ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ «САРАТОВСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ ИМЕНИ ГАГАРИНА Ю.А.»

Институт прикладных информационных технологий и коммуникаций

Кафедра «Прикладные информационные технологии»

Направление 09.03.03 «Прикладная информатика»

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА

«Разработка программного обеспечения оформления европротокола участниками дорожно-транспортного происшествия»

Студент (ка) _____Шмидтке Дмитрий Евгеньевич_____
фамилия, имя, отчество

курс 4 группа б2-ПИНФ41

Руководитель
доцент каф. ПИТ, к.т.н. 11.06.20 А.В. Ермаков
должность, ученая степень, уч. звание | подпись, дата | Инициалы Фамилия

Допущен к защите
Протокол № 12 от «11» июня 2020 года

Зав. кафедрой «Прикладные информационные технологии»

кандидат технических наук, доцент 11.06.20 О.А. Торопова
ученая степень, уч. звание | подпись, дата | Инициалы Фамилия

Саратов 2020г

СОДЕРЖАНИЕ

Введение.....4
1 ОПИСАНИЕ ЗАДАЧИ И ОБЗОР СУЩЕСТВУЮЩИХ СИСТЕМ.....7
1.1 Формирование требований к системе.....8
1.1.1 Требования к системе в целом.....8
1.1.2 Требования к численности и квалификации персонала системы.....9
1.1.3 Требования к надежности.....9
1.1.4 Требования к безопасности.....10
1.1.5 Требования к эргономике и технической эстетике 10
1.1.6 Требования к эксплуатации, техническому обслуживанию, ремонту и хранению компонентов системы.....11
1.2 Обзор существующих решений.....12
1.2.1 Мобильное приложение «Помощник ОСАГО».....12
1.2.2 Мобильное приложение «ДТП Европротокол».....13
1.2.3 Мобильное приложение «WreckCheck».....15
1.3 Итоги сравнительного анализа.....16
2 ОБЗОР АРХИТЕКТУРЫ ПРИЛОЖЕНИЯ И ТЕХНОЛОГИЙ РАЗРАБОТКИ.....18
2.1 Основные понятия.....18
2.2 Ключевые особенности Android.....19
2.3 Подбор инструментов.....21
2.4 Описание предметной области посредством диаграмм IDEF.....23
2.5 Описание предметной области посредством диаграмм UML.....27
3 ОПИСАНИЕ РЕАЛИЗАЦИИ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ.....30
3.1 Реализация мобильного приложения.....30
3.1.1 Базовые принципы работы клиентской части

мобильного приложения.....30
3.1.2 Компоненты клиентской части приложения.....31
3.1.3 Активация компонентов.....37
3.1.4 Файл манифеста.....39
3.1.5 Объявление возможностей компонентов.....40
3.1.6 Объявление требований приложения.....41
3.1.7 Ресурсы приложения.....42
3.1.8 Основные экраны работы мобильного приложения 43
3.2 Реализация серверной части.....52
3.2.2 Генерация заполненного бланка европротокола..54
3.2.3 Отправление бланка европротокола на электронную почту пользователей.....56
Заключение.....59
Список использованных источников.....61
ПРИЛОЖЕНИЕ А.....66

ВВЕДЕНИЕ

С каждым днем количество автомобилей на дорогах современных городов увеличивается, а в следствии увеличивается и количество аварий, особенно в неблагоприятных погодных условиях. Это негативно влияет на дорожную ситуацию в городе, работу аварийных служб, вызывает загруженность сотрудников полиции. Отдельную проблему составляет необходимость ожидать экипаж ГИБДД для оформления происшествия. Уменьшить время оформления аварий помогает европротокол.

Европротокол — это облегченное оформление документов о дорожно-транспортном происшествии, осуществляемое без участия представителей правоохранительных органов. Европротокол оформляется методом заполнения бланка уведомления о дорожно-транспортном инциденте участниками аварии (водителями) без помощи других лиц. Это дает возможность не только значительно уменьшить время на оформление ДТП, но и быстро убрать машины с дороги, тем самым облегчив проезд другим участникам дорожного движения и минимизировав риск появления новых столкновений [1].

На сегодняшний день, по данным Российских страховых компаний, всего лишь около 40-50% аварий оформляется по европротоколу [2]. Это связанно с недостаточной степенью его нормативной проработки, неопытностью водителей в данной области. Так же в стрессовой ситуации после дорожного инцидента водителям непросто заполнить бланк извещения без

ошибок. Но использование европротокола при оформлении аварии несет в себе большое количество преимуществ, таких как:

- Экономия времени участников ДТП. В приказе МВД №185 не указан срок, за который служащий ГИБДД должен прибыть на вызов без пострадавших [3]. В связи с этим водители могут ждать по несколько часов, что приводит к ухудшению дорожной обстановки;
- Уменьшение дорожных заторов. Зачастую, дорожные заторы возникают из-за большого времени оформления аварий, большинство из которых попадают под условия оформления европротокола. Службы полиции, скорой помощи, пожарные могут потерять драгоценное время в дорожных пробках, что приведет к необратимым последствиям [4];
- Отсутствие санкций полиции. Для виновной стороны данный аспект может означать экономию заметной суммы денежных средств. А иногда позволяет избежать и более неприятных последствий нарушения.

Исходя из вышесказанного, становится очевидно, что большинства негативных последствий аварии можно избежать, уменьшив время оформления дорожно-транспортного происшествия. Помочь в этом может автоматизация процесса оформления ДТП, путем создания максимально доступного и безопасного мобильного приложения для оформления европротокола.

Целью работы является разработка мобильного приложения, которое может быть использовано участниками дорожно-транспортного происшествия для самостоятельного оформления всех необходимых документов об этом происшествии по стандарту европротокола с использованием мобильных устройств непосредственно на месте происшествия.

Для достижения цели необходимо решить следующие задачи:

1. Провести анализ предметной области и сформулировать требования к приложению.
2. Провести анализ приложений, соответствующих тематике оформления европротокола, и сделать выводы по результатам сравнительного анализа.
3. Разработать структуру мобильного приложения для оформления европротокола участниками дорожно-транспортного происшествия.
4. Реализовать мобильное приложение под Android устройства.
5. Провести тестирование разработанного мобильного приложения.

1 ОПИСАНИЕ ЗАДАЧИ И ОБЗОР СУЩЕСТВУЮЩИХ СИСТЕМ

Система для упрощенного оформления ДТП без участия сотрудников полиции должна быть предназначена для комплексного информационно-аналитического обеспечения процессов заполнения европротокола, в части исполнения следующих процессов:

- Проверка возможности составления европротокола;
- Упрощение заполнения бланков извещения о ДТП в электронном виде;
- Генерация и отправление на почту участникам ДТП заполненного бланка европротокола.

Система для упрощенного мобильного оформления ДТП без участия сотрудников полиции должна иметь возможность применяться в случаях, соответствующих следующим условиям, согласно действующему законодательству РФ:

- Дорожно-транспортное происшествие произошло в результате взаимодействия (столкновения) двух транспортных средств (включая транспортные средства с прицепами к ним), гражданская ответственность владельцев которых застрахована по ОСАГО;
- В результате дорожно-транспортного происшествия вред причинен только двум транспортным средствам (включая транспортные средства с прицепами к ним), т. е. в результате столкновения нет пострадавших, погибших и не причинен вред иному имуществу;

- Обстоятельства причинения вреда в связи с повреждением транспортных средств в результате дорожно-транспортного происшествия, характер и перечень видимых повреждений транспортных средств не вызывают разногласий участников дорожно-транспортного происшествия;
- Размер страхового возмещения, причитающегося потерпевшему в счет возмещения вреда, причиненного его транспортному средству, не может превышать 100 тысяч рублей [5].

1.1 Формирование требований к системе

1.1.1 Требования к системе в целом

Общий процесс работы приложения должен заключается в следующем:

После авторизации пользователей необходимо произвести проверку условий, необходимых для составления европротокола. Затем, во избежание мошенничества, система должна предоставить создание защищенной сессии следующего вида: один из пользователей создает QR-код, а второй пользователь, отсканировав его, подключается к сессии. После этого каждый пользователь должен иметь возможность заполнить электронный бланк извещения, включая схему ДТП и личную подпись. Далее система должна генерировать бланк извещения о ДТП установленного образца и отправлять его участникам ДТП на электронную почту. После этого, пользователям остается лишь распечатать бланк и предоставить его в страховую компанию.

Исходя из этого, можем выделить следующие требования к системе:

- В приложении происходит полное электронное заполнение европротокола, дополнительных действий от пользователя не требуется;
- Для начала оформления ДТП должна создаваться защищенная сессия между участниками аварии;
- После окончания процесса оформления ДТП, в приложении генерируется заполненный бланк европротокола, который автоматически отправляется на почту участников ДТП. Желательно автоматическое отправления европротокола в электронном виде в страховые компании;
- Приложение должно быть доступно на территории всей Российской Федерации.

1.1.2 Требования к численности и квалификации персонала системы

На начальном этапе система должна обеспечивать возможность одновременной работы с ней не менее 500 пользователей.

Минимальное количество персонала, требуемого для поддержания работы программы, должно составлять не менее 1 единицы — системный администратор.

Системный администратор должен обладать высоким уровнем квалификации и практическим опытом выполнения работ по установке, настройке и администрированию программных и технических средств, применяемых в системе.

В перечень задач, выполняемых системным администратором, должны входить: задача поддержания работоспособности технических средств серверной части; задачи поддержания работоспособности системных программных средств; задача создания резервных копий базы данных.

1.1.3 Требования к надежности

При разработке функциональных возможностей приложения должны быть предусмотрены механизмы проведения функционального и нагрузочного тестирования.

При разработке приложения должны быть учтены следующие возможные причины простоя и предусмотрены механизмы резервирования его основных функциональных компонентов как программного, так и аппаратного обеспечения:

1. Плановые отключения приложения. Позволяют выполнять профилактические мероприятия, проводить наращивание аппаратного обеспечения, выполнять установку пакетов обновлений на программное обеспечение. Плановое отключение должно проводиться в нерабочее время персоналом, обладающим требуемой квалификацией.
2. Отказы программных средств приложения. Возможность и последствия возникновения данных нарушений должны быть проанализированы для всех функциональных блоков мобильного приложения.

3. Необходимо предусмотреть механизмы протоколирования процессов информационного обмена и возможность автоматической повторной отправки запросов.

Проникновение вирусов может вызвать катастрофические последствия для всего приложения. Необходимо выработать систему защитных мер и стратегию защиты компонентов мобильного приложения антивирусного программного обеспечения.

1.1.4 Требования к безопасности

Безопасность данных системы необходимо обеспечивать следующими средствами:

1) Средства защиты данных при хранении их на сервере:
   a) Применение сессионного механизма с ограниченным временем жизни сессии.
   b) Отсутствие доступа к критически-важным данным посредствам использования сессии.

1.1.5 Требования к эргономике и технической эстетике

Пользователи должны работать посредством визуального графического интерфейса. Ввод-вывод данных, прием управляющих команд и отображение результатов их исполнения должны выполняться в интерактивном или автоматическом режимах.

Экранные формы должны проектироваться с учетом требований унификации:

1. Для обозначения одних и тех же операций должны использоваться одинаковые графические

значки, кнопки и другие управляющие (навигационные) элементы. Должны быть унифицированы термины, используемые для описания идентичных понятий, операций и действий пользователя.

2. Реакция на действия оператора (нажатие кнопки, ввод текста) должна быть типовой для каждого действия над одними и теми же графическими элементами, независимо от их расположения на экране [6].

При использовании с мобильных устройств должна быть доступна возможность перехода по системе и ввода информации при помощи пальца, ввода текстовой информации при помощи электронной клавиатуры.

1.1.6 Требования к эксплуатации, техническому обслуживанию, ремонту и хранению компонентов системы

Информационная система должна функционировать в следующих режимах:

- Штатный режим работы;
- Режим регламентного обслуживания;
- Аварийный режим.

Функции по обслуживанию и администрированию системы должны выполняться, как правило, в нерабочее время.

В штатном режиме должны выполняться все требования по функциям, реализуемым компонентами

системы. В штатном режиме функционирование системы должно быть круглосуточным, без выходных.

Режим регламентного обслуживания предназначен для проведения работ по обновлению и техническому обслуживанию компонентов системы. При работе в данном режиме допускаются перерывы в работе системы, с предварительным информированием пользователей. Регламентное обслуживание должно выполняться персоналом, обладающим требуемой квалификацией.

В аварийный режим приложение переходит в случае временной неработоспособности каналов связи, а также в случае выхода из строя аппаратного и/или программного обеспечения. В аварийном режиме осуществляется поиск неисправностей и проведение работ по их устранению. Работа пользователей с приложением в аварийном режиме невозможна.

1.2 Обзор существующих решений

1.2.1 Мобильное приложение «Помощник ОСАГО»

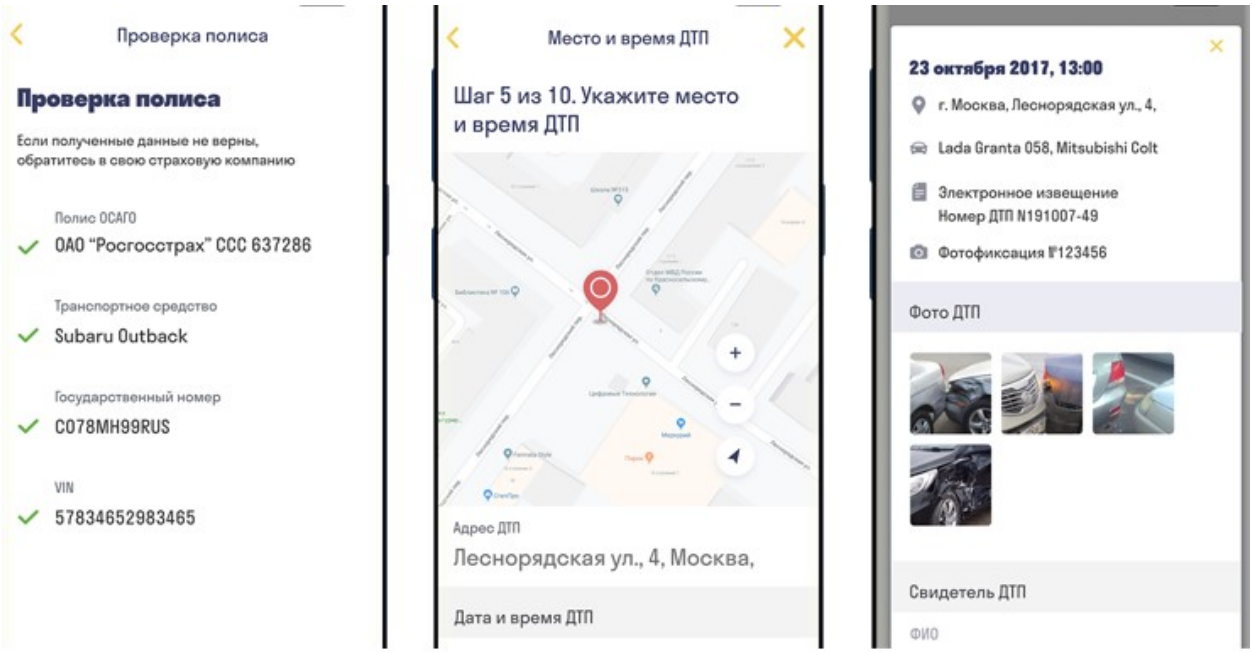

Рисунок 1.1 - Мобильное приложение «Помощник ОСАГО»

Приложение «Помощник ОСАГО» дает возможность оформить уведомление об аварии электронным документом и выслать его в информационную систему страхования, где оно станет доступно страховщику. Так же, перечень возможностей предоставленного приложения включает произведение фотосъемки места аварии. Фотографии так же передаются страховой компании посредством информационной системы. Приложением могут пользоваться только люди, имеющие подтвержденную учетную запись на портале «Госуслуги». Без этого работа приложения невозможна [7].

