РОСЖЕЛДОР

Федеральное государственное бюджетное образовательное   
учреждение высшего образования

«Ростовский государственный университет путей сообщения»

(ФГБОУ ВО РГУПС)

Допустить к защите в ГЭК

Зав. кафедрой «ВТ и АСУ»

\_\_\_\_\_\_\_\_\_\_\_\_\_ О.В. Игнатьева

«\_\_\_\_\_\_»\_\_\_\_\_\_\_\_\_\_\_\_2022 г.

**Создание микросервисного приложения для организации работы  
контакт-центра**

**ПОЯСНИТЕЛЬНАЯ ЗАПИСКА**

к выпускной квалификационной (бакалаврской) работе

АВБ 12.02.13

Направление подготовки 09.03.01 «Информатика и вычислительная техника», профиль «Информационные системы и технологии на транспорте»

Обучающийся \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ И.А. Мазуров

Руководитель работы

д.п.н., профессор \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ О.И. Соколова

Нормоконтроль

ст. преподаватель \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Н.Р. Осипова

Научный консультант

к.т.н., научный сотрудник \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ С. А. Кулькин

2022

РОСЖЕЛДОР

Федеральное государственное бюджетное образовательное учреждение

высшего образования

«Ростовский государственный университет путей сообщения»

(ФГБОУ ВО РГУПС)

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Кафедра «ВТ и АСУ»

УТВЕРЖДАЮ

Зав. кафедрой

\_\_\_\_\_\_\_\_\_\_\_\_\_\_ О.В. Игнатьева

«\_\_\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_\_\_ 2022 г.

ЗАДАНИЕ

на выпускную квалификационную(бакалаврскую) работу

Студенту Мазурову Илье Андреевичу \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(фамилия, имя, отчество)

Группа АВБ-4-032

**1. Тема работы:** Разработка микросервисного приложения для организации работы контакт-центра

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Утверждена приказом по университету № 108/ос от 26.01.2022 г.

Срок сдачи студентом законченной работы «08» июня 2022 г.

**2. Исходные данные к работе** \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

1. Литература по программированию на языке C#\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

2. Литература по разработке приложений на платформе ASP.NET Core MVC

3. Литература по проектированию информационных систем\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**3. Содержание расчетно-пояснительной записки (перечень вопросов, подлежащих разработке)**

1. Анализ предметной области разрабатываемого приложения

2. Актуальность разрабатываемого приложения\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

3. Анализ существующих приложений\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

4. Техническое задание\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 5. Проектирование приложения\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

6. Построение диаграммы прецедентов\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

7. Построение диаграммы классов\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

8. Построение диаграммы последовательности\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

9. Расчет FP-метрик, LOC-метрик и COCOMO-метрик\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

10. Программная реализация приложения\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

11. Выбор технических средств для разрабатываемого приложения\_\_\_\_\_\_\_\_\_\_\_

12. Проектирование базы данных приложения\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

13. Разработка приложения\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

14. Разработка интерфейса приложения\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

15. Создание руководства пользователя приложением\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**4. Перечень графического материала (с точным указанием названий слайдов презентации)**

1. Титульный лист\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

2. Цели работы \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

3. Функционал приложения\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

4. Диаграмма прецедентов \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

5. Диаграмма классов \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

6. Диаграмма последовательностей\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

7. Демонстрация интерфейса. Панель авторизации\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

8. Демонстрация интерфейса. Главная страница \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

9. Демонстрация интерфейса. Страница отправки сообщения клиенту\_\_\_\_\_\_\_\_

10. Демонстрация интерфейса. Страницы звонка и работы со статистикой\_\_\_\_\_

11. Демонстрация интерфейса. Панель администратора\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

12. Заключение\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Дата выдачи задания « 07 » февраля 2022 г.

Руководитель работы \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Соколова О.И.\_\_\_\_

(подпись) (инициалы, фамилия)

Задание принял к исполнению « 07 » февраля 2022 г.

Студент\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_Мазуров И.А.\_\_\_

(подпись) (инициалы, фамилия)

**Реферат**

Выпускная квалификационная работа содержит 82 листа пояснительной записки, включающей 26 рисунков, 11 таблиц, 14 источников и 1 приложение.

ИНФОРМАЦИОННЫЕ ТЕХНОЛОГИИ, ИНФОРМАТИКА, UML-ДИАГРАММА, МИКРОСЕРВИСНОЕ ПРИЛОЖЕНИЕ, ASP.NET CORE MVC, C#, HTML, CSS, WEBSTORM, ИНФОРМАЦИОННАЯ СИСТЕМА, ПРЕЦЕДЕНТ, КЛАСС, MICROSOFT SQL SERVER, БАЗА ДАННЫХ, СУБД, POSTRGESQL, RIDER, VISUAL STUDIO.

Объектом исследования является микросервисное приложение для организации работы контакт-центра.

Цель работы – разработка микросервисного приложения для организации работы контакт-центра.

В выпускной квалификационной работе кратко описаны правила пользования микросервисным приложением для организации работы контакт-центра.

Приведена структура программы в виде списка модулей и характеристики каждого. Расчеты метрик позволили определить оценки по стоимости и затратам на разработку микросервисного приложения для организации работы контакт-центра. На основе диаграмм прецедентов, последовательности, взаимодействий и классов установлены отношения между [актёрами](https://ru.wikipedia.org/wiki/%D0%90%D0%BA%D1%82%D1%91%D1%80_(UML)) и [прецедентами](https://ru.wikipedia.org/wiki/%D0%9F%D1%80%D0%B5%D1%86%D0%B5%D0%B4%D0%B5%D0%BD%D1%82_(UML)), спроектированы и созданы классы, интерфейсы и отношения между ними.

Рассмотрены современные среды разработки, их возможности и преимущества. Выбраны оптимальные для разрабатываемого программного средства инструменты.

В результате разработки было разработано микросервисное приложение для организации работы контакт-центра.

Содержание

[**Введение** 6](#_Toc106092109)

[1 Анализ предметной области разрабатываемого приложения 9](#_Toc106092110)

[1.1 Актуальность разработки микросервисного приложения для организации работы контакт-центра 9](#_Toc106092111)

[1.2 Аналитический обзор существующих приложений для организации работы контакт-центра 12](#_Toc106092112)

[1.3 Техническое задание на создание микросервисного приложения для организации работы контакт-центра 16](#_Toc106092113)

[1.4 Выбор языка и среды разработки микросервисного приложения для организации работы контакт-центра 18](#_Toc106092114)

[1.5 Вывод по разделу 21](#_Toc106092115)

[2 Проектирование микросервисного приложения для организации работы контакт-центра 22](#_Toc106092116)

[2.1 Анализ требований на основе диаграммы прецедентов и сценария использования микросервисного приложения для организации работы контакт-центра 22](#_Toc106092117)

[2.2 Построение диаграммы классов микросервисного приложения для организации работы контакт-центра 25](#_Toc106092118)

[2.3 Построение разработки диаграммы операционную последовательности привлечь микросервисного приложения для организации работы контакт-центра 31](#_Toc106092119)

[2.4 Расчет функциональных и размерно-ориентированных метрик для микросервисного приложения для организации работы контакт-центра 32](#_Toc106092120)

[2.5 Расчет модели оценки стоимости разработки микросервисного приложения для организации работы контакт-центра 38](#_Toc106092121)

[2.6 Проектирование базы данных микросервисного приложения для организации работы контакт-центра 40](#_Toc106092122)

[2.7 Вывод по второму разделу 41](#_Toc106092123)

[3 Программная реализация приложения для организации работы контакт-центра 43](#_Toc106092124)

[3.1 Интерфейс микросервисного приложения для организации работы контакт-центра 43](#_Toc106092125)

[3.2 Руководство пользователя к микросервисному приложению для организации работы контакт-центра 54](#_Toc106092126)

[3.3 Выводы по третьему разделу 59](#_Toc106092127)

[Заключение 60](#_Toc106092128)

[Список используемых источников 61](#_Toc106092129)

[Приложение А 63](#_Toc106092130)

**Введение**

Использование информационных технологий больше не сводится только к установке оборудования или программного обеспечения, решению компьютерных проблем или контролю за тем, кто может получить доступ к конкретной системе. Сегодня IT-сфера востребована также в следующем:

1. поддержке сети и устройства для максимального времени безотказной работы;
2. автоматизации процессов для повышения эффективности бизнеса;
3. исследовании, внедрении и управлении новыми технологиями для удовлетворения меняющихся потребностей бизнеса;
4. поддержке уровня обслуживания, безопасности и возможности подключения для обеспечения непрерывности и долговечности бизнеса.

Из всего вышесказанного можно сделать вывод, что IT-сфера все больше становится популярна в области бизнеса.

Благодаря информационным технологиям многие традиционные бизнес-модели и концепции претерпели изменение. Информационные технологии дали новые возможности и перспективы, а также позволили повысить эффективность ведения бизнеса. Множество повседневных рутинных и ручных задач были автоматизированы и оптимизированы.

Одним из главных нововведений со стороны информационных технологий стало обеспечение непрерывной связи с клиентами. Каждый человек, имеющий доступ в интернет способен получить обратную связь или ответ на интересующий его вопрос различными способами, например, по электронной почте или через чат на веб-сайте. Но остаются люди, которые не готовы долго ждать ответов на свои вопросы, или которые предпочитают общаться с агентами по обслуживанию клиентов в реальном времени, для всех этих вариантов связи было создано единое решение – контакт-центр.

На сегодняшний день любое дело, основывающееся на продаже товаров и услуг обязано иметь постоянно работающую техническую поддержку и связь с клиентом – это стандарт современного бизнеса. Ранее с этим успешно справлялись колл-центры, основными задачами которых были: обработка звонков и информирование клиентов в интересах бизнеса. В современном мире потребности клиентов меняются, и компаниям приходится искать все больше способов для контакта с конечными пользователями. Выходом из такой ситуации является – создание или организация контакт-центра, который представляет собой следующий эволюционный этап развития колл-центров.

Рассматриваемая тема выпускной квалификационной работы является актуальной, так как посвящена разработке микросервисного приложения для организации работы контакт-центра. Использование контакт-центра сейчас актуально для самых разных отраслей, и большинство компаний, которые начинают заниматься предоставлением товаров и услуг, являются крупными финансовыми организациями и IT-компаниями, обязаны иметь средства высокоэффективной службы поддержки.

Целью выпускной квалификационной работы является создание микросервисного приложения для организации работы контакт-центра, которое позволит организации управлять всеми взаимодействиями с клиентами по различным каналам связи.

Для достижения поставленной цели необходимо выполнить следующие взаимосвязанные задачи:

1. Провести анализ предметной области. Изучить и проанализировать существующие разработки в предметной области. Разработать техническое задание согласно требованиям.
2. Спроектировать приложение на основе диаграмм прецедентов, классов и последовательности. Рассчитать время разработки приложения на основе функционально-ориентированных метрик и модели издержек разработки.
3. Реализовать приложение на основе поставленного технического задания и проекта. Применить спроектированные UML диаграммы и полученные при изучении предметной области теоретические знания при разработке приложения.

Комплексный характер темы выпускной квалификационной работы определил необходимость изучения и практического применения трудов национальных и зарубежных разработчиков программного обеспечения: Джулии Коннелл - профессора Высшей школы бизнеса Ньюкасла [1], Джона Берджесс - профессора Высшей школы бизнеса Ньюкасла [2], Джеффри Рихтера – IT-специалиста и разработчика, автора нескольких книг по технологиям Microsoft [3], Эндрю В. Троелсена - менеджера по технологиям в [Thomson Reuters](https://en.wikipedia.org/wiki/Thomson_Reuters) в подразделении Enterprise Content Platform [4] и других, которые изучали и проектировали информационные системы контакт-центров. Их труды были использованы при разработке и создании микросервисного приложения для организации работы контакт-центра.

В первой главе был осуществлен анализ предметной области для разрабатываемого приложения для организации работы контакт-центра, рассмотрены программы-аналоги, разработано техническое задание для дальнейшего проектирования и разработки.

Во второй главе при проектировании микросервисного приложения для организации работы контакт-центра был выполнен анализ требований к разрабатываемой системе на основе построения диаграмм UML. На основе построения диаграмм прецедентов спроектированы основные функциональные возможности микросервисного приложения для организации работы контакт-центра, построение диаграммы классов позволило сформировать основу программной реализации, расчет функционально-ориентированных метрик и модели издержек дает представление о примерных сроках разработки приложения.

В третьей главе описан процесс разработки программного кода микросервисного приложения для организации работы контакт-центра, дано описание взаимодействия частей приложения, а также содержится описание интерфейса готового приложения и его функционал.

# 1 Анализ предметной области разрабатываемого приложения

## 1.1 Актуальность разработки микросервисного приложения для организации работы контакт-центра

В настоящее время информационные технологии стали активно применяться в различных областях нашей жизни, исключением не стала и такая сложная тема, как взаимодействие клиентов и бизнеса. Широкое внедрение цифровых устройств и постоянное подключение к Интернету меняют порядок того, как клиенты желают общаться с организациями. Разговоры больше не ограничиваются традиционным голосовым каналом, вместо этого клиенты ожидают беспрепятственного взаимодействия с компаниями по нескольким каналам, такими как чат, мобильные устройства, видео и социальные сети.

Правильным решением в такой ситуации будет интеграция в организацию работы компании новых технологий, а именно контакт-центра. Контакт-центр – это бизнес-подразделение внутри организации, которое управляет взаимодействием с клиентами [8]. В отличие от колл-центра, который получает запросы только по телефону, контакт-центр обрабатывает входящие и исходящие сообщения клиентов по нескольким каналам, таким как телефон, Интернет, чат, электронная почта, приложения для обмена сообщениями, социальные сети, текстовые сообщения, факс и традиционная почта [9]. Контакт-центры используют различные типы передовых технологий, чтобы помочь быстро решать проблемы клиентов, отслеживать и собирать данные о взаимодействие с клиентами с целью улучшения производительности работы бизнеса [10].

