors));
            return Ok(response);
        }

        var result = _service.Update(model);

        if (result)
        {
            response = ResultDto.Ok(result);
        }

        else
        {
            response = ResultDto.Fail("Не удалось обновить
стадию проекта");
        }

        return Ok(response);
    }
    [Authorize(Roles = "admin")]
    [Route("Project/Stages/{id}")]
    [HttpDelete]
    public IHttpActionResult Delete(int id)
    {
        ResultDto response;

        if (id < 1)
        {
            response = ResultDto.Fail("ID может быть только
положительным числом");
        }
    }

```

```

        else
        {
            var result = _service.Delete(id);

            if (result)
            {
                response = ResultDto.Ok(result);
            }

            else
            {
                response = ResultDto.Fail("Стадия проекта с
даннным ID не найдена");
            }
        }

        return Ok(response);
    }
}
}

```

Приложение А.2.40

Файл «SpecializationsController.cs»

```

namespace TeamHunter.Controllers
{
    [Authorize]
    public class SpecializationController : ApiController
    {
        private readonly ISpecializationService _service;
        public SpecializationController(ISpecializationService
service)
        {
            _service = service;
        }

        [HttpGet]
        [Route("Specializations")]
        public IHttpActionResult List([FromUri]DataRequest
request, [FromUri]SpecializationFilter filter)
        {
            if (request == null)
            {
                request = new DataRequest();
            }

            if (filter == null)
            {
                filter = new SpecializationFilter();
            }

            var list = _service.List(request, filter);
            ResultDto response = ResultDto.Ok(list);

```

```

        return Ok(response);
    }
    [Route("Specializations/{id}")]
    [HttpGet]
    public IActionResult GetSpecialisation(int id)
    {
        ResultDto response;
        var model = _service.GetById(id);
        response = ResultDto.Ok(model);
        if (model == null)
            response = ResultDto.Fail("Не удалось получить
информацию о специализации");
        return Ok(response);
    }

    [Authorize(Roles = "admin")]
    [Route("Specialization")]
    [HttpPut]
    public IActionResult Create(SpecializationDto
model)
    {
        ResultDto response;
        var validator = new SpecializationValidator();
        var validate = validator.Validate(model);
        if (!validate.IsValid)
        {
            var errors = string.Join(", ",
validate.Errors.Select(a => a.ErrorMessage));
            response =
ResultDto.Fail(string.Format("Ошибка валидации данных
специализации! {0}", errors));
            return Ok(response);
        }

        var addId = _service.Create(model);
        response = ResultDto.Ok(addId);
        if (addId < 1)
            response = ResultDto.Fail("Не удалось добавить
специализацию");
        return Ok(response);
    }
    [Authorize(Roles = "admin")]
    [Route("Specialization")]
    [HttpPost]
    public IActionResult
EditSpecialization(SpecializationDto model)
    {
        ResultDto response;
        var validator = new SpecializationValidator();
        var validate = validator.Validate(model);
        if (!validate.IsValid)
        {

```

```

        var errors = string.Join(", ",
validate.Errors.Select(a => a.ErrorMessage));
        response = ResultDto.Fail(string.Format("Ошибка
валидации данных специализации! {0}", errors));
        return Ok(response);
    }
    var result = _service.Update(model);
    response = ResultDto.Ok(result);
    if (!result)
        response = ResultDto.Fail("Не удалось обновить
специализацию");
    return Ok(response);
}
[Authorize(Roles = "admin")]
[Route("Specializations/{id}")]
[HttpDelete]
public IHttpActionResult Delete(int id)
{
    ResultDto response;
    var result = _service.Delete(id);
    response = ResultDto.Ok(result);
    if (!result)
        response = ResultDto.Fail("Не удалось удалить
специализацию, т.к. она не существует");
    return Ok(response);
}
}
}

```

Приложение A.2.41

Файл «ApplicationOAuthBearerAuthenticationProvider.cs»

```

namespace TeamHunter.Providers
{
    public class ApplicationOAuthBearerAuthenticationProvider
        : OAuthBearerAuthenticationProvider
    {
        public override Task
RequestToken(OAuthRequestTokenContext context)
        {
            if (context == null) throw new
ArgumentNullException("context");

            if
(!context.Request.Headers.ContainsKey("Authorization"))
            {
                var cookies = context.Request.Cookies;
                var cookie = cookies.FirstOrDefault(c => c.Key
== IdentityPaths.TokenName);
                if
(!cookie.Equals(default(KeyValuePair<string, string>)))
                {
                    context.Token = cookie.Value;

```

```

        }
    }
    return Task.FromResult<object>(true);
}
}
}

```

Приложение А.2.42

Файл «ApplicationOAuthProvider.cs»

```

namespace TeamHunter.Providers
{
    public class ApplicationOAuthProvider :
        OAuthAuthorizationServerProvider
    {
        private readonly string _publicClientId;

        public ApplicationOAuthProvider(string publicClientId)
        {
            if (publicClientId == null)
            {
                throw new
                ArgumentException("publicClientId");
            }

            _publicClientId = publicClientId;
        }

        public override Task
        ValidateClientRedirectUri(OAuthValidateClientRedirectUriContext
        context)
        {
            if (context.ClientId == _publicClientId)
            {
                Uri expectedRootUri = new
                Uri(context.Request.Uri, "/");

                if (expectedRootUri.AbsoluteUri ==
                context.RedirectUri)
                {
                    context.Validated();
                }
                else if (context.ClientId == "web")
                {
                    var expectedUri = new
                    Uri(context.Request.Uri, "/");

                    context.Validated(expectedUri.AbsoluteUri);
                }
            }
            return Task.FromResult<object>(null);
        }
    }
}

```

Приложение А.2.43

Файл «BearerOnCookieAuthentication.cs»

