Федеральное государственное автономное
образовательное учреждение
высшего образования
«СИБИРСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

Институт Космических и Информационных Технологий
институт
Информационные системы
кафедра

УТВЕРЖДАЮ
Заведующий кафедрой

_____ С.А. Виденин
подпись, инициалы, фамилия

«___» _____ 20___г.

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА

09.03.02 Информационные системы и технологии

Разработка клиентской части веб-сервиса для регистрации случаев нарушения техники безопасности в транспортном филиале компании

Руководитель _____ С.А. Виденин
подпись, дата инициалы, фамилия

Выпускник _____ Н.А. Тулин
подпись, дата инициалы, фамилия

Рецензент _____ _____
подпись, дата инициалы, фамилия

Консультант _____ _____
подпись, дата инициалы, фамилия

Красноярск 2018

РЕФЕРАТ

Выпускная квалификационная работа по теме «Разработка клиентской части веб-сервиса для регистрации случаев нарушения техники безопасности в транспортном филиале компании» содержит 44 страницы текстового документа, 27 иллюстраций, 4 таблицы, 17 использованных источников.

ВЕБ СЕРВИС, МОБИЛЬНОЕ ПРИЛОЖЕНИЕ, ANDROID, ФУНКЦИОНАЛЬНОЕ РЕАКТИВНОЕ ПРОГРАММИРОВАНИЕ.

Объектом исследования является Заполярный транспортный филиал ПАО «ГМК "Норильский никель».

Предметом исследования является процесс подачи жалоб о нарушении техники безопасности на территории филиала и подконтрольного ему порта.

Целью работы является автоматизация и упрощение процесса подачи и обработки жалоб за счет внедрения веб-сервиса.

Задачи:

- опрос заказчика;
- изучение существующих технологий и осуществление выбора;
- составление технического задания;
- разработка мобильного приложения;
- тестирование;
- предоставление продукта заказчику.

В результате было разработано приложение на ОС Android в качестве клиентской части веб-сервиса, обеспечивающего автоматизированную работу с жалобами о нарушении техники безопасности на территории предприятия.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

В настоящее время, с повсеместным распространением и развитием веб и мобильных технологий, большинство организаций заинтересовано в своем веб-сайте или веб-приложении для наибольшего охвата клиентской базы либо для той или иной автоматизации внутренних процессов.

В связи с этим, было решено посвятить выпускную квалификационную работу именно этой тематике. Ценность работы заключается в получении таких практических навыков веб-разработчика как: взаимодействие с заказчиком, работа в команде, использование документации, разработка и тестирование.

Целью выпускной квалификационной работы является автоматизация и упрощение процесса подачи и обработки жалоб на нарушение требований техники безопасности в организации ЗТФ ПАО «ГМК Норильский никель».

Для достижения цели было решено совместными силами разработать веб-сервис. В данной работе будет осуществлена разработка клиентской части веб-сервиса, а именно – мобильного приложения на ОС Android.

Учитывая все вышесказанное, можно выделить следующие задачи:

- опрос заказчика;
- изучение существующих технологий и осуществление выбора;
- составление технического задания;
- разработка мобильного приложения;
- тестирование;
- предоставление продукта заказчику.

1 Анализ предметной области

1.1 Актуальность

Проект нацелен на автоматизацию и упрощение подачи жалоб на нарушение требований техники безопасности в транспортном филиале компании.

Предпосылками этого являются участившиеся несчастные случаи, связанные с нарушением техники безопасности как в самом заполярном транспортном филиале, так и в подконтрольном ему порту. Для анализа таких случаев и их предотвращения необходимо понимание общей картины.

Другой проблемой является трудоемкий процесс подачи жалоб на нарушение техники безопасности – на данный момент подача жалобы осуществляется через сайт с формой либо через электронную почту. Также следует принять во внимание то, что большая часть нарушений происходит в порту, где у работников нет доступа к персональным компьютерам.

Именно поэтому возникла потребность в возможности своевременной и мобильной подачи жалобы о нарушениях, используя мобильные устройства, такие как мобильный телефон или планшет.

Согласно условиям, напрашивается следующее решение:

- необходимо разработать сервер, который в перспективе сможет взаимодействовать с разными типами клиентов, такими как веб-сайт и мобильные клиенты различных операционных систем;
- необходимо разработать мобильное приложение (по договоренности было решено разработать приложение для ОС Android).

Исходя из всего вышесказанного, можно сделать вывод, что найти готовое существующее решение не представляется возможным. В этом и заключается актуальность разработки данного проекта.

1.2 Техническое задание

1.2.1 Общие положения

В настоящем документе приводится полный набор требований к Веб-сервису, необходимых для реализации.

Подпись Заказчика и Исполнителя на настоящем документе подтверждает их согласие с нижеследующими фактами и условиями:

- при реализации необходимо выполнить работы в объёме, указанном в настоящем Техническом задании.

- все неоднозначности, выявленные в настоящем Техническом задании после его подписания, подлежат двухстороннему согласованию между Сторонами.

Цели создания Веб-сервиса:

- с точки зрения создателей Веб-сервиса: упростить процесс подачи жалоб на нарушения техники безопасности и сделать его удобным для конечного пользователя, предоставить организации возможность для анализа полученных данных;

- с точки зрения организации: регистрация случаев нарушения ТБ для дальнейшего принятия решений их устранения;

- с точки зрения пользователя: упростить процесс подачи жалоб на нарушение ТБ, сделать его мобильным и доступным.

1.2.2 Функциональные требования

Сервер должен быть разработан с использованием технологии ASP.NET Core и веб-сервера Kestrel.

Мобильное приложение должно быть разработано под устройства с ОС Android 4.4 версии и выше.

Аутентификация и авторизация:

- требуется вынести модуль аутентификации и авторизации, взаимодействующей с основной базой данных организации, в отдельный сервис для возможности его использования в других проектах;

- аутентификация должна осуществляться посредством фамилии сотрудника и его табельного номера, хранящихся в корпоративной системе 1С;

- вход в приложение должен осуществляться как для обычного пользователя, так и для руководителя; необходимо предусмотреть создание новых ролей;

- любой пользователь должен иметь возможность установить дополнительную меру безопасности в виде пароля.

Общие требования:

- пользователь должен иметь возможность отправить заявку на нарушение ТБ, добавить к ней описание и прикрепить фото и видео файлы. Выбор адресата заявки должен осуществляться автоматически;

- пользователь должен иметь возможность редактировать заявки, если на них еще не поступил ответ;

- руководитель должен иметь возможность ответить на заявку и делегировать ее решение кураторам;

- как пользователь, так и руководитель должны видеть список всех заявок, которые были отправлены или присланы;

- все действия в приложении должны иметь возможность уведомления их участников как непосредственно в приложении, так и по электронной почте.

1.2.3 Требования к хранению

Должно быть предусмотрено хранение всех данных по нарушениям техники безопасности на сервере не менее года.

1.2.4 Модель данных

В проекте будут использованы следующие сущности модели данных:

- User – представляет пользователя;

- Auth – используется для аутентификации и авторизации;

- Attachment – вложение, прикрепленное к заявке (фото или видео);

- Application – заявка о нарушении ТБ.

Подробная модель данных, содержащая поля сущностей, их типы данных и описание представлена в таблице 1.

Таблица 1 - Модель данных

Сущность	Поля		
	Название	Тип данных	Описание
User	Id	Integer	Идентификатор
	RoleId	Integer	Идентификатор роли пользователя
	FirstName	String	Имя
	LastName	String	Фамилия
	PersonnelNumber	String	Табельный номер
	Email	String	Адрес электронной почты
	Password	String	Пароль
Role	Id	Integer	Идентификатор
	Name	String	Название роли
Application	Id	Integer	Идентификатор
	UserId	Integer	Идентификатор пользователя
	ReplyId	Integer	Идентификатор ответа
	Theme	String	Тема заявки
	Message	String	Текст заявки
	Status	Integer	Статус заявки: 0 – не рассмотрена; 1 – отвечена;
	CreatedAt	Date	Дата создания
	ModifiedAt	Date	Дата изменения
Attachment	Id	Integer	Идентификатор
	ApplicationId	Integer	Идентификатор заявки
	Type	Integer	Тип: 0 – фото; 1 – видео.
	Url	String	Адрес доступа к вложению
	ThumbnailUrl	String	Адрес доступа к миниатюре

Окончание таблицы 1

Сущность	Поля		
	Название	Тип данных	Описание
Reply	Id	Integer	Идентификатор
	UserId	Integer	Идентификатор пользователя
	Message	String	Текст ответа
	Danger	Int	Уровень опасности: 0 – легкий; 1 – средний; 2 – тяжелый;
	CreatedAt	Date	Дата создания
	ModifiedAt	Date	Дата изменения
Notification	Id	Integer	Идентификатор
	UserId	Integer	Идентификатор пользователя
	ApplicationId	Integer	Идентификатор заявки
	Type	Integer	Тип: 0 – создано; 1 – обновлено;

В графическом формате модель данных можно посмотреть на рисунке 1.

