ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«БЕЛГОРОДСКИЙ ГОСУДАРСТВЕННЫЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ»
(Н И У « Б е л Г У »)

ИНСТИТУТ ИНЖЕНЕРНЫХ ТЕХНОЛОГИЙ И ЕСТЕСТВЕННЫХ НАУК

КАФЕДРА МАТЕМАТИЧЕСКОГО И ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ ИНФОРМАЦИОННЫХ СИСТЕМ

АВТОМАТИЗИРОВАННАЯ СИСТЕМА ПРИЕМА ЗАЯВОК ДЛЯ ООО «ПАРУСНИК»

Выпускная квалификационная работа
обучающегося по направлению подготовки
02.03.02. Фундаментальная информатика и информационные технологии
очной формы обучения, группы 07001301
Дебелого Антона Альбертовича

Научный руководитель
к.т.н., доцент
Чашин Ю. Г.

БЕЛГОРОД 2017

ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ

Успешная работа организации напрямую связанна с реализацией и обслуживанием лицензий. В зависимости от того, как реализована работа сотрудников, взаимодействия их с клиентами, и от автоматизации всех исполнительных процессов производится оценка работы организации в общем. Увеличение количества автоматизированных процессов между сотрудниками и клиентами – залог успеха.

Актуальность темы заключается в том, что рано или поздно любая услуга превращается в приложение, так как это облегчает исполнение желаемых процедур с обеих сторон. Объектом исследования была выбрана компания ООО «Парусник» в г. Белгород. Предметом исследования является необходимость организации в мобильном приложении автоматизированного приёма заявок.

ООО «Парусник-Белгород» основана в 1995 году и в настоящее время является генеральным дилером Корпорации «ПАРУС» в Белгородской области на рынке государственных заказчиков.

Организация предоставляет полный список программных продуктов для полной, комплексной автоматизации управления деятельностью бюджетных учреждений, так же предоставляют услуги внедрения, обучения, методологическое и консалтинговое пособничество, усовершенствование и доработку программного обеспечения. Так же организация «Парусник-Белгород» фигурирует как серебряный партнер «Oracle» и компании «БАРС Груп».

В качестве главного программного продукта по автоматизации деятельности бюджетных учреждений организация эксплуатирует «Парус-Бюджет 8» Система основана на базе двухзвенной архитектуры клиент-сервер на основе СУБД Oracle с возможностью взаимодействия с web-сервисами.

Цели, выполняемые компанией, акцентированы на реализацию новых принципов государственного управления и увеличение эффективности использования финансов общества на всех уровнях бюджетной системы государства.

Цель выпускной квалификационной работы заключается в повышении эффективности процессов обработки заявок за счет разработки и внедрения мобильного приложения автоматизированного приёма заявок, с использованием актуальных средств разработки.

Для достижения поставленной цели в выпускной квалификационной работе следует решить следующие задачи:

– изучить предметную область;
– ориентироваться с оптимальными средствами разработки;
– выбрать метод реализации создания программы;
– спроектировать базу данных будущего приложения;
– разработать функционал приложения;
– протестировать и внедрить разработанную информационную систему.

Выпускная квалификационная работа состоит из 3 частей, которые рассматривают этапы анализа, проектирования и разработки.

Первая часть – анализ предметной области и постановка задачи для разработки автоматизированного приёма заявок Компании ООО «Парусник». В данной главе рассмотрены технико-экономическая характеристика компании, ее структура и подразделения.

Вторая часть – Проектирование мобильного приложения. В данной главе рассмотрено техническое, информационное, программное и технологическое обеспечение. Анализируя каждый из видов обеспечения, были выбраны определенные решения для достижения поставленной цели выпускной квалификационной работы.

Третья часть – «Разработка мобильного приложения». Данная часть работы включает в себя рассмотрение информационной модели

разрабатываемой системы, разработку и тестирование мобильного приложения.

Выпускная квалификационная работа выполнена на 52 листах, включает в себя 3 части, введение, заключение, список используемых источников и записки от организации, что код программы является её собственностью и огласке не поддаётся.

ГЛАВА 1. АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ И ПОСТАНОВКА ЗАДАЧИ ДЛЯ РАЗРАБОТКИ АВТОМАТИЗИРОВАННОГО ПРИЁМА ЗАЯВОК КОМПАНИИ ООО «ПАРУСНИК»

1.1 Характеристика деятельности ООО «Парусник»

ООО «Парусник-Белгород» основана в 1995 году и в настоящее время является генеральным дилером Корпорации «ПАРУС» в Белгородской области на рынке государственных заказчиков.

Организация предоставляет полный список программных продуктов для полной, комплексной автоматизации управления деятельностью бюджетных учреждений, так же предоставляют услуги внедрения, обучения, методологическое и консалтинговое пособничество, усовершенствование и доработку программного обеспечения. Так же организация «Парусник-Белгород» фигурирует как серебряный партнер «Oracle» и компании «БАРС Груп».

В качестве главного программного продукта по автоматизации деятельности бюджетных учреждений организация эксплуатирует «Парус-Бюджет 8» Система основана на базе двухзвенной архитектуры клиент-сервер на основе СУБД Oracle с возможностью взаимодействия с web-сервисами.

Цели, выполняемые компанией, акцентированы на реализацию новых принципов государственного управления и увеличение эффективности использования финансов общества на всех уровнях бюджетной системы государства.

Функциональные и архитектурные превосходства решений «Парус» разрешают не только проблемы ключевых учётно-управленческих задач, но и

оптимизацию эксплуатационных затрат; осуществляют поддержку сервисов для включения государственной власти и муниципального управления в систему электронного внутриведомственного и межведомственного взаимодействия. Компетенции организации в области автоматизации административно-хозяйственной деятельности представляются в продуктивной модели «электронной экономики» отрасли, основанной на программно-целевых приёмах планирования и комбинирования деятельности и совместном управлении ресурсами.

В настоящее время организация занимается автоматизацией учёта:

- заработной платы;
- оплаты услуг в учреждениях;
- бухгалтерского учёта;
- продуктов питания;
- расчета тарификации в учреждениях здравоохранения;
- расчета подушевого финансирования в сфере образования;
- учет в системе «кадры и штатное расписание»;
- автотранспортных средств;
- учёт материально-технического снабжения;
- планирование и финансирование;
- консолидации отчётности и анализа финансово-хозяйственной деятельности и т.д.

Групповой подход к автоматизации разрешает реализовывать проекты, как постепенно, так и выполнять готовые решения, обеспечивая эффективное управления, планировку и анализ процессов предприятий и учреждений.

На данный момент главным направление ООО «Парусник-Белгород» является централизация информационных ресурсов, а преимущественно создание единой системы бухгалтерского учёта в органах власти, объединяющую все муниципальные органы и подведомственные учреждения.

Система находится на сервере органов власти, а пользователи в подведомственных учреждениях совершают работу с программным продуктом с помощью Web-доступа. Этот подход обеспечивает использование объединенной учётной политики, унифицированные настройки с разделением прав доступа, разрешая уменьшить количество ошибок в учёте и поднять достоверность и быстродействие данных для центрального аппарата.

Централизация бухгалтерского учета на платформе «Парус» – это основа для построения современной системы управленческого учета, обеспечивающей полноценную информационную поддержку финансово-экономической службы.

Таким образом, исследуемая компания ООО «Парусник-Белгород» является крупной компанией разработки, внедрения и сопровождения комплексных информационно-технологических и консалтинговых решений в сфере государственного и муниципального управления, а также оказания государственных и муниципальных услуг региональными и муниципальными органами власти.

Результатом многолетнего сотрудничества компании ООО «Парусник-Белгород» с органами федеральной и региональной государственной власти стало внедрение централизованных решений в сфере здравоохранения, образования и силовых структур, администраций районов:

- Администрация Губернатора Белгородской области.
- Департамент финансов и бюджетной политики, отделы и управления финансов и бюджетной политики, муниципальные районы области с городскими и сельскими поселениями и учреждениями.
- Управление внутренних дел Белгородской области с подразделениями.
- Управление вневедомственной охраны при УВД Белгородской области с подразделениями.

- Управление здравоохранения Белгородской области с подведомственными учреждениями.
- Управление социальной защиты населения администрации Белгородской области с подведомственными учреждениями.
- Управление культуры Белгородской области с подведомственными учреждениями.
- Белгородская таможня.
- Арбитражный суд Белгородской области.
- Управление Федеральной службы судебных приставов по Белгородской области.
- Управление Судебного департамента в Белгородской области.

За прошедшие годы работы были достигнуты следующие результаты:

- Число клиентов превысило 800
- Создано структурное подразделение в г. Старый Оскол
- Численность персонала превысила 60 человек.
- Выстроены партнерские отношения с АНО «Белинфоналог» и ООО «ИТС».

За прошедшие годы (11 лет) работы были достигнуты следующие результаты:

- Число клиентов превысило 800;
- Создано структурное подразделение в г. Старый Оскол;
- Численность персонала превысила 150 человек;
- Выстроены партнерские отношения с АНО «Белинфоналог» и ООО «ИТС».

На рис. 1.1 представлен анализ рынка программных продуктов автоматизирующие бюджетный учёт Белгородской области. Как видно, лидирующие позиции остаются за ПП «Парус».

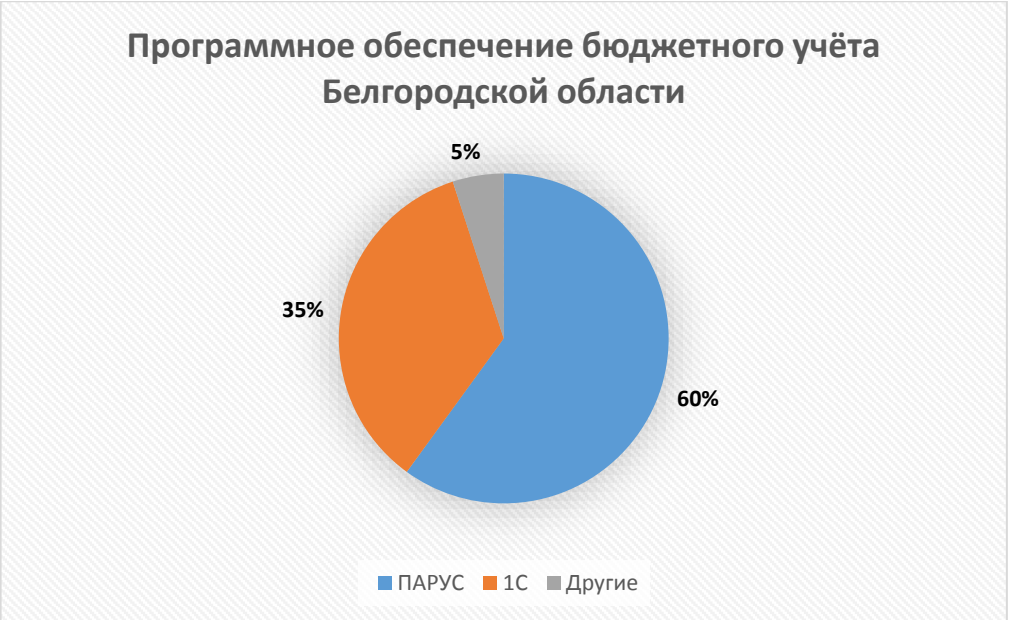

Рис. 1.1. Обзор рынка ПП бюджетной сферы Белгородской области.

Далее рассматривается организационная структура управления.

1.2. Организационная структура управления

Организационная структура компании ООО «Парусник-Белгород» представлена на рис. 1.2. Данная схема организационной структуры управления отражает статическое положение подразделений и должностей и характер связи между ними.

Тип данной структуры – линейно-функциональная структура – ступенчатая иерархическая. При ней линейные руководители являются единоначальниками, а им оказывают помощь функциональные органы. Линейные руководители низших ступеней административно не подчинены функциональным руководителям высших ступеней управления.