Данное приложение не соответствует всем требованиям, выдвинутых к нашему приложению, например

- Недоступно мобильное оформление схемы ДТП, ее необходимо фиксировать на листке бумаги;

- Второй участник ДТП не имеет возможности скорректировать информацию в полученной им заполненной форме;
- Авторизация возможна только при наличии учетной записи на портале «ГосУслуги»;
- Оформление извещения о ДТП в виде электронного документа возможно только в том случае, если ДТП произошло на территориях Республики Татарстан, городов федерального значения Москвы, Санкт-Петербурга, Московской области или Ленинградской области.

1.2.2 Мобильное приложение «ДТП Европротокол»

Приложение помогает обеспечить фиксирование и передачу в АИС ОСАГО данных о дорожно-транспортном происшествии для задач обращения пострадавшего в организацию сферы страхования в случае составления протокола без участия уполномоченных на то представителей правоохранительных органов в вариантах, когда у участвующих ДТП есть противоречия в отношении инцидента (в данном случае макс. выплата 100 тыс.руб.), или когда противоречий нет и участники оформляют так именуемый «безлимитный Европротокол».

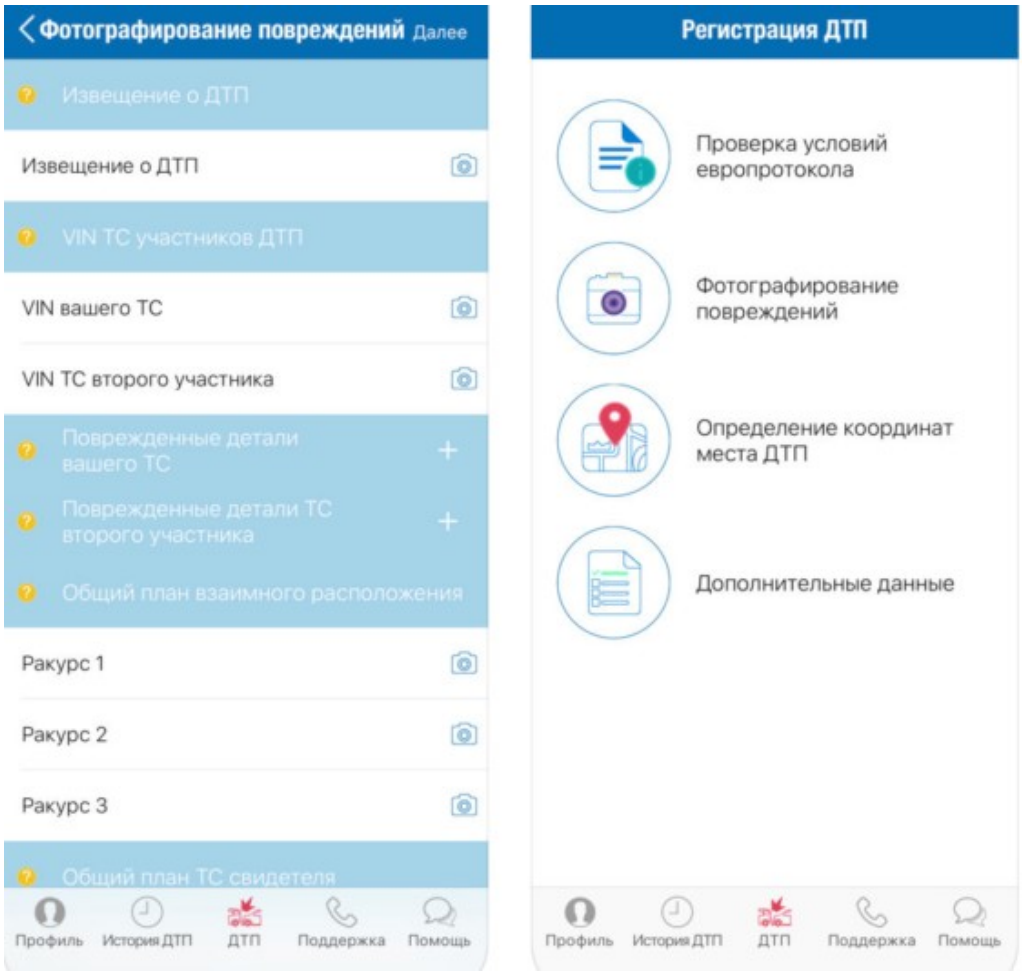

Рисунок 1.2 - Мобильное приложение «ДТП Европротокол»

После установки этого приложения на телефон либо планшет, снаряженный фотокамерой, нужно выполнить авторизацию через портал «Госуслуги» (нужна подтвержденная учетная запись). Дальше делается фотосъемка ТС и их повреждений на месте ДТП, считывание координат места ДТП и время ДТП на базе использования сигналов глобальных спутниковых навигационных систем либо базисных станций мобильной связи и передача приобретенных фотоизображений с дополнительной сведениями о дорожно-транспортном происшествии через приложение в АИС ОСАГО, из которой

они могут быть получены страховщиками для изучения обращения пострадавшего о страховом возмещении, в порядке, который был установлен нормативно-правовыми актами России. Без авторизации через портал «Госуслуги» приложение предоставляет возможность проверки полиса страхования автогражданской ответственности, методом сканирования QR-кода, также методом ручного ввода номера и серии полиса [8].

Мобильное приложение «ДТП Европротокол» так же удовлетворяет лишь некоторым требованиям нашего приложения. В нем не выполняются такие требования, как:

- Полное заполнение европротокола в электронном виде, без дополнительных требований от пользователя;
- Создание защищенной сессии между участниками аварии посредством QR-кода;
- Генерация заполненного бланка европротокола после окончания процесса оформления ДТП и автоматическая отправка его на почту участникам аварии.

1.2.3 Мобильное приложение «WreckCheck»

Рисунок 1.3 - Мобильное приложение «WreckCheck»

Мобильное приложение WreckCheck от NAIC обрисовывает в общих чертах, что делать немедленно после автомобильной аварии, и пошагово продвигает пользователей по созданию их собственного отчета об аварии. В нем также содержатся советы о том, как оставаться спокойным в дороге, и позволяет легко делать фотографии и документировать необходимую информацию для подачи страхового возмещения. Кроме того, приложение позволяет пользователям отправлять по электронной почте заполненный отчет об аварии непосредственно им и их страховым агентам.

Данное приложение на текущий момент поддерживает работу только в США [9].

Приложение «WreckCheck» не соответствует многим требованиям, предъявленных к нашему приложению, а именно:

- Создание защищенной сессии между участниками аварии посредством QR-кода;
- Генерация заполненного бланка европротокола после окончания процесса оформления ДТП и автоматическая отправка его на почту участникам аварии;
- Приложение должно быть доступно на территории всей Российской Федерации;
- Авторизация в приложение производится с помощью полиса ОСАГО, желателен вариант авторизации посредством сервиса «ГосУслуги»;
- Отсутствует возможность описывать схему ДТП.

1.3 Итоги сравнительного анализа

В результате анализа предметной области и обзора существующих решений нам удалось поставить требования и найти ряд программных продуктов предназначенных для решения указанной задачи.

Однако же, те решения, что были найдены, не удовлетворяют требованиям поставленным нами при обзоре предметной области. В частности наиболее важно что:

- В приложении «Помощник ОСАГО» у пользователя нет возможности полностью произвести оформление ДТП через мобильное приложение, некоторые детали придется фиксировать на бумаге;

- В приложении «ДТП Европротокол» необходима система мониторинга на автомобиле, что существенно снижает количество людей, которые могут воспользоваться этим приложением для оформления ДТП;
- Приложение WreckCheck в целом не доступно на территории Российской Федерации и не соответствует её задачам.

На основании произведенного анализа можно сделать вывод, о необходимости самостоятельной разработки приложения соответствующего поставленным требованиям.

2 ОБЗОР АРХИТЕКТУРЫ ПРИЛОЖЕНИЯ И ТЕХНОЛОГИЙ РАЗРАБОТКИ

Мобильные устройства это очень важная часть жизни современного человека. Они позволяют налаживать коммуникации между людьми, слушать музыку, просматривать видеоинформацию, играть в игры, и несут в себе еще множество различных функций. Мобильный телефон это своеобразная копия компьютера которую постоянно можно иметь при себе.

На текущий момент, самая распространенная операционная система на смартфонах -Андройд. ОС андройд поддерживает огромное число устройств, от различных производителей. Основная причина распространения ОС Android - это бесплатные средства разработки, тогда как разработка под систему IOS нуждается в больших исходных издержках.

2.1 Основные понятия

Android — операционная система для телефонов, веб-планшетов, цифровых книг, цифровых проигрывателей, наручных часов, игровых нетбуков, смартбуков, очков Гугл, телевизоров, а так же многих других устройств [10].

Android SDK — среда разработки приложений для операционной системы Android [10].

Сериализация (в программировании) — процесс перевода какой- либо структуры данных в последовательность битов [11].

JSON (JavaScriptObjectNotation) — простой формат обмена данными, удобный для чтения и написания, как

человеком, так и компьютером [32].

Java—представляет собой язык программирования и платформу вычислений, которая была впервые выпущена SunMicrosystems в 1995 году [10].

Open JDK - проект по созданию полностью совместимого Java Development Kit, состоящего исключительно из свободного и открытого исходного кода, включающий в себя компилятор Java (javac), стандартные библиотеки классов Java, примеры, документацию, различные утилиты и исполнительную систему Java (JRE) [12].

API (интерфейс программирования приложений, интерфейс прикладного программирования) (англ. application programming interface) — набор готовых классов, процедур, функций, структур и констант, которые предоставляются приложением (библиотекой, обслуживанием) либо операционной системой для использования во внешних программных продуктах. Применяется разработчиками ПО при написании различных приложений.

2.2 Ключевые особенности Android

ОС Android базирована на ядре линукс и реализации виртуальной машины Java от Google. Вначале разрабатывалась фирмой AndroidInc., которую в 2005 году приобрела Google. Затем Google инициировала создание альянса OpenHandsetAlliance (OHA), который в текущее время занимается помощью и развитием площадки. Андройд позволяет создавать Java-приложения, которые

управляют устройством посредством созданных Google библиотек. AndroidNativeDevelopmentKit позволяет портировать (но не отлаживать) библиотеки и элементы приложений, которые были написаны на Си и других языках. ОС Андройд установлена на 86% телефонах на 2014 год [13].

Андройд является широко распространенной операционной системой (ОС) для мобильных устройств – телефонов и планшетов. Эта система имеет огромное количество отличительных черт, делающих ее узнаваемой и удобной для огромного количества пользователей во всем мире. Операционная система Андройд является неприхотливой и способна работать на различных конфигурациях. Благодаря этому большая часть крупных производителей используют на своих устройствах эту ОС. Одним из основных преимуществ системы андройд, является то, что система основана на ядре линукс, которая имеет открытый программный код, что дает неограниченные возможности создателям приложений.

Андройд быть может запущен на устройствах, которые имеют размер оперативной памяти меньше 256 Мб. Более новые версии системы требуют 512 Мб оперативной памяти, что также является небольшим значением для современных аппаратов. Система не требует наличия высокопроизводительного микропроцессора и может работать на устройствах, снаряженных ядром с частотой 600 МГц. Операционная система дает возможность установки приложений с официального репозитория

Google, предоставляющего наибольшую в мире базу программ. Это связанно с тем, что каждый создатель приложений может без помощи других написать ПО для телефона и выложить его в магазине.

Необходимо отметить, что приложения на устройствах под управлением Андройд могут быть установлены как конкретно с телефона либо планшетного компьютера, так и через ПК, методом загрузки файла .apk и его следующей установки на аппарате. Отличительной индивидуальностью Андройд является его интегрированность с сервисами Google – Gmail, Hangouts, Voice Search и т.п. На Андройд в официальном порядке реализована поддержка Chrome, что позволяет синхронизировать открываемые в браузере вкладки на телефоне с компьютерным браузером. Система имеет интуитивно понятный интерфейс. Все нужные приложения располагаются сразу на главном экране и в меню аппарата, вызываемое нажатием на центральную сенсорную кнопку либо клавишу на дисплее. Все опции размещаются в секции «Опции», а каждое действие пользователя поясняется объяснениями и подсказками при первом запуске устройства. Операционная система стремительно реагирует на нажатия пользователя и производит установку и загрузку подходящих программ и файлов со скоростью, которая не проигрывает другим передовым мобильным ОС [14].

2.3 Подбор инструментов

Существует несколько самых распространенных операционных систем для смартфонов, такие как, IOS,

Android, WindowsPhone, BlackBerry, Simbian. Для создания приложения выбран Android, потому что:

- Android – операционная система с открытым исходным кодом;
- Распространенность ОС Android, показано на рисунке 2.1 [14];
- Доступ к разработке любому пользователю;
- Абсолютно бесплатная для разработки.

Рисунок 2.1 - Доля устройств на различных ОС

Для создания приложения на ОС Андройд, могут быть выбраны разные среды разработки, например, Eclipse, Embarcadero JBuilder, JDeveloper и т.д, но для работы выбрана AndroidStudio от компании Google. Предпосылкой выбора стала удобность интерфейса и быстродействие программы. Андройд Studio — это встроенная среда разработки (IDE) для работы с площадкой Андройд, которая была анонсирована 16 мая 2013 года на мероприятии Google I/O [15].

Среда разработки находилась в свободном доступе начиная с версии 0.1, размещенной в мае 2013, а потом

перешла в стадию проведения тестов, начиная с версии 0.8, выпущенной в июне 2014 года. Стабильная версия 1.0 была выпущена в декабре 2014 года, тогда же закончилась поддержка плагина Android Development Tools (ADT) для Eclipse. AndroidStudio, основана на программном обеспечении IntelliJ IDEA от компании JetBrains, официальное средство разработки Андройд приложений. Эта среда разработки доступна для Windows, OS X и линукс.

За счёт своей лаконичности по сравнению с XML, формат JSON может быть более подходящим для сериализации сложных структур. Он хорошо подходит в задачах обмена данными как между браузером и сервером (AJAX), так и между самими серверами (программные HTTP-интерфейсы). Поскольку формат JSON является подмножеством синтаксиса языка JavaScript, то он может быть быстро десериализован встроенной функцией eval(). Кроме того, возможна вставка вполне работоспособных JavaScript-функций.

Технология Java включает в себя средства трансляции начального текста программы - исходного кода - в специальную форму, подходящую для выполнения компьютером, и средства исполнения Java-программ на различных платформах, другими словами в разных операционных системах и на различном аппаратном обеспечении. Главное преимущество Java -технологии в том, что преобразованная на этапе трансляции в особый код Java-программа полностью "машинонезависима". Если

исполняемый код, полученный из программ на других распространенных языках, обычно не подходит для выполнения на компьютере "иной платформы", то к исполняемому коду Java подобное ограничение не относится. При этом, необходимо, чтобы для "целевой платформы" имелась реализация так называемой Java-машины - среды выполнения Java –программ [33].

Java -технологии, получили обширное распространение, а платформонезависимость Java, позволившая интегрировать средства выполнения Java-программ в браузеры, которые работают в самых различных операционных системах, определила распространение Java в качестве элемента Web-технологий. Java используется в разработке сложных интерактивных частей, которые связаны с веб-сайтами. К примеру, на Java реализуются сложные инструменты для работы с базами данных, размещенными в Web. Либо графические интерфейсы, которые требуют вывода сложных интерактивных частей [16].

2.4 Описание предметной области посредством диаграмм IDEF

Представление модели предметной области на основе графических языков позволяет сделать работу более информативной. Функционирование любой системы удобно описывать на основе нотации методологии структурного анализа SADT. Используемый стандарт описания IDEF0 позволяет строить иерархию диаграмм. IDEF0 – модель

предназначена для описания функционирования организации на основе подхода «от общего к частному». Такие модели считаются наиболее наглядными, причем для их построения не требуется сложных построений.

Методология IDEF0 широко используется благодаря простой и понятной для понимания графической нотации, применение которой для построения модели очень удобно [17]. На диаграммах отображаются функции системы. На рисунке 2.2 изображена начальная контекстная диаграмма системы [18].