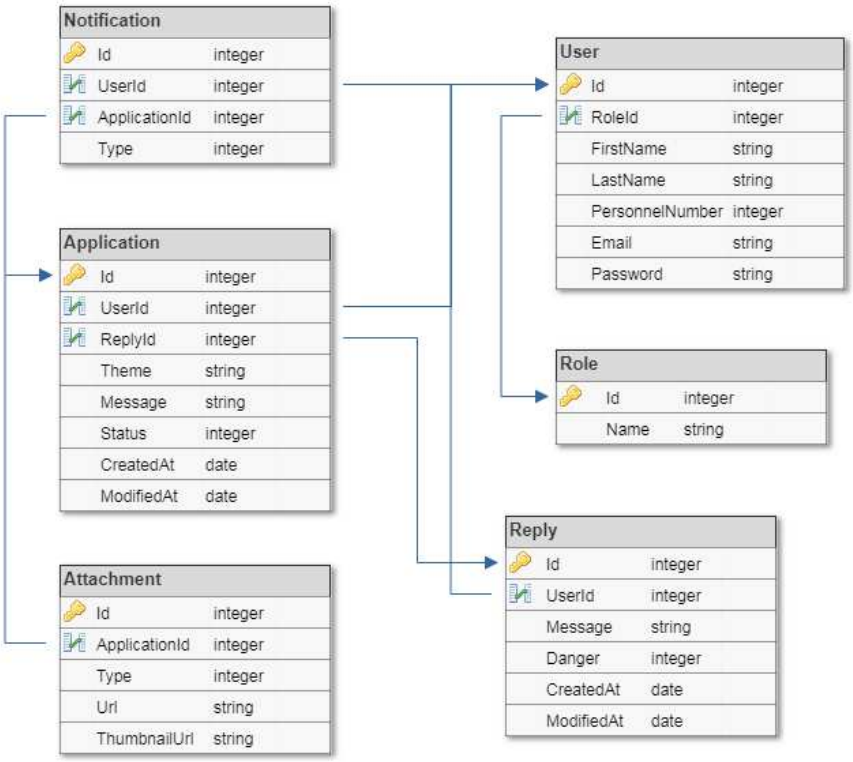

Рисунок 1 - Модель данных

Данные, которые будут передаваться в теле HTTP запроса (body) при обращении к серверу отображены в таблице 2.

Таблица 2 - Виды тел HTTP запросов к серверу

Наименование	Тело запроса	
	Название	Тип данных
SignInRequest	last_name	String
	personnel_number	String
	password (необязательно)	String
	client_id	String
RefreshTokenRequest	refresh_token	String
	client_id	String
UpdatePasswordRequest	password	String
RecoverPasswordRequest	last_name	String
	personnel_number	String
CreateApplicationRequest	theme	String
	message	String
	files	Массив Multipart
UpdateApplicationRequest	id	Integer
	theme	String
	message	String
	files	Массив Multipart
CreateReplyRequest	application_id	Integer
	message	String
	danger	Integer
UpdateReplyRequest	reply_id	Integer
	message	String
	danger	Integer

Данные, которые будут передаваться в теле HTTP ответа (body) при обращении к серверу отображены в таблице 3.

Таблица 3 - Виды тел HTTP ответов сервера

Наименование	Тело ответа	
	Название	Тип данных
AuthResponse	access_token	String
	refresh_token	String
	role	Integer

Окончание таблицы 3

Наименование	Тело ответа	
	Название	Тип данных
UserResponse	id	Integer
	first_name	String
	last_name	String
	email	String
	role	Integer
ApplicationResponse	id	Integer
	user_id	Integer
	reply_id	Integer
	theme	String
	message	String
	status	String
	created_at	Date
	modified_at	Date
AttachmentResponse	id	Integer
	application_id	Integer
	type	Integer
	url	String
	thumbnail_url	String
ReplyResponse	id	Integer
	user_id	Integer
	message	String
	danger	Integer
	created_at	Integer
	modified_at	Integer
NotificationResponse	id	Integer
	application_id	Integer
	type	Integer
StatisticsResponse	applications_number	Integer
	replies_number	Integer
	danger_average	Integer
	date	Date

В таблице 4 представлены возможные запросы к серверу с указанием пути, типа запроса, его краткого описания, необходимости авторизации и видов тел

HTTP запроса и ответа (для подробной информации о них см. предыдущие таблицы).

Таблица 4 - Возможные запросы к серверу

Путь	Тип запроса	Авторизация	GET параметры		Тело HTTP запроса	Тело HTTP ответа	Описание запроса
			Параметр	Тип данных			
api/auth/signin	POST	-	-		SignInRequest	AuthResponse	Аутентифицировать пользователя по персональному номеру
api/auth/refreshtoken	POST	-	-		RefreshTokenRequest	AuthResponse	Подлить токен авторизации
api/auth/password	PUT	+	-		UpdatePasswordRequest	-	Сменить или установить пароль
api/auth/recover-password	POST	-	-		RecoverPasswordRequest	-	Восстановить пароль через электронную почту
api/users/{id}	GET	+	id	Integer	-	UserResponse	Получить данные о пользователе
api/applications	GET	+	page	Integer	-	Массив ApplicationResponse	Получить список заявок о нарушении ТБ
			per_page	Integer			
			query	String			
api/applications/{id}	GET	+	id	Integer	-	ApplicationResponse	Получить данные заявки о конкретной заявке
api/applications	POST (multipart/form-data)	+	-		CreateApplicationRequest	ApplicationResponse	Создать новую заявку
api/applications	PUT (multipart/form-data)	+	-		UpdateApplicationRequest	ApplicationResponse	Редактировать заявку
api/applications/{id}/attachments	GET	+	id	Integer	-	Массив AttachmentResponse	Получить список вложений

Окончание таблицы 4

Путь	Тип запроса	Авторизация	GET параметры		Тело HTTP запроса	Тело HTTP ответа	Описание запроса
			Параметр	Тип данных			
api/attachments/{id}	GET	+	id	Integer	-	AttachmentResponse	Получить данные о вложении
api/attachments/{id}	DELETE	+	id	Integer	-	-	Удалить вложение
api/replies/{id}	GET	+	id	Integer	-	ReplyResponse	Получить данные об ответе на заявку
api/replies	POST	+	-		CreateReplyRequest	ReplyResponse	Добавить ответ на заявку
api/replies	PUT	+	-		UpdateReplyRequest	ReplyResponse	Редактировать ответ на заявку
api/notifications	GET	+	-		-	Массив NotificationResponse	Получить список уведомлений
api/statistics	GET	+	year	Integer	-	Массив StatisticsReponse	Получить статистику
			month	Integer			

Коды возможных HTTP ответов:

- 200 – возвращается при каждом успешном запросе;
- 400 – возвращается при неверном запросе, сформированным клиентом;
- 401 – возвращается, если пользователь не авторизован;
- 403 – возвращается, если уровень доступа пользователя недостаточен;

1.2.5 Перспективы развития

При разработке веб-сервиса необходимо предусмотреть возможность будущей интеграции как клиентов иных мобильных ОС, так и веб-сайта.

2 Теоретическая часть

2.1 Интегрированные среды разработки

IntelliJ IDEA — интегрированная среда разработки программного обеспечения для многих языков программирования, в частности Java, JavaScript, Python, разработанная компанией JetBrains [1].

Первая версия появилась в январе 2001 года и быстро приобрела популярность как первая среда для Java с широким набором интегрированных инструментов для рефакторинга, которые позволяли программистам быстро реорганизовывать исходные тексты программ. Дизайн среды ориентирован на продуктивность работы программистов, позволяя сконцентрироваться на функциональных задачах, в то время как IntelliJ IDEA берёт на себя выполнение рутинных операций.

Начиная с шестой версии продукта IntelliJ IDEA предоставляет интегрированный инструментарий для разработки графического пользовательского интерфейса. Среди прочих возможностей, среда хорошо совместима со многими популярными свободными инструментами разработчиков, такими как CVS, Subversion, Apache Ant, Maven и JUnit. В феврале 2007 года разработчики IntelliJ анонсировали раннюю версию плагина для поддержки программирования на языке Ruby.

Начиная с версии 9.0, среда доступна в двух редакциях: Community Edition и Ultimate Edition. Community Edition является полностью свободной версией, доступной под лицензией Apache 2.0, в ней реализована полная поддержка Java SE, Groovy, Scala, а также интеграция с наиболее популярными системами управления версиями. В редакции Ultimate Edition, доступной под коммерческой лицензией, реализована поддержка Java EE, UML-диаграмм, подсчёт покрытия кода, а также поддержка других систем управления версиями, языков и фреймворков.

Поддерживает как Java, так и Kotlin. Интерфейс см. на рисунке 2.

