Министерство образования Республики Беларусь

Учреждение образования

«Белорусский государственный университет информатики и радиоэлектроники»

Факультет компьютерных систем и сетей

Кафедра информатики

Курсовой проект по дисциплине:

«Программирование»

Пояснительная записка к курсовой работе

Тема работы:

«Органайзер для Android смартфонов»

Исполнитель

студент гр. 653502 \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Кабанец С.О.

(подпись дата)

Руководитель

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Козуб В.Н.

(подпись дата)

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(оценка)

Минск

2017 год

## Содержание

|  |  |
| --- | --- |
| 1. Анализ существующих решений | 4 |
| 1.1 Any.do | 4 |
| * 1. Todoist | 5 |
| 1.3 SplenDO (Список задач) | 6 |
| 1.4 Google Календарь | 6 |
| 2. Обзор мобильной разработки | 8 |
| 2.1 «Нативная» разработка против кроссплатформенной | 8 |
| 2.2 Технические решения разработки | 10 |
| 2.3 Разработка под Андроид | 11 |
| 3. Обзор архитектуры проекта. | 13 |
| 3.1 Общие сведения | 13 |
| 3.2 Структура проекта | 14 |
| 3.2.1 Пакет ui | 14 |
| * + 1. Пакет data | 16 |
| * + 1. Пакет util | 16 |
| * + 1. Пакет database | 17 |
| 3.3 База данных. | 18 |
| Выводы | 19 |
| Список использованных источников | 20 |

## Введение

Жизнь современного человека очень насыщена. Порой мы не успеваем уследить за всеми делами, планируем на дни вперед и забываем. Некоторые спасаются, оставляя заметки в календарях, что висят на стенах в квартирах. Или стикерами заклеивают весь монитор компьютера. Но разве это действительно удобно? Да, это помогает нам лучше планировать свое время и не позволяет допустить какую-нибудь досадную оплошность, например, забыть про встречу с другом.

А сколько раз Вы тратили время не на необходимые вещи, просто потому что забыли о них. Казалось бы, действительно важные вещи не забудешь, но ведь существует масса не обязательных вещей, которые мы бы хотели сделать. Например, позвонить родным. Забываясь в ежедневной рутине дел, мы не можем уследить за всем.

Тогда человек и придумал календарь. Вроде бы удобно. Год разбит на месяцы, а месяцы на дни. Можно вписать свои задачи на каждый день, например, в ежедневник.

Но это не удобно. Легко можно забыть ежедневник дома, а календари, как правило, стоят на столе или висят на стенах. Когда в мире появились первые смартфоны, эта проблема кардинально изменилась. Приход умного телефона в наши жизни, изменил их и теперь невозможно представить себе человека без смартфона. Ведь это так удобно!

Почти всегда с собой, занимает мало времени, а забыть его где-нибудь равносильно катастрофе. Здесь-то и появились первые календари и заметочники. Казалось бы, проблема решена. Теперь мы можем вписывать свои ежедневные задачи в телефон, удобно их просматривать и ни о чем не забывать.

Но людям этого мало. Им нужно еще более удобно. Теперь еще и важно какие именно функции предоставляет приложение, насколько удобно его использовать, насколько красивый интерфейс. С каждым годом на рынок выходит все больше мобильных приложений и ставки явно повышаются.

**Глава I. Анализ существующих решений**

Я решил создать мобильное приложение под Android OS (далее Андроид). Я выбрал именно эту платформу, так как на сегодняшний день это самая распространенная мобильная операционная система в мире. В 2016 году доля ранка составила 86,2%. Согласитесь, это очень существенная цифра.

Для начала, я решил посмотреть на существующие решения и проанализировать их. Что можно позаимствовать, а что делать не стоит. У всего есть свои слабые стороны, которые можно устранить в своем приложении.