На данной диаграмме мы можем увидеть, что входными данными системы являются:

- Данные о бланках извещения, в соответствии с установленными регламентом образцами европротокола;
- Пользовательские данные – персональные данные, которые вводит пользователь во время работы приложения или данные которые поступают в приложение в результате работы другого сервиса, например «ГосУслуги». К таким данным относится: номер полиса ОСАГО, данные водительского удостоверения, автомобильные данные и т.д;
- Данные об аварии. Заполняются пользователем непосредственно в приложении. К таким данным относится: время и место аварии, схема ДТП, как двигались автомобили перед возникновением аварии и т.д.

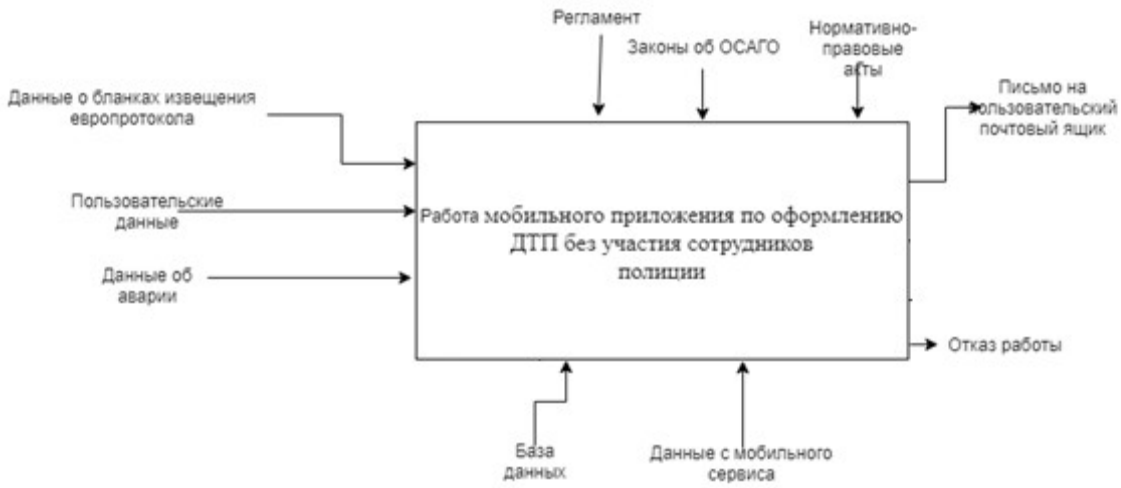

Рисунок 2.2 - Начальная контекстная диаграмма

Выходными данными системы является письмо на пользовательский почтовый ящик. После оформления европротокола каждый из пользователей получает письмо на почтовый ящик, в котором содержится заполненный бланк извещения о ДТП.

На рисунке 2.3 изображены бизнес-процессы работы мобильного приложения.

На схеме наглядно видно, на каком этапе какие управляющие элементы и какие механизмы задействованы. Так, на первом шаге происходит проверка условий для составления европротокола, исходя из полученных пользовательских данных, которая регламентируется законами об ОСАГО.

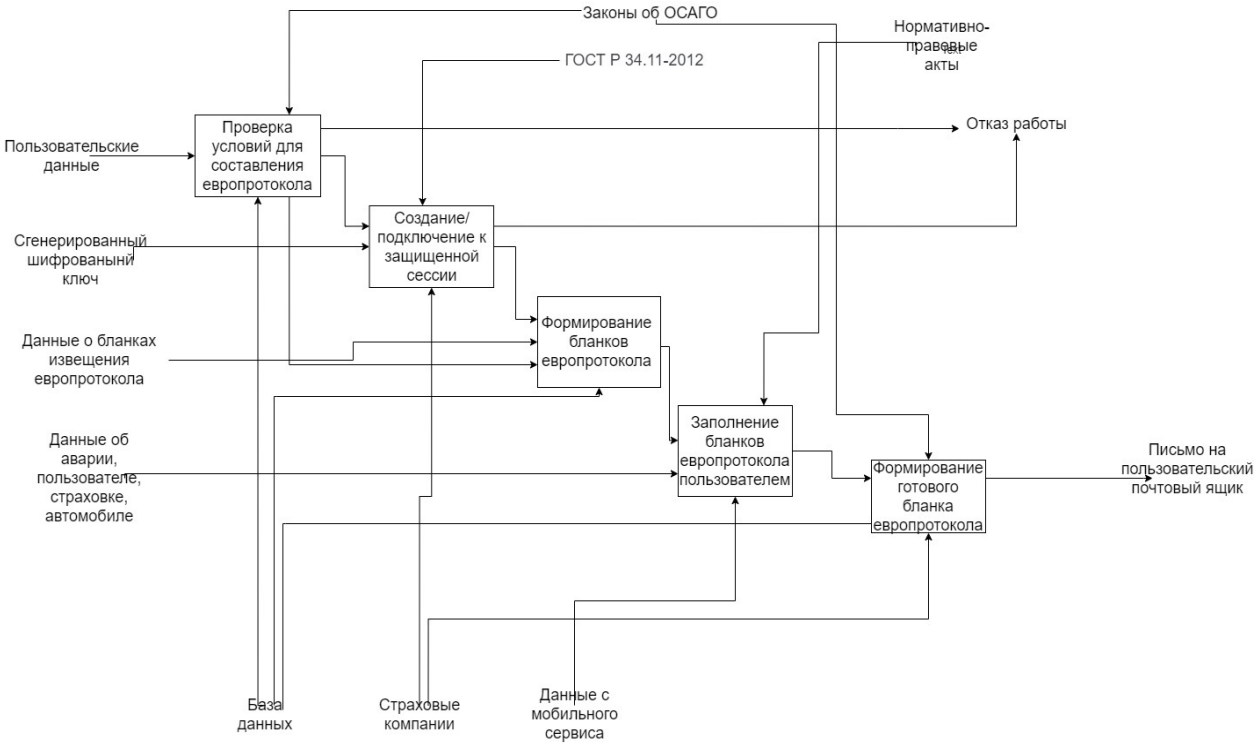

Рисунок 2.3 - IDEF0 диаграмма мобильного приложения

Далее происходит создание или подключение к защищенной сессии с использованием сгенерированного шифрованного ключа, которая определяется ГОСТ Р 34.11-2012, в котором описывается алгоритм и процедуры вычисления хэш-функции для любой последовательности двоичных символов, которые применяются в криптографических методах защиты информации.

После этого происходит формирование бланков европротокола, на основании данных о бланках извещения европротокола из базы данных. Далее с использованием данных об аварии, пользователе, страховке и автомобиле осуществляется заполнение бланков европротокола пользователем. Это заполнение регламентируется нормативно-правовыми актами об ОСАГО. После этого начинается процесс формирования готового бланка европротокола, который получает на вход заполненные

пользователем данные о европротоколе и на выходе отправляет письмо на пользовательский почтовый ящик.

IDEF3 -методология моделирования и стандарт документирования процессов, происходящих в системе. IDEF3 широко применяется при разработке информационных систем. При этом используется инструмент визуального моделирования бизнес-процессов [22]. Для процесса оформления европротокола с помощью мобильного приложения, построена диаграмма семейства IDEF3 и представлена на рисунке 2.4.

На диаграмме представлено описание технологических процессов информационный системы, с указанием того, что происходит на каждом этапе процесса. На диаграмме присутствуют разворачивающие потоки данных соединения, предназначенные для завершения одного действия и вызова начала выполнения других. А также сворачивающие, соединения выполняющие обратные действия [19].

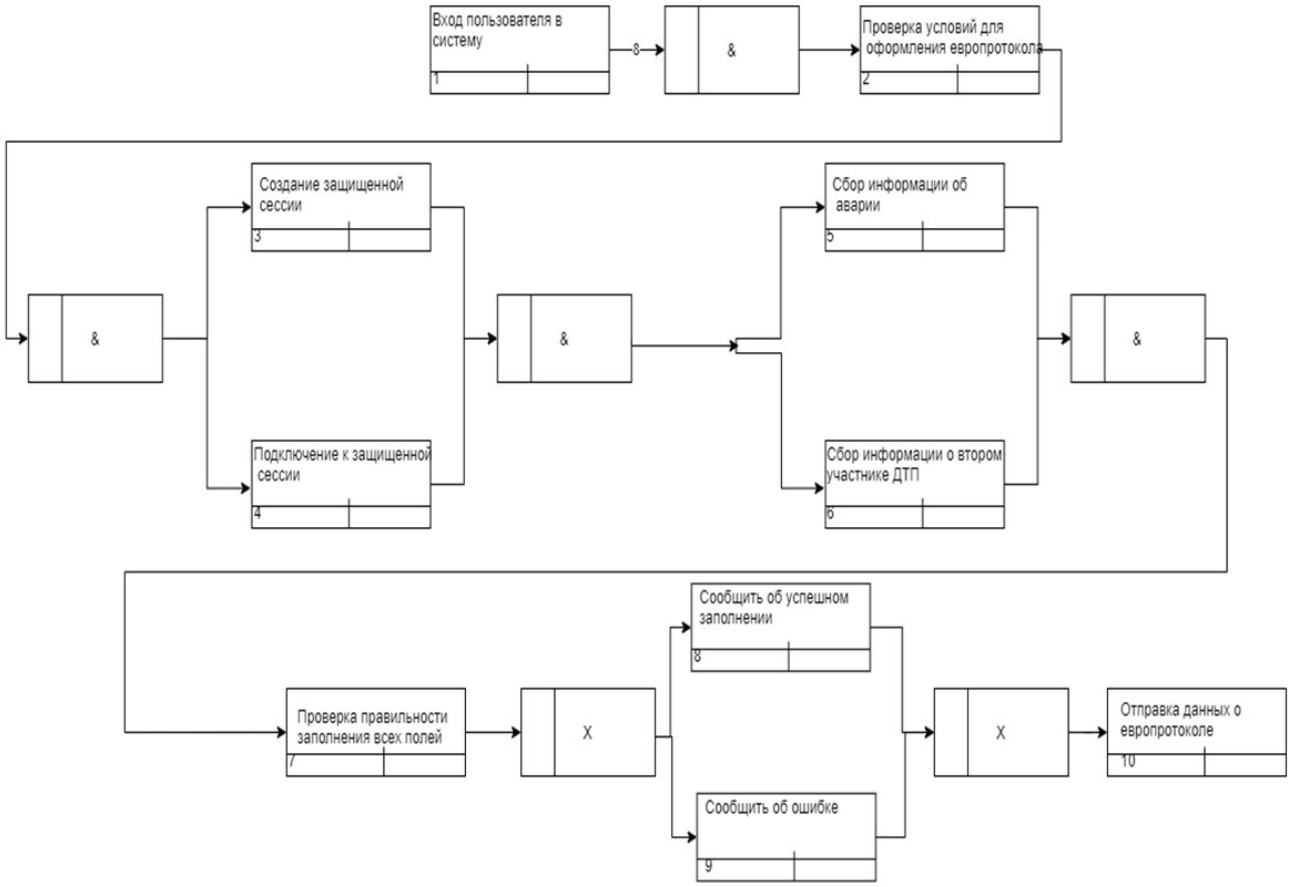

Рисунок 2.4 - Диаграмма IDEF3 для оформления европротокола

Работа системы начинается со входа пользователя в систему. После этого начинается процесс проверки условий возможности оформления европротокола. Далее параллельно происходит создание и подключение к защищенной сессии и сбор информации об аварии от двух пользователей. Затем, на седьмом шаге, системой осуществляется проверка правильности заполнения всех полей, в результате которой пользователю сообщается об успешном заполнении и происходит отправка данных о европротоколе или сообщается об ошибке.

2.5 Описание предметной области посредством диаграмм UML

Для корректного дополнения описания предметной области используется унифицированный язык моделирования UML. Описание производится с помощью диаграмм прецедентов, диаграмм классов, а также диаграмм последовательности и других менее популярных диаграмм. Благодаря подробному описанию предметной области средствами UML появляются все условия для создания мобильного приложение, которое будет удовлетворять требованиям заказчика [20].

Моделирование предметной области с помощью языка UML включает в себя концептуальную и логическую диаграммы вариантов использования UML [21]. На рисунке 2.5 представлена концептуальная диаграмма вариантов использования UML.

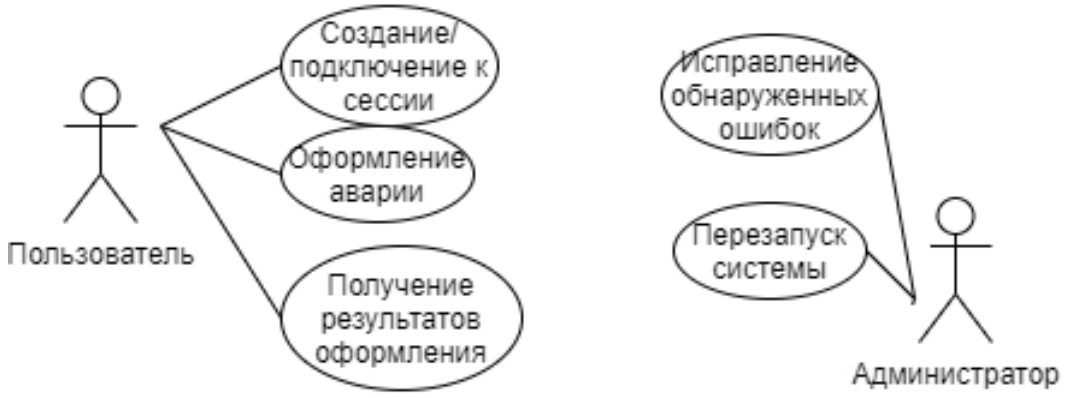

Рисунок 2.5 - Концептуальная диаграмма вариантов использования

Диаграмма на рисунке 2.5 описывает сценарии включающая такие действующие лица, как администратор информационной системы и пользователь.

Для более детального описания предметной области была составлена диаграмма последовательностей,

представляющая собой описание жизненного цикла основных объектов системы [22].

На диаграмме последовательности большая часть объектов является экземплярами класса или сущности, которые обладают поведением. В роли объектов могут выступать пользователи, инициирующие взаимодействие, классы, обладающие поведением в системе или программные компоненты, в редких случаях, системы в целом.

Объекты располагаются слева направо таким образом, чтобы крайним слева был тот объект, который инициирует взаимодействие.

Важной частью объекта на диаграмме последовательности является линия жизни объекта. Линия жизни показывает время, в течение которого объект существует в Системе. Периоды активности объекта в момент взаимодействия показываются с помощью фокуса управления, который отображается в виде прямоугольника на линии жизни объекта. Временная шкала на диаграмме направлена сверху вниз. Диаграмма последовательностей представлена на рисунке 2.6.

Из данной диаграммы видно, что основными участниками являются пользователи приложения. Они взаимодействуют с классами-разграничителями, такими, как форма проверки условий ДТП и форма данных об аварии посредством ввода данных и заполнения форм. Те, в свою очередь, взаимодействуют с активными элементами системы – классами контроллерами, для выполнения

операций над объектами, например создание шифрованного ключа, проверка правильности заполнения полей и т.д.. На заключительном этапе работы системы класс-контроллер, выполняющий формирование европротокола отправляет документ такому участнику процесса, как электронная почта пользователя, что и является завершающим этапом жизненного цикла приложения.