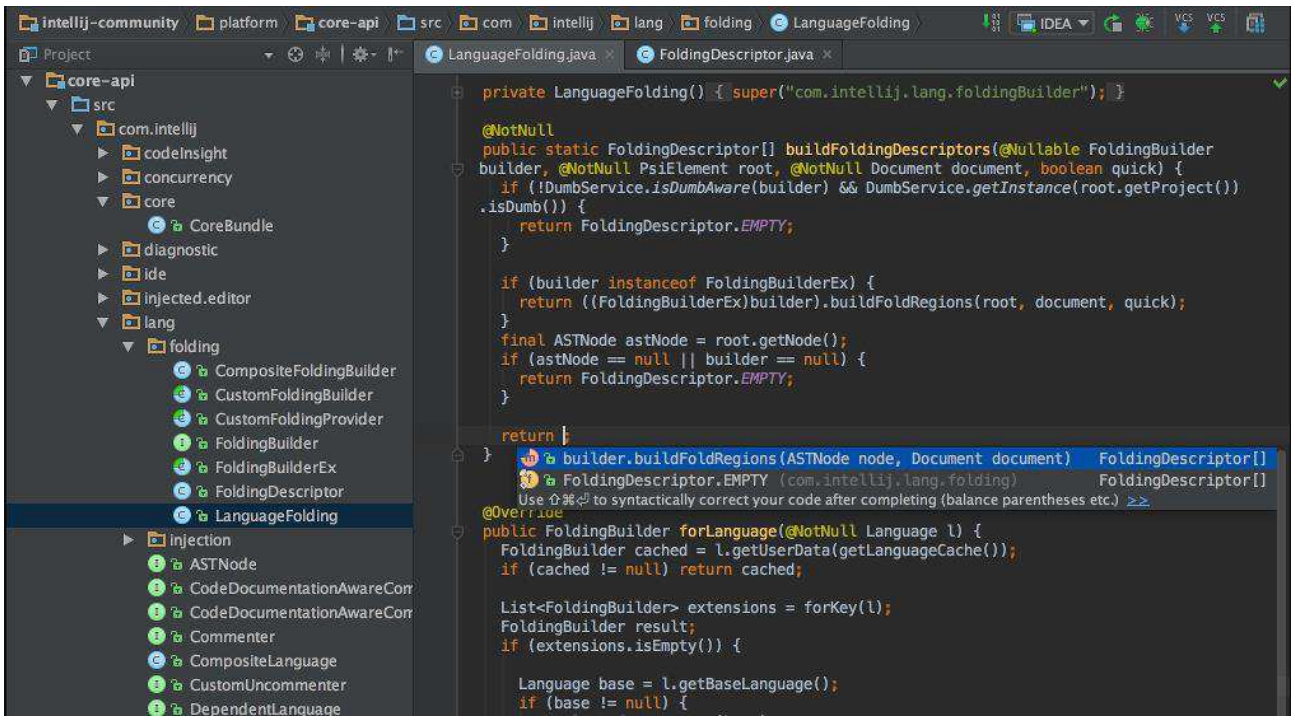

Рисунок 2 - Интерфейс среды разработки IntelliJ IDEA

Android Studio — это интегрированная среда разработки (IDE) для работы с платформой Android, анонсированная 16 мая 2013 года на конференции Google I/O.

IDE находилась в свободном доступе начиная с версии 0.1, опубликованной в мае 2013, а затем перешла в стадию бета-тестирования, начиная с версии 0.8, которая была выпущена в июне 2014 года. Первая стабильная версия 1.0 была выпущена в декабре 2014 года, тогда же прекратилась поддержка плагина Android Development Tools (ADT) для Eclipse.

Android Studio, основанная на программном обеспечении IntelliJ IDEA от компании JetBrains, - официальное средство разработки Android приложений. Данная среда разработки доступна для Windows, OS X и Linux. 17 мая 2017, на ежегодной конференции Google I/O, Google анонсировал язык Kotlin, используемый в Android Studio, официальным языком программирования для платформы Android в добавление к Java и C++.

Интерфейс см. на рисунке 3.

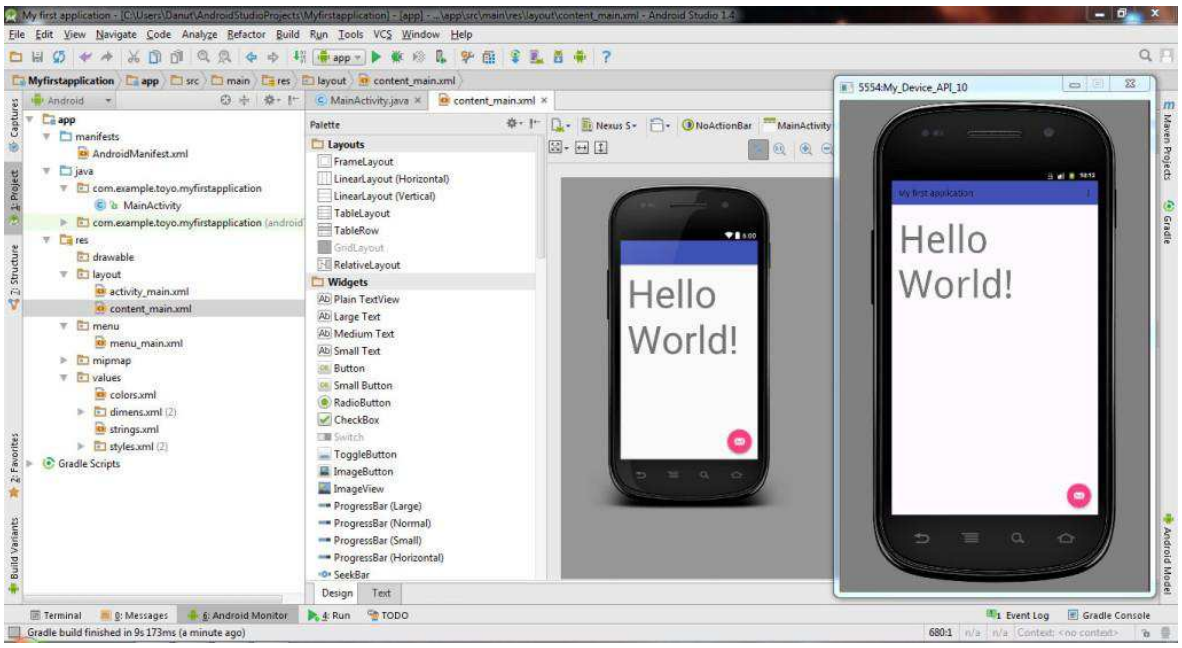

Рисунок 3 - Интерфейс среды разработки Android Studio

2.2 Функциональное реактивное программирование

Как известно, мобильные приложения по своей природе являются достаточно сложными программами с большим количеством запутанной и сложной событийной логики. Данный факт порождает огромное количество проблем при разработке и дальнейшей поддержки программы. Эти проблемы можно в какой-то степени свести на нет, используя подход функционального реактивного программирования [2, с.2].

Функциональное реактивное программирование (FRP) – это особый метод программирования, улучшающий код в той области, которая является типичным источником сложности (а значит и багов) – распространение событий.

FRP может быть рассмотрен с разных точек зрения, как:

- замена широко используемому паттерну "наблюдатель", так же известному как слушатели и коллбэки;

- компонуемый, модульный способ написания событийной логики;

- иной способ мышления: программа - это реакция на ввод, либо поток данных;

- что-то, что привносит порядок в управление состоянием программы;

- нечто фундаментальное: существует мнение, что каждый, кто решает проблемы с помощью паттерна "наблюдатель", рано или поздно изобретет FRP;

- легковесная программная библиотека для стандартного языка программирования;

- полноценный встраиваемый язык для логики состояний.

Реактивное программирование - широкий термин, означающий, что программа:

- основана на событиях;

- действует в ответ на ввод;

- рассматривается как поток данных, а не традиционный поток управления.

Согласно манифесту реактивного подхода [3], реактивные системы должны соответствовать принципам:

- отзывчивости: система, по возможности, должна реагировать своевременно. Отклик должен быть быстрым и последовательным, что упрощает обработку ошибок и повышает доверие пользователей к продукту;

- устойчивости: даже при сбое, система должна быть отзывчивой. Устойчивость достигается за счет репликации, автономности, изоляции и делегирования. Все части системы должны восстанавливаться без ущерба для самой системы. За восстановление компонент отвечает другой (внешний) компонент;

- эластичности: система должна быть отзывчивой при любой нагрузке. Реактивные системы могут реагировать на изменение скорости ввода, путем увеличения или уменьшения используемых ресурсов. Системы обеспечивают эластичность, используя масштабирующие алгоритмы и сильные стороны используемых платформ;

- ориентированности на сообщения: реактивные системы полагаются на асинхронную передачу сообщений, для разграничения компонент и обеспечения слабой связанности, изоляции и прозрачности. Это позволяет управлять нагрузкой и контролировать поток событий. Неблокирующие связи позволяют потреблять ресурсы только во время активности, снижая общую нагрузку.

2.2.1 Сравнение со слушателями и коллбэками

У слушателей и коллбэков есть множество недостатков, которых не имеет FRP. Они заключаются в следующем:

- непредсказуемый порядок – в сложной сети слушателей, порядок, в котором приходят события может зависеть от порядка, в котором вы зарегистрировали слушателей, что может быть критичным. В FRP неважно в каком порядке обрабатываются события, потому что это никак не отслеживается;

- пропущенное первое событие – сложно гарантировать, что все слушатели были зарегистрированы к моменту срабатывания первого события. В FRP существуют транзакции – вот и гарантии;

- беспорядочные состояния – коллбэки возвращают ваш код в стиль конечных автоматов, который быстро превращается в хаос;

- проблемы с потоками – попытка сделать слушатели потокобезопасными может привести к взаимной блокировке, и трудно гарантировать, что ни один коллбэк не отработает после дерегистрации события. FRP решает эти проблемы.

- утечки – если вы забудете дерегистрировать слушателя, произойдет утечка памяти. Слушатели меняют естественную зависимость данных, но не меняют зависимость поддержания актуальности. FRP способен делать это.

- случайная рекурсия – порядок, в котором вы обновляете локальные состояния и извещаете слушателей может быть опасным – сделать ошибку легко. FRP устраняет эту проблему.

2.2.2 Основные понятия

Поток – это последовательность, состоящая из постоянных событий, отсортированных по времени. В нем может быть три типа сообщений: значения (данные некоторого типа), ошибки и сигнал о завершении работы (см. рисунок 4, где линия – поток, кружочки – события).

