МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное бюджетное образовательное учреждение

высшего образования

**«КУБАНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»**

**(ФГБОУ ВО «КубГУ»)**

**Факультет компьютерных технологий и прикладной математики**

**Кафедра вычислительных технологий**

**КУРСОВАЯ РАБОТА**

**РАЗРАБОТКА МОБИЛЬНОГО ПРИЛОЖЕНИЯ ДЛЯ ОТОБРАЖЕНИЯ РАСПИСАНИЯ НА ПЛАТФОРМЕ ANDROID**

Работу выполнил \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_Н. Е. Касаткин

(подпись)

Направление 02.03.02 — «Фундаментальная информатика и информационные технологии»

Направленность (профиль) «Вычислительные технологии»

Научный руководитель,

канд. техн. наук, доц.\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_Т.А. Приходько

(подпись)

Нормоконтролер,

канд. техн. наук, доц.\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(подпись)

**СОДЕРЖАНИЕ**

[Введение 3](#_Toc10133791)

[1 Постановка задачи 5](#_Toc10133792)

[1.1 Рассмотрение возможных решений проблемы 5](#_Toc10133793)

[1.2 Оценка целесообразности приведённого решения 6](#_Toc10133794)

[2 Теоретические основы разработки 7](#_Toc10133795)

[2.1 Выбор языка разработки 7](#_Toc10133796)

[2.2 Анализ охвата аудитории и deployment target 9](#_Toc10133797)

[2.3 Разработка автономных приложений. БД Realm. 10](#_Toc10133798)

[3 Разработка приложения 14](#_Toc10133799)

[3.1 Основные сведения о приложении 14](#_Toc10133800)

[3.2 Особенности реализации 14](#_Toc10133801)

[3.3 Проблемы при разработке 19](#_Toc10133802)

[Заключение 20](#_Toc10133803)

[Список использованных источников 21](#_Toc10133804)

[Приложение А. Файл mainActivity.kt 22](#_Toc10133805)

[Приложение Б. Файл startActivity.kt 26](#_Toc10133806)

# **Введение**

На сегодняшний день практически каждый человек имеет в своем распоряжении смартфон. Студенты высших учебных заведений также являются активными пользователями мобильных устройств, которые помогают быстро получаться доступ к актуальной информации. Поэтому обусловлено использование мобильного приложения как вспомогательного элемента в образовательной системе.

Один из возможных аспектов студенческой жизни, куда можно внедрить мобильное приложение, – это расписание занятий. Классический способ информирования студентов – это расписание, расположенное на стенде факультета. Недостатком такого способа является невозможность узнать расписание, не будучи рядом с этим стендом.

Идея разработки мобильного приложения происходит из предположения, что студентам будет удобно иметь локальную копию расписания на своем смартфоне, доступ к которой можно получить практически мгновенно, не беспокоясь при этом об актуальности информации.

В ходе данной работы будут изложены аспекты разработки offline-first мобильного приложения, рассмотрены способы адаптации приложения под разные устройства, а также будет разработано приложение для операционной системы Android для отображения расписания занятий.

Таким образом, в рамках курсовой работы ~~будут выполнены~~ планируется:

* рассмотрение альтернатив решения проблемы просмотра расписания;
* анализ целесообразности разработки мобильного приложения;
* сравнение языков программирования для разработки мобильных приложений для операционной системы Android;
* анализ распределения аудитории пользователей относительно разных версий операционной системы Android;
* рассмотрение принципов offline-first разработки;
* описание преимуществ NoSQL базы данных Realm для мобильных приложений;
* описание взаимодействия приложения с сервером;
* реализация мобильного приложения для отображения расписания на выбранном языке программирования.

# **Постановка задачи**

## **Рассмотрение возможных решений проблемы**

Просмотр своего расписания – это одно из действий, которые студент выполняет чаще всего. Зачастую расписание для разных специальностей одного факультета находится на стенде или на информационной доске этого факультета. При этом занятия могут проходить в разных корпусах университета, так что постоянно возвращаться к расписанию и смотреть, какие занятия и в каких аудиториях они будут проходить, представляется совершенно неэффективной тратой времени. Иногда на это может уходить большая часть времени между парами, которое можно было бы потратить на подготовку, общение и другие более полезные вещи.

Одним из решений этого является создание фотографии всего расписания, с целью последующего его просмотра на экране мобильного телефона. Однако этот способ также имеет два существенных недостатка:

1. Студенты в процессе обучения неизбежно делают много фотографий. К этому всему добавляются фотографии, не относящиеся к учебе. В следствие этих фактов нужная фотография расписания теряется в общей массе, что вновь приводит к проблеме траты значительного времени на поиск информации о расписании.
2. Фотография, сделанная единожды, позволяет сохранить расписание, актуальное на момент ее создания. Однако все дальнейшие изменения порядка занятий или аудиторий никак не будут отражены на ней. Это приводит проблеме неактуальности информации. Для ее актуализации необходимо снова возвращаться к информационному стенду и делать новый снимок.

## **Оценка целесообразности приведённого решения**

Современные смарт-гаджеты дают возможность организовать и оптимизировать процесс получения информации вне зависимости от места и времени. Причем использование мобильных устройств молодым населением становится все более повсеместным. Большинство современных студентов технически и психологически готовы к использованию мобильных технологий в учебном процессе, и необходимо рассматривать новые возможности для более эффективного использования потенциала мобильных устройств и технологий. Разработка мобильного приложения покроет почти всю целевую аудиторию. Также подобное приложение позволит достичь постоянной актуальности информации в условиях наличия соединения с сетью Интернет за счет организации взаимодействия смартфона с сервером, где будет храниться расписание, а Offline-first подход при разработке позволит использовать приложение даже при отсутствии стабильного соединения.