```
namespace TeamHunter.Modules
{
    public class BearerOnCookieAuthentication : OwinMiddleware
    {
        public BearerOnCookieAuthentication(OwinMiddleware
next) : base(next) { }

        public override async Task Invoke(IOwinContext context)
        {
            var cookies = context.Request.Cookies;
            var cookie = cookies.FirstOrDefault(c => c.Key ==
IdentityPaths.TokenName);

            if
(!context.Request.Headers.ContainsKey("Authorization"))
            {
                if
(!cookie.Equals(default(KeyValuePair<string, string>)))
                {
                    var ticket = cookie.Value;

                    context.Request.Headers.Add("Authorization",      new[]      {
$"Bearer {ticket}" });
                }

                await Next.Invoke(context);
            }
        }
    }
}
```

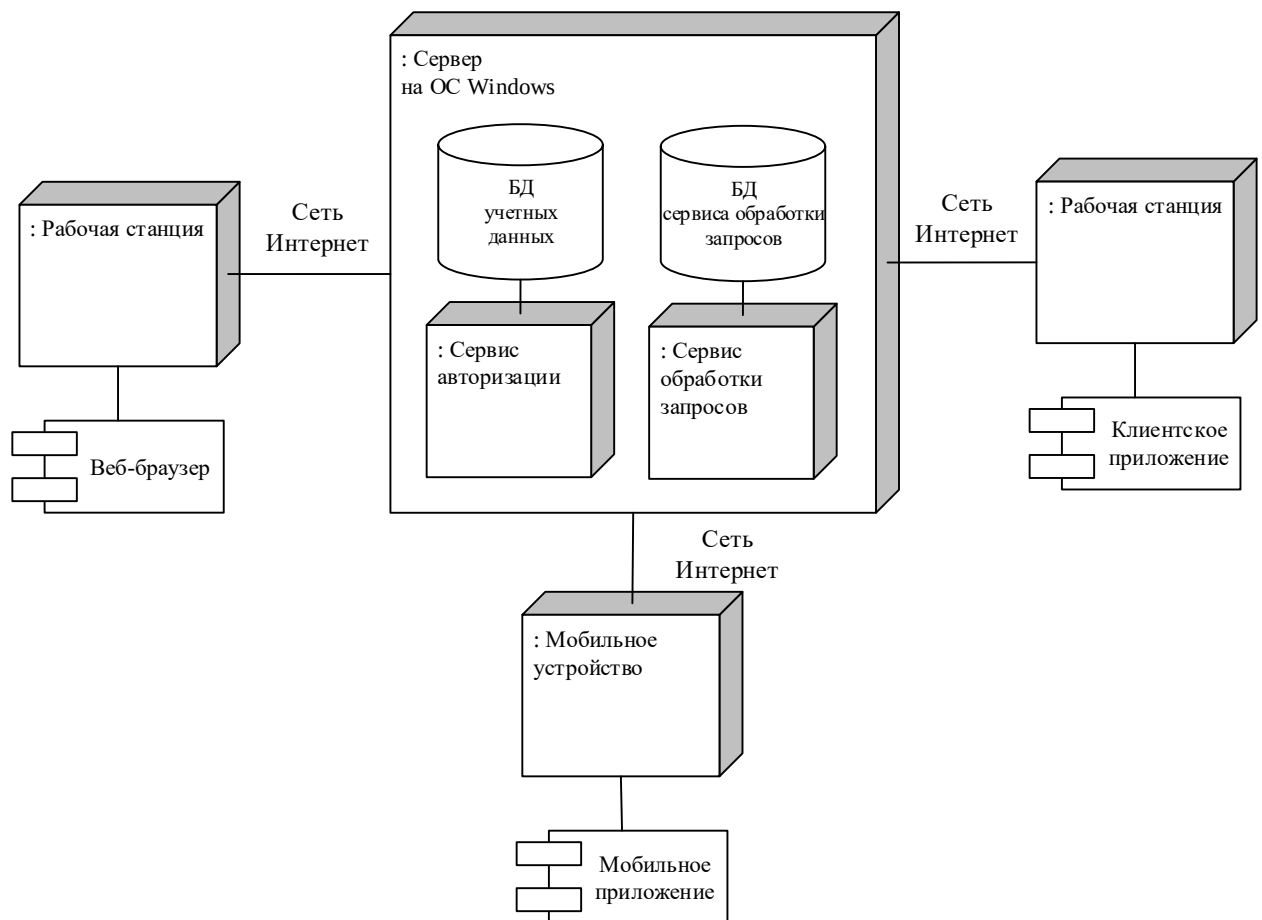
ПРИЛОЖЕНИЕ Б

ДИАГРАММЫ UML

(рекомендуемое)

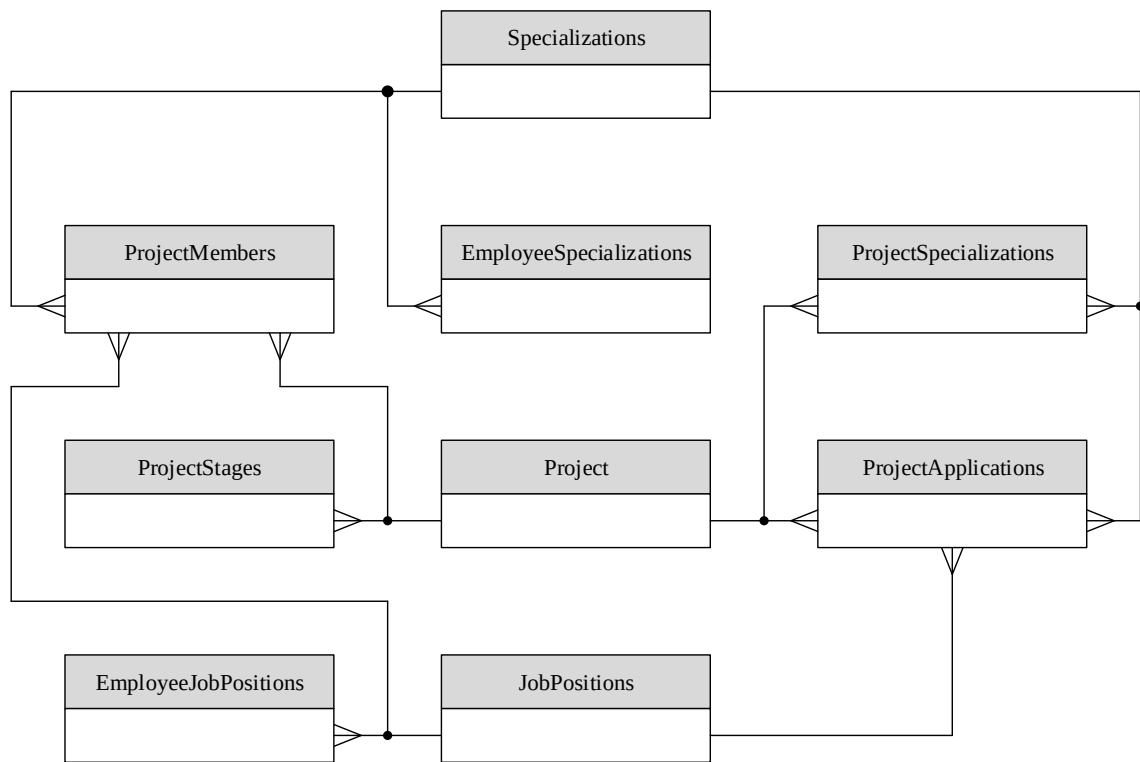
Приложение Б.1

Диаграмма развертывания



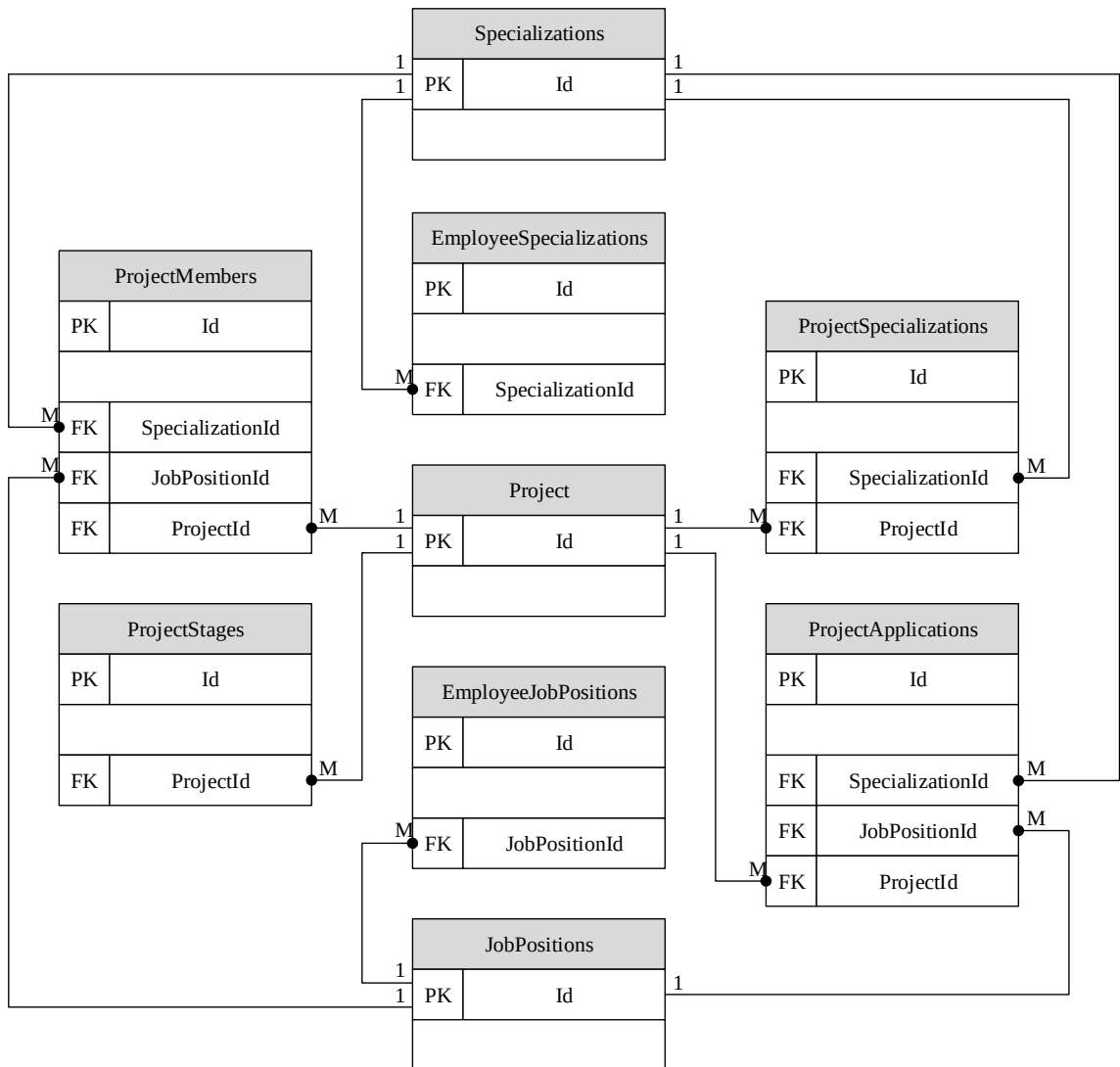
Приложение Б.2

Диаграмма сущность-связь



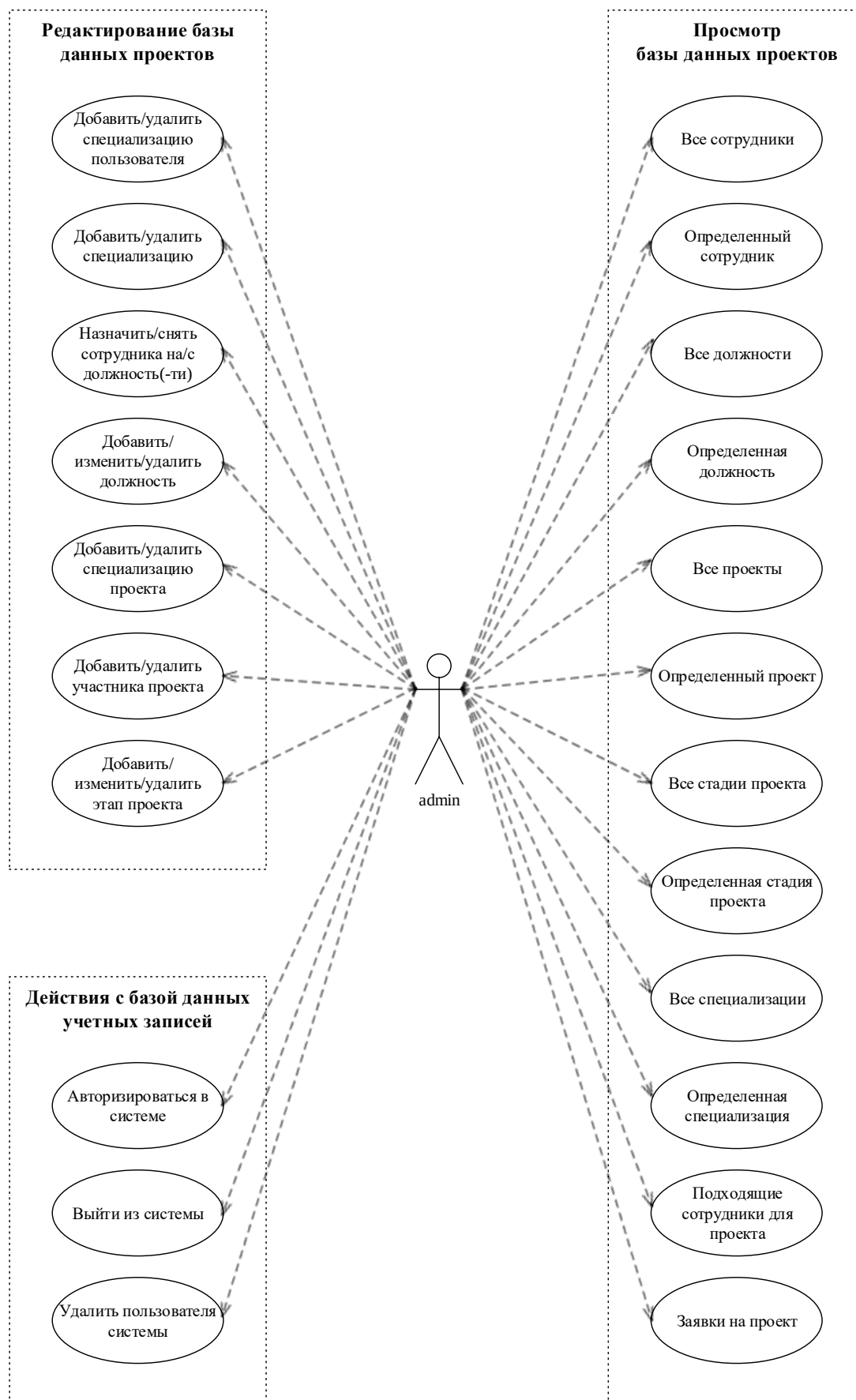
Приложение Б.3

Диаграмма классов базы данных



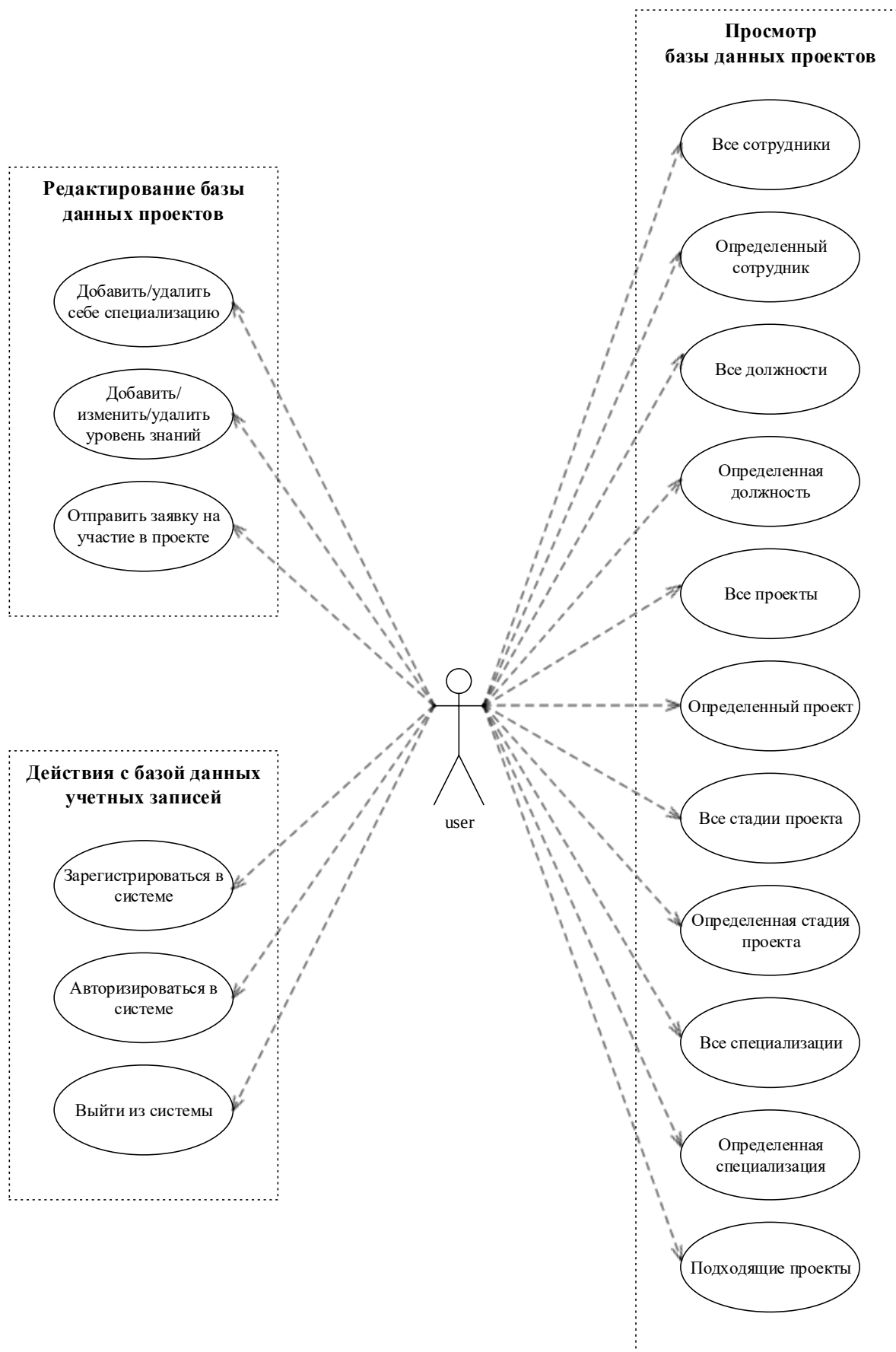
Приложение Б.4

Диаграмма прецедентов относительно администратора

