Федеральное государственное автономное
образовательное учреждение
высшего образования
«СИБИРСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»
Институт космических и информационных технологий
институт
Информационных систем
кафедра

УТВЕРЖДАЮ
Заведующий кафедрой ИС
_____ Л.С. Троценко

«13» мая 2018 г.

БАКАЛАВСКАЯ РАБОТА

09.03.02 «Информационные системы и технологии»

Автоматизация внутренних бизнес-процессов компании, оказывающей услуги по обслуживанию оргтехники

Научный руководитель _____ доцент, к.т.н. С.А. Виденин
подпись, дата

Выпускник _____ О.Ю. Анучина
подпись, дата

Нормоконтролер _____ Ю.В. Шмагрис
подпись, дата

Красноярск 2018

Федеральное государственное автономное

образовательное учреждение

высшего образования

«СИБИРСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

ИНСТИТУТ КОСМИЧЕСКИХ И ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ

Кафедра «Информационных систем»

УТВЕРЖДАЮ

Заведующий кафедрой ИС

_____ С.А Виденин

подпись инициалы, фамилия

«02» марта 2018 г.

ЗАДАНИЕ

НА ВЫПУСКНУЮ КВАЛИФИКАЦИОННУЮ РАБОТУ

в форме бакалаврской работы

Студентке _____Анучиной Оксане Юрьевне_____

фамилия, имя, отчество

Группа ЗКИ13-13б Направление (специальность) 09.03.02_____

номер код

_____Информационные системы и технологии_____

наименование

Тема выпускной квалификационной работы: Автоматизация внутренних бизнес-процессов компании, оказывающей услуги по обслуживанию оргтехники _____

Утверждена приказом по университету № 3758/с от __14.03.2018_____

Руководитель ВКР: С.А. Виденин, доцент, заведующий кафедрой Информационные системы

инициалы, фамилия, должность, ученое звание и место работы

Исходные данные для ВКР: Теоретические сведения о способах проектирования информационных систем, разработке программного обеспечения с использованием ASP.NET Core, React, Xamarin.Forms_____

Перечень разделов ВКР: Проектирование ИС, Описание реализации, Программная реализация, Тестовое внедрение_____

Перечень графического материала: Презентация, выполненная в Microsoft: PowerPoint 2013_____

Руководитель ВКР _____ С.А. Виденин

подпись инициалы и фамилия

Задание приняла к исполнению _____ О.Ю. Анучина

подпись, инициалы и фамилия студента

«02» марта 2018 г.

РЕФЕРАТ

Выпускная квалификационная работа по теме «Автоматизация внутренних бизнес-процессов компании, оказывающей услуги по обслуживанию оргтехники» содержит 81 страницу текстового документа, 1 приложение, 19 использованных источников, 23 иллюстраций, 4 таблицы.

ПРОЕКТИРОВАНИЕ ИС, ОПИСАНИЕ РЕАЛИЗАЦИИ, ПРОГРАММНАЯ РЕАЛИЗАЦИЯ, ТЕСТОВОЕ ВНЕДРЕНИЕ, ПРИЛОЖЕНИЕ.

Цель работы – автоматизация внутренних бизнес-процессов компании, оказывающей услуги по обслуживанию оргтехники с помощью внедрения информационной системы для работы инженеров.

Для достижения поставленной цели были сформулированы следующие задачи:

– Изучить способы проектирования информационных систем;

– Выбрать оптимальный способ и его помощью спроектировать информационную систему;

– Выполнить программную реализацию всех компонентов информационной системы;

– Повысить эффективность процесса работы с обращениями клиентов компании ООО «Техномакс-Красноярск», оказывающей услуги по обслуживанию оргтехники.

В ходе работы были изучены способы проектирования информационных систем, определены основные этапы каждого из них, выбран оптимальный способ подхода к проектированию информационных систем, спроектирована информационная система для работы инженеров технической поддержки, выполнена программная реализация компонентов информационной системы, произведено тестовое внедрение информационной системы.

В итоге была разработана и протестирована в опытной эксплуатации информационная система, способная автоматизировать внутренние бизнес-процессы организации, оказывающей услуги по обслуживанию оргтехники, за счет использования мобильного приложения, веб-приложения и веб-сервиса.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Современный мир динамично развивается, постоянно увеличивая темпы, что проявляется не только в различных сферах деятельности, используемых технологиях и методах производства, но и в предъявлении к ним определенных требований. Создаваемые сегодня специалистами информационные системы должны учитывать возможные изменения в различных сферах деятельности, её управления, поддержки. Для того, чтобы повысить эффективность управления, информационная система должна отвечать требованиям окружающего мира, потребностям рынка, поставщиков и заказчиков.

Предметная область, структура данных и возможность интеграции являются наиболее чувствительными к изменениям аспектами, на которые необходимо обратить внимание при проектировании информационных систем.

Общая идеология подхода к проектированию информационных систем основывается на приоритетах и потребностях общества. К приоритетам относятся: повышение уровня абстракции, совместное создание и использование баз знаний, упрощение инструментария для создания информационных систем.

Обозначенные аспекты проектирования, в сопоставлении с общей идеологией создания информационных систем, дают возможность сформулировать общие требования, которым должна удовлетворять информационная система. К данным требованиям относятся: возможность использования информационной системы в различных предметных областях, возможность информационной системы динамически адаптироваться к изменению предметной области и требований пользователей, гибкость информационной системы к работе со слабоструктурированными данными и обеспечению внутренней и внешней интеграции.

Цель работы – автоматизация внутренних бизнес-процессов компании, оказывающей услуги по обслуживанию оргтехники с помощью внедрения информационной системы для работы инженеров.

Для выполнения поставленной цели работы необходимо выполнить следующие задачи:

– Изучить способы проектирования информационных систем;

– Выбрать оптимальный способ и его помощью спроектировать информационную систему;

– Выполнить программную реализацию всех компонентов информационной системы;

– Повысить эффективность процесса работы с обращениями клиентов компании ООО «Техномакс-Красноярск», оказывающей услуги по обслуживанию оргтехники.

1 Проектирование ИС

В последнее время отчетливо проявляется тенденция перехода от детального управления внутренней деятельностью к управлению заказчиками и поставщиками. Конкурентоспособность компании все больше зависит от способности создавать и углублять взаимоотношения с другими компаниями (партнерами, конкурентами, заказчиками или поставщиками). Причинами этого являются:

– расширение экономического пространства, на котором функционируют предприятия;

– появление нового стратегического ресурса - информации;

– необходимость учитывать фактор времени.

Для решения поставленных задач необходимо иметь эффективную систему управления предприятием, включающую систему менеджмента качества, и информационную систему их поддержки.

1.1 Подходы к проектированию ИС

Можно выделить два основных подхода к проектированию систем управления предприятием и информационных систем их поддержки, а именно, структурный и процессный [1].

Первый подход основан на использовании организационной структуры компании, когда проектирование системы идет по структурным подразделениям. Технологии деятельности в этом случае описываются через технологии работы структурных подразделений, а взаимодействие структурных подразделений - через модель верхнего уровня. Если компания представляет собой сложную структуру типа холдинг или предприятие-сеть, то необходимо также иметь модель взаимодействия всех входящих в него элементов, в которой будут отражены не только технологические моменты, но также финансовые и юридические.

Главным недостатком структурного подхода является привязка к организационной структуре, которая очень быстро меняется, поэтому приходится часто вносить изменения в Системный проект информационной системы. Хорошо, если на предприятии есть обученные специалисты, способные быстро и качественно актуализировать информационную систему. Как правило, это достаточно трудоемкий, длительный и утомительный процесс.

Несколько по-иному обстоит дело при процессном подходе. Этот подход ориентирован не на организационную структуру, а на бизнес-процессы. Он представляется наиболее перспективным. Бизнес-процессы, в отличие от организационной структуры, меняются реже. Как правило, основных бизнес-процессов на предприятии немного, обычно не более десяти.

Процессный подход подводит к необходимости перехода на так называемое "тощее производство" или "тощую" ресурсосберегающую организационную структуру. Основными чертами такой реорганизации являются:

– широкое делегирование полномочий и ответственности исполнителям;

– сокращение количества уровней принятия решения;

– сочетание принципа целевого управления с групповой организацией труда;

– автоматизация технологий выполнения бизнес-процессов.

В качестве основного при реализации проекта был выбран процессный подход при проектировании информационной системы.

1.2 Описание решаемой проблемы

Разработанная информационная система должна решить проблему недостаточно оперативного обмена информацией между всеми сотрудниками компании, которые задействованы в процессе решения заявки клиента,

позволить оптимально организовать рабочий процесс всех отделов и сэкономить время. Использование мобильного приложения позволит инженерам добавлять комментарии к заявкам находясь на территории клиента, что положительно отразится на возможности отслеживания состояние обращения и повысит эргономичность его работы.

Заказчик услуг сможет запрашивать у исполнителя реальные прогнозы по срокам исполнения заявки за счет наличия у менеджера деталей исполнения предыдущих заявок и ориентируясь на специфику текущего запроса, отраженных в информационной системе. Наличие отчетности инженеров позволяет описать во внутренней базе знаний способы решения часто возникающих проблем в разрезе клиентов. Исключается возможность ошибки в изменении статуса инцидента и его потеря.

Анализ времени исполнения инцидентов сможет помочь при принятии решения о завершении испытательного срока у новых инженеров и получить объективную картину занятости специалиста при работе за пределами офиса.

1.3 Обоснование выбранного подхода

Технологии работы определяются характером деятельности Компании и существуют независимо от её организационно-штатной структуры. Каждая технология работы состоит из последовательно и параллельно выполняющихся функций, которые различным образом могут конфигурироваться под любого исполнителя и должностного лица, посредством механизма полномочий. Полномочия регулируют доступ пользователей к прикладным задачам, информации базы данных и обобщённым данным, характеризующим состояния процессов деятельности Компании.

Прикладная задача представляет собой программный модуль, поддерживающий некий логически законченный процесс, направленный на учёт данных и облегчение пользователям выполнение своих обязанностей в

рамках технологии работы. Каждая технология работы Компании включает несколько прикладных задач, логически увязанных между собой путём использования в следующей прикладной задаче результатов предыдущих, хранящихся в корпоративной базе данных.

Состав прикладных задач формируется на основе рациональных технологий деятельности Компании, которые представлены в функционально-информационных моделях. Функционально-информационные модели представляют собой иерархическую структуру функций, связанных между собой с помощью интерфейсных дуг.

Моделирование выполняется по принципу "сверху-вниз". Первые диаграммы модели описывают наиболее общие функции, которые постепенно декомпозируются вплоть до самого "нижнего" уровня "элементарных" функций.

В качестве примера приведем прикладные задачи по базовой технологии обслуживания оргтехники. К таким задачам относятся следующие:

– учёт клиентов компании (поддержание справочника клиентов в актуальном состоянии);

– формирование, учёт и контроль заявок на обслуживание оргтехники;

– поддержка процесса согласования с клиентом приемки выполненных работ;

– формирование перечня требуемого оборудования/комплектующих;

– контроль платежей клиентов компании при формировании заявки;

– поддержка статистики однотипных обращений;

– подготовка материалов для формирования единой базы знания инженеров;

– поддержка интеграции с учетными системами компании.

В сравнении с имеющимся, новое решение позволит обеспечить более удобный режим доступа заинтересованных лиц к информации, повысить быстродействие, обеспечить надёжное хранение данных и более полный охват

функций, подлежащих автоматизации, а также предоставит основу для простого расширения с целью решения задач по учёту расчётов с клиентами, учету расходов материалов\требуемого оборудования, и другим задач, которые поставит бизнес-заказчик. Позволит организовать географически распределенную вовлеченность инженеров в процесс решения инцидента пользователя.

В ходе проведения анализа бизнес-процессов были выявлены следующие проблемы:

– недостаточно оперативный обмен информацией между обращающимся за помощью заявителем и всеми взаимодействующими с ним лицами;

– отсутствие у сотрудников компании возможности отслеживать состояние обращения;

– отсутствие возможности автоматизированного формирования отчета о работе инженера за период.

Решение приведенных проблем позволит сократить задержки при выполнении заявок, исключить ошибки планирования визитов, повысить эргономичность работы инженера, вести электронный журнал действий инженеров, оптимально распределить время занятости группы инженеров компании.

2 Описание реализации

2.1 Видение способа реализации

Средствами создания и обработки заявок в компании, обслуживающей оргтехнику, являются, входящие в состав информационной системы, мобильное приложение и веб-приложение, посредством работы с веб-сервисом. Обработка заявок осуществляется инженерами компании. Прием заявок производится сотрудником компании в телефонном режиме. Выделены следующие основные роли пользователей системы:

– «инженер» – технический специалист, осуществляющий работу через мобильное приложение и выполняющий работу на территории заказчика. Наделен правами на изменение статусов наряда по конкретному инциденту. Ответственен за выполнение нарядов в срок, отражение результатов своей работы, формирование запросов на инструкции/комплектующие;

– «менеджер» – специалист, сопровождающий работу информационной системы и ответственный за принятие решения по заявкам согласно своей области ответственности. Наделен правами на администрирование учётных записей внутри организации, приём сотрудников и\или изменение их роли внутри организации, построение отчетов по выполнению инцидентов. Ответственен за управление сотрудниками и их ролями внутри организации;

– «администратор» – технический специалист, сопровождающий программную и аппаратную части информационной системы;

– «специалист технической поддержки» - технический специалист, выполняющий обработку удаленных обращений и выполняющий распределение заявок инженерам. Наделен правами на изменение состояния инцидента и наряда, выставления нарядов на инженеров. Ответственен за обработку поступающих инцидентов, предлагает решение. Осуществляет контроль сроков выполнения нарядов, выставленных на инженеров;

– «специалист call-центра» - сотрудник, принимающий заявки от клиентов в телефонном режиме и регистрирующий их в информационной системе.

Далее перечислены типовые задачи сотрудников. Работник технической поддержки - удаленно обрабатывает инциденты от пользователей, если требуется выезд, создает наряд инженеру, контролирует исполнение наряда. Инженер – просматривает имеющиеся в системе наряды, планирует свой день, выполняет обращения по 1 за раз. Менеджер организации - управляет учётными записями в рамках своей организации, контролирует права доступа и роли УЗ внутри своей организации, несет ответственность за исполнение инцидентов. Сотрудник колл-центра - регистрирует инциденты в системе через веб-сайт.

Ключевыми потребностями пользователей являются:

– Специалист технической поддержки не всегда имеет возможность подключиться удаленно к заявителю инцидента для оказания помощи, ему требуется направить инженера на территорию заявителя. Принимать обращения в телефонном режиме он не должен, чтобы не тратить время. Вести контроль за выполнением наряда удобно в случае, если инженер оперативно меняет статус через мобильное приложение;

– Инженер затрачивает большое количество времени на получение обращений, и планирование своего дня;

– Менеджер не может оперативно оценивать состояние рабочего времени инженеров и работников технической поддержки, не может контролировать доступность инженерам нарядов, не имеет возможности оперативного изменения статуса сотрудников;

– Компания, обслуживающая оргтехнику, нуждается в системе, которая бы ускорила и оптимизировала вышеуказанные процессы.

Информационная система имеет клиент-серверную архитектуру, представленную на рисунке 1.

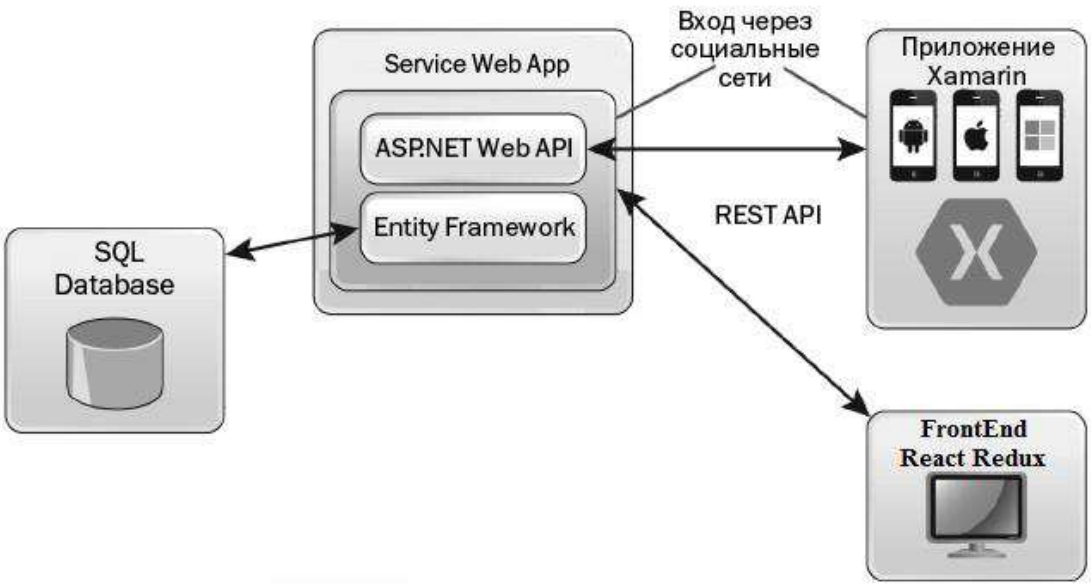

Рисунок 1 –Архитектура информационной системы

Клиентская часть представлена мобильным приложениям, реализованное с помощью фреймворка для кроссплатформенной разработки мобильных приложений Xamarin, для различных операционных систем (Android, iOS, Windows Phone) и веб-приложением. Необходимость разработки мобильного приложения объясняется расширенным функционалом, по сравнению с мобильной версией веб-приложения, а именно:

– возможность сжимать информацию (фото- и видео-контент) на клиентском устройстве перед отправкой на сервер с целью экономии трафика мобильного Интернета;

– возможность создавать и сохранять новые заявки на устройстве при отсутствии доступа в Интернет, а при получении доступа к сети отправлять данные заявки на сервер;

– возможность делиться текущим местоположением устройства на карте.

Серверная часть информационной системы представлена сервером базы данных (MSSQL), необходимым для хранения и управления данными

информационной системы и веб-сервисом, отвечающим за функционирование веб-сайта (React Redux) и взаимодействие с мобильным приложением.

Средствами реализации информационной системы являются современные инструменты и среды разработки. Веб-приложение представляет собой Frontend Single Page Application (SPA) с использованием библиотек React, Redux. Веб-сервис является программным интерфейсом приложения, реализованным на API MVC ASP.Net Core 2.0. Мобильное приложение разрабатывается с использованием фреймворка Xamarin, позволяющего разрабатывать мобильные приложения на языке C# для мобильных платформ iOS, Android, Windows Phone.

Анализ разработанных рациональных функционально-информационных моделей технологий работ Компании позволяет систематизировать и сформировать группы документов по типовым алгоритмам "прохождения" между должностными лицами и исполнителями Компании.

В процессе выполненного объединения потоков входных и выходных документов сформированы две типовые группы документов:

– документы поддержки бизнес-процессов;

– отчётные документы.

Операции над документами, выполняемые должностными лицами и исполнителями, описываются в функционально информационных моделях рациональных технологий деятельности Компании, функциональных и информационных спецификациях автоматизированных рабочих мест и процедурах документооборота.