Данное приложение решает не только проблему расписания для студентов, а также и для преподавателей, для которых поиск своих пар (после составления или существенного изменения расписания) в общем расписании для всех четырёх курсов занимает немалое количество времени. Отображение расписания конкретного преподавателя также поможет и студентам, которым нужно найти этого преподавателя, например, для сдачи зачёта во время зачётной недели.

Еще одно преимущество – это возможность масштабирования решаемой задачи, то есть в последствии можно добавлять функционал, отличный от отображения актуального расписания.

# **Теоретические основы разработки**

## **Выбор языка разработки**

Основными языками программирования для разработки приложений под операционную систему Android являются Kotlin и Java. Приведем краткую сравнительную характеристику.

Kotlin – это новейший статически типизированный язык программирования с открытым исходным кодом. Он может эффективно запускаться на виртуальной машине Java (JVM). Kotlin разработан JetBrains и официально поддерживается Google [2].

**Достоинства Kotlin:**

1. Улучшенная производительность. Последовательный и интуитивный синтаксис Kotlin обеспечивает повышение производительности команд разработчиков. Для написания и деплоймента программы требуется меньше времени и меньше строк кода. В результате вы получаете готовое приложение быстрее;
2. Полная совместимость с Java, то есть методы Java можно вызывать из Kotlin;
3. Обратная совместимость со всеми предыдущими версиями;
4. Поддержка Android Studio. Android Studio предоставляет расширенную поддержку Kotlin и инструменты адаптации. Разработчики могут работать одновременно на Kotlin и на Java.

**Недостатки Kotlin:**

1. Меньшая поддержка сообщества. У Kotlin маленькое сообщество разработчиков, в связи с чем количество ресурсов для изучения этого языка ограничено. Может встретиться много вопросов без ответов;
2. Нехватка разработчиков. Очень ограниченное количество людей занимаются разработкой на Kotlin. Недостаток талантов в разработке Android-приложений касается Kotlin в большей степени, чем Java;
3. Недостаток документации на русском языке.

Java считается одним из лучших языков для разработки приложений. Одной из основных особенностей этого языка является объектно-ориентированность.[2]

**Достоинства Java:**

1. Портируемость. С помощью виртуальной машины Java программы на этом языке могут запускаться практически в любой системе. В результате Java получает лидерство среди Android-приложений;
2. Большое сообщество. Благодаря обширным сообществам на GitHub и Stack Overflow разработчики могут получить помощь практически по любой проблеме;
3. Кроссплатформенность. Применение этого языка не ограничивается лишь разработкой Android-приложений. Java прекрасно подходит и для разработки кроссплатформенных приложений.
4. Ресурсы. Поскольку Android разработан на Java, этот язык имеет уже готовые библиотеки и SDK для облегчения процесса разработки.

**Недостатки Java:**

1. Скорость. Java требует больше памяти и, по сравнению с Kotlin работает намного медленнее;
2. Сложность кода. В следствии того что программы на Java получаются длинными, код этих программ требует больше времени для написания, в нём больше ошибок.

Основным требованием к приложению является скорость работы. Стоит задача максимально снизить время, затрачиваемое на получение информации. Часть функций языка Kotlin выполняется быстрее по сравнению с другими языками программирования. Использование языка Kotlin позволит выиграть в скорости как при запуске приложения, так и при дальнейшей работе с базой данных. Также этот язык отличается более высокой скоростью и простотой разработки. Ввиду явного перевеса в сторону языка Kotlin в качестве языка разработки будет выбран именно он.

## **Анализ охвата аудитории и deployment target**

При разработке мобильного приложения также стоит задача обеспечить как можно больший охват целевой аудитории. На этот показатель будет диапазон устройств, на который будет доступно приложение, а также минимальная версия операционной системы, необходимая для работы приложения, – так называемый deployment target. Необходимо соблюсти грань между охватом и доступным функционалом, так как старые версии системы не поддерживают многие прогрессивные практики в программировании.

Каждый месяц ведется статистика по долям установленных версий Android на устройствах по всему миру. Как видно из рисунка 1 по состоянию на май 2019 года самая популярная версия Android – это 8.X (Oreo)[1].

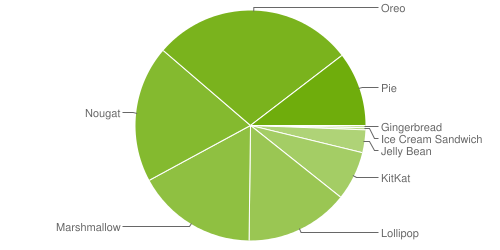


Рисунок 1 – Диаграмма доли установленных версий Android относительно общего числа устройств [1]

Выбрав в качестве deployment target версию Android KitKat мы потеряем не более четырёх процентов аудитории, однако это позволит использовать современные средства разработки и оптимизации приложений.

## **Разработка автономных приложений. БД Realm.**

Хотя современные технологии позволяют создавать прогрессивные веб приложения (способные работать в оффлайн режиме, использовать push-уведомления), нативные приложения все равно зарекомендовали себя с лучшей стороны, когда речь идет о разработке автономных приложений. Оффлайн-режим — очень важная вещь, и стоит думать о нем с самого начала работы над приложением. Offline-first подход к разработке мобильных приложений подразумевает под собой некоторый свод правил и техник для успешной разработки автономного мобильного приложения.