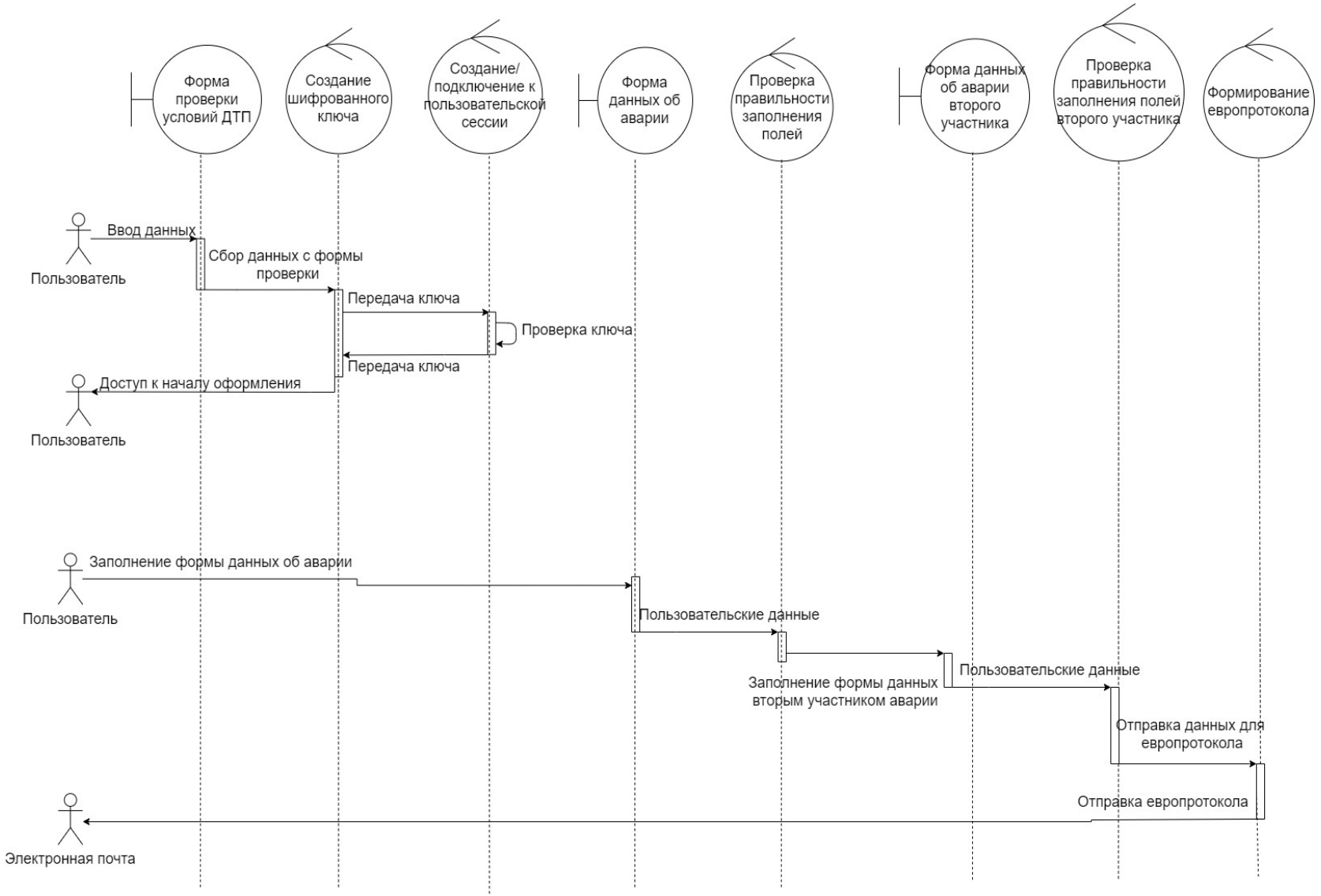
Форма проверки условий ДТП
Создание шифрованного ключа
Создание/ подключение к пользовательской сессии
Форма данных об аварии
Проверка правильности заполнения полей
Форма данных об аварии второго участника
Проверка правильности заполнения полей второго участника
Формирование европротокола
Ввод данных
Пользователь
Сбор данных с формы проверки
Передача ключа
Проверка ключа
Передача ключа
Доступ к началу оформления
Пользователь
Заполнение формы данных об аварии
Пользователь
Пользовательские данные
Заполнение формы данных вторым участником аварии
Пользовательские данные
Отправка данных для европротокола
Отправка европротокола
Электронная почта

Рисунок 2.6 - Диаграмма последовательностей для мобильного приложения

3 ОПИСАНИЕ РЕАЛИЗАЦИИ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

3.1 Реализация мобильного приложения

3.1.1 Базовые принципы работы клиентской части мобильного приложения

При создании мобильного приложения использовался язык программирования Java. С помощью инструментов андройд SDK(Software Development Kit) написанный код вместе со всеми нужными файлами компилируется в программный пакет андройда – файл .apk. Данный файл представляет собой файл архива. В нем находится все, что необходимо для работы мобильного приложения, а так же он позволяет установить данное приложение на любом устройстве, работающем на ОС андройд.

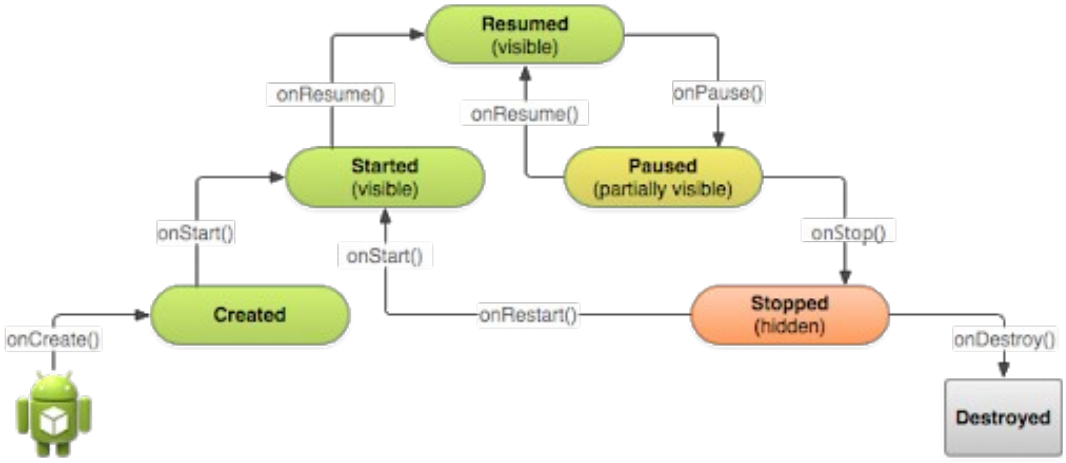

Рисунок 3.1 - Жизненный цикл приложения

Операционная система Андройд представляет из себя многопользовательскую систему линукс, в которой каждое приложение является некоторым пользователем. По стандарту, система определяет каждому приложению особый идентификатор пользователя который используется

лишь системой и известен приложению. Система устанавливает ограничения для всех файлов в приложении, так, чтобы они были доступны только лишь пользователю с идентификатором, который назначен данному приложению. Код приложения запускается изолированно от остальных приложений каждый процесс имеет собственную виртуальную машину (ВМ). По стандарту, каждое приложение запускается в своем процессе. Система запускает процесс тогда, когда требуется выполнить какой-нибудь элемент приложения, после чего заканчивает процесс, если он больше не требуется или если системе требуется высвободить память для остальных приложений. Таким образом, в системе реализуется так называемый «принцип предоставления малых прав», то есть каждому приложению доступны лишь те элементы, которые ему необходимы для работы, и никакие другие. Благодаря этому в системе формируется безопасная среда, где у приложения нет доступа к тем областям системы, которые ему не требуются. Но у приложения есть варианты предоставления собственных данных иным приложениям и доступа к системным службам:

- При необходимости доступа одного приложения к другому, им можно назначить единый идентификатор пользователя. Тогда любая из них сумеет обращаться к файлам другого приложения. В целях экономии ресурсов, зачастую приложения со схожими идентификаторами пользователя запускаются в одном процессе, при этом они используют одну виртуальную

машину. Так же оба этих приложения должны быть подписаны единым сертификатом;

- Если приложению необходимо использовать данные пользователя, оно может запросить доступ к таким данным, например, к контактам пользователя, смс-сообщениям, подключаемой карте памяти (SD-карте), камере, блютуз и др. Все необходимые разрешения выдаются приложению при его установке [23].

3.1.2 Компоненты клиентской части приложения

Мобильное приложение состоит из блоков, они являются некими точками входа системы в приложение. Но далеко не все составляющие приложения представляют точки входя для пользователя, некоторые из них зависимы друг от друга. Так же, каждый из элементов является самостоятельной структурой и выполняет свою роль. Любой из элементов представляет собой особый элемент структуры, который в общем определяет работу мобильного приложения [24].

Все компоненты приложения «Европротокол онлайн» можно отнести к одному из 4 типов. Составляющие каждого типа созданы для определенной цели, они имеют свой актуальный цикл, определяющий метод начала и окончания существования элемента. Составляющие бывают последующих типов:

Операции

Операция (Activity) - это элемент приложения, выдающий экран, и с которым пользователи могут вести взаимодействие для реализации каких-то действий, к примеру ввести сведения о машине, отсканировать QR-код. Каждой операции присваивается окно для прорисовки соответствующего пользовательского интерфейса. Обычно окно отображается во весь экран, но его размер быть может меньше, и оно может располагаться поверх остальных окон.

Приложение состоит из нескольких операций, которые слабо соединены вместе. Одна из операций в приложении обозначена как «главная», которая предлагается пользователю при первом запуске приложения. При этом, любая операция может запустить другую операцию для реализации разных действий. Всякий раз, когда запускается новая операция, предшествующая операция останавливается, но система хранит ее в стеке («стек переходов назад»). При старте новой операции она помещается в стек переходов назад и новая операция отображается для пользователя. Стек переходов назад работает по принципу «последним вошёл — первым вышел», потому после того как пользователь окончил текущую операцию и нажал клавишу «Назад», текущая операция удаляется из стека (и уничтожается), и возобновляется предшествующая операция. Когда операция останавливается из-за пуска новой операции, для извещения об изменении ее состояния используются способы оборотного вызова актуального цикла операции. Существует несколько способов, которые может принимать

операция в результате изменения собственного состояния — создание операции, ее остановка, восстановление либо уничтожение системой [25];

Также каждый обратный вызов представляет возможность выполнить определенное действие, подходящее для изменения состояния. К примеру, в случае остановки операция обязана освободить любые крупные объекты, к примеру, подключение к сети. При восстановлении операции можно повторно получить нужные ресурсы и осуществить возобновление прерванных действий. Подобные изменения состояния являются частью жизненного цикла операции (рисунок 3.2).

Службы

Служба (Service) является элементом приложения, который может выполнять продолжительные операции в фоновом режиме и в нем отсутствует пользовательский интерфейс. Другой компонент приложения имеет возможность запустить службу, которая будет продолжать работу в фоновом режиме, даже в случае, когда пользователь перейдет в другое приложение. Также, компонент может привязаться к службе для взаимодействия с ней и даже выполнять межпроцессное взаимодействие (IPC). К примеру, служба может обрабатывать сетевые транзакции, осуществлять ввод-вывод файла либо вести взаимодействие с поставщиком информации в фоновом режиме.

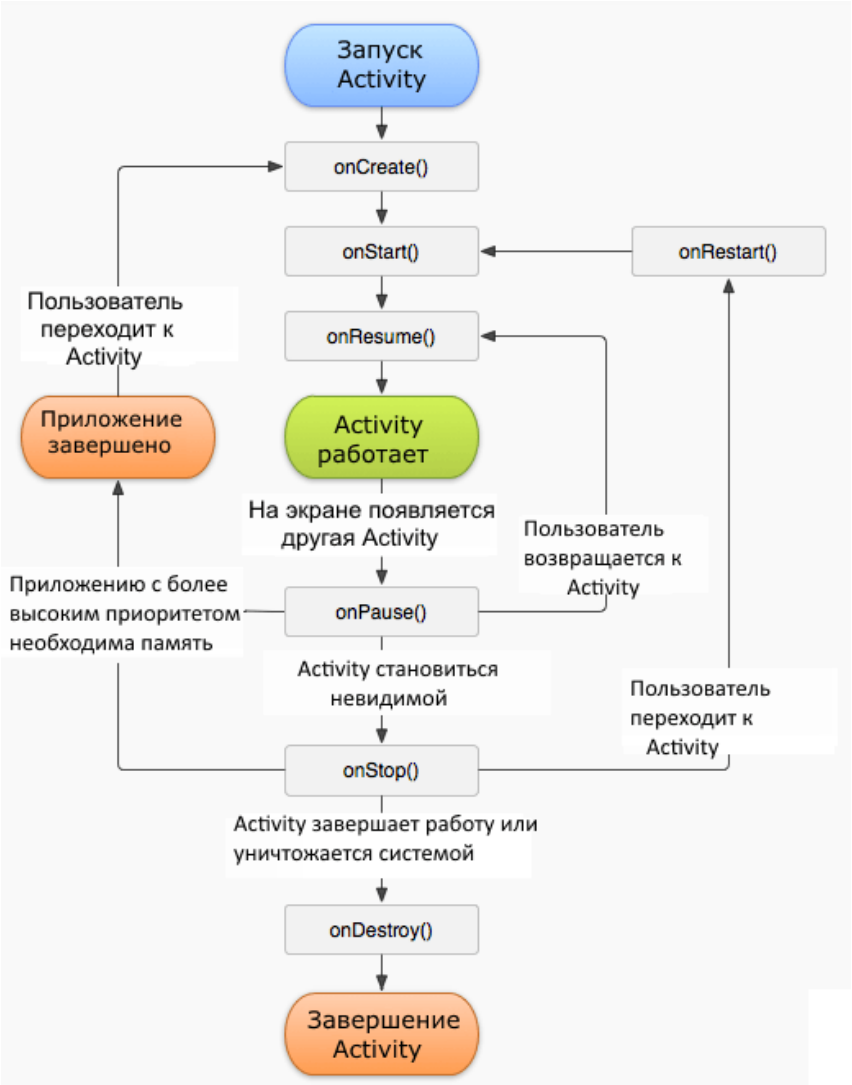

Рисунок 3.2 - Жизненный цикл операции

Практически служба может принимать две формы:

- Запущенная. Служба является «запущенной», когда элемент приложения (к примеру, операция) запускает ее вызовом startService(). После пуска служба может осуществлять работу в фоновом режиме в течение неограниченного времени, даже в том случае, если уничтожен элемент, который ее запустил. Обычно запущенная служба делает одну операцию и не возвращает результатов вызывающему элементу. К примеру, она может загружать либо выгружать файл по сети. Когда операция выполнена, служба обязана

остановить выполнение без помощи других служб;

- Привязанная. Служба является «привязанной», когда компонент приложения привязывается к ней вызовом bindService(). Привязанная служба возвращает интерфейс клиент-сервер, позволяющий компонентам вести взаимодействие со службой, отправлять запросы, получать результаты и даже делать это среди различных действий, с помощью межпроцессного взаимодействия (IPC). Привязанная служба работает лишь пока к ней привязан еще один компонент приложения. К службе могут быть привязаны несколько компонент сразу, но когда они все отменяют привязку, служба уничтожается [26].

Поставщики контента

Поставщик информации (Content provider) управляют общим набором данных, используемых приложением. Данные должны храниться в любом месте, к которому у приложения будет доступ, например в базе данных, сети, файловой системе и т.д.. В случае приложения «Европротокол онлайн», данные хранятся в файловой системе.

Приложения могут изменять, а так же запрашивать необходимые данные с помощью поставщика информации. Например, в системе андройд для управления сведениями контактов пользователя используется свой поставщик информации. Каждое приложение, которое получило надлежащие разрешения, может запросить часть этого

поставщика информации, для чтения и записи информации об определенном пользователе. Поставщики информации также употребляются для чтения и записи данных, доступ к которым наружным элементам приложение не предоставляет.

Поставщик информации относится к подклассу класса ContentProvider. Он должен реализовывать обычный набор API-интерфейсов, при помощи которых остальные приложения будут осуществлять транзакции [28].

Приемники широковещательных сообщений

Элемент, реагирующей на оповещения, которые распространяются по всей системе называется приемник широковещательных сообщений (Broadcast receiver). Большинство этих оповещений рассылает сама система, например оповещение о выключении экрана, низком уровне заряда аккумулятора либо об открытии камеры. Так же оповещения могут отправляться приложениям, к примеру, чтобы передать им информацию о том, что какие-либо данные были загружены на устройство и готовы для использования. Невзирая на то, что у приемников широковещательных сообщений отсутствует пользовательский интерфейс, они могут добавлять уведомления в строку состояния, для того, чтобы предупредить пользователя о каком-то событии. Однако чаще всего, приемники широковещательных сообщений являются шлюзом для остальных компонент. Они были созданы для реализации работ малого размера, например инициирования реализации службой определенных

ответных действий, при поступлении сигнала.

Приемник широковещательных сообщений относится к подклассу класса BroadcastReceiver, а каждое подобное сообщение является объектом Intent.