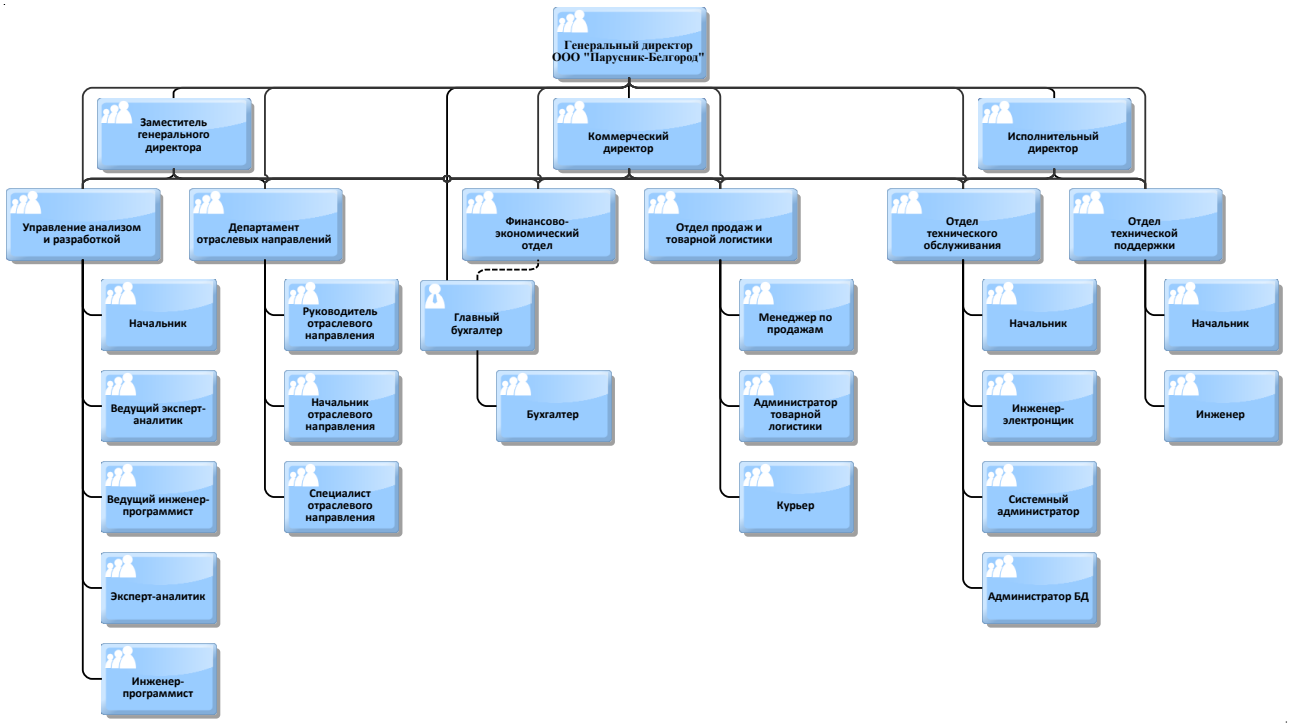

Рис. 1.2. Организационная структура ООО «Парусник-Белгород»

Для того что бы описать основные бизнес процессы компании были предприняты следующие методы сборы информации:

1) Изучение внутренней документации: регламентов, должностных инструкций;

2) Интервьюирование.

При анализе структуры компании была выявлена четкая взаимосвязь отдела и бизнес-процесса, закрепленного за ним, что говорит о продуманности вопроса организационной структуры.

Основной бизнес-процесс организации - это деятельность в области внедрения и сопровождения программных продуктов Корпорации «Парус».

Вспомогательные бизнес процессы организации рассматриваться не будут (процессы отдела бухгалтерии, процессы материально технического обеспечения и т.д.).

Для того, чтобы оценить взаимосвязь бизнес-процессов и отделов, перечислим бизнес-процессы каждого отдела.

Отдел продаж и товарной логистики (далее ОПиТЛ) – первичный процесс (дающий начало остальным процессам) так как предполагает продажу ПП «Парус-Бюджет».

Функциями ОПиТЛ являются:

1. Осуществление первичных и вторичных продаж всей линейки ПП «Парус»;
2. Сбор и анализ данных, необходимых для дальнейшего продвижения программных продуктов;
3. Отслеживание изменений ценовой политики, новых версий и релизов Корпорацией «Парус»;
4. Оформление договоров и счетов на первичные, вторичные покупки и ЛО программных продуктов «Парус»;
5. Отслеживание оплаты по договорам и возврата полного пакета документов (договоров, товарных накладных, актов);
6. Отгрузка ПП;
7. Передача прав на ПП, замена рабочих мест у клиентов;
8. Составление смет клиентам для планирования денежных средств;
9. Участие в торгах в качестве поставщика;
10. Обработка входящей корреспонденции;
11. Доставка корреспонденции всех отделов заказчику.

Деятельность отдела продаж и товарной логистики направлена на поиск потенциальных клиентов, продвижение новых решений «Парус» (проведение демонстраций и презентаций и консультирование по новым направлениям).

В процессе используется решение корпорации Парус «Управление деловыми процессами» (далее УДП), который позволяет хранить информацию о существующих и потенциальных клиентах, с контактными данными, с их лицензиями на ПП «Парус», договора и всю сопутствующую информацию. Данный модуль предназначен для автоматизации деловых

процессов предприятия, управления документооборотом, а также создания корпоративной информационной системы.

Результат работы с клиентом является договор на оказание услуг по установке, тестированию и сопровождению ПП «Парус», его адаптации, модификации и разработке дополнительного функционала и других работ и услуг по ГОСТу Р ИСО/МЭК 14764-2002.

Отдел технического обслуживания (далее ОТО) – специализирован на оказании помощи клиентам с настройкой локально-вычислительных сетей, настройкой сетевого оборудования, установкой и настройкой операционных систем и другого системного программного обеспечения, администрированием баз данных, настройкой и администрированием серверного оборудования.

Функции ОТО:

1. Регистрация сотрудников как пользователей сети (домена) Организации;
2. Регистрация сотрудников как пользователей терминала;
3. Обеспечение работоспособности/доступности следующих информационных ресурсов/программных продуктов (при необходимости их использования):
   - 3.1. ПП Парус-Реализация и склад (БД Организации);
   - 3.2. ПП ParusnikLite;
   - 3.3. ПП Учёт заявок на автотранспорт;
   - 3.4. СБИС агентский;
   - 3.5. Доступ к сети Интернет;
   - 3.6. Доступ к электронной почте (личной /отдела/ общей для Организации);
   - 3.7. Доступ к печати необходимых документов;
   - 3.8. Доступ к дистрибутивам/установленным версиям релизов ПП Парус и прочих ПП;
   - 3.9. Доступ к общим/личным ресурсам для хранения информации;

3.10. Обеспечение антивирусной защиты сети Организации.

4. Своевременная заправка картриджей/ремонт оргтехники в сервис-центрах.

5. Проведение работ по увеличению надежности, ускорению работы, безопасности, конфиденциальности и удобства пользования вышеуказанным программным обеспечением.

6. Настройка и сопровождение следующих информационных ресурсов: сайт parusnik.org, FTP сервер ftp.parusnik.org, почтовый сервер mail.parusnik.org.

7. Программирование цифровой АТС Организации.

8. Регулярное резервное копирование БД Организации ПП Парус, Клиент-Банк, электронной почты и пр.

Отдел технической поддержки (далее ОТП) – основной задачей отдела является оперативное оказание помощи клиентам по телефону («Горячая линия») или при помощи средств удаленного доступа (AmmyyAdmin - лицензированного). В данном отделе реализуются процессы обратной связи с клиентами, их информирование о нововведениях.

Если на данном уровне специалисты не уполномочены решать вопросы клиента, они передают информацию о проблеме с помощью информационных систем управления проектами «Redmine» или «УДП» далее в другие отделы: департамент отраслевых направлений, управление анализом и разработкой или в отдел технического обслуживания.

Департамент отраслевых направлений (далее ДОН) – создан для обеспечения качественной и квалифицированной поддержки пользователей, а также для перевода организаций с имеющихся учётных систем на работу в ПП «Парус». Управление включает в себя отдельные направления:

- Здравоохранение;
- Образование;
- Силовые структуры;
- Муниципальное управление;

- Управление государственными закупками;
- Управление социальной защиты населения;
- Фонд социального страхования.

В каждом направлении есть руководитель и специалисты по внедрению, сопровождению ПП «Парус» по отдельным модулям: бухгалтерский учёт, заработная плата, учёт продуктов питания и т. д.

В данные отделы поступает информация о проблемах клиента через систему управлениями проектами «Redmine», «УДП» либо напрямую от клиента. В процессе используются инструкции о доработках, написанные Управлением анализа и разработки (далее УАиР), техническая документация к ПП «Парус», бюджетное законодательство. Доступ к законам, указам, ведомственным актам и др. предоставляется в программе «Консультант плюс».

Для решения некоторых вопросов необходима доработка ПП «Парус», тогда инициируется задача в системе «Redmine», которая запускает бизнес-процесс анализа проблемы УиИР. Задача должна быть описана с двух сторон: со стороны клиента и со стороны возможностей ПП «Парус». К задаче необходимо прикреплять текст ошибки, если таковой имеется, а также подробные скриншоты.

Для решения некоторых вопросов необходима техническая поддержка, например, ежемесячное обновление ПП «Парус». На данный бизнес процесс влияют компоненты «из вне». Корпорация «ПАРУС» на своем сайте и по e-mail оповещает дистрибьюторов о выходе обновлений ко всем ПП линейки: «Парус 7», «Парус 8» и «Парус 10». Эти данные дублируются в системе управлениями проектами «Redmine» – тем самым информируя всех сотрудников компании. Данные обновления скачиваются через FTP на сервер корпорации «Парус». Руководитель каждого направления принимает решение о необходимости установки нового релиза либо отдельных патчей на реальные базы клиентов. Исходя из решений руководителей (и не только) данные релизы и патчи предварительно тестируются на тестовых базах. При

успешном тестировании патч или релиз устанавливается отделом технического обслуживания либо отделом программистов при отделе УАИР на выбранные для обновления базы.

Управление анализом и разработкой (УАиР) – основными задачами являются: анализ и тестирование возможностей новых программных продуктов, комплексный анализ бизнес-процессов клиента, написание технических заданий и дополнительного функционала под потребности клиента, написание аналитической и технической документации.

Данный отдел получает и перерабатывает все вопросы от клиентов по поводу реализации конкретного функционала системы. При необходимости доработок регистрирует события для Корпорации «ПАРУС», которая устраняет ошибки либо дорабатывает функционал при условии того, что данные изменения должны входить в поставку в связи с изменениями законодательства. Если у клиента существует потребность, которая не предусмотрена как «обязательная» законодательством, то компания «Парусник-Белгород» может реализовать все необходимое своими силами.

УАиР состоит условно из двух подотделов: аналитического отдела и отдела разработки. Перед аналитиками ставятся следующие задачи:

- повышение эффективности работы программных средств, используемых либо реализуемых предприятием;
- улучшение и модернизация указанных средств, - в рамках задач, поставленных руководством УАиР;
- повышение качества и уровня знаний сотрудников смежных отделов с помощью проведения обучений, - в рамках своего направления аналитической деятельности;
- своевременное и точное обеспечение выполнения планов работ и задач, поставленных руководством УАиР;
- подготовка пользовательских инструкций в рамках своего аналитического направления;
- проведение занятий с сотрудниками других отделов;

- проведение демонстраций и вебинаров для клиентов и потенциальных заказчиков;
- проведение интервьюирования у потенциального заказчика, построение частной схемы бизнес-процесса по результатам интервьюирования;
- написание технических заданий в рамках своего аналитического направления, тестирование доработок/разработок, выполненных по написанным техническим заданиям;
- участие в тестировании новых программных продуктов или модулей, подготовка отчета о проведенном тестировании.
- Перед программистами ставятся следующие задачи:
- написание дополнительного функционала для программных продуктов, используемых ООО «Парусник-Белгород»;
- анализ и исправление, если возможно, своими силами ошибок и недочетов в программных продуктах, используемых ООО «Парусник-Белгород»;
- участие в разработке нового функционала, в рамках задач, поставленных начальником отдела;
- подготовка технических пособий в случае наличия поставленной задачи;
- проведение специализированных занятий с сотрудниками других подразделений;
- участие в тестировании новых программных продуктов или модулей, подготовка отчета о проведенном тестировании.