Я просмотрел множество приложений, но кратко приведу анализ только четырех из них. По сути, существует как минимум три немного разные области очень друг на друга похожие: “to do” список, органайзер, календарь. В целом они выполняют одну и ту же функцию: сохраняют записки, с привязкой к дате, и показывают нам. Но специфика “to do” списка, по моим наблюдениям, заключается в том, что эти приложения отображают записи как список и не имеют встроенного календаря. Так же некоторые предоставляют интересную функциональность по отправке задач другим людям для обсуждения (хорошо подходит для менеджеров). Органайзеры и календари почти одно и то же, разве что календари имеют упор на различное отображение времени: день в неделе, недели в месяце, дни в месяце, месяцы в году, недели в году. Органайзеры – это что-то среднее между первой и второй группой.

Важно отметить, что по этим запросам в Play Market отображаются иногда одни и те же приложения.

Итак, анализировать будем Any.do, Todoist, Список задач и Google Календарь. Существует масса интересных приложений, кроме этих, но для краткости ограничимся ими.

Далее приведены отличительные особенности каждого приложения.

### 1.1 Any.do

10 млн. скачиваний.

Положительные стороны:

1. Бесспорно легкий дизайн. Не напрягает зрение и не отвлекает от основной цели. Вместе с тем, сразу скажем, что местами разработчики перестарались и сделали уж слишком разреженный интерфейс, который, казалось бы, должен облегчить работу, но, на самом деле, только заставляет делать больше действий.

2. Подсказки при вводе задачи или подзадачи, заметки. На мой взгляд, приятный бонус. Хотя сам так и не воспользовался.

3. Возможность разбить задачи на категории. Очень удобно в использовании.

4. Довольно обширная функциональность. Лично не пришлось пользоваться, но, уверен, многим пригодится. Например, отправка задачи на почту другу или прикрепление заметки с медиа файлами.

5. Возможность создавать напоминания на различное время.

Минусы вытекают из плюсов:

Первое, что мне лично не понравилось – это регистрация. Создано это для синхронизации между устройствами, что, безусловно, великолепно. Но, ведь я не знаю, что я скачал. Может быть, мне не понравиться приложение. Я не хочу регистрироваться или привязывать к существующему аккаунту то, что возможно через пять минут удалю. Хотя и заняла привязка к Google аккаунту около минуты, стоило бы сделать это отдельной функцией подключаемой по желанию, а не по принуждению.

2. Обучение. Это небольшой минус, но тем не менее. Может быть, стоило сделать интерфейс интуитивно понятным и отказаться от маленьких значков в пользу текста.

3. Полное отсутствие календаря. Единственная разбивка на время это сегодня, завтра и предстоящие. Я бы хотел планировать на недели и месяцы вперед и удобно наблюдать, как приближаются эти дни.

4. Нет временной шкалы в специальных разделах по типу «Личное».

5. Не видно установлено ли напоминание. Может быть, мне не нужно напоминать уже. Но я не замечу, что поставил лишнее напоминание, потому что оно ни как не отражается в основном списке.

6. Не видно подзадачи. Чтобы их посмотреть придется раскрыть информацию о задачи и потом в отдельном окне просмотреть подзадачи, что требует лишние действия.

### 1.2 Todoist

10 млн. скачиваний.

Анализ был проведен только бесплатной версии.

Во многом функциональность напоминает Any.do. Из основных плюсов нужно отметить приятный и удобный пользовательский интерфейс. Вместе с тем, плюсы на этом заканчиваются. Очень много удобной функциональности находятся в Премиум версии, которая стоит около 27 долларов за годовую подписку. В эту платную функциональность входят комментарии, напоминания, метки и другое. Похожую функциональность предоставляет any.do бесплатно. Также по-прежнему отсутствует календарь для удобного поиска даты.

* 1. SplenDO (Список задач)

1 млн. скачиваний.

Отличное приложение с простой функциональностью. Судя по описанию от разработчика, это его основное преимущество и отличие – простота в использовании. Можно сказать, что это приложение не такое красивое, как два предыдущих, но в этом и его отличительная черта – простой и незамысловатый дизайн. Многим людям нравится использовать более строгое приложение, нежели приложением с множеством анимаций.

Плюсы:

1. На мой взгляд, процесс создания задачи куда более простой, нежели у предыдущих приложений. Тут поле ввода задачи находится прямо на экране со списком задач и не требует перехода к новым окнам.
2. Можно добавить несколько задач одновременно.
3. Можно оставлять напоминания.