Одной из основных особенностей системы андройд является то, что любое приложение имеет возможность запустить элемент другого приложения. Например, в случае, когда необходимо добавить в приложение функцию создания фото, с помощью камеры устройства, то есть возможность вызвать стандартное приложение камеры устройства, вместо того, чтобы самим добавлять операцию создания фото в приложение, внедрения кода или установки ссылки. После завершения этой операции активности вернется в приложения, и можно будет продолжить взаимодействие с ним. Для конечного пользователя весь этот процесс будет выглядеть как единое приложение.

Во время запуска системой какого-либо компонента, она запускает процесс для приложения, если он не был запущен до этого. После этого создаются экземпляры классов, необходимые данному элементу. В связи с этим, в отличии от большинства других систем, в приложениях для андройд нет единой точки входа (например, функции main()). Поэтому, если приложение запустит операцию фотографирования в приложении для камеры, она будет осуществляться в процессе, относящемся к стороннему приложению, а не в процессе основного приложения.

Одно приложение не может вызвать элемент из

другого приложения самостоятельно, так как система запускает каждое из них в некотором процессе, с ограниченными правами для доступа в другие приложения. Это может сделать только сама андройд система. Поэтому, для вызова элемента из другого приложения, необходимо передать системе информацию о намерении (Intent) запуска определенного компонента. После этого система и активирует конкретный элемент.

3.1.3 Активация компонентов

Активация большинства типов компонент, а именно: операций, служб и приемников широковещательных сообщений происходит при помощи асинхронного сообщения, которое называется намерение(Intent).

Объекты намерений являются некими связующими для компонент, во время реализации программной части приложения. Это могут быть компоненты главного, либо же стороннего приложения, словно мессенджеры, которые посылают другим компонентам запрос на выполнение каких-либо действий. Объект намерения является экземпляром класса Intent, в котором описаны запросы на активацию как отдельных компонент, так и отдельных типов компонент. Соответственно, само намерение может быть явным либо не явным.

Для служб и операций в классе Intent описаны действия, которые необходимы для выполнения, например выслать(send), посмотреть(view). Так же объект намерения может задавать URI (Uniform Resource Identifier – унифицированный идентификатор ресурса) данных, вместе

с которыми нужно выполнить действие (кроме прочих сведений, которые нужны запускаемому компоненту). Таким образом, объект Intent может передавать запрос на выполнение операции "показать изображение". В случаях, когда операцию можно запустить, чтобы получить результат она вернет результат так же в виде экземпляра Intent (например, если отправить сообщение Intent, для выбора пользователем контакта, при его возврате, ответном сообщении будет содержаться URI, который будет указывать на выбранный пользователем контакт).

Для элемента Broadcast receiver, в сообщении Intent просто описывается передаваемое ему объявлении, например, при передачи сообщения с информацией о том, что батарея разряжена, широковещательное сообщение будет содержать лишь одну строку: «Батарея разряжена».

Последний тип компонентов – продавцы информации – активизируется по запросу от ContentResolver, а не с помощью сообщений Intent. Это сделано для того, чтобы активизацию не пришлось выполнять компоненту, который уже и так выполняет транзакции с поставщиком. Поэтому, Content resolver (процедура определения контента) является обработчиком всех прямых транзакций с поставщиком контента, а компонент просто вызывает методы объекта Content resolver. Так же, в целях безопасности, это позволяет сформировать слой, абстрагирующий поставщика контента от компонента, который запрашивает информацию.

Таким образом, для активации элементов каждого типа имеются такие способы, как:

- Запуск операции или определение для нее нового действия, с помощью передачи объекта методу startActivity() или startActivityForResult(), в тех случаях, когда необходим возврат результата из операции;
- Запуск службы, или выдача новых инструкций уже работающей службе, осуществляется при передачи объекта Intent в метод startService(). Так же возможно осуществление привязки к службе, с помощью передачи объкта Intent методу bindService();
- При передачи объекта Intent таким методам, как sendOrderedBroadcast(), sendBroadcast() и sendStickyBroadcast() будет инициирована рассылка сообщений;
- Если у объекта ContentResolver вызвать метод query(), будет выполнен запрос к поставщику контента [30].

3.1.4 Файл манифеста

Для того, чтобы запустить какой-либо компонент, необходимо сообщить о нем системе андройд. Для этого используется файл манифеста - AndroidManifest.xml. Он располагается в корневой папке приложения, и в нем описаны все компоненты приложения. Так же, файл манифеста используется и для других целей, таких как:

Указание всех требуемых приложению полномочий доступа (чтение контактов пользователя, доступ в интернет и т.п.)

Указание минимального уровня API, необходимого для запуска приложения, с учетом используемых им API-интерфейсов

Указание аппаратных и программных функций, требуемых приложению (камера, служба Bluetooth, сенсорный экран и т.п.)

Если приложение связано с отличными от API-интерфейсов платформы андройд библиотеками, таких как Google, то они так же указываются в файле манифеста.

Основной задачей манифеста является информирование системы о всех компонентах приложения. Примеры объявления операций в файле манифеста указаны ниже:

Указание на ресурсы для иконки мобильного приложения осуществляется с помощью атрибута android:icon в элементе <application>. Полное имя подкласса Activity указывается в атрибуте android:name в элементе <activity>. Такой атрибут, как android:label указывает строку, которая должна использоваться в качестве метки операции, отображаемой для пользователя.

Объявление всех компонент приложения происходит следующим образом:

- Элементы операций - <activity>;
- Элементы служб - <service>;
- Элементы приемников широковещательных сообщений - <receiver>;
- Элементы поставщиков контента - <provider>.

Системе не доступны операции, службы, поставщики контента, которые есть в исходном коде, но при этом не объявлены в файле манифеста, поэтому они не могут быть запущены. Но приемники широковещательных сообщений могут быть как объявлены в манифесте, так и динамически созданы в коде и после зарегистрированы в системе при помощи вызова registerReceiver() [29].

3.1.5 Объявление возможностей компонентов

Объектами Intent в приложении происходит запуск операций, приемников широковещательных сообщений и служб. Для этого в них, посредством имени класса компонента, явно указывается имя целевого компонента. Но в полной мере возможности Intent объектов раскрываются во время использования концепции неявных сообщений Intent, в которых просто описывается тип требуемого действия и данные, для которых будет совершено это действие. При этом система сама находит на устройстве компонент, который имеет возможность выполнить действие, описанное в сообщении, и выполняет его. Если на устройстве существует несколько компонент, которые могут выполнить это действие, то пользователь сам выбирает, какая из них должна использоваться. Это происходит путем определения системой компонент, которые могут ответить на данное сообщение, с помощью сравнения сообщения с фильтрами объектов, которые указаны в файле манифеста других приложений, хранящихся на устройстве.

В файле манифеста приложения указываются фильтры объектов Intent, во время объявления операций. Для того, чтобы операция могла реагировать на сообщения Intent от других приложений, указываются фильтры объектов Intent, при помощи дочернего элемента <intent-filter>, который был добавлен для объявления компонент.

3.1.6 Объявление требований приложения

Не все устройства, работающие на ОС андройд имеют одинаковые функциональные возможности. Чтобы приложение «Европротокол онлайн» не могло быть установлено на устройствах, в которых отсутствуют функции, необходимые приложению, определяется профиль для типов устройств, поддерживаемых приложением. Это делается с помощью указания в файле манифеста требований программного, а так же аппаратного обеспечения.

По большей части эти указания носят информационный характер, так как система их не читает. Но такие внешние службы, как Google Play, для обеспечения фильтрации приложений под каждое устройство пользователей, читают данные указания.

Например, приложению «Европротокол онлайн» требуется возможность подключения к интернету и в нем используются API-интерфейсы уровня Android 4.1 (уровень API 25). После объявления этих требований в файле манифеста приложения, его нельзя будет установить из магазина Google Play на те устройства, у которых

установлена ОС Android версии ниже 4.1 или отсутствует возможность подключения к интернету.

Так же, у разработчиков программного обеспечения имеется возможность объявить, что, например, приложение использует камеру мобильного телефона, но для работы приложения она не является крайне необходимой. В таком случае атрибуту required задается значение "false", а мобильное приложение во время работы проверит, имеется ли возможность использовать камеру на устройстве, и при необходимости отключит функции, связанные с использованием камеры [27].

3.1.7 Ресурсы приложения

Большинству мобильных приложений, в том числе и приложению «Европротокол онлайн» необходимы такие ресурсы, как изображения, видеофайлы, стили, цвета, макеты пользовательских интерфейсов и все остальное, связанное с визуальным представлением приложения. Такие ресурсы существуют отдельно от исходного кода приложения. При помощи ресурсов у разработчиков есть возможность менять различные характеристики приложения, не затрагивая при этом его исходный код. А так же, при использовании альтернативных ресурсов приложение получает дополнительную оптимизацию, для работы с устройствами, разных типов, например при использовании экранов разных размеров или для различных языков текста.

Инструменты SDK задают уникальный целочисленный идентификатор для каждого ресурса, который включается в

проект Android. Он может использоваться для того, чтобы ссылаться на какой-либо ресурс из кода приложения, или из других ресурсов, которые определены в XML.

Например, в приложении имеется файл логотипа приложения с именем logotype.png , который хранится в папке resources/logo/, SDK сформирует идентификатор ресурса под именем R.drawable.logotype, с помощью которого вставлять изображение в пользовательский интерфейс и ссылаться на него из кода приложения.

Еще один важнейший аспект хранения ресурсов приложения отдельно от его исходного кода – возможность использовать «альтернативные ресурсы» для разных устройств и их конфигураций [31]. Таким образом можно перевести строки пользовательского интерфейса и сохранить их в отдельных файлах с помощью определения этих строк в XML документе. После этого система Android применит к тексту интерфейса необходимый язык, используя квалификатор языка и учитывая язык, выбранный пользователем.

Для разных типов ресурсов приложение поддерживает соответствующие квалификаторы. Квалификатор – короткая строка, включающаяся в имена каталогов ресурсов для определения конфигурации устройства, для которой будут использоваться ресурсы. Таким образом, у разработчика есть возможность создать разные макеты, соответствующие разным размерам и ориентациям экранов устройства. Благодаря этому, в книжной ориентации устройства(когда оно будет расположено вертикально), кнопки в макете

будут соответствующе расположены по вертикали, а при альбомной ориентации устройства(экран устройства расположен горизонтально), кнопки разместятся по горизонтали. Для этого используется два разных макета, к именам которых применяется соответствующие квалификаторы. Благодаря этому, система автоматически применяет соответствующие макеты, в зависимости от ориентации устройства.

3.1.8 Основные экраны работы мобильного приложения

Переключение между рабочими экранами (view) приложения, в основном, осуществляется за счет нажатия пользователем кнопок «ДАЛЕЕ» и «НАЗАД». При этом происходит вызов метода startActivity() класса Activity, в который передается объект Intent, который содержит текущий контекст приложения и класс, относящийся к следующей Activity.

На стартовом этапе работы приложения пользователю предлагается выбрать, к какому экрану перейти – к проверке соблюдения условий европротокола, или получить справку о европротоколе. Снимок стартового экрана работы мобильного приложения приведен на рисунке 3.3.

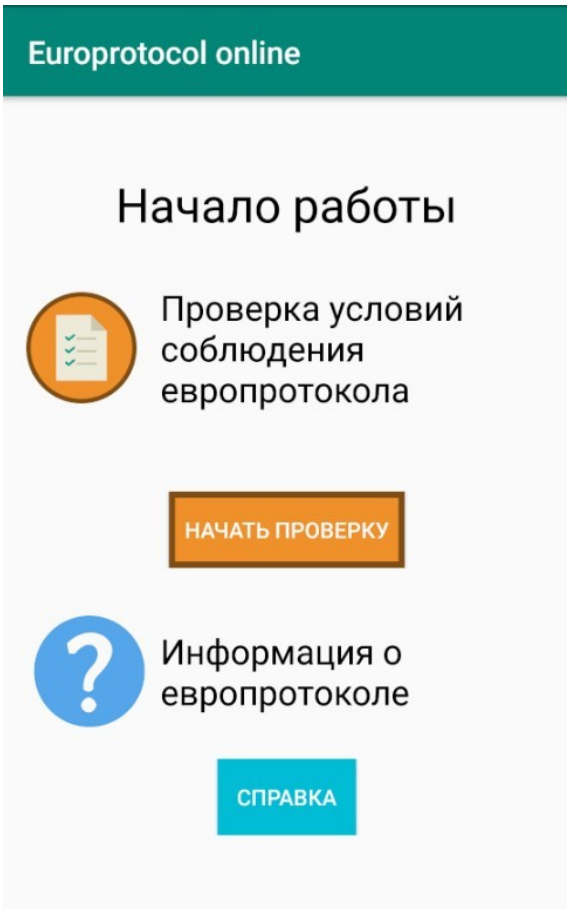

Рисунок 3.3 - Начальный экран приложения

При нажатии на кнопку «СПРАВКА» открывается «activity_help»(рисунок 3.4).

На открывшейся странице пользователю предлагается скачать «Правила представления информации о дорожно-транспортном происшествии страховщику». Кнопка «Скачать постановление» является гиперссылкой, ведущей на скачивание постановления Правительства РФ от 1 октября 2014 г. N 1002 "Об утверждении Правил представления информации о дорожно-транспортном происшествии страховщику и требований к техническим средствам контроля, обеспечивающим некорректируемую регистрацию информации" в формате .pdf.

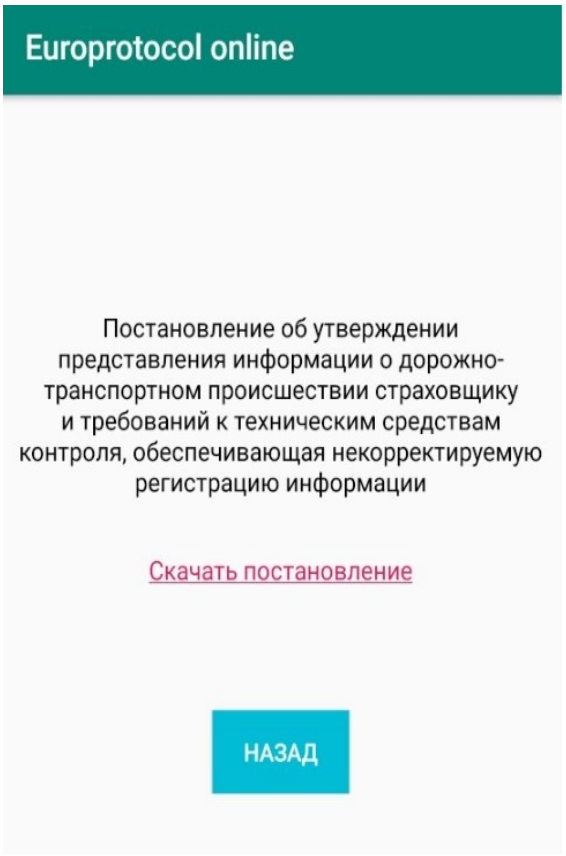

Рисунок 3.4 - Экран загрузки постановления

Это реализуется путем добавления гиперссылки перед текстом и вызова у TextView, содержащей текст «Скачать постановление» вызова метода setMovementMethod, в который передается экземпляр объекта LinkMovementMethod.

При нажатии на кнопку «НАЧАТЬ ПРОВЕРКУ» на начальном экране приложения, происходит переход к экранам, осуществляющих проверку основных условий возможности составления европротокола. Пример приведен на рисунке 3.5.

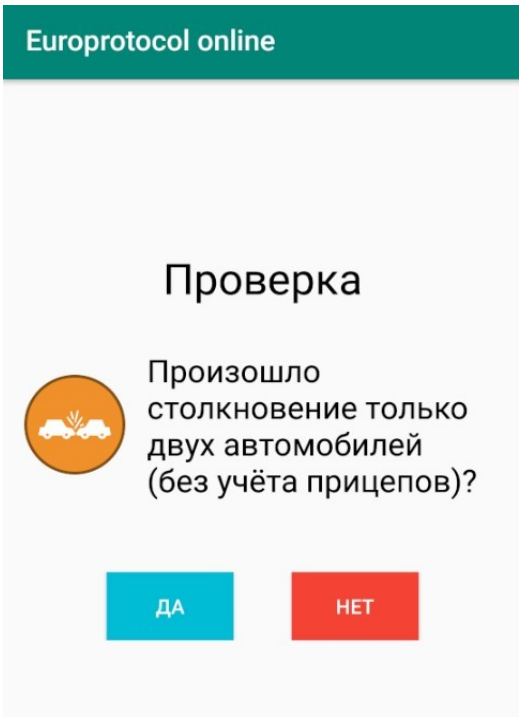

Рисунок 3.5 - Проверка условий

После того, как пользователи проходят проверку, открывается экран создания и подключения к сессии создания европротокола.(рисунок 3.6).