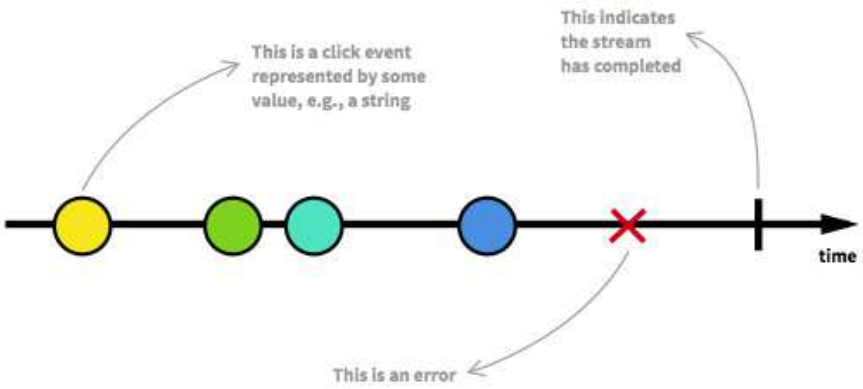

Рисунок 4 - Реактивный поток

Обозреватель (observable, источник данных) – т.н. трансформированный поток или источник данных, отправляющий все события и их объекты своим подписчикам.

Обозреватель бывает двух типов:

- «Холодный» (Cold observable) – источник, отсылающий новые данные только после подписки.

- «Горячий» (Hot observable) – источник, отсылающий данные сразу после создания.

2 Наблюдатель (Observer) – функция, подписанная на события потока.

3 Операторы – команды, позволяющие трансформировать потоки.

2.2.3 Пример использования подхода

Допустим, мы хотим регистрировать двойные, тройные и т.д. нажатия пользователя на кнопку, но игнорировать одинарные.

Первым шагом нам нужно разбить события нажатия на группы, согласно определенным промежуткам времени (допустим 250 миллисекунд).

Вторым шагом мы должны получить размеры эти групп.

И последним шагом – отфильтровать их по размеру (нас интересуют те, которые имеют размер 2 и более)

Результатом будет опять же поток событий (но уже преобразованный согласно нашим целям) на который можно подписаться и использовать по своему усмотрению.

Весь процесс изображен на рисунке 5 (серые прямоугольники – операторы).

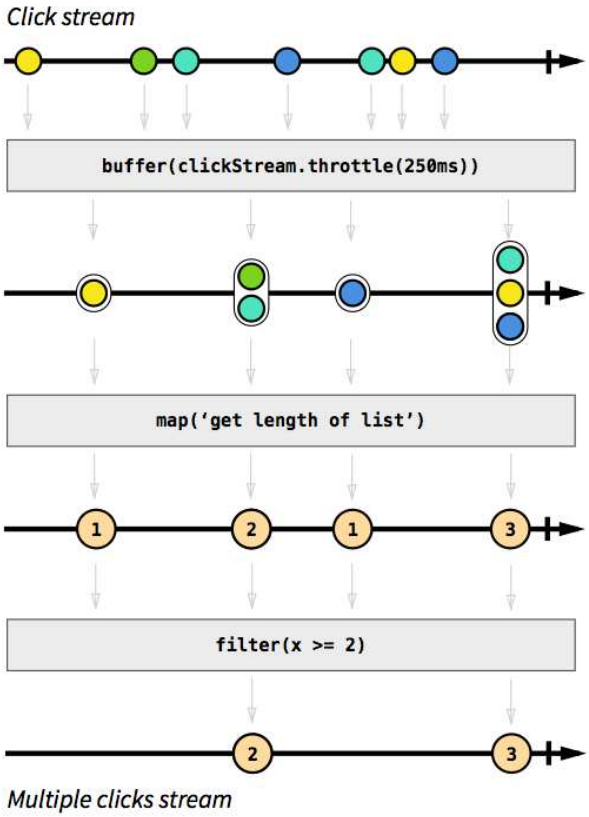

Рисунок 5 - Преобразования реактивного потока

2.2.4 Реализации FRP-подхода на ОС Android

На ОС Android самой популярной реализацией реактивного подхода является библиотека ReactiveX [4]. На данный момент она довольно распространена и используется во многих фреймворках.

ReactiveX реализует FRP-подход на следующих платформах: Java, Javascript, C#, Scala, Clojure, C++, Lua, Ruby, Python, Go, Groovy, JRuby, Kotlin, Swift, PHP, Elixir, Dart.

Библиотека одинаково успешно может применяться как для фронтенда, так и для бэкенда, а также кросплатформенной разработки.

Где используется ReactiveX:

- в Netflix Open Source Software Center для пользовательского интерфейса. Также Netflix принимает активное участие в улучшении RxJS (ветвь ReactiveX на JavaScript).

- в Microsoft, принимающей активное участие в разработке Rx.NET, который входит в .NET Foundation.

- в SoundCloud используется RxJava для android-приложений для упрощения асинхронного конкурентного кода.

В ОС Android ReactiveX представлена библиотеками RxJava и RxAndroid [5]. RxJava является главным модулем, а RxAndroid расширением, которое позволяет выполнять реактивные функции в разных потоках. Код, представленный на рисунке 6, необходим для того, чтобы избежать блокировки главного потока, при подписке на обозревателя.

```
1. someObservable
2.     .subscribeOn(Schedulers.io())
3.     .observeOn(AndroidSchedulers.mainThread())
4.     .subscribe(someSubscriber);
```

Рисунок 6 - Встроенные объекты RxAndroid

Рассмотрим пример ситуации, в которой пригодилась бы ReactiveX: имеется список адресов изображений, которые необходимо вывести на экран. Допустим, что у нас есть функция *downloadImage*, которая может скачать изображение и вернуть observable с ним и объект *imageViewAdapter*, отвечающий за отрисовку изображений на экране. Тогда необходимо преобразовать список адресов в observable и с помощью оператора *map* применить к каждому адресу функцию *downloadImage*. Все, что остается – подписать *imageViewAdapter* на результат этого реактивного потока. Код представлен на рисунке 7.

```
List<string> links = listof(
    "http://.../image1.png",
    "http://.../image2.png",
    ...
);

Observable<String> observable = Observable.from(links);

observable
    .map({ link -> downloadImage(link) })
    .subscribe({ image -> imageViewAdapter.add(image) })
```

Рисунок 7 - Реактивное получение изображений

3 Разработка мобильного приложения

3.1 Программные средства

В качестве IDE была выбрана IntelliJ IDEA от JetBrains. Ее преимущество в частых и регулярных обновлениях – крупные выходят каждые несколько месяцев. В них добавляются как свежие функциональные возможности, так и улучшается интерфейс. Также среда очень гибко настраивается под предпочтения пользователя, что ускоряет разработку.

Выбор языка разработки стоял между Java и Kotlin. Выбор был сделан в пользу Kotlin, так как он более лаконичен, в отличии от строгой Java, что также влияет на скорость написания кода. К тому же в Kotlin есть null-безопасность, то есть присутствует проверка NullPointerException на этапе компиляции. Но самый важный фактор – язык Kotlin заточен под функциональное программирование, что идеально вписывается в принятые архитектурные решения проекта.

3.2 Архитектура

В виду использования реактивного программирования в проекте, в качестве архитектурного решения в проекте решено было использовать open-source библиотеку RxPM [6]. В ней реализован паттерн Presentation Model [7] с использованием реактивного подхода. По словам авторов, RxPM довольно схож с MVVM-паттерном, за исключением технологии биндингов (binding). Вместо этого необходимо одноразово вручную связать свойства из View со свойствами из Presentation Model (см. рисунок 8).

```
1.    pm.email bindTo _view.signIn_email
2.    pm.pass bindTo _view.signIn_pass
```

Рисунок 8 - Привязка View к Presentation Model

Также было принято решение разрабатывать мобильное приложение в стиле «одна активити» (single activity application). Такое устройство приложения

позволяет создать более плавные переходы между интерфейсными окнами, в целом ускорить работу приложения и сделать его более отзывчивым. Для контроля за переходами, хранением состояний и управлением жизненного цикла интерфейсных окон была использована библиотека Conductor [8]. RxPM имеет полную совместимость с ней и переопределяет ее класс Controller, который является представлением интерфейсного окна (далее для краткости при упоминании интерфейсных окон будет использоваться термин «контроллер» [не путать с контроллерами из паттерна MVC]), связывая со своим Presentation Model.

Для уменьшения связности компонент (loose-coupling) и внедрения зависимостей была использована библиотека Dagger 2 от Google [9] (форк от Dagger компании Square). С помощью нее можно разбить зависимости на модули и внедрять их в контроллеры. Главный компонент Dagger будет привязан к жизненному циклу activity мобильного приложения, из которого создаются все контроллеры приложения.

3.3 Реализация реактивного подхода

Как было упомянуто прежде, в проекте используется реактивная библиотека RxAndroid в связке с RxPM. Также для более красивой и лаконичной привязки свойств виджетов к реактивным свойствам из Presentation Model используется библиотека RxBinding от Jake Wharton [10].

В Presentation Model библиотеки RxPM представлено три вида реактивных свойств:

- *State* – представляет состояние слоя *View* и содержит последнее переданное в него значение, например, может отображать завершился ли HTTP-запрос;

- *Action* – представляет пользовательские действия, например, может реагировать на нажатия кнопок во *View*;

- *Command* – представляет команду, посланную во *View*, к примеру, вывести какое-либо сообщение на экран.

На рисунке 9 представлен пример из Presentation Model (PM) *AppListPM*, которое работает в паре с контроллером *AppListController*, отображающим список заявок на экране.

```
1.   //states
2.   val applications = State<List<Application>>(emptyList())
3.   val inProgress = State(false)
4.
5.   //commands
6.   val onError = Command<Throwable>()
7.
8.   //actions
9.   val refreshApps = Action<Unit>()
10.  val loadMoreApps = Action<Unit>()
11.  val findApps = Action<CharSequence?>()
```

Рисунок 9 - Пример реактивных свойств Presentation Model

Описание представленных свойств:

- состояние *applications* – содержит список заявок, которые необходимо вывести на экран;

- состояние *inProgress* – отображает выполняется ли HTTP-запрос в данный момент;

- команда *onError* – используется для вывода ошибок на экран;

- действие (action) *refreshApps* – получить первую страницу заявок с сервера, при этом все ранее полученные заявки будет стерты;

- действие *loadMoreApps* – получить следующую страницу заявок с сервера;

- действие *findApps* – применить фильтр поиска к заявкам.