В отделе УАИР за каждым направлением (модулем) закреплен отдельный работник – эксперт-аналитик, что позволяет работнику полностью видеть общую картину, с вносимыми изменениями, доработкам и новыми потребностями, осуществлять сбор и анализ данных, необходимых для улучшения качества ПП.

1.3 Анализ уровня автоматизации

IT инфраструктура является технологической подложкой для работы других слоёв корпоративной архитектуры, состоит из:

1. Технологических сервисов:

1) Аппаратного обеспечения, сетевого оборудования:

Операционные системы: Windows 7 Профессиональная, ServicePack 1 – для персональных компьютеров и Windows Server 2008 R2 для серверов.

2) Базовые сетевые службы – DHCP, DNS – есть.

2. Сервисы сове местной работы:

1) Служба хранения файловых ресурсов – есть.

2) Служба сетевой печати – есть.

3) Система терминального доступа – есть.

4) Система управления базами данных – PL/SQL Developer.

3. Сервисы для обеспечения безопасности:

1) Система резервного копирования – есть.

2) Система безопасного доступа в Интернет – есть.

3) Система удаленного доступа к локальной сети – есть.

4) Система антивирусной защиты – KasperskyEndpointSecurity 10.

5) Система автоматического обновления ПО – есть.

6) Криптография – КриптоПро ЭП, ViPNet.

4. Службы обеспечивающие работоспособность коммуникаций:

1) Сервисы сообщений – отсутствует.

2) Система электронной почты – есть.

3) Система унифицированных коммуникаций: IP-телефония Asterisk; мгновенные сообщения ICQ.

1.4. Постановка цели и задач для реализации Мобильного приложения автоматизированного приёма заявок

В результате анализа деятельности ООО «Парусник-Белгород», стало понятно, что основное направление работы организации является продажа лицензий, их обслуживание и работа с клиентами, по присылаемым от них же заявкам. Исходя из предметной области предприятия была поставлена цель выпускной работы.

Целью прохождения преддипломной практики является автоматизация приёма заявок, приходящих от клиентов, пользуемых программным продуктом «Парус 8».

Для реализации поставленной цели необходимо решение следующих задач:

- подробное изучение предметной области, а именно деятельности ООО «Парусник-Белгород»;

- проведение анализа уровня автоматизации ООО «Парусник-Белгод»;

- проектирование и разработка базы данных для мобильного приложения;

- проектирование пользовательского интерфейса для разрабатываемого программного продукта;

- тестирование контрольного примера рабочего мобильного приложения.

Для решения поставленной цели и задач было выбрано следующее программное обеспечение и технологии реализации: Visual Studio 2017 CE, Oracle SQL Developer, Visual Studio Code, .NET Core, .NET Framework, Xamarin. Forms, Docker, Redis, Rabbit MQ, язык программирования C#.

ГЛАВА 2. ПРОЕКТИРОВАНИЕ МОБИЛЬНОГО ПРИЛОЖЕНИЯ

2.1 Проектирование баз данных

Концептуальное проектирование базы данных - это процесс создания информационной схемы организации в виде самостоятельных представлений сведений пользователей, которые являются независимыми от атрибутов конкретно используемой СУБД и прочих физических обстоятельств.

Отмечают несколько стадий семантического проектирования:

- Превращение топической инфологической модели данных в топическое логическое представление;
- Выбор настройки связей исходя из схемы топической логической структуры данных;
- Тестирование модели с помощью положения нормализации;
- Тестирование модели касательно логических операций пользователей;
- Проектирование блок-схемы сущность-связь;
- Выбор условий поддержки неделимости данных;
- Рассмотрение созданной топической логической структуры данных с реальными пользователями.

Семантическая модель объясняет понятия тематики, отношений между ними, информационные ограничения, устанавливаемые предметной областью.

Концептуальная модель данных - первоначальным макетом будущей структуры данных. Данная модель основывается в понятиях информационных единиц без присоединения к определенной СУБД. Тем не менее, семантическая структура данных не всегда обязана быть

сформулирована методами реляционной структуры данных. Ключевым методом создания концептуальной структуры данных в настоящее время являются разнообразные представления ER-диаграмм (Entity-Relationship, диаграммы сущность-связь). Единственную ER-структуру возможно конвертировать как в реляционную структуру данных, так и в структуру данных для многоуровневых сетевых СУБД, или в постреляционную структуру данных.

Ключевыми терминами ER-диаграммы являются сущность, связь и атрибут.

Сущность — это реальный или условный объект, представляющий весомое значение для интересуемой предметной области, сведения которой необходимо сохранить. Без учёта нюансов и тонкостей предметной области возможно утверждать, что сущности аналогичны таблицам реляционной структуры.

Связи в структуре позиционируются в виде метода, благодаря которому указываются отношения между сущностями, располагающимися в предметной области. Во время изучения отношений между сущностями возможно создание бинарных (между двумя типами сущностями), тернарных (между тремя типами сущностями) и общем случае n-арных связей.

В тематике работы различают типы объектов, предоставленные в информационной системе в любой момент времени окончательным набором экземпляров конкретного субъекта. В свою очередь субъект объекта идентифицируется по принадлежащему набору свойств, характерных признаков или параметров.

Связь - это объединение среди сущностей, включающих по одной сущности из каждого участвующего в связи типа сущности.

Атрибут представляет собой атомарный элемент структуры сущности, характеризующийся множеством неделимых единиц (атрибут «Наименование» сущности типа «ИП», атрибут «Минуты» сущности типа «Дата»).

Один или некоторое кол-во атрибутов сущности предметной области могут выполнять функцию ключевого атрибута, благодаря которому определяют экземпляры сущностей (сущность «Заявка» - ключ совокупность атрибутов «Дата», «Сумма», «Клиент» или один атрибут «Номер заявки»).

Разнообразные типы сущностей и образцы одного типа сущности объединены установленными отношениями, выраженными в виде связей в классической ER-модели.

Тип связи — рациональное объединение между объектами разных типов. Тип связи является подбором объединений между двумя типами сущностей. Каждому субъекту сущности принадлежит имя, описывающее его функцию.

Связи по признаку множественности могут быть трех типов:

- «один-к-одному»
- «один-ко-многим»
- «многие-ко-многим»

На рис. 2.1 показано инфологическое проектирование базы данных мобильного приложения автоматизированного приёма заявок.

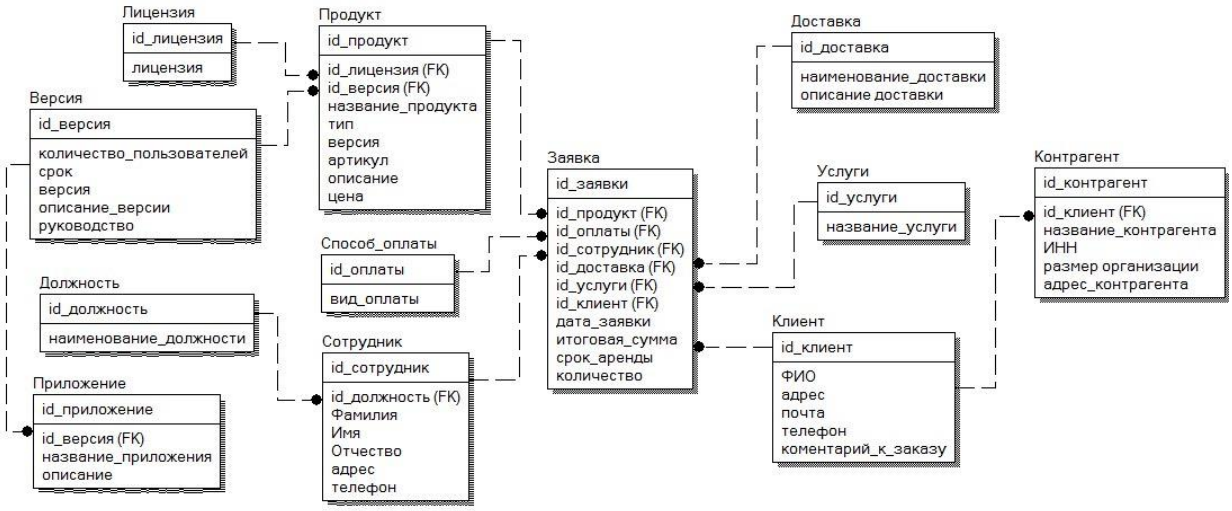

Рис. 2.1 Инфологическое проектирование базы данных мобильного приложения автоматизированного приёма заявок

Сущностями данной базы данных являются: лицензия, версия, должность, приложение, продукт, способ оплаты, сотрудник, заявка, доставка, услуги, клиент, контрагент.

После проведения концептуального проектирования будущей базы данных для мобильного приложения автоматизированного приёма заявок, необходимо создание логической схемы для учёта специфики определённой структуры данных. Логическое проектирование БД - подготовка реализации описательной системы данных на дополнительных записывающих устройствах; в этом случае мы взяли для примера основные связи, компоновку файлов и идентификационных номеров, которые необходимы для реализации удобного доступа к информации и все взаимодействия применимые с ограничениями целостности и средствами конфиденциальности.

Логическое проектирование, как известно, выполняет функции конечного этапа реализации структуры базы данных, завершение которого способствует в работе проектировщика принимать решения о реализации функций создаваемой структуры данных. Используя логическое проектирование структуры данных, в первую очередь необходимо задать определенную целевую СУБД. Из этого следует, что логическое проектирование не может выполнять свои функции без заданной СУБД. Между концептуальным и логическим проектированием присутствует непрерывная взаимная коммуникация, потому что действия, выполняемые на стадии логического проектирования с задачей оптимизации системы, обеспечивающие влияние на структуру данных логической модели.

Главной задачей логического проектирования структуры данных - это описательная функция физического выполнения логической разработки структуры данных.

Стадии логического проектирования структуры данных:

- Размещение общей логической модели данных в среду направленной СУБД;

- Конструирование таблиц данных в среде, направленной СУБД;
- Исполнение бизнес-правил организации;
- Конструирование физического описания БД;
- Анализ логических операций (установка приоритетов);
- Выбор директории;
- Выбор дополнительных идентификаторов;
- Постановка запросов к памяти.
- Создание защитных механизмов.
- Создание видов пользователя.
- Постановка регулирования доступа.

На рис. 2.2 показано даталогическое проектирование базы данных мобильного приложения.

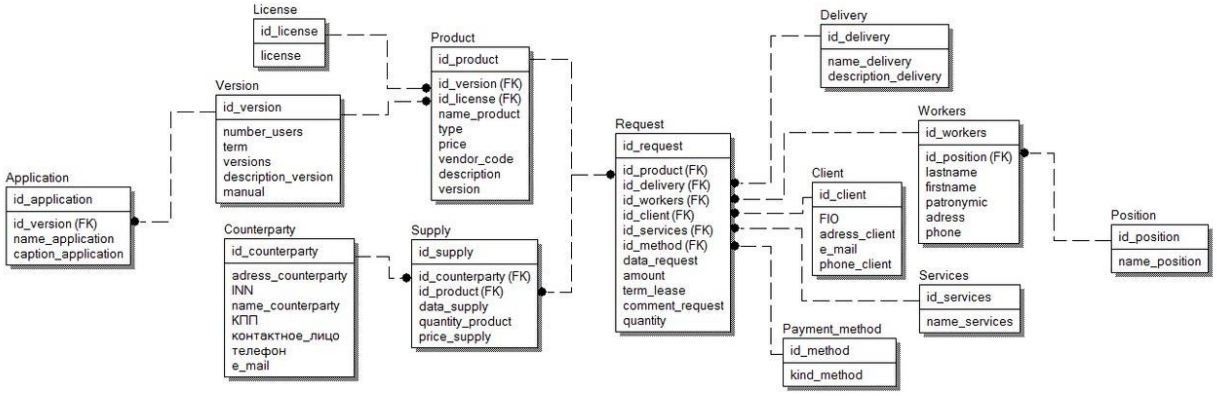

Рис. 2.2 Даталогическое проектирование базы данных CRM-приложения

После создания схем базы данных, необходимо выбрать интегрированную среду разработки.