Контакт-центр обладает рядом достоинств, которые помогут любой организации стать лучше [11]:

1. Улучшенный клиентский опыт. Контакт-центр часто является основным каналом для большинства взаимодействий клиентов с компанией. Через контакт-центр можно обеспечить отличный опыт взаимодействия несколькими способами. Например, клиенты могут воспользоваться возможностями самообслуживания, предоставляемыми контакт-центром, для быстрого выполнения общих задач. Контакт-центры могут обеспечить подключение клиентов к подходящему агенту для более быстрого решения их проблемы. Кроме того, поскольку контакт-центр поддерживает несколько видов связи, клиенты могут взаимодействовать по выбранному ими каналу.
2. Повышение эффективности. Поскольку контакт-центры являются центральной точкой взаимодействия с клиентами, они могут повысить производительность бизнеса, отвечая на многочисленные запросы клиентов. Контакт-центры могут предоставить организации возможность автоматически направлять запросы на наиболее подходящий ресурс, что позволяет операторам реагировать быстро и эффективно.
3. Расширенное понимание и видимость. Контакт-центр объединяет взаимодействие с клиентами по всем каналам связи. Данные, собранные в результате этих взаимодействий, дают ценную информацию, которая может помочь в принятии важных бизнес-решений. Эта информация может помочь с усовершенствованием дизайна продукта, вопросами качества продукта, а также с выявлением шаблонов и проблемных областей на пути клиента к улучшенному взаимодействию с компанией.

Разрабатываемое микросервисное приложение для организации работы контакт-центра сделает доступной возможность любым компаниям организовать эффективную работу. Такое приложение подойдет любой отрасли, где важна коммуникация с клиентом: онлайн-магазинам, банкам, сфере грузоперевозок, туристическим агентствам и другим компаниям. С помощью организации контакт-центров можно достичь высокой эффективности в работе службы поддержки, которая необходима Интернет-провайдерам и операторам сотовой связи, логистическим и транспортным компаниям.

Использование для реализации микросервисного приложения вполне логично, ведь микросервисы, обычно называемые «микросервисной архитектурой», представляют собой способ структурирования приложения таким образом, что бизнес-возможности, которые должны быть предоставлены в приложении, могут быть отделены, созданы и развернуты как независимые службы [12]. Вместо того, чтобы разрабатывать приложение как единое целое (так называемая [монолитная архитектура приложения](https://www.thesunflowerlab.com/blog/choose-microservices-monolithic-application-architecture/)), вся бизнес-функциональность приложения разбивается на уникальные процессы. Каждый процесс проектируется и разрабатывается как самостоятельный сервис, содержащий свою бизнес-логику и имеющий свой набор обособленных данных.

Микросервисная архитектура помогает снизить сложность разработки. Большие или сложные приложения можно разбить на более простые, легко разрабатываемые и поддерживаемые сервисы, которые можно обновлять по мере изменения динамики бизнеса и необходимости переделывать рабочие процессы, чтобы они отражали реальную работу [13].

Команде разработчиков также легче практиковать непрерывную доставку и развертывание функций кода для поддержки динамических процессов, включающих частые или динамически изменяющиеся бизнес-среды. Кроме того, это помогает бизнесу постоянно развивать свой технологический стек и внедрять новые процессы и методы, чтобы оставаться конкурентоспособными на рынке.

Одной из главных причин использования микросервисного приложения является необходимость возможности масштабирования в зависимости от бизнес-требований. Крупные организации имеют различные бизнес-процессы и требования. Их необходимо автоматизировать и переделывать по мере изменения масштаба. Монолитные архитектуры требуют времени для создания, изменения и развертывания. С помощью микросервисов [конкретную службу можно перепроектировать и развернуть](https://www.thesunflowerlab.com/cloud-deployment/) за короткое время, не влияя на объем других процессов или сервисов. Новые сервисы можно создавать в короткие сроки и развертывать независимо [14].

Архитектура микросервисов не определяет и не ссылается на конкретный набор технологий, процессов или инструментов. Скорее, она сосредотачивается на целях. Для любого бизнеса чем раньше будут автоматизированы рабочие процессы, тем быстрее будет доставка. Кроме того, автоматизированные процессы должны быть надежными и согласованными, с точки зрения обеспечивания. Реальная ценность микросервисов для бизнеса может быть реализована путем сосредоточения внимания на двух ключевых аспектах – скорости и надежности – путем их эффективного балансирования в соответствии с потребностями.

Таким образом, в данном подразделе была рассмотрена и обоснована актуальность разработки микросервисного приложения для организации работы контакт-центра.

## 1.2 Аналитический обзор существующих приложений для организации работы контакт-центра

Для более полного понимания о том, что представляет из себя контакт-центр и для составления более четкой структуры о функциональных возможностях разрабатываемого программного средства, необходимо изучить рынок аналогичных приложений или приложений, предоставляющих близкий к разрабатываемому приложению функционал.

Контакт-центры делятся на несколько типов в зависимости от их реализации:

1. Аппаратные контакт-центры. Организации могут устанавливать и размещать аппаратные контакт-центры на физических локальных серверах. Следовательно, аппаратные варианты требуют от организаций достаточного пространства для размещения и мощности в обслуживание серверов, эффективных процедур аварийного восстановления и компетентных процессов обновления оборудования.
2. Облачные контакт-центры. В данном случае контакт-центры размещаются на Интернет-серверах облачных провайдеров и фильтруют все входящие и исходящие сообщения. Агенты могут получить доступ к облачным контакт-центрам из любой точки сети Интернет. Эти центры функционируют так же, как и другие.
3. Hosted контакт-центры. Для этого варианта организация передает инфраструктуру другой компании, которая управляет системами извне. Такой подход может минимизировать первоначальные затраты на техническое обслуживание, что часто приводит к повышению эффективности инвестиций в проекты.
4. Виртуальные контакт-центры. Они позволяют агентам работать удаленно. Виртуальные контакт-центры, обеспечивая гибкость и комфорт для операторов, одновременно снижают расходы компаний.

Но несмотря на разную реализацию, все они выполняют схожие функции, имея одну суть разработки. Чтобы более детально разобраться в функционале каждого вида контакта-центра, необходимо рассмотреть несколько существующих аналогов.

На первом месте в России по популярности находится компания Mango Office – это одна из крупнейших телекоммуникационных компаний страны, является абсолютным лидером российского рынка виртуальных АТС и одним из ведущих поставщиков SaaS-решений (англ. software as a service модель предоставления лицензии на программное обеспечение по подписке).

Рассматриваемая компания предоставляет множество сервисов, которые включают в себя: виртуальную АТС, коллтрекинг, интеграции с различными CRM-системами, сквозную аналитику, речевую аналитику, бизнес-аналитику и контакт-центр.

В данном случае клиенту предоставляют услуги облачного контакт-центра, в число которых входит [5]:

1. Омниканальное общение - работа со всеми заявками из единого окна: звонки, письма, сообщения из чата на сайте, социальные сети, мессенджеры, формы обратной связи и заказы на обратный звонок. Вся история коммуникаций из любых каналов связи с клиентом сохраняется в карточке-сделки Контакт-центра.
2. Умная маршрутизация звонков - качественный прием обращений: голосовое меню, автоинформатор о времени ожидания. Выстраивание индивидуального маршрута звонков специально под бизнес-процессы: гибкие алгоритмы распределения звонков.
3. Голосовые роботы - прием и обработка звонков роботизированными сервисами по индивидуальным заданным сценариям. Массовые исходящие обзвоны клиентских баз.
4. Чат-боты – прием и обработка роботом сообщений от клиентов из соцсетей, мессенджеров, чата на сайте и диалоги с ними по заданным сценариям. Адресация текстового обращения на сотрудника при необходимости. Автоматическое создание карточки нового клиента или занесение заявки от текущего клиента в CRM.
5. Анализ работы сотрудников - контроль работы операторов по любым заданным параметрам в динамике: количество пропущенных, количество перезвонов, время на линии, выполнение задач сотрудниками, время на обработку текстового обращения, успешные/неуспешные звонки и многое другое.
6. Интеграции - Контакт-центр MANGO OFFICE уже интегрирован с основными CRM-системами и множеством других бизнес-приложений. Благодаря готовым интеграциям внедрение проходит максимально быстро и бесшовно.

По данному решению можно сделать вывод, что оно применимо в разных бизнес-подразделениях для решения разных задач: организация отделов продаж, отделов логистики, построение и эксплуатация колл-центров, и контакт-центров.

IPT Call Center – один из поставщиков услуг в сфере аутсорсинговых цифровых технологий, ориентированных на обслуживание быстрорастущих, революционных, рыночных и технологических компаний, помогающих контролировать взаимодействие, выполнять сложные задачи и развивать свои бренды.

Данная компания предлагает полный аутсорсинг бизнес-процессов, который повышает лояльность к бренду, включая колл-центр, автоответчик, чат, техническую поддержку и поддержку по электронной почте.

В функции контакт-центра включены [6]:

1. Запуск и сопровождение центра бесперебойной связи с клиентами с использованием инструментов call/контакт-центра: телефон, электронная почта, онлайн чат.
2. 100% записанных звонков и хранение записей до 3-х месяцев.
3. Поддержание стандарта обеспечения параметра SLA 80/20, т.е. 80% поступивших звонков в течение 20 секунд.
4. Запуск IVR (англ. Interactive Voice Response, интерактивное голосовое меню) - приветственное голосовое объявление, позволяющее выбирать пункты меню с помощью клавиатуры телефона.
5. Юридическое сопровождение: получение согласия на обработку персональных данных и маркетингового согласия, а также выполнение информационного обязательства GDPR (англ. General Data Protection Regulation общий регламент защиты персональных данных).
6. Индивидуальная оценка в зависимости от масштаба проекта.

По данному варианту контакт-центра можно сделать вывод, что данная компания предоставляет минимальный набор услуг, которые увеличивают ценность и количество клиентов, а также уровень обслуживания.

Также рассмотрим американскую компанию Concetrix. Concetrix – это американская компания, [предоставляющая бизнес-услуги](https://en.wikipedia.org/wiki/Business_services) , специализирующаяся на привлечении клиентов и повышении эффективности бизнеса.

Компания создает инновационные решения, сочетая таланты с технологиями, чтобы помочь установить глубокие связи с клиентами, которые повышают лояльность к бренду и оптимизируют результаты бизнеса. Одной из услуг, предоставляемой данной компанией является контакт-центр.

Контакт-центр Concetrix – это облачное решение, которое обеспечивает последовательное, насыщенное и персонализированное взаимодействие с клиентами, укрепляющее доверие и лояльность к бренду.

Многоуровневые управляемые услуги устраняют проблемы, связанные с управлением решением самостоятельно. Concetrix позаботится об инфраструктуре и операциях контакт-центра, чтобы организация могла сосредоточиться на своем бизнесе. ​

Список функций, включающих в себя контакт-центр [7]:

1. Мониторинг.
2. Оповещения.
3. Продажа билетов.
4. Управление инцидентами.
5. Проактивные коммуникации с клиентами.
6. Управление хранилищем.
7. Отчетность по управлению эффективностью.
8. Интегрированная аналитика данных.
9. Оптимизация решения.

Таким образом в данном разделе был изучен рынок приложений для организации работы контакт-центра. Были рассмотрены аналоги разрабатываемого программного средства и их основные функции.

## 1.3 Техническое задание на создание микросервисного приложения для организации работы контакт-центра

Составим техническое задание на создание микросервисного приложения для организации работы контакт-центра.

Полное наименование системы: микросервисное приложение для организации работы контакт-центра.

Краткое наименование: контакт-центр.

Назначение системы - контакт-центр – это центральная точка, из которой организации управляют всеми взаимодействиями с клиентами по различным каналам связи. Их главное предназначение состоит в том, чтобы предложить клиентам эффективную и действенную [техническую поддержку, наладить обслуживание клиентов](https://www.techtarget.com/searchcustomerexperience/definition/customer-service-and-support) и помочь бизнесу в продажах.

Контакт-центр должен улучшить клиентский опыт, например, клиенты могут воспользоваться возможностями самообслуживания, предоставляемыми контакт-центром, для быстрого выполнения общих задач.

Основным назначением контакт-центра является также повышение эффективности, поскольку контакт-центры являются центральной точкой взаимодействия с клиентами, они могут повысить производительность бизнеса, отвечая на многочисленные запросы клиентов.

Также контакт-центр предназначен для расширения понимания и видимости, другими словами, контакт-центр должен позволить воспользоваться данными, собранными в ходе взаимодействий с клиентами для принятия важных бизнес решений. Эта информация может помочь с дизайном продукта, вопросами качества, а также с выявлением шаблонов и проблемных областей при взаимодействии клиента с компанией.

Микросервисное приложение для организации работы контакт-центра создается с целью:

* улучшить клиентский опыт - [программное обеспечение контакт-центра собирает данные о клиентах](https://www.techtarget.com/searchcustomerexperience/feature/10-examples-of-AI-in-customer-service) из каждого используемого канала и объединяет их в единый профиль клиента. Поскольку большинство клиентов контакт-центра взаимодействуют по нескольким каналам, центр собирает больше данных. Больше данных может позволить [контакт-центру адаптировать клиентский опыт для конкретных абонентов](https://www.techtarget.com/searchcustomerexperience/feature/6-ways-to-build-customer-loyalty-for-your-business) и лучше маршрутизировать вызовы и другие виды связи;
* улучшить информацию о клиентах – контакт-центры могут улучшить [профилирование клиентов](https://www.techtarget.com/searchcustomerexperience/definition/customer-profiling). Когда клиенты взаимодействуют с колл-центрами или контакт-центрами, они делятся информацией о своих личных предпочтениях и поведении, которую агенты могут собирать и использовать для улучшения клиентского опыта при будущих взаимодействиях. Организации также могут интегрировать программное обеспечение CRM с контакт-центром, чтобы собирать больше данных о клиентах и эффективно их анализировать;
* увеличить экономию времени и денег - контакт-центры [позволяют клиентам самостоятельно обслуживать и решать свои проблемы](https://www.techtarget.com/searchcustomerexperience/feature/13-customer-retention-strategies-that-work) с помощью двустороннего обмена мгновенными сообщениями на основе ключевых слов, обмена текстовыми сообщениями или общения с чат- [ботом](https://www.techtarget.com/searchcustomerexperience/definition/chatbot) . Такое самообслуживание сокращает время, которое операторы проводят у телефона, сокращая время ожидания клиентов и снижая общие расходы.

Приложение в виду относительно большого количества функционального обеспечения должно представлять собой микросервисную систему. Каждая отдельная служба должна быть создана в виде отдельного сервиса, абсолютно изолированного от других сервисов.

Система должна поддерживать следующие режимы функционирования:

* основной режим работы операциониста;
* режим работы администратора.

В основном режиме работы операциониста система предоставляет доступ к взаимодействию с входящими и исходящими взаимодействиями.

Режим работы администратора расширяет возможности основного режима работы операциониста посредством открытие дополнительных разделов работы со статистикой. При этом функционал доступный в основном режиме сохраняется.