Также в PM прописывается, как необходимо реагировать на действия (actions). Для примера см. рисунок 10. В этом примере происходит подписка реактивного состояния *applications* на действие *refreshApps* с использованием реактивных преобразований.

```
refreshApps.observable
    .skipWhileInProgress(inProgress.observable)
    .flatMapSingle {
        resetPage()
        resetSearch()
        api.apps(page).format()
    }
    .retry()
    .map { applications.clear().addAll(it) }
    .subscribe(applications.consumer)
    .untilDestroy()

private fun <T> Single<T>.format(): Single<T> {
    return this
        .subscribeOn(Schedulers.io())
        .observeOn(AndroidSchedulers.mainThread())
        .bindProgress(inProgress.consumer)
        .doOnError(onError.consumer)
}
```

Рисунок 10 - Пример подписки состояния на действие

Построчно разберем что тут происходит:

- оператор *skipWhileInProgress* – если значение свойства *inProgress* в аргументе будет истинным цепочка преобразований будет прервана;

- оператор *flatMapSingle* – позволяет осуществить какие-либо действия внутри реактивного потока, в реактивный поток будет выброшено последнее упомянутое значение;

- функция *resetPage* – сбрасывает счетчик страниц на первую;

- функция *resetSearch* – очищает фильтр поиска;

- функция *api.apps(page)* – обращается к серверу за заявками с указанной страницы;

- функция- помощник *format* – указывает в каком потоке осуществить подписку на результат HTTP-запроса *api*, чтобы пользовательский интерфейс не заблокировался, а также подписывает свойства *InProgress* и *onError* на эту операцию;

- оператор *retry* позволяет возобновить подписку при возникшей ошибке;

- в операторе *map* (он аналогичен *flatMapSingle*) очищается список заявок, а затем добавляются заявки, полученные из запроса, и выбрасываются в поток;

- в операторе *subscribe* указывается какое состояние будет подписано на полученное значение из реактивного потока;

- оператор *untilDestroy* является внутренним оператором библиотеки RxPM и отвечает за жизненный цикл подписки, то есть подписка будет уничтожена вместе с PM, для того чтобы избежать утечек памяти.

После этого необходимо только подписаться на реактивные свойства PM в соответствующем контроллере. Для примера см. рисунок 11.

```
1.    //states and commands
2.    pm.inProgress.observable bindTo sr.refreshing()
3.
4.    pm.applications.observable bindTo { rvAdapter.setItems(it) }
5.
6.    pm.onError.observable bindTo onError()
7.
8.    //actions
9.    sr.refreshes() bindTo pm.refreshApps.consumer
```

Рисунок 11 - Пример подписки виджетов на реактивные свойства PM

Пояснения к рисунку 11:

- *sr – SwipeRefreshLayout* – виджет, позволяющий обновлять данные перетаскиванием списка сверху вниз (см. рисунок 12);

- функции *refreshing* и *refreshes* входят в упомянутую выше библиотеку RxBinding;

- *rvAdapter* – адаптер списка заявок, отвечающий за их отрисовку.

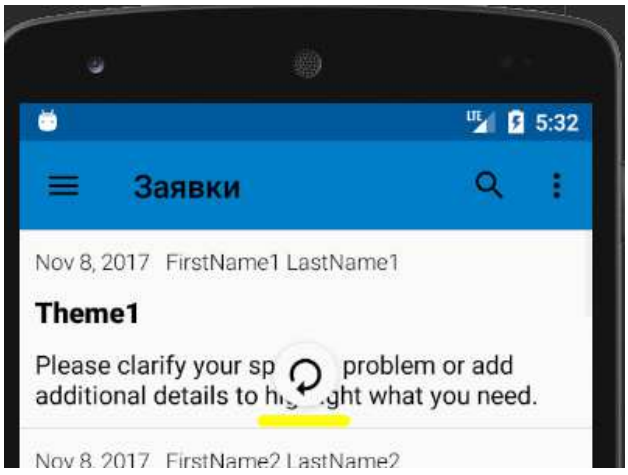

Рисунок 12 - Индикатор обновления данных

3.4 Жизненный цикл activity и контроллеров

Вследствие того, что мобильное приложение по своей природе довольно интерактивно, в его работе возникают ситуации, требующие перерисовки пользовательского интерфейса, такие как:

- изменение ориентации экрана;
- сворачивание и разворачивание приложения;
- вызов экранной клавиатуры;
- и другие.

При разработке мобильного приложения все эти нюансы должны быть учтены, иначе возможна потеря тех или иных внутренних либо введенных пользователем данных.

Жизненный цикл контроллеров и activity см. на рисунке 13 [11].

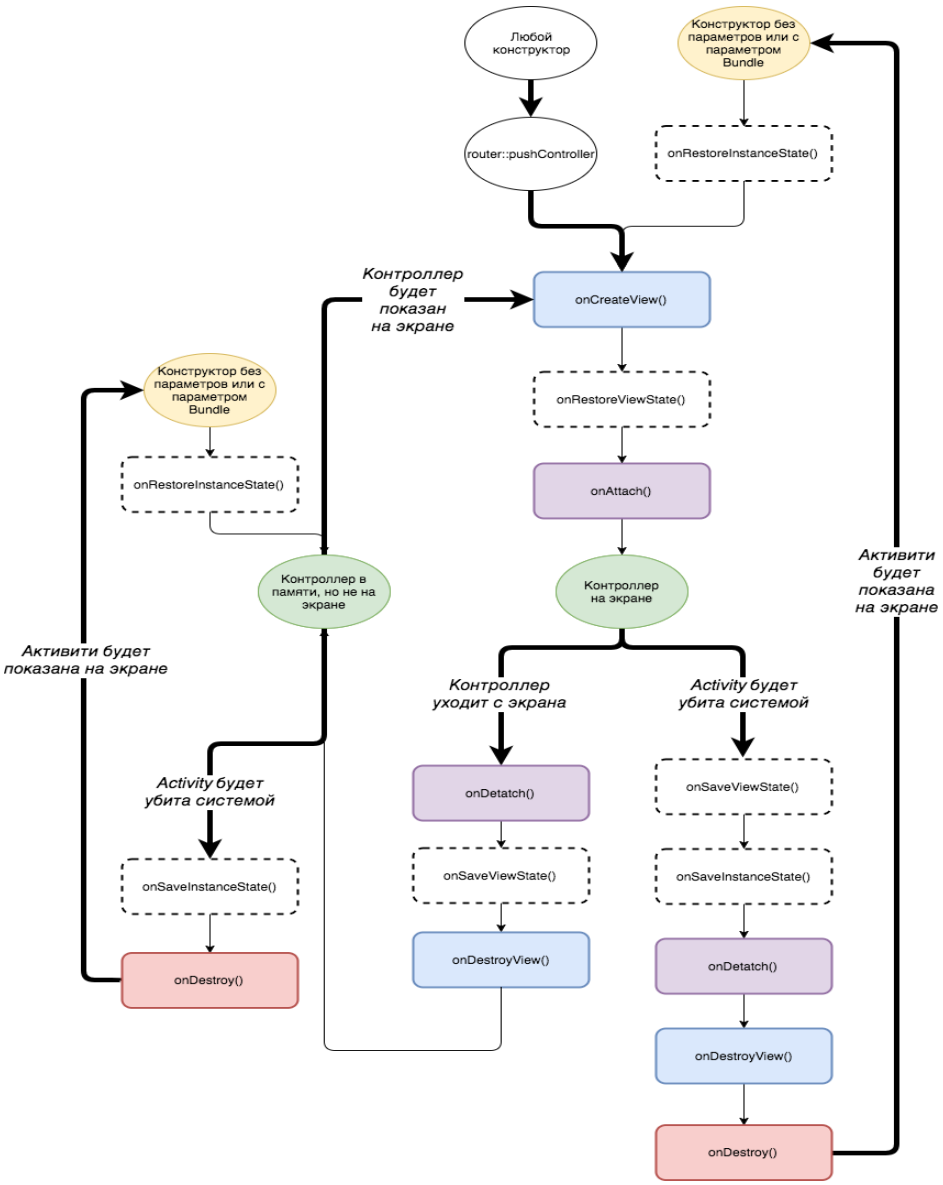

Рисунок 13 - Жизненный цикл activity и контроллеров

Благодаря библиотеке RxPM, все данные, относящиеся к бизнес логике (то есть те, которые привязаны к Presentation Model – сюда можно отнести все input-виджеты) сохраняют свое состояние автоматически, т.к. Presentation Model не уничтожается и не зависит от жизненного цикла контроллера. При пересоздании контроллера вызывается метод *OnBindPresentationModel*, определенный в интерфейсе библиотеки, и всем полям при привязке возвращаются их потерянные значения.

Те же данные, которые не содержатся в бизнес-логике, а относятся только лишь к пользовательскому интерфейсу можно сохранить в *Bundle* (объект, который предназначен для передачи состояния при смене конфигурации приложения и не зависит от жизненного цикла элементов интерфейса) в специально предназначенных для этого переопределенных методах контроллера (которые заимствованы у *Fragment* и *Activity*) – *onSaveInstanceState* и *onRestoreInstanceState*.