В сравнении с имеющимся, новое решение позволит обеспечить более удобный режим доступа заинтересованных лиц к информации, повысить быстродействие, обеспечить надёжное хранение данных и более полный охват функций, подлежащих автоматизации. Позволит организовать географически распределенную вовлеченность инженеров в процесс решения инцидента пользователя.

Система является законченной независимой разработкой. В перспективе возможно использование системы в комплексе с системами автоматизации других отделов. Коммуникации – на уровне доступа к общей базе данных через сервис обмена данными (Web API).

Сводка возможностей информационной системы приведена в таблице 1.

Таблица 1 – Возможности информационной системы

Выгода	**Поддерживающие возможности**
Упрощение работы работника технической поддержки	Выполнение инцидентов из общего пула; управление жизненным циклом нарядов (приёмка успешно выполненных, возврат обращения в коллекцию активных нарядов в случае неуспешного выполнения инженером); обратная связь с заявителями и инженерами.
Ускорение обращения информации	Система позволит ускорить процесс получения необходимой информации о времени выполнения инцидента работником технической поддержки и инженером, также оптимизирует взаимодействие работника технической поддержки и менеджера организации.
Формирование единой базы для планирования и последующего анализа	Все заинтересованные пользователи со своих рабочих мест имеют доступ к оперативной информации о инцидентах, нарядах и инструкциях.
Возможность индивидуального подхода к каждому инциденту	Система позволяет классифицировать инциденты по категориям (подключение рабочего места, устранение сбоя ПО, обслуживание оргтехники и т.д.)
Отказ от излишних коммуникаций	Система позволяет пользователям получать нужную им информацию самостоятельно, не отвлекая от работы других участников процесса

Место использования системы предполагается: серверная часть и веб-интерфейс располагаются на серверах компании-разработчика (хостинг-

провайдера Azure). Мобильная часть приложения установлена на устройства географически распределённых пользователей.

Информационная система поддерживает интеграцию с учетной системой компании и прочими системами, автоматизирующими смежные участки деятельности, за счет разработки дополнительных методов обмена данными в веб-сервисе.

2.2 Спецификация требований

Список действий пользователей при работе в информационной системе приведен в таблице 2.

Таблица 2 – Список действий пользователей при работе в информационной системе

Действующее лицо	Действие	Последовательность действий
Инженер, Специалист технической поддержки, Менеджер, Администратор, Специалист call-центра	Регистрация	1. Сотрудник выбирает пункт "зарегистрироваться";
		2. Система отображает интерфейс регистрации, в котором отображается список социальных сетей, поддерживаемых приложением с целью регистрации;
		3. Сотрудник выбирает интересующую его социальную сеть, выполняет авторизацию в форме входа в соцсеть, после чего возвращается в приложение;
		4. Система запрашивает согласие на импорт данных из соцсети в личный профиль;
		5. Система отображает импортированные данные в виде формы с полями для ввода, предоставляя возможность изменения данных;
		6. Сотрудник нажимает кнопку "отправить регистрацию", после чего профиль и учётные данные отправляются на центральный сервер системы;
		7. Система регистрирует событие регистрации в журнале безопасности;
		8. Система отображает сообщение об успешной регистрации и регистрирует событие регистрации нового инженера в журнале безопасности.

Продолжение Таблицы 1

Инженер, Специалист технической поддержки, Менеджер, Администратор, Специалист call-центра	Вход в систему	1. Пользователь выбирает пункт "вход через социальную сеть";
		2. Система отображает окно с поддерживаемыми социальными сетями;
		3. Пользователь выбирает интересующую его социальную сеть;
		4. Система отображает в отдельном окне интерфейс входа с использованием социальной сети;
		5. После корректного входа в социальную сеть происходит переход на главную страницу;
		6. Система регистрирует событие входа в систему в журнале безопасности.
Менеджер, Администратор, Специалист call-центра	Просмотр списка клиентов и поиск по имени/адресу	1. Пользователь выбирает пункт "клиенты";
		2. Система отображает таблицу со списком клиентов, содержащую основные данные в столбцах и форму для поиска;
		3. Пользователь просматривает список клиентов в постраничном режиме;
		4. Пользователь заполняет поля для поиска (любое поле из таблицы) и нажимает на кнопку "поиск";
		5. Система изменяет содержимое таблицы со списком клиентов, оставив в нем только тех, которые соответствуют критериям поиска.
Инженер, Специалист технической поддержки, Менеджер, Администратор, Специалист call-центра	Просмотр информации по клиенту	1. Последовательность действий просмотра списка клиентов;
		5. Пользователь выбирает в списке клиентов необходимого и нажимает на него;
		6. Система отображает форму с информацией по клиенту в режиме просмотра. На форме есть кнопка для перехода в режим редактирования;
		7. Система регистрирует в журнале событие просмотра информации по клиенту.
Менеджер, Администратор, Специалист call-центра	Прием инцидента от клиента	1. Пользователь выбирает пункт "создать заявку";
		2. Система отображает таблицу со списком клиентов, зарегистрированных в системе. Пользователь находит необходимого клиента и выбирает его;
		3. Система отображает форму для заполнения сути обращения, контактной информации, вложений (если они имеются);
		4. Пользователь нажимает на кнопку отправки формы;
		5. Система создает новую заявку, регистрирует событие в журнале безопасности и назначает исполнителя;

Продолжение Таблицы 1

Инженер, Специалист технической поддержки, Менеджер, Администратор, Специалист call-центра	Просмотр конкретного инцидента	1. Последовательность действий просмотра списка инцидентов;
		2. Пользователь выбирает в списке обращений необходимое и нажимает на него;
		3. Система отображает форму с информацией по обращению в режиме просмотра. На форме есть кнопки для изменения состояния и передачи обращения;
		4. Система регистрирует в журнале безопасности событие просмотра обращения.
Инженер, Специалист технической поддержки, Менеджер, Администратор, Специалист call-центра	Передача инцидента	1. Последовательность действий просмотра инцидента;
		2. Пользователь выбирает из списка (инженеры организации/кого требуется назначить исполнителем по обращению и нажимает на кнопку "сохранить";
		3. Система сохраняет изменения в текущем обращении и регистрирует в журнале безопасности событие изменения ответственного исполнителя по обращению.
Инженер, Специалист технической поддержки, Менеджер, Администратор, Специалист call-центра	"Приёмка" переданного ранее инцидента	1.Последовательность действий просмотра инцидента;
		2. Пользователь связывается с заявителем по телефону или эл. почте и выясняет как выполнены работы;
		3. Если подтверждение получено, работник переводит обращение в состояние "выполнено", выбрав состояние из списка и нажав на кнопку "сохранить";
		4. Если работы выполнены не полностью, работник повышает приоритет обращения, выбрав в списке приоритет и сохраняет с помощью кнопки "сохранить";
		5. Система регистрирует в журнале безопасности событие изменения состояния обращения.
Специалист технической поддержки, Менеджер, Администратор, Специалист call-центра	Просмотр списка инженеров	1. Пользователь выбирает пункт "пользователи";
		2. Система отображает таблицу со списком инженеров (ФИО, состояние трудоустройства, дата трудоустройства, краткий список компетенций, штатный работник/фрилансер) и полем для поиска инженера;
		3. Пользователь просматривает список инженеров в постраничном режиме;
		4. Пользователь заполняет поля для поиска (любое поле из таблицы) и нажимает на кнопку "поиск";
		5.Система изменяет содержимое таблицы со списком инженеров, оставив в нем только записи, удовлетворяющие выбранным критериям поиска.

Продолжение Таблицы 1

Специалист технической поддержки, Менеджер, Администратор, Специалист call-центра	Просмотр конкретного инженера	1. Последовательность действий просмотра списка инженеров;
		2. Пользователь выбирает в списке инженеров необходимого и нажимает на него;
		3. Система отображает форму с информацией по инженеру в режиме просмотра. На форме есть кнопка для перехода в режим редактирования;
		4. Система регистрирует в журнале безопасности событие просмотра информации по инженеру.
Инженер, Специалист технической поддержки, Менеджер, Администратор, Специалист call-центра	Составление отчёта по инцидентам за период	Аналогично просмотру списка инцидентов с фильтрацией.
Менеджер, Администратор	Изменение информации об клиенте	1. Последовательность действий просмотра информации по клиенту;
		2. Пользователь переходит в режим редактирования и вносит изменения;
		3. Система проверяет заполнены ли все обязательные поля и возможно ли сохранить внесенные изменения (полностью заменить одну организацию на другую запрещено);
		4. Система сохраняет изменения и отображает страницу с результатами;
		5. Система регистрирует событие изменения информации об организации.
Администратор	Изменение роли сотрудника	1. Последовательность просмотра пользователя;
		2. Система открывает форму просмотра сотрудника с полями, доступными для изменения (ФИО, состояние трудоустройства, контактная информация, дата трудоустройства, краткий список компетенций, текущая роль внутри организации);
		3. Пользователь изменяет требуемые поля на форме и нажимает на кнопку "сохранить";
		4. Система проверяет заполнение обязательных полей и сохраняет измененного сотрудника;
		5. Система отправляет на адрес электронной почты сотрудника письмо с информацией об изменении его статуса внутри организации;
		6. Система регистрирует в журнале безопасности событие изменения статуса сотрудника.

Окончание Таблицы 1

Администратор	Блокировка доступа сотруднику	1. Последовательность действий просмотра пользователя;
		2. Пользователь нажимает на кнопку "блокировка";
		3. Система изменяет статус занятости сотрудника и отправляет на адрес его электронной почты письмо с информацией о блокировке его учетной записи;
		4. Система регистрирует в журнале безопасности событие блокировки сотрудника.
Инженер, Специалист технической поддержки	Принятие инцидента в работу	1. Последовательность действий просмотра инцидента;
		2. Пользователь нажимает на кнопку "принять";
		3. Система регистрирует событие просмотра и изменение статуса инцидента в журнале безопасности сервиса.
Инженер, Специалист технической поддержки	Завершение работы по инциденту	1. Последовательность действий просмотра инцидента;
		2. Пользователь заполняет текстовое поле с решением нажимает на кнопку "завершить";
		3. Система регистрирует событие просмотра и изменение статуса инцидента в журнале безопасности сервиса.
Администратор	Просмотр активности любого пользователя сервиса	1. Пользователь выбирает пункт "Журнал безопасности";
		2. Система выводит таблицу со списком записей журнала безопасности, содержащую основные данные в столбцах и форму для поиска;
		3. Пользователь просматривает список записей в постраничном режиме;
		4. Пользователь заполняет поля для поиска (любое поле из таблицы) и нажимает на кнопку "поиск";
		5. Система изменяет таблицу, оставив в ней только записи, удовлетворяющие указанным критериям поиска;
		6. Пользователь выбирает в списке записей необходимую и нажимает на неё;
		7. Система отображает форму для просмотра данных по записи;
		8. Система регистрирует событие просмотра журнала безопасности в журнале безопасности.

2.3 Описание компонентов системы

2.3.1 ASP.NET Core

ASP.NET Core является кроссплатформенной, высокопроизводительной средой с открытым исходным кодом для создания современных облачных приложений, подключенных к Интернету. ASP.NET Core позволяет выполнять следующие задачи:

– Создавать веб-приложения и службы, приложения IoT и серверные части для мобильных приложений;

– Использовать избранные средства разработки в Windows, macOS и Linux;

– Выполнять развертывания в облаке или локальной среде;

– Работать в .NET Core или .NET Framework.

Миллионы разработчиков использовали и продолжают использовать ASP.NET 4.x для создания веб-приложений. ASP.NET Core — это модификация ASP.NET 4.x с архитектурными изменениями, формирующими более рациональную и более модульную платформу.

ASP.NET Core предоставляет следующие преимущества:

– Единое решение для создания пользовательского веб-интерфейса и веб-API;

– Интеграция современных клиентских платформ и рабочих процессов разработки;

– Облачная система конфигурации на основе среды;

– Встроенное введение зависимостей;

– Упрощенный высокопроизводительный модульный конвейер HTTP-запросов;

– Возможность размещения в IIS, Nginx, Apache, Docker или в собственном процессе;

– Параллельное управление версиями приложения, ориентированное на .NET Core;

– Инструментарий, упрощающий процесс современной веб-разработки;

– Возможность сборки и запуска в ОС Windows, macOS и Linux;

– Открытый исходный код и ориентация на сообщество.

ASP.NET Core поставляется полностью в виде пакетов NuGet. Использование пакетов NuGet позволяет оптимизировать приложения для включения только необходимых зависимостей. На самом деле для приложений ASP.NET Core 2.x, ориентированных на .NET Core, требуется только один пакет NuGet. За счет небольшого размера контактной зоны приложения доступны такие преимущества, как более высокий уровень безопасности, минимальное обслуживание и улучшенная производительность.

Приложения ASP.NET Core могут выполняться в .NET Core или .NET Framework. Приложения ASP.NET Core, предназначенные для .NET Framework, не являются кроссплатформенными — они выполняются только в Windows. Отказ от поддержки .NET Framework в ASP.NET Core не планируется. Как правило, ASP.NET Core состоит из библиотек .NET Standard. Приложения, написанные для .NET Standard 2.0 запускаются везде, где есть поддержка .NET Standard 2.0 [7].

При использовании .NET Core существуют некоторые преимущества, и их число увеличивается с каждым выпуском. Преимущества .NET Core по сравнению с .NET Framework включают:

– Кроссплатформенность. Выполняется на macOS, Linux и Windows;

– Повышение производительности;

– Управление параллельными версиями;

– Новые интерфейсы API;

– Открыть исходный код.

На рисунке 2 приведена схема архитектуры приложения ASP.NET Core.

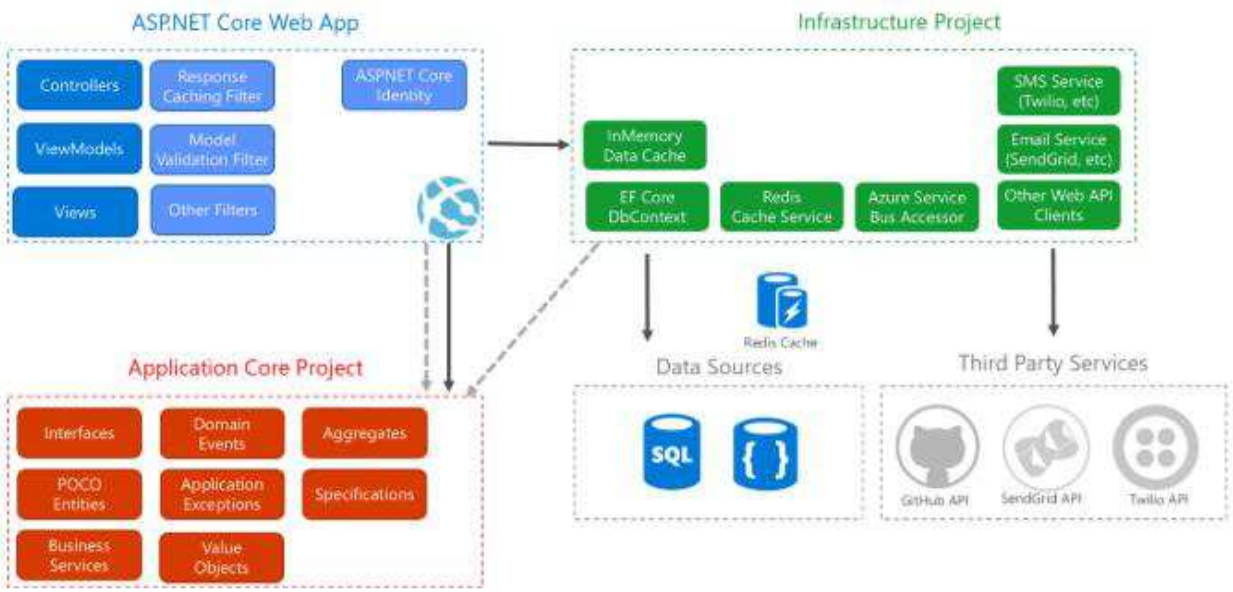

Рисунок 2 - Схема архитектуры приложения ASP.NET Core

2.3.2 Entity Framework

Платформа Entity Framework представляет собой набор технологий ADO.NET, обеспечивающих разработку приложений, связанных с обработкой данных. Архитекторам и разработчикам приложений, ориентированных на обработку данных, приходится учитывать необходимость достижения двух совершенно различных целей. Они должны моделировать сущности, связи и логику решаемых бизнес-задач, а также работать с ядрами СУБД, используемыми для сохранения и получения данных [2]. Данные могут распределяться по нескольким системам хранения данных, в каждой из которых применяются свои протоколы, но даже в приложениях, работающих с одной системой хранения данных, необходимо поддерживать баланс между требованиями системы хранения данных и требованиями написания эффективного и удобного для обслуживания кода приложения.

В Entity Framework разработчики получают возможность работать с данными, представленными в форме относящихся к конкретным доменам объектов и свойств, таких как клиенты и их адреса, не будучи вынужденными обращаться к базовым таблицам и столбцам базы данных, где хранятся эти

данные. Entity Framework дает разработчикам возможность работать с данными на более высоком уровне абстракции, создавать и сопровождать приложения, ориентированные на работу с данными, одновременно с этим сокращая объем кода, по сравнению с традиционными приложениями.

Entity Framework придает значимость моделям, позволяя разработчикам выполнять запросы к сущностям и связям в модели домена (называется концептуальной моделью в Entity Framework), опираясь на Entity Framework для их преобразования операции команды, определяемые источником данных. Это позволяет отказаться от применения в приложениях жестко заданных зависимостей от конкретного источника данных.

При работе в режиме Code First концептуальная модель сопоставлена с режимом хранения в коде. Entity Framework может вывести концептуальную модель, основанную на типах объектов и дополнительных конфигурациях, которые можно задать. Метаданные сопоставления формируются во время выполнения на основе сочетания определений типов домена и дополнительной информации о конфигурации, которая указана в коде. Entity Framework при необходимости создает базу данных на основе метаданных.

EF Core предоставляет универсальный API для работы с данными. И если, к примеру, мы решим сменить целевую СУБД, то основные изменения в проекте будут касаться прежде всего конфигурации и настройки подключения к соответствующим провайдерам. А код, который непосредственно работает с данными, получает данные, добавляет их в БД и т.д., останется прежним.

Entity Framework Core многое унаследовал от своих предшественников, в частности, Entity Framework 6. В тоже время надо понимать, что EF Core - это не новая версия по отношению к EF 6, а совершенно иная технология, хотя в целом принципы работы у них будут совпадать. Поэтому в рамках EF Core используется своя система версий. Текущая версия - 2.0 была выпущена в августе 2017 года. И технология продолжает развиваться.

Как технология доступа к данным Entity Framework Core может использоваться на различных платформах стека .NET. Это и стандартные

платформы типа Windows Forms, консольные приложения, WPF, ASP.NET 4.6/4.7. Это и новые технологии как UWP и ASP.NET Core. При этом кроссплатформенная природа EF Core позволяет задействовать ее не только на ОС Windows, но и на Linux и Mac OS X.

На рисунке 3 показана архитектура доступа к данным Entity Framework.