Таким образом, в данном разделе, согласно техническим требованиям предметной области и особенностям разработки микросервисного программного обеспечения, было разработано техническое задание для разработки микросервисного приложения для организации работы контакт-центра.

## 1.4 Выбор языка и среды разработки микросервисного приложения для организации работы контакт-центра

Для реализации микросервисного приложения для организации работы контакт-центра будет использоваться база данных (далее БД) Microsoft SQL Server. Microsoft SQL Server – это реляционная СУБД. Главной особенностью данной БД является ее возможность поддерживать широкий спектр приложений для обработки транзакций, бизнес-аналитики и аналитики в корпоративных IT-средах. SQL Server привязан к Transact-SQL (далее T-SQL). T-SQL – это процедурный язык. Он добавляет в SQL объявление переменных, управление транзакциями, обработку ошибок и исключений, а также обработку строк, а также добавляет к стандартному языку набор проприетарных программных расширений.

Для взаимодействия приложения с БД используется ORM (Object Relation Mapping) или объектно-реляционное отображение. Этот метод программирования, служит для преобразования несовместимых типов данных в объектно-ориентированные языки программирования. Суть данного метода заключается в том, что создается некая абстракция – «виртуальная объектная база», запросы которой, преобразуется в SQL команды. Классы будут соответствовать таблицам в базе, а экземпляры этих классов – конкретным строкам таблицы. Entity Framework - это объектно-реляционный преобразователь, который позволяет разработчикам .NET работать с базой данных, используя объекты .NET. Это устраняет необходимость в большей части кода доступа к данным, который обычно приходится писать разработчикам. Entity Framework помещается между бизнес-сущностями (доменными классами) и базой данных. Он сохраняет данные, хранящиеся в свойствах бизнес-объектов, а также извлекает данные из базы данных и автоматически преобразует их в объекты бизнес-объектов.

Серверная часть разрабатываемого микросервисного приложения основана на технологии ASP.NET Core MVC (.NET6) – это платформа Microsoft, которая служит для создания веб-приложений и API-интерфейсов с использованием шаблона проектирования Model-View-Controller и не только.

Основные преимущества данной технологии:

1. Открытый исходный код – исходную версию данной платформы можно бесплатно загрузить, изменить и скомпилировать, получив таким образом свою версию данного продукта.
2. Кроссплатформенность - ASP.NET Core MVC является кроссплатформенным как для разработки, так и для развертывания. Он доступен для всех операционных систем — Windows, Linux и macOS.
3. Полный контроль над HTML и HTTP – в данной платформе имеется полный контроль над HTML, это значит, что можно создавать как простые, так и сложные типы HTML, стилизованные с помощью CSS, и отображать их в браузере. Точно так же имеется полный контроль над HTTP-запросами, передаваемыми между браузером и сервером.
4. Расширяемая структура - можно создавать приложения, которые в будущем можно расширить до любых уровней. Ключевые особенности этого фреймворка, которые придают ему расширяемую мощь: компоненты просмотра, вспомогательный функции тегов, маршрутизация.
5. Простая интеграция тестирования - можно разделить различные области приложения на независимые части и тестировать их независимо. Среды тестирования, такие как xUnit и MOQ, можно легко интегрировать для моделирования любого сценария.
6. API - интерфейсы ASP.NET Core MVC могут в полной мере использовать инновации языка и CLR, знакомые программистам на C#, такие как ключевое слово await, методы расширения, лямбда-выражения, анонимные и динамические типы и встроенный в язык запрос (LINQ).

В качестве среды разработки для подобных приложений Microsoft рекомендует использовать разработанную ими Visual Studio, но для разработки данного проекта будет использоваться среда Rider от JetBrains, поскольку данная среда разработки имеет ряд преимуществ:

1. Кроссплатформенность – среда Rider способна работать на различных платформах Windows, macOS и Linux с одинаковой стабильностью, в то время как Visual Studio преимущественно работает на Windows.
2. ReSharper – среда Rider включает в себя функции, популярного расширения ReSharper для Visual Studio, которые позволяют проводить проверки кода, рефакторинг и анализ.
3. Поддержка систем контроля версий – среда Rider поддерживает интеграцию с такими системами контроля версий, как Git, Mercurial и т.д.
4. Подключение к БД.
5. Поддержка возможностей для Front-end разработки.

Таким образом была рассмотрена СУБД для реализации целей работы, в качестве которой была выбрана Microsoft SQL Server, платформа, на которой будет разрабатываться будущее приложение - ASP.NET Core MVC, а также среда разработки Rider от JetBrains.

## 1.5 Вывод по разделу

В данном разделе был осуществлен анализ предметной области, для разрабатываемого микросервисного приложения для организации работы контакт-центра.

На основе анализа предметной области выделена актуальность разработки приложения для организации работы контакт-центра.

Аналитический обзор позволил сравнить существующее ПО со схожим функционалом как у разрабатываемого приложения. Преимущества и недостатки рассматриваемых систем указали на возможные преимущества и недостатки разрабатываемой системы.

На основе предметной области приложения было разработано техническое задание для правильной реализации микросервисного приложения для организации работы контакт-центра.

# 2 Проектирование микросервисного приложения для организации работы контакт-центра

## **2.1 Анализ требований на основе диаграммы прецедентов и сценария использования микросервисного приложения для организации работы контакт-центра**

Для более детального проектирования разрабатываемой системы необходимо обратиться к UML-языку проектирования, благодаря которому в минимальные сроки будет возможным спланировать все нюансы будущей информационной системы.

Моделирование в UML можно представить как некоторый процесс поуровневого спуска от наиболее обшей и абстрактной концептуальной модели исходной системы к логической, а затем и к физической модели, соответствующей программной системы.

Одной из основных диаграмм в языке UML является диаграмм прецедентов. Эта диаграмма отображает отношения между различными пользователями системами(актерами) назначение системы или, другими словами, то, что система будет делать в процессе своего функционирования.

Для отображения основных функциональных возможностей системы, благодаря которым пользователей может получить необходимый результат, была разработана диаграмма прецедентов, изображенная на рисунке ниже (Рисунок 2.1) и демонстрирующая основные прецеденты и их взаимосвязи между пользователями системы в рамках данной выполняет программы.

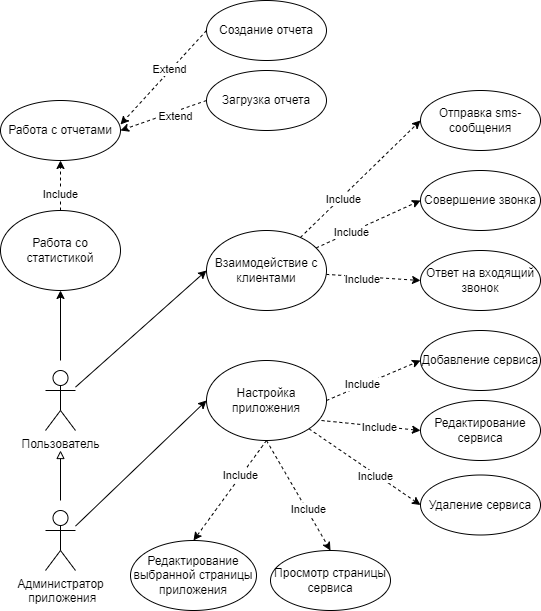


Рисунок 2.1 – сущностей Диаграмма автоматически прецедентов

Благодаря проектированию данной UML-диаграммы было сформировано представление о структуре будущей информационной системы, а также отображены основные связи между основными функциональными прецедентами системы. Информация, полученная при построении данной диаграммы, упростит и ускорит процесс дальнейшей разработки проекта приложения.

На основе существующей диаграммы прецедентов разрабатываются сценарии вариантов использования, то есть детальное описание каждого возможного прецедента. Данная процедура помогает предоставить удобное и понятное техническое задание разработчику, который, зачастую, не должен вникать в бизнес-логику информационной системы.

Сценарий: Отправка сообщения.

Актеры: Пользователь, Система.

Цель: отправить сообщение.

Предусловие: Профиль пользователя не авторизован.

Краткое описание: Пользователь пользуется своим профилем для отправки сообщения. Пользователь проходит авторизацию в системе, далее запускает необходимую процедуру после чего система совершает отправку письма по указанному номеру.

Тип: Базовый

Раздел – «Типичный ход событий»:

Действия актеров:

1. Пользователь попадает в Окно Авторизации.
2. Пользователь вводит свои данные для авторизации (логирования) в соответствующие поля приложения.
3. Система создает подключение к БД для того, чтобы сверить введённые данные с имеющимися в БД.

Исключение №1: Ошибка авторизации, не удалось создать подключение к БД.

1. Система сверяет введенные данные с имеющимися в БД.

Исключение №2: Ошибка авторизации, введенные данные не найдены в БД.

1. Система выдает сообщение пользователю о успешной авторизации.
2. Пользователь вводит номер телефона и текст сообщения и кликает на соответствующую кнопку для его отправки.
3. Система отправляет сообщение.

Исключение №3: При отправке сообщения произошла ошибка, которая не дает возможности совершить отправку.

1. Система выводит на экран пользователя сообщение об успешной отправке.
2. Система готова к дальнейшему использованию.

Раздел – «Исключения»:

Действия актеров:

Исключение №1: Ошибка авторизации, не удалось создать подключение к БД.

1. Система выдаёт сообщение о соответствующей ошибке и возвращает пользователя на шаг 2 (с сохранением введенных данных).
2. Пользователь изменяет свои данные для авторизации (логирования) в соответствующих полях приложения.

Исключение №2: Ошибка авторизации, введенные данные не найдены в БД.

1. Система выдаёт сообщение о соответствующей ошибке и возвращает пользователя на шаг 2 (с сохранением введенных данных).
2. Пользователь изменяет свои данные для авторизации (логирования) в соответствующих полях приложения.

Исключение №3: При отправке сообщения произошла ошибка, которая не дает возможности совершить отправку.

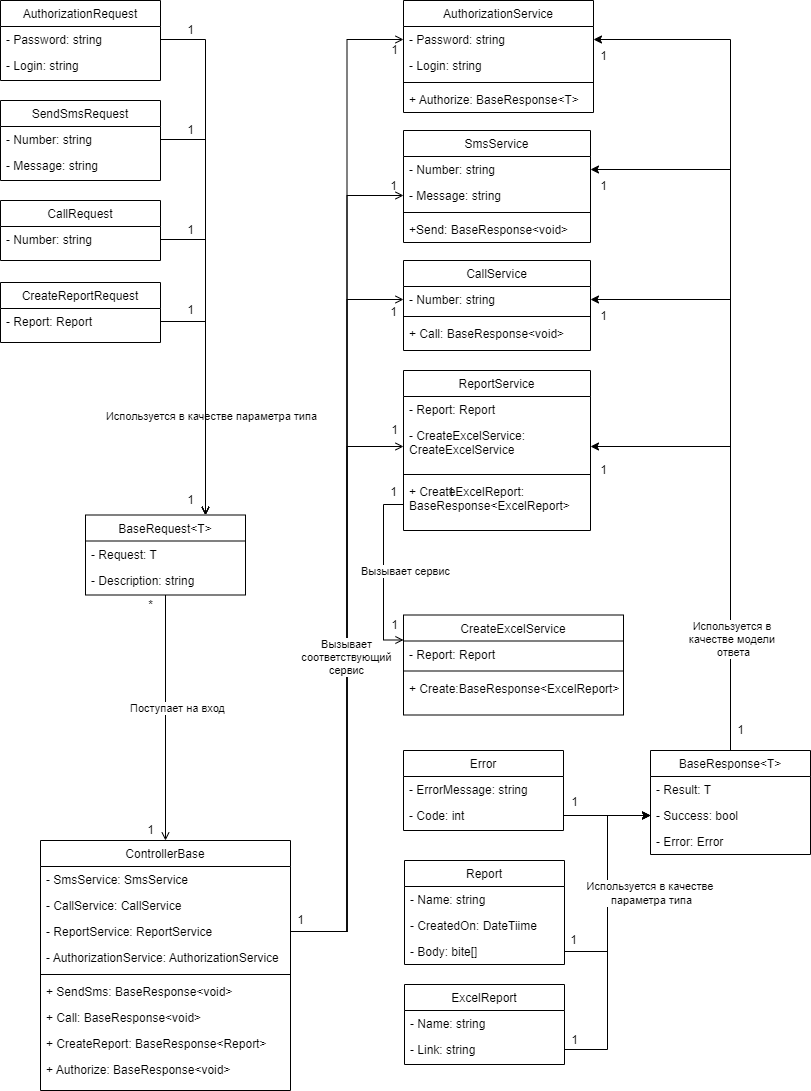
1. Система выдаёт сообщение о соответствующей ошибке и возвращает пользователя на шаг 6.
2. Пользователь исправляет ошибку и снова отправляет сообщение.

Таким образом, была рассмотрена диаграмма прецедентов, содержащая в себе одну роль. Была построена модель функционирующей системы с учетом особенностей темы.

## 2.2 Построение диаграммы классов микросервисного приложения для организации работы контакт-центра

Для построения модели предметной области отлично подходит еще одна структурная диаграмма UML-языка проектирования – диаграмма классов.

Диаграмма классов для микросервисного приложения для организации работы контакт-центра представлена на рисунке 2.3.



элементоразделу Рисунок 2.3 – Диаграмма классов разрабатываемого приложения

Диаграмма классов составляется до начала работы над программным кодом приложения, поэтому необходимо ее делать максимально абстрактной, но, при этом, необходимо максимально точно выразить функционал классов и их взаимосвязи между собой. Для упрощения понимания данной диаграммы также составляется специальная таблица, описывающая поведение конкретных классов в проектируемой системе. Описание анализа классов низкий приведено в таблице 2.1.