Для примера см. 14. Это кусок кода из контроллера *AttachmentViewController*, который отображает фото либо видео. В листинге видно, что сохраняется состояние *videoPlaying,* то есть проигрывается видео или нет, а также *videoProgress*, отвечающее за момент времени на котором находится проигрыватель. При смене ориентации экрана либо сворачивании приложения вызывается метод *OnSaveInstanceState*, в котором настройки записываются в *Bundle*. Затем, при разворачивании приложения либо завершении поворота экрана, вызывается метод *OnRestoreInstanceState*, в который поступает тот же самый *Bundle* с записанными настройками и состояниям возвращаются их значения.

```
1.    override fun onSaveInstanceState(outState: Bundle) {
2.        super.onSaveInstanceState(outState)
3.        outState.putBoolean(KEY_VIDEO_PLAYING, videoPlaying)
4.        outState.putInt(KEY_VIDEO_PROGRESS, videoProgress)
5.    }
6.
7.    override fun onRestoreInstanceState(savedInstanceState: Bundle) {
8.        super.onRestoreInstanceState(savedInstanceState)
9.        videoPlaying = savedInstanceState.getBoolean(KEY_VIDEO_PLAYING, true)
10.       videoProgress = savedInstanceState.getInt(KEY_VIDEO_PROGRESS, 0)
11.   }
```

Рисунок 14 - Пример сохранения состояния контроллера

3.5 Разделение пользовательских прав доступа

Согласно требованиям заказчика в приложении должно быть несколько групп пользователей: как минимум обычный пользователь и начальник. Предполагается, что у них должен быть немного различный интерфейс и функционал.

Так как в окружении Android нет встроенного решения для этой проблемы, то было решено написать модуль, который бы выдавал разные контроллеры для разных групп пользователей.

В примере (см. рисунок 15) показано, что в контроллер *AppDetails* (детали заявки) могут зайти как обычный пользователь, так и начальник, но каждому из них будет назначен контроллер с разным интерфейсом и функционалом. Ниже можно увидеть, что в контроллер *AttachmentView* (просмотр медиа-вложения) может зайти любой авторизованный пользователь, а в контроллер *AddApp* (создание новой заявки) только обычный пользователь (так как начальник не может создавать заявки).

```
1.    ControllerTypes.AppDetails to hashMapOf (
2.
3.      UserRights.USER to { args: Array<out Any>? ->
4.        AppDetailsUserController(args!![0] as Application)
5.      },
6.
7.      UserRights.HEAD to { args: Array<out Any>? ->
8.        AppDetailsHeadController(args!![0] as Application)
9.      }
10.
11.   ),
12.
13.   ControllerTypes.AttachmentView to hashMapOf (
14.
15.     UserRights.ANY_AUTH to { args: Array<out Any>? ->
16.       AttachmentViewController(args!![0] as Attachment)
17.     }
18.
19.   ),
20.
21.   ControllerTypes.AddApp to hashMapOf (
22.
23.     UserRights.USER to _: Array<out Any>? ->
24.       AddAppController()
25.     }
26.
27.   )
```

Рисунок 15 - Пример разделения типов контроллеров по разным группам пользователей

3.6 Взаимодействие с сервером

Так как API сервера создано в REST-стиле, то необходимо использовать HTTP-клиент. Самым популярным и гибким HTTP-клиентом на Android является Retrofit от компании Square [12]. Он позволяет использовать middleware-перехватчики (например, для авторизации), предоставляет автоматическую сериализацию и десериализацию данных и определяет API в виде интерфейса.

В примере (см. рисунок 16) показан пример GET-запроса. В аннотации к методу указывается тип и путь, передаваемый в адресную строку. В аннотации к

параметрам указывается название и тип параметра (*@Query* – обычный GET-параметр, *@QueryMap* – маппинг множества параметров, который в дальнейшем будет преобразован в такие же обычные параметры).

```
1.    @GET("apps")
2.    fun apps(
3.        @Query("page") page: Int,
4.        @Query("per_page") perPage: Int = Constants.PER_PAGE_DEFAULT,
5.        @QueryMap query: Map<String, String> = mapOf()
6.    ): Single<List<Application>>
```

Рисунок 16 - пример GET запроса

3.7 Механизм аутентификации и авторизации

Так как, подход REST подразумевает отсутствие хранения состояния на сервере, то в качестве средства аутентификации и авторизации решено было использовать JSON Web Tokens (JWT).

Также, для большей безопасности, решено было использовать пару токенов: *AccessToken* (для авторизации) и *RefreshToken* (для обновления *AccessToken*). Предполагается, что *AccessToken* отправляется при каждом запросе к серверу и потому уязвим для перехвата и ему требуется назначить короткое время жизни. *RefreshToken* же получит значительно большее время жизни. Таким образом, при перехвате *AccessToken* злоумышленник получит доступ к аккаунту пользователя лишь на малое время.

Все, что требуется от клиента – аутентифицироваться, получить токены и сохранить их локально. Затем отправлять *AccessToken* при каждом запросе к серверу для авторизации. *RefreshToken* должен отправляться автоматически для обновления первого токена.

Для сохранения токенов было решено использовать механизм *SharedPreferences*. Google рекомендует использовать его для хранения небольшого количества строковых данных.

Для простоты, было решено написать модуль аутентификации и авторизации, который бы получал токены с помощью API, сохранял их и выдавал другим модулям по запросу. В проекте он называется *AuthManager*. Он может:

- аутентифицироваться, используя пользовательские данные (при этом, сервер пришлет как токены, так и тип пользовательских прав);

- сохранить токены в SharedPreferences и считать их оттуда;

- обновить *AccessToken*, используя *RefreshToken*;

- деаутентифицироваться, стереть токены и известить подписанные модули;

- отдать токены по запросу других модулей;

- отдать тип пользовательских прав по запросу других модулей;

- определить авторизован ли пользователь и вернуть результат в другой модуль по запросу.

Схему аутентификации см. на рисунке 17.

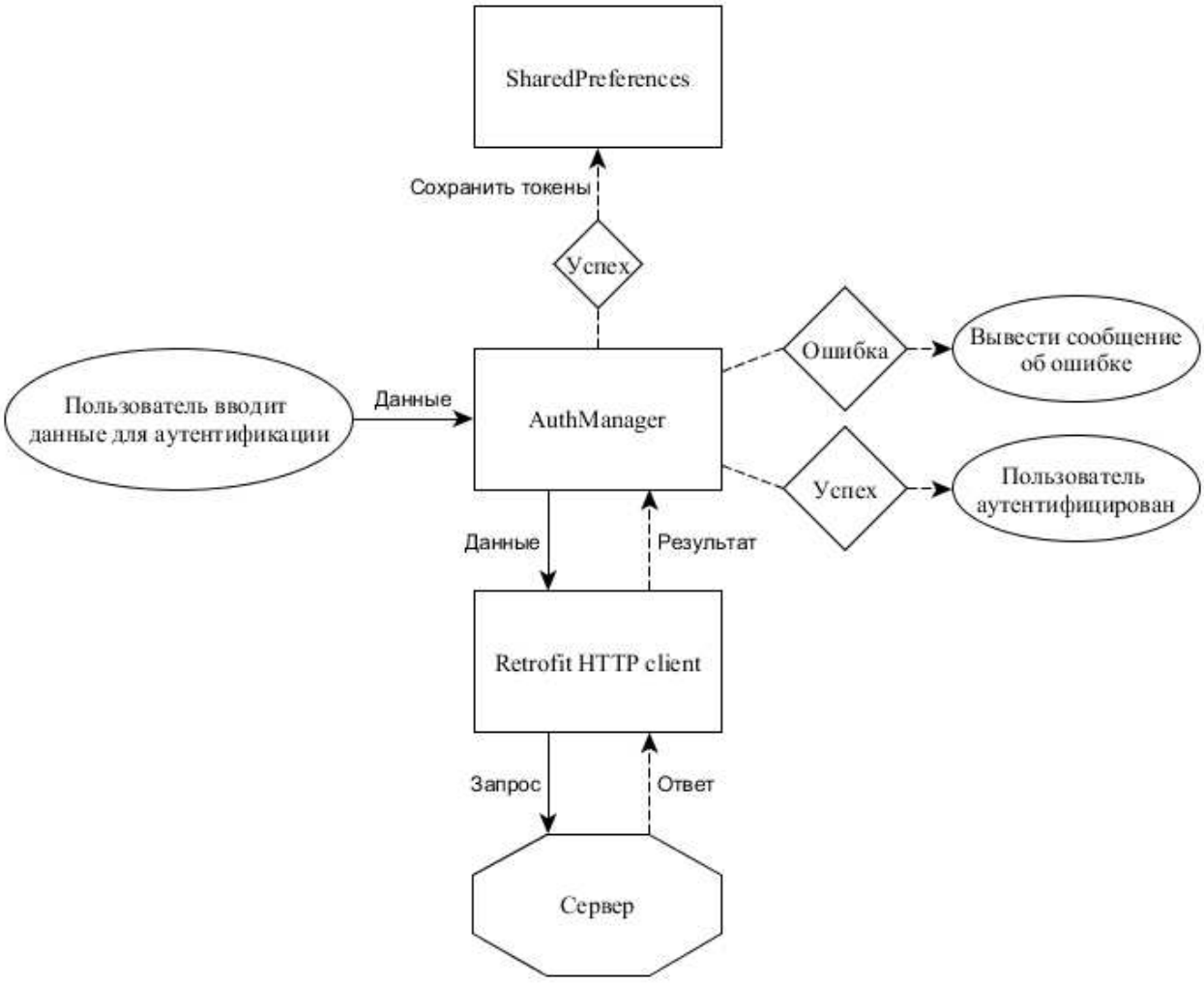

Рисунок 17 - Схема аутентификации

Для передачи токена при каждом запросе используется перехватчик *TokenAuthInterceptor* от Retrofit. Он считывает токен из *AuthManager* и добавляет его в заголовок запроса в формате «*Bearer [AccessToken]*».