Прежде всего изначально приложение проектируется для работы в оффлайн режиме, чтобы пользователь мог продолжать работу в отсутствие интернета и прозрачно синхронизироваться, когда связь появляется. Зачастую большую часть работы на себя берет сервер, однако в данном случае сервер максимально отвязывается от приложения. На стороне сервера остаётся только лёгкий JSON API для доступа к базе данных.

Также для обеспечения независимости приложения от серверной части используют мобильные базы данных, способные сохранять некоторые данные на долгий период времени, и которые не отчищаются при перезагрузке приложения или всего устройства (в отличие от оперативной памяти).

В разрабатываемом приложении будет реализован такой принцип работы. Главное требование – прозрачность обновления информации для пользователя. Как только появляется интернет, клиентская часть синхронизируется при необходимости с серверной, а пользователь продолжает работу, как ни в чем не бывало.

Для этого будет реализована работа с JSON-файлами. JSON (JavaScript Object Notation) - это легкий формат, который используется для обмена данными. Он гораздо короче в записи, чем XML, что положительно скажется на скорости загрузки данных с сервера, а также с ним легко работать и легко извлекать информацию. На сегодняшний день объекты json являются стандартом де-факто в обмене информацией с сервером.

Был выбран следующий сценарий работы:

1. при загрузке приложения выполняется асинхронный запрос к серверу с целью получить json-файл с более актуальной информацией;
2. для отображения выбираются сначала данные, хранящиеся в локальной базе данных на устройстве;
3. если запрос был обработан сервером и был получен json-файл, то новая информация заносится в локальную базу данных;
4. обновляется отображаемая в приложении информация.

В качестве локальной базы данных будет выступать NoSQL БД Realm. Realm — кросс-платформенная мобильная база данных для Android (доступная в Kotlin и Java) и iOS. Realm была создана, чтобы стать лучше и быстрее, чем SQLite и Core Data. Она не только лучше и быстрее, но и простая в использовании.

Основные особенности и плюсы Realm:

1. Работает в режиме Zero copy – поддерживает операции, в ходе которых процессор не выполняет задачу копирования данных из одной области памяти в другую;
2. Основана на ACID – требованиях к транзакционной системе, обеспечивающих наиболее надёжную и предсказуемую работу;
3. Является NoSQL базой данных.

Требования ACID заключаются в следующем:

1. Atomicity (Атомарность) – гарантирует, что никакая транзакция не будет зафиксирована в системе частично. Будут либо выполнены все её подоперации, либо не выполнено ни одной. Поскольку на практике невозможно одновременно и атомарно выполнить всю последовательность операций внутри транзакции, вводится понятие «отката» (rollback): если транзакцию не удаётся полностью завершить, результаты всех её до сих пор произведённых действий будут отменены и система вернётся во «внешне исходное» состояние;
2. Consistency (Согласованность) – согласованность данных друг с другом, целостность данных, а также внутренняя непротиворечивость;
3. Isolation (Изолированность) – во время выполнения транзакции параллельные транзакции не должны оказывать влияния на её результат;
4. Durability (Долговечность) – если пользователь получил подтверждение от системы, что транзакция выполнена, он может быть уверен, что сделанные им изменения не будут отменены из-за какого-либо сбоя.

Преимущества Realm как NoSQL базы данных заключаются в том, что скорость обработки данных гораздо выше, чем в любых реляционных аналогах, а зачастую и NoSQL аналогах [4].

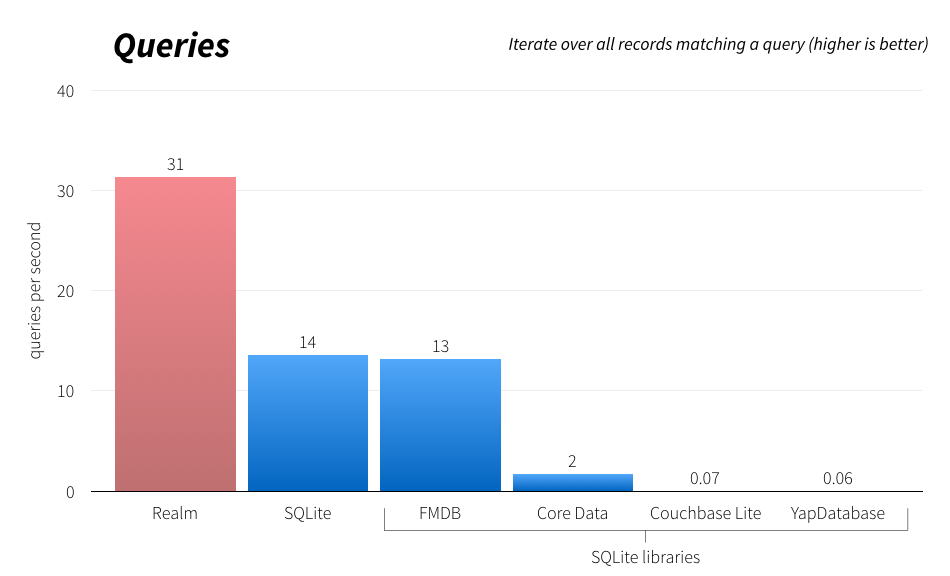


Рисунок 2 – Сравнение скорости чтения Realm с другими мобильными БД

В NoSQL нет ограничений на типы хранимых данных, а при необходимости можно добавлять новые типы данных, что позволит масштабировать и изменять сложность системы в будущем, например, добавлением новых полей к объектам json. Для этого не придется заново создавать некие таблицы или пересоздавать старые. Еще одним достоинством Realm является чрезвычайно простой механизм запросов.