Рисунок 3.6 - Экран создания/подключения к сессии

После нажатия на кнопку «ОФОРМИТЬ ЕВРОПРОТОКОЛ» открывается экран создания соединения двух пользователей.(рисунок 3.7).

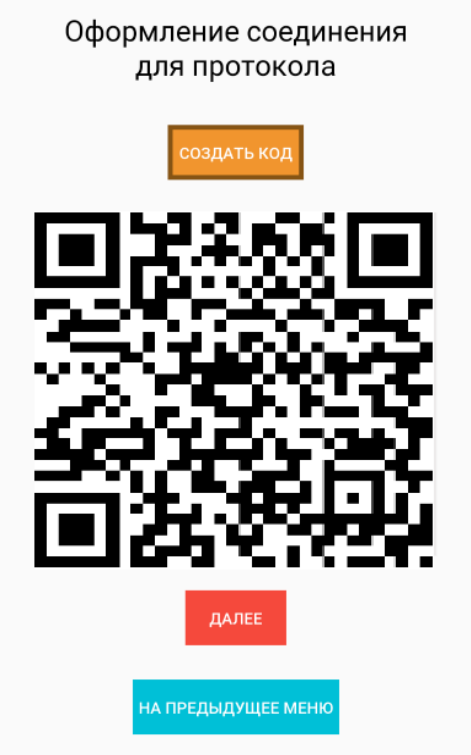

Рисунок 3.7 - Экран создания сессии

При этом создается шифрованный ключ сессии, который сохраняется в объект, хранящий данные пользователя. Затем используется цифровое кодирование: вся последовательность символов разбивается на группы по 3 цифры, и каждая группа (трёхзначное число) переводится в 10-битное двоичное число и добавляется к последовательности бит. Если общее количество символов не кратно 3, то, если в конце остаётся 2 символа, полученное двузначное число кодируется 7 битами, а если 1 символ, то 4 битами. Далее каждое число переводится в двоичный вид и объединяется в один битовый поток. После чего, путем

вызова метода compress() у объекта Bitmap, в который передается поток, создается QR-код, содержащий в себе информацию, о ключе сессии.

При нажатии на кнопку «ПРИСОЕДИНИТЬСЯ К ОФОРМЛЕНИЮ» осуществляется переход к процессу подключения пользователя к созданной сессии посредством сканирования QR-кода с помощью камеры мобильного телефона (рисунок 3.8).

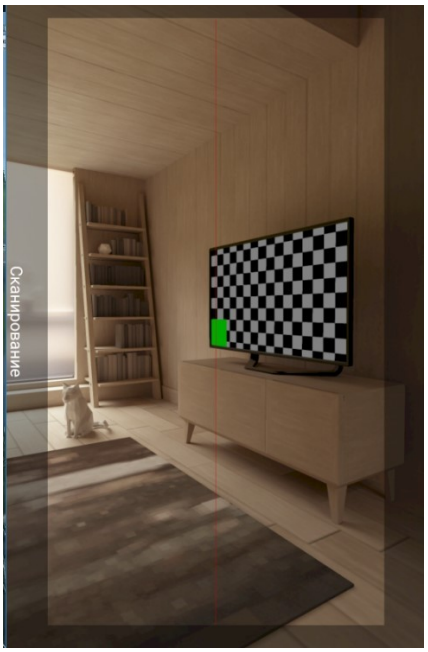

Рисунок 3.8 - Экран подключения к сессии

Для этого необходимо получать изображение с камеры. Причем получение изображения должно происходить не по клику, а непрерывно. Для этого Activity реализовывает интерфейс Camera. В методе PreviewCallback и в методе onPreviewFrame есть возможность работать с preview-картинками в реальном времени. Чтобы распознавание кода не привело к задержкам в интерфейсе, каждая полученная картинка обрабатывается в отдельном потоке. Поток, который

успешно распознал код, вызывает обратное преобразование картинки в шифрованный ключ в виде строки и переход на другое Activity, при этом шифрованный ключ сохраняется в объекте пользователя.

Далее оба пользователя переходят процессу заполнения данных, необходимых для составления европротокола (персональные данные, данные об аварии, данные страховой компании и т.д.). Примеры экранов заполнения данных предоставлены на рисунках 3.9 и 3.10.

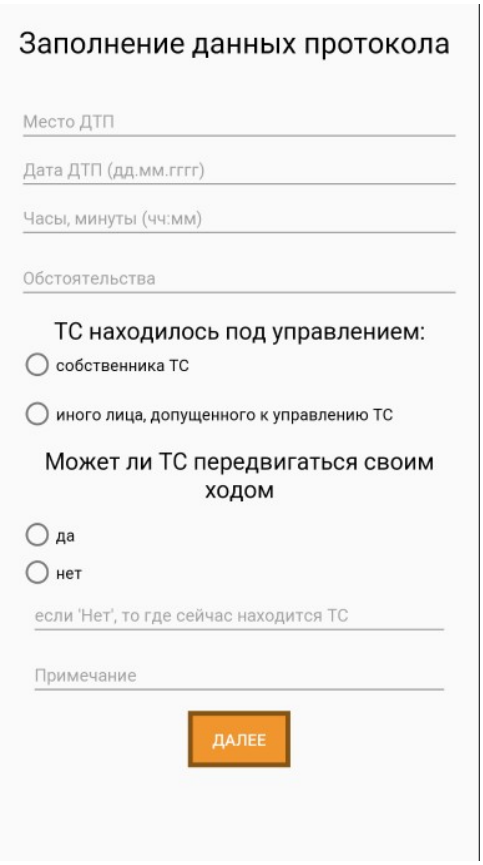

Рисунок 3.9 - Заполнение данных протокола

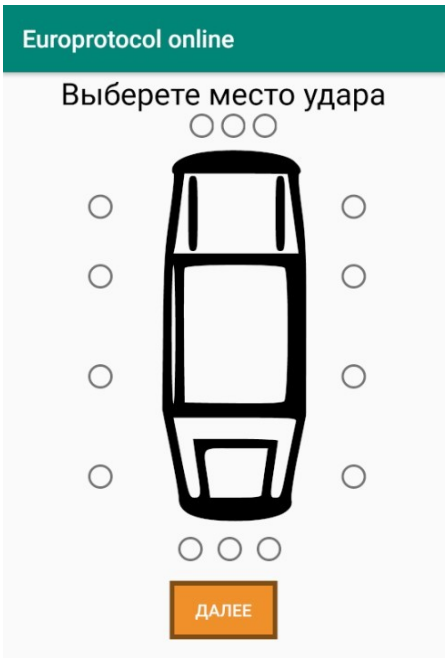

Рисунок 3.10 - Выбор места удара

Так же, к экранам заполнения личных данных относятся экраны, осуществляющие прием данных о схеме ДТП(рисунок 3.11) и личной подписи пользователя, путем «рисования» их на экране мобильного телефона.

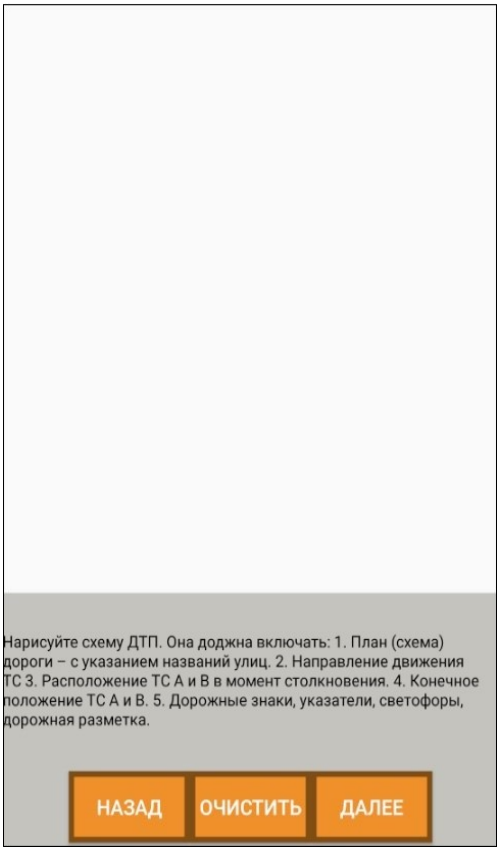

Рисунок 3.11 - Прием личной подписи

Для этого происходит обработка касаний и перемещений пальца по View-компоненту. Они состоят из трех типов событий:

- Нажатие (палец прикоснулся к экрану);
- Движение (палец движется по экрану);
- Отпускание (палец оторвался от экрана).

Все эти события захватываются в обработчике OnTouchListener, который присвоен для View-компонента. Этот обработчик имеет объект MotionEvent, из которого извлекаются тип события и координаты.

Непосредственно для рисования используется объект Canvas. Canvas является лишь инструментом для рисования, а весь результат сохраняется на Bitmap. Мы не можем напрямую попросить Bitmap нарисовать на себе

линию или круг, поэтому Canvas выступает посредником и помогает нарисовать то, что нужно. Самым важным шагом в создании пользовательского «рисунка» является переопределение onDraw() метода. Метод onDraw() является Canvas объектом, который представление может использовать для рисования. Canvas класс определяет методы для рисования текста, линии, растровых изображений, а также множества других графических примитивов.

После того, как пользователь нажимает на кнопку «ДАЛЕЕ», изображение с Canvas объекта преобразуется в Bitmap объект, а после в массив байтов для того, чтобы можно было передать этот рисунок на сервер в формате JSON.

После введения пользователями всех данных, происходит переход к последнему экрану приложения(рисунок 3.12), где пользователь вводит почтовый ящик, на который придет заполненный документ европротокола.

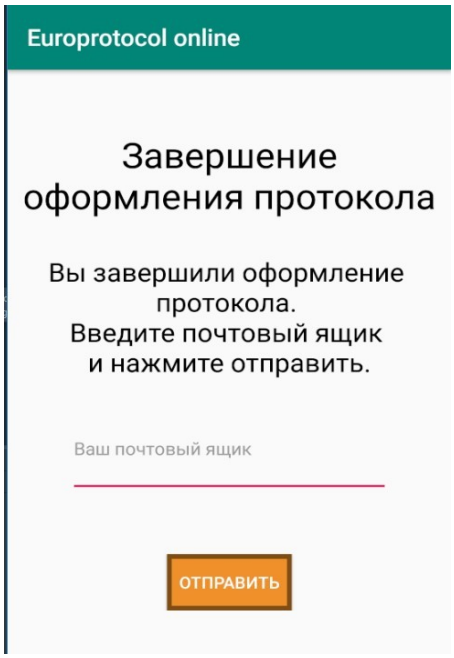

Рисунок 3.12 - Завершающий экран

После нажатия пользователем кнопки «ОТПРАВИТЬ» устанавливается соединение с сервером приложения, объект пользователя, со всеми данными преобразуется в строку формата JSON и отправляется на сервер для генерации заполненного документа европротокола.

3.2 Реализация серверной части

Сервер представляет собой консольное приложение, выполняющее такие функции, как:

- Прием запроса от клиента в формате JSON;
- Перевод JSON в POJO объект пользователя;
- Преобразование картинок, полученных в виде байтового массива из JSON в картинку формата .png;
- Генерацию заполненного данными пользователей документа европротокола, в формате .docx;
- Отправление документа европротокола на электронную почту пользователей.

3.2.1 Использование JSON объектов

JSON (JavaScript Object Notation) – особый текстовый формат описания данных, подходящий для чтения и редактирования человеком и легкий для обработки [29]. JSON является подмножеством языка JavaScript. Однако благодаря комфортному синтаксису, может употребляться со всеми языками программирования для хранения и передачи данных. JSON нередко противопоставляют XML, т. к. у них однообразные сферы внедрения [34]. К примеру, в интернет разработке применяется подход AJAX (Asynchronous JavaScript and XML), но в этом подходе чаще используется JSON заместо XML. JSON более лаконичен, чем XML, и поэтому, быть может удобнее для описания больше трудных структур [35]. Для конвертации JAVA объектов в формат JSON и назад можно выделить следующие библиотеки.

- COM.GOOGLE.CODE.GSON;
- JSON.SIMPLE;
- JSON-P;
- Jackson.

Для мобильного приложения «Европротокол онлайн» была выбрана библиотека Jackson от FasterXML. Это набор инструментов для обработки данных, который был основан на потоковом JSON-парсере и генераторе [35]. Достоинства Jackson:

- Простота использования – Jackson API предоставляет фасад высочайшего уровня для упрощения частых случаев использования;

- Нет необходимости создавать карты – Jackson API обеспечивает отображение по умолчанию для большей части объектов, которые будут сериализованы.

  Быстрота работы – Jackson работает достаточно быстро, занимает не много памяти и хорошо справляется с данными огромных размеров;
- Отсутствие зависимостей – библиотека Jackson не нуждается в иных библиотеках, кроме JDK;
- Open Source – библиотека Jackson имеет открытый исходный код и бесплатна для использования.

  Джексон предлагает три альтернативных способа обработки JSON.

JSON объекты в приложении имеют большой объем, в связи с тем, что включают себя изображения, преобразованные в байтовый массив. Поэтому скорость передачи JSON является важной частью, при работе приложения с большим количеством клиентов [36].

При работе мобильного приложения все данные полученные от пользователя в виде POJO объекта, перед отправкой на сервер преобразуются в JSON объект, а после принятия сервером объекта конвертируются обратно в POJO, для дальнейшей работы с данными на сервере.

3.2.2 Генерация заполненного бланка европротокола

Для работы с документами в формате .docx на сервере используется библиотека Apache POI.

Apache POI – это пользующийся популярностью API, позволяющий создавать, изменять и показывать файлы MS Office при помощи программ Java. Это библиотека с открытым исходным кодом, которая была разработана и распространяемая Apache Software Foundation для разработки либо изменения файлов Microsoft Office с внедрением программы Java. Он содержит классы и методы для декодирования вводимых пользователем данных либо файла в документы MS Office.

Составляющие Apache POI:

Apache POI содержит классы и способы для работы со всеми составными документами OLE2 MS Office. Перечень компонент этого API приведен ниже.

- POIFS (Файловая система реализации плохой обфускации) – этот компонент является основным фактором всех других элементов POI. Он используется для явного чтения разных файлов;
- HSSF (формат электронной таблицы) – используется для чтения и записи в формате xls файлов MS-Excel;
- XSSF (XML Spreadsheet Format) – используется для формата файлов .xlsx MS-Excel;
- HPSF (формат набора свойств) – используется для извлечения наборов свойств из файлов MS-Office;
- HWPF (формат текстового процессора) – используется для чтения и записи файлов расширений doc MS-Word;
- XWPF (XML-формат текстового процессора) – используется для чтения и записи файлов расширения docx в MS-Word;

- HSLF (формат макета слайда) – используется для чтения, создания и редактирования презентаций PowerPoint;
- HDGF (формат Horrible DiaGram) – содержит классы и методы для двоичных файлов MS-Visio.