Чтобы определить, что *AccessToken* истек, используется еще один middleware от Retrofit – *TokenAuthenticator*. Он вызывается каждый раз, когда сервер возвращает код ошибки 401 (Unauthorized). Используя три попытки, *TokenAuthenticator* через *AuthManager* пытается обновить *AccessToken*. Если попытки истекли, то он вызывает метод деаутентификации *AuthManager*, который затем извещает интерфейс, чтобы тот вернул пользователя на окно аутентификации.

Схему авторизации см. на рисунке 18.

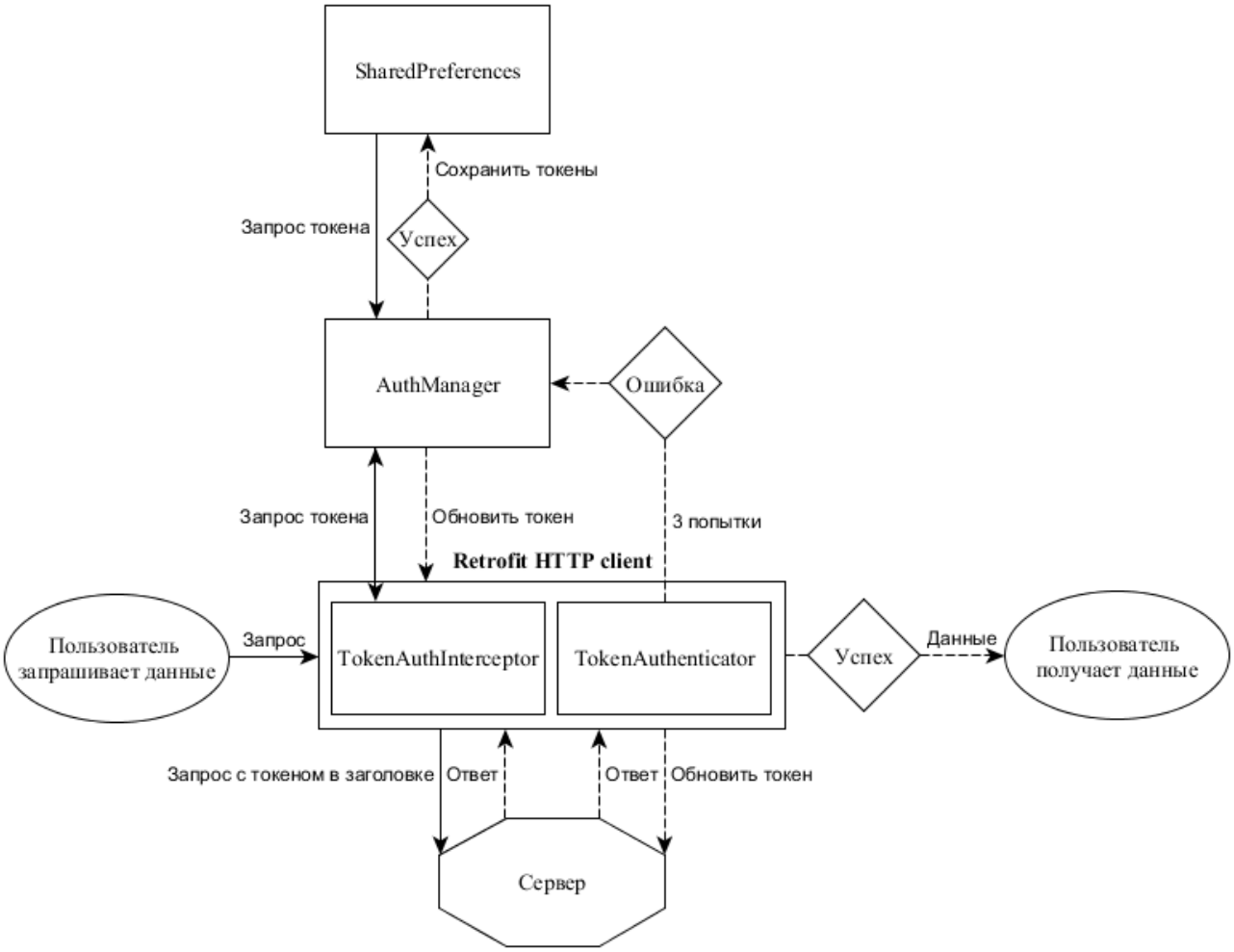

Рисунок 18 - Схема авторизации

3.8 Механизм внедрения зависимостей

Как было упомянуто выше, в проекте используется библиотека Dagger 2 для внедрения зависимостей (dependency injection, DI). Так как в приложении используется стиль single activity application, то для DI целесообразно использовать единственный компонент, состоящий из нескольких модулей:

- ApiModule, в котором расположено все, что касается обращений к серверу и сериализации;

- BuildersModule, в котором явно определены все контроллеры, в которые будут внедряться зависимости;

- AppModule, в котором будут определены различные механизмы, которым важен жизненный цикл activity (например, кэш фото и видео файлов, а также настройки *SharedPreferences*);

- UtilsModule, в котором определяются классы-помощники, такие как *AuthManager*, менеджер прав доступа и любые другие вспомогательные вещи.

Для работы Dagger 2 необходимо осуществить инъекцию компонента в точке входа (entry point) приложения (см. рисунок 19).

```
1.    override fun onCreate() {
2.        super.onCreate()
3.        DaggerAppComponent.builder()
4.                .appModule(AppModule(this))
5.                .contextModule(ContextModule(this))
6.                .build()
7.                .inject(this)
8.    }
```

Рисунок 19 - Инъекция главного компонента DI

Для внедрения зависимостей в контроллеры целесообразно было переопределить контроллеры, предоставляемые библиотекой RxPM, и включить туда инъекцию (см. рисунок 20).

```
1.    abstract class InjectablePmController<T: PresentationModel>
2.    (args: Bundle? = null): PmController<T>(args), HasControllerInjector {
3.
4.        @Inject lateinit var controllerInjector:
5.          DispatchingAndroidInjector<Controller>
6.
7.        override fun controllerInjector() = controllerInjector
8.
9.        override fun onContextAvailable(context: Context) {
10.           super.onContextAvailable(context)
11.           ConductorInjection.inject(this)
12.       }
13.
14.   }
```

Рисунок 20 - Переопределенный контроллер с инъекцией

Затем стало возможным внедрять необходимые зависимости с помощью аннотации *@Inject* в поля контроллера либо в его конструктор.

3.9 Просмотр и кэширование фото и видео

Для просмотра и масштабирования фото была использована библиотека *PhotoView* [13] от Chris Banes, разработчика из компании Google. Она позволяет масштабировать изображение по тапу и восстанавливать масштаб по двойному тапу, а также перемещаться по изображению (см. рисунок 21).

Рисунок 21 - Просмотр вложения

Для загрузки и кэширования изображений в приложении используется библиотека *Picasso* [14] от компании Square. Она позволяет автоматически асинхронно скачивать и кэшировать изображения по одному лишь HTTP-адресу (см рисунок 22).

```
1.    val img = picasso.load(attachment.thumbnailUrl)
2.        .error(R.drawable.ic_error)
3.        .fit()
4.        .centerCrop()
```

Рисунок 22 - Использование Picasso

Для просмотра видео используется встроенный виджет *VideoView*, имеющий все необходимые элементы управления: воспроизведение, пауза, перемотка (см. рисунок 23).

Рисунок 23 - Виджет VideoView

Для потоковой загрузки (загрузка видео осуществляется по частям прямо во время воспроизведения) и кэширования видео используется библиотека *AndroidVideoCache* [15]. Ее использование происходит аналогично библиотеке *Picasso* в одну строчку кода (см. рисунок 24).

```
1.    videoCache.getProxyUrl(attachment.url)
```

Рисунок 24 - Использование AndroidVideoCache

3.10 Интерфейс

Интерфейс приложения задается в соответствующих .xml файлах. Файлы с полноценными интерфейсными окнами называются макетами (layout). Так же элемент интерфейса любого масштаба можно вынести в отдельный файл и тогда он будет называться компонентом. Такие компоненты пригодны к многоразовому использованию и построению из них интерфейса.

Кроме того, существуют уже готовые виджеты со своими свойствами, встроенными событиями и другими параметрами. Некоторые из них являются достаточно сложными компонентами, которые имеют в своем составе другие компоненты.

Для виджетов-списков, которые могут динамически подгружать данные, необходимо использовать классы-адаптеры. В них можно переопределять как данные, которые необходимо отобразить, так и вид, в котором они будут представлены.

В следующем примере (см. рисунок 25) используется список заявок с бесконечной прокруткой, данные для которого динамически подгружаются с сервера.