Все объекты, полученные из Realm, являются, так называемыми «живыми объектами», своеобразными прокси к базе данных. Объекты не копируются из базы в промежуточные переменные. Вместо этого в переменную записывается аналог ссылки на объект. Такое взаимодействие позволяет, например, избежать дополнительных запросов-выборок к БД в случае изменения данных – переменные, содержащие ссылки на изменившиеся объекты автоматически будут иметь в качестве своих полей новые данные. А так как запросы к серверу будут выполняться асинхронно, то обновление данных в БД не потребует удаления старых объектов и записи, вместо этого может быть выполнено редактирование «живых объектов».

# **Разработка приложения**

## **Основные сведения о приложении**

Основу приложения составляют две активити:

1. startActivity: отвечает за логику приложения, исполняемую во время загрузки. Сюда входит асинхронная отправка запроса на сервер, обработка ответа сервера и сохранение изменений в локальной базе данных; Также содержит в себе два пикера (элемент UI, позволяющий выбирать из альтернатив), с помощью которых выбирается группа или преподаватель, чье расписание нужно отображать в приложении. При отсутствии доступа к интернету данная активити оповещает пользователя о том, что не удалось получить/обновить данные и передаёт управление mainActivity;
2. mainActivity: главный элемент программы, содержит tableLayout для отображения расписания (название занятия, ФИО преподавателя, аудитория проведения и время проведения), а также recycleView для удобной навигации по дням недели.

## **Особенности реализации**

При запуске приложения пользователя встречает загрузочный экран, представленный startActivity. Его визуальная составляющая делится на две части, которые отображаются в разные моменты времени:

1. простой фон с логотипом приложения (рисунок 3) отображается во время получения данных с сервера;
2. два пикера, которые помогают пользователю выбрать необходимый параметр для отображения расписания (рисунок 4) отображаются сразу после получения актуального JSON-файла с сервера.



Рисунок 3 – startActivity во время получения данных

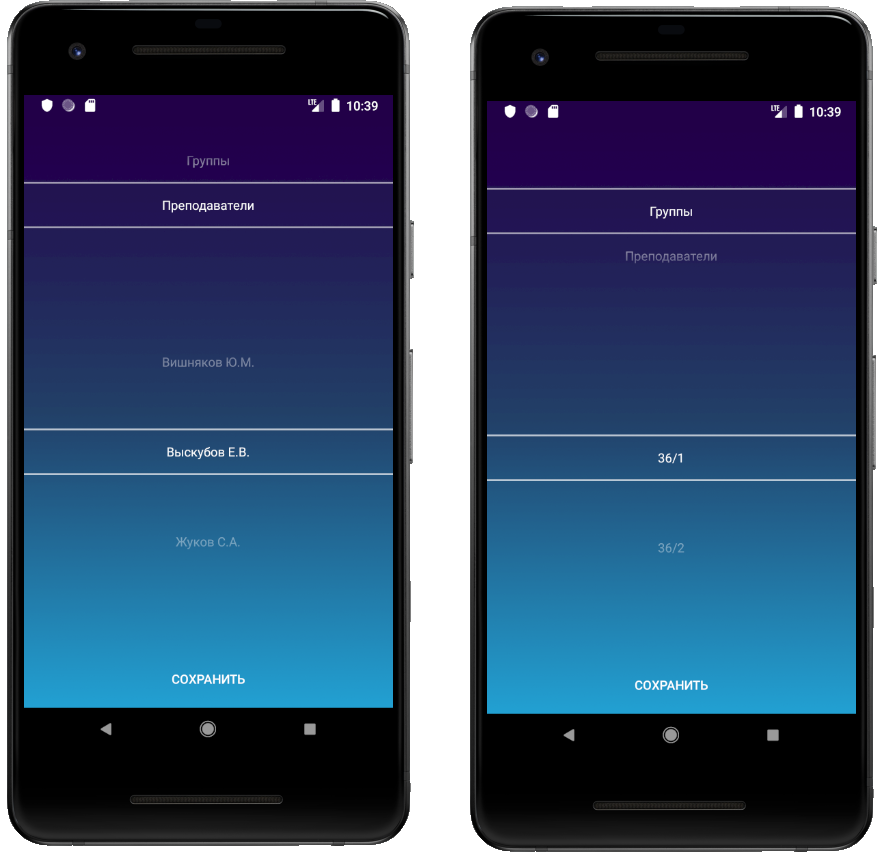


Рисунок 4 – startActivity после полученных данных

Программная составляющая состоит, во-первых, из проверки установленных пользовательских настроек. Все настройки пользователя хранятся в базе данных. И при запуске приложения записываются в глобальные переменные, чтобы сократить количество запросов к базе данных. Данные из этих переменных удаляются после перезапуска приложения. В данном приложении пользовательские настройки – это выбранная группа или преподаватель, чье расписание необходимо отображать, а также версия расписания (полученная от сервера), необходимая для проверки актуальности сохраненных данных. Во-вторых, здесь же реализуется отправка запроса на сервер. Делается это с помощью библиотеки OkHTTP3, которая позволяет работать с HTTP запросами, а также обрабатывать ошибки. В случае успешного получения ответа от сервера, обновляется информация в локальной базе данных.