2.2 Выбор интегрированной среды разработки

При создании мобильного приложения необходимо определиться с интегрированной средой разработки. Для комфортной разработки на всех этапах создания были выбраны следующие программные продукты:

- ERWin
- Visual Studio 2017 Community Edition
- Oracle SQL Developer
- Visual Studio Code

Выбор программного средства ERWin. ERwin считается стандартной программой для структурирования баз данных, принадлежащих к категории I - CASE. ERwin позволяет интегрировать модели со всех уровней иерархии. Структура высшего уровня образуется на первичных этапах конструирования информационных систем. Структура низшего уровня организуется на стадии построения и разработки кода и тестирования ПО.

Главная задача ERwin это конструирование информации, в следствие чего со стороны организации структуры данных его использование лимитировано. Программа организует создание структуры данных высшего уровня, позволяя автоматически преобразовывать их в модели низшего уровня, образует наглядную таблицу структуры данных и отображение данных на уровне разработки кода.

Нормой проектирования ERwin data modeler принято считать два шаблона: IDEF1X и IE (Information Engineering), организованные в таблицах «сущность-связь».

Схематические изображения структуры данных, проектируемые при помощи ERwin data modeler, разделяются на два типа: концептуальная и логическая модель.

CASE устройство CA ERwin r9 имеет функции, необходимые для создания структуры данных:

- Графическое понятие комплексных структур данных. За счет графических средств erwin разрешено в автоматическом режиме формировать модели, какие обеспечивают приятное понятие всей структуры данных;
- Применение обычных частей. В erwin предусмотрен комплект типовых моделей (рефренные модели), какие разрешают нарастить эффективность работы и исключить ошибки дублирования информации;
- Сравнение моделей данных и баз данных. За счет этого прибора имеется вероятность вести автоматическое сопоставление и синхронизацию частей бизнес действий (из моделей данных) с веществами базы данных;
- Интеграция с иными средствами моделирования. Erwin владеет широкими способностями по экспорту/ ввозу моделей. За счет интегрированных средств имеется вероятность обмениваться моделями с иными средствами моделирования, в том числе, поддерживающих uml нотации.

Преимущества, предоставляемые erwin, разрешено разглядывать по отношению к case средствам, направленным на моделирование данных. С данной точки зрения у пакета ca erwin r9 есть последующие достоинства:

- Возможность взаимодействия юзеров. Erwin предоставляет разные способности по размену информацией меж ролями внутри организации. За счет web портала бизнес аналитики и технические спецы имеют все шансы обретать доступ к моделям данных в понятных для них представлениях;
- Стандартные представления частей. Для снабжения целостности представления частей моделей данных в erwin употребляются стандартизованные представления имен объектов, стандартизованные типы данных и обычные образцы моделей (рефренные модели);
- Применение раскрытой архитектуры. Erwin гарантирует большие способности по интеграции с иными приборами моделирования действий и разработки информационных систем (наиболее 120 разных приборов);

- Визуальное понятие огромных массивов данных. За счет сильной графической системы и системы навигации разрешено снабдить графическое понятие моделей данных разной структуры и ступени вложенности;
- Совместная служба клиентов с репозиторием. Для действенной общей работы над моделями erwin гарантирует автоматический контроль версий, управление доступом, управление конфликтами и изменением моделей.

Выбор интегрированной среды Visual Studio 2017 CE. Visual Studio 2017 CE - это комплект программных устройств для организации программного продукта: от начальной стадии планировки до создания интерфейса пользователя, разработки кода, его тестирования и отладки, анализ свойств кода и производительности, развертывания в средах пользователей и сбора информации телеметрии по применению. Эти приборы предусмотрены для очень действенной общей работы; все они доступны в встроенной среде разработки (ide) Visual Studio.

Visual studio разрешено применять для организации разных типов приложений, от обычных приложений для магазина и игр для мобильных устройств, до огромных и трудных систем, обслуживающих компании и центры работы с данными. Возможно формирование:

Приложения и игры, с поддержкой кроссплатформенного режима;

Веб-представительства и веб-службы на базе Asp. Net, Jquery, Angularjs и прочих основных платформах;

Приложения для самых различных платформ и устройств, подключая, но не ограничивая функционал: Office, Sharepoint, Hololens, Kinect и т.д.;

Игры и графические дополнения для всех устройств Windows, учитывая Xbox, с поддержкой Directx.

В стандартном режиме Visual Studio гарантирует взаимодействие c#, c и c++, javascript, f# и visual basic. Visual Studio отлично работает и интегрируется со посторонними приложениями, к примеру, unity и apache cordova, с поддержкой расширений набора средств visual studio для unity и

приборов visual studio для apache cordova аналогично. Так же присутствуют возможности для расширения функционала Visual Studio путём собственных доработок и создания новых инструментов.

Можно выделить несколько преимуществ Visual Studio CE 2017:

- Реализация приложения для абсолютно любой популярной платформы;
- В одном инструменте находятся все необходимые конструкторы, редакторы и отладчики, а также доступ к множеству расширений;
- Работает с актуальными языками программирования - TypeScript, JavaScript, Phyton и прочими;
- Установка отдельных модулей позволяет выбрать необходимый спектр инструментов;
- Лёгкий рефакторинг, а также оптимизированный поиск ошибок и отладка;
- Синхронизация с мощными веб-платформами;
- Распоряжение кодом в репозиториях Git;
- Неограниченное число пользователей для одного программного продукта.

Выбор среды разработки Oracle SQL Developer. Oracle SQL Developer – программное обеспечение для создания баз данных на языках SQL и PL/SQL с возможностями управления, сосредоточенных на использовании в среде Oracle Database

Данный программный продукт находится в бесплатном доступе, а сама среда основана на языке программирование Java, функционирует кроссплатформенно, то есть везде, где доступна среда исполнения JAVA SE, находясь в постоянном совершенствовании. Также Oracle SQL Developer поддерживает функционирование с Oracle Database, что позволяет подключение дополнительных плагинов для интеграции из среды к прочим СУБД, например, доступ к IBM DB 2, Microsoft Access, Microsoft SQL Server, Teradata DB, SyBase ASE, MySQL. Для наибольшего удобства, максимально

локализована. Oracle SQL Developer поставляется через Oracle Web Agent – модулем расширения для Apache web-server, организующим создание динамических web-страниц с использованием PL/SQL с Oracle SQL Developer. Так же в оболочке поддерживается возможность реализации расширений, позволяющих реализовывать дополнительные функции. Эти расширения реализовываются как со стороны корпорации Oracle, так и со стороны пользователей или других разработчиков. К примеру, существуют расширения, позволяющие картографическую визуализацию хранимой в структуре данных геоинформации или позволяющие визуализацию ER-диаграмм. Существует возможность сторонней работы с приложением без установки на персональный компьютер, ведь все необходимые настройки находятся и хранятся в XML файлах.

Из отрицательных качеств Oracle SQL Developer можно отметить лишь непривычный интерфейс (проблема исчезает в дальнейшем изучении ПО) и высокие требования к системе, в особенности к оперативной памяти устройства, при котором ведется разработка. Кроме того, разработка при платформе Windows все надстройки SQL Developer находятся не в реестрах, а в XML файлах профиля пользователя, что обязательно нужно учитывать.

В процессе подробного ознакомления с данным программным продуктом, он был избран как самый оптимальный и выгодный, для создания баз данных для будущего мобильного приложения.

Выбор среды разработки Visual Studio Code. Visual Studio Code— мультиплатформенный редактор исходного кода, который поддерживает основные функции системы программных средств, созданных при Microsoft. Данная программа представляется как «лёгкий» редактор кода для мультиплатформенного создания web-приложений и облачных сервисов. Также данный программный продукт находится в бесплатном доступе, постоянно дорабатывается как открытое программное обеспечение с доступом в версиях для всех популярных платформ (Windows, Linux, OS X)

Преимущественной частью VSC считаются разработки свободного проекта Atom, разрабатываемые компанией GitHub. VSC представляет собой надстройкой Electron, объединяющим в себе преобразователь о содержимом web-страниц на базе Chromium и Node.JS. Во время создания были использованы наработки, полученные при создании web-редактора Monaco, необходимого для Visual Studio Online.

В данной программе имеется встроенный отладчик, сервисы для взаимодействия с репозиториями и средствами облегчения понимания работы программы, навигация по коду, автоматическое дополнение аналогичных конструкций и разнообразных контекстных подсказок Продукт поддерживает разработку для платформ ASP.NET и Node.js, и позиционируется как легковесное решение, которое позволяет обойтись без полной интегрированной среды разработки. Среди поддерживаемых языков и технологий: JavaScript, C++, C#, TypeScript, jade, PHP, Python, XML, Batch, F#, Docker, CoffeeScript, Java, HandleBars[en], R, Objective C, PowerShell, Lua, Visual Basic, Markdown, JSON, HTML, CSS, LESS и SASS, Swift.

2.3 Конструирование интерфейса пользователя

Важную роль в создании мобильного приложения играет дружественный и многофункциональный интерфейс пользователя.

Интерфейс пользователя – это взаимодействие информационной модели предметной области, средств программирования и ряд взаимодействий работника с основанием данных фирмы и компонентами, которые дают возможность создания информационной системы в процессе работы с программной средой разработки.

Под информационной системой имеется ввиду осознание главного направленности работы фирмы, формируемое с поддержкой зрительных объектов, представляющих ряд и взаимодействие комплекта определённых компонентов.

Программный продукт и методы проектирования пользовательского интерфейса ориентируются составом программ, которые есть в использовании у работника фирмы, и от установленной цели пред ним.

Действенная работа пользователя ориентируется не только активным вероятностям оказавшегося в его использовании программного продукта, но и приемлемостью для работника данных вероятностей. Качество пользовательского интерфейса разрешает применить способности потенциально имеющихся ресурсов, а, например, же считается автономным описанием программного продукта, сравнимого по значениям с этими аспектами: надёжность и действенное использование вычисляемых ресурсов.

Одно из ведущих плюсов пользовательского интерфейса произведено в том, собственно, что работник всякий раз понимает, что он управляет процессами программного продукта, а не продукт управляет им.

При разработке комфортного и активного пользовательского интерфейса, он обязан отвечать определенному ряду качеств:

- естественность;
- согласованность;
- дружественность;
- принцип «обратной связи» с пользователем;
- простота;
- гибкость;
- эстетическая притягательность.

Натуральный интерфейс — это интерфейс, который не разрешает работнику заменять нормальные для его работы способности заключения задач. Данное определение соответствует что, выдаваемые приложением итоги и всплывающие окна не обязаны обосновываться вспомогательной информацией.

Согласованность гарантирует работникам выносить имеющую информацию в заключения, поставленные задачи, осваивать перечень возможностей скорее, и в следствие этого уделять больший интерес

установленной цели, а не проводить время с ознакомлением разного семейства компонентов и их выполняемых операций. Этим образом согласованность готовит пользовательский интерфейс больше известным и прогнозируемым, например, же дает информацию о поведении интерактивных составляющих.

В пользовательском интерфейсе для реализации свойства согласованности необходимо учитывать его различные критерии:

- Согласованность в соответствии с товаром. Это означает, что каждая команда системы должна исполнять только одну операцию, при любых изменениях в программном продукте.
- Согласованность в пределах среды разработки. Поддержание согласованности с интерфейсом, предоставляемым операционной системой (например, ОС Windows), пользовательское приложение может «опираться» на те знания и навыки пользователя, которые он получил ранее при работе с другими приложениями.
- Согласованность в использовании метафор. Если поведение некоторого программного объекта выходит за рамки того, что обычно подразумевается под соответствующей ему метафорой, у пользователя могут возникнуть трудности при работе с таким объектом.

Существуют следующие принципы проектирования интерфейса:

- дружественность интерфейса (иначе принцип прощения пользователя) – данный принцип предполагает разрешение совокупности наборов действий и предупреждений пользователей о ситуациях, которые могут тем или иным образом навредить системе или сведениям в ней: предполагается возможность отмены или редактирования уже выполненных действий.