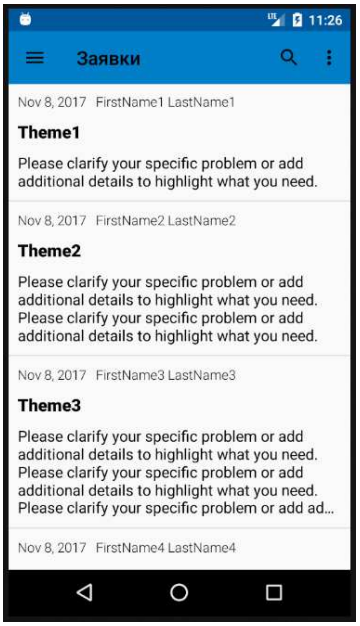

Рисунок 25 - Пример интерфейса с динамическим списком

Список использует переопределенный адаптер с модифицированным *ViewHolder*’ом. *ViewHolder* – это элемент списка, к которому можно привязать определенный интерфейсный компонент (см. рисунок 26).

```
override fun onCreateViewHolder(parent: ViewGroup, viewType: Int): ViewHolder {
    val view = LayoutInflater
        .from(parent.context).inflate(R.layout.row_app_list, parent, false)
    return ViewHolder(view)
}
override fun onBindViewHolder(holder: ViewHolder, position: Int) {
    val app = apps[position]
    holder.itemView.setOnClickListener { appOnClick.onNext(app) }
    holder.id = app.id
    holder.theme.text = app.theme
    holder.createdAt.text = DateFormat.getDateInstance().format(app.createdAt)
    holder.userName.text =
        if (authManager.rights() == UserRights.USER)
            activity.resources.getString(app.destination.resourceId)
        else app.userName

    holder.text.text = app.text
}
```

Рисунок 26 - Пример инициализации ViewHolder'a в адаптере

На данный момент в приложении существует девять контроллеров:

- SignInController, отвечающий за аутентификацию пользователя;

- PasswordRequiredController для ввода пароля пользователя (если он установлен), а также для его сброса;

- ProfileController, в котором пользователь может изменить личные данные;

- OptionsController, содержащий настройки приложения;

- AppListController, отображающий список доступных заявок;

- AddAppController для создания новой заявки;

- AppDetailsUserController, в котором пользователь может посмотреть детали заявки;

- AppDetailsHeadController аналогично содержит детали заявки, но с возможностью руководителю ответить на нее;

- AttachmentViewController отвечает за отображение медиа, как фото, так и видео.

Схему переходов между контроллерами см. на рисунке 27.

Где **USER** – доступ только для обычного пользователя, **HEAD** – доступ только для руководителя.

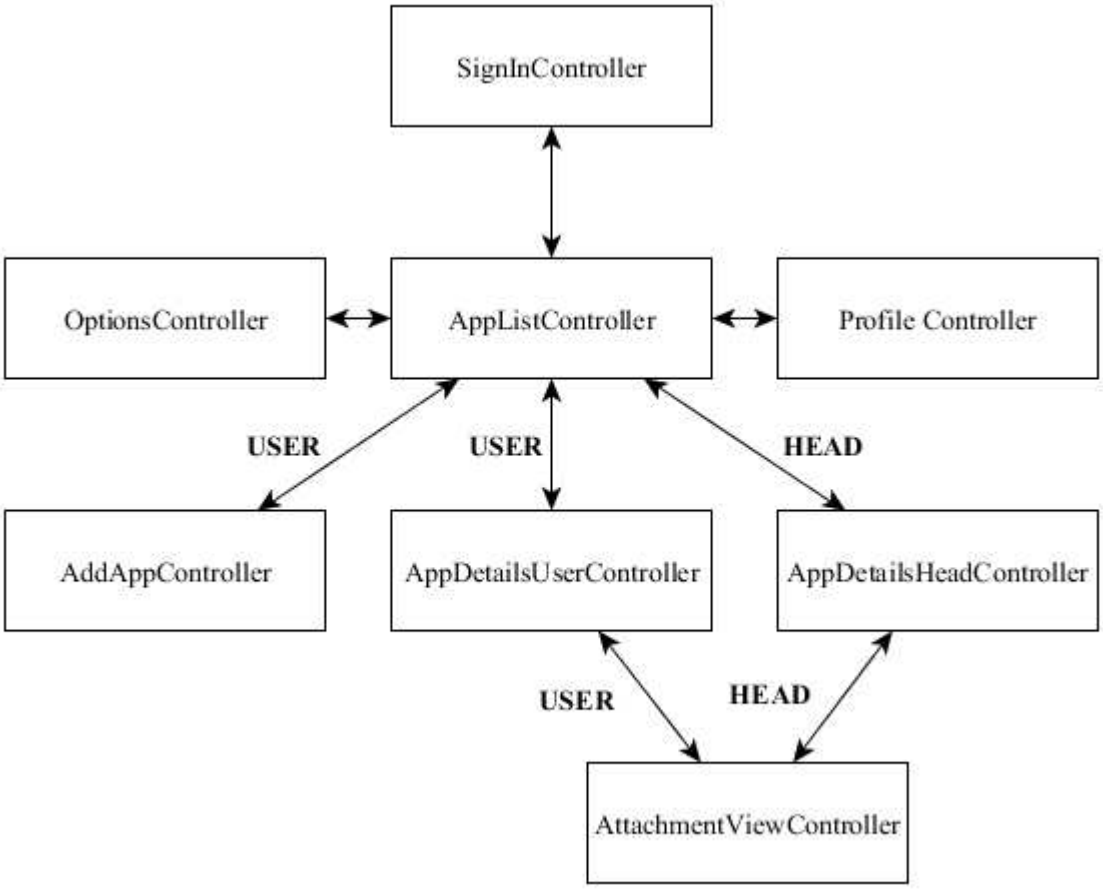

Рисунок 27 - Схема переходов между контроллерами

ЗАКЛЮЧЕНИЕ

В результате выполнения выпускной квалификационной работы были выполнены следующие задачи:

- опрос заказчика;

- анализ предметной области, рассмотрение существующих технологий и их выбор;

- составление технического задания, исходя из требований заказчика и выбранных технологий;

- разработка мобильного приложения на ОС Android;

- тестирование мобильного приложения и его взаимодействие с сервером;

- предоставление готового продукта заказчику.

На данном этапе осуществляется внедрение веб-сервиса на предприятие. Благодаря мобильному приложению сотрудники предприятия получат возможность удобной и мобильной подачи жалоб о нарушении техники безопасности. Ожидается, что с помощью данных, предоставляемых разработанным веб-сервисом, руководство компании сможет оперативно устранять имеющиеся угрозы и количество несчастных случаев существенно сократится.

Таким образом, задачи данной выпускной квалификационной работы можно считать выполненными, а цель достигнутой.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1 IntelliJ IDEA [Электронный ресурс] : статья на Википедии – Режим доступа: https://ru.wikipedia.org/w/index.php?title=IntelliJ_IDEA&oldid=92987896.

2 Blackheath, S., Functional Reactive Programming. / S. Blackheath, A. Jones. – Manning Publications Co, 2016. – 337 с.

3 The Reactive Manifesto [Электронный ресурс] : манифест реактивного подхода – Режим доступа: https://www.reactivemanifesto.org/.

4 Библиотека RxAndroid [Электронный ресурс] : страница на github.com – Режим доступа: https://github.com/ReactiveX/RxAndroid.

5 Библиотека ReactiveX [Электронный ресурс] : страница аккаунта ReactiveX на github.com – Режим доступа: https://github.com/ReactiveX.

6 Библиотека RxPM [Электронный ресурс] : страница на github.com – Режим доступа: https://github.com/dmdevgo/RxPM.

7 Presentation Model [Электронный ресурс] : описание паттерна Presentation Model Мартином Фаулером – Режим доступа: https://martinfowler.com/eaaDev/PresentationModel.html.

8 Библиотека Conductor [Электронный ресурс] : страница на github.com – Режим доступа: https://github.com/bluelinelabs/Conductor.

9 Библиотека Dagger 2 [Электронный ресурс] : страница на github.com – Режим доступа: https://github.com/google/dagger.

10 Библиотека RxBinding [Электронный ресурс] : страница на github.com – Режим доступа: https://github.com/JakeWharton/RxBinding.

11 Разбираемся с Conductor [Электронный ресурс] : статья на habr.com – Режим доступа: https://habr.com/post/329532/.

12 Библиотека Retrofit [Электронный ресурс] : страница на github.com – Режим доступа: https://github.com/square/retrofit.

13 Библиотека PhotoView [Электронный ресурс] : страница на github.com – Режим доступа: https://github.com/chrisbanes/PhotoView.

14 Библиотека Picasso [Электронный ресурс] : страница на github.com – Режим доступа: https://github.com/square/picasso.

15 Библиотека AndroidVideoCache [Электронный ресурс] : страница на github.com – Режим доступа: https://github.com/danikula/AndroidVideoCache.

16 Различия между MVVM и остальными MV*-паттернами [Электронный ресурс] : статья из блога компании MobileUp на habr.com – Режим доступа: https://habr.com/company/mobileup/blog/313538/.

17 RxPM — реактивная реализация паттерна Presentation Model [Электронный ресурс] : статья из блога компании MobileUp на habr.com – Режим доступа: https://habr.com/company/mobileup/blog/342850/.

Федеральное государственное автономное
образовательное учреждение
высшего образования
«СИБИРСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

Институт Космических и Информационных Технологий
институт
Информационные системы
кафедра

УТВЕРЖДАЮ
Заведующий кафедрой
_____ Л.С. Троценко
подпись инициалы, фамилия
«13» июня 2018 г.

БАКАЛАВРСКАЯ РАБОТА

09.03.02 Информационные системы и технологии

Разработка клиентской части веб-сервиса для регистрации случаев нарушения техники безопасности в транспортном филиале компании

Руководитель _____ 13.06.18 С.А. Виденин
подпись, дата инициалы, фамилия

Выпускник _____ 13.06.18 Н.А. Тулин
подпись, дата инициалы, фамилия

Нормоконтролер _____ 13.06.18 Ю.В.Шмагрис
подпись, дата инициалы, фамилия

Красноярск 2018