секунд Таблица 2.1 – структура Описание затрат классов микросервисного приложения для организации работы контакт-центра

|  |  |
| --- | --- |
| этому Наименование | рисунок Описание |
| BaseRequest<T> | Класс-модель, необходимый для передачи информации запроса с клиентской части приложения на серверную, содержит в себе описание в виде текстовой информации, а также сам запрос, имеющий один из типов, созданных для отправки запросов. |
| AuthorizationRequest | Класс, который является типом запроса на авторизацию пользователя. Содержит в себе информацию о данных пользователя, а именно об логине и пароле в виде текстовых данных. Данный класс используется при определении класса BaseRequest в качестве параметра типа при отправке запроса с клиентской части на серверную. |
| SendSmsRequest | Класс, который является типом запроса для отправки сообщения клиенту. Содержит в себе информацию о номере телефона, на который необходимо отправить сообщение и сам текст сообщения, в виде текстовых данных. Данный класс используется при определении класса BaseRequest в качестве параметра типа при отправке запроса с клиентской части на серверную. |

Продолжение таблицы 2.1

|  |  |
| --- | --- |
| CallRequest | Класс, который является типом запроса для звонка клиенту. Содержит в себе информацию о номере телефона, на который необходимо совершить звонок, в виде текстовых данных. Данный класс используется при определении класса BaseRequest в качестве параметра типа при отправке запроса с клиентской части на серверную. |
| CreateReportRequest | Класс, который является типом запроса для создания отчета. Содержит в себе информацию о создаваемом отчете виде данных класса Report. Данный класс используется при определении класса BaseRequest в качестве параметра типа при отправке запроса с клиентской части на серверную. |
| ControllerBase | Главный класс взаимодействия пользователя с приложением. Класс служит для приема http-запросов с клиентской части на серверную и выполняет роль маршрутизатора, то есть вызывает сервис, соответствующий полученному запросу, и передает в него полученные данные, а также принимает ответ о выполненной операции сервисом и передает его на клиентскую часть приложения, если это необходимо. Содержит в себе экземпляры классов, которые реализуют, необходимые для работы приложения сервисы. |

Продолжение таблицы 2.1

|  |  |
| --- | --- |
| AuthorizationService | Класс инкапсулирует механизм авторизации в приложении. Данный сервис вызывается из основного класса ControllerBase, на вход принимает запрос типа BaseRequest<AuthorizationRequest>. После выполнения своей задачи данный класс формирует ответ, имеющий тип BaseResponse, и передает его обратно вызывающему классу - ControllerBase. |
| SmsService | Класс инкапсулирует механизм отправки смс клиенту. Данный сервис вызывается из основного класса ControllerBase, на вход принимает запрос типа BaseRequest<SmsRequest>. После выполнения своей задачи данный класс формирует ответ, имеющий тип BaseResponse, и передает его обратно вызывающему классу - ControllerBase. |
| CallService | Класс инкапсулирует механизм совершения звонка клиенту. Данный сервис вызывается из основного класса ControllerBase, на вход принимает запрос типа BaseRequest<CallRequest>. После выполнения своей задачи данный класс формирует ответ, имеющий тип BaseResponse, и передает его обратно вызывающему классу - ControllerBase. |
| ReportService | Класс инкапсулирует механизм работы с отчетами. Данный сервис вызывается из основного класса ControllerBase, на вход принимает запрос типа BaseRequest<CreateReportRequest>. После выполнения своей задачи данный класс формирует ответ, имеющий тип BaseResponse<Report>, и передает его обратно вызывающему классу - ControllerBase. |

Продолжение таблицы 2.1

|  |  |
| --- | --- |
| CreateExcelService | Класс инкапсулирует механизм работы с Excel-данными. Данный сервис вызывается из основного сервиса ReportService, на вход принимает запрос типа Report. После выполнения своей задачи данный класс формирует ответ, имеющий тип ExcelReport, и передает его обратно вызывающему классу - ReportService. |
| BaseResponse<T> | Класс-модель, необходимый для передачи информации ответа с серверной части приложения на клиентскую, содержит в себе результат в виде булевского значения, ошибку типа Error, а также сам результат, имеющий один из типов, созданных для возврата ответа. |
| Error | Класс, который является типом для хранения информации об ошибке. Содержит в себе информацию о об ошибке в текстовом формате и код ошибки, представляющий собой числовое значение. Данный класс используется при отправке ответа в случае, если во время работы приложения произошла ошибка. |
| Report | Класс, который является типом для передачи информации об отчете. Содержит в себе название, создаваемого отчета в виде текстовых данных, дату его создания, а также сам отчет представляемый собой массив байт. Данный класс используется при отправке ответа в качестве результата работы сервиса по построению отчетов. |

Продолжение таблицы 2.1

|  |  |
| --- | --- |
| ExcelReport | Класс, который является типом для передачи Excel-данных. Содержит в себе название, создаваемого отчета в виде текстовых данных, а также ссылку на созданный отчет. Данный класс в качестве типа результата, возвращаемого сервисом CreateExcelReport |

Таким образом, в данном разделе была построена модель предметной области будущего микросервисного приложения для организации работы контакт-центра посредством разработки диаграммы классов, кроме того, для упрощения восприятия данной диаграммы было дано краткое описание наполнения и поведения для каждого из классов.

## 2.3 Построение разработки диаграммы операционную последовательности привлечь микросервисного приложения для организации работы контакт-центра

Диаграммы последовательности (sequence diagram) являются видом диаграмм взаимодействия языка UML, которые описывают отношения объектов в различных условиях. Условия взаимодействия задаются сценарием, полученным на этапе разработки диаграмм вариантов использования.

Выполненная диаграмма последовательностей микросервисного приложения для организации работы контакт-центра представлена на рисунке ниже (Рисунок 2.4). На диаграмме последовательности продемонстрирован процесс взаимодействия класса-контроллера с классами сервисами при осуществлении запроса на создание отчета.

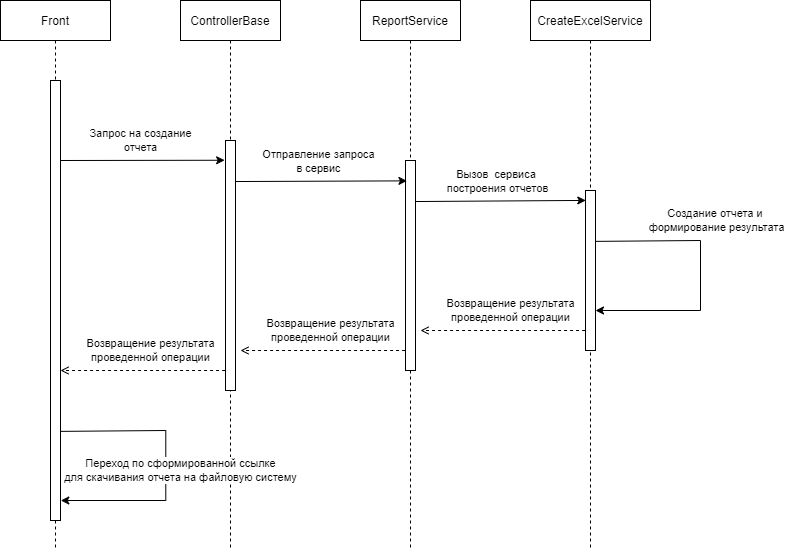


Рисунок 2.4 – Диаграмма последовательности

Из диаграммы видно, что инициатором осуществления процессов является пользователь, использующий веб-интерфейс. Кроме того, видно, что запрос обрабатывается тремя разными классами. Это сделано для того, чтобы отделить бизнес-логику от технической реализации создания отчета. После получения результата работы программы происходит автоматическое скачивание файла.

Таким образом, в результате построения диаграммы последовательности микросервисного приложения для организации работы контакт-центра разработаны подробные описания процессов внутри приложения, которые будут отражать функциональную ценность информационной системы.

## 2.4 Расчет функциональных и размерно-ориентированных метрик для микросервисного приложения для организации работы контакт-центра

Для количественной оценки разрабатываемого приложения используют LOC-оценку. LOC-оценка используется для измерения объема программного обеспечения путем подсчета количества строк исходного кода.

Цель этой деятельности - сформировать предварительные оценки, которые позволят:

* предъявить заказчику корректные требования по стоимости и затратам на разработку программного продукта;
* составить план программного проекта.

В оценке стоимости ПО используются 2-е единицы оценки: функциональная точка Function Point (FP) и строка кода Line of Code (LOC).

При расчете FP и LOC оценок необходимо выполнить анализ предварительной трудоемкости и стоимости разработки программы.

Используется 5 информационных характеристик описание которых представлено ниже, а их оценка в таблице 1.

* 1. Количество внешних вводов. Подсчитываются все вводы пользователя, по которым поступают разные прикладные данные. Вводы должны быть отделены от запросов, которые подсчитываются отдельно.
  2. Количество внешних выводов. Подсчитываются все выводы, по которым к пользователю поступают результаты, вычисленные программным приложением. В этом контексте выводы означают отчеты, экраны, распечатки, сообщения об ошибках. Индивидуальные единицы данных внутри отчета отдельно не подсчитываются.
  3. Количество внешних запросов. Под запросом понимается диалоговый ввод, который приводит к немедленному программному ответу в форме диалогового вывода. При этом диалоговый ввод в приложении не сохраняется, а диалоговый вывод не требует выполнения вычислений. Подсчитываются все запросы - каждый учитывается отдельно.
  4. Количество внутренних логических файлов. Подсчитываются все логические файлы (то есть логические группы данных, которые могут быть частью базы данных или отдельным файлом).
  5. Количество внешних интерфейсных файлов. Подсчитываются все логические файлы из других приложений, на которые ссылается данное приложение.
  6. Каждой из выявленных характеристик ставится в соответствие сложность. Для этого характеристике назначается низкий, средний или высокий ранг, а затем формируется числовая оценка ранга.

Данные для определения ранга и оценки сложности транзакций и файлов приведены в таблицах:

Таблица 2.2 – Ранг и оценка сложности внешних вводов микросервисного приложения для организации работы контакт-центра

|  |  |  |  |
| --- | --- | --- | --- |
| Ссылки на файлы | Элементы данных | | |
|  | 1-4 | 5-15 | >15 |
| 0-1 | Низкий (3) | Низкий (3) | Средний (4) |
| 2 | Низкий (3) | Средний (4) | Высокий (6) |
| >2 | Средний (4) | Высокий (6) | Высокий (6) |

Таблица 2.3 – Ранг и оценка сложности внешних выводов

|  |  |  |  |
| --- | --- | --- | --- |
| Ссылки на файлы | Элементы данных | | |
|  | 1-4 | 5-19 | >19 |
| 0-1 | Низкий (4) | Низкий (4) | Средний (5) |
| 2-3 | Низкий (4) | Средний (5) | Высокий (7) |
| >3 | Средний (5) | Высокий (7) | Высокий (7) |

Таблица 2.4 – Ранг и оценка сложности внешних запросов

|  |  |  |  |
| --- | --- | --- | --- |
| Ссылки на файлы | Элементы данных | | |
|  | 1-4 | 5-19 | >19 |
| 0-1 | Низкий (3) | Низкий (3) | Средний (4) |
| 2-3 | Низкий (3) | Средний (4) | Высокий (6) |
| >3 | Средний (4) | Высокий (6) | Высокий (6) |

Таблица 2.5 – Ранг и оценка сложности внутренних логических файлов

|  |  |  |  |
| --- | --- | --- | --- |
| Типы элементов записей | Элементы данных | | |
|  | 1-19 | 20-50 | >50 |
| 1 | Низкий (7) | Низкий (7) | Средний (10) |
| 2-5 | Низкий (7) | Средний (10) | Высокий (15) |
| >5 | Средний (10) | Высокий (15) | Высокий (15) |

Таблица 2.6 – Ранг и оценка сложности внешних интерфейсных файлов

|  |  |  |  |
| --- | --- | --- | --- |
| Ссылки на файлы | Элементы данных | | |
|  | 1-19 | 20-50 | >50 |
| 1 | Низкий (5) | Низкий (5) | Средний (7) |
| 2-5 | Низкий (5) | Средний (7) | Высокий (10) |
| >5 | Средний (7) | Высокий (10) | Высокий (10) |

Таблица 2.7 – Исходные данные для расчета FP-оценки

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Имя характеристики | Ранг, количество, сложность | | | |
| Низкий | Средний | Высокий | Итого |
| Внешние вводы | 0\*3=0 | 0\*4=20 | 10\*6=0 | 60 |
| Внешние выводы | 0\*4=0 | 0\*5=0 | 10\*7=70 | 70 |
| Внешние запросы | 1\*3=3 | 0\*4=0 | 0\*6=0 | 3 |
| Внутренние логические файлы | 0\*7=0 | 35\*10=350 | 0\*15=0 | 350 |
| Внешние интерфейсные файлы | 0\*5=0 | 0\*7=0 | 0\*10=0 | 0 |
| Всего | | | | 483 |

Кол-во функциональных указателей вычисляется по формуле 2.1:

(2.1),

где Fi – коэффициенты регулировки сложности в диапазоне:

* 0 – не влияет;
* 1 – случайное влияние;
* 2 – небольшое влияние;
* 3 – среднее влияние;
* 4 – важное влияние;
* 5 – основное влияние;

Значение коэффициентов регулировки сложности представлены в таблице 2.8.

Таблица 2.8 – Значение коэффициентов регулировки сложности

|  |  |  |  |
| --- | --- | --- | --- |
| № | Системный параметр | Описание | Коэффициент |
| 1 | Передача данных | Сколько средств связи требуется для передачи или обмена информацией, или с приложением или системой? | 4 |
| 2 | Распределенная обработка данных | Как обрабатываются распределенные данные и функции обработки? | 3 |
|  |  |  |  |

Продолжение таблицы 2.8

|  |  |  |  |
| --- | --- | --- | --- |
| 3 | Производительность | Нуждается пользователь в фиксации времени ответа или производительности? | 4 |
| 4 | Распространённость используемой конфигурации | Насколько распространена текущая аппаратная платформа, на которой будет выполняться приложение? | 4 |
| 5 | Скорость транзакций | Как часто выполняются транзакции? (каждый день, каждую неделю, каждый месяц) | 4 |
| 6 | Оперативный ввода данных | Какой процент информации надо вводить в режиме онлайн? | 3 |
| 7 | Эффективность работы конечного пользователя | Приложение проектировалось для обеспечения эффективной работы конечного пользователя? | 5 |
| 8 | Оперативное обновление | Как много внутренних файлов обновляется в онлайновой транзакции? | 4 |
| 9 | Сложность обработки | Выполняет ли приложение интенсивную логическую или математическую обработку? | 1 |
| 10 | Повторная используемость | Приложение разрабатывалось для удовлетворения требований одного или многих пользователей? | 1 |
| 11 | Легкость инсталляции | Насколько трудны преобразование и инсталляция приложения? | 1 |
| 12 | Легкость эксплуатации | Насколько эффективны и/или автоматизированы процедуры запуска, резервирования и восстановления? | 3 |
| 13 | Разнообразные условия размещения | Была ли спроектирована, разработана и поддержана возможность инсталляции приложения в разных местах для различных организаций? | 4 |
| 14 | Простота изменений | Была ли спроектирована, разработана и поддержана в приложении простота изменений? | 3 |
| Всего | | | 44 |

Теперь можно рассчитать количество функциональных указателей:

.