Если в настройках не установлена текущая группа, то приложение автоматически перенаправляет пользователя на startActivity (рисунок 3 и рисунок 4), однако если у пользователя не выбрана текущая группа (то есть это первый запуск приложения) и отсутствует интернет, то никакая информация не будет отображаться в приложении, так как она не была получена. При первом же подключении к интернету актуальная информация будет занесена в локальную базу данных.

В strartActivity происходит выбор группы или преподавателя. Их список получается из json-файла, который возвращает сервер. После выбора данные из json сохраняются в локальную базу данных, чтобы потом можно было работать с приложением в условиях отсутствия подключения к сети Интернет. Также в пользовательские настройки записывается сделанный выбор, чтобы впоследствии пользователю не пришлось выбирать группу.

Каждая запись в базе данных содержит данные об одной паре. Структура PairClass, которая содержит эти данные, представлена на рисунке 5.

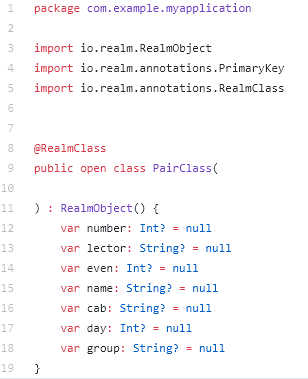


Рисунок 5 – структура PairClass

Каждое занятие характеризуется семью полями: названием, преподавателем, аудиторией проведения, номером (какое по счету в день), четностью/нечетностью проведения, днём и группой, у которой это занятие проводится.

После выбора пользователь попадает на главный экран – mainActivity (рисунок 6). Здесь отображается расписание выбранной пользователем группы (или преподавателя). Для этого программа получает текущую дату, по ней определяется чётность или нечётность текущей недели (информация об этом хранится в глобальной переменной), строится список дней, который состоит из четырнадцати элементов, которые содержат числа текущей недели и следующей. Все эти данные передаются в контроллер recycleView для создания удобного горизонтального списка (календаря), при клике на элемент которого обновляется расписание в соответствии с выбранным днём.



Рисунок 6 – mainActivity

Календарь рассчитан на две недели для того, чтобы можно было посмотреть расписание как в неделю числителя, так и в неделю знаменателя. Текущий день дополнительно помечается красным цветом, а отображаемый день имеет затемнённый фон. Каждый раз, когда приложение открывается, выбранным по умолчанию будет текущий день, а четность недели будет автоматически переключаться при наступлении новой недели. Также здесь реализована вся логика обращения к базе данных, заполнение таблицы этими данными, жесты управления и прочие элементы пользовательского интерфейса. Дополнительно имеется кнопка для перехода в startActivity для смены выбора группы.

## **Проблемы при разработке**

Основной проблемой при разработке данного приложения была реализация контроллера, позволяющего переключаться между днями в календаре.

Проблема заключалась в элементе recycleView. Дело в том, что данный виджет не имеет встроенного метода onClickItem (в отличии от listView, который не был использован из-за того, что его нельзя расположить горизонтально). В документации виджета не указано, где необходимо устанавливать слушатели. Из-за чего было принято решение, данный метод ~~нужно было~~ реализовать прямо в адаптере, однако слушатель должен быть установлен на каждый элемент календаря, что означает, что его установка невозможна в методе onBindViewHolder, ~~из-за того что~~ поскольку данный метод вызывается неограниченное количество раз (количество зависит от действия пользователя), что неоптимально. Поэтому данный слушатель устанавливается в классе ViewHolder, экземпляр которого создается в методе onCreateViewHolder.

Второй главной проблемой данного контроллера является анимация (покраска его содержимого в разные цвета в зависимости от выбранной недели и дня). Проблема заключалась в том, что в оперативной памяти устройства хранилось не более 10 дней (из 14 возможных), а значит при клике на один из крайних дней в календаре (ПН, ВТ или СР) фон которых затемнялся для удобства пользователя, затемнялся также фон и ПТ, СБ и ВС второй недели соответственно. Это происходило из-за того, что дни, которые не хранились в ОЗУ оказывались в пуле, который хранился в постоянной памяти устройства и их свойства менялись вместе с нулевым, первым и вторым элементами массива из пула.

Решением данных проблем стало добавление функции, вызываемой при клике на элемент календаря, которая полностью обновляет все элементы recycleView.

# **Заключение**

В ходе курсовой работы была поставлена задача создания мобильного приложения на операционной системе Android для информирования студентов. Были приведены аргументы об актуальности данной разработки, проведен анализ охвата аудитории, рассмотрены средства для разработки, в том числе языки программирования и локальная база данных Realm.

В рамках этой работы был разработан дизайн приложения, а также реализовано само приложение, в котором можно смотреть расписание групп и преподавателей. В приложении настроена работа с сервером для прозрачной синхронизации данных, реализована автономность работы с помощью локальной базы данных и контроль версий расписания.

В дальнейшем планируется реализовать полную поддержку устройств с разными версиями ОС и разными разрешениями экранов, систему оповещения пользователей об изменениях в расписании и напоминания о предстоящих занятиях на следующий день. Рассматривается перспектива внедрения данного приложения в учебный процесс.