Не редко бывают случаи, когда при наличии хорошо спроектированного интерфейса пользователи системы могут совершать ряд ошибок. Ошибки могут быть следующих типов:

- физического типа – тип ошибки, при котором происходит случайный выбор неверной команды или данных;

- логического типа – тип ошибки, при котором происходит принятие неверного решения на выбор команды или данных.

Эффективный интерфейс программного продукта обязан позволять избегать ситуации, которые возможно закончатся ошибками системы. Кроме этого, он обязан уметь приспособиться к возможным ошибкам пользователя и облегчать ему процесс устранения результатов этих ошибок.

- Принцип «обратной связи» - данный принцип предполагает обеспечение обратной связи для действий пользователя. Любое действие пользователя обязано получать зрительное, а бывает, что и звуковое доказательство того, что система приняла введенную команду; при всем этом вид реакции, насколько это возможно, должен учесть природу исполненного действия.

Обратная взаимосвязь эффективна в том случае, когда она реализуется вовремя, то есть как можно ближе к точке заключительного взаимодействия пользователя с системой. В случае, когда компьютер обрабатывает поступившее от пользователя задание, может быть полезно дать пользователю информацию, которая касается взаимодействия процессов, а также возможность прервать данное действие в случае необходимости.

- Простота интерфейса – данный принцип предполагает обеспечение простого пользовательского интерфейса. При всем этом подразумевается не упрощенство, а обеспечение легкости в его изучении и в применении. Помимо этого, он обязан давать доступ ко всему перечню многофункциональных возможностей, предусмотренным этим продуктом. Однако, реализация доступа к большому объему многофункциональных возможностей и обеспечение простоты работы противоречат друг другу. Разработка эффективного интерфейса необходима для балансирования данных целей.

Одни из вероятных путей поддержки простоты пользовательского интерфейса служит предоставление на экране сведений, минимально нужных для исполнения пользователем следующего шага выполнения задания. А именно, следует игнорировать многословные командные названия и сообщения. Необдуманные названия затрудняют пользователю извлечение значимых данных.

- Гибкость интерфейса программного продукта – данный принцип предполагает способность учитывать подготовленность и производительность труда пользователя. Данное свойство системы подразумевает вероятность изменения конфигурации текстуры разговора. Концепция эластичного интерфейса в текущее время считается одной из главных сфер исследования взаимодействия человека и персонального компьютера. Главной проблемой является выбор и применение признаков для определения потребности внесения конфигураций и их сущностей.
- Эстетическая привлекательность – данный принцип предполагает способность зрительного представления применяемых объектов, гарантирующих передачу необходимых сведений о поведении и действиях различных объектов.

Стоит отметить, что качество интерфейса программного продукта трудно расценить количественными характеристиками, но и объективную оценку можно получить на базе характеристик, а именно:

- временной промежуток, нужный конкретному человеку для достижения значения познаний и навыков по работе с приложением;

- сохранение приобретенных навыков по прошествии некоторого временного промежутка;

- скорость решения задач при помощи программного продукта, в котором учитывается не быстрота работы программы, а время, нужное для достижения поставленной цели;

- личная удовлетворенность человека при работе с программным продуктом.

При создании мобильного приложения были учтены все вышестоящие требования и, соответственно, реализованы. На рис. 2.3 представлено стартовое окно мобильного приложения.

Рис. 2.3. Стартовое окно мобильного приложения

Стартовое окно мобильного приложения содержит два состояния перехода: «Вход в систему» (предназначено для зарегистрированных пользователей) и «Зарегистрироваться» (Создания новой учётной записи пользователя). Далее подробно изучим процесс регистрации клиента. На рис. 2.4 представлена форма регистрации клиента.

Рис. 2.5. Окно регистрации

Далее необходимо выбрать роль. На рис. 2.6 представлено окно выбора роли.

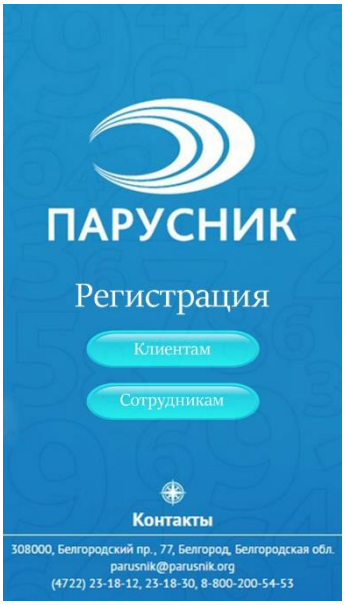

Рис. 2.6. Выбор роли

Данное состояние перехода предлагает нам пройти процесс регистрации либо для сотрудника ООО «Парусник», либо как клиент, но нас интересует исключительно регистрация клиента. Для регистрации клиента ему необходимо ввести данные, указанные в полях с максимальной точностью, для его регистрации в базах данных ООО «Парусник». Все поля обязательны для заполнения. Все внесенные данные автоматически

направляются для сверки с базой данных ООО «Парусник» и если существует ИНН, введенный клиентом, в базе данных УДП, то регистрация для пользователя прошла успешно, и мы имеем право допустить его к дальнейшей работе с программой. На рис. 2.7 представлен выбор модулей

Рис. 2.7. Выбор доступных модулей

В процессе сотрудничества с ООО «Парусник» клиент приобретает различного рода лицензии, в которых содержатся обслуживаемые модули (Бух. учёт, Web-обслуживание, Программное обслуживание и т.д.). Номера данных лицензий закреплены в базе данных и непосредственно связаны с ИНН, вводимым клиентом при регистрации, и если есть хотя бы один обслуживаемый модуль, то он отобразится в приложении, и его можно будет избрать для дальнейшей работы с ним. На рис. 2.8 представлена форма для отправки сообщения клиентом.

Рис. 2.8. Сообщение от клиента

После выбора доступного модуля, клиенту предлагается оставить сообщение-заявку, в которой будет указано: конкретная дата обращения, описание сути проблемы с указанием типа заявки и постановкой приоритета желаемой даты выполнения. После того как клиент оставил свою первую заявку, мы имеем право перенести его на основное окно приложения, которое отвечает за его основные функции. На рис. 2.9 показано основное окно.

Рис. 2.9. Основное окно приложения

Данное окно содержит 3 состояния перехода, выполняющих следующие функции: «Принятые сообщения» показывают ответы, приходящие от организации. «Ваши заявки» отображают заявки, оставленные клиентом ранее. «Новая заявка».

На рис. 2.10. показано окно «Принятые сообщения».

Рис. 2.10. Окно «Принятые сообщения»

На рис. 2.11. показано окно «Ваши заявки».

Рис. 2.11. Окно «Ваши заявки»

Состояние перехода «Принятые сообщения» содержит в себе таблицу с сообщениями-ответами на заявку, поданную клиентом. В этой таблице содержится стандартный флаг для выделения сообщений и работы с ними (Отметить как прочитанное, удаление и т.д.,), от какого сотрудника поступило данное сообщение, тема этого сообщения и дата отправки. Аналогично и с заявками, оставленными клиентом. Только присутствует номер заявки по которому она будет идентифицироваться в базе данных, тип заявки (звонок-консультация, вызов специалиста, продление лицензии и т.д.), ФИО сотрудника, взявшего и работающего с заявкой, дата принятия заявки, и статус её готовности.

2.4 Модульная организация программы

Создаваемое приложение представляет собой множество технологий разработки, интегрированных сред программирования, редакторов пользовательского интерфейса, объединенных в одну целую систему. Для удобного схематического обозначения всех компонентов необходимо создать модульную схему с дальнейшим пояснением связей взаимодействия и значения каждого модуля отдельно. На рис. 2.12изображена модульная схема, где указаны все компоненты, их функции и взаимодействие друг с другом.

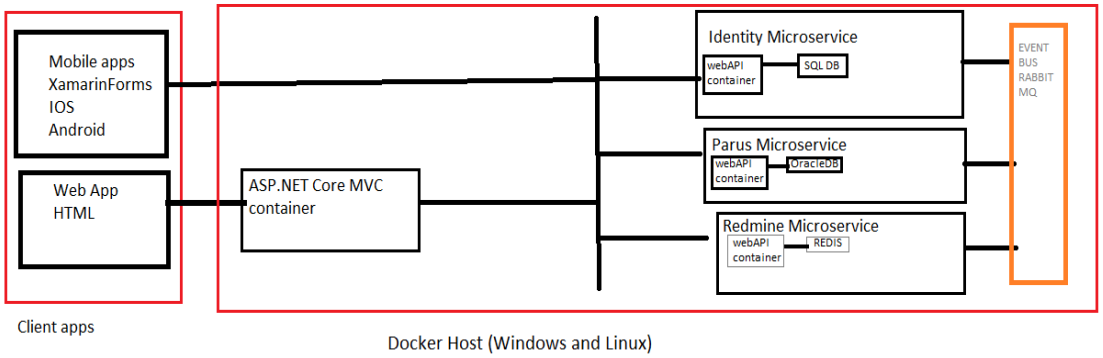

Рис. 2.12. Модульная организация программы

В данной модульной схеме отображены все технологии и компоненты, использованные для создания программы, но необходимо рассмотреть их назначение и функционирование по отдельности.

Присутствует два глобальных блока: «Client apps» и «Docker Host». Client apps отвечает за визуальное оформление мобильного приложения и веб-приложения (непосредственно сайт ООО «Парусник»). Основные визуальные представления для мобильного приложения были разработаны на платформе XamarinForms, а для web-приложения использовался язык программирования HTML.

Основное внимание стоит обратить на второй блок, отвечающий за исполнения задач и целей программы. Docker Host представляет собой изолированный контейнер, в котором выполняется весь функционал программы. Он включает в себя несколько микросервисов и меньших контейнеров, исполняющих определенные функции и взаимодействующие друг с другом. Мобильное приложение обращается к одному из микросервисов (или к нескольким), в следствии чего происходит ряд процедур изменяющих коренную базу данных ООО «Парусник».

Identity Microservice созданный при участии технологий ASP. NET Core 1.1, Entity Framework 1.1, Microsoft SQL Server, выполняет функции аутентификатора, отвечает за регистрацию пользователя в приложении и сохраняет введенные данные для дальнейшего пользования приложением без

очередного ввода необходимых данных, предоставляет доступ к остальным функциям программы.

Parus Microservice созданный при участии технологий ASP. NET Core 1.1, Dapper, Oracle DB необходим для взаимодействия с базой данных ООО «Парусник» и сверки введенной информации пользователей уже с существующей. Аналогично функционирует и RedMine Microservice. Обмен данными между микросервисами осуществляется через шину RabbitMQ.

Если рассматривать веб приложение, всё происходит аналогичным путём за исключением момента присутствия ASP. NET Core MVC container – специализированного контейнера, который заводит заявки через веб-приложение.

ГЛАВА 3. РАЗРАБОТКА МОБИЛЬНОГО ПРИЛОЖЕНИЯ

3.1 Разработка информационной базы данных

После успешного проектирования всех этапов будущего мобильного приложения необходимо приступить к созданию основного функционала программы путём написания эффективного, работоспособного кода. Для начала необходимо автоматизировать внесение данных в базу данных ООО «Парусник» и осуществить сверку информации, введенной пользователем, и уже существующей информацией для выдачи токена доступа и дальнейшей работой с программой.

Для этого нам необходимо разработать следующий код, представленный на листинге 3.1.