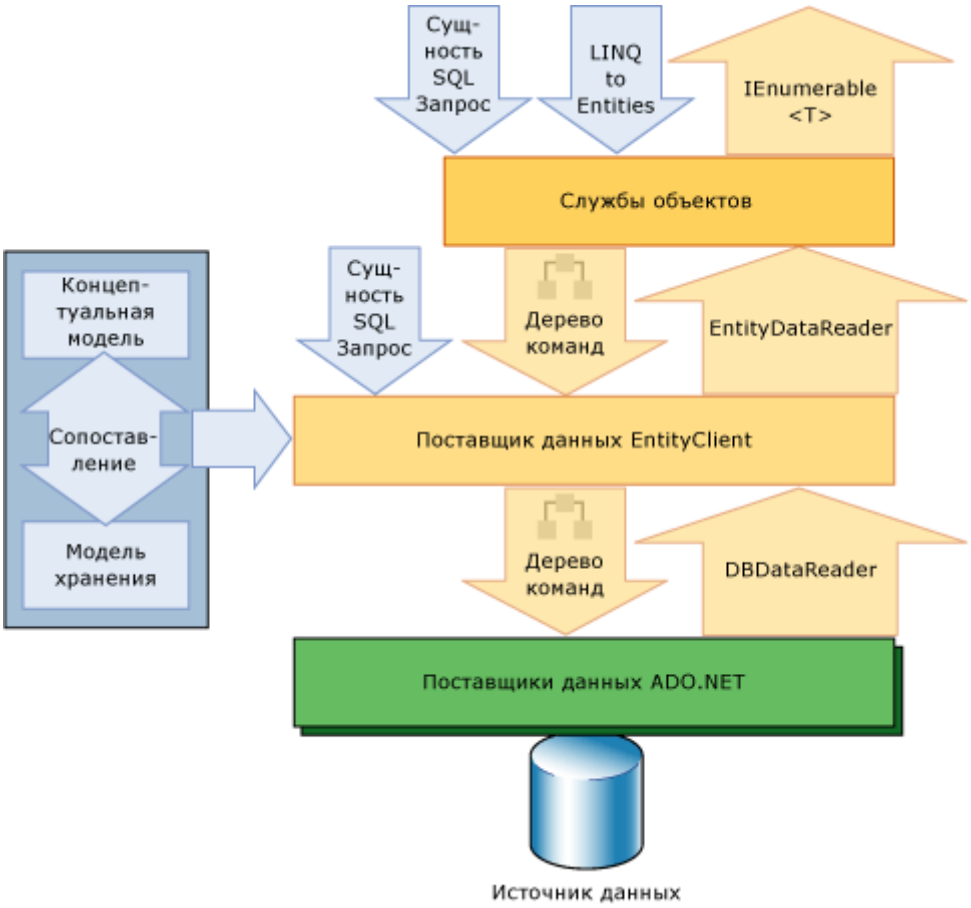

Рисунок 3 – Архитектура доступа к данным Entity Framework

2.3.3 React Redux

Single Page Application – сокращенно SPA, в переводе на русский язык означает «Приложение одной страницы». Другими словами, SPA – это web-приложение, размещенное на одной web-странице, которая для обеспечения работы загружает весь необходимый код вместе с загрузкой самой страницы. Приложение такого типа появились сравнительно недавно, с началом эры HTML5 и SPA является типичным представителем приложений на HTML5.

Как мы знаем, HTML5 это нечто иное как HTML + CSS3 + JavaScript + [несколько новых тегов]. Таким образом, SPA - это приложения написанные на языке JavaScript [5]. И, следовательно, немного перефразировав предыдущие определение получаем: «SPA – это web-приложение, размещенное на одной странице, которая для обеспечения работы загружает все javascript-файлы (модули, виджиты, контролы и т.д.), а также файлы CSS вместе с загрузкой самой страницы».

React (иногда React.js или ReactJS) — JavaScript-фреймворк с открытым исходным кодом для разработки пользовательских интерфейсов.

React разрабатывается и поддерживается Facebook, Instagram и сообществом отдельных разработчиков и корпораций.

React может использоваться для разработки одностраничных и мобильных приложений. Его цель — предоставить высокую скорость, простоту и масштабируемость. В качестве библиотеки для разработки пользовательских интерфейсов, React часто используется с другими библиотеками, такими как Redux.

Сегодня Redux — это одно из наиболее интересных явлений мира JavaScript. Он выделяется из сотни библиотек и фреймворков тем, что грамотно решает множество разных вопросов путем введения простой и предсказуемой модели состояний, уклоне на функциональное программирование и неизменяемые данные, предоставления компактного API.

На рисунке 4 приведена схема приложения React с использованием Redux.

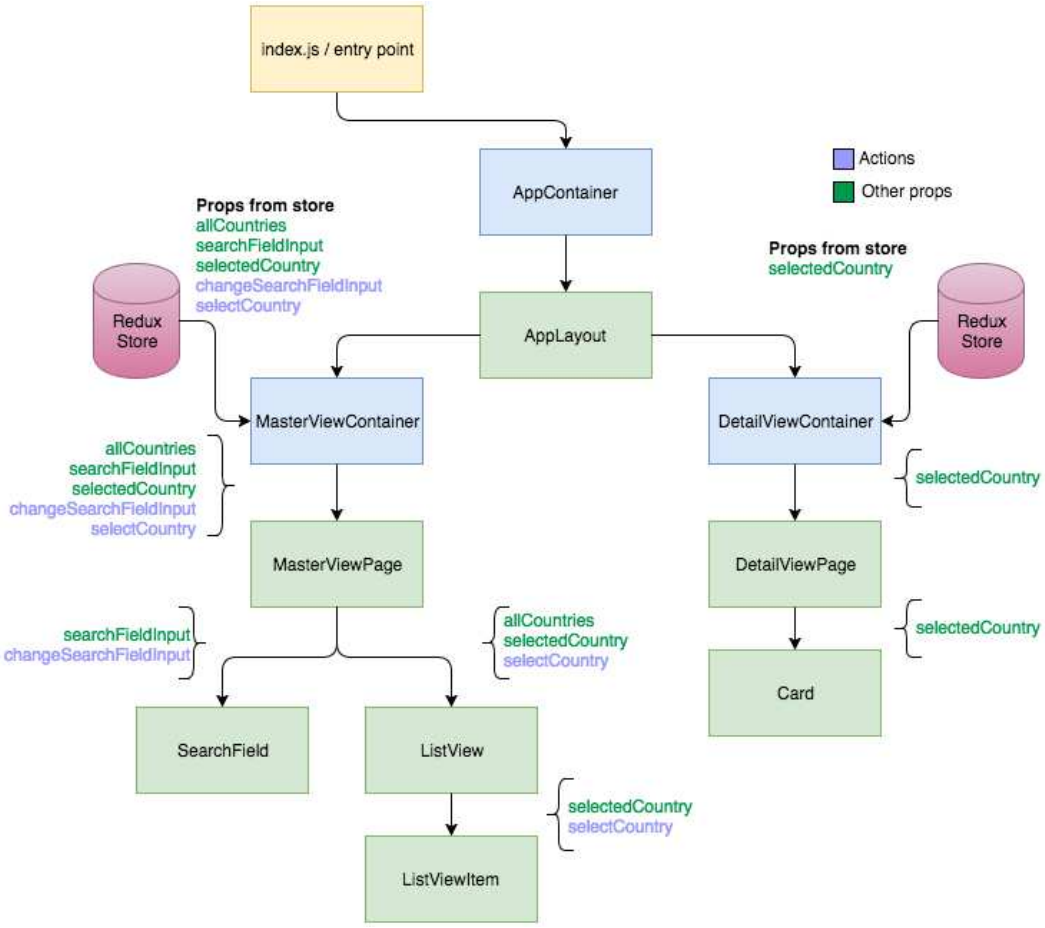

Рисунок 4 – Схема приложения React с использованием Redux

2.3.4 Xamarin

Xamarin — это фреймворк для кроссплатформенной разработки мобильных приложений (iOS, Android, Windows Phone) с использованием языка C#. Позволяет вести разработку программного кода на своем любимом языке, с применением всех его возможностей. При этом вы имеете полный доступ ко всем возможностям SDK платформы и родному механизму создания UI, получая на выходе приложение, которое, строго говоря, ничем не отличается от нативных и (по крайней мере по заверениям) не уступает им в производительности [6].

Фреймворк состоит из нескольких основных частей:

– Xamarin.IOS — библиотека классов для C#, предоставляющая разработчику доступ к iOS SDK;

– Xamarin.Android — библиотека классов для C#, предоставляющая разработчику доступ к Android SDK;

– Компиляторы для iOS и Android;

– IDE Xamarin Studio;

– Плагин для Visual Studio.

На рисунке 5 приведена схема архитектуры приложения Xamarin.

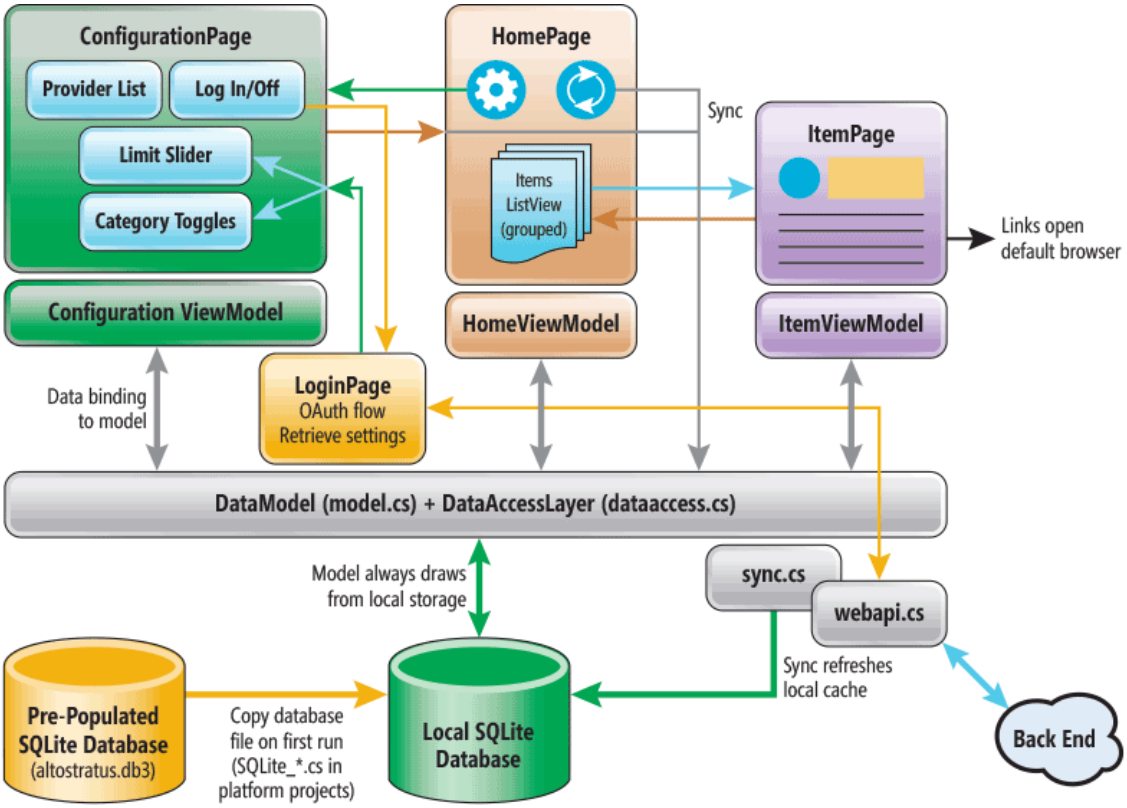

Рисунок 5 – Архитектура приложения Xamarin

3 Программная реализация

3.1 Веб-сервис

Веб-сервис и FrontEnd-точка входа в веб-приложение скомпонованы в одном решении и состоят из шести проектов MVC ASP.Net Core 2.0. Настроенная область видимости проектов в решении позволяет реализовать разделение на уровни.

Проект DAL (Data Access Layer) содержит интерфейсы асинхронных команд и запросов согласно принципу Command-Query Responsibility Segregation, для улучшения абстракции и ослабления связанности компонентов системы [3]. На рисунке 6 приведен список интерфейсов, содержащихся в проекте.

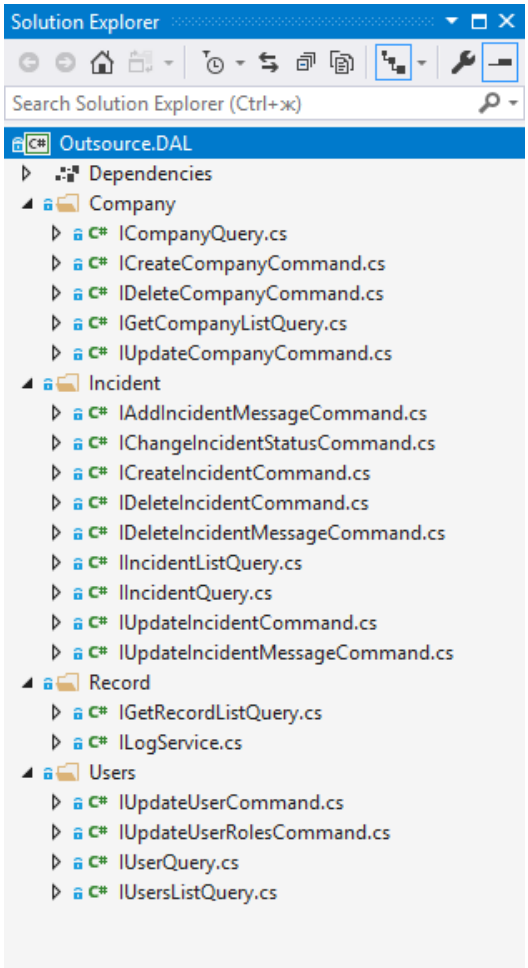

Рисунок 6 – Список интерфейсов проекта Data Access Layer

Проект DAL.Implementation - содержит реализации интерфейсов команд и запросов, которые загружаются в IoC контейнер, и внедряются в требуемые методы как внешние зависимости. На рисунке 7 приведен список описанных классов.

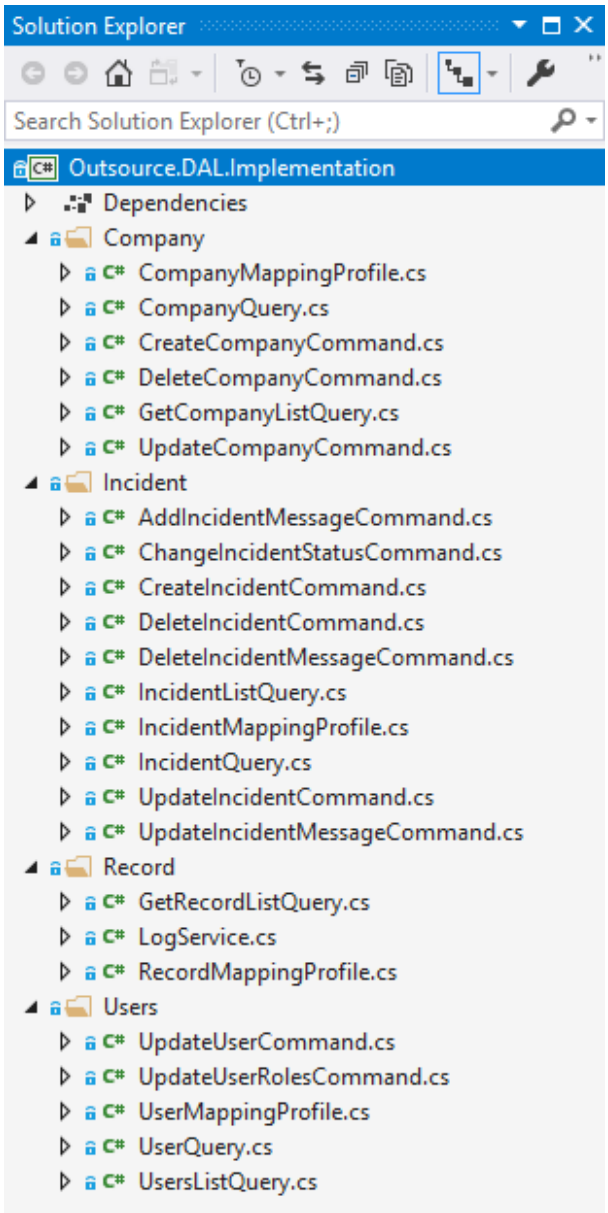

Рисунок 7 – Список классов проекта реализации DAL

Проект DB (DataBase) - содержит контекст для доступа к данным, который так же внедряется из IoC-контейнера. На рисунке 8 приведено содержимое проекта.

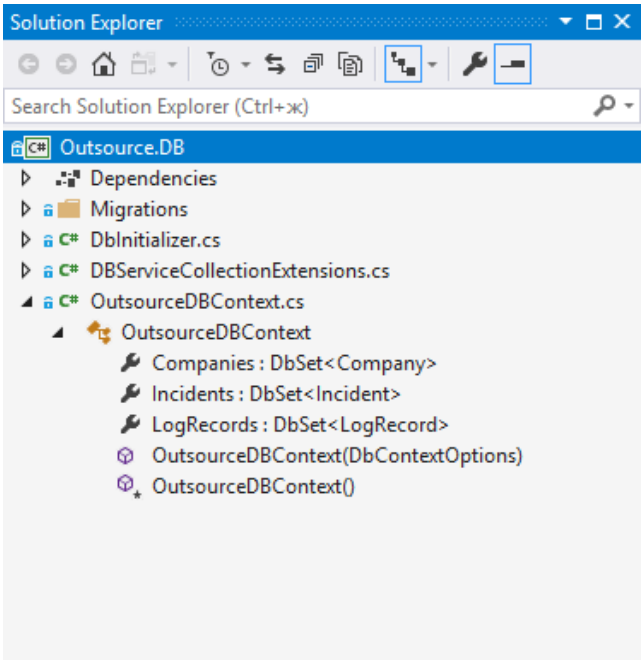

Рисунок 8 – Содержимое проекта контекста доступа к данным

Проект Entities - содержит доменные модели (примитивы), проецируемые на таблицы БД при помощи ORM Entity Framework. На рисунке 9 приведен список моделей.

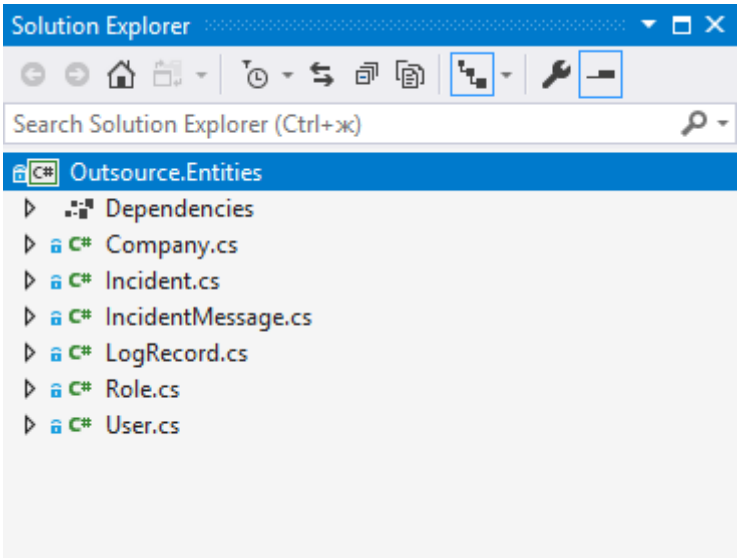

Рисунок 9 – Список моделей проекта Entities

Проект ViewModels - содержит модели презентации, передаваемые клиентам из сервиса WebAPI, а также модели исключений\фильтры. Список моделей приведен на рисунке 10.