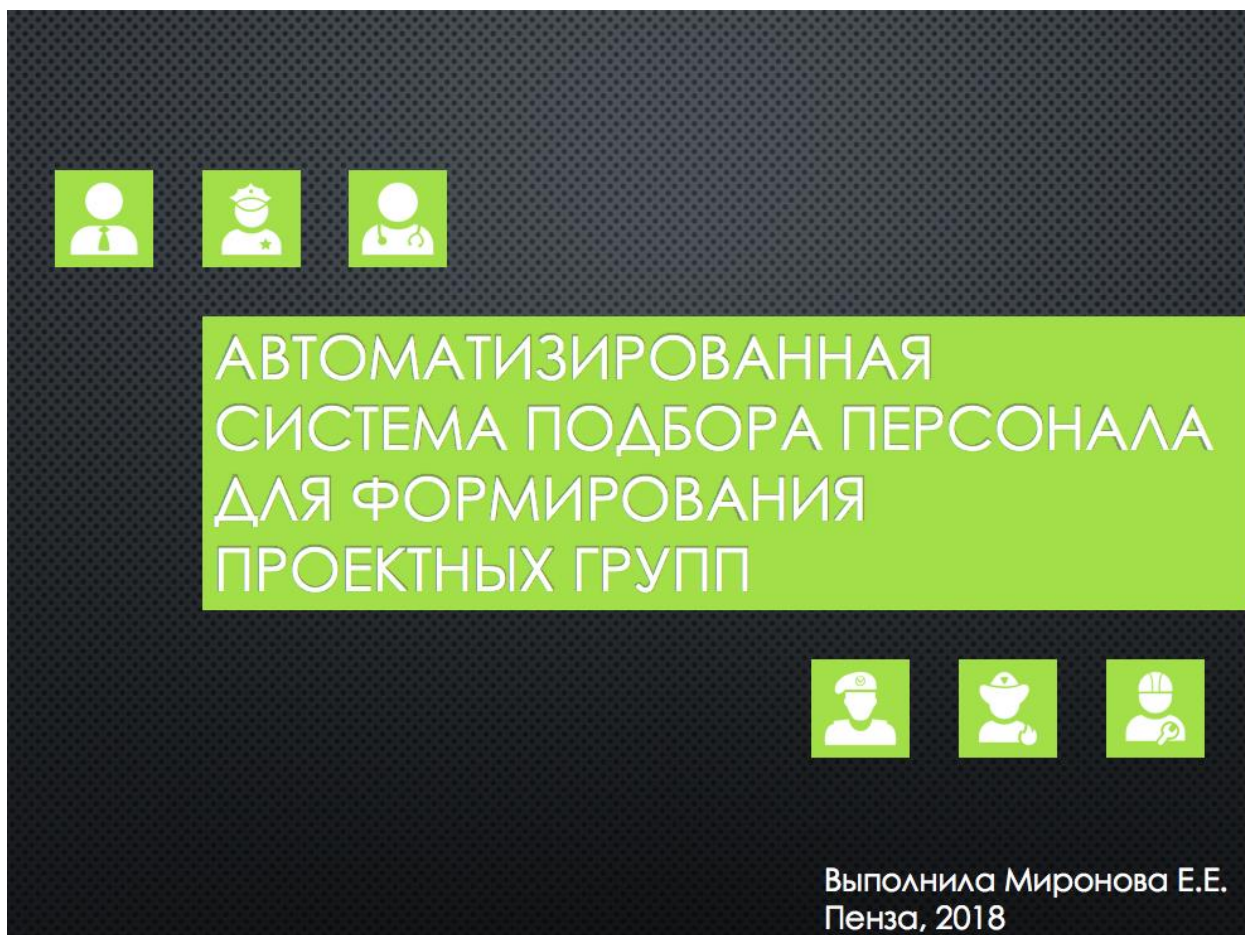


Приложение Б.5

Диаграмма прецедентов относительно пользователя



ПРИЛОЖЕНИЕ В
ПРЕЗЕНТАЦИЯ
(рекомендуемое)





ЦЕЛЬ РАБОТЫ

Разработка автоматизированной системы подбора персонала для формирования проектных групп



ЗАДАЧИ

- > Выбрать средства и методы
- > Спроектировать БД учетных данных пользователей
- > Спроектировать БД информации о проектах и сотрудниках
- > Реализовать спроектированные базы данных
- > Разработать сервис авторизации
- > Разработать сервисы для обработки запросов пользователей
- > Определить роли пользователей
- > Разграничить возможности пользователей
- > Обеспечить корректность выполняемых запросов