Минусы:

1. Отсутствует календарь для удобного поиска даты.

1.4 Google Календарь

100 млн. скачиваний.

Первым делом, нужно отметить, что Google Календарь имеет такое большое количество скачиваний (100 млн.), не только по причине того, что приложение действительно хорошее, но и по причине того, что Android является разработкой Google и пользователи, безусловно, доверяют ему.

Плюсы:

1. Календарь. Существует возможность просматривать дни, недели, месяцы, а также 3 дня на одном экране. Этому сопутствует временная шкала для просмотра задач по времени.
2. По-прежнему остается список всех задач разбитый на месяцы и недели.
3. Интересное разделение на цели и напоминания. По сути, разница лишь в том, что цели – это заранее составленный список дел, которые могут делать большинство людей, например, «Заняться спортом» или «Уделить время близким».
4. Умное составление расписания с учетом предпочтений.
5. При подключении к Интернет скачивает список государственных праздников.

Мне не удалось найти какие-нибудь важные отрицательные особенности этого приложения. Оно действительно сделано очень хорошо и достойно своих 100 млн. скачиваний.

Отсюда можно сделать следующие выводы:

1. Дизайн стоит делать простым, анимированым, но не слишком разреженным.
2. Должна быть возможность навигации по календарю.
3. Нужны напоминания.
4. Нужна разбивка на группы и навигация по ним.

В целом уже этого достаточно для неплохо органайзера. Конечно, можно еще добавить подзадачи и отправка другу, но это оставим «на потом».

**Глава II. Обзор мобильной разработки**

2.1 «Нативная» разработка против кроссплатформенной

Существует несколько подходов к созданию мобильного приложения. Первый вариант это разработка при помощи инструментов предоставляемых компанией-создателем данной операционной системы. Этот способ называется «нативная» разработка. В случае Андроид это Android SDK от Google. В противоположность этому способу существует способ кроссплатформенной разработки. Принцип основывается на использовании сторонних фреймворков, а название кроссплатформенный получило благодаря возможности портировать одно приложение почти без изменений на несколько разных операционных систем. Так Xamarin позволяет разрабатывать приложение на Android и iOS при помощи C#. А фреймворк QT при помощи С++ предоставляет возможность создавать приложения на множество операционных систем: Windows, Linux, macOS, Android, iOS.

Важно отметить, что кроссплатформенная разработка не является разработкой одного приложения, так же как и нативная. Кроссплатформенная разработка позволяет оставлять неизменным лишь бизнес логику в то время как пользовательский интерфейс разрабатывается каждый раз с нуля для каждой операционной системы. Поэтому в итоге, мы получаем разные приложения с одним общим ядром. Однако не все кроссплатформенные решения обязуют создавать новые интерфейсы пользователя под каждую систему. Qt способна запустить одно приложение без изменений на нескольких платформах, но никто не гарантирует, что оно отобразится хоть сколько-нибудь правильно. В виду этого каждое приложение либо подгоняется, либо перерабатывается заново. При чем, последний вариант предпочтительней т.к. существуют стандарты внешнего вида приложения для каждой платформы, которым следует соответствовать. Сегодня уже давно не 2000-х год и люди научились отличать хорошую дизайнерскую работу от «сделанной на коленке». Это особенно хорошо видно на рынке мобильных приложений.

В свою очередь не менее распространенный Xamarin не позволяет разработчику запустить одно приложение на нескольких платформах в виду того, что под каждую операционную систему используется свой внутренний набор инструментов.

В то же время, нативная разработка подразумевает создание не просто разных приложений средствами одного фреймворка, а абсолютно независимое создание приложений никак между собой не связанных. Вплоть до дизайна. Android и iOS приложения имеют во многом разный интерфейс, связанный с философией данных систем, хотя общие черты, конечно, имеются. Нативные приложения пишутся как правило на разных языках. Основным языком программирования под Android является Java, а под iOS – Swift. Однако существуют еще множество языков программирования, на которых ведется разработка приложения под эти операционные системы, при чем, это все еще нативная разработка. Так под Андроид пишут на таких языках как Scala и Kotlin (набирает большую популярность). Эти языки используют JVM для запуска скомпилированного кода, поэтому могут легко использоваться рядом с Java. Именно поэтому они используют Android SDK и именно поэтому относятся к нативной разработке. Под iOS можно писать также на Objective-C.