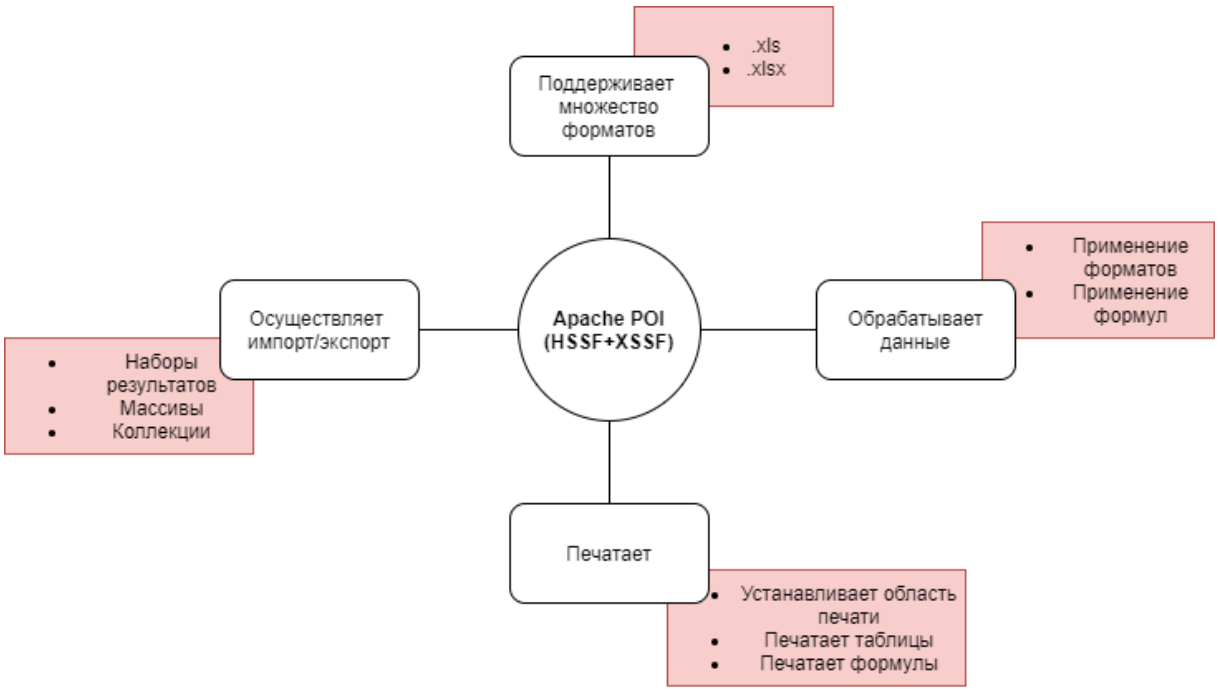

Рисунок 3.13 - Возможности библиотеки Apache POI

Считанный в память docx документ представляет из себя экземпляр класса XWPFDocument, который в последствии разбирается на составляющие. С этой целью используются следующие особые классы

- Отдельные классы XWPFHeader и XWPFFooter — для работы (считывания/создания) верхнего и нижнего колонтитулов. Доступ к ним получается через специальный класс-поставщик XWPFHeaderFooterPolicy;
- Класс XWPFParagraph — для работы с параграфами;

- Класс XWPFWordExtractor — для парсинга содержимого всей страницы .docx документа [37].

Генерация документов на сервере происходит следующим образом:

В ресурсах сервера хранится исходный бланк европротокола с расставленными в нем на определенных позициях тегами. После получения запроса с клиента вызывается метод, в котором каждый тег заменяется символом / строкой, которая берется из POJO с данными, пришедшими с клиента. Изображения в свою очередь преобразуются из байтового массива в файл формата .png, и так же вставляются в нужные места документа.

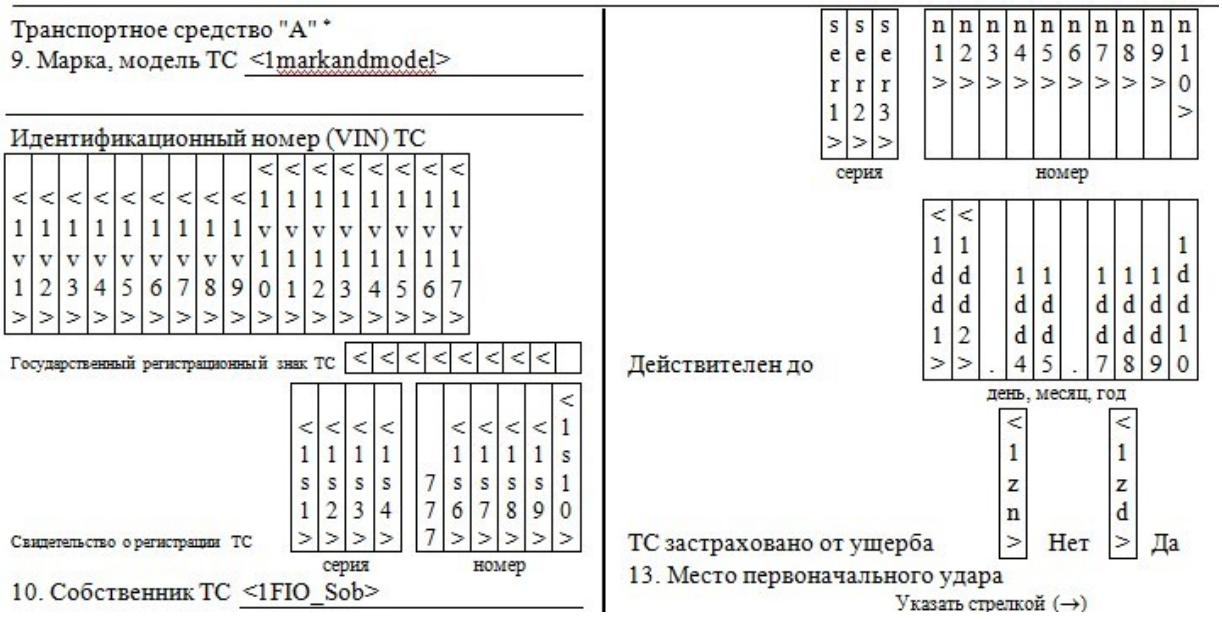

Рисунок 3.14 - Часть исходного бланка европротокола

3.2.3 Отправление бланка европротокола на электронную почту пользователей

Для передачи заполненного бланка европротокола на электронную почту пользователей используется JavaMail API.

JavaMail API предоставляет независимую от платформы и не зависящую от протокола структуру для создания приложений почты и обмена сообщениями. JavaMail API предоставляет набор абстрактных классов, которые определяют объекты, составляющие почтовую систему. Это необязательный пакет для чтения, составления и отправки электронных сообщений.

JavaMail предоставляет элементы, используемые для добавления интерфейса к системе обмена сообщениями, в том числе системные составляющие и интерфейсы. Однако эта спецификация не описывает какую-то определенную реализацию, JavaMail содержит в себе несколько классов, реализующих эталоны обмена сообщениями RFC822 и MIME. Эти классы поставляются как часть пакета классов JavaMail (рисунок 3.15).

Рисунок 3.15 - Архитектура JavaMail

Абстрактный механизм JavaMail API похож на другие J2EE API, такие как JDBC, JNDI и JMS. Как видно на рисунке 3.15, JavaMail API делится на две основные части:

- Независимая от приложения часть. Интерфейс прикладного программирования (API) используется элементами приложения для отправки и получения почтовых отправлений вне зависимости от используемого основного поставщика либо протокола;
- Зависимая от службы часть: интерфейс поставщика услуг (SPI) передает данные на языках протокола, таких как SMTP, POP, IMAP и протокол передачи сетевых новостей (NNTP). Он используется для подключения поставщика услуг электронной почты к платформе J2EE [38].

Для отправки электронного сообщения с прикрепленным документом европротокола выполняются следующие действия:

1. Происходит получение сессии, с использованием логина и пароля почтового ящика, созданного для отправления писем пользователям.
2. Создается объект MimeMessage и в него устанавливается From, To и Subject сообщения.
3. Устанавливается фактическое сообщение с помощью метода «messageBodyPart.setText()».
4. Создается объект MimeMultipart. В него добавляется messageBodyPart с установленным в нем фактическим сообщением к этому составному объекту.
5. Затем добавляется вложение, посредством создания обработчика данных.
6. Объект MimeMultipart добавляется в сообщение.
7. Происходит отправление сообщения с помощью транспортного объекта.

Заключение

В ходе выполнения ВКР было разработано мобильное приложение для упрощенного оформления ДТП без участия сотрудников полиции «Европротокол онлайн», в частности были решены следующие задачи:

1. Проведен анализ приложений, имеющихся на Google Play и соответствующих тематике оформления европротокола.
2. Разработана структура мобильного приложения «Европротокол онлайн».
3. Создано мобильное приложение, реализующее следующие функции:
    - Создание защищенной сессии между пользователями;
    - Получение от пользователей всех данных, необходимых для заполнения бланка европротокола;
    - Отправление пользовательских данных на сервер мобильного приложения;
    - Генерация заполненного бланка европротокола для каждого из пользователей;
    - Автоматическое отправление заполненного бланка европротокола на электронные почты пользователей.
4. Спроектирован и реализован пользовательский интерфейс.
5. Приложение было протестировано и показало свою работоспособность как на стандартных эмуляторах,

взятых из SDK Android, так и на реальных устройствах на платформе Android.

Программное обеспечение клиентской части приложения написано на языке Java в среде Android Studio.

Программное обеспечение серверной части приложения написано на языке Java в среде Intellij Idea 2018.3.2.

Приложение может использоваться водителями автомобильных средств, попавших в ДТП и имеющих смартфон с операционной системой Android версии не ниже 2.1, для составления заполненного документа европротокола при помощи мобильного телефона.

Результаты данной выпускной квалификационной работы докладывались на XVI международной научно-практической конференции «Проблемы управления в социально-экономических и технических системах» в секции «Методы разработки информационных систем и систем управления» в 2020 году.

Список использованных источников

1. Европротокол [Электронный ресурс]: Режим доступа: https://www.alfastrah.ru/individuals/auto/osago/evroprotokol-pri-dtp/ (Дата обращения 13.04.2020).
2. Риа новости. Сколько ДТП в России оформляется по европротоколу [Электронный ресурс]: Режим доступа: https://ria.ru/20191119/1561100437.html (Дата обращения 13.04.2020).
3. Приказ МВД России от 02.03.2009 N 185 (ред. от 22.12.2014) "Об утверждении Административного регламента Министерства внутренних дел Российской Федерации исполнения государственной функции по контролю и надзору за соблюдением участниками дорожного движения требований в области обеспечения безопасности дорожного движения" (Зарегистрировано в Минюсте России 18.06.2009 N 14112).
4. Владимиров С.Н. Транспортные заторы в условиях мегаполиса / Известия Московского государственного технического университета МАМИ №1(19), 2014 т.3 С.77-83.
5. Постановление Правительства РФ от 28 августа 2019 г. N 1108

   "Об утверждении Правил представления страховщику информации о дорожно-транспортном происшествии, обеспечивающих получение страховщиком некорректируемой информации о дорожно-транспортном происшествии, и требований к техническим средствам контроля и составу информации о дорожно-транспортном

происшествии, а также о признании утратившими силу некоторых актов Правительства Российской Федерации" .
6. Требования к эргономике и технической эстетике [Электронный ресурс]: Режим доступа: https://studfile.net/preview/5663679/page:3/ (Дата обращения 13.04.2020).
7. Помощник ОСАГО - оформление извещения о ДТП в виде электронного документа [Электронный ресурс]: Режим доступа: https://autoins.ru/evropeyskiy-protokol/uproshchennoe-oformlenie-dtp/mob_app/ (Дата обращения 13.04.2020).
8. Мобильное приложение «ДТП. Европротокол» [Электронный ресурс]: Режим доступа: https://autoins.ru/faq/mp_ep/ (Дата обращения 13.04.2020).
9. WreckCheck mobile application [Электронный ресурс]: Режим доступа: https://play.google.com/store/apps/details?id=org.naic.android.wreckcheck&hl=ru (Дата обращения 13.04.2020).
10. Медникс З., Дорнин Л. Программирование под Android. Издательство Питер, 2012.
11. Каймин В.А. Информатика: Учебное пособие: Изд. 2-е. Издательство РИОР, 2007.
12. Амелин К. С., Граничин О. Н., Кияев В. И., Корявко А. В.. Введение в разработку приложений для мобильных платформ. Издательство ВВМ, 2011.
13. Мельникова О.М.: Смартфоны на Android. Издательство Эксмо, 2013.
14. John Wiley & Sons. Reto Meier Professional Android 4 Application Development. Wrox, 2012.

15. Варакин М.В. Разработка мобильных приложений под Android. УЦ «Специалист» при МГТУ им. Н. Э. Баумана, 2012.

16. Варакин М.В. Разработка мобильных приложений под Android. УЦ«Специалист» при МГТУ им. Н. Э. Баумана, 2012.

17. Котляров В.П. Основы тестирования программного обеспечения. Издательство Бином, 2009.

18. Рынок мобильных ОС [Электронный ресурс]: Режим доступа: http://w7phone.ru/rynok-mobilnyx-os-statistika-za-sentyabr-2017-141927/ (Дата обращения 13.04.2020).

19. Давыдов, Станислав IntelliJ IDEA. Профессиональное программирование на Java. Наиболее полное руководство / Станислав Давыдов , Алексей Ефимов. - М.: БХВ-Петербург, 2011. - 800 с.

20. Зелковиц, М. Принципы разработки программного обеспечения / М. Зелковиц, А. Шоу, Дж. Гэннон. - М.: Мир, 2018. - 364 с.

21. Голощапов, Алексей Google Android. Программирование для мобильных устройств / Алексей Голощапов. - М.: БХВ-Петербург, 2012. - 448 с.

22. Перерва, Андрей Дмитриевич Путь аналитика. Практическое руководство IT-специалиста / Перерва Андрей Дмитриевич. - М.: Питер, 2016. - 654 с.

23. Семенов, С.С. Методы принятия решений в задачах оценки качества и технического уровня сложных

технических систем / С.С. Семенов. - Москва: СПб. [и др.] : Питер, 2016. - 500 с.

24. Гома, Хассан UML Проектирование систем реального времени, распределенных и параллельных приложений / Хассан Гома. - М.: ДМК Пресс, 2018. - 700 с.
25. Даниэль, Арсеновски Рефакторинг в C# и ASP.NET для профессионалов / Арсеновски Даниэль. - М.: Диалектика / Вильямс, 2018. - 265 с.
26. Основы создания приложений [Электронный ресурс]: Режим доступа: https://developer.android.com/guide/components/fundamentals?hl=ru (Дата обращения 11.05.2020).
27. Компоненты Android приложения [Электронный ресурс]: Режим доступа: https://sites.google.com/site/interviewknowages/-android (Дата обращения 11.05.2020).
28. Основные сведения о поставщике контента [Электронный ресурс]: Режим доступа: https://developer.android.com/guide/topics/providers/content-provider-basics?hl=ru (Дата обращения 11.05.2020).
29. Манифест приложения [Электронный ресурс]: Режим доступа: https://developer.android.com/guide/topics/manifest/manifest-intro?hl=ru (Дата обращения 17.05.2020).
30. Объект намерения Intent [Электронный ресурс]: Режим доступа: http://java-online.ru/android-intent.xhtml (Дата обращения 17.05.2020).

31. Создание Android приложений. Структура Android приложения. Основные UI паттерны разработки Android приложений [Электронный ресурс]: Режим доступа: https://beasthackerz.ru/fleshka/sozdanie-android-prilozhenii-struktura-android-prilozheniya-osnovnye-ui-patterny.html (Дата обращения 18.05.2020).
32. JSON (JavaScript Object Notation) [Электронный ресурс]: Режим доступа: https://ru.bmstu.wiki/JSON_(JavaScript_Object_Notation) (Дата обращения 22.05.2020).
33. Нотон Java. Справочное руководство. Все, что необходимо для программирования на Java / Нотон, Патрик. - М.: Бином, 2015. - 448 с.
34. JSON.simple Обзор [Электронный ресурс]: Режим доступа: https://ru.it-brain.online/tutorial/json_simple/json_simple_quick_guide/ (Дата обращения 23.05.2020).
35. Протокол JSONP [Электронный ресурс]: Режим доступа: https://learn.javascript.ru/ajax-jsonp (Дата обращения 23.05.2020).
36. Java-библиотеки для работы с JSON [Электронный ресурс]: Режим доступа: https://tproger.ru/translations/java-json-library-comparison/ (Дата обращения 23.05.2020).
37. Apache POI – Краткое руководство [Электронный ресурс]: Режим доступа: https://coderlessons.com/tutorials/java-tekhnologii/izuchite-apache-poi/apache-poi-kratkoe-rukovodstvo (Дата обращения 24.05.2020).