Листинг 3.1. Авторизация

```
using System; namespace Microsoft.AspNetCore.Identity{
   public class IdentityUser : IdentityUser<string>{
      public IdentityUser(){
         Id = Guid.NewGuid().ToString();}
      public IdentityUser(string userName) : this(){
         UserName = userName;}}
   public class IdentityUser<TKey> where TKey : IEquatable<TKey>{
      public IdentityUser() { }
      public IdentityUser(string userName) : this(){ UserName = userName;}
      public virtual TKey Id { get; set; }
      public virtual string UserName { get; set; }
      public virtual string NormalizedUserName { get; set; }
      public virtual string Email { get; set; }
      public virtual string NormalizedEmail { get; set; }
      public virtual bool EmailConfirmed { get; set; }
      public virtual string PasswordHash { get; set; }
      public virtual string SecurityStamp { get; set; }
      public virtual string ConcurrencyStamp { get; set; } = Guid.NewGuid().ToString();
      public virtual string PhoneNumber { get; set; }
      public virtual bool PhoneNumberConfirmed { get; set; }
      public virtual bool TwoFactorEnabled { get; set; }
      public virtual DateTimeOffset? LockoutEnd { get; set; }
      public virtual bool LockoutEnabled { get; set; }
      public virtual int AccessFailedCount { get; set; }
      public override string ToString(){ return UserName;} } }
```

Если все данные пользователя введены верно, то информация, введенная им, сохраняется при микросервисе, для дальнейшего использования. Таким образом мы создаём полноценное представление модели пользователя.

Далее нам необходимо создать более упрощённую модель, унаследованную от первоначальной модели, указанной в листинге 3.2.

Листинг 3.2. Упрощение модели авторизации

```
using Microsoft.AspNetCore.Identity.EntityFrameworkCore;
namespace Parusnik.OnlineAssistant.Identity.Core.Models{
   public class User : IdentityUser{
      public string FirstName { get; set; }
      public string LastName { get; set; }
      public string CompanyName { get; set; }
      public string CompanyInn { get; set; }
      public string CompanyRegistrationPrefix { get; set; }
      public string CompanyRegistrationNumber { get; set; } } }
```

Данный класс позволяет нам использовать упрощенную модель данных о пользователе в дальнейшей необходимости. Также можно добавить некоторые формальности, представленные на листинге 3.3.

Листинг 3.3. Дополнительная информация

```
using Parusnik.OnlineAssistant.Identity.Core.Models;
namespace Parusnik.OnlineAssistant.Identity.API.Models{
   public class ApplicationUser : User{
      public Gender Gender { get; set; }}
   public enum Gender{
      NotSpecified = 0,
      Male = 1,
      Female = 2}}
```

Теперь нам необходимо построить физическую структуру базы данных для микросервисов. Для этого создаём класс, объединяющий всю необходимую информацию из прошлых моделей пользователя, показанную на листинге 3.4.

Листинг 3.4. Построение физической структуры

```
using Microsoft.AspNetCore.Identity.EntityFrameworkCore;
using Microsoft.EntityFrameworkCore;
using Parusnik.OnlineAssistant.Identity.Core.Models;
```

Продолжение. Листинг 3.4. Построение физической структуры

```
namespace Parusnik.OnlineAssistant.Identity.API.Data{
   public class ApplicationDbContext : IdentityDbContext<User>{
      public ApplicationDbContext(DbContextOptions<ApplicationDbContext> options)
         : base(options){}
      protected override void OnModelCreating(ModelBuilder builder){
         base.OnModelCreating(builder);}}}
```

На этом этапе мы завершаем разработку информационной базы данных.

3.2 Программная часть разработки мобильного приложения

После создания базы данных для микросервисов, необходимо обеспечить их стабильное, равномерное функционирование, оптимизируя затраты оперативной памяти и избежание потерь данных. Для начала приложению необходимо понять, что модель данных о пользователе, сконструированная в самом микросервисе идентична той, что находится в базе данных ООО «Парусник». Для достижения цели необходимо разработать следующий класс, показанный на листинге 3.5.

Листинг 3.5. Сверка данных

```
using System;
using System.Threading.Tasks;
using Microsoft.AspNetCore.Mvc;
using Microsoft.Extensions.Logging;
using Parusnik.OnlineAssistant.Parus.API.Areas.IdentityService.Models;
using Parusnik.OnlineAssistant.Parus.API.Areas.IdentityService.Services;
namespace Parusnik.OnlineAssistant.Parus.API.Areas.IdentityService.Controllers{
   [Area("IdentityService")]
   [Route("api/v1/parus/[controller]")]
   public class IdentityController : Controller{
      private readonly SignInManager<Customer> _signInManager;
      private readonly ILogger _logger;
      public IdentityController(SignInManager<Customer> signInManager,
         ILogger<IdentityController> logger){
         _signInManager = signInManager;
         _logger = logger;}
      [HttpGet]
      [Route("[action]")]
      public async Task<IActionResult> Authorize([FromQuery] string inn,
         [FromQuery] string registrationPrefix,
```

Продолжение. Листинг 3.5. Сверка данных

```
            [FromQuery] string registrationNumber){
            try{
               var result = await _signInManager.ValidateCredentialsAsync(inn,
registrationPrefix, registrationNumber);
               _logger.LogInformation(result.Succeeded ? "Authorization succeeded." :
"Authorization failed.");
               return Ok(result);}
            catch (Exception ex){
               _logger.LogError($"Authorize error: {ex.Message}");
               return NotFound();}}}}
```

В процессе прохождения модели данных через этот класс возможно выявить, что, либо вся информация пользователем была введена верно, так как нашла все возможные совпадения с БД ООО «Парусник», либо одно из полей было введено некорректно или вовсе не заполнено, в следствие чего приложение возвращает отказ в доступе. Если же вся информация верна, клиент имеет дальнейший доступ к приложению. По мимо этого необходимо учесть, как выглядит модель взаимодействия клиента с БД ООО «Парусник». Данная модель представлена ниже, в листинге 3.6.

Листинг 3.6. Проверка достоверности

```
using System;
using System.Threading.Tasks;
using Microsoft.AspNetCore.Mvc;
using Microsoft.Extensions.Logging;
using Parusnik.OnlineAssistant.Parus.API.Areas.IdentityService.Models;
using Parusnik.OnlineAssistant.Parus.API.Areas.IdentityService.Services;
namespace Parusnik.OnlineAssistant.Parus.API.Areas.IdentityService.Controllers{
   [Area("IdentityService")]
   [Route("api/v1/parus/[controller]")]
   public class IdentityController : Controller{
      private readonly SignInManager<Customer> _signInManager;
      private readonly ILogger _logger;
      public IdentityController(SignInManager<Customer> signInManager,
         ILogger<IdentityController> logger){
         _signInManager = signInManager;
         _logger = logger;}
      [HttpGet]
      [Route("[action]")]
      public async Task<IActionResult> Authorize([FromQuery] string inn,
         [FromQuery] string registrationPrefix,
         [FromQuery] string registrationNumber){
         try{
```

Продолжение. Листинг 3.6. Проверка достоверности

```
var result = await _signInManager.ValidateCredentialsAsync(inn, registrationPrefix, registrationNumber);
logger.LogInformation(result.Succeeded ? "Authorization succeeded." : "Authorization failed.");
return Ok(result);}
catch (Exception ex){
logger.LogError($"Authorize error: {ex.Message}");
return NotFound();}  }}}
```

Любая информация, введенная пользователем, и сформированная в модель должна проходить именно через этот класс.

Для заключительного этапа задания функционала мобильному приложению необходимо внедрить зависимости или же независимости от объектов, для оптимизации работы приложения и во избежание потери данных или их «зависание». Разработаем класс, способствующий этому, представлен на листинге 3.7.

Листинг 3.7. Оптимизация работы

```
using Microsoft.AspNetCore.Builder;
using Microsoft.AspNetCore.Hosting;
using Microsoft.Extensions.Configuration;
using Microsoft.Extensions.DependencyInjection;
using Microsoft.Extensions.DependencyInjection.Extensions;
using Microsoft.Extensions.Logging;
using Parusnik.Data.Oracle.Extensions;
using Parusnik.OnlineAssistant.Parus.API.Areas.IdentityService.Models;
using Parusnik.OnlineAssistant.Parus.API.Areas.IdentityService.Services;
namespace Parusnik.OnlineAssistant.Parus.API{
   public class Startup{
      public Startup(IHostingEnvironment env){
         var builder = new ConfigurationBuilder()
            .SetBasePath(env.ContentRootPath)
            .AddJsonFile("appsettings.json", optional: false, reloadOnChange: true)
            .AddJsonFile($"appsettings.{env.EnvironmentName}.json", optional: true)
            .AddEnvironmentVariables();
         Configuration = builder.Build();}
      public IConfiguration Configuration { get; set; }
      public void ConfigureServices(IServiceCollection services){
         services.AddMvc();
         services.AddOracleDataAccess(options =>{
            options.ConnectionString = Configuration["Database:ConnectionString"];
            options.Schema = Configuration["Database:Schema"];});
services.TryAddScoped<ICustomerStore<Customer>,CustomerStore<Customer>>();
services.TryAddScoped<ICustomerValidator<Customer>,CustomerValidator<Customer
>>();
```

Продолжение. Листинг 3.7. Оптимизация работы

```
        services.TryAddScoped<IdentityErrorDescriber, IdentityErrorDescriber>();
        services.TryAddScoped<CustomerManager<Customer>,
CustomerManager<Customer>>();
        services.TryAddScoped<SignInManager<Customer>, SignInManager<Customer>>();}
        public void Configure(IApplicationBuilder app, IHostingEnvironment env,
ILoggerFactory loggerFactory){
                loggerFactory.AddConsole(Configuration.GetSection("Logging"));
                loggerFactory.AddDebug();
                if (env.IsDevelopment()){
                   app.UseDeveloperExceptionPage();}
                   app.UseMvcWithDefaultRoute();}}}
```

На данном этапе мы заканчиваем организацию функционала мобильного приложения

3.3 Тестирование мобильного приложения

Тестирование функционала приложения будет проходить с помощью технологий модульного тестирования таких как Xunit и MOQ. Прежде чем запускать приложение и предоставлять его к работе с реальными пользователями, его необходимо протестировать со случайным набором данных, сгенерированных самостоятельно разработчиком. Вышеупомянутые методы мы используем для генерации кода изоляции интерфейса и виртуальных методов от не виртуальных и статических методов.

Реализация технологии Xunit представлена на листинге 3.8.

Листинг 3.8. Реализация технологии Xunit

```
using System.Linq;
using Microsoft.AspNetCore.Identity;
using Parusnik.OnlineAssistant.Identity.Core;
using Xunit;
namespace Parusnik.OnlineAssistant.Parus.API.Test{
   public static class CustomerIdentityResultAssert{
      public static void InSuccess(CustomerIdentityResult result){
         Assert.NotNull(result);
         Assert.True(result.Succeeded);}
      public static void IsFailure(CustomerIdentityResult result){
         Assert.NotNull(result);
         Assert.False(result.Succeeded);}
```

Продолжение. Листинг 3.8. Реализация технологии Xunit

```
public static void IsFailure(CustomerIdentityResult result, string error){
    Assert.NotNull(result);
    Assert.False(result.Succeeded);
    Assert.Equal(error, result.Errors.First().Description);}
 public static void IsFailure(CustomerIdentityResult result, IdentityError error){
    Assert.NotNull(result);
    Assert.False(result.Succeeded);
    Assert.Equal(error.Description, result.Errors.First().Description);
    Assert.Equal(error.Code, result.Errors.First().Code);}}}
```

Реализация технологии MOQ представлена на листинге 3.9.

Листинг 3.9. Реализация технологии MOQ

```
using System;using System.Text;
using Microsoft.Extensions.Logging;using Moq;
using Parusnik.OnlineAssistant.Parus.API.Areas.IdentityService.Services
namespace Parusnik.OnlineAssistant.Parus.API.Test{
public static class MockHelpers{
public static StringBuilder LogMessage = new StringBuilder();
public static Mock<CustomerManager<TCustomer>>
MockCustomerManager<TCustomer>() where TCustomer : class{
var store = new Mock<ICustomerStore<TCustomer>>();
var manager = new Mock<CustomerManager<TCustomer>>(store.Object, new
CustomerValidator<TCustomer>(), null);
return manager;} public static Mock<ILogger<T>>
MockLoggerManager<T>(StringBuilder logStore = null) where T : class{
logStore = logStore ?? LogMessage; var logger = new Mock<ILogger<T>>();
logger.Setup(x => x.Log(It.IsAny<LogLevel>(), It.IsAny<EventId>(),
It.IsAny<object>(); It.IsAny<Exception>(), It.IsAny<Func<object, Exception, string>>()))
Callback((LogLevel logLevel, EventId eventId, object state, Exception exception,
Func<object, Exception, string> formatter) =>{
logStore.Append(formatter == null ? state.ToString() : formatter(state, exception));
logStore.Append(" ");});logger.Setup(x => x.BeginScope(It.IsAny<object>())).Callback((object
state) =>{logStore.Append(state.ToString());logStore.Append(" ");});
TestCustomerManager<TCustomer>(ICustomerStore<TCustomer> store = null) where
TCustomer : class{store = store ?? new Mock<ICustomerStore<TCustomer>>().Object;
var validator = new Mock<ICustomerValidator<TCustomer>>();
var logger = new Mock<ILogger<CustomerManager<TCustomer>>>();
var customerManager = new CustomerManager<TCustomer>(store, validator.Object,
logger.Object);return customerManager;}}}
```

Совместно с двумя технологиями, указанными выше, объединим их в специальную модель сверки введенных тестовых данных, создавая таким образо\м некий механизм извлечения, с БД ООО «Парусник». Тестирование метода доступа извлечения выглядит таким образом показана на листинге 3.10.