Исходя из выше сказанного, возникает вопрос: а когда разработчики используют кроссплатформенные технологии?

Существует три варианта, когда необходимо использовать кроссплатформенные решения:

1. Приложение изначально планируется на несколько платформ и разработчики не хотят тратить ресурсы на разработку нативных приложений. Так как нативный способ требует кардинально изменять существующие решения, то это очень дорогой подход. Хотя бы потому что придется нанимать как минимум двух специалистов для каждой платформы, ведь очень тяжело найти программиста, который способен нативно создать качественное приложение на две разные платформы. Потребуется больше времени ведь один человек не способен делать два дела одновременно, и, в конечном итоге, все равно придется нанимать кого-либо еще для поддержки приложений.
2. Приложение уже имеет какие-либо модули, написанные на определенном языке, и существует кроссплатформенный фреймворк, поддерживающий данный язык. Тогда для того чтобы избежать переписывания кода, он может быть портирован при помощи данного фреймворка
3. Разработчик знает определенный язык программирования, ему нужна мобильная версия чего-либо, и существует фреймворк поддерживающий данный язык. Так называемый случай – «нравится». Относится только к разработчикам-одиночкам или небольшим группам разработчиков.

Основным плюсом нативной разработки является отсутствие в ней приведенных выше недостатков кроссплатформенных решений.

В последнее время в мире мобильной разработки существует тенденция отхода программистов от классической разработки. Под классической разработкой подразумевается программирование продиктованное компанией-создателем данной платформы. Так, для Андроид существует язык Java, паттерн архитектуры MVC, Android SDK. Сегодня создается очень много интересных технологий от сторонних разработчиков, которые призваны облегчить программирование под ту или иную платформу. Их создают одиночные программисты или целые компании, как правило, для своих внутренних нужд, однако позже увидев, что это хорошо, они выкладывают в общий доступ свои наработки.

Например, в недавнем времени появился такой инструмент как Android NDK. Он создан для запуска С++ кода на Андроид. Используется в основном разработчиками игр, так как способен увеличить скорость работы приложения в разы, если есть необходимость использовать сложные математические расчеты.

Android NDK изначально был разработкой одного человека и через некоторое время Google включила его в свою поддержку. После этого этот проект получил несколько другой вектор развития, но его разработчик создал новый похожий проект и продолжает развивать его по-своему и по сей день.

2.2 Технические решения разработки

Среди шаблонов архитектуры в настоящее время имеют большую популярность MVP, MVVM и Clean Architecture. Все они имеют свои плюсы и минусы, и споры об их использовании не утихают до сих пор. Популярная ранее MVC довольно быстро стала отвергаться большинством разработчиков. Связано это с тем, что ключевой элемент Controller быстро превращается в God-object (антипаттерн), т.е. выполняет очень много разрозненных функций, которые не должны быть в одном месте согласно «дядюшке Бобу» (и его легендарной концепции SOLID. в частности единство ответственности). Поэтому и были придуманы MVP, MVVM и другие.

На недавней конференции по мобильной разработке Mobius 2017 проходившей в Санкт-Петербурге, и на которую я, к сожалению, не смог попасть лично, очень много докладов было посвящено архитектуре, что говорит о том, что сейчас в мире Андроид это довольно острая проблема.

Также, в настоящее время становится популярным такая технология как Reactive. В частности RxJava. Она позволяет создавать код куда более лаконичным и простым, имеет большую популярность и реализовано для многих языков. Ввиду относительно небольшой сложности программы (по сравнению с большими коммерческими проектами) я использовать эту технологию не буду.

Можно еще много говорить про мобильную разработку. Про такие вещи как Dagger 2, и инструменты прототипирования и тестирования. Но не считаю это необходимым, ведь они отлично подходят для больших систем, а в простых приложениях будут излишни.