38. JavaMail API [Электронный ресурс]: Режим доступа: https://coderlessons.com/tutorials/java-tekhnologii/izuchite-javamail-api/javamail-api-kratkoe-rukovodstvo (Дата обращения 24.05.2020).

ПРИЛОЖЕНИЕ А

Исходный код серверной части программного обеспечения

Листинг А.1 – Исходный код класса «Main.java» серверной части программного обеспечения, запускающего работу серверной части приложения

```
package main.java.com.company;
public class Main {
    public static void main(String[] args) {
        Server server = Server.getServer();
        Thread t = new Thread(server);
        t.start();
    }
}
```

Листинг А.2 – Исходный код класса «Server.java» серверной части программного обеспечения, осуществляющего работу сервера в режиме ожидания нового соединения.

```
import java.io.IOException;
import java.net.ServerSocket;
public class Server implements Runnable {
    private static volatile Server instane = null;        /* Порт, на который
сервер принимает соеденения */
    private final int SERVER_PORT = 6789;        /* Сокет, который
обрабатывает соединения на сервере */
    private ServerSocket serverSoket = null;
    private Server() {
    }
```

```
    public static Server getServer() {
```



```
     if (instane == null) {
            synchronized (Server.class) {
                if (instane == null) {
                    instane = new Server();
                }
            }
        }
        return instane;
    }
    @Override
    public void run() {
        try {            /* Создаем серверный сокет, которые принимает
соединения */
            serverSoket = new ServerSocket(SERVER_PORT);
            System.out.println("Start server on port: " + SERVER_PORT);
            while (true) {
                ConnectionWorker worker = null;
                try {                    /* ждем нового соединения  */
                    worker = new ConnectionWorker(serverSoket.accept());
                    System.out.println("Get client connection"); /* создается
новый поток, в котором обрабатывается соединение */
                    Thread t = new Thread(worker);
                    t.start();
                } catch (Exception e) {
                    System.out.println("Connection error: " + e.getMessage());
                }
            }
        } catch (IOException e) {
```

```
            System.out.println("Cant start server on port " + SERVER_PORT
+ ":" + e.getMessage());
        } finally { /* Закрываем соединение */
            if (serverSoket != null) {
```

ПРОДОЛЖЕНИЕ ПРИЛОЖЕНИЯ А

```
 try {
                    serverSoket.close();
                }
             catch (IOException e) {
                }
            }
        }
    }
```

Листинг А.3 – Исходный код класса «ConnectionWorker.java» серверной части программного обеспечения, осуществляющего создание сессии между клиентской и серверной частью приложения.

```
package main.java.com.company;
import com.fasterxml.jackson.databind.ObjectMapper;
import com.google.gson.Gson;
import com.google.gson.stream.JsonReader;
import main.java.com.company.model.UserData;
import org.apache.poi.openxml4j.exceptions.InvalidFormatException;
import java.io.IOException;
import java.io.InputStream;
import java.io.StringReader;
import java.net.Socket;
public class ConnectionWorker implements Runnable {
```

```
    /* сокет, через который происходит обмен данными с клиентом*/
private Socket clientSocket = null;  /* входной поток, через который
получаем данные с сокета */
    private InputStream inputStream = null;
    DocGenerator generator = new DocGenerator();
    Gson gson = new Gson();
    public static String firstEmail;
    public ConnectionWorker(Socket socket) {
```



```
     clientSocket = socket;
    }
    @Override
    public void run() {  /* получаем входной поток */
        try {
            inputStream = clientSocket.getInputStream();
        } catch (IOException e) {
            System.out.println("Cant get input stream");
        }
/* создаем буфер для данных */
        byte[] buffer = new byte[2048 * 128];
        while (true) {
            try {
                System.gc();
                int count = inputStream.read(buffer, 0, buffer.length);  /*
проверяем, какое количество байт к нам прийшло */
                if (count > 0) {
                    ObjectMapper mapper = new ObjectMapper();
                    UserData userData = mapper.readValue(new String(buffer,
0, count), UserData.class);
                    System.out.println(userData);
                    try {
                        MailSender ms = new MailSender();
```

```
                    if(!userData.getConnection()) {
                        generator.generateFirstUserFirstPart(userData);
                        System.gc();
                        generator.generateFirstUserSecondPart(userData);
                        System.gc();
                        firstEmail = userData.getUserEmail();
                    }
                    else {
```

ПРОДОЛЖЕНИЕ ПРИЛОЖЕНИЯ А

```
            generator.generateSecondUserFirstPart(userData);
                        System.gc();
                        generator.generateSecondUserSecondPart(userData);
                        ms.sendMail(firstEmail,"F:\\1\\blank_output2_1.docx",
"F:\\1\\blank_output2.docx");
                        ms.sendMail(userData.getUserEmail(),"F:\\1\\
blank_output2_1.docx", "F:\\1\\blank_output2_2.docx");
                    }
            System.gc();
                } catch (InvalidFormatException e) {
                    e.printStackTrace();
                } catch (Exception e) {
                    e.printStackTrace();
                }
              } else /* если получили -1, значит прервался наш поток с
данными  */
                if (count == -1) {
                System.out.println("close socket");
                clientSocket.close();
                break;
              }
            } catch (IOException e) {
              System.out.println(e.getMessage());
```

```
        }
      }
   }
}
```

Листинг А.4 – Исходный код класса «MailSender.java» серверной части программного обеспечения, осуществляющего отправление сообщений на электронную почту пользователей

ПРОДОЛЖЕНИЕ ПРИЛОЖЕНИЯ А

```
import javax.mail.*;
import javax.mail.internet.InternetAddress;
import javax.mail.internet.MimeBodyPart;
import javax.mail.internet.MimeMessage;
import javax.mail.internet.MimeMultipart;
import java.io.IOException;
import java.util.Properties;
public class MailSender {
   public void sendMail(String email, String pathToFile, String
pathToSecondFile) {
       final String username = "europrotocol2020";
       final String password = "yncganykiigxqjrv";
       String fromMail = "europrotocol2020@yahoo.com";
       Properties properties = new Properties();
       properties.put("mail.smtp.auth", "true");
       properties.put("mail.smtp.starttls.enable", "true");
       properties.put("mail.smtp.host", "smtp.mail.yahoo.com");
       properties.put("mail.smtp.port", "587");
       Session session = Session.getInstance(properties, new
Authenticator() {
```

```
            @Override
            protected PasswordAuthentication getPasswordAuthentication() {
                return new PasswordAuthentication(username,password);
            }
        });
        MimeMessage msg = new MimeMessage(session);
        try {
            msg.setFrom(new InternetAddress(fromMail));
            msg.addRecipient(Message.RecipientType.TO, new
InternetAddress(email));
            msg.setSubject("Европротокол");
            msg.setText("Ваш готовый образец европротокола.
Предоставьте его в вашу страховую компанию.");
            Multipart emailContent = new MimeMultipart();
```



```
MimeBodyPart docAttachment = new MimeBodyPart();
            docAttachment.attachFile(pathToFile);
            MimeBodyPart docAttachmentSecond = new MimeBodyPart();
            docAttachmentSecond.attachFile(pathToSecondFile);
            emailContent.addBodyPart(docAttachment);
            emailContent.addBodyPart(docAttachmentSecond);
            msg.setContent(emailContent);
            Transport.send(msg);
        } catch (MessagingException e) {
            e.printStackTrace();
        } catch (IOException e) {
            e.printStackTrace();}}}
        }}}
```

Листинг А.5 – Исходный код класса «TextReplacer.java» серверной части программного обеспечения, осуществляющего часть генерации документа.

```
package main.java.com.company;
import java.util.HashMap;
import java.util.List;
import java.util.Map;
import java.util.regex.Matcher;
import java.util.regex.Pattern;
import org.apache.poi.xwpf.usermodel.*;
public class TextReplacer {
   private static Map<Integer, XWPFRun>
getPosToRuns(XWPFParagraph paragraph) {
      int pos = 0;
      Map<Integer, XWPFRun> map = new HashMap<Integer,
XWPFRun>(10);
      for (XWPFRun run : paragraph.getRuns()) {
         String runText = run.text();
         if (runText != null) {
```



```
      for (int i = 0; i < runText.length(); i++) {
            map.put(pos + i, run);
         }
         pos += runText.length();
      }
   }
      return (map);
   }

   public static <V> void replace(XWPFDocument document,
Map<String, V> map) {
      List<XWPFParagraph> paragraphs = document.getParagraphs();
      for (XWPFParagraph paragraph : paragraphs) {
         replace(paragraph, map);}}
```

```
    public static <V> void replace(XWPFDocument document, String
searchText, V replacement) {
        List<XWPFParagraph> paragraphs = document.getParagraphs();
        for (XWPFParagraph paragraph : paragraphs) {
            replace(paragraph, searchText, replacement);
        }
    }

    private static <V> void replace(XWPFParagraph paragraph,
Map<String, V> map) {
        for (Map.Entry<String, V> entry : map.entrySet()) {
            replace(paragraph, entry.getKey(), entry.getValue());
        }
    }

    public static <V> void replace(XWPFParagraph paragraph, String
searchText, V replacement) {
        boolean found = true;
```



```
while (found) {
            found = false;
            int pos = paragraph.getText().indexOf(searchText);
            if (pos >= 0) {
                found = true;
                Map<Integer, XWPFRun> posToRuns =
getPosToRuns(paragraph);
                XWPFRun run = posToRuns.get(pos);
                XWPFRun lastRun = posToRuns.get(pos + searchText.length() -
1);
                int runNum = paragraph.getRuns().indexOf(run);
                int lastRunNum = paragraph.getRuns().indexOf(lastRun);
```

```
                String texts[] = replacement.toString().split("\n");
                run.setText(texts[0], 0);
                XWPFRun newRun = run;
                 for (int i = 1; i < texts.length; i++) {
                    newRun.addCarriageReturn();
                    newRun = paragraph.insertNewRun(runNum + i);
                    newRun.setText(texts[i]);
                    newRun.setBold(run.isBold());
                    newRun.setCapitalized(run.isCapitalized());
                    newRun.setColor(run.getColor());

newRun.setDoubleStrikethrough(run.isDoubleStrikeThrough());
                    newRun.setEmbossed(run.isEmbossed());
                    newRun.setFontFamily(run.getFontFamily());
                    newRun.setFontSize(run.getFontSize());
                    newRun.setImprinted(run.isImprinted());
                    newRun.setItalic(run.isItalic());
                    newRun.setKerning(run.getKerning());
                    newRun.setShadow(run.isShadowed());
                    newRun.setSmallCaps(run.isSmallCaps());
                    newRun.setStrikeThrough(run.isStrikeThrough());
```



```
            newRun.setSubscript(run.getSubscript());
                    newRun.setUnderline(run.getUnderline());
                }
                for (int i = lastRunNum + texts.length - 1; i > runNum +
texts.length - 1; i--) {
                    paragraph.removeRun(i);
                }
            }
        }
```

```
    }
}
```

Листинг А.6 – Исходный код класса «WordReplacer.java» серверной части программного обеспечения, осуществляющего часть генерации документа.

```
package main.java.com.company.xandryex;
import main.java.com.company.xandryex.utils.TextReplacerS;
import org.apache.poi.util.Units;
import org.apache.poi.xwpf.usermodel.*;
import java.io.*;
import java.util.List;
public class WordReplacer {
    private XWPFDocument document;
    private TextReplacerS replacer;
    /**
     * Creates WordReplacer with file to modify.
     *
     * @param docxFile file of type docx.
     * @throws IOException thrown if file is not found or is not required
type.
     */
    public WordReplacer(File docxFile) throws IOException {
        InputStream inputStream = new FileInputStream(docxFile);
```



```
    init(new XWPFDocument(inputStream));
    }
    public WordReplacer()  {
    }
    /**
     * Creates WordReplacer with XWPFDocument to modify.
```

```
     * @param xwpfDoc to modify.
     */
    public WordReplacer(XWPFDocument xwpfDoc) {
        init(xwpfDoc);
    }
private void init(XWPFDocument xwpfDoc) {
        if (xwpfDoc == null) throw new NullPointerException();
        document = xwpfDoc;
        replacer = new TextReplacerS();
    }
    /**
     * Replaces all occurrences of a bookmark only in the text of the file
with a replacement string.
     * @param bookmark word to replace.
     * @param replacement word of replacement.
     */
    public void replaceWordsInText(String bookmark, String replacement)
{
        replacer.replaceInText(document, bookmark, replacement);
    }
    /**
     * Replaces all occurrences of a bookmark only in tables of the file with
a replacement string.
     * @param bookmark word to replace.
     * @param replacement word of replacement.
     */
```



```
  public void replaceWordsInTables(String bookmark, String replacement)
{
        replacer.replaceInTable(document, bookmark, replacement);
    }
    /**
```

```
     * Most of the time we want our template files untouched. Creates file
from path, saves the modified document to it and returns it.
     * @param path filepath (dirs + filename).
     * @return modified file.
   * @throws Exception thrown if some issues while saving occur - mostly
due to unavailable file or permissions.
     */
    public File saveAndGetModdedFile(String path) throws Exception {
        File file = new File(path);
        return saveToFile(file);
    }
    /**
     * Most of the time we want our template files untouched. Saves the
modified document to the given file and returns it.
     * @param file to save to.
     * @return modified file.
     * @throws Exception thrown if some issues while saving occur - mostly
due to unavailable file or permissions.
     */
    public File saveAndGetModdedFile(File file) throws Exception {
        return saveToFile(file);
    }
    public XWPFDocument getModdedXWPFDoc() {
        return document;
    }
    private File saveToFile(File file) throws Exception {
        FileOutputStream out = null;
```



```
  try {
            out = new FileOutputStream(file, false);
            document.write(out);
            document.close();
```

```
            return file;
        } catch (Exception e) {
            throw e;
        } finally {
            if (out != null) {
                out.flush();
                out.close();
             }
        }
    }
    public XWPFDocument replaceImage(XWPFDocument document,
String imageOldName, String imagePathNew, int newImageWidth, int
newImageHeight, int cor) throws Exception {
        try {
            int imageParagraphPos = -1;
            XWPFParagraph imageParagraph = null;

            List<IBodyElement> documentElements =
document.getBodyElements();
            for(IBodyElement documentElement : documentElements){
                imageParagraphPos ++;
                if(documentElement instanceof XWPFParagraph){
                    imageParagraph = (XWPFParagraph) documentElement;
                    if(imageParagraph != null && imageParagraph.getCTP() !=
null &&
imageParagraph.getCTP().toString().trim().indexOf(imageOldName) != -
1) {
                        break;
                    }
                }
            }
```

ОКОНЧАНИЕ ПРИЛОЖЕНИЯ А

```
if (imageParagraph == null) {
            throw new Exception("Unable to replace image data due to the
exception:\n"
                    + "'" + imageOldName + "' not found in in document.");
        }
        ParagraphAlignment oldImageAlignment =
imageParagraph.getAlignment();

        // remove old image
        document.removeBodyElement(imageParagraphPos);
        // now add new image
        // BELOW LINE WILL CREATE AN IMAGE
        // PARAGRAPH AT THE END OF THE DOCUMENT.
        // REMOVE THIS IMAGE PARAGRAPH AFTER
        // SETTING THE NEW IMAGE AT THE OLD IMAGE POSITION
        XWPFParagraph newImageParagraph =
document.createParagraph();
        XWPFRun newImageRun = newImageParagraph.createRun();
        //newImageRun.setText(newImageText);
        newImageParagraph.setAlignment(oldImageAlignment);
        try (FileInputStream is = new FileInputStream(imagePathNew)) {
            newImageRun.addPicture(is,
XWPFDocument.PICTURE_TYPE_JPEG, imagePathNew,
                    Units.toEMU(newImageWidth),
Units.toEMU(newImageHeight));
        }
        // set new image at the old image position
        imageParagraphPos-=cor;
        document.setParagraph(newImageParagraph,
imageParagraphPos);
        // NOW REMOVE REDUNDANT IMAGE FORM THE END OF
DOCUMENT
```

```
document.removeBodyElement(document.getBodyElements().size() - 1);
            return document;
        } catch (Exception e) {
            throw new Exception("Unable to replace image '" +
imageOldName + "' due to the exception:\n" + e);}}}
```