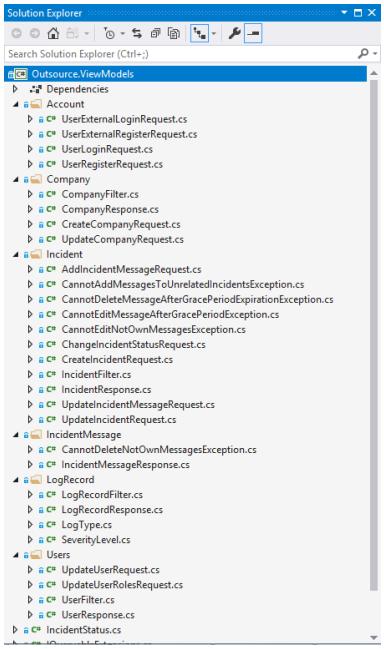

Рисунок 10 - Список моделей проекта ViewModel

Проект Web - содержит точку входа в сервис WebAPI, его контроллеры для авторизации и доступа к данным, а также точку входа в Frontend SPA Содержимое проекта представлено на рисунке 11.

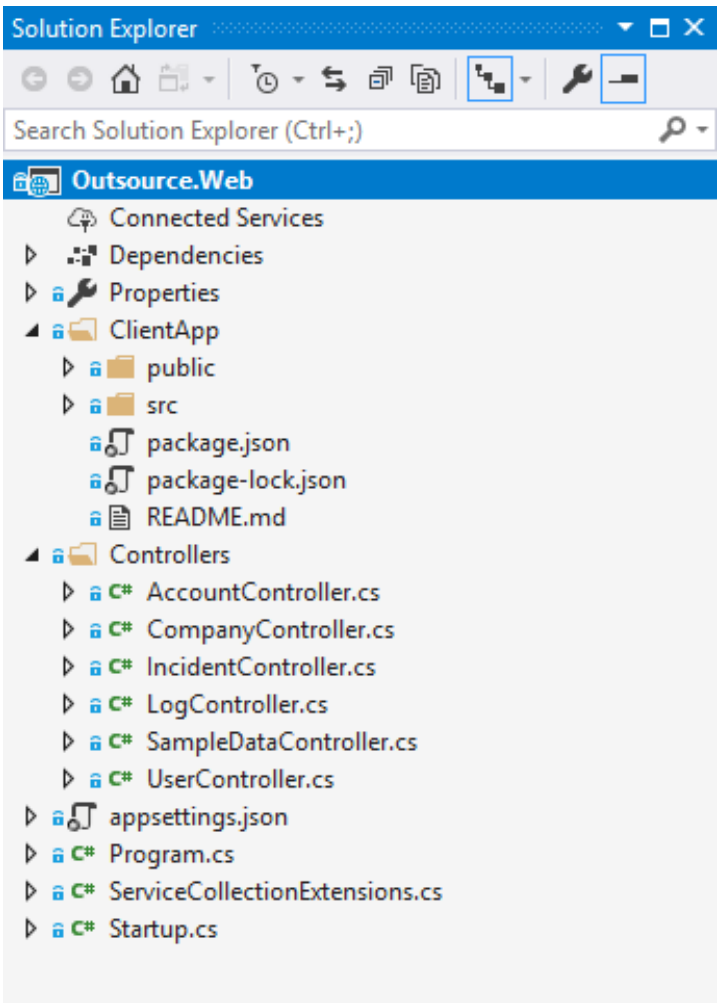

Рисунок 11 – Содержимое проекта Web

Рассмотрим поток и маршруты данных между frontend и backend-частями системы. Пользователь проходит аутентификацию в приложении, после чего переходит к модулю, реализующему требуемую функциональность. После заполнения форм и\или запуска определённых функций, данные из свойств и состояния компонента передаются через события (redux action) в хранилище. Из хранилища данные передаются в формате JSON-объекта backend-сервису при помощи POST\PUT запроса. На стороне backend-сервиса происходит маршрутизация запроса конкретному контроллеру на основании таблицы маршрутизации, которая строится при помощи компонента ASP.Net, называющегося attribute-based routing.

Контроллер при инициализации, получает из встроенного IoC-контейнера необходимые зависимости (в частности, экземпляр класса

UserManager, необходимого для работы системы журналирования событий безопасности). После инициализации контроллера на основании атрибутов маршрутизации выбирается конкретный метод, который следует вызвать, после чего, проверяются полномочия пользователя на осуществление данного запроса. В случае, если запрос не авторизован - если запрос помечен как осуществленный при помощи AJAX (установлен заголовок "X-Requested-With" со значением "XMLHttpRequest") , в ответ возвращается ответ с кодом 401. Если запрос не помечен как AJAX - происходит переадресация на страницу входа.

После проверки, в случае успешно авторизованного запроса, методу передаются из IoC-контейнера все требуемые зависимости. В число зависимостей большинства методов входят:

Интерфейс запроса\команды (например, IIncidentListQuery) содержащийся в проекте DAL - вместо него IoC-контейнер подставляет конкретную реализацию из проекта DAL.Implementation, зарегистрированную при помощи метода services.AddTransient().

Объект, содержащий информацию о запросе, находящийся в проекте ViewModels - может быть помечен атрибутом [FromBody] - в этом случае, ASP.Net десериализует JSON-тело запроса в один или несколько объектов требуемых типов. Данный тип десериализации применяется для POST\PUT запросов. Для запросов, не имеющих тела (GET, DELETE) десериализация происходит из строки URI.

После инициализации метода, в нём у команды\запроса вызывается метод RunAsync (в случае метода, запрашивающего данные) либо ExecuteAsync (в случае метода, изменяющего данные) с передачей им требуемых входных данных.

Команда получает свои зависимости (в том числе и контекст данных из проекта DB) через конструктор из IoC-контейнера. После инициализации команды и её зависимостей, команда получает модель запроса (все модели содержатся в проекте ViewModels), после чего при помощи AutoMapper

происходит трансляция данных из модели в доменную модель (объект DTO), содержащуюся в проекте Entities. Затем происходит выборка\изменение данных, после чего происходит обратная трансляция из коллекции\экземпляра доменной модели в модель представления из проекта ViewModels. После этого, модель представления возвращается контроллеру, вызвавшему команду.

Контроллер сериализует модель представления в JSON-формат, и возвращает её клиентскому приложению в redux. Изменение данных в хранилище redux вызывает повторный рендер всех компонентов, которые подписаны на его изменение. Общая схема структуры работы веб-сервиса приведена на рисунке 12.

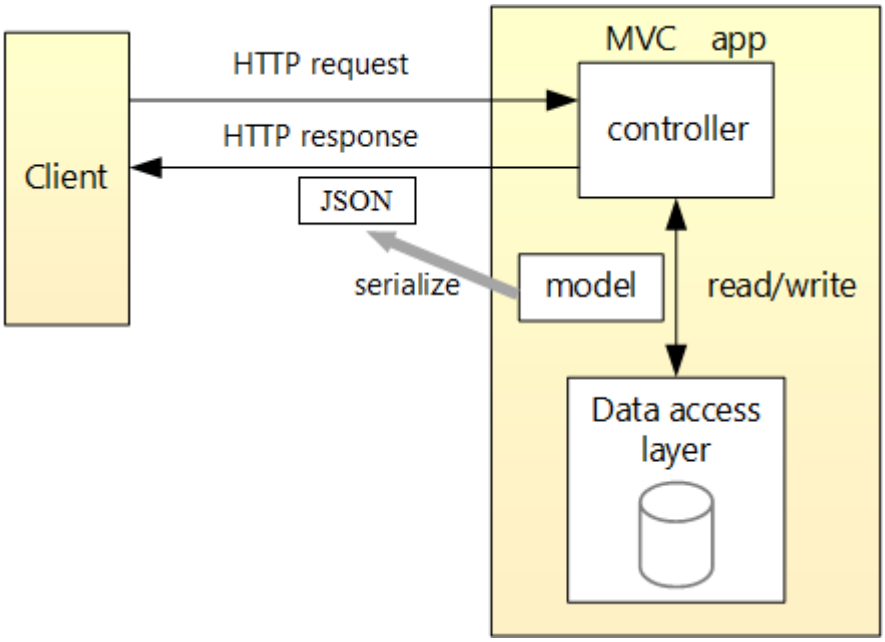

Рисунок 12 – Общая структура веб-сервиса

3.2 Веб-приложение

Интерфейс пользователя, в котором выполняются основные действия менеджерами, сотрудниками колл-центра и инженерами технической поддержки (находящимися в офисе, при использовании для обслуживания

клиента варианта с удалённой технической поддержкой) представляет собой веб-приложение, написанное на JavaScript.

Для загрузки приложения применяется схема так называемого fallback-маршрута, при которой на все маршруты, которые не смогла смаршрутизировать backend-часть, отдаётся файл index.html из директории ClientApp\public, в котором происходит инициализация и загрузка клиентского интерфейса. При инициализации приложения в него передаётся весь URL запроса, после чего приложение маршрутизирует запрос пользователя уже на клиенте, без перезагрузки страницы.

Проект создан используя command-line interface create-react-app, который позволяет корректно проинициализировать проект стартовыми файлами, и необходимыми пакетами.

Всё клиентское приложение располагается в проекте Outsource.Web, в подкаталоге ClientApp. При запуске проекта Outsource.Web в режиме отладки, сборщиком выполняется команда "npm install", сверяющая содержимое файла package.json, и содержимое подкаталога node_modules, в случае расхождений - недостающие пакеты скачиваются и устанавливаются). При сборке проекта в режиме релизной сборки после выполнения "npm install" выполняется "npm run build", создающая оптимизированную копию клиентского приложения, и собирающая все файлы проекта в монолитные css\js\html файлы с целью оптимизации скорости загрузки, уменьшения объёма загружаемых файлов и увеличения скорости работы клиентского приложения.

Приложение написано на JavaScript спецификации ES6\ES7 с использованием библиотек React, Redux для основной функциональности, axios для работы с сетевыми запросами, styled-components для работы с CSS для улучшения пользовательского опыта при использовании клиентской части, а также плагинами и сопутствующими компонентами для указанных библиотек.

Поскольку поддержка спецификаций ES6\ES7 даже в современных браузерах является неполной (см рис.1), в приложении используется

библиотека Babel, производящая транспиляцию ES6\ES7 в код спецификации ES5, который может быть обработан основными современно используемыми браузерами. Список браузеров с поддержкой ES6 приведен на рисунке 13.

Desktop browsers

Konq 4.14	IE 11	Edge 16	Edge 17	FF 52 ESR	FF 59	FF 60 ESR	CH 66, OP 53	CH 67, OP 54	SF 11	SF 11.1
0/2	0/2	0/2	0/2	0/2	0/2	0/2	0/2	0/2	2/2	2/2
No	No	No	No	No	No	No	No	No	Yes	Yes
No	No	No	No	No	No	No	No	No	Yes	Yes
0/7	0/7	7/7	7/7	6/7	7/7	7/7	7/7	7/7	7/7	7/7
0/5	0/5	5/5	5/5	5/5	5/5	5/5	5/5	5/5	5/5	5/5
0/15	0/15	15/15	15/15	15/15	15/15	15/15	15/15	15/15	15/15	15/15
0/6	0/6	6/6	6/6	6/6	6/6	6/6	6/6	6/6	6/6	6/6
0/9	0/9	9/9	9/9	7/9	9/9	9/9	9/9	9/9	9/9	9/9
0/4	0/4	4/4	4/4	4/4	4/4	4/4	4/4	4/4	4/4	4/4
0/5	0/5	5/5	5/5	5/5	5/5	5/5	5/5	5/5	5/5	5/5
0/5	0/5	5/5	5/5	5/5	5/5	5/5	5/5	5/5	5/5	5/5
0/22	0/22	22/22	22/22	21/22	22/22	22/22	22/22	22/22	22/22	22/22
0/24	0/24	24/24	24/24	23/24	24/24	24/24	24/24	24/24	24/24	24/24
0/24	0/24	23/24	23/24	21/24	24/24	24/24	24/24	24/24	24/24	24/24
0/2	0/2	2/2	2/2	1/2	2/2	2/2	2/2	2/2	2/2	2/2
0/2	0/2	2/2	2/2	2/2	2/2	2/2	2/2	2/2	2/2	2/2
2/16	12/16	16/16	16/16	16/16	16/16	16/16	16/16	16/16	16/16	16/16
0/12	10/12	12/12	12/12	12/12	12/12	12/12	12/12	12/12	12/12	12/12
No	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
0/13	0/13	13/13	13/13	13/13	13/13	13/13	13/13	13/13	13/13	13/13
0/24	0/24	24/24	24/24	24/24	24/24	24/24	24/24	24/24	24/24	24/24
0/8	0/8	8/8	8/8	8/8	8/8	8/8	8/8	8/8	8/8	8/8
0/27	0/27	27/27	27/27	25/27	27/27	27/27	27/27	27/27	27/27	27/27
8/46	16/46	46/46	46/46	46/46	46/46	46/46	46/46	46/46	46/46	46/46
0/19	8/19	19/19	19/19	18/19	19/19	19/19	19/19	19/19	19/19	19/19
0/19	8/19	19/19	19/19	18/19	19/19	19/19	19/19	19/19	19/19	19/19
0/12	6/12	12/12	12/12	11/12	12/12	12/12	12/12	12/12	12/12	12/12
0/11	0/11	11/11	11/11	10/11	11/11	11/11	11/11	11/11	11/11	11/11
0/34	0/34	34/34	34/34	34/34	34/34	34/34	34/34	34/34	34/34	34/34
0/20	0/20	20/20	20/20	20/20	20/20	20/20	20/20	20/20	20/20	20/20
0/8	0/8	8/8	8/8	8/8	8/8	8/8	8/8	8/8	8/8	8/8
0/12	0/12	12/12	12/12	12/12	12/12	12/12	12/12	12/12	12/12	12/12
0/26	0/26	17/26	17/26	26/26	26/26	26/26	26/26	26/26	26/26	26/26
1/4	1/4	4/4	4/4	4/4	4/4	4/4	4/4	4/4	4/4	4/4
3/17	0/17	16/17	16/17	12/17	17/17	17/17	17/17	17/17	17/17	17/17
0/2	0/2	2/2	2/2	2/2	2/2	2/2	2/2	2/2	2/2	2/2
0/10	0/10	10/10	10/10	10/10	10/10	10/10	10/10	10/10	10/10	10/10
0/6	0/6	1/6	1/6	6/6	6/6	6/6	6/6	6/6	6/6	6/6
0/11	0/11	11/11	11/11	11/11	11/11	11/11	11/11	11/11	11/11	11/11
0/10	0/10	10/10	10/10	9/10	9/10	10/10	10/10	10/10	10/10	10/10
6/9	0/9	9/9	9/9	9/9	9/9	9/9	9/9	9/9	9/9	9/9
14/17	0/17	17/17	17/17	17/17	17/17	17/17	17/17	17/17	17/17	17/17
No	No	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
0/11	0/11	11/11	11/11	11/11	11/11	11/11	11/11	11/11	11/11	11/11
0/4	0/4	4/4	4/4	4/4	4/4	4/4	4/4	4/4	4/4	4/4
0/6	0/6	6/6	6/6	6/6	6/6	6/6	6/6	6/6	6/6	6/6
0/4	0/4	4/4	4/4	4/4	4/4	4/4	4/4	4/4	4/4	4/4
0/6	0/6	6/6	6/6	6/6	6/6	6/6	6/6	6/6	6/6	6/6
0/5	0/5	5/5	5/5	5/5	5/5	5/5	5/5	5/5	5/5	5/5
0/36	0/36	24/36	24/36	35/36	36/36	36/36	36/36	36/36	35/36	35/36
0/11	0/11	11/11	11/11	11/11	11/11	11/11	11/11	11/11	11/11	11/11
0/2	0/2	2/2	2/2	2/2	2/2	2/2	2/2	2/2	2/2	2/2
0/6	0/6	6/6	6/6	6/6	6/6	6/6	6/6	6/6	6/6	6/6

Рисунок 13 - Поддержка ES6 современными браузерами

Current aligned	Usage relative	Date relative	Show all							
IE	Edge *	Firefox	Chrome	Safari	IOS Safari *	Opera Mini *	Chrome for Android	UC Browser for Android	Samsung Internet
			49		10.3				
	16	59	65		11.2				4
11	17	60	66	11.1	11.3	[1] all	66	11.8	6.2
	18	61	67	12					
		62	68	TP					
			69						

Рисунок 14 - Поддержка ES5 современными браузерами

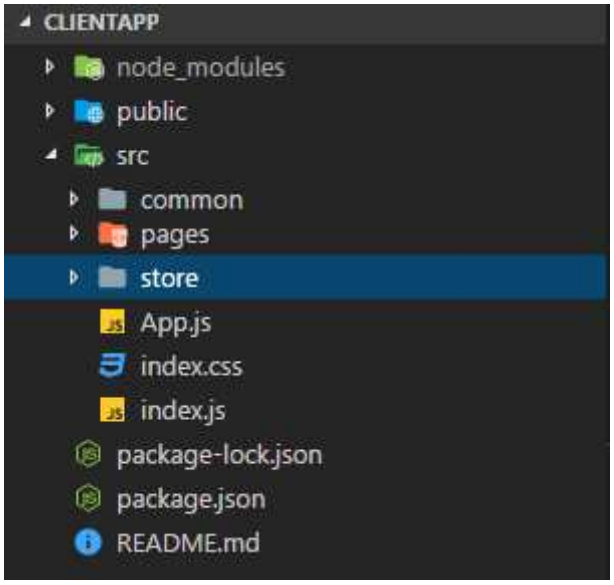

Рисунок 15 - Общая структура проекта представлена на рисунке

Проект содержит следующие основные элементы:

– Папка node_modules - содержит необходимые для работы и построения приложения зависимости в виде js-библиотек.

– Каталог public содержит index.html, favicon.ico (показывается в адресной строке браузера), и manifest.json (необходим для создания ярлыков на веб-приложение под ОС Android).

– Каталог src содержит основной код клиентского приложения.

– package.json и package-lock.json являются служебными файлами npm, в которых содержится информация о необходимых для работы\сборки\разработки приложения.

– README.md - стандартное описание проектов, созданных при помощи cli create-react-app, содержащее информацию о построении проекта, и краткие решения типовых задач, которые могут встать перед разработчиком.

Рассмотрим подробнее каталог src, и структуру приложения в целом. При загрузке клиентской части, во время обработки файла index.html в нём вызывается файл index.js, в котором содержится код, инициализирующий React приложение, и необходимую инфраструктуру. В частности, в нём:

– Создаётся объект History, который впоследствии передаётся в компонент Router (из библиотеки react-router) позволяющий осуществлять маршрутизацию запросов внутри клиентского приложения;

– При помощи функции configureStore создаётся и настраивается глобальное хранилище данных, которые будет служить т.н. Single source of thruth (единственное место, откуда приложение будет получать данные). Впоследствии это хранилище передаётся в компонент Provider (из библиотеки react-redux);

– Устанавливается глобальный перехватчик ошибок в ответах axios, для корректной обработки сетевых ошибок, возникающих при общении с backend-сервисом.