Листинг 3.10. Тестирование метода доступа извлечения

```
using System.Threading.Tasks;
using Microsoft.AspNetCore.Mvc;
using Microsoft.Extensions.Logging;using Moq;
using Parusnik.OnlineAssistant.Parus.API.Areas.IdentityService.Controllers;
using Parusnik.OnlineAssistant.Parus.API.Areas.IdentityService.Models;
using Parusnik.OnlineAssistant.Parus.API.Areas.IdentityService.Services;
using Xunit;
namespace Parusnik.OnlineAssistant.Parus.API.Test{
   public class IdentityControllerTest{ [Fact]
      public async Task AuthorizeReturnsNotFoundWhenManagerThrowsExceptions(){
         var customerManager = MockHelpers.MockCustomerManager<Customer>();
         var loggerSignInManager =
MockHelpers.MockLoggerManager<SignInManager<Customer>>();
         var signInManager = new Mock<SignInManager<Customer>>(
            customerManager.Object,
            new IdentityErrorDescriber(),
            loggerSignInManager.Object);
         var loggerIdentityController = new Mock<ILogger<IdentityController>>();
         var controller = new IdentityController(signInManager.Object,
loggerIdentityController.Object);
         var result = await controller.Authorize("1234567890", "PR", "0000");
         Assert.IsType<NotFoundResult>(result);}   }}
```

На данном этапе процесс тестирования подходит к концу, и в конечном итоге мы имеем полноценное приложение, работающее с базой данных организации, полностью оптимизированное, протестированное, способное к самореализации.

ЗАКЛЮЧЕНИЕ

В процессе выполнения ВКР было спроектировано и разработано мобильное приложение автоматизированного приёма заявок.

Также были решены поставленные задачи, а именно:

- подробно изучена предметная область - ООО «Парусник-Белгород»;

- проведен анализ уровня автоматизации организации для разработки мобильного приложения;

- проектирована и разработана база данных для мобильного приложения;

- проектирован пользовательский интерфейс для разработанного программного продукта;

- протестирован контрольный пример.

Мобильное приложение позволило автоматизировать процесс приёма, обработки и отчетности заявок, приходящих от клиентов организации, что позволило оптимизировать работу организации, и в целом, положительно влияет на качество обслуживания клиентов.

СПИСОК ИСПОЛЬЗОВАННОЙ ЛИТЕРАТУРЫ

1. Alessandro Del Sole. Visual Studio Code Succinctly. — SyncFusion Inc., 2016. — 128 c

2. Валерий Коржов. Многоуровневые системы клиент-сервер. Издательство Открытые системы (17 июня 1997)

3. Harper, Sue (October 20, 2009), Oracle SQL Developer

4. Hotka, Dan (October 10, 2006), Oracle SQL Developer Handbook

5. Dirk Merkel Docker: lightweight Linux containers for consistent development and deployment

6. Emrah Ayanoglu; Yusuf Aytaş; Dotan Nahum. Mastering RabbitMQ. — Packt Publishing

7. Steve Fenton. TypeScript Succinctly. — Syncfusion Ink., 2014

8. Nathan Rozentals. Mastering TypeScript - Build enterprise-ready, industrial strength web applications using TypeScript and leading JavaScript Frameworks. — Packt Publishing, 2015

9. Steve Fenton. Pro TypeScript: Application-Scale JavaScript Development. — Apress, 2014.

10. Dan Maharry. TypeScript revealed. — Apress, 2013.

11. Кристиан Нейгел и др. C# 5.0 и платформа .NET 4.5 для профессионалов = Professional C# 5.0 and .NET 4.5. — М.: «Диалектика», 2013

12. Эндрю Троелсен. Язык программирования C# 5.0 и платформа .NET 4.5, 6-е издание = Pro C# 5.0 and the .NET 4.5 Framework, 6th edition. — М.: «Вильямс», 2013

13. Джеф Просиз. Программирование для Microsoft .NET = Programming Microsoft .NET. — М.: Русская редакция, 2003.

14. Адам Фримен. ASP.NET MVC 5 с примерами на C# 5.0 для профессионалов, 5-е издание = Pro ASP.NET MVC 5. — М.: «Вильямс», 2014

15. Джесс Чедвик, Тодд Снайдер, Хришикеш Панда. ASP.NET MVC 4: разработка реальных веб-приложений с помощью ASP.NET MVC = Programming ASP.NET MVC 4: Developing Real-World Web Applications with ASP.NET MVC. — М.: «Вильямс», 2013.

16. Адам Фримен. ASP.NET 4.5 с примерами на C# 5.0 для профессионалов, 5-е издание = Pro ASP.NET 4.5 in C#, 5th Edition. — М.: «Вильямс», 2014.

17. Даниэль Арсеновски. Рефакторинг в C# и ASP.NET для профессионалов = Professional Refactoring in C# & ASP.NET. — М.: «Диалектика», 2009.

18. Гэри Неббет. Справочник по базовым функциям API Windows NT/2000 = Windows NT/2000 Native API Reference. — М.: «Вильямс», 2002.

19. Kossiakoff A., Sweet W. N., Seymour S. J., Biemer S. M. Systems Engineering Principles and Practice. — 2-е изд. — Hoboken, New Jersey: A John Wiley & Sons, 2011.

20. Pyster, A., D. Olwell, N. Hutchison, S. Enck, J. Anthony, D. Henry, and A. Squires (eds). Guide to the Systems Engineering Body of Knowledge (SEBoK) version 1.0. — The Trustees of the Stevens Institute of Technology, 2012

ПРИЛОЖЕНИЕ

CustomerIdentityResult.cs

```
using System.Collections.Generic;
using System.Linq;
using Microsoft.AspNetCore.Identity;
namespace Parusnik.OnlineAssistant.Identity.Core
   public class CustomerIdentityResult
   {
      private readonly List<IdentityError> _errors = new List<IdentityError>();
      public bool Succeeded { get; protected set; }
      public IEnumerable<IdentityError> Errors => _errors;
      public static CustomerIdentityResult Success { get; } =
         new CustomerIdentityResult { Succeeded = true };
      public static CustomerIdentityResult Failed(params IdentityError[] errors)
      {
         var result = new CustomerIdentityResult() { Succeeded = false };
         if (errors != null)
         {
            result._errors.AddRange(errors);
         }
         return result;
      }
      public override string ToString()
      {
         return Succeeded ?
            "Succeeded" : $"Failed : {string.Join(",", Errors.Select(x =>
x.Code).ToList())}";
      }
   }
}
```

CustomerIdentityResultTest.cs

```
using System.Linq;
using Parusnik.OnlineAssistant.Identity.Core;
using Xunit;

namespace Parusnik.OnlineAssistant.Parus.API.Test
{
   public class CustomerIdentityResultTest
   {
      [Fact]
      public void VerifyDefaultConstructor()
      {
         var result = new CustomerIdentityResult();
```

```
            Assert.False(result.Succeeded);
            Assert.Equal(0, result.Errors.Count());
        }

        [Fact]
        public void NullFailedCustomerEmptyErrors()
        {
            var result = CustomerIdentityResult.Failed();

            Assert.False(result.Succeeded);
            Assert.Equal(0, result.Errors.Count());
        }
    }
}
```

IdentityControllerTest.cs

```
using System.Threading.Tasks;
using Microsoft.AspNetCore.Mvc;
using Microsoft.Extensions.Logging;
using Moq;
using Parusnik.OnlineAssistant.Parus.API.Areas.IdentityService.Controllers;
using Parusnik.OnlineAssistant.Parus.API.Areas.IdentityService.Models;
using Parusnik.OnlineAssistant.Parus.API.Areas.IdentityService.Services;
using Xunit;

namespace Parusnik.OnlineAssistant.Parus.API.Test
{
    public class IdentityControllerTest
    {
        [Fact]
        public async Task AuthorizeReturnsNotFoundWhenManagerThrowsExceptions()
        {
            // Arrange
            var customerManager = MockHelpers.MockCustomerManager<Customer>();
            var loggerSignInManager =
MockHelpers.MockLoggerManager<SignInManager<Customer>>();

            var signInManager = new Mock<SignInManager<Customer>>(
                customerManager.Object,
                new IdentityErrorDescriber(),
                loggerSignInManager.Object);

            var loggerIdentityController = new Mock<ILogger<IdentityController>>();
            var controller = new IdentityController(signInManager.Object,
loggerIdentityController.Object);

            // Act
            var result = await controller.Authorize("1234567890", "PR", "0000");

            // Assert
```

```
            Assert.IsType<NotFoundResult>(result);
        }
    }
}
```