Так как это количество функциональных указателей, округляем его до ближайшего целого и получаем 527 функциональных точек.

Одним из преимуществ функциональной оценки является то, что при небольших манипуляциях ее можно преобразовать в размерно-ориентированную оценку. Для данного преобразования используются специальные таблицы конвертации (таблица 2.9). Для расчета LOC-оценки необходимо умножить значение из таблицы для используемого языка на количество функциональных метрик.

Зная кол-во функциональных указателей, можем получить число строк кода. В языке C# одна функциональная точка примерно равна 53 строкам кода (Таблица 2.9).

Таблица 2.9 – Кол-во операторов на один FP

|  |  |
| --- | --- |
| Язык программирования | Кол-во операторов на один FP |
| Assembler | 320 |
| C | 128 |
| Cobol | 106 |
| Fortran | 106 |
| Pascal | 90 |
| C++ | 64 |
| LISP | 64 |
| Prolog | 64 |
| C# | 53 |
| Java | 53 |
| Kotlin | 49 |
| Visual Basic | 32 |
| Smalltalk | 22 |
| Perl | 21 |

Формула для пересчета из FP в LOC:

LOC = FP \* Количество Операторов

Получим, что число строк кода микросервисного приложения для организации работы контакт-центра равно:

LOC = 527 \* 53 = 27931 строк кода.

Таким образом в данном разделе были вычислены основные оценки проектируемой системы – размерно-ориентированные и функциональные, кроме того, были выявлены их преимущества и недостатки по сравнению с другими способами оценки разрабатываемого приложения.

## 2.5 Расчет модели оценки стоимости разработки микросервисного приложения для организации работы контакт-центра

Очень важным показателем при разработке какого-либо программного продукта является время разработки системы. Для определения предварительного времени разработки используют ранее рассчитанную размерно-ориентированную оценку.

Constructive cost model (COCOMO – модель издержек разработки) – это алгоритмическая модель оценки стоимости разработки программного обеспечения.

Основное уравнение этой модели имеет вид (формула 2.2):

(2.2)

где:

* А = 2.5 – масштабный коэффициент;
* Размер – размер ПО выраженный в тыс. LOC;
* Me – множитель поправки зависит от семи формирователей затрат, характеризирующих продукт, процесс и персонал (Таблица 2.5);
* Показатель степени B отражает нелинейную зависимость затрат от размера проекта (от длины кода LOC).

Значение показателя степени B измеряется в диапазоне от 1.01 до 1.26, зависит от 5-и масштабных факторов Wi и вычисляется по формуле 2.3:

. (2.3)

Общая характеристика масштабных факторов Wi приведена в таблице 2.10.

Таблица 2.10 – Характеристика масштабных факторов Wi

|  |  |  |
| --- | --- | --- |
| Масштабный фактор Wi | Пояснение | Wi |
| 1 Предсказуемость, наличие прецедентов | Отражает предыдущий опыт организации в реализации проектов этого типа. Очень низкий (=5) означает отсутствие опыта. Сверхвысокий (=0) означает, что организация полностью знакома с этой прикладной областью | 3-среднее |

Продолжение таблицы 2.10

|  |  |  |
| --- | --- | --- |
| 2 Гибкость разработки | Отражает степень гибкости процесса разработки. Очень низкий (=5) означает, что используется заданный процесс. Сверхвысокий (=0) означает, что клиент установил только общие цели | 3-среднее |
| 3 Разрешение рисков в архитектуре | Отражает степень выполняемого анализа риска. Очень низкий (=5) означает малый анализ. Сверхвысокий (=0) означает полный и сквозной анализ риска | 4-среднее |
| 4 Связность группы | Отражает, насколько хорошо разработчики группы знают друг друга, и насколько удачно они совместно работают. Очень низкий (=5) означает очень трудные взаимодействия. Сверхвысокий (=0) означает интегрированную группу без проблем взаимодействия | 1-очень высокий |
| 5 Зрелость процесса | Означает зрелость процесса в организации. Вычисление этого фактора может выполняться по вопроснику CMM | 3-среднее |
| Всего | | 14 |

Из таблицы 2.10 следует что численное значение показателя B = 1,15.

Множитель поправки Ме зависит от набора формирователей затрат *ЕМi,* перечисленных в таблице 2.11.

Таблица 2.11 – Формирователи затрат для раннего этапа проектирования

|  |  |  |
| --- | --- | --- |
| Обозначение | Название | EMi |
| PERS | Возможности (способности) персонала | Среднее=1 |
| RCPX | Надежность и сложность продукта | Среднее=1 |
| RUSE | Требуемое повторное использование | Среднее=1 |
| PDIF | Трудность (сложность) платформы | Среднее=1 |
| PREX | Опытность персонала | Среднее=1 |
| FCIL | Средства поддержки | Среднее=1 |
| SCED | Сроки | Среднее=1 |

Для расчета множителя поправки, зависящего от 7-и формирователей затрат, характеризующих продукт, процесс и персонал, воспользуемся формулой 2.4:

. (2.4)

Из формулы 2.4 следует что множитель поправки равен 1s.

Для расчета затрат на разрабатываемый продукт, необходимо воспользоваться формулой 2.2. В итоге получим:

В итоге проведённые расчетов можно сделать вывод что, один человек может выполнить поставленную задачу за 7,5 месяцев.

однипользователь Таким данных образом в данном разделе при помощи COCOMO-модели издержек разработки и вычисленной ранее размерно-ориентированной оценки были выявлены предполагаемые сроки разработки микросервисного приложения для организации работы контакт-центра.

## 2.6 Проектирование базы данных микросервисного приложения для организации работы контакт-центра

Разрабатываемое микросервисное приложения для организации работы контакт-центра в качестве хранилища данных должно иметь БД, в пункте 3.1 была выбрана и описана СУБД SQL Server.

В базе данных для каждой сущности приложения формируется таблица, если это необходимо, каждая таблица имеет колонки в соответствии с полями и атрибутами данной сущности (Рисунок 2.5).

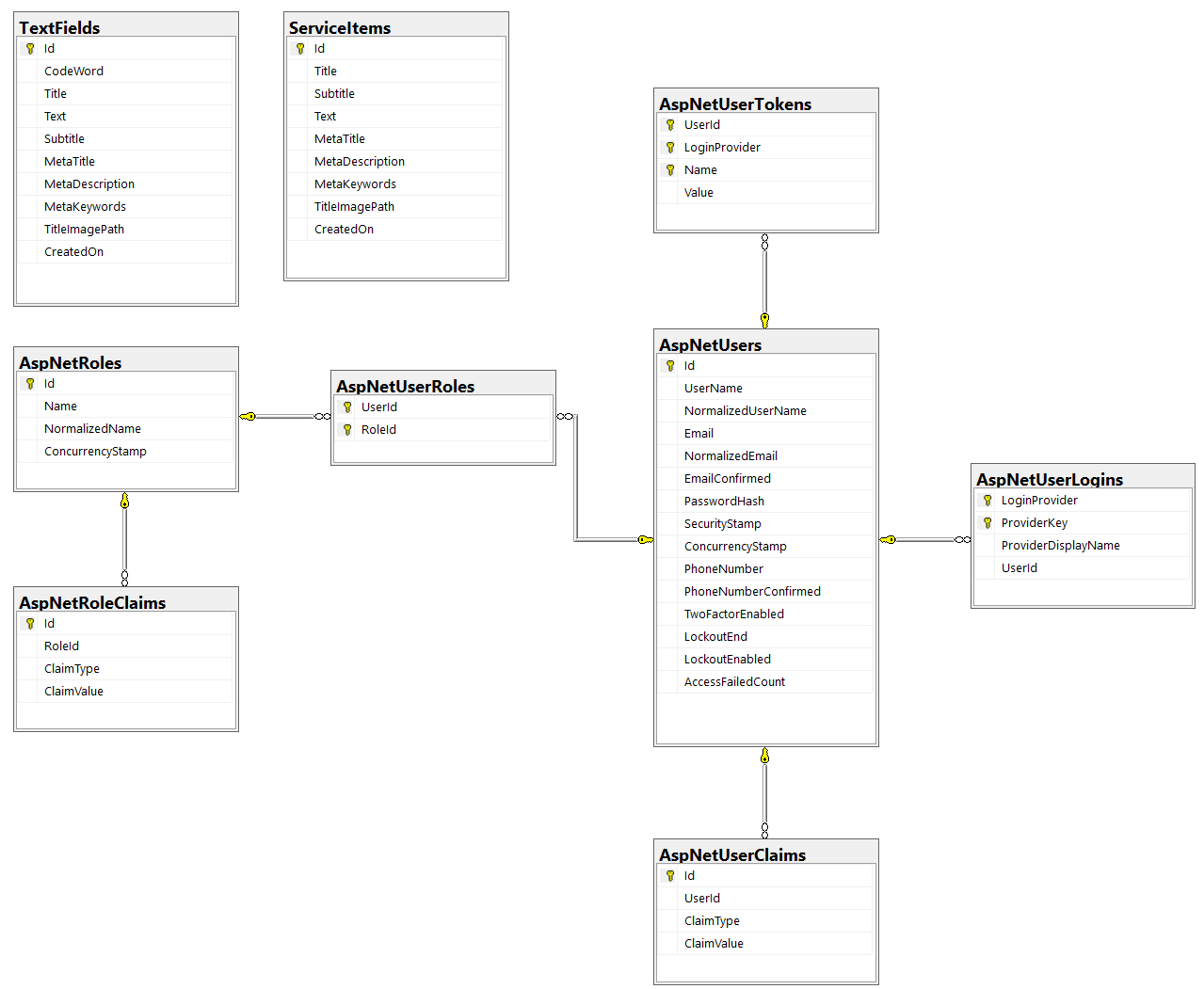


Рисунок 2.5 – Диаграмма базы данных разрабатываемого приложения

Из диаграммы базы данных на рисунке 2.5 видно, что было создано девять таблиц:

1. TextFields – таблица, в которой содержатся данные о текстовых полях, расположенных на сайте. Содержит десять полей. Не связана с другими таблицами.
2. ServiceItems – таблица, в которой хранятся данные, о всех доступных сервисах. Содержит девять полей. Не связана с другими таблицами.
3. AspNetUsers – таблица, в которой хранятся данные о пользователях. Содержит пятнадцать полей, не имеет ссылок на другие таблицы.
4. AspNetTokens – таблица, в которой хранятся данные о токенах пользователей. Содержит четыре поля. Имеет внешний ключ на таблицу пользователей.
5. AspNetUserLogins – таблица, в которой хранятся данные о связи пользователей с из логинами. Содержит четыре поля. Имеет внешний ключ на таблицу пользователей.
6. AspNetUserClaims - таблица, в которой хранятся данные о связи пользователей с объектами Claim (Отдельные данные о пользователе). Содержит 4 поля. Имеет внешний ключ на таблицу пользователей.
7. AspNetUserRoles – является развязочной таблицей, имеющей ссылки на таблицу пользователей и на таблицу ролей. Содержит два поля.
8. AspNetRoles – таблица ролей. Содержит четыре поля. Не имеет ссылок на другие таблицы.
9. AspNetRolesClaims – таблица, которая хранит информацию о связи ролей и их Claim. Содержит четыре поля. Имеет ссылку на таблицу ролей.

Таким образом, спроектирована БД микросервисного приложения для организации работы контакт-центра, в которой содержится девять таблиц.

## 2.7 Вывод по второму разделу

В данном разделе был выполнен аналитический обзор требований при разработке микросервисного приложения для организации работы контакт-центра на основе диаграммы прецедентов, который заключался в определении главных ролей, актеров, прецедентов и их взаимодействии, а также в построении самой диаграммы прецедентов.

Был выполнен анализ структурных принципов проектирования приложения и модели предметной микросервисного приложения для организации работы контакт-центра, на основе которых, были построены диаграммы классов и последовательности, и определены главные функциональные задачи. Осуществлены расчеты функционально- и размерно-ориентированных метрик для микросервисного приложения контакт-центр (FP, LOC, COCOMO). В итоге проведённые расчетов можно сделать вывод что, один человек может выполнить поставленную задачу за 7,5 месяцев.

# 3 Программная реализация приложения для организации работы контакт-центра

## 3.1 Интерфейс микросервисного приложения для организации работы контакт-центра

Веб-интерфейс микросервисного приложения для организации работы контакт-центра содержит в себе множество полей ввода и вывода, кнопок и других стандартных элементов любой страницы. В этом пункте будут описаны все эти элементы на интерфейсе.

При запуске приложения пользователь попадает на страницу авторизации. На этой странице находится два поля для ввода текста (далее - текстбокс) для ввода логина и пароля пользователя (Рисунок 3.1). Кроме того, панель содержит в себе одну кнопку для инициации авторизации(«Войти»), а также чекбокс («Запомнить меня?»), который позволяет запомнить пользователя, и при дальнейшем входе в приложение не требовать логин и пароль.



Рисунок 3.1 – Панель авторизации

Навигация по разделам приложения происходит по основному меню, которое расположено в верхней части страницы. Каждый пункт меню является кнопкой, которая производит загрузку соответствующего контента   
(Рисунок 3.2).

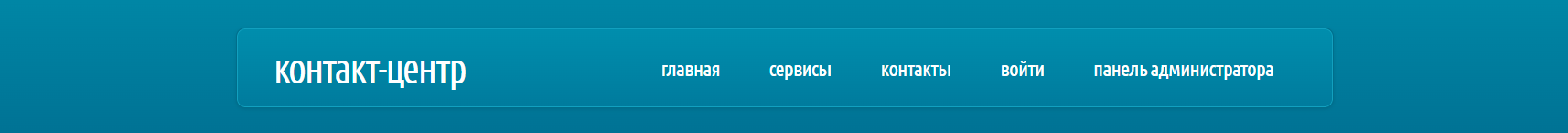


Рисунок 3.2 – Основное меню приложения

На главной странице приложения находится основная информация, которую способен редактировать только администратор приложения (Рисунок 3.2).