После инициализации инфраструктуры проекта, при помощи запроса document.getElementById получается ссылка на div-элемент с html-идентификатором root. Данная ссылка передаётся в функцию ReactDOM.render, которая выполняет рендеринг базового JSX-кода приложения.

```
const rootElement = document.getElementById('root');

ReactDOM.render(
  <Provider store={store}>
    <ConnectedRouter history={history}>
      <App />
    </ConnectedRouter>
  </Provider>,
  rootElement);
```

Рисунок 16 - Вызов функции render

Вся инфраструктура хранения и обработки данных в приложении расположена в отдельных файлах в каталоге store. Методы доступа к данным и их хранение сгруппировано по функционалу, к которому относятся указанные данные (авторизация - auth.js, информация об обслуживаемых компаниях - companies, работа с инцидентами - incidents.js, журналы безопасности - logs.js, и т.д.).

Также в каталоге store находится configureStore.js, который импортирует все редьюсеры, и собирает их в один корневой, подключает библиотеку thunk для подключения асинхронности в действия (action) доступа к данным. В случае, если приложение собрано в режиме отладки, дополнительно включается поддержка браузерного расширения redux, что позволяет ускорить разработку.

Внутреннее устройство модуля хранения и доступа к данным рассмотрим на примере incidents.js, как самого многофункционального.

Общая структура модуля представлена на рисунке 17.

```
export const actions = {…
};
export const reducer = (state = initialState, action) => {…
}

export const selectors = {…
};
```

Рисунок 17 - Общая структура модуля хранения и доступа к данным

Как видно из рисунка, модуль экспортирует функцию reducer, и 2 объекта (actions, selectors), каждый из ключей которых является самостоятельной javascript-функцией.

Редьюсер это функция, принимающая состояние (state), и действие (action), и возвращающая новое состояние после выполнения каких-либо вычислений. Эта функция импортируется модулем configureStore, и регистрируется в составе хранилища. Действия вызываются компонентами приложения, и отправляют определённые события, на которые впоследствии реагирует редьюсер.

Селектор - это функция, получающая состояние, и возвращающая часть этого состояния. Селекторы необходимы для понижения связанности кода, и повышения переносимости компонентов, т.к. это позволяет инкапсулировать особенности реализации структуры данных внутри состояния.

В данном модуле используются следующие действия (каждый метод отправляет события, оканчивающиеся на '_START' - начало запроса,'_SUCCESS' - запрос успешен,'_FAIL' - ошибка в запросе):

– get - отвечает за получение списка инцидентов, применяя фильтр. Запрос отправляется методом GET на точку "/api/incident". Префиксом событий является 'GET_INCIDENT'. В момент начала запроса отправляется событие 'GET_INCIDENT_START', без нагрузки. В ответ на это, редьюсер обновляет состояние, устанавливая флаг loading = true. В случае успешности запроса отправляется событие 'GET_INCIDENT_SUCCESS' с нагрузкой в виде поля data из ответа сервера, редьюсер реагирует на данное событие

устанавливая флаг loading в false, и выгружая информацию из ответа в массив companies, и служебные поля (номер страницы, общее количество страниц, и т.д.). В случае ошибки - отправляются события 'GET_INCIDENT_FAIL' с нагрузкой в виде объекта ошибки, а также вызывается метод sendError, публикующий информацию об ошибке в редьюсер error.

– getSingle - отвечает за получение детальной информации по инциденту, отправляет события с префиксом 'DETAILS_INCIDENT'. В случае успешности - редьюсер переносит информацию из ответа в поле incidentDetail

– addComment - отвечает за добавление комментариев к конкретному инциденту, редьюсер в случае успеха также обновляет поле incidentDetail обновленной моделью, возвращаемой с backend-сервиса. Префиксом событий является 'COMMENT_ADD'.

– save - позволяет сохранить обновленный инцидент. Префикс - 'SAVE_INCIDENT', в случае успешности - происходит навигация на "/incident/details/{id}", где {id} - это поле Id из модели, полученной с backend-сервиса.

– assignTo - позволяет назначить инцидент на какого-либо пользователя. Префикс - 'ASSIGN_INCIDENT'.

– new - позволяет создать новый инцидент, префикс - 'CREATE_INCIDENT'. В случае успешности запроса - происходит навигация на "/incident/details/{id}", где {id} - это поле Id из модели, полученной с backend-сервиса.

Помимо редьюсера и методов общения с сервисом, в данном модуле представлены селекторы, позволяющие получить часть глобального состояния. К ним относятся:

– getList - возвращает выборку из всего массива инцидентов, хранящегося в состоянии. В выборку попадают идентификатор, заголовок, название компании, статус, пользователь на которого назначен инцидент, и его статус;

– getDetails - возвращает детали инцидента, а также статус загрузки и сохранения (поля incidentDetail, saving, loading).

При первоначальной загрузке системы, в модуле App происходит подключение к хранилищу, и получение из него информации о том, аутентифицирован ли пользователь. В случае, если пользователь аутентифицирован, то из хранилища также загружается информация о нём, и наборе его ролей. В момент рендера модуля App на страницу, в методе render формируется полный список пунктов меню, который вместе с информацией о пользователе передаётся в компонент Layout. Также в данном методе выводится компонент Modal для отображения модальных сообщений об ошибках (получает своё содержимое из свойства, привязанного к селектору getError), и таблица маршрутизации, формируемая в виде списка компонентов AuthenticatedRoute, которым передаётся список ролей пользователя, а также список ролей, которым разрешёна навигация по конкретному маршруту. Таблица маршрутизации построена таким образом, что последним в списке стоит перенаправление на корень приложения, что приводит к тому, что все запросы, которые не могут быть смаршутизированы внутри роутера - перенаправляются в корень, таким образом улучшается пользовательский опыт, и повышается надежность и безопасность приложения.

После рендера компонента App, из метода ComponentDidMount вызывается действие checkAuthStatus, которое проверяет, есть ли в localStorage браузера JWT-токен, а также срок его действия. В случае, если токен есть, и он действителен.

Компонент AuthenticatedRoute находится в каталоге common, и представляет собой обёртку над компонентом Route из библиотеки react-router. AuthenticatedRoute подключается к хранилищу, и получает из него информацию о том, аутентифицирован ли пользователь. В зависимости от того, аутентифицирован ли пользователь, обладает ли он необходимой ролью, и есть ли ограничение по ролям у данного маршрута - выполняется одно из действий, перечисленных на рисунке 18.

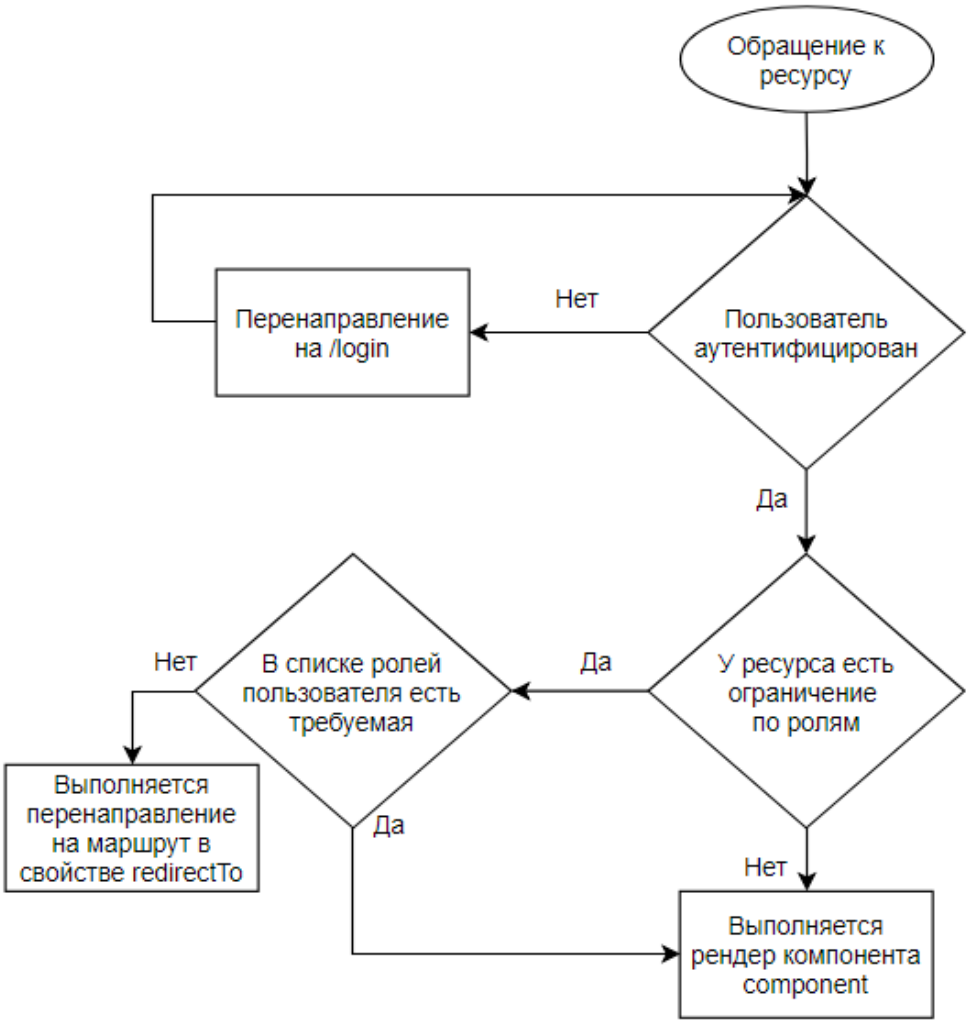

Рисунок 18 – Блок-схема обработки обращения к ресурсу

В каталоге pages содержатся модули, позволяющие работать с тем или иным функционалом системы. Список модулей представлен на рисунке 19.

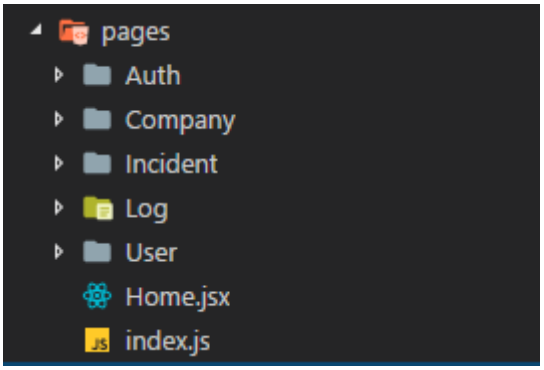

Рисунок 19 - Список модулей системы

Компонент Layout выводит компоненты, вокруг которых он обёрнут, и в дополнение к этому выводит компонент NavMenu, который выводит компоненты меню приложения в зависимости от набора ролей пользователя.

После загрузки приложения, компонент App рендерит маршрут "/", который использует страницу Home, которая строится из компонента Home.jsx. Страница Home содержит приветственную информацию для пользователя. На рисунке 20 представлен интерфейс работы веб-приложения.

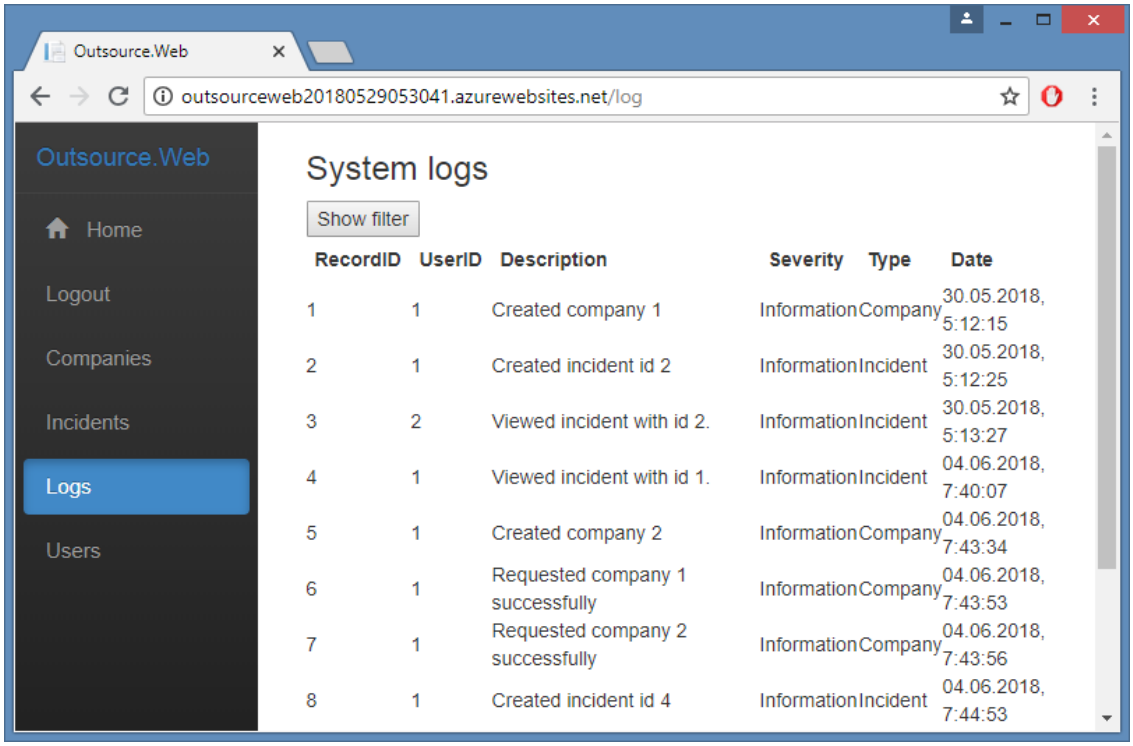

Рисунок 20 – Интерфейс работы веб-приложения

Дальнейшую организацию структуры конкретных модулей рассмотрим на примере модуля Incident.

Каталог модуля содержит файл index.js, реэкспортирующий компонент Incident в виде именованного экспорта (named export) из файла Incident.jsx.

Компонент Incident выводит на страницу заголовок-название модуля (<h3>Incident</h3>), и компонент Switch, позволяющий осуществить вывод одного из вложенных маршрутов.

Таким образом, компонент App маршрутизирует только маршруты "верхнего уровня" вида "/log", "/incident", "/user", и т.д. Вся дальнейшая маршрутизация запроса осуществляется внутри родительского компонента, которому передаётся весь URI.

Маршруты также выводятся в виде AuthenticatedRoute, что позволяет более гранулировано управлять правами пользователя, запрещая или разрешая доступ не только на уровне компонента, но и на уровне конкретного действия. Ролевая модель доступа внутри модуля incident представлена в таблице 3.

Таблица 3 – Ролевая модель доступа модуля incident

URI	Описание возможных действий	Компонент	Необходимая роль для доступа к компоненту
/:page	Вывод списка инцидентов (страница №:page, с возможностью фильтрации	List.jsx	Администратор системы, Менеджер, Инженер технической поддержки, Специалист колл-центра
/new	Регистрация нового инцидента от клиента	New.jsx	Администратор системы, Менеджер, Специалист колл-центра
/edit/:id	Редактирование инцидента с id = :id	Edit.jsx	Администратор системы, Менеджер
/details/:id	Просмотр инцидента с id = :id	Details.jsx	Администратор системы, Менеджер, Инженер технической поддержки, Специалист колл-центра

Все неизвестные маршруты внутри компонента - вызывают перенаправление на /1, что вызывает вывод первой страницы инцидентов.

Дополнительно к AuthenticatedRoute для защиты приложения от несанционированного доступа применяется компонент ShowForRole, получающий список ролей, информацию о пользователе, и компонент, который необходимо вывести при совпадении одной из ролей пользователя с требуемой ролью для компонента. Это позволяет ограничить доступ к части страницы.

К примеру, данным компонентом защищен функционал создания нового инцидента, вызываемый из компонента List. Поскольку компонент List доступен одной из тех ролей, которым недоступен маршрут /new (а именно

роли "Инженер технической поддержки"), это действие защищается компонентом ShowForRole, скрывающим кнопку "создать инцидент" у сотрудников технической поддержки для улучшения пользовательского опыта, и соблюдения бизнес-процессов компании.

Компонент Details выводит подробности инцидента, запрашивая их и информацию о текущем пользователе через свои свойства из глобального хранилища (store). Из данного компонента пользователи, обладающие ролями "Администратор системы" или "Менеджер" могут удалять\редактировать инцидент.

Внутри компонента Details внизу страницы выводится компонент MessageForm, обёрнутый в Toggleable (позволяет скрывать часть страницы), что позволяет всем пользователям системы, имеющим доступ к маршруту /details/:id оставлять комментарии к данному инциденту. Комментарии выводятся без фильтрации, с сортировкой по дате в восходящем порядке.

Также, из компонента Details пользователи, обладающие ролями "Администратор системы" или "Менеджер" инициировать процесс назначения инцидента на конкретного исполнителя. Данный функционал обеспечивается компонентом Assign, который выводит зарегистрированных пользователей, и кнопку "Назначить". Компонент выводится в модальном режиме, блокируя взаимодействие пользователя с другими компонентами.

Компонент New представляет собой простую форму с валидацией вводимых значений по требованиям аналогично backend-сервису. В случае, если какой-либо элемент ввода не прошёл валидацию, возможность создать инцидент блокируется. После создания инцидента происходит перенаправление пользователя внутри приложения на маршрут /incident/details/:id, где :id это идентификатор нового инцидента, полученный с сервера.

Компонент Edit является усовершенствованной версией компонента New, и отличается от него набором полей (отсутствует возможность изменить компанию, от имени которой создан инцидент, но добавлена возможность

изменить статус заявки), а также тем, что он получает свои первоначальные значения из свойств, которые заполняются автоматически из хранилища redux.

Компонент List выводится по умолчанию при переходе по маршруту /incident/1 (или другой номер страницы), и позволяет просматривать список инцидентов в постраничном режиме с поддержкой фильтрации.

Постраничный вывод реализован при помощи компонента Pagination, который используя переданные ему через свойства данные об общем количестве страниц, и текущей - позволяет вывести по 1 кнопке на каждую страницу вывода, отдельно подсвечивая текущую страницу, на которой находится пользователь приложения.

При монтировании компонента в DOM-дерево, а также при обновлении состояния и свойств компонента, вызывается функция applyFilter, компилирующая фильтр в один объект, и запуская соответствующее действие. В случае, если номер страницы не изменился - повторный запрос не отправляется.

При выводе таблицы со списком инцидентов, обработчик щелчка строки таблицы настроен таким образом, что после щелчка пользователь перенаправляется по маршруту /incident/details/:id, где :id это идентификатор инцидента, по которому был выполнен щелчок.

3.3 Мобильное приложение

Мобильное приложение реализовано для платформы Android в демонстративных целях. Решение состоит из двух проектов, первый из которых содержит общие для всех поддерживаемых платформ компоненты.

Для аутентификации используется библиотека Xamarin.Auth, позволяющая реализовать вход в систему через социальные сети. В общем виде, процедура аутентификации с помощью сервисов Google приведена на рисунке 21.