InitialSchema.Designer.cs

```
using System;
using Microsoft.EntityFrameworkCore;
using Microsoft.EntityFrameworkCore.Infrastructure;
using Microsoft.EntityFrameworkCore.Metadata;
using Microsoft.EntityFrameworkCore.Migrations;
using Parusnik.OnlineAssistant.Identity.API.Data;
using Parusnik.OnlineAssistant.Identity.Core.Models;

namespace Parusnik.OnlineAssistant.Identity.API.Data.Migrations
{
    [DbContext(typeof(ApplicationDbContext))]
    [Migration("00000000000000_InitialSchema")]
    partial class InitialSchema
    {
        protected override void BuildTargetModel(ModelBuilder modelBuilder)
        {
            modelBuilder
                .HasAnnotation("ProductVersion", "1.1.2")
                .HasAnnotation("SqlServer:ValueGenerationStrategy",
SqlServerValueGenerationStrategy.IdentityColumn);


modelBuilder.Entity("Microsoft.AspNetCore.Identity.EntityFrameworkCore.IdentityRole", b =>
                {
                    b.Property<string>("Id")
                        .ValueGeneratedOnAdd();

                    b.Property<string>("ConcurrencyStamp")
                        .IsConcurrencyToken();

                    b.Property<string>("Name")
                        .HasMaxLength(256);

                    b.Property<string>("NormalizedName")
                        .HasMaxLength(256);

                    b.HasKey("Id");

                    b.HasIndex("NormalizedName")
                        .IsUnique()
                        .HasName("RoleNameIndex");

                    b.ToTable("AspNetRoles");
                });
```

```
modelBuilder.Entity("Microsoft.AspNetCore.Identity.EntityFrameworkCore.IdentityRoleClaim
<string>", b =>
                    {
                        b.Property<int>("Id")
                            .ValueGeneratedOnAdd();

                        b.Property<string>("ClaimType");

                        b.Property<string>("ClaimValue");

                        b.Property<string>("RoleId")
                            .IsRequired();

                        b.HasKey("Id");

                        b.HasIndex("RoleId");

                        b.ToTable("AspNetRoleClaims");
                    });

modelBuilder.Entity("Microsoft.AspNetCore.Identity.EntityFrameworkCore.IdentityUserClaim
<string>", b =>
                    {
                        b.Property<int>("Id")
                            .ValueGeneratedOnAdd();

                        b.Property<string>("ClaimType");

                        b.Property<string>("ClaimValue");

                        b.Property<string>("UserId")
                            .IsRequired();

                        b.HasKey("Id");

                        b.HasIndex("UserId");

                        b.ToTable("AspNetUserClaims");
                    });

modelBuilder.Entity("Microsoft.AspNetCore.Identity.EntityFrameworkCore.IdentityUserLogin<
string>", b =>
                    {
                        b.Property<string>("LoginProvider");

                        b.Property<string>("ProviderKey");

                        b.Property<string>("ProviderDisplayName");
```

```
                    b.Property<string>("UserId")
                        .IsRequired();

                    b.HasKey("LoginProvider", "ProviderKey");

                    b.HasIndex("UserId");

                    b.ToTable("AspNetUserLogins");
                });

modelBuilder.Entity("Microsoft.AspNetCore.Identity.EntityFrameworkCore.IdentityUserRole<s
tring>", b =>
                {
                    b.Property<string>("UserId");

                    b.Property<string>("RoleId");

                    b.HasKey("UserId", "RoleId");

                    b.HasIndex("RoleId");

                    b.ToTable("AspNetUserRoles");
                });

modelBuilder.Entity("Microsoft.AspNetCore.Identity.EntityFrameworkCore.IdentityUserToken
<string>", b =>
                {
                    b.Property<string>("UserId");

                    b.Property<string>("LoginProvider");

                    b.Property<string>("Name");

                    b.Property<string>("Value");

                    b.HasKey("UserId", "LoginProvider", "Name");

                    b.ToTable("AspNetUserTokens");
                });

            modelBuilder.Entity("Parusnik.OnlineAssistant.Identity.Core.Models.User", b =>
                {
                    b.Property<string>("Id")
                        .ValueGeneratedOnAdd();

                    b.Property<int>("AccessFailedCount");

                    b.Property<string>("CompanyInn");

                    b.Property<string>("CompanyName");
```

```
b.Property<string>("CompanyRegistrationNumber");

b.Property<string>("CompanyRegistrationPrefix");

b.Property<string>("ConcurrencyStamp")
    .IsConcurrencyToken();

b.Property<string>("Email")
    .HasMaxLength(256);

b.Property<bool>("EmailConfirmed");

b.Property<string>("FirstName");

b.Property<int>("Gender");

b.Property<string>("LastName");

b.Property<bool>("LockoutEnabled");

b.Property<DateTimeOffset?>("LockoutEnd");

b.Property<string>("NormalizedEmail")
    .HasMaxLength(256);

b.Property<string>("NormalizedUserName")
    .HasMaxLength(256);

b.Property<string>("PasswordHash");

b.Property<string>("PhoneNumber");

b.Property<bool>("PhoneNumberConfirmed");

b.Property<string>("SecurityStamp");

b.Property<bool>("TwoFactorEnabled");

b.Property<string>("UserName")
    .HasMaxLength(256);

b.HasKey("Id");

b.HasIndex("NormalizedEmail")
    .HasName("EmailIndex");

b.HasIndex("NormalizedUserName")
    .IsUnique()
    .HasName("UserNameIndex");

b.ToTable("AspNetUsers");
```

```
                });

modelBuilder.Entity("Microsoft.AspNetCore.Identity.EntityFrameworkCore.IdentityRoleClaim
<string>", b =>
                {

b.HasOne("Microsoft.AspNetCore.Identity.EntityFrameworkCore.IdentityRole")
                        .WithMany("Claims")
                        .HasForeignKey("RoleId")
                        .OnDelete(DeleteBehavior.Cascade);
                });

modelBuilder.Entity("Microsoft.AspNetCore.Identity.EntityFrameworkCore.IdentityUserClaim
<string>", b =>
                {
                    b.HasOne("Parusnik.OnlineAssistant.Identity.Core.Models.User")
                        .WithMany("Claims")
                        .HasForeignKey("UserId")
                        .OnDelete(DeleteBehavior.Cascade);
                });

modelBuilder.Entity("Microsoft.AspNetCore.Identity.EntityFrameworkCore.IdentityUserLogin<
string>", b =>
                {
                    b.HasOne("Parusnik.OnlineAssistant.Identity.Core.Models.User")
                        .WithMany("Logins")
                        .HasForeignKey("UserId")
                        .OnDelete(DeleteBehavior.Cascade);
                });

modelBuilder.Entity("Microsoft.AspNetCore.Identity.EntityFrameworkCore.IdentityUserRole<s
tring>", b =>
                {

b.HasOne("Microsoft.AspNetCore.Identity.EntityFrameworkCore.IdentityRole")
                        .WithMany("Users")
                        .HasForeignKey("RoleId")
                        .OnDelete(DeleteBehavior.Cascade);

                    b.HasOne("Parusnik.OnlineAssistant.Identity.Core.Models.User")
                        .WithMany("Roles")
                        .HasForeignKey("UserId")
                        .OnDelete(DeleteBehavior.Cascade);
                });
            }
        }
    }
```

ApplicationDbContextModelSnapshot.cs

```
using System;
using Microsoft.EntityFrameworkCore;
using Microsoft.EntityFrameworkCore.Infrastructure;
using Microsoft.EntityFrameworkCore.Metadata;
using Microsoft.EntityFrameworkCore.Migrations;
using Parusnik.OnlineAssistant.Identity.API.Data;
using Parusnik.OnlineAssistant.Identity.Core.Models;
namespace Parusnik.OnlineAssistant.Identity.API.Data.Migrations
{
   [DbContext(typeof(ApplicationDbContext))]
   partial class ApplicationDbContextModelSnapshot : ModelSnapshot
   {
      protected override void BuildModel(ModelBuilder modelBuilder)
      {
         modelBuilder
            .HasAnnotation("ProductVersion", "1.1.2")
            .HasAnnotation("SqlServer:ValueGenerationStrategy",
SqlServerValueGenerationStrategy.IdentityColumn);

modelBuilder.Entity("Microsoft.AspNetCore.Identity.EntityFrameworkCore.IdentityRole", b =>
                  {
                     b.Property<string>("Id")
                        .ValueGeneratedOnAdd();
                     b.Property<string>("ConcurrencyStamp")
                        .IsConcurrencyToken();
                     b.Property<string>("Name")
                        .HasMaxLength(256);
                     b.Property<string>("NormalizedName")
                        .HasMaxLength(256);
                     b.HasKey("Id");
                     b.HasIndex("NormalizedName")
                        .IsUnique()
                        .HasName("RoleNameIndex");
                     b.ToTable("AspNetRoles");
                  });

modelBuilder.Entity("Microsoft.AspNetCore.Identity.EntityFrameworkCore.IdentityRoleClaim
<string>", b =>
                  {
                     b.Property<int>("Id")
                        .ValueGeneratedOnAdd();
                     b.Property<string>("ClaimType");
                     b.Property<string>("ClaimValue");
                     b.Property<string>("RoleId")
                        .IsRequired();
                     b.HasKey("Id");
                     b.HasIndex("RoleId");
                     b.ToTable("AspNetRoleClaims");
                  });
```

```
modelBuilder.Entity("Microsoft.AspNetCore.Identity.EntityFrameworkCore.IdentityUserClaim
<string>", b =>
                    {
                        b.Property<int>("Id")
                            .ValueGeneratedOnAdd();
                        b.Property<string>("ClaimType");
                        b.Property<string>("ClaimValue");
                        b.Property<string>("UserId")
                            .IsRequired();
                        b.HasKey("Id");
                        b.HasIndex("UserId");
                        b.ToTable("AspNetUserClaims");
                    });

modelBuilder.Entity("Microsoft.AspNetCore.Identity.EntityFrameworkCore.IdentityUserLogin<
string>", b =>
                    {
                        b.Property<string>("LoginProvider");
                        b.Property<string>("ProviderKey");
                        b.Property<string>("ProviderDisplayName");
                        b.Property<string>("UserId")
                            .IsRequired();
                        b.HasKey("LoginProvider", "ProviderKey");
                        b.HasIndex("UserId");
                        b.ToTable("AspNetUserLogins");
                    });

modelBuilder.Entity("Microsoft.AspNetCore.Identity.EntityFrameworkCore.IdentityUserRole<s
tring>", b =>
                    {
                        b.Property<string>("UserId");
                        b.Property<string>("RoleId");
                        b.HasKey("UserId", "RoleId");
                        b.HasIndex("RoleId");
                        b.ToTable("AspNetUserRoles");
                    });

modelBuilder.Entity("Microsoft.AspNetCore.Identity.EntityFrameworkCore.IdentityUserToken
<string>", b =>
                    {
                        b.Property<string>("UserId");
                        b.Property<string>("LoginProvider");
                        b.Property<string>("Name");
                        b.Property<string>("Value");
                        b.HasKey("UserId", "LoginProvider", "Name");
                        b.ToTable("AspNetUserTokens");
                    });
                modelBuilder.Entity("Parusnik.OnlineAssistant.Identity.Core.Models.User", b =>
                    {
                        b.Property<string>("Id")
                            .ValueGeneratedOnAdd();
```

```
                    b.Property<int>("AccessFailedCount");
                    b.Property<string>("CompanyInn");
                    b.Property<string>("CompanyName");
                    b.Property<string>("CompanyRegistrationNumber");
                    b.Property<string>("CompanyRegistrationPrefix");
                    b.Property<string>("ConcurrencyStamp")
                        .IsConcurrencyToken();
                    b.Property<string>("Email")
                        .HasMaxLength(256);
                    b.Property<bool>("EmailConfirmed");
                    b.Property<string>("FirstName");
                    b.Property<int>("Gender");
                    b.Property<string>("LastName");
                    b.Property<bool>("LockoutEnabled");
                    b.Property<DateTimeOffset?>("LockoutEnd");
                    b.Property<string>("NormalizedEmail")
                        .HasMaxLength(256);
                    b.Property<string>("NormalizedUserName")
                        .HasMaxLength(256);
                    b.Property<string>("PasswordHash");
                    b.Property<string>("PhoneNumber");
                    b.Property<bool>("PhoneNumberConfirmed");
                    b.Property<string>("SecurityStamp");
                    b.Property<bool>("TwoFactorEnabled");
                    b.Property<string>("UserName")
                        .HasMaxLength(256);
                    b.HasKey("Id");
                    b.HasIndex("NormalizedEmail")
                        .HasName("EmailIndex");
                    b.HasIndex("NormalizedUserName")
                        .IsUnique()
                        .HasName("UserNameIndex");
                    b.ToTable("AspNetUsers");
                });

modelBuilder.Entity("Microsoft.AspNetCore.Identity.EntityFrameworkCore.IdentityRoleClaim
<string>", b =>
                {

b.HasOne("Microsoft.AspNetCore.Identity.EntityFrameworkCore.IdentityRole")
                        .WithMany("Claims")
                        .HasForeignKey("RoleId")
                        .OnDelete(DeleteBehavior.Cascade);
                });

modelBuilder.Entity("Microsoft.AspNetCore.Identity.EntityFrameworkCore.IdentityUserClaim
<string>", b =>
                {
                    b.HasOne("Parusnik.OnlineAssistant.Identity.Core.Models.User")
                        .WithMany("Claims")
                        .HasForeignKey("UserId")
                        .OnDelete(DeleteBehavior.Cascade);
```

```
                });

modelBuilder.Entity("Microsoft.AspNetCore.Identity.EntityFrameworkCore.IdentityUserLogin<
string>", b =>
                {
                    b.HasOne("Parusnik.OnlineAssistant.Identity.Core.Models.User")
                        .WithMany("Logins")
                        .HasForeignKey("UserId")
                        .OnDelete(DeleteBehavior.Cascade);
                });

modelBuilder.Entity("Microsoft.AspNetCore.Identity.EntityFrameworkCore.IdentityUserRole<s
tring>", b =>
                {

b.HasOne("Microsoft.AspNetCore.Identity.EntityFrameworkCore.IdentityRole")
                        .WithMany("Users")
                        .HasForeignKey("RoleId")
                        .OnDelete(DeleteBehavior.Cascade);
                    b.HasOne("Parusnik.OnlineAssistant.Identity.Core.Models.User")
                        .WithMany("Roles")
                        .HasForeignKey("UserId")
                        .OnDelete(DeleteBehavior.Cascade);
                });
            }
        }
    }
```