Ввиду отсутствия достаточного количества времени для изучения и относительно небольшой сложности проекта, к сожалению, здесь я не использовал многое то, что используется сегодня мобильными разработчиками. А так как код получился без использования сложным вещей и без узких мест, то не стал нагружать проект юнит тестированием, ведь, как известно, их тоже нужно сопровождать и поддерживать, что отнимает дополнительные ресурсы. Не считаю это необходимым здесь.

2.3 Разработка под Андроид

С первой версии Андроид Google приняла правильную политику касательно версий операционной системы. Так, приложения, написанные на предыдущие версии Андроид, будут запускаться на последующих версиях. Таким образом, обеспечивается совместимость приложений с последующими версиями. Однако, конечно же, приложение никак не изменится внешне и останется выглядеть «старой», что не понравиться пользователю. Обратная совместимость невозможна. Т.е. то, что работает на новых версиях легко, может не работать на старых версиях. Это привносит еще больше проблем в жизнь Андроид разработчика, потому что Google очень любит добавлять много новшеств в каждую последующую версию Андроид и говорить, что именно так и только так нужно создавать свои приложения. Поэтому в код постоянно приходится вставлять проверки на версию SDK и реализовывать функциональность несколько раз для нескольких версий. При чем, порой подход может меняться кардинально.

Но, прислушавшись к негодованиям разработчиков, Google начла выпускать библиотеки поддержки (support libraries) которые призваны обеспечить обратную совместимость вплоть до весьма ранних версий Андроид. Это, безусловно, облегчает жизнь разработчика, но, к сожалению, не все новшества сопровождаются библиотеками поддержки, что заставляет разработчиков создавать свои «кастомные» решения. Благодаря такому ресурсу как GitHub, разработчики могут делиться своими решениями друг с другом, что очень удобно. Например, последнее «кастомное», что я использовал, была красивая анимация Ripple (волна), которую ввели в Android 5.0 и не поддержали для предыдущих версий.

Очень важным решением при начале разработки своего приложения является выбор версии SDK. Это важно, ведь именно от этого зависит, какую функциональность придется переделывать под предыдущие версии, а какую нет. Например, если выбрать SDK 21, что соответствует Android 5.0, то почти не придется что-либо реализовывать несколько раз. Однако покроется только 5% устройств, зарегистрированных в Google Play (по данным Google). С другой стороны если выбрать SDK 15, или Android 4.0.3, то поддерживаться будет более 97% устройств, зарегистрированных в Google Play, но головной боли будет больше.

Так как проект – органайзер, т.е. он должен быть доступен для всех, то версия была выбрана 15, т.е. абсолютное большинство устройств.

**Глава III. Обзор архитектуры проекта.**

3.1 Общие сведения

В Java класса разбиваются по пакетам. Пакет это объединение классов по определенной общей цели. Например, работа с базой данных, подключение к Интернет и т.д. Пакеты используются также в Андроид для поиска определенных данных в других приложениях. Так, можно из одного приложения вызвать другое, если знать имя корневого пакета. Поэтому чтобы не возникло ошибок при установке из-за того, что пакет совпал с пакетом уже существующего приложения, необходимо давать уникальное имя пакету. Абсолютно все разработчики должны использовать структура очень похожую на доменное имя сайта только наоборот. Поэтому разработчики придумывают доменное имя для, своего блога или сайта-визитки, например simonlab.com, проверяют, существует ли такое в интернете, и если нет - используют в своем приложении. Так гарантируется, что не совпадет имя проекта с существующим. Правилом хорошего тонна является прописать в пакете также и название программы и что-нибудь вроде раздела на сайте-визитке, в котором разработчик бы расположил свой проект. Т.е. organizer.projects.simonlib.com. Теперь остается только отразить доменное имя наоборот. Получим com.simonlib.projects.organizer. Далее, прописывается структура проекта.

Не обязательно регистрировать доменное имя, хотя это конечно желательно. Нужно это, во-первых для удобной навигации и во-вторых для гарантии несовпадения имен пакетов. Если приложение предоставит доступ к своим данным, то через такой корневой пакет, как был показан выше, легко определить владельца программы и что она из себя представляет.