- > Протестировать систему



ВЫБРАННЫЕ СРЕДСТВА И МЕТОДЫ



Язык программирования C#



Платформа ASP.NET



Спецификация OWIN



Entity Framework



Microsoft SQL Server



OAuth2



JSON



Microsoft Visual Studio



Internet Information Services



Advanced REST Client



ПРОЕКТИРОВАНИЕ БАЗЫ ДАННЫХ

Сущности

Сотрудники

Проекты

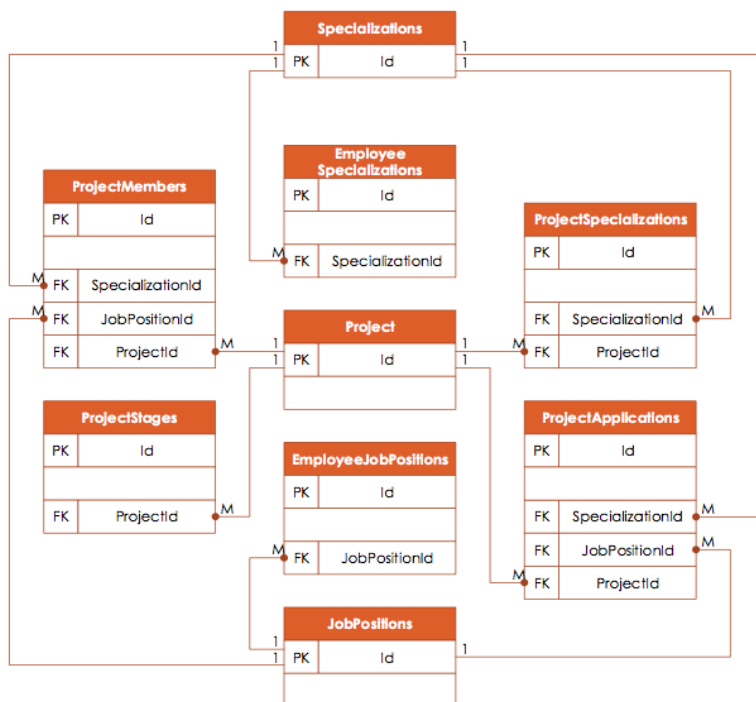
Специализации

Должности

Участники проекта

Этапы проекта

Заявки



0 1 0 1
1 0 1 1
0 1 1 0

РЕАЛИЗАЦИЯ БАЗЫ ДАННЫХ

Классы

DbEmployeeJobPosition

DbProjectApplication

DbEmployeeSpecialization

DbProjectMember

DbJobPosition

DbProjectStage

DbProject

DbSpecialization