Также планируется создать новый дизайн для приложения, с целью соответствия его корпоративному стилю факультета и университета.

# **Список использованных источников**

1. Distribution dashboard [Электронный ресурс]. – 2013. – URL: https://developer.android.com/about/dashboards (дата обращения: 30.05.2019г).
2. Java и Kotlin: что будет лучшим выбором в 2019 году? [Электронный ресурс]. – 2014. – URL: https://techrocks.ru/2019/01/28/java-vs-kotlin-best-language-for-android-in-2019/ (дата обращения: 30.05.2019г).
3. Kotlin Programming Language [Электронный ресурс]. – 2019. – URL: https://kotlinlang.org/ (дата обращения: 30.05.2019г).
4. Introducing Realm [Электронный ресурс]. – 2014. – URL: https://realm.io/blog/introducing-realm/ (дата обращения: 25.06.2019г).
5. Tomasz Nurkiewicz. Reactive Programming with RxJava: Creating Asynchronous, Event-Based Applications / Tomasz Nurkiewicz, Ben Christensen. – O’Reilly Media. – 2016 (1 edition). – 372с.

# **Приложение А**

Файл mainActivity.kt

**package com.example.myapplication**

**lateinit var currentDay: String**

**public var currentWeek = true**

**public var currentSelectedWeek = true**

**lateinit var currentSelectedDay: String**

**lateinit var mAdapter: MainAdapter**

**private lateinit var realm: Realm**

**class MainActivity : AppCompatActivity() {**

**override fun onCreate(savedInstanceState: Bundle?) {**

**super.onCreate(savedInstanceState)**

**setContentView(R.layout.activity\_main)**

**val res: Resources = resources**

**val days = res.getStringArray(R.array.daysOfWeek)**

**val date: ArrayList<String> = ArrayList(14)**

**val cal = Calendar.getInstance()**

**cal.firstDayOfWeek = GregorianCalendar.MONDAY**

**cal.set(Calendar.HOUR\_OF\_DAY, 0)**

**cal.clear(Calendar.MINUTE)**

**cal.clear(Calendar.SECOND)**

**cal.clear(Calendar.MILLISECOND)**

**currentDay = cal.get(Calendar.DAY\_OF\_MONTH).toString()**

**currentSelectedDay = currentDay**

**cal.set(Calendar.DAY\_OF\_WEEK, cal.firstDayOfWeek)**

**currentWeek = cal.get(Calendar.WEEK\_OF\_YEAR) % 2 == 0**

**currentSelectedWeek = cal.get(Calendar.WEEK\_OF\_YEAR) % 2 == 0**

**date.add(cal.get(Calendar.DAY\_OF\_MONTH).toString())**

**for (i in 1..13) {**

**cal.add(Calendar.DATE, 1)**

**date.add(cal.get(Calendar.DAY\_OF\_MONTH).toString())**

**}**

**recyclerView.layoutManager = LinearLayoutManager(this, OrientationHelper.HORIZONTAL, false)**

**mAdapter = MainAdapter(days, date)**

**recyclerView.adapter = mAdapter**

**realm = Realm.getDefaultInstance()**

**tableLayout.setOnTouchListener(object : OnSwipeTouchListener(this) {**

**override fun onSwipeLeft() {**

**val index = date.indexOf(currentSelectedDay)**

**if (index<13){**

**currentSelectedDay = date[index+1]**

**reloadData()**

**if (index+1<7){**

**recyclerView.layoutManager!!.scrollToPosition(0)**

**} else {**

**recyclerView.layoutManager!!.scrollToPosition(13)**

**}**

**}**

**}**

**override fun onSwipeRight() {**

**val index = date.indexOf(currentSelectedDay)**

**if (index>0){**

**currentSelectedDay = date[index-1]**

**reloadData()**

**if (index-1<7){**

**recyclerView.layoutManager!!.scrollToPosition(0)**

**} else {**

**recyclerView.layoutManager!!.scrollToPosition(13)**

**}**

**}**

**}**

**})**

**}**

**fun reloadData() {**

**mAdapter.notifyDataSetChanged()**

**}**

**fun updateCurrentDay(pos: Int) {**

**val res: Resources = resources**

**val days = res.getStringArray(R.array.daysOfWeek)**

**val full\_days = res.getStringArray(R.array.daysOfWeekFull)**

**val now = if (pos<7) {**

**pos**

**} else {**

**pos%7**

**}**

**val currentDaysPairs = realm.where(PairClass::class.java).equalTo("group", getParametr()).equalTo("day", now).findAll()**

**clearFields()**

**currentSelectedWeek = if (pos < 7) {**

**currentWeek**

**} else {**

**!currentWeek**

**}**

**for (pair in currentDaysPairs) {**

**insertPairIntoTable(pair)**

**}**

**for (iterator in days.indices) {**

**if (iterator == pos%7) {**

**currentDayText.text = full\_days[iterator]**

**break**

**}**

**}**

**}**

**fun getParametr(): String{**

**val test = realm.where(selectedGroup::class.java).findAll()**

**if (test.isNotEmpty()) groupName.text = test[0]?.numberOfgroup**

**return if (test.isNotEmpty()){**

**when(test[0]!!.numberOfgroup){**

**"36/1" -> "36\_1"**

**"36/2" -> "36\_2"**

**else -> "36\_1"**

**}**

**} else {**

**"36\_1"**

**}**

**}**

**fun insertPairIntoTable(pair: PairClass) {**

**when (pair.number) {**

**0 -> {**

**if ((pair.even == 2) || ((pair.even == 0) && (currentSelectedWeek)) || ((pair.even == 1) && (!currentSelectedWeek)) ){**