В программе может быть только один корневой пакет. При этом в папке Java кода в проекте приложения может находиться несколько разных пакетов никак друг с другом не связанных.

Далее название пакета расширяется внутренней структурой программы. По-моему, в любом приложении должны существовать такие подпакеты как ui, util, data. Первый предоставляет оболочку для отрисовки экрана. Именно здесь находятся все ключевые классы программы. По желанию этот пакет может быть дальше расширен подпакетами activity, fragment, view или другими. В моем случае я не стал этого делать, потому что получилось не большое классов. В util находятся, так называемые, утилитарные классы. Кстати говоря, наличие таких классов в последнее время не считается хорошим решением, и исправляется новыми технологиями мобильной разработки. Но, как я уже говорил, в проекте не используются эти нововведения, поэтому наличие этого пакета оправдано. В этом проекте находятся классы, выполняющие черновую работу, которая необходима, но не относиться ни к бизнес логике, ни к отрисовке. Часто можно встретить преобразование из одних элементов в другие. В нашем случае это работа по разбору календаря и преобразование стандартного класса Calendar к нашему классу OrganizerDate. И пакет data содержит в себе классы, которые хранят специфические структуры данных.

Отсюда видим, что ui является модулем view, а data модулем model в MVC, MVP и MVVM. Также, могут быть такие подпакеты как presenter или interactors в зависимости от выбранной архитектуры.

Прежде чем переходить к структуре проекта, необходимо заранее сказать, что такое view в Android. Все элементы на экране являются view, но не все доступны разработчику для изменения. Например, кнопки, полосы прокрутки, списки, таблицы, так называемые, карточки – все это view.

3.2 Структура проекта

Рассмотрим структуру проекта.

Подпакеты ui, util и т.д. пакета com.simonlib.projects.organizer я буду называть пакетами для простоты, потому что именно они несут смысловую нагрузку.

Есть 4 пакета: ui, util, database и data.

3.2.1 Пакет ui

Здесь 9 классов.

Основным является класс OrganizerActivity. По сути, это класс, который только запускает фрагменты. Здесь надо сказать пару слов о фрагментах. Все окна, которые открываются в приложении являются Activty. Разработчик наследует этот класс и говорит программе, как должно выглядеть окно. Но начиная с Андроид 3,0 были введены фрагменты. Фрагменты это почти то же самое, что и Активити, если не вдаваться в детали. Но Активити может быть одно на экране, а фрагментов внутри одного активити, соответственно на одном экране, может быть много. Это полезно, когда приложение запускается на планшете с большим экраном. Тогда на него кладут несколько фрагментов. Вы, конечно же, видели как в приложениях, где есть список и содержимое, активно применяется этот подход. На смартфонах сначала открывается список, а потом содержимое списка. На планшетах открывается список и его содержимое рядом одновременно. Поэтому я использую несколько фрагментов: первый – это календарь с днями, а второй – записи на этот день. Это OrganizerCalendarFragment и OrganizerDayFragment соответственно. Так вот OrganizerActivity открывает и уничтожает эти фрагменты, когда потребуется.

Далее необходимо отметить о паттерне Адаптер, который широко применяется в Android SDK. Он нужен чтобы связать некоторое view (виджет, который будет на экране) с данными, которые это view должно отобразить. Так как view заранее не может знать какого типа данные мы захотим в нем расположить, а также сколько их будет и как с ними работать, то создается класс адаптера. Он наследует некоторый тип адаптера из SDK. Переопределяя некоторые методы, зависящие от типа адаптера, разработчик может сказать, что и как обрабатывать view. Адаптеры могут быть одни и те же для нескольких типов view. Например, список (ListView) и таблица (GridView) принимают одинаковые типы адаптеров.