```
public class DbSpecialization
{
    public int Id {get; set;}
    public string Name {get; set;}
    public string Description {get; set;}

    public virtual ICollection<DbProject> Projects {get; set;}
    public virtual ICollection<DbEmployeeSpecialization> EmployeesSpecializations {get; set;}
    public virtual ICollection<DbProjectApplication> ProjectApplication {get; set;}
}
```



РАЗРАБОТКА СЕРВИСОВ

Компонент авторизации

Регистрация

Вход в систему

Выход из системы

Удаление
пользователя

Компонент обработки запросов пользователей

Работа с
персоналом

Работа с
участниками
проекта

Работа со
списком
должностей

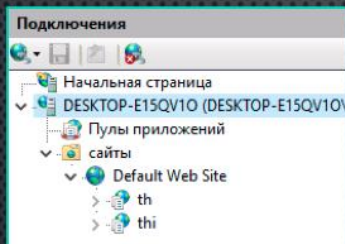
Работа с этапами
проекта

Работа с
проектами

Работа со
списком
специализаций

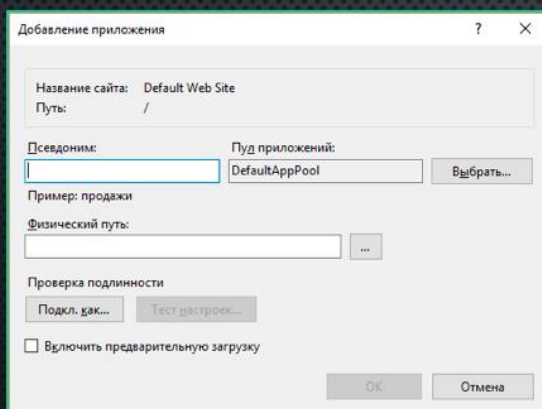
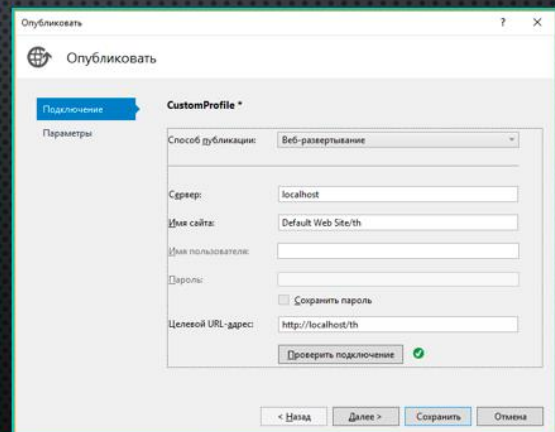


ПУБЛИКАЦИЯ



<http://localhost/thi>

<http://localhost/th>



ТЕСТИРОВАНИЕ

Регистрация нового пользователя

<http://localhost/thi/Registration>

Headers	Authorization	Body	Variables	Actions
Body content type: application..				
Editor view				
Form data (www-url-form-encoded)				
ENCODE PAYLOAD DECODE PAYLOAD				
Email		user1@mail.com		×
Username		user1		×
FirstName		Аркадий		×
LastName		Аркадиев		×
Phone		880005553535		×
Password		qwerty123		×
PasswordConfirm		qwerty123		×

HTTP Response code 200