**name0.text = pair.name**

**aud0.text = pair.cab**

**prepod0.text = pair.lector**

**}**

**}**

**1 -> {**

**if ((pair.even == 2) || ((pair.even == 0) && (currentSelectedWeek)) || ((pair.even == 1) && (!currentSelectedWeek)) ) {**

**name1.text = pair.name**

**aud1.text = pair.cab**

**prepod1.text = pair.lector**

**}**

**}**

**2 -> {**

**if ((pair.even == 2) || ((pair.even == 0) && (currentSelectedWeek)) || ((pair.even == 1) && (!currentSelectedWeek)) ) {**

**name2.text = pair.name**

**aud2.text = pair.cab**

**prepod2.text = pair.lector**

**}**

**}**

**3 -> {**

**if ((pair.even == 2) || ((pair.even == 0) && (currentSelectedWeek)) || ((pair.even == 1) && (!currentSelectedWeek)) ) {**

**name3.text = pair.name**

**aud3.text = pair.cab**

**prepod3.text = pair.lector**

**}**

**}**

**4 -> {**

**if ((pair.even == 2) || ((pair.even == 0) && (currentSelectedWeek)) || ((pair.even == 1) && (!currentSelectedWeek)) ) {**

**name4.text = pair.name**

**aud4.text = pair.cab**

**prepod4.text = pair.lector**

**}**

**}**

**5 -> {**

**if ((pair.even == 2) || ((pair.even == 0) && (currentSelectedWeek)) || ((pair.even == 1) && (!currentSelectedWeek)) ) {**

**name5.text = pair.name**

**aud5.text = pair.cab**

**prepod5.text = pair.lector**

**}**

**}**

**6 -> {**

**if ((pair.even == 2) || ((pair.even == 0) && (currentSelectedWeek)) || ((pair.even == 1) && (!currentSelectedWeek)) ) {**

**name6.text = pair.name**

**aud6.text = pair.cab**

**prepod6.text = pair.lector**

**}**

**}**

**7 -> {**

**if ((pair.even == 2) || ((pair.even == 0) && (currentSelectedWeek)) || ((pair.even == 1) && (!currentSelectedWeek)) ) {**