Рисунок 3.3 – Главная страница

Если авторизованный пользователь не является администратором приложения, то при попытке войти на панель администратора его перенаправит на окно авторизации (Рисунок 3.1). Если при авторизации были введены данные администратора, то становится доступна панель администратора (Рисунок 3.4). На панели администратора можно увидеть два основных раздела – раздел доступные сервисы и раздел редактирования страниц сайта. В разделе доступные сервисы администратор имеет возможность добавить новый сервис, который будет использоваться в приложении по кнопке «Добавить сервис». Также администратор имеет возможность редактировать уже существующие сервисы по кнопке «Редактировать» и удалять их по кнопке «Удалить». В разделе редактирования страниц администратор может открыть любую из них на редактирование по нажатию на название странице в предложенном списке. Также администратор может выйти из своего аккаунта по кнопке «Выйти».

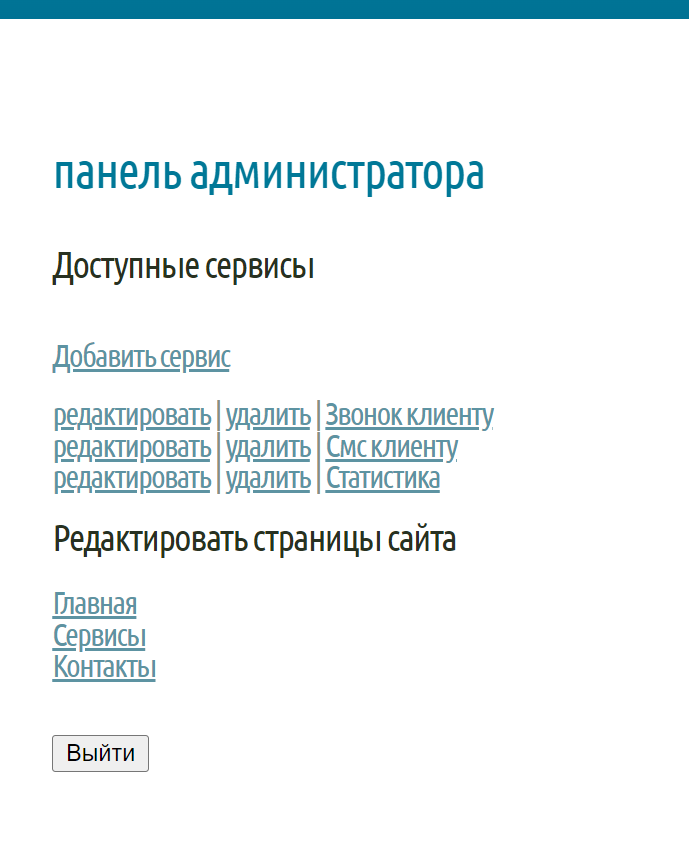


Рисунок 3.4 Панель администратора

На странице сервисов распложена информация о каждом сервисе, который доступен на данный момент операционисту, информация представлена в виде списка для удобства использования, каждый сервис выделятся с помощью отличительной иконки, которая закрепляется за ним, также можно увидеть полное название сервиса и его краткое описание (Рисунок 3.10).

При нажатии на кнопку «Добавить сервис» происходит открытие страницы создания сервиса (Рисунок 3.5), где можно увидеть следующие текстбоксы:

* Название сервиса;
* Краткое описание сервиса;
* Полное описание сервиса;
* Метатег названия сервиса;
* Метатег описание;
* Метатег ключевое слово.

Также на странице расположены две функциональные кнопки – первая «Выберите файл» для выбора иконки будущего сервиса, и вторая «Сохранить» для сохранения всей записанной информации.

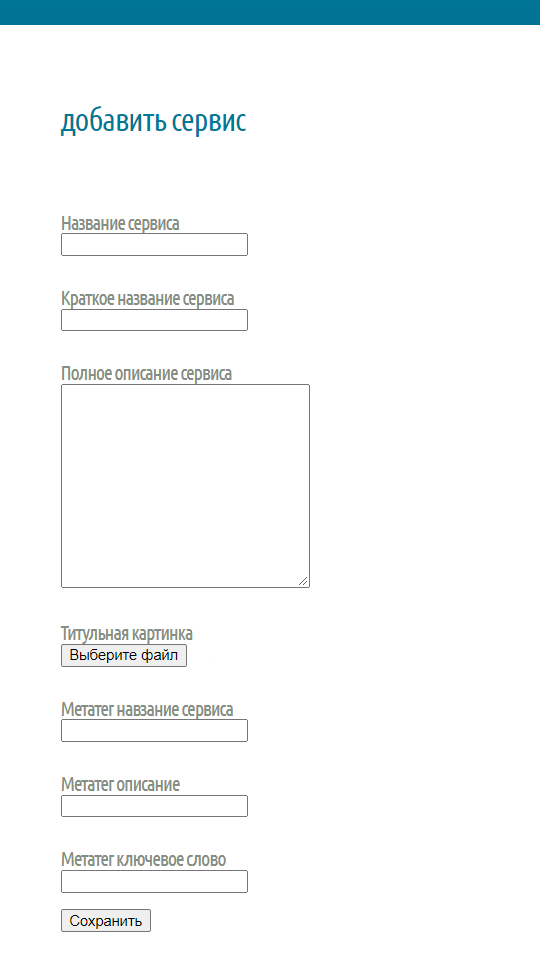


Рисунок 3.5 – Панель создания нового сервиса

При нажатии на кнопку «Редактировать» происходит открытие страницы, похожей на ту страницу, что открывается при создании нового сервиса, только уже с предзаполненной информацией, которую можно отредактировать (Рисунок 3.6)



Рисунок 3.6 – Страница редактирования сервиса

При нажатии на кнопку «Удалить» на панели администратора в разделе просмотра доступных сервисов происходит удаление всей информации о том или ином сервисе.

При нажатии в разделе редактирования страниц сайта на название выбранной страницы происходит открытие панели редактирования (Рисунок 3.7), которая для каждой сущности одинакова, на ней расположены следующие текстбоксы:

* Название страницы (заголовок);
* Содержание страницы;
* Метатег название страницы;
* Метатег описание;
* Метатег ключевое слово.

Также на странице расположена одна кнопка «Сохранить», которая позволяет отредактировать запись по данной сущности в БД.

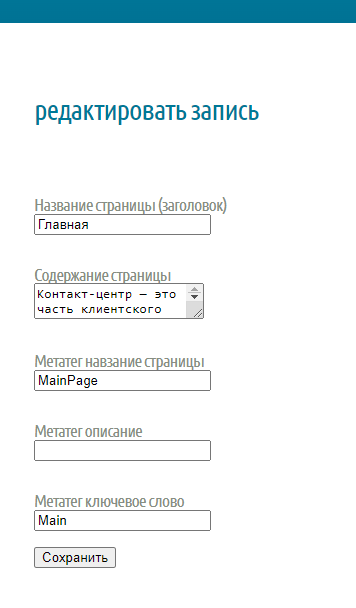


Рисунок 3.7 – Панель редактирование информации о странице

В правой части приложения на каждой странице расположено отдельное меню с доступными сервисами, данное меню сделано для быстрого перехода операциониста к необходимой функциональности (Рисунок 3.8). Абсолютно каждая иконка того или иного сервиса является кнопкой и при нажатии открывает страницу соответствующего сервиса.

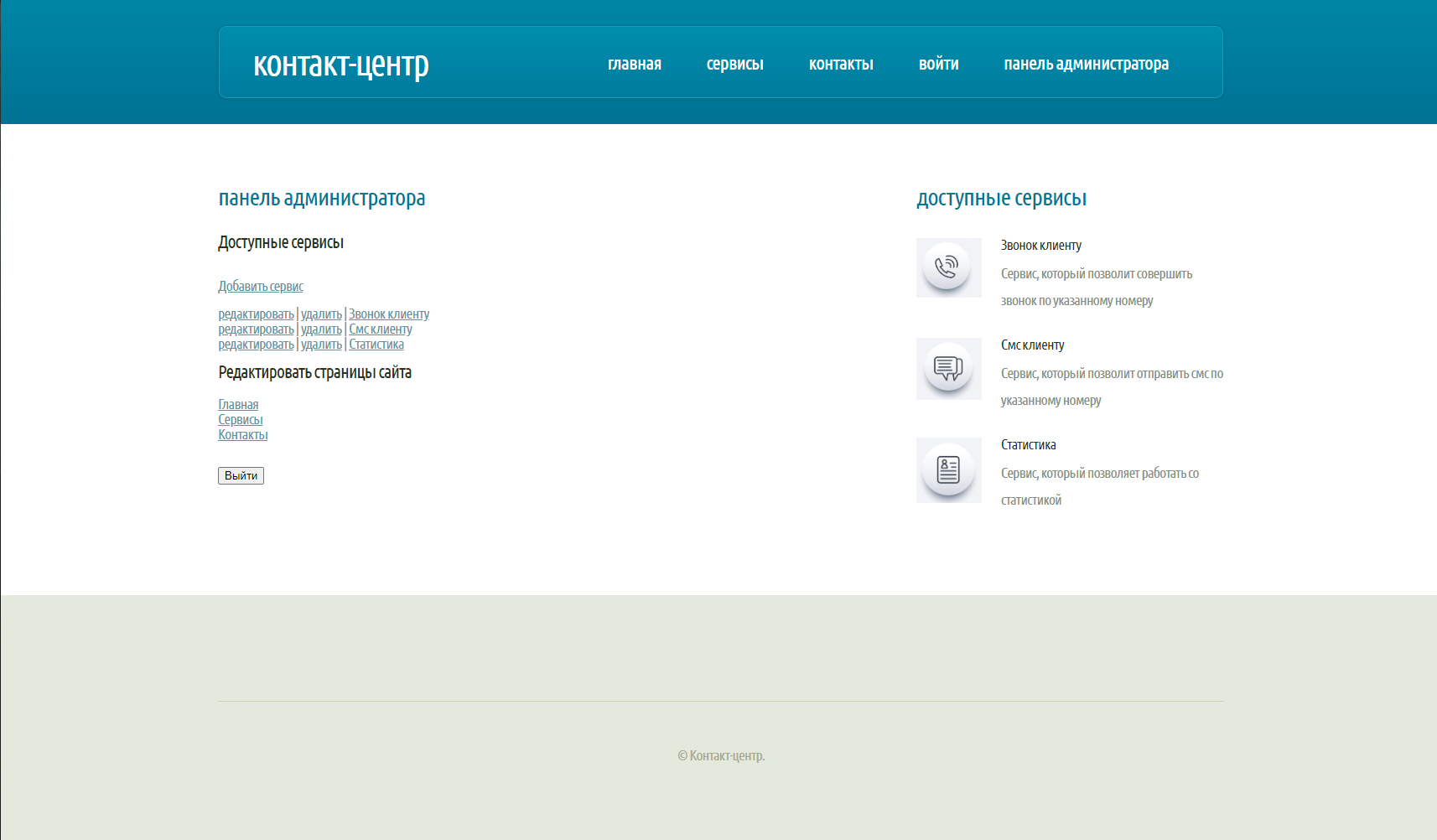
****

Рисунок 3.8 – Панель доступных сервисов на панели администратора

Также данное меню можно увидеть, но уже в более крупном виде при переходе на страницу сервисы (Рисунок 3.9).



Рисунок 3.9 – Страница сервисов

Каждый элементы в списке сервисов имеет свойство нажатия, благодаря чему можно легко перейти на страницу каждого сервиса.

Если нажать на сервис «Смс клиенту», то откроется страница, заполнения данных, на которой будет два текстбокса, первый для ввода номера клиента, а второй для ввода текста сообщения, также на форме присутствует кнопка «Отправить», которая инициирует отправку сообщения по указанному номеру (Рисунок 3.10).



Рисунок 3.10 – Страница отправки сообщения клиенту

При нажатии на кнопку отправить происходит отправка сообщения, страница обнуляется, и под названием сервиса появляется статус отправленного сообщения (Рисунок 3.11).



Рисунок 3.11 – Статус сообщения на странице

При нажатии на странице сервисов на сервис совершения звонка откроется небольшая панель, на которой будет расположено всего четыре элемента, а именно поля для ввода номера телефона, поле для ввода имени клиента, которому совершается звонок, а также кнопки «Позвонить» и «Сбросить»   
(Рисунок 3.12).



Рисунок 3.12 – Страница сервиса совершения звонка

При нажатии на сервис работы со статистикой пользователь попадает на страницу следующую страницу (Рисунок 3.13)



Рисунок 3.13 – Страница работы со статистикой

На данной странице у пользователя есть выбор, какую статистку ему загрузить, выбор производится с помощью трех чекбоксов, также необходимо возле каждого выбранного чекбокса прописать значения, по которым пользователь хочет получить статистику, а после нажать на кнопку «Получить». При нажатии на кнопку происходит скачивания файла в формате excel (Рисунок 3.14).



Рисунок 3.14 – Скачивание файла со статистикой

Таким образом, рассмотрено созданное приложение для организации работы контакт-центра: описано каждое поле ввода и вывода данных, показаны скриншоты всех компонентов интерфейса.

Полный листинг кода приложения можно увидеть в приложении А.

## 3.2 Руководство пользователя к микросервисному приложению для организации работы контакт-центра

Микросервисное приложение для организации работы контакт-центра обладает простым функционалом. В этом пункте будет описано руководство пользователя для этого приложения.

Для перехода в приложения необходимо получить URL от системного администратора этого приложения, также пользователь не может самостоятельно зарегистрироваться, данные своего аккаунта он может получить только у администратора. Перейдя по полученному URL, операционист попадает на окно авторизации, где должен ввести свои данные (Рисунок 3.15).



Рисунок 3.15 – Окно авторизации

После успешной авторизации операционист перенаправит на главную страницу. На главной странице пользователь может прочесть основную информацию о приложении, которую составил системный администратор. Для выполнения своей работы операционист должен либо перейти на страницу «Сервисы» через главное меню, расположенное в верхней части сайта (Рисунок 3.16), либо используя боковое меню перейти на необходимый ему сервис (Рисунок 3.17).



Рисунок 3.16 – Основная страница сайта



Рисунок 3.17 – Страница сервисы

У пользователя есть три основные возможности, а именно:

* Совершить звонок клиенту;
* Отправить клиенту сообщение;
* Получить статистику.

Каждое действие вызывается по нажатию на него на странице «Сервисы».

Если нажать на сервис «Смс клиенту», то откроется страница для заполнения данных, на которой будет два поля для ввода, первое для номера клиента, а второе для текста сообщения, также на форме присутствует кнопка «Отправить», которая инициирует отправку сообщения по указанному номеру (Рисунок 3.18).



Рисунок 3.18 – Страница отправки сообщения клиенту

При нажатии на кнопку отправить происходит отправка сообщения, страница обнуляется, и под названием сервиса появляется статус отправленного сообщения (Рисунок 3.18).