Месяцы в календаре пролистываются движением пальца вбок. Существует класс ViewPager который обеспечивает прокрутку. Для создания адаптера нужно унаследовать класс FragmentStatePagerAdapter либо FragmentPagerAdapter. Разница лишь в том, будет задерживаться каждая страница в памяти или нет. Я выбрал FragmentStatePagerAdapter, и в моем случае наследником будет OrganizerViewPagerAdapter. Он обязательно должен создает Fragment, который и будет являться содержимым ViewPager. Создание фрагментов обосновано: гораздо легче наполнить содержим разработчику и легче управлять ими ViewPager’у. В моем случае это OrganizerCalendarPage. Здесь нет постфикса Fragment, так как я хочу подчеркнуть, что это не совсем отдельный и независимый Fragment, а, по сути, лишь частью прокрутки.

Внутри этого класса создается таблица RecyclerView с GridLayoutManager внутри из конкретных дней. В Android существует два варианта создания списка и два варианта создания таблицы. Первый и немного устаревший это специальные классы ListView и GridView. Их особенность заключается в том, что они не сохраняют элементы, когда те пропадают с экрана при прокрутке. Однако существует RecyclerView, который способен создавать и списки, и таблицы и может кэшировать элементы в памяти, чем достигается более плавная прокрутка.

RecyclerView требует адаптер RecyclerView.Adapter, который является обобщенным классом. Его параметром обязательно должен быть наследник ViewHolder. Суть в том, что произвольные данные должны быть положены внутрь ViewHolder, который будет отдан адаптеру на обработку. Этот паттерн считается наиболее предпочтительным. В моем случае это OrganizerCalendarPageAdapter и OrganizerDayGridViewHolder соотвественно.

OrganizerCalendarPageItem является элементом таблицы RecyclerView, которая отображает месяц. Соотвественно этот класс – день в месяце. Но не обязательно это должно быть просто число. Здесь может находиться и дополнительная информация. Можно было бы и не создавать этот класс, а все оставить в ViewHolder адаптера, что некоторые и делают, но тогда, на мой взгляд, нарушается инкапсуляция и единство ответственности. Поэтому, желая сделать код более понятным, я инкапсулировал все, что связано с отрисовкой дня в месяце в этом классе.

При нажатии на день в месяце открывается список с записями. Это отдельный фрагмент, который называется OrganizerDayFragment. Внутри находится список – тот же RecyclerView, но внутри ListLayoutManager – и для него создан OrganizerDayListAdapter.

Остался класс OrganizerTabView. Из всех классов это полноценный view. Он представляет из себя название месяца и год и две кнопки вправо и влево. Он связывается с ViewPager и изменяет месяц и год в зависимости от того, какой ViewPager показывает месяц. Также способен пролистывать ViewPager программно в сторону соответствующею нажатой кнопке.

* + 1. Пакет data

Пакет data хранит структуры данных. Здесь 2 класса.

OrganizerEntry – это POJO (plane old java object) класс. Его задача – собрать вместе дату и запись. Нужен для удобной работы с записями внутри программы.

OrganizerDate – класс, который работает с датой в контексте данной программы. По своей сути, нужен только, чтобы собрать вместе день, месяц и год и удобно с ними работать. Уже существует класс Calendar в Android SDK, который предоставляет почти всю возможную работу с датой и временем, но он довольно «тяжелый» и дает слишком много лишней функциональности. Поэтому был разработан класс OrganizerDate. В отличии от POJO классов этот способен парсить строки и создавать себя из Calendar и наоборот преобразовываться в простую строку и в объект Calendar.

* + 1. Пакет util

Пакет util содержит всего один класс OrganizerUtil. Это, если так можно выразиться, «статический» класс. Как известно такого понятия в Java нет, но я подразумеваю класс, который состоит только из статических методов. Здесь преобразование даты в разные форматы: для отображения на экране, для записи в базу данных, выборка только уникальных записей из списка и доставание даты из Calendar по смещению относительно текущей даты (активно используется в пакете ui). Получается, что это своего рода god-object, что конечно же плохо. В проектах обычно такие util классы разбиваются на более специализированные, например DataUtil. Но ввиду небольшого объема программы я посчитал это излишним.

* + 1. Пакет database

Пакет database содержит 5 классов.