```
{
  "Value": "91ef6a4e-a12f-483d-bcfb-75abadd2b3af",
  "Success": true,
  "Error": ""
}
```



ТЕСТИРОВАНИЕ

Вход в систему

<http://localhost/thi/OAuth/Token>

grant_type	password	×
Username	user1	×
Password	qwerty123	×

```
200 OK 801.00 ms DETAILS ▾  
{  
  "access_token": "34yGIyFiIe5lPyoy_5FrnnorSRzrwHkF4IDabVO-  
yrrFnVUMNBivjmg2FjdpKFpB4Q_h8EF9iVnDgpKUXR1AWwftQ8IvtiYo9LqYuwqzGuuMXN9fyecKATUKzf  
zqO4dCvCuIFC8aUgFrg2Ddw7sRMs0Kbeu321Dly0MfjM_YIRyF7ydw5491adWrl1MtNu1UASuj1xKOSM2mB  
V7s6e6On1VS-  
IS2hQJocLYr7RU6YDeXbrOrg3foJGU12B_T9J3LjG7ogOp1T2oDeA0Kt22kTK8eYVOYzCAQIocNDuY0q_v7  
X_gaAUPSR68GGiailXrCD9b9_CpoEn_Gi0SGSOUXIWm0DcFBxXKqrKLn1Q890Xa59N-  
Cn_zDCX01hnWiU8Ej_uXX7axqr06pQoiCM7kDhJFdh2-mFD-  
luXan0ShhAnX8t_TS4nuvYHhVq9Fy1y91USyCA9EUshKlbF2Wdm0ft1jh0orKvdj6G0-  
spDT06NwSCoLGIz-FvWiv5V1St8I2HJw",  
  "token_type": "bearer",  
  "expires_in": 1209599  
}
```



ТЕСТИРОВАНИЕ

Общедоступный запрос

```
Request URL  
http://localhost/th/Specializations  
SEND  
Parameters ▾  
200 OK 302.00 ms DETAILS ▾  
{  
  "-Value": {  
    "Total": 4,  
    "-Data": [Array(4)]  
  },  
  "-0": {  
    "Id": 1,  
    "Name": "Дизайнер",  
    "Description": "Разработка макета сайтов"  
  },  
  "-1": {  
    "Id": 2,  
    "Name": "Программист",  
    "Description": "Разработка приложений"  
  },  
  "-2": {  
    "Id": 3,  
    "Name": "Мобильный разработчик iOS",  
    "Description": "Разработка мобильных приложений на базе iOS"  
  },  
  "-3": {  
    "Id": 4,  
    "Name": "Мобильный разработчик Android",  
    "Description": "Разработка мобильных приложений на базе ОС Android"  
  },  
  "Success": true,  
  "Error": ""  
}
```

Запрос с ограничением прав

```
Method Request URL  
PUT http://localhost/th/JobPositions  
SEND  
Parameters ▾  
Headers Authorization Body Variables Actions  
Body content type Editor view  
application/x-www-form-... Form data (www-url-form-encoded)  
ENCODE PAYLOAD DECODE PAYLOAD  
Title Начальник  
Description Командует  
ADD FORM PARAMETER  
401 Unauthorized 68.00 ms DETAILS ▾  
{  
  "Message": "Authorization has been denied for this request."  
}
```

```
{  
  "Message": "Authorization  
has been denied for the  
request."  
}
```



ТЕСТИРОВАНИЕ

Просмотр подходящих проектов

```
{
  "-Value": {
    "Total": 2,
    "-Data": [Array(2)]
    -0: {
      "Stages": [Array(0)],
      "Id": 1,
      "Created": "2018-05-14T00:00:00",
      "Title": "Система подбора персонала",
      "Description": "Подбирает персонал",
      "IsOpen": true,
      "StartDate": "2018-05-14T00:00:00",
      "EndDate": "2018-05-14T00:00:00"
    },
    -1: {
      "Stages": [Array(0)],
      "Id": 2,
      "Created": "2018-05-14T00:00:00",
      "Title": "Сервер авторизации",
      "Description": "Управление учетными записями",
      "IsOpen": true,
      "StartDate": "2018-07-21T00:00:00",
      "EndDate": "2020-01-12T00:00:00"
    }
  },
  "Success": true,
  "Error": ""
}
```

Добавление специализации

ProjectId	2	×
SpecializationId	4	×

ADD FORM PARAMETER

200 OK 6115.00 ms [DETAILS](#)

```
{
  "Value": true,
  "Success": true,
  "Error": ""
}
```



ТЕСТИРОВАНИЕ

Просмотр подходящих проектов