**name7.text = pair.name**

**aud7.text = pair.cab**

**prepod7.text = pair.lector**

**}**

**}**

**}**

**}**

**fun goChange(view: View){**

**val intent = Intent(this, StartActivity::class.java)**

**intent.putExtra("startedFromMain", true);**

**startActivity(intent)**

**finish()**

**}**

**override fun onDestroy() {**

**super.onDestroy()**

**realm.close()**

**}**

**}**

**class MainAdapter(val daysOfWeek: Array<String>, val date: ArrayList<String>) :**

**RecyclerView.Adapter<MainAdapter.CustomViewHolder>() {**

**override fun getItemCount(): Int {**

**return daysOfWeek.size**

**}**

**override fun onCreateViewHolder(parent: ViewGroup, viewType: Int): CustomViewHolder {**

**val view: View = LayoutInflater.from(parent.context).inflate(R.layout.row\_post, parent, false)**

**return CustomViewHolder(view, date)**

**}**

**override fun onBindViewHolder(holder: CustomViewHolder, position: Int) {**

**holder.nameOfDay.text = daysOfWeek[position]**

**holder.allDate.text = date[position]**

**if (position < 7) {**

**holder.nameOfDay.setBackgroundResource(R.color.test)**

**} else {**

**holder.nameOfDay.setBackgroundResource(R.color.defaultBlue)**

**}**

**if (date[position] == currentDay) {**

**holder.nameOfDay.setBackgroundResource(R.color.defaultRed)**

**}**

**if (date[position] == currentSelectedDay) {**

**(holder.itemView.context as MainActivity).updateCurrentDay(position)**

**holder.allDate.setBackgroundResource(R.color.defaultGrey)**

**} else {**

**holder.allDate.setBackgroundResource(R.color.transparent)**

**}**

**}**

**class CustomViewHolder(v: View, date: ArrayList<String>) : RecyclerView.ViewHolder(v) {**

**val nameOfDay: TextView = itemView.findViewById(R.id.nameDay)**

**val allDate: TextView = itemView.findViewById(R.id.date)**

**init {**

**itemView.setOnClickListener {**

**val position = layoutPosition**

**currentSelectedDay = date[position]**

**(itemView.context as MainActivity).reloadData()**

**}**

**}**

**}**

**}**

# Приложение Б

Файл startActivity.kt

**package com.example.myapplication**

**class StartActivity : AppCompatActivity() {**

**public lateinit var realm: Realm**

**public lateinit var arr: Array<String>**

**public lateinit var arr2: Array<String>**

**public lateinit var groupsArr: Array<String>**

**override fun onCreate(savedInstanceState: Bundle?) {**

**super.onCreate(savedInstanceState)**

**setContentView(R.layout.activity\_start)**

**Realm.init(this)**

**realm = Realm.getDefaultInstance()**

**val startedFromMain = intent.getBooleanExtra("startedFromMain", false)**

**if (!startedFromMain) {**

**fetchJSON()**

**} else {**

**val test = realm.where(version::class.java).findAll()**

**if (test.isNotEmpty()){**

**goOutData()**

**} else {**

**fetchJSON()**

**}**

**}**

**}**

**fun goOutData() {**

**arr = arrayOf("Группы", "Преподаватели")**

**arr2.sort()**

**groupsArr = arrayOf("36/1", "36/2")**

**imageView.visibility = View.INVISIBLE**

**numberPicker.visibility = View.VISIBLE**

**numberPicker2.visibility = View.VISIBLE**

**saveButton.visibility = View.VISIBLE**

**numberPicker.minValue = 0**

**numberPicker.maxValue = arr.size - 1**

**numberPicker.displayedValues = arr**

**var firstItem = numberPicker.getChildAt(0)**

**if (firstItem != null) {**

**firstItem.visibility = View.INVISIBLE**

**}**

**numberPicker.wrapSelectorWheel = false**

**numberPicker.descendantFocusability = NumberPicker.FOCUS\_BLOCK\_DESCENDANTS**

**numberPicker2.minValue = 0**

**numberPicker2.maxValue = groupsArr.size - 1**

**numberPicker2.displayedValues = groupsArr**

**firstItem = numberPicker2.getChildAt(0)**

**if (firstItem != null) {**

**firstItem.visibility = View.INVISIBLE**

**}**

**numberPicker2.wrapSelectorWheel = false**

**numberPicker2.descendantFocusability = NumberPicker.FOCUS\_BLOCK\_DESCENDANTS**

**numberPicker.setOnValueChangedListener { picker, oldVal, newVal ->**

**if (newVal == 0) {**

**numberPicker2.displayedValues = null**

**numberPicker2.minValue = 0**

**numberPicker2.maxValue = groupsArr.size - 1**

**numberPicker2.displayedValues = groupsArr**

**numberPicker2.value = 0**

**} else {**

**numberPicker2.displayedValues = null**

**numberPicker2.minValue = 0**

**numberPicker2.maxValue = arr2.size - 1**

**numberPicker2.displayedValues = arr2**

**numberPicker2.value = 0**

**}**

**}**

**}**

**fun clicked(view: View) {**

**val test = realm.where(selectedGroup::class.java).findAll()**

**if (test.isEmpty()) {**

**realm.executeTransaction { realm ->**

**val v = realm.createObject<selectedGroup>()**

**v.numberOfgroup = numberPicker2.displayedValues[numberPicker2.value]**

**}**

**} else {**

**realm.executeTransaction { realm ->**

**val result = realm.where(selectedGroup::class.java).findAll()**

**result.deleteAllFromRealm()**

**val v = realm.createObject<selectedGroup>()**

**v.numberOfgroup = numberPicker2.displayedValues[numberPicker2.value]**

**}**

**}**

**goNext()**

**}**

**fun goNext() {**

**val intent = Intent(this, MainActivity::class.java)**

**startActivity(intent)**

**finish()**

**}**

**fun fetchJSON() {**

**val url = "https://api.jsonbin.io/b/5cd8487a4c004c0eb49725bb/latest"**

**val request = Request.Builder().url(url).build()**

**val client = OkHttpClient()**

**client.newCall(request).enqueue(object : Callback {**

**override fun onFailure(call: Call, e: IOException) {**

**println("error $e")**

**runOnUiThread {**

**val toast = Toast.makeText(**

**applicationContext,**

**"Не удалось получить/обновить данные",**

**Toast.LENGTH\_LONG**

**)**

**toast.setGravity(Gravity.BOTTOM, 0, 40)**

**toast.show()**

**}**

**goNext()**

**}**

**override fun onResponse(call: Call, response: Response) {**

**val body = response.body()?.string()**

**val gson = GsonBuilder().create()**

**val topic = gson.fromJson(body, mainJSON::class.java)**

**runOnUiThread {**

**val test = realm.where(version::class.java).findAll()**

**if (test.isEmpty()) {**

**realm.executeTransaction { realm ->**

**val v = realm.createObject<version>()**

**v.version = topic.version**

**}**

**goInsertData(topic)**

**goOutData()**

**} else {**

**if (test[0]!!.version!!.toDouble() == topic.version.toDouble()) {**

**goNext()**

**} else {**

**realm.executeTransaction { realm ->**

**realm.deleteAll()**

**}**

**realm.executeTransaction { realm ->**

**val v = realm.createObject<version>()**

**v.version = topic.version**

**}**

**goInsertData(topic)**

**goOutData()**

**}**

**}**

**}**

**}**

**})**

**}**

**fun goInsertData(topic: mainJSON) {**

**val group1 = topic.groups.group1**

**val group2 = topic.groups.group2**

**realm.executeTransaction { realm ->**

**for (item in 0 until group1.size) {**

**val pair = realm.createObject<PairClass>()**

**pair.group = "36\_1"**

**pair.cab = group1[item].cab**

**pair.day = group1[item].day**

**pair.even = group1[item].even**

**pair.lector = group1[item].lector**

**pair.name = group1[item].name**

**pair.number = group1[item].number**

**}**

**for (item in 0 until group2.size) {**

**val pair = realm.createObject<PairClass>()**

**pair.group = "36\_2"**

**pair.cab = group2[item].cab**

**pair.day = group2[item].day**

**pair.even = group2[item].even**

**pair.lector = group2[item].lector**

**pair.name = group2[item].name**

**pair.number = group2[item].number**

**}**

**}**

**}**

**override fun onDestroy() {**

**super.onDestroy()**

**realm.close()**

**}**

**}**