OrganizerDatabaseHelper - это необходимый для работы с базой данных класс, который создает, уничтожает и обновляет базу данных.

OrganizerDatabaseContract задает внутреннюю архитектуру базы данных. Так, мы имеем одну маленькую таблицу с тремя колонками: дата записи, текст записи и уникальный id, который создаст сама SQLite.

OrganizerDatabaseLoader – класс, который наследует AsyncTackLoader. Суть в том, что чтение из базы данным может занять некоторое время. Если чтение делать в один поток с графическим интерфейсов (основным потоком, в котором работает программа), то при медленном чтении из базы приложение не будет реагировать на действия пользователя и через 3 секунды появится сообщение: «Приложение не отвечает». Хотя на современных смартфонах Вы почувствуете замедленнее лишь при попытке прочесть около 50 000 записей, никогда не знаешь заранее, при каких нагрузках на смартфон будет запускать приложение, поэтому было бы неплохо делать приложение по возможности быстрее. Существует принцип: пока не нужно оптимизировать – не оптимизируй. Однако создание многопоточности (или асинхронности) скорее архитектурное решение и чем раньше оно будет принято, тем лучше для всего приложения. Кроме того, что данный класс загружает данные асинхронно, он еще и привязывается к отображаемому фрагменту и при, например попытке закрыть его, класс прекратит загрузку данных. Это логично, ведь зачем загружать данные которые не будут отображены.

С целью сделать асинхронными запись и изменение данных создан класс OrganizerDatabaseAsyncOperations.

OrganizerDatabase представляет интерфейс для работы с базой данных для всего приложения. Так, все выше описанные классы ничего не знают о том, какие конкретно данные вставлять или выбирать из базы. Об этом заботиться класс OrganizerDatabase, который упаковывает специфичные данные и отдает их OrganizerDatabaseAsyncOperations или OrganizerDatabaseLoader. Он же их и распаковывает. Таким образом, существует одна отдельная точка входа.

Кроме классов java в проекте также существуют и xml файлы ресурсов. Они являются разметкой отдельных экранов или их элементов, либо же описанием каких-либо свойств элементов, что предотвращает дублирование кода. Эти xml файлы парсятся внутри java классов или же, в контексте Андроида, инфлейтятся (от inflate, раздувание). Это слова пошло от метода inflate, который преобразует содержимое xml файла в java класс View.

3.3 База данных.

SQLite - компактная [встраиваемая](https://ru.wikipedia.org/wiki/%D0%92%D1%81%D1%82%D1%80%D0%B0%D0%B8%D0%B2%D0%B0%D0%B5%D0%BC%D0%B0%D1%8F_%D0%A1%D0%A3%D0%91%D0%94) [реляционная база данных](https://ru.wikipedia.org/wiki/%D0%A0%D0%B5%D0%BB%D1%8F%D1%86%D0%B8%D0%BE%D0%BD%D0%BD%D0%B0%D1%8F_%D0%B1%D0%B0%D0%B7%D0%B0_%D0%B4%D0%B0%D0%BD%D0%BD%D1%8B%D1%85). [Исходный код](https://ru.wikipedia.org/wiki/%D0%98%D1%81%D1%85%D0%BE%D0%B4%D0%BD%D1%8B%D0%B9_%D0%BA%D0%BE%D0%B4) библиотеки передан в [общественное достояние](https://ru.wikipedia.org/wiki/%D0%9E%D0%B1%D1%89%D0%B5%D1%81%D1%82%D0%B2%D0%B5%D0%BD%D0%BD%D0%BE%D0%B5_%D0%B4%D0%BE%D1%81%D1%82%D0%BE%D1%8F%D0%BD%D0%B8%D0%B5). Слово «встраиваемый» означает, что SQLite не использует парадигму [клиент-сервер](https://ru.wikipedia.org/wiki/%D0%9A%D0%BB%D0%B8%D0%B5%D0%BD%D1%82-%D1%81%D0%B5%D1%80%D0%B2%D0%B5%D1%80), то есть [движок](https://ru.wikipedia.org/wiki/%D0%94%D0%B2%D0%B8%D0%B6%D0%BE%D0%BA