```
{
  "-Value": {
    "Id": "51ef644e-a12f-483d-bc9b-75abadd2b3af",
    "Username": "user1",
    "FirstName": "Аркадий",
    "LastName": "Аркадьев",
    "Email": "user1@mail.com",
    "Phone": null,
    "JobPositions": [Array(0)],
    "-Specializations": [Array(2)]
    -0: {
      "Id": 2,
      "Name": "Программист",
      "Description": "Разработка приложений"
    },
    -1: {
      "Id": 4,
      "Name": "Мобильный разработчик Android",
      "Description": "Разработка мобильных приложений на базе ОС Android"
    }
  },
  "ProjectApplications": [Array(0)],
  "CurrentProject": null,
  "ProjectHistory": [Array(0)],
  "Success": true,
  "Error": ""
}
```

Method GET Request URL http://localhost/th/Employee/PotentialProjects

Parameters

200 OK 1685.00 ms

```
{
  "-Value": {
    "Total": 1,
    "-Data": [Array(1)]
    -0: {
      "Stages": [Array(0)],
      "Id": 2,
      "Created": "2018-05-14T00:00:00",
      "Title": "Сервер авторизации",
      "Description": "Управление учетными записями",
      "IsOpen": true,
      "StartDate": "2018-07-21T00:00:00",
      "EndDate": "2020-01-12T00:00:00"
    }
  },
  "Success": true,
  "Error": ""
}
```



ТЕСТИРОВАНИЕ

Просмотр потенциальных сотрудников

```
{
  "-Value": {
    "Total": 2,
    "-Data": [Array(2)]
    -0: {
      "Stages": [Array(0)],
      "Id": 1,
      "Created": "2018-05-14T00:00:00",
      "Title": "Система подбора персонала",
      "Description": "Подбирает персонал",
      "IsOpen": true,
      "StartDate": "2018-05-14T00:00:00",
      "EndDate": "2018-05-14T00:00:00"
    },
    -1: {
      "Stages": [Array(0)],
      "Id": 2,
      "Created": "2018-05-14T00:00:00",
      "Title": "Сервер авторизации",
      "Description": "Управление учетными записями",
      "IsOpen": true,
      "StartDate": "2018-07-21T00:00:00",
      "EndDate": "2020-01-12T00:00:00"
    }
  ],
  "Success": true,
  "Error": ""
}
```

<http://localhost/th/Projects/2/PotentialMembers>

```
{
  "-Value": {
    "Total": 1,
    "-Data": [Array(1)]
    -0: {
      "Id": "91ef6a4e-a12f-483d-bcfb-75abadd2b3af",
      "Username": "user1",
      "FirstName": "Аркадий",
      "LastName": "Аркадиев",
      "Email": "user1@mail.com",
      "Phone": null,
      "JobPositions": [Array(0)],
      "-Specializations": [Array(2)]
      -0: {
        "Id": 2,
        "Name": "Программист",
        "Description": "Разработка приложений"
      },
      -1: {
        "Id": 4,
        "Name": "Мобильный разработчик Android",
        "Description": "Разработка мобильных приложений на базе ОС Android"
      }
    }
  ],
  "Success": true,
  "Error": ""
}
```



ТЕСТИРОВАНИЕ

Добавление сотрудника в список участников проекта

Создание заявки <http://localhost/th/Projects/1CreateApplication/6/4>

Просмотр заявок

Method

GET

Request URL

http://localhost/th/Project/1/Applications

SEND

Parameters

200 OK

599.00 ms

DETAILS

<>

Value (Object)

Total

1

Data (Array 1)

Id	Username	FirstName	LastName	Email	Phone	JobPositions		Specializations	
						Id	Title	Id	Name
91ef6a4e-a12f-483d-bcfb-75abadd2b3af	user1	Аркадий	Аркадиев	user1@mail.com		hide array		hide array	
						6	Разработчик	4	Мобильный разработчик Android



ТЕСТИРОВАНИЕ

Добавление сотрудника в список участников проекта

Добавление сотрудника по заявке

Method: PUT Request URL: http://localhost:th/Projects/AddMember/2 SEND

Parameters: 200 OK 1020.00 ms DETAILS

```
{
  "Value": {
    "Id": 1,
    "Success": true,
    "Error": ""
  }
}
```



ТЕСТИРОВАНИЕ

Добавление сотрудника в список участников проекта

Просмотр информации о сотруднике, после добавления в проект

```
{
  "-Value": {
    "Id": "91ef6a4e-a12f-483d-bcfb-75abadd2b3af",
    "Username": "user1",
    "FirstName": "Аркадий",
    "LastName": "Аркадиев",
    "Email": "user1@mail.com",
    "Phone": null,
    "JobPositions": [Array(0)],
    "-Specializations": [Array(2)]
  },
  "-0": {
    "Id": 2,
    "Name": "Программист",
    "Description": "Разработка приложений"
  },
  "-1": {
    "Id": 4,
    "Name": "Мобильный разработчик Android",
    "Description": "Разработка мобильных приложений на базе ОС Android"
  }
},
  "-ProjectApplications": [Array(1)]
  "-0": {
    "Id": 2,
    "GuId": "91ef6a4e-a12f-483d-bcfb-75abadd2b3af",
    "ProjectId": 1,
    "JobPositionId": 6,
    "SpecializationId": 4
  }
},
  "-CurrentProject": {
    "Stages": [Array(0)],
    "Id": 1,
    "Created": "2018-05-14T00:00:00",
    "Title": "Система подбора персонала",
    "Description": "Подбирает персонал",
    "IsOpen": true,
    "StartDate": "2018-05-14T00:00:00",
    "EndDate": "2018-05-14T00:00:00"
  }
}
```



КОНЕЦ

СПАСИБО
ЗА ВНИМАНИЕ