При нажатии на странице сервисов на сервис совершения звонка откроется небольшая панель, на которой будут расположены следующие элементы (Рисунок 3.19):

* поле для ввода номера телефона;
* поле для ввода имени клиента;
* кнопка «Позвонить»;
* кнопка «Сбросить».



Рисунок 3.19 – Страница сервиса совершения звонка

При нажатии на сервис работы со статистикой пользователь попадает на страницу, которая дает возможность получить всю необходимую статистику либо за конкретную дату, либо за конкретный период, либо за все время (Рисунок 3.20)

Рисунок 3.20 – Страница работы со статистикой

Пользователь может выбрать, какую статистку ему загрузить, выбор производится с помощью трех чекбоксов, также необходимо возле каждого выбранного чекбокса прописать значения, по которым пользователь хочет получить статистику, а после нажать на кнопку «Получить». При нажатии на кнопку происходит скачивания файла в формате excel (Рисунок 3.20).

Таким образом, приведено руководство пользователя микросервисного приложения для организации работы контакт-центра. Были приведены подробные описания для каждой функции приложения со скриншотами.

## 3.3 Выводы по третьему разделу

Разработано микросервисное приложение для организации работы контакт-центра. Приложение использует платформу ASP.NET Core MVC, что позволяет объединить и фронтенд часть(интерфейс) и бэкенд часть (серверная часть) в одном решении. Проверена работа программного средства, описан интерфейс, составлено руководство пользователя со скриншотами программы.

# Заключение

В ходе выполнения выпускной квалификационной работы было разработано микросервисное приложение для организации работы контакт-центра.

Целевая аудитория приложения – это компания, которая должна иметь связь со своими клиентами.

В процессе написания выпускной квалификационной работы были выполнены следующие задачи:

* анализ предметной области. Рассмотрены преимущества и особенности применения понятия «Контакт-центр». Проведено обоснование проектных решений по видам обеспечения, в результате которых для реализации проекта были использованы средства разработки и моделирования;
* спроектировано микросервисное приложения для организации работы контакт-центра на основе диаграмм прецедентов, классов, последовательностей. Рассчитаны функционально-ориентированные метрики и LOC-метрики с анализом рисков разрабатываемого мобильного веб-приложения;
* реализовано микросервисное приложения для организации работы контакт-центра. Применены теоретические и практические методы системного проектирования сложных программных средств и построения моделей для создания приложения.

Подводя итоги написания выпускной квалификационной работы, можно сказать, что все поставленные задачи решены, следовательно, главная цель, поставленная в выпускной квалификационной работе, также достигнута.

# Список используемых источников

1. Developments in the Call Centre Industry // Call Centre URL: https://library.oapen.org/bitstream/id/713b0179-54f4-4967-8799-d8c1af3915bb/1005859.pdf (Дата обращения 12.02.2022).
2. Call Centre Industry // Call Centre Industry URL: https://www.routledge.com/Developments-in-the-Call-Centre-Industry-Analysis-Changes-and-Challenges/Connell-Burgess/p/book/9780415511483 (Дата обращения: 12.02.2022).
3. **Рихтер Д**. CLR via C#. Программирование на платформе Microsoft .NET Framework 4.5 на языке C#. 4-е изд. – Питер, 2017. – 890c.
4. **Троелсен Э.** Язык программирования C# 7 и платформы .NET и .NET Core. 8-e изд. – Питер 1123с.
5. Услуги облачного контакт-центра // ContactCenter URL: https://www.naumen.ru/products/phone/blog/oblachnyy-call-tsentr-preimushchestva-i-vygoda/ (Дата обращения: 16.02.2022).
6. Функции облачного контакт-центра // ContactCenter Function URL: https://www.voiptime.net/ru/hosted\_contact\_center\_services\_ru ().
7. Возможности контакт-центра // CC Func URL: https://cloudcontact.ru/services/review/ (Дата обращения 19.02.2022).
8. Как выбрать кц? // Contact URL: https://www.voiptime.net/ru/hosted\_contact\_center\_services\_ru (Дата обращения 25.02.2022).
9. Основные обязанности кц // center URL: https://www.voiptime.net/ru/call-center-operator-responsibilities-ru.html (Дата обращения 01.03.2022).
10. КЦ что это такое // сс URL: https://www.okcall.ru/stati/185/ (Дата обращения 14.03.2022).
11. Достоинства контакт-центров // contact-center.ru URL: https://www.contact-center.ru/o-kompanii/ (Дата обращения 15.04.2022).
12. Микросервисы // Архитектура микросервисов URL: https://habr.com/ru/post/249183/ (Дата обращения 17.04.2022).
13. Микросервисная архитектура // Сервисы URL: https://mcs.mail.ru/blog/prostym-jazykom-o-mikroservisnoj-arhitekture (Дата обращения 20.04.2022).
14. Microservice // arh URL: https://habr.com/ru/company/raiffeisenbank/blog/346380/ (Дата обращения 27.04.2022).

# Приложение А

(обязательное)

Листинг кода

using ContactCenter.Domain;

using ContactCenter.Domain.Repositories.Abstract;

using ContactCenter.Domain.Repositories.EntityFramework;

using ContactCenter.Service;

using Microsoft.AspNetCore.Identity;

using Microsoft.EntityFrameworkCore;

var builder = WebApplication.CreateBuilder(args);

builder.Configuration.AddJsonFile("appsettings.json");

builder.Services.AddTransient<ITextFieldsRepository, TextFieldsRepository>();

builder.Services.AddTransient<IServiceItemsRepository, ServiceItemsRepository>();

builder.Services.AddTransient<ISmsService, SmsService>();

builder.Services.AddTransient<ICallService, CallService>();

builder.Services.AddTransient<IStatisticsService, StatisticsService>();

builder.Services.AddTransient<DataManager>();

var connection = builder.Configuration.GetConnectionString("DefaultConnection");

builder.Services.AddDbContext<AppDbContext>(x => x.UseSqlServer(connection ?? throw new InvalidOperationException()));

builder.Services.AddIdentity<IdentityUser, IdentityRole>(options =>

{

options.User.RequireUniqueEmail = true;

options.Password.RequiredLength = 3;

options.Password.RequireNonAlphanumeric = false;

options.Password.RequireLowercase = false;

options.Password.RequireUppercase = false;

options.Password.RequireDigit = false;

}).AddEntityFrameworkStores<AppDbContext>().AddDefaultTokenProviders();

builder.Services.ConfigureApplicationCookie(options =>

{

options.Cookie.Name = "contactCenterAuth";

options.Cookie.HttpOnly = true;

options.LoginPath = "/account/login";

options.AccessDeniedPath = "/account/accessdenied";

options.SlidingExpiration = true;

});

builder.Services.AddAuthorization(x =>

{

x.AddPolicy("AdminArea", policy => { policy.RequireRole("admin"); });

});

builder.Services.AddControllersWithViews(x =>

{

x.Conventions.Add(new AdminAreaAuthorization("Admin", "AdminArea"));

}).AddSessionStateTempDataProvider();

var app = builder.Build();

if (app.Environment.IsDevelopment())

{

app.UseDeveloperExceptionPage();

}

app.UseStaticFiles();

app.UseRouting();

app.UseCookiePolicy();

app.UseAuthentication();

app.UseAuthorization();

app.UseEndpoints(endpoints =>

{

endpoints.MapControllerRoute("admin", "{area:exists}/{controller=Home}/{action=Index}/{id?}");

endpoints.MapControllerRoute("default", "{controller=Home}/{action=Index}/{id?}");

});

app.Run();

using ContactCenter.Domain;

using Microsoft.AspNetCore.Mvc;

namespace ContactCenter.Areas.Admin.Controllers;

[Area("Admin")]

public class HomeController : Controller

{

private readonly DataManager \_dataManager;

public HomeController(DataManager dataManager)

{

\_dataManager = dataManager;

}

public IActionResult Index()

{

return View(\_dataManager.ServiceItemsRepository.GetServiceItems());

}

}

using ContactCenter.Domain;

using ContactCenter.Domain.Entities;

using ContactCenter.Service;

using Microsoft.AspNetCore.Mvc;

namespace ContactCenter.Areas.Admin.Controllers

{

[Area("Admin")]

public class ServiceItemsController : Controller

{

private readonly DataManager \_dataManager;

private readonly IWebHostEnvironment \_hostingEnvironment;

public ServiceItemsController(DataManager dataManager, IWebHostEnvironment hostingEnvironment)

{

\_dataManager = dataManager;

\_hostingEnvironment = hostingEnvironment;

}

public IActionResult Edit(Guid id)

{

var entity = id == default ? new ServiceItem() : \_dataManager.ServiceItemsRepository.GetServiceItem(id);

return View(entity);

}

[HttpPost]

public IActionResult Edit(ServiceItem model, IFormFile? titleImageFile)

{

if (ModelState.IsValid)

{

if (titleImageFile != null)

{

model.TitleImagePath = titleImageFile.FileName;

using var stream = new FileStream(Path.Combine(\_hostingEnvironment.WebRootPath, "images/", titleImageFile.FileName), FileMode.Create);

titleImageFile.CopyTo(stream);

}

\_dataManager.ServiceItemsRepository.SaveServiceItem(model);

return RedirectToAction(nameof(HomeController.Index), nameof(HomeController).CutController());

}

return View(model);

}

[HttpPost]

public IActionResult Delete(Guid id)

{

\_dataManager.ServiceItemsRepository.DeleteServiceItem(id);

return RedirectToAction(nameof(HomeController.Index), nameof(HomeController).CutController());

}

}

}

using ContactCenter.Domain;

using ContactCenter.Domain.Entities;

using ContactCenter.Service;

using Microsoft.AspNetCore.Mvc;

namespace ContactCenter.Areas.Admin.Controllers;

[Area("Admin")]

public class TextFieldsController : Controller

{

private readonly DataManager \_dataManager;

public TextFieldsController(DataManager dataManager)

{

\_dataManager = dataManager;

}

public IActionResult Edit(string codeWord)

{

var entity = \_dataManager.TextFieldsRepository.GetTextField(codeWord);

return View(entity);

}

[HttpPost]

public IActionResult Edit(TextField model)

{

if (ModelState.IsValid)

{

\_dataManager.TextFieldsRepository.SaveTextField(model);

return RedirectToAction(nameof(HomeController.Index), nameof(HomeController).CutController());

}

return View(model);

}

}

@model IQueryable<ServiceItem>

@{

const string strTitle = "Панель администратора";

ViewBag.Title = strTitle;

}

<div>

<h2>@strTitle</h2>

<div>

<h3>Доступные сервисы</h3>

<div class="div-box">

<a asp-area="Admin" asp-controller="ServiceItems" asp-action="Edit" asp-route-id="">Добавить сервис</a>

</div>

@if (Model.Any())

{

<div>

@foreach (var entity in Model)

{

<div>

<a asp-area="Admin" asp-controller="ServiceItems" asp-action="Edit" asp-route-id="@entity.Id">редактировать</a>

|

<form style="display: inline-block;" id="form-@entity.Id" asp-area="Admin" asp-controller="ServiceItems" asp-action="Delete" method="post">

<input type="hidden" name="id" value="@entity.Id">

<a href="#" onclick="document.getElementById('form-@entity.Id').submit();">удалить</a>

</form>

|

<a asp-area="" asp-controller="Services" asp-action="Index" asp-route-id="@entity.Id">

@($"{entity.Title}")

</a>

</div>

}

</div>

}

</div>

<div class="div-box">

<h3>Редактировать страницы сайта</h3>

<a asp-area="Admin" asp-controller="TextFields" asp-action="Edit" asp-route-codeWord="PageIndex">Главная</a>

<a asp-area="Admin" asp-controller="TextFields" asp-action="Edit" asp-route-codeWord="PageServices">Сервисы</a>

<a asp-area="Admin" asp-controller="TextFields" asp-action="Edit" asp-route-codeWord="PageContacts">Контакты</a>

</div>

<div class="div-box">

<form asp-area="" asp-controller="Account" asp-action="Logout" method="post">

<input type="submit" value="Выйти" />

</form>

</div>

</div>

@model ServiceItem

@{

const string strTitle = "Редактировать запись";

ViewBag.Title = strTitle;

}

<script src="js/ckeditor/ckeditor.js"></script>

<div>

<h2>@strTitle</h2>

<div>

<form asp-area="Admin" asp-controller="ServiceItems" asp-action="Edit" method="post" enctype="multipart/form-data">

<input type="hidden" asp-for="Id" />

<input type="hidden" asp-for="CreatedOn" />

<input type="hidden" asp-for="TitleImagePath" />

<div asp-validation-summary="All"></div>

<div class="div-box">

<label asp-for="Title"></label>

<input asp-for="Title" />

<span asp-validation-for="Title"></span>

</div>

<div class="div-box">

<label asp-for="Subtitle"></label>

<input asp-for="Subtitle" />

<span asp-validation-for="Subtitle"></span>

</div>

<div class="div-box">

<label asp-for="Text"></label>

<textarea asp-for="Text"></textarea>

<span asp-validation-for="Text"></span>

</div>

<div class="div-box">

<label asp-for="TitleImagePath"></label>

<input type="file" name="titleImageFile" id="titleImageFile" />

<div>

<img class="img-block" src="~/images/@Model.TitleImagePath" />

</div>

</div>

<div class="div-box">

<label asp-for="MetaTitle"></label>

<input asp-for="MetaTitle" />

<span asp-validation-for="MetaTitle"></span>

</div>

<div class="div-box">

<label asp-for="MetaDescription"></label>

<input asp-for="MetaDescription" />

<span asp-validation-for="MetaDescription"></span>

</div>

<div class="div-box">

<label asp-for="MetaKeywords"></label>

<input asp-for="MetaKeywords" />

<span asp-validation-for="MetaKeywords"></span>

</div>

<input type="submit" value="Сохранить" />

</form>

</div>

</div>

<script>

window.onload = function() {

var newCKEdit = CKEDITOR.replace('@Html.IdFor(x=>x.Text)');

newCKEdit.updateElement();

}

</script>

@model TextField

@{

const string strTitle = "Редактировать запись";

ViewBag.Title = strTitle;

}

<script src="js/ckeditor/ckeditor.js"></script>

<div>

<h2>@strTitle</h2>

<div>

<form asp-area="Admin" asp-controller="TextFields" asp-action="Edit" method="post" enctype="multipart/form-data">