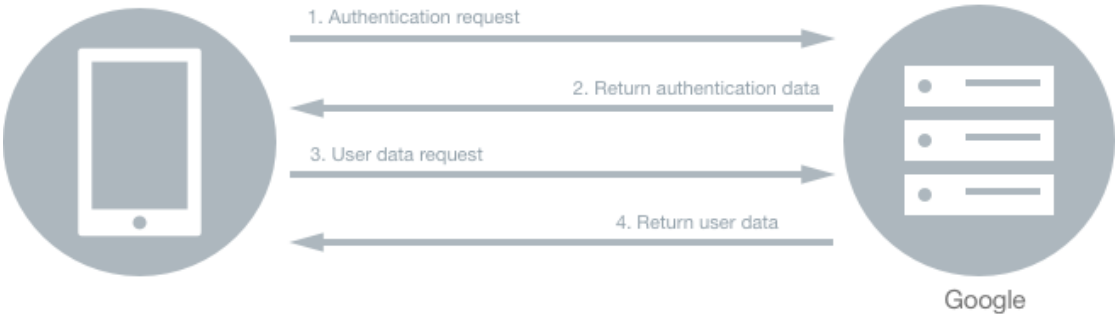

Рисунок 21 – Процедура аутентификации Xamarin.Auth

Xamarin.Auth позволяет сохранять между запусками приложения информацию о пользователе, записывать полученные cookie и токены. Эта возможность реализована при запуске приложения. Сохраненный токен, полученный ранее от веб-сервиса информационной системы, проверяется на дату истечения и, в случае если срок его действия не истек, используется для отправки запросов на получение/отправку данных при работе приложения. Если токен устарел, происходит перенаправление на страницу входа.

Страница входа содержит ссылку для перехода на процедуру аутентификации Google. В соответствии с текущими правилами безопасности мобильных приложений, актуальным способом проведения процедуры аутенитификации является использования страницы браузера. Поэтому страница для ввода данных учетной записи Google открывается в браузере. Пользователь вводит свои учетные данные и, в случае успешного завершения, приложения, используя настраиваемую схему URI, отлавливает возврат в приложение. Полученный токен Google отправляется в веб-приложение информационной системы для проверки подленности и, в случае корректной проверки, выдает свой JWT-токен пользователю для отправки запросов. На рисунке 22 приведена структура обмена данными метода аутентификации веб-сервиса информационной системы.

После процедура аутентификации, пользователь имеет возможность просматривать список инцидентов, открывать каждый из них и вносить в них изменения. На рисунке 23 показан интерфейс пользователя в эмуляторе Android с тестовым набором инцидентов.

POST /api/account/extlogin

Parameters Try it out

Name	Description
request (body)	Example Value | Model { "token": "string" } Parameter content type application/json-patch+json

Responses Response content type application/json

Code	Description
200	Success
401	Unauthorized

Рисунок 22 – Структура обмена данными метода аутентификации веб-сервиса

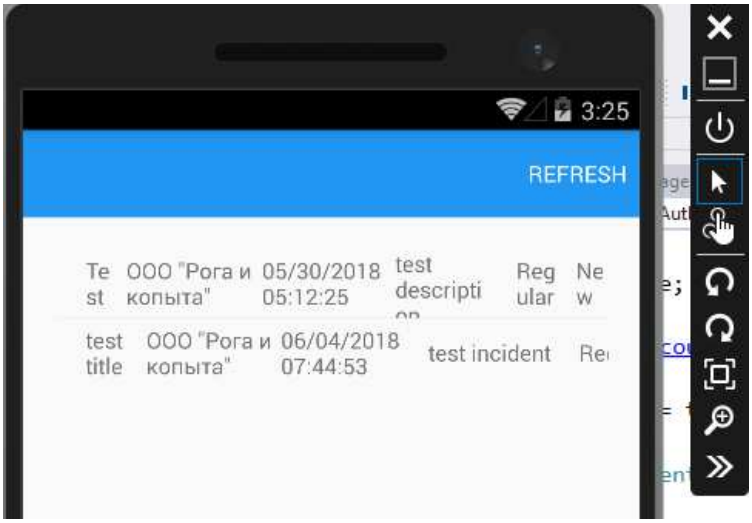

Рисунок 23 – Интерфейс пользователя мобильного приложения

4 Тестовое внедрение

4.1 Конфигурация тестовой среды

Серверная часть системы работает на платформе IBM PC. Операционная система: Microsoft Windows Server либо *nix (поддерживается технология ASP.Net Core). База данных MSSQL, язык программирования C#.

Пользовательский интерфейс представляет собой веб-портал, написанный с использованием технологий ASP.Net Core Web API, JS (React.js).

Мобильная часть системы в виде приложения для инженера под основные мобильные платформы (Android, Windows Phone, iOS), написанных на языке C#, с использованием платформы Xamarin.

4.2 Описание методики тестирования

В качестве ключевого показателя оценки эффективности процесса работы с обращениями клиентов компании, оказывающей услуги по обслуживанию оргтехники выбраны количество обработанных инцидентов, скорость получения деталей инцидента и время, потраченное на обработку инцидента.

Для проведения тестирования выбран инженер компании, данные о заявке которому в течение дня поступали только через мобильное приложение разработанной информационной системы. Инженер выполнял плановые посещения клиентов для выполнения работ, входящих в перечень входящих в абонентскую плату. Установленная периодичность плановых посещений составляет 1 месяц. Сравнив показатели предыдущего месяца, посчитаем разницу, оценив эффект от использования разработанной информационной системы. Для посещения каждого клиента создавался инцидент после того, как были завершены работы на территории предыдущего клиента.

Таким образом по окончании рабочего дня было посчитано общее количество обработанных инцидентов за день, среднее время появления данных о инциденте после его заведения и среднее время обработки инцидента.

4.3 Результаты тестирования

Результаты показателей эффективности отображены в таблице 4.

Таблица 4 – Результаты показателей эффективности

Показатель/Месяц	Апрель	Май
Количество инцидентов, шт.	4	7
Скорость получения инцидента, мин.	5	0,2
Среднее время обработки инцидента, мин.	120	68,6

Следует отметить, что количество инцидентов по плановым работам может зависеть от количества обращений персонала обслуживаемого клиента при его посещении. Поэтому время одного визита может занять несколько часов и сбить график посещения. В результате измерений количество посещений при использовании мобильного приложения выросло, по сравнению с предыдущим месяцем.

Скорость получения данных о новом инциденте зависит от скорости подключения мобильного устройства к сети инетернет. В случае с предыдущим месяцем, данные сравнивались со средним временем на совершение телефонного звонка инженеру из офиса. В результате использования разработанной информационной системы, мобильное приложение получало данные об инциденте с веб-сервиса уже через 5 секунд после того как инцидент был зарегистрирован. При активной нагрузке на веб-сервис одновременной работой большого количества инженеров и офисных сотрудников компании, данный показатель времени может снижаться.

Среднее время обработки инцидента включает в себя затраты на дорогу до следующего клиента, однако показатель при использовании разработанной информационной системы показа двойное преимущество по сравнению с предыдущим месяцем.

И итоге, тестирование дало положительные результаты и отзывы участников, что позволяет утверждать, что внедрение разработанной информационной системы востребовано в данной компании.

ЗАКЛЮЧЕНИЕ

Процессный подход к анализу и моделированию бизнес-процессов и последующей разработке требований к информационным системам, позволяет оперативно сопровождать (изменять и дорабатывать) описанные рациональные технологии работ, безболезненно (параллельно с эксплуатацией) для пользователей модернизировать информационную систему Компании, наращивать мощность базы данных и поддерживать её в актуальном состоянии.

Другим важнейшим преимуществом применения процессного подхода является возможность формализации технологии выполнения работ по реорганизации деятельности предприятий и проектированию информационных систем поддержки рациональных бизнес-процессов. На основе формализации были созданы методическое обеспечение и соответствующая ему автоматизированная технология выполнения работ по автоматизации бизнес-процессов компании, обслуживающей оргтехнику.

В ходе выполнения бакалаврской работы выполнены следующие задачи:

– проведен анализ бизнес-процессов и моделирование применения технологий при работе инженеров компании, оказывающей услуги по обслуживанию оргтехники;

– разработан проект информационной системы, позволяющей автоматизировать работу инженеров компании;

– произведена опытная эксплуатация информационной системы, позволившая повысить эффективность рабочего процесса сотрудников компани, обслуживающей оргтехнику.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Аверченков, В. И. Информационные системы на производстве : учебное пособие для вузов. / В. И. Аверченков, Ф. Ю. Лозбинев А. А. Тищенков. – Москва : Издательский центр «Флинта», 2011. – 247 с.
2. Йордан, Э. Структурные модели в объектно-ориентированном анализе и проектировании / Э. Йордан, С. Аргила. – Москва : ЛОРИ, 1999 – 75с.
3. Гусева, Т.И. Проектирование баз данных в примерах и задачах / Т.И. Гусева. – Москва : Радио и связь, 1992. – 160 с.
4. Смирнова, Г.Н. Проектирование экономических информационных систем : учебник / Г.Н. Смирнова, А.А. Сорокин, Ю.Ф. Тельнов. - Москва : Финансы и статистика, 2002 – 36с.
5. Васильев, А.Н. Java / А.Н. Васильев. – Санкт-Петербург : Питер, 2011. – 270 с.
6. Подробно о Xamarin. [Электронный ресурс] : техн. информация / ООО «Хабр». - Москва, 2013. – Режим доступа: http://habrahabr.ru/post/188130/.
7. Рот, Д. Введение в ASP.NET Core. [Электронный ресурс] : техн. информация / Д. Рот, Р. Андерсон, Ш. Луттин. - 2018. – Режим доступа: https://docs.microsoft.com/ru-ru/aspnet/core/index?view=aspnetcore-2.0.
8. Рогозов, Ю. И. Системный подход к созданию метода разработки информационных объектов на основе метамоделей / Ю. И. Рогозов. – Москва : Информатизация и связь, 2011. – С. 57-62.
9. Фримен, А. ASP.NET Core MVC с примерами на C# для профессионалов / А. Фримен. – Санкт-Петербург : Диалектика, 2017. – 368 с.
10. Бэнкс, А. React и Redux. Функциональная веб-разработка. / А. Бэнкс, Е. Порселло. – Санкт-Петербург : Питер, 2018. – 178 с.
11. Меняев, М. Ф. Системы управления организацией / М. Ф. Меняев. – Москва : Омега-Л, 2003. – 464 с.

12. Советов, Б. Я. Информационные технологии : учеб. для вузов / Б. Я. Советов, В. В. Цехановский. – Москва : Высш. шк., 2003. – 263 с.

13. Гагарина, Л. Г. Технология разработки программного обеспечения / Л. Г. Гагарина, Е. В. Кокорева, Б. Д. Виснадул. – Москва : ИД «ФОРУМ», 2012. – 400 с.

14. Маклаков, С. В. Создание информационных систем с AllFusion Modeling Suite / С. В. Маклаков. – Москва : ПРЕСС, 2011. – 512 с.

15. Тихоренко, О. М. Модели массового обслуживания в информационных системах / О. М. Тихоренков. – Москва : Бином, 2006. – 327 с.

16. Гайдамакин, Н. А. Автоматизированные информационные системы, базы и банки данных / Н. А. Гайдамакин. – Москва : Гелиос АРВ, 2002. – 97 с.

17. Рындин, А. А. Проектирование корпоративных информационных систем / А. А. Рындин, А. В. Хаустович, Д. В. Долгих. – Воронеж : Кварта, 2003. – 448 с.

18. Фаронов, В. А. Программирование на языке высокого уровня / В. А. Фаронов. – Санкт-Петербург: Питер, 2006. – 640 с.

19. СТО 4.2 –07 –2014 Система менеджмента качества. Общие требования к построению, изложению и оформлению документов учебной деятельности / Сибир. федер. ун-т. – Красноярск: СФУ, 2014 - 60 с.

ПРИЛОЖЕНИЕ А

Исходный код программы

```
#Part of BackEnd
using System;
using System.Collections.Generic;
using System.Text;
using System.Threading.Tasks;
using Outsource.ViewModels.Incident;

namespace Outsource.DAL.Incident
{
    public interface IAddIncidentMessageCommand
    {
        Task<IncidentResponse> ExecuteAsync(int incidentId, int id,
AddIncidentMessageRequest request);
    }
    public interface IAssignIncidentCommand
    {
        Task<IncidentResponse> ExecuteAsync(int incidentId, AssignIncidentRequest
request, User user);
    }
    public interface IChangeIncidentStatusCommand
    {
        Task<IncidentResponse> ExecuteAsync(int incidentId, User user,
ChangeIncidentStatusRequest request);
    }
    public interface ICreateIncidentCommand
    {
        Task<IncidentResponse> ExecuteAsync(CreateIncidentRequest request, User user);
    }
    public interface IDeleteIncidentCommand
    {
        Task ExecuteAsync(int incidentId, int id);
    }
    public interface IDeleteIncidentMessageCommand
    {
        Task ExecuteAsync(int incidentId, int messageId, User user);
    }
    public interface IIncidentListQuery
    {
        Task<ListResponse<IncidentListResponse>> RunAsync(ListOptions options,
IncidentFilter filter);
    }
    public interface IIncidentQuery
    {
        Task<IncidentResponse> RunAsync(int incidentId, User user);
    }
    public interface IUpdateIncidentCommand
    {
        Task<IncidentResponse> ExecuteAsync(int incidentId, UpdateIncidentRequest
request, Entities.User user);
    }
    public interface IUpdateIncidentMessageCommand
    {
        Task<IncidentResponse> ExecuteAsync(int incidentId, int messageId, User user,
UpdateIncidentMessageRequest request);
    }
   public class AddIncidentMessageCommand : IAddIncidentMessageCommand
```

```
    {
        private ILogService _logService;
        private IMapper _mapper;
        private OutsourceDBContext _context;

        public AddIncidentMessageCommand(ILogService logService, IMapper mapper,
OutsourceDBContext context)
        {
            _logService = logService;
            _mapper = mapper;
            _context = context;
        }
        public async Task<IncidentResponse> ExecuteAsync(int incidentId, int userId,
AddIncidentMessageRequest request)
        {
            var incidentFromDb = await _context.Incidents
                .Include(i=>i.Messages).FirstOrDefaultAsync(i=>i.Id==incidentId);
            if (incidentFromDb != null)
            {
                var newMessage = _mapper.Map<Entities.IncidentMessage>(request);
                newMessage.Author = await _context.Users.FindAsync(userId);
                incidentFromDb.Messages.Add(newMessage);
                await _context.SaveChangesAsync();
            }
            return _mapper.Map<IncidentResponse>(incidentFromDb);
        }
    }
    public class AssignIncidentCommand : IAssignIncidentCommand
    {
        private OutsourceDBContext _context;
        private ILogService _logService;
        private IMapper _mapper;

        public AssignIncidentCommand(OutsourceDBContext context, ILogService logService,
IMapper mapper)
        {
            _context = context;
            _logService = logService;
            _mapper = mapper;
        }
        public async Task<IncidentResponse> ExecuteAsync(int incidentId,
AssignIncidentRequest request, User user)
        {
            var incident = await _context.Incidents.FindAsync(incidentId);
            if (incident != null)
            {
                var assignedUser = await _context.Users.FindAsync(request.UserId);
                if (assignedUser != null)
                {
                    var prevUser = incident.AssignedTo;
                    incident.AssignedTo = assignedUser;
                    await _context.SaveChangesAsync();
                    string message =
                    prevUser == null
                        ? $"assigned incident {incidentId} to
{assignedUser.Name}[{assignedUser.Id}]"
                        : $"assigned incident {incidentId} to
{assignedUser.Name}[{assignedUser.Id}] from {prevUser.Name}[{prevUser.Id}]";


                    await _logService.Log(user, LogType.Incident,
SeverityLevel.Information, message);
                }
            }
```

```
            return _mapper.Map<IncidentResponse>(incident);
        }
    }
    public class ChangeIncidentStatusCommand : IChangeIncidentStatusCommand
    {
        private IMapper _mapper;
        private ILogService _logService;
        private OutsourceDBContext _context;
        private UserManager<User> _userManager;

        public ChangeIncidentStatusCommand(IMapper mapper,
            ILogService logService,
            OutsourceDBContext context,
            UserManager<User> userManager)
        {
            _mapper = mapper;
            _logService = logService;
            _context = context;
            _userManager = userManager;
        }
        public async Task<IncidentResponse> ExecuteAsync(int incidentId, User user,
ChangeIncidentStatusRequest request)
        {
            var incident = await _context.Incidents.Include(i =>
i.Messages).FirstOrDefaultAsync(i => i.Id == incidentId);
            if (incident != null)
            {
                if (!(await UserHaveRolesToChangeStatusAsync(user))
                    && !incident.AffiliatedUsers.Contains(user)
                    && incident.AssignedTo != user)
                {
                    throw new CannotEditUnrelatedIncidentException(
                        $"You cannot change incident with id {incidentId} " +
                        $"due to insufficient permissions.");
                }
                var oldStatus = Enum.GetName(typeof(IncidentStatus), incident.Status);
                var newStatus = Enum.GetName(typeof(IncidentStatus), request.Status);
                if (oldStatus != newStatus)
                {
                    _mapper.Map(request, incident);
                    await _logService.Log(user, LogType.Incident,
SeverityLevel.Information, $"Changed incident with {incidentId} status from {oldStatus}
to {newStatus}");
                    await _context.SaveChangesAsync();
                }
            }
            return _mapper.Map<IncidentResponse>(incident);
        }
        private async Task<bool> UserHaveRolesToChangeStatusAsync(User user)
        {
            var userRoles = await _userManager.GetRolesAsync(user);
            return userRoles.Contains("Admin") || userRoles.Contains("Manager");
        }
    }
   public class CreateIncidentCommand : ICreateIncidentCommand
    {
        private OutsourceDBContext _context;
        private ILogService _logService;
        private IMapper _mapper;

        public CreateIncidentCommand(OutsourceDBContext context, ILogService logService,
IMapper mapper)
        {
            _context = context;
```

```
            _logService = logService;
            _mapper = mapper;
        }
        public async Task<IncidentResponse> ExecuteAsync(CreateIncidentRequest request,
User user)
        {
            var incident = _mapper.Map<Entities.Incident>(request);
            incident.Author = user;
            await _context.Incidents.AddAsync(incident);
            await _context.SaveChangesAsync();
            await _logService.Log(user, LogType.Incident, SeverityLevel.Information,
$"Created incident id {incident.Id}");
            return _mapper.Map<IncidentResponse>(incident);
        }
    }
    public class DeleteIncidentCommand : IDeleteIncidentCommand
    {
        private IMapper _mapper;
        private ILogService _logService;
        private OutsourceDBContext _context;

        public DeleteIncidentCommand(IMapper mapper, ILogService logService,
OutsourceDBContext context)
        {
            _mapper = mapper;
            _logService = logService;
            _context = context;
        }
        public async Task ExecuteAsync(int incidentId, int userId)
        {
            Entities.Incident incidentToRemove = await
_context.Incidents.FirstOrDefaultAsync(c => c.Id == incidentId);
            var user = await _context.Users.FindAsync(userId);
            await _logService.Log(user, LogType.Incident, SeverityLevel.Warning, $"Trying
to delete incident {incidentId}.");
            if (incidentToRemove != null)
            {
                _context.Incidents.Remove(incidentToRemove);
                await _context.SaveChangesAsync();
                await _logService.Log(user, LogType.Incident, SeverityLevel.Information,
$"Deleted incident {incidentId}.");
            }
        }
    }

        private IQueryable<IncidentListResponse>
ApplyFilter(IQueryable<IncidentListResponse> query, IncidentFilter filter)
        {
            if (filter.AuthorId.HasValue)
                query = query
                    .Include(q => q.Author)
                    .Where(u => u.Author.Id == filter.AuthorId.Value);
            if (!string.IsNullOrWhiteSpace(filter.Description))
                query = query
                    .Where(q => q.Description.Contains(filter.Description));
            if (!string.IsNullOrWhiteSpace(filter.Title))
                query = query
                    .Where(q => q.Title.Contains(filter.Title));
            if (filter.Status.HasValue)
                query = query
                    .Where(q => q.Status == filter.Status.Value);
            return query;
        }
    }
```

```
    public class IncidentMappingProfile : Profile
    {
        public IncidentMappingProfile()
        {
            CreateMap<CreateIncidentRequest, Entities.Incident>()
                .ForMember(dest => dest.CreatedAt, opts => opts.ResolveUsing(v =>
DateTime.Now))
                .ForMember(dest => dest.ChangedAt, opts => opts.ResolveUsing(v =>
DateTime.Now));
            CreateMap<Entities.Incident, IncidentResponse>(MemberList.Source);
            CreateMap<ChangeIncidentStatusRequest, Entities.Incident>(MemberList.Source)
                .ForMember(dest => dest.ChangedAt, opts => opts.ResolveUsing(v =>
DateTime.Now));
            CreateMap<UpdateIncidentRequest, Entities.Incident>(MemberList.Source)
                .ForMember(dest => dest.ChangedAt, opts => opts.ResolveUsing(v =>
DateTime.Now));
            CreateMap<AddIncidentMessageRequest,
Entities.IncidentMessage>(MemberList.Source)

.ForMember(dest=>dest.ChangedAt,opts=>opts.ResolveUsing(v=>DateTime.Now));
            CreateMap<UpdateIncidentMessageRequest, Entities.IncidentMessage>()
                .ForMember(dest => dest.ChangedAt, opts => opts.ResolveUsing(v =>
DateTime.Now));
            CreateMap<ChangeIncidentStatusRequest, Entities.Incident>(MemberList.Source)
                .ForMember(dest => dest.ChangedAt, opts => opts.ResolveUsing(v =>
DateTime.Now));
            CreateMap<Entities.Incident, IncidentListResponse>()
                .ForMember(dest => dest.CompanyName, opts => opts.MapFrom(source =>
source.Company.Name))
                ;
        }
    }
    public class IncidentQuery : IIncidentQuery
    {
        private OutsourceDBContext _context;
        private UserManager<User> _userManager;
        private ILogService _logService;

        public IncidentQuery(OutsourceDBContext context, ILogService logService)
        {
            _context = context;
            _logService = logService;
        }
        public async Task<IncidentResponse> RunAsync(int incidentId, User user)
        {
            var incident = await
_context.Incidents.Include(i=>i.Messages).ProjectTo<IncidentResponse>().FirstOrDefaultAsy
nc(i => i.Id == incidentId);
            await _logService.Log(user, LogType.Incident, SeverityLevel.Information,
$"Viewed incident with id {incidentId}.");
            return incident;
        }
    }
   public class UpdateIncidentCommand : IUpdateIncidentCommand
    {
        private ILogService _logService;
        private OutsourceDBContext _context;
        private IMapper _mapper;

        public UpdateIncidentCommand(ILogService logService, OutsourceDBContext context,
IMapper mapper)
        {
            _logService = logService;
            _context = context;
```

```
            _mapper = mapper;
        }

        public async Task<IncidentResponse> ExecuteAsync(int incidentId,
UpdateIncidentRequest request, User user)
        {
            var incidentFromDb = await _context.Incidents
                .Include(i => i.Messages).FirstOrDefaultAsync(i=>i.Id == incidentId);
            if (incidentFromDb != null)
            {
                _mapper.Map(request, incidentFromDb);
                await _context.SaveChangesAsync();
            }
            return _mapper.Map<IncidentResponse>(incidentFromDb);
        }
    }
   public class UpdateIncidentMessageCommand : IUpdateIncidentMessageCommand
    {
        private OutsourceDBContext _context;
        private IMapper _mapper;
        private ILogger _logService;
        private UserManager<User> _userManager;
        private TimeSpan GRACE_PERIOD = TimeSpan.FromMinutes(180);
        public UpdateIncidentMessageCommand(
            ILogger logger,
            IMapper mapper,
            OutsourceDBContext context,
            UserManager<Entities.User> userManager
            )
        {
            _context = context;
            _mapper = mapper;
            _logService = logger;
            _userManager = userManager;
        }
        public async Task<IncidentResponse> ExecuteAsync(int incidentId, int messageId,
User user, UpdateIncidentMessageRequest request)
        {
            var incidentFromDb = await _context.Incidents
                .Include(i => i.Messages).FirstOrDefaultAsync(i => i.Id == incidentId);
            if(incidentFromDb == null||!incidentFromDb.Messages.Any(i=>i.Id ==
messageId))
            {
                return null;
            }
            var message = incidentFromDb.Messages.First(i => i.Id==messageId);
            var userRoles = await _userManager.GetRolesAsync(user);
            if (!(await UserIsAdminOrManager(user))
                && message.Author != user)
            {
                throw new CannotEditNotOwnMessagesException($"You cannot edit messsage
{messageId}, because you aren't it's owner.");
            }
            if(message.Author == user && DateTime.Now - message.ChangedAt > GRACE_PERIOD)
            {
                throw new CannotEditMessageAfterGracePeriodException($"You cannot edit
message {messageId} because it's grace period expired.");
            }
            if (request.Text != message.Text)
            {
                _mapper.Map(request, message);
                await _context.SaveChangesAsync();
            }
            return _mapper.Map<IncidentResponse>(incidentFromDb);
```

```
        }
        private async Task<bool> UserIsAdminOrManager(Entities.User user)
        {
            var userRoles = await _userManager.GetRolesAsync(user);
            return userRoles.Contains("Admin") || userRoles.Contains("Manager");
        }
    }
    public class DbInitializer
    {
        public static void Initialize(OutsourceDBContext context, UserManager<User>
userManager, RoleManager<Role> roleManager)
        {
            context.Database.EnsureCreated();
            if (!context.Roles.Any())
            {
                var roles = new string[] { "Admin", "Manager",
                "SupportEngineer", "FieldEngineer", "CallAgent" };
                foreach (var role in roles)
                {
                    roleManager.CreateAsync(new Role { Name = role }).Wait();
                }
            }
            if (!context.Users.Any())
            {
                var admin = new User { UserName = "Admin", Name="Service Administrator",
Email="Admin" };
                var adminRole = roleManager.FindByNameAsync("Admin").Result;
                userManager.CreateAsync(admin, "7Jb3V^@1C2ha&2g881").Wait();
                userManager.AddToRoleAsync(admin, adminRole.Name).Wait();
            }
        }
    }
    public static class DBServiceCollectionExtensions
    {
        public static IServiceCollection AddDB(this IServiceCollection services,
IConfiguration configuration)
        {
            services.AddDbContext<OutsourceDBContext>(options =>

options.UseSqlServer(configuration.GetConnectionString("DefaultConnection")));
            services.AddIdentity<User, Role>()
                .AddEntityFrameworkStores<OutsourceDBContext>()
                .AddDefaultTokenProviders();
            return services;
        }
    }
    public class OutsourceDBContext : IdentityDbContext<User, Role, int>
    {
        public DbSet<Company> Companies { get; set; }
        public DbSet<Incident> Incidents { get; set; }
        public DbSet<LogRecord> LogRecords { get; set; }
        public OutsourceDBContext(DbContextOptions options) : base(options)
        {
        }

        protected OutsourceDBContext()
        {
        }
    }
    public class Incident
    {
        [Key]
        public int Id { get; set; }
        [Required]
```

```
        [MaxLength(512)]
        public string Title { get; set; }
        [MaxLength(4096)]
    public class AddIncidentMessageRequest
     {
        [Required]
        [MaxLength(4096)]
        public string Text { get; set; }
     }
    public class AssignIncidentRequest
     {
        public int UserId { get; set; }
     }
    public class IncidentResponse
     {
        public int Id { get; set; }
        public string Title { get; set; }
        public UserResponse Author { get; set; }
        public string Description { get; set; }
        public CompanyResponse Company { get; set; }
        public IncidentStatus Status { get; set; }
        public ICollection<IncidentMessageResponse> Messages { get; set; }
        public ICollection<UserResponse> AffiliatedUsers { get; set; }
        public IncidentPriority Priority { get; set; }
        public UserResponse AssignedTo { get; set; }
     }
     public class UpdateIncidentMessageRequest
     {
        [Required]
        [MaxLength(4096)]
        public string Text { get; set; }
     }
     [Produces("application/json")]
     [Route("api/Incident")]
     [Authorize]
     public class IncidentController : Controller
     {
        private UserManager<User> _userManager;

        public IncidentController(UserManager<User> userManager)
        {
            _userManager = userManager;
        }
        [HttpGet]
        [ProducesResponseType(200, Type = typeof(ListResponse<IncidentListResponse>))]
        [Authorize(Roles = "Admin, Manager, FieldEngineer, SupportEngineer, CallAgent")]
        public async Task<IActionResult> GetList(ListOptions options,
            IncidentFilter filter, [FromServices]IIncidentListQuery query)
        {
            ListResponse<IncidentListResponse> response = await query.RunAsync(options,
filter);
            return Ok(response);
        }
        [Authorize(Roles = "Admin, Manager, FieldEngineer, SupportEngineer, CallAgent")]
        [HttpGet("{incidentId}", Name = "GetSingleIncident")]
        [ProducesResponseType(200, Type = typeof(IncidentResponse))]
        [ProducesResponseType(404)]
        public async Task<IActionResult> GetSingle(int incidentId,
[FromServices]IIncidentQuery query)
        {
            var user = await _userManager.GetUserAsync(HttpContext.User);
            IncidentResponse response = await query.RunAsync(incidentId, user);
            return response != null
                ? (IActionResult)Ok(response)
```

```
                : NotFound($"Incident with id {incidentId} not found.");
        }
        [HttpPost("{incidentId}/status")]
        [ProducesResponseType(200, Type = typeof(IncidentResponse))]
        [ProducesResponseType(404)]
        [ProducesResponseType(400)]
        [Authorize(Roles = "Admin, Manager, FieldEngineer, SupportEngineer, CallAgent")]
        public async Task<IActionResult> ChangeStatus(int incidentId,
            [FromBody]ChangeIncidentStatusRequest request,
            [FromServices]IChangeIncidentStatusCommand command)
        {
            if (!ModelState.IsValid)
                return BadRequest(ModelState);
            try
            {
                var user = await _userManager.GetUserAsync(HttpContext.User);
                IncidentResponse response = await command.ExecuteAsync(incidentId, user,
request);
                return response != null
                    ? (IActionResult)Ok(response)
                    : NotFound($"Incident with id {incidentId} not found.");
            }
            catch (CannotEditUnrelatedIncidentException ex)
            {
                return BadRequest(ex);
            }
        }
        [HttpDelete("{incidentId}")]
        [ProducesResponseType(204)]
        [Authorize(Roles = "Admin")]
        public async Task<IActionResult> Delete(int incidentId,
[FromServices]IDeleteIncidentCommand command)
        {
            var user = await _userManager.GetUserAsync(HttpContext.User);
            await command.ExecuteAsync(incidentId, user.Id);
            return NoContent();
        }


        [Authorize(Roles = "Admin, Manager, CallAgent")]
        [HttpPut("{incidentId}/assign")]
        [ProducesResponseType(200, Type = typeof(IncidentResponse))]
        [ProducesResponseType(404)]
        public async Task<IActionResult> Assign(int incidentId,
[FromBody]AssignIncidentRequest request, [FromServices]IAssignIncidentCommand command)
        {
            var user = await _userManager.GetUserAsync(User);
            var response = await command.ExecuteAsync(incidentId, request, user);
            return response != null
                ? (IActionResult)Ok(response)
                : NotFound($"Incident with id {incidentId} cannot be assigned");
        }
    }
#Part of FrontEnd

import React, { Component, Fragment } from 'react';
import { connect } from "react-redux";
import styled from 'styled-components';

import { Modal } from '../../common/Modal';
import { actions } from '../../store/users';
import { actions as incActions } from '../../store/incidents';

const Wrapper = styled.div`
```

```
    display: flex;
    flex-direction: column;
`
const Row = styled.div`
    display: flex;
    flex-direction: row;
    justify-content: center;
`
class Assign extends Component {
    state = {
        userId: null
    }
    onChange = event => this.setState({ userId: event.target.id.split('-')[1] });
    componentDidMount() {
    }
    componentDidUpdate(prevProps, prevState) {
        if (this.props.show && prevProps.show != this.props.show)
            this.props.onRequestData();
    }
    onConfirm = () => this.props.onAssign(this.props.incId, this.state.userId);
    render() {
        const { engineers } = this.props;
        const engineersData = engineers.length > 0
            ? engineers.map(en => (
                <Row key={en.id}>
                    <input
                        type="radio"
                        id={`radio-${en.id}`}
                        name="selectEngineer"
                        onChange={this.onChange}
                    />
                    <label htmlFor={`radio-${en.id}`}>{en.name}</label>
                </Row>
            ))
            : "no engineers in organization!"
        const data = this.props.show
            ? (
                <Modal
                    show={this.props.show}
                    onCancel={this.props.onCancel}
                    onConfirm={this.onConfirm}
                >
                    <Wrapper>{engineersData}</Wrapper>
                </Modal >
            )
            : ""
        return (
            <Fragment>
                {data}
            </Fragment>
        );
    }
}
const mapStateToProps = state => ({
    engineers: state.users.engineers,
});
const mapDispatchToProps = dispatch => ({
    onRequestData: () => dispatch(actions.getEngineers()),
    onAssign: (incId, userId) => dispatch(incActions.assignTo(incId, userId)),
});
export default connect(mapStateToProps, mapDispatchToProps)(Assign);


import React, { Component, Fragment } from 'react';
```

```
import { connect } from 'react-redux';
import { NavLink } from 'react-router-dom';
import styled from 'styled-components';

import { Spinner } from '../../common/Spinner';
import { actions, selectors } from '../../store/incidents';
import { selectors as authSelectors } from '../../store/auth';
import Toggleable from '../../common/Toggleable';
import MessageForm from './MessageForm';
import MessageList from './MessageList';
import Enums from '../../common/Enums';
import Assign from './Assign';
import ShowForRole from '../../common/ShowForRole';

const BadgeWrapper = styled.div`
display:flex;
`
const Badge = styled.div`
flex:1;
`
class Details extends Component {
    componentDidMount() {
        const { id } = this.props.match.params;
        this.props.requestData(id);
    }
    state = {
        showAssignDialog: false
    }
    getRootPath = path => path.split('/').slice(0, -2).join('/');
    editHandler = () => this.props.history.push(

`${this.getRootPath(this.props.location.pathname)}/edit/${this.props.match.params.id}`,
        { ...this.props.user });
    addCommentHandler = text => this.props.addComment(this.props.match.params.id, text);
    onAssignToggle = () => this.setState(
        prevState => ({ showAssignDialog: !prevState.showAssignDialog })
    )
    render() {
        const listPath = this.getRootPath(this.props.location.pathname);
        const { id } = this.props.match.params;
        const { email = null, name = null, phoneNumber = null } = this.props.author ||
{};
        const { address = null, name: companyName = null } = this.props.company || {};
        return (
            <Fragment>
                <NavLink to={listPath}>Go back to list</NavLink>
                {this.props.loading
                    ? <Spinner />
                    : <Fragment>
                        <h3>{this.props.title} [#{id}]
[{Enums.status[this.props.status]}] Priority: {Enums.priority[this.props.priority]}</h3>
                        <BadgeWrapper>
                            <Badge>
                                <div>Email: {email}</div>
                                <div>Name: {name}</div>
                                <div>Phone: {phoneNumber}</div>
                            </Badge>
                            <Badge>
                                <div>Company: {companyName}</div>
                                <div>Address: {address}</div>
                            </Badge>
                        </BadgeWrapper>
                        <h4>Description:</h4>
                        <div>{this.props.description}</div>
```

```
                        <button onClick={this.editHandler}>Edit</button>
                        {"    "}
                        <NavLink to="/">Delete</NavLink>

                        <ShowForRole roles={["Admin", "Manager"]}
                            user={this.props.userinfo} render={() => (<Fragment>
                                <button onClick={this.onAssignToggle}>Assign</button>
                                <Assign
                                    show={this.state.showAssignDialog}
                                    onCancel={this.onAssignToggle}
                                /></Fragment>)
                            }
                        />
                        <div>Assigned to: {this.props.assignedTo ?
this.props.assignedTo.name : "not set"}</div>
                        <hr />
                        <h4>Incident history</h4>
                        <MessageList messages={this.props.messages} />
                        <br />
                        <Toggleable
                            showText="Add comment"
                            hideText="Cancel"
                        >
                            <MessageForm
                                onSave={this.addCommentHandler}
                                canSave={!this.props.saving}
                            />
                        </Toggleable>
                    </Fragment>
                }
            </Fragment>
        );
    }
}

const mapStateToProps = state => ({
    ...selectors.getDetails(state),
    userinfo: { ...authSelectors.getUserInfo(state) },
});
const mapDispatchToProps = dispatch => ({
    requestData: id => dispatch(actions.getSingle(id)),
    addComment: (incId, text) => dispatch(actions.addComment(incId, text)),
});
export default connect(mapStateToProps, mapDispatchToProps)(Details);

import React, { Component, Fragment } from 'react';
import { connect } from "react-redux";

import Enums from '../../common/Enums';
import Textarea from '../../common/Textarea';
import { validate, validateFormInState } from '../../common/validate.js';
import { Spinner } from '../../common/Spinner';
import Input from '../../common/Input';
import Select from '../../common/Select';
import { actions, selectors } from '../../store/incidents';

class Edit extends Component {
    state = {
        title: {
            value: null,
            valid: true,
            changed: false,
            validation: {
                required: true,
```

```
                maxLength: 512
            },
        },
        description: {
            value: null,
            valid: true,
            changed: false,
            validation: {
                maxLength: 4096
            },
        },
        status: {
            value: 0,
            valid: true,
            changed: false,
            validation: {},
        },
        priority: {
            value: null,
            valid: true,
            changed: false,
            validation: {}
        },
        formIsValid: false,
        id: this.props.match.params.id,
        loading: this.props.loading,
    }
    componentDidMount() {
        if (this.props.id === null) {
            this.props.requestSingle(this.props.match.params.id);
        }
    }
    onChangeHandler = (event) => {
        const param = this.state[event.target.id];
        param.value = event.target.value;
        param.valid = validate(event.target.value, param.validation);
        param.changed = true;
        const formIsValid = validateFormInState(this.state);
        this.setState({ [event.target.id]: { ...param }, formIsValid });
    }
    cancelHandler = () => {
        this.props.history.action === "PUSH"
            ? this.props.history.goBack()
            : this.props.history.push(this.props.location.pathname.replace('edit',
'details'));
    }
    onSaveHandler = () => {
        const title = this.get("title");
        const id = +this.get("id");
        const description = this.get("description");
        const status = this.get("status");
        const priority = this.get("priority");
        this.props.onSave({ title, description, status, priority }, id);
    }
    get = valName =>
        this.state[valName].value == null ? this.props[valName] :
this.state[valName].value;
    render() {
        const title = this.get("title");
        const description = this.get("description");
        const id = this.get("id");
        const status = this.get("status");
        const priority = this.get("priority");
        return (
```

```
            this.props.loading
                ? <Spinner />
                : (
                    <Fragment>
                        <div>Edit</div>
                        <Input
                            id="title"
                            placeholder="Title"
                            value={title}
                            onChange={this.onChangeHandler}
                            valid={this.state.title.valid}
                            changed={this.state.title.changed}
                        />
                        <Input
                            id="description"
                            placeholder="description"
                            value={description}
                            onChange={this.onChangeHandler}
                            valid={this.state.description.valid}
                            changed={this.state.description.changed}
                        />
                        <Select
                            options={Enums.status}
                            valid={this.state.status.valid}
                            changed={this.state.status.changed}
                            value={status}
                            onChange={this.onChangeHandler}
                            id="status"
                        />
                        <Select
                            options={Enums.priority}
                            valid={this.state.status.valid}
                            changed={this.state.status.changed}
                            value={priority}
                            onChange={this.onChangeHandler}
                            id="priority"
                        />
                        <button disabled={!this.state.formIsValid
                            && !this.props.saving}
                            onClick={this.onSaveHandler}
                        >Save
                        </button>
                        <button onClick={this.cancelHandler}>Cancel</button>
                    </Fragment>
                )
        );
    }
}

const mapStateToProps = state => ({ userinfo: selectors.getUserInfo(state) });
export default connect(mapStateToProps)(Incident);


import React, { Fragment, Component } from 'react';
import { connect } from "react-redux";

import { actions, selectors } from '../../store/incidents';
import { selectors as authSelectors } from '../../store/auth';
import Toggleable from '../../common/Toggleable';
import { Pagination } from '../../common/Pagination';
import { Spinner } from '../../common/Spinner';
import Table from '../../common/Table';
import enums from '../../common/Enums';
import ShowForRole from '../../common/ShowForRole';
```

```
class List extends Component {
    state = {
        filter: {
            title: "",
            description: "",
            status: "",
            authorId: "",
            affiliatedUserId: ""
        }
    }
    componentDidMount() {
        this.applyFilter();
    }
    componentDidUpdate(prevProps, prevState) {
        const { page = 1 } = this.props.match.params;
        const prev = prevProps.match.params.page;
        if (prev !== undefined && prev !== page)
            this.applyFilter();
    }
    applyFilter = () => {
        const { page = 1 } = this.props.match.params;
        const { filter } = { ...this.state };
        filter.page = page;
        this.props.requestData(filter);
    }
    rowClickHandler = (id) =>
        this.props.history
            .push(`${this.props.match.path.split('/').slice(0, -
1).join('/')}/details/${id}`);
    render() {
        const { params: { page = 1 }, path } = this.props.match;
        const prefix = path.split('/').slice(0, -1).join('/');
        let data = this.props.loading
            ? <Spinner />
            : (<Table
                headers={["Id", "Title", "Company", "Status", "Assigned to", "Priority"]}
                items={this.props.incidents}
                handler={this.rowClickHandler}
                remaps={{ status: enums.status, priority: enums.priority }}
            />);
        return (
            <Fragment>
                <Toggleable
                    showText="Show filter"
                    hideText="Hide filter"
                >
                    <label htmlFor="title">Name:</label>
                    <br />
                    <input id="title" value={this.state.filter.title}
onChange={this.onChangeHandler} />
                    <br />
                    <button onClick={this.applyFilter}>apply</button>
                    <button>reset</button>
                </Toggleable>
                <ShowForRole
                    roles={["Admin", "Manager", "CallAgent"]}
                    user={this.props.userinfo}
                    render={() => (
                        <button
                            onClick={() =>
                                this.props.history
                                    .push("/incident/new")}
                        >Create new</button>
```

```
                    )}
                />
                {data}
                <Pagination
                    totalPages={this.props.pages}
                    current={page}
                    prefix={prefix}
                />
            </Fragment >
        );
    }
}
const mapStateToProps = state => ({
    ...selectors.getList(state),
    userinfo: authSelectors.getUserInfo(state),
});
const mapDispatchToProps = dispatch => ({
    requestData: filter => dispatch(actions.get(filter)),
});
export default connect(mapStateToProps, mapDispatchToProps)(List);

import React from 'react';
import styled from 'styled-components';

const Text = styled.div`
background-color: #eee;
padding: 16px;
border-radius: 16px 0;
flex:auto;
`
const Info = styled.div`
flex:initial;
display: flex;
border-right: 1px solid #ccc;
margin-right: 16px;
padding-right:16px;
align-items: center;
justify-content: center;
`
export default props => {
    if (!props.author)
        return null;
    return (
        <div style={{ display: 'flex' }}>
            <Info><div>{props.author.name}</div></Info>
            <Text>{props.text}</Text>
        </div>
    )
}

import React, { Component, Fragment } from 'react';

class MessageForm extends Component {
    state = {
        text: "",
    }
    onChangeHandler = event => this.setState({ text: event.target.value });
    onSend = () => {
        this.props.onSave && this.props.onSave(this.state.text);
        this.setState({ text: "" });
    };
    render() {
        return (
            <Fragment>
```

```
                <br />
                <textarea
                    value={this.state.text}
                    onChange={this.onChangeHandler}
                    style={{ width: '100%' }}
                />
                <br />
                <br />
                <button
                    onClick={this.onSend}
                    disabled={!this.props.canSave}
                >Send
                </button>
            </Fragment>
        )
    }
}

export default MessageForm;

import React from 'react';
import Message from './Message';

export default props => props.messages.map(message =>
    <Message
        {...message}
        key={`msg-${message.id}`}
    />);

import React, { Component, Fragment } from 'react';
import { connect } from 'react-redux';

import { validateFormInState, validate } from '../../common/validate';
import { Spinner } from '../../common/Spinner';
import Input from '../../common/Input';
import Select from '../../common/Select';
import Enums from '../../common/Enums';
import Textarea from '../../common/Textarea';
import { actions } from '../../store/incidents';
import SelectCompany from './SelectCompany';
import { Modal } from '../../common/Modal';

class New extends Component {
    state = {
        title: {
            value: "",
            valid: false,
            changed: true,
            validation: {
                required: true,
                maxLength: 512
            },
        },
        description: {
            value: "",
            valid: false,
            changed: true,
            validation: {
                maxLength: 4096
            },
        },
        priority: {
            value: 0,
            valid: true,
```

```
            changed: true,
            validation: {}
        },
        company: { id: -1, name: "" },
        formIsValid: false,
        showSelect: false
    }

    onChangeHandler = (event) => {
        const param = this.state[event.target.id];
        param.value = event.target.value;
        param.valid = validate(event.target.value, param.validation);
        param.changed = true;
        const formIsValid = validateFormInState(this.state) && this.state.company.id !==
-1;
        this.setState({ [event.target.id]: { ...param }, formIsValid });
    }
    onCancelHandler = () => {
        this.props.history.action === "PUSH"
            ? this.props.history.goBack()
            : this.props.history.push("/incident/1");
    }
    onSaveHandler = () => {
        const { title: { value: title }, description: { value: description }, priority: {
value: priority }, company: { id: companyId }
        } = this.state;
        this.props.onSave({ title, description, priority, companyId });
    }
    onSelectCompanyHandler = data => {
        this.setState({ company: { ...data }, showSelect: false })
        console.log(data);
    }
    onShowSelection = () => {
        this.setState({ showSelect: true })
    }
    render() {
        return (
            this.props.saving
                ? <Spinner />
                : (
                    <Fragment>
                        <div>Create new incident</div>
                        <div>Company: {`${this.state.company.name}   `}
                            <button onClick={this.onShowSelection}>Select</button>
                        </div>
                        <Modal show={this.state.showSelect}
                            onCancel={() => this.setState({ showSelect: false })}
                            noOk={true}
                        >
                            <SelectCompany
                                onConfirm={this.onSelectCompanyHandler}
                            />
                        </Modal>
                        <Input
                            id="title"
                            placeholder="Title"
                            value={this.state.title.value}
                            onChange={this.onChangeHandler}
                            valid={this.state.title.valid}
                            changed={this.state.title.changed}
                        />
                        <Textarea
                            id="description"
                            placeholder="description"
```

```
                            value={this.state.description.value}
                            onChange={this.onChangeHandler}
                            valid={this.state.description.valid}
                            changed={this.state.description.changed}
                        />
                        <Select
                            options={Enums.priority}
                            valid={this.state.priority.valid}
                            changed={this.state.priority.changed}
                            value={this.state.priority.value}
                            onChange={this.onChangeHandler}
                            id="priority"
                        />
                        <button disabled={!this.state.formIsValid
                            && !this.props.saving}
                            onClick={this.onSaveHandler}
                        >Save
                        </button>
                        <button onClick={this.onCancelHandler}>Cancel</button>
                    </Fragment>
                )
        );
    }
}
const mapStateToProps = state => ({ saving: state.incs.saving });
const mapDispatchToProps = dispatch => ({
    onSave: data => dispatch(actions.new(data)),
});
export default connect(mapStateToProps, mapDispatchToProps)(New);

import React, { Component, Fragment } from 'react';
import { connect } from 'react-redux';
import styled from 'styled-components';

import { selectors, actions } from '../../store/companies';
import { Pagination } from '../../common/Pagination';
import Toggleable from '../../common/Toggleable';
import Table from '../../common/Table';
import { Spinner } from '../../common/Spinner';

const ModalWrapper = styled.div`
width:80%;
`
class SelectCompany extends Component {
    state = {
        selected: {
            name: "",
            id: -1
        },
        filter: {
            inn: "",
            name: "",
        },
        currentPage: 1,
    }
    componentDidMount() {
        this.applyFilter();
    }
    componentDidUpdate(prevProps, prevState) {
        const { currentPage: page } = this.state;
        const prev = prevState.currentPage;
        if (prev !== undefined && prev !== page)
            this.applyFilter();
    }
```

```
    onChangeHandler = event => {
        const company = this.props.companies.find(c => c.id === +event.target.id);
        if (company !== undefined) {
            this.setState({ selected: { ...company } })
        }
    }
    applyFilter = () => {
        const { currentPage: page } = this.state;
        const { filter } = { ...this.state };
        filter.page = page;
        this.props.requestData(filter);
    }
    onChangePageHandler = page => {
        this.setState({ currentPage: page });
        console.log(`changed page to: ${page}`)
    }
    onSelectHandler = () => {
        this.state.selected.id != -1 && this.props.onConfirm({ ...this.state.selected })
    }
    render() {
        const { currentPage: page = 1 } = this.props;
        let data = this.props.loading
            ? <Spinner />
            : this.props.companies.map(({ name, id, inn }) =>
                <div key={`company-${id}`}>
                    <input
                        type="radio"
                        id={id}
                        name="companyselect"
                        onChange={this.onChangeHandler}
                    /><label htmlFor={id}>{name} [{inn}]</label>
                </div>
            );
        return (
            <Fragment>
                <Toggleable
                    showText="Show filter"
                    hideText="Hide filter"
                >
                    <label htmlFor="name">Name:</label>
                    <br />
                    <input id="name" value={this.state.filter.name}
onChange={this.onChangeHandler} />
                    <br />
                    <label htmlFor="inn">INN:</label>
                    <br />
                    <input id="inn" value={this.state.filter.inn}
onChange={this.onChangeHandler} />
                    <br />
                    <button onClick={this.applyFilter}>apply</button>
                    <button>reset</button>
                </Toggleable>
                {data}
                <Pagination
                    totalPages={this.props.pages}
                    current={page}
                    prefix={""}
                    handler={this.onChangePageHandler}
                />
                <button
                    onClick={this.onSelectHandler}
                >Select</button>
            </Fragment>
        );
```

```
    }
}
const mapStateToProps = state =>
    selectors.getList(state);
const mapDispatchToProps = dispatch => ({
    requestData: (filter) => dispatch(actions.get(filter)),
});
export default connect(mapStateToProps, mapDispatchToProps)(SelectCompany);
#Part of Mobile
using System;
using System.Collections.Generic;
using System.Text;
    class UserExternalLoginRequest
    {
        public string Token { get; set; }
    }
    public class IncidentItemManager
    {
        IRestService restService;

        public IncidentItemManager(IRestService service)
        {
            restService = service;
        }
        public Task<List<IncidentItem>> GetTasksAsync()
        {
            return restService.RefreshDataAsync();
        }
        public Task SaveTaskAsync(IncidentItem item, bool isNewItem = false)
        {
            return restService.SaveIncidentItemAsync(item, isNewItem);
        }
        public Task DeleteTaskAsync(IncidentItem item)
        {
            return restService.DeleteIncidentItemAsync(item.Id.ToString());
        }
    }
    public interface IRestService
    {
        Task<List<IncidentItem>> RefreshDataAsync();
        Task SaveIncidentItemAsync(IncidentItem item, bool isNewItem);
        Task DeleteIncidentItemAsync(string id);

    public class IncidentListResponse
    {
        public int Id { get; set; }
        public DateTime CreatedAt { get; set; }
        public string Title { get; set; }
        public string Description { get; set; }
        public UserResponse AssignedTo { get; set; }
        public UserResponse Author { get; set; }
        public string CompanyName { get; set; }
        public IncidentPriority Priority { get; set; }
        public IncidentStatus Status { get; set; }
    }
<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
             xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
             x:Class="OAuthNativeFlow.Views.IncidentItemPage">
    <StackLayout Margin="20" VerticalOptions="StartAndExpand">
        <Label Text="Title" />
        <Entry x:Name="TitleEntry" Text="{Binding Path=Title}" Placeholder="incident
title" />
        <Label Text="CompanyName" />
```

```
        <Entry x:Name="CompanyEntry" Text="{Binding Path=CompanyName}" />
        <Label Text="CreatedAt" />
        <Entry x:Name="CreatedAtyEntry" Text="{Binding Path=CreatedAt}" />
        <Label Text="Description" />
        <Entry x:Name="DescriptionEntry" Text="{Binding Path=Description}" />
        <Label Text="AssignedTo" />
        <Entry x:Name="AssignedToEntry" Text="{Binding Path=AssignedTo}" />
        <Label Text="Author" />
        <Entry x:Name="AuthorEntry" Text="{Binding Path=Author}" />
        <Label Text="Priority" />
        <Entry x:Name="PriorityEntry" Text="{Binding Path=Priority}" />
        <Label Text="Status" />
        <Entry x:Name="StatusEntry" Text="{Binding Path=Status}" />
        <Button Text="Save" Clicked="OnSaveActivated" />
        <Button Text="Delete" Clicked="OnDeleteActivated" />
        <Button Text="Cancel" Clicked="OnCancelActivated" />
    </StackLayout>
</ContentPage>

namespace Mobile.Views
{
    [XamlCompilation(XamlCompilationOptions.Compile)]
       public partial class IncidentItemPage : ContentPage
       {
        bool isNewItem;
        public IncidentItemPage (bool isNew = false)
              {
                      InitializeComponent ();
            isNewItem = isNew;
        }
        async void OnSaveActivated(object sender, EventArgs e)
        {
            var todoItem = (IncidentItem)BindingContext;
            await App.IncidentManager.SaveTaskAsync(todoItem, isNewItem);
            await Navigation.PopAsync();
        }
        async void OnDeleteActivated(object sender, EventArgs e)
        {
            var todoItem = (IncidentItem)BindingContext;
            await App.IncidentManager.DeleteTaskAsync(todoItem);
            await Navigation.PopAsync();
        }
        void OnCancelActivated(object sender, EventArgs e)
        {
            Navigation.PopAsync();
        }
    }
<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
             xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
             x:Class="OAuthNativeFlow.Views.IncidentListPage">
    <ContentPage.ToolbarItems>
        <ToolbarItem Text="Refresh" Clicked="Refresh">
        </ToolbarItem>
    </ContentPage.ToolbarItems>
    <ListView x:Name="listView" Margin="20" ItemSelected="OnItemSelected">
        <ListView.ItemTemplate>
            <DataTemplate>
                <ViewCell>
                    <StackLayout Padding="20,0,0,0" HorizontalOptions="StartAndExpand"
Orientation="Horizontal">
                        <Label Text="{Binding Title}" VerticalTextAlignment="Center" />
                        <Label Text="{Binding CompanyName}"
VerticalTextAlignment="Center" />
```

```
                    </StackLayout>
                </ViewCell>
            </DataTemplate>
        </ListView.ItemTemplate>
    </ListView>
</ContentPage>
       [XamlCompilation(XamlCompilationOptions.Compile)]
       public partial class IncidentListPage : ContentPage
       {
              public IncidentListPage ()
              {
                     InitializeComponent ();
            Account account =
AccountStore.Create().FindAccountsForService(Constants.AppName).FirstOrDefault();
            string tokenString = account.Properties.FirstOrDefault(t => t.Key ==
"token").Value;
            if (GetTokenExpiredDate(tokenString) < DateTime.Now)
            {
                Navigation.PushModalAsync(new OAuthNativeFlowPage());
            }
        }
        protected async override void OnAppearing()
        {
            listView.ItemsSource = await App.IncidentManager.GetTasksAsync();
        }
        protected async void Refresh()
        {
            listView.ItemsSource = await App.IncidentManager.GetTasksAsync();
        }
        void OnItemSelected(object sender, SelectedItemChangedEventArgs e)
        {
            var todoItem = e.SelectedItem as IncidentItem;
            var todoPage = new IncidentItemPage();
            todoPage.BindingContext = todoItem;
            Navigation.PushAsync(todoPage);
        }
        private DateTime GetTokenExpiredDate(string tokenString)
        {
            var token = new JwtSecurityToken(tokenString);
            double longDate = (double)Convert.ToDouble(token.Payload.Exp.ToString());
            DateTime dt = new DateTime(1970, 1, 1, 0, 0, 0);
            dt = dt.AddSeconds(longDate);
            return dt;
        }
    }
    [Activity(Label = "Mobile.Droid", Icon = "@drawable/icon", LaunchMode =
LaunchMode.SingleInstance, Theme = "@style/MyTheme", MainLauncher = true,
ConfigurationChanges = ConfigChanges.ScreenSize | ConfigChanges.Orientation)]
    public class MainActivity :
global::Xamarin.Forms.Platform.Android.FormsAppCompatActivity
    {
        protected override void OnCreate(Bundle bundle)
        {
            TabLayoutResource = Resource.Layout.Tabbar;
            ToolbarResource = Resource.Layout.Toolbar;

            base.OnCreate(bundle);

            global::Xamarin.Forms.Forms.Init(this, bundle);
       global::Xamarin.Auth.Presenters.XamarinAndroid.AuthenticationConfiguration.Init(th
is, bundle);

            LoadApplication(new App()); } }
```

Федеральное государственное автономное
образовательное учреждение
высшего образования
«СИБИРСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»
Институт космических и информационных технологий
институт
Информационных систем
кафедра

УТВЕРЖДАЮ
Заведующий кафедрой ИС
_____ Л.С. Троценко
«13» мая 2018 г.

БАКАЛАВСКАЯ РАБОТА

09.03.02 «Информационные системы и технологии»

Автоматизация внутренних бизнес-процессов компании, оказывающей услуги по обслуживанию оргтехники

Научный руководитель	13.06.18г подпись, дата	доцент, к.п.н.	С.А. Виденин
Выпускник	13.06.18г подпись, дата		О.Ю. Анучина
Нормоконтролер	13.06.18г подпись, дата		Ю.В. Шмагрис

Красноярск 2018