<input type="hidden" asp-for="Id" />

<input type="hidden" asp-for="CreatedOn" />

<input type="hidden" asp-for="CodeWord" />

<div asp-validation-summary="All"></div>

<div class="div-box">

<label asp-for="Title"></label>

<input asp-for="Title" />

<span asp-validation-for="Title"></span>

</div>

<div class="div-box">

<label asp-for="Text"></label>

<textarea asp-for="Text"></textarea>

<span asp-validation-for="Text"></span>

</div>

<div class="div-box">

<label asp-for="MetaTitle"></label>

<input asp-for="MetaTitle" />

<span asp-validation-for="MetaTitle"></span>

</div>

<div class="div-box">

<label asp-for="MetaDescription"></label>

<input asp-for="MetaDescription" />

<span asp-validation-for="MetaDescription"></span>

</div>

<div class="div-box">

<label asp-for="MetaKeywords"></label>

<input asp-for="MetaKeywords" />

<span asp-validation-for="MetaKeywords"></span>

</div>

<input type="submit" value="Сохранить" />

</form>

</div>

</div>

<script>

window.onload = function() {

var newCKEdit = CKEDITOR.replace('@Html.IdFor(x=>x.Text)');

newCKEdit.updateElement();

}

</script>

using ContactCenter.Models;

using Microsoft.AspNetCore.Authorization;

using Microsoft.AspNetCore.Identity;

using Microsoft.AspNetCore.Mvc;

namespace ContactCenter.Controllers;

[Authorize]

public class AccountController : Controller

{

private readonly UserManager<IdentityUser> \_userManager;

private readonly SignInManager<IdentityUser> \_signInManager;

public AccountController(UserManager<IdentityUser> userManager, SignInManager<IdentityUser> signInManager)

{

\_userManager = userManager;

\_signInManager = signInManager;

}

[AllowAnonymous]

public IActionResult Login(string returnUrl)

{

ViewBag.returnUrl = returnUrl;

return View(new LoginViewModel());

}

[HttpPost]

[AllowAnonymous]

public async Task<IActionResult> Login(LoginViewModel model, string returnUrl)

{

if (ModelState.IsValid)

{

var user = await \_userManager.FindByNameAsync(model.UserName);

if (user != null)

{

await \_signInManager.SignOutAsync();

var result = await \_signInManager.PasswordSignInAsync(user, model.Password, model.RememberMe, false);

if (result.Succeeded)

{

return Redirect(returnUrl ?? "/");

}

}

ModelState.AddModelError(nameof(LoginViewModel.UserName), "Неверный логин или пароль");

}

return View(model);

}

[Authorize]

public async Task<IActionResult> Logout()

{

await \_signInManager.SignOutAsync();

return RedirectToAction("Index", "Home");

}

}

using ContactCenter.Domain;

using Microsoft.AspNetCore.Mvc;

namespace ContactCenter.Controllers;

public class HomeController : Controller

{

private readonly DataManager \_dataManager;

public HomeController(DataManager dataManager)

{

\_dataManager = dataManager;

}

public IActionResult Index()

{

return View(\_dataManager.TextFieldsRepository.GetTextField("PageIndex"));

}

public IActionResult Contacts()

{

return View(\_dataManager.TextFieldsRepository.GetTextField("PageContacts"));

}

}

using ContactCenter.Domain;

using ContactCenter.Service;

using Microsoft.AspNetCore.Mvc;

namespace ContactCenter.Controllers;

public class ServicesController : Controller

{

private readonly DataManager \_dataManager;

private readonly ISmsService \_smsService;

private readonly ICallService \_callService;

public ServicesController(DataManager dataManager, ISmsService smsService, ICallService callService)

{

\_dataManager = dataManager;

\_smsService = smsService;

\_callService = callService;

}

public IActionResult Index(Guid id)

{

if (id != default)

{

return View("Show", \_dataManager.ServiceItemsRepository.GetServiceItem(id));

}

ViewBag.TextField = \_dataManager.TextFieldsRepository.GetTextField("PageServices") ?? throw new InvalidOperationException();

return View(\_dataManager.ServiceItemsRepository.GetServiceItems());

}

[HttpGet]

public IActionResult Call()

{

return View();

}

[HttpPost]

public IActionResult Call(string phoneNumber)

{

\_callService.Call(phoneNumber);

return View();

}

[HttpGet]

public IActionResult Send()

{

return View();

}

[HttpPost]

public IActionResult Send(string phoneNumber, string textMessage)

{

var result = \_smsService.Send(textMessage);

ViewBag.Message = result

? $"Сообщение по номеру: {phoneNumber} успешно отправлено!"

: $"Сообщение по номеру: {phoneNumber} не удалось отправить!";

return View();

}

[HttpGet]

public IActionResult Statistics()

{

return View();

}

}

using System.ComponentModel.DataAnnotations;

namespace ContactCenter.Domain.Entities;

public abstract class EntityBase

{

protected EntityBase() => CreatedOn = DateTime.UtcNow;

[Required]

public Guid Id { get; set; }

[Display(Name = "Название (заголовок)")]

public virtual string? Title { get; set; }

[Display(Name = "Краткое описание")]

public virtual string? Subtitle { get; set; }

[Display(Name = "Полное описание")]

public virtual string? Text { get; set; }

/// <summary>

/// Название кнопки

/// </summary>

[Display(Name = "Метатег навзание сервиса")]

public string MetaTitle { get; set; } = "";

/// <summary>

/// Название сервиса

/// </summary>

[Display(Name = "Метатег описание")]

public string MetaDescription { get; set; } = "";

[Display(Name = "Метатег ключевое слово")]

public string MetaKeywords { get; set; } = "";

[Display(Name = "Титульная картинка")]

public virtual string? TitleImagePath { get; set; }

[DataType(DataType.Time)]

public DateTime CreatedOn { get; set; }

}

using System.ComponentModel.DataAnnotations;

namespace ContactCenter.Domain.Entities;

public class ServiceItem : EntityBase

{

[Required(ErrorMessage = "Заполните название сервиса")]

[Display(Name = "Название сервиса")]

public override string? Title { get; set; }

[Display(Name = "Краткое название сервиса")]

public override string? Subtitle { get; set; }

[Display(Name = "Полное описание сервиса")]

public override string? Text { get; set; }

}

using System.ComponentModel.DataAnnotations;

namespace ContactCenter.Domain.Entities;

public class TextField : EntityBase

{

[Required]

public string CodeWord { get; set; }

[Display(Name = "Название страницы (заголовок)")]

public override string? Title { get; set; } = "Главная страница";

[Display(Name = "Содержание страницы")]

public override string? Text { get; set; } = "Содержание заполняется администратором";

}

using ContactCenter.Domain.Entities;

using Microsoft.AspNetCore.Identity;

using Microsoft.AspNetCore.Identity.EntityFrameworkCore;

using Microsoft.EntityFrameworkCore;

namespace ContactCenter.Domain;

public class AppDbContext : IdentityDbContext<IdentityUser>

{

private readonly string \_roleId;

private readonly string \_userId;

public AppDbContext(DbContextOptions<AppDbContext> options) : base(options)

{

\_roleId = Guid.NewGuid().ToString();

\_userId = Guid.NewGuid().ToString();

}

public DbSet<TextField>? TextFields { get; set; }

public DbSet<ServiceItem>? ServiceItems { get; set; }

protected override void OnModelCreating(ModelBuilder modelBuilder)

{

base.OnModelCreating(modelBuilder);

modelBuilder.Entity<IdentityRole>().HasData(new IdentityRole

{

Id = \_roleId,

Name = "admin",

NormalizedName = "ADMIN"

});

modelBuilder.Entity<IdentityUser>().HasData(new IdentityUser

{

Id = \_userId,

UserName = "admin",

NormalizedUserName = "ADMIN",

Email = "admin@email.com",

NormalizedEmail = "ADMIN@email.com",

EmailConfirmed = true,

PasswordHash = new PasswordHasher<IdentityUser>().HashPassword(null, "admin"),

SecurityStamp = string.Empty

});

modelBuilder.Entity<IdentityUserRole<string>>().HasData(new IdentityUserRole<string>

{

RoleId = \_roleId,

UserId = \_userId

});

modelBuilder.Entity<TextField>().HasData(new TextField

{

Id = Guid.NewGuid(),

CodeWord = "PageIndex",

Title = "Главная"

});

modelBuilder.Entity<TextField>().HasData(new TextField

{

Id = Guid.NewGuid(),

CodeWord = "PageServices",

Title = "Сервисы"

});

modelBuilder.Entity<TextField>().HasData(new TextField

{

Id = Guid.NewGuid(),

CodeWord = "PageContacts",

Title = "Контакты"

});

}

}

using ContactCenter.Domain.Repositories.Abstract;

namespace ContactCenter.Domain;

public class DataManager

{

public readonly ITextFieldsRepository TextFieldsRepository;

public readonly IServiceItemsRepository ServiceItemsRepository;

public DataManager(ITextFieldsRepository textFieldsRepository, IServiceItemsRepository serviceItemsRepository)

{

TextFieldsRepository = textFieldsRepository;

ServiceItemsRepository = serviceItemsRepository;

}

}

using System.ComponentModel.DataAnnotations;

namespace ContactCenter.Models;

public class LoginViewModel

{

[Required]

[Display(Name = "Логин")]

public string UserName { get; set; }

[Required]

// ReSharper disable once Mvc.TemplateNotResolved

[UIHint("password")]

[Display(Name = "Пароль")]

public string Password { get; set; }

[Display(Name = "Запомнить меня?")]

public bool RememberMe { get; set; }

}

using Microsoft.AspNetCore.Mvc;

using Microsoft.AspNetCore.Mvc.ApplicationModels;

using Microsoft.AspNetCore.Mvc.Authorization;

namespace ContactCenter.Service;

public class AdminAreaAuthorization : IControllerModelConvention

{

private readonly string \_area;

private readonly string \_policy;

public AdminAreaAuthorization(string area, string policy)

{

\_area = area;

\_policy = policy;

}

public void Apply(ControllerModel controller)

{

if (controller.Attributes.Any(a =>

a is AreaAttribute attribute && attribute.RouteValue.Equals(\_area, StringComparison.OrdinalIgnoreCase))

|| controller.RouteValues.Any(r =>

r.Key.Equals("area", StringComparison.OrdinalIgnoreCase) && r.Value.Equals(\_area, StringComparison.OrdinalIgnoreCase)))

{

controller.Filters.Add(new AuthorizeFilter(\_policy));

}

}

}

using ContactCenter.Domain.Entities;

using ContactCenter.Domain.Repositories.Abstract;

using Microsoft.EntityFrameworkCore;

namespace ContactCenter.Domain.Repositories.EntityFramework;

public class ServiceItemsRepository : IServiceItemsRepository

{

private readonly AppDbContext \_context;

public ServiceItemsRepository(AppDbContext context)

{

\_context = context;

}

public IQueryable<ServiceItem>? GetServiceItems()

{

return \_context.ServiceItems;

}

public ServiceItem? GetServiceItem(Guid id)

{

if (\_context.ServiceItems != null) return \_context.ServiceItems.FirstOrDefault(x => x.Id == id);

throw new Exception("Не удалось получить сервис по указанному идентификатору");

}

public void SaveServiceItem(ServiceItem entity)

{

\_context.Entry(entity).State = entity.Id == default

? EntityState.Added

: EntityState.Modified;

\_context.SaveChanges();

}

public void DeleteServiceItem(Guid id)

{

\_context.ServiceItems?.Remove(new ServiceItem

{

Id = id

});

\_context.SaveChanges();

}

}

using ContactCenter.Domain.Entities;

using ContactCenter.Domain.Repositories.Abstract;

using Microsoft.EntityFrameworkCore;

namespace ContactCenter.Domain.Repositories.EntityFramework;

public class TextFieldsRepository : ITextFieldsRepository

{

private readonly AppDbContext \_context;

public TextFieldsRepository(AppDbContext context)

{

\_context = context;

}

public IQueryable<TextField>? GetTextFields()

{

return \_context.TextFields;

}

public TextField? GetTextField(Guid id)

{

if (\_context.TextFields != null) return \_context.TextFields.FirstOrDefault(x => x.Id == id);

throw new Exception("Не удалось получить текстовое поле по указанному идентификатору");

}

public TextField? GetTextField(string codeWord)

{

if (\_context.TextFields != null) return \_context.TextFields.FirstOrDefault(x => x.CodeWord == codeWord);

throw new Exception("Не удалось получить текстовое поле по указанному ключевому слову");

}

public void SaveTextField(TextField entity)

{

\_context.Entry(entity).State = entity.Id == default

? EntityState.Added

: EntityState.Modified;

\_context.SaveChanges();

}

public void DeleteTextField(Guid id)

{

\_context.TextFields?.Remove(new TextField

{

Id = id

});

\_context.SaveChanges();

}

}

using ContactCenter.Domain.Entities;

namespace ContactCenter.Domain.Repositories.Abstract;

public interface IServiceItemsRepository

{

/// <summary>

/// Получить все доступные сервисы.

/// </summary>

/// <returns></returns>

IQueryable<ServiceItem>? GetServiceItems();

/// <summary>

/// Получить сервис по идентификатору.

/// </summary>

/// <param name="id"></param>

/// <returns></returns>

ServiceItem? GetServiceItem(Guid id);

/// <summary>

/// Сохранить сервис или сохранить изменения.

/// </summary>

/// <param name="entity"></param>

void SaveServiceItem(ServiceItem entity);

/// <summary>

/// Удалить сервис.

/// </summary>

/// <param name="id"></param>

void DeleteServiceItem(Guid id);

}

using ContactCenter.Domain.Entities;

namespace ContactCenter.Domain.Repositories.Abstract;

public interface ITextFieldsRepository

{

/// <summary>

/// Получить все текстовые поля.

/// </summary>

/// <returns></returns>

IQueryable<TextField>? GetTextFields();

/// <summary>

/// Выбрать поле по идентификатору.

/// </summary>

/// <param name="id"></param>

/// <returns></returns>

TextField? GetTextField(Guid id);

/// <summary>

/// Выбрать поле по ключевому слову.

/// </summary>

/// <param name="codeWord"></param>

/// <returns></returns>

TextField? GetTextField(string codeWord);

/// <summary>

/// Сохранить текстовое поле или сохранить изменения.

/// </summary>

/// <param name="entity"></param>

void SaveTextField(TextField entity);

/// <summary>

/// Удалить текстовое поле.

/// </summary>

/// <param name="id"></param>

void DeleteTextField(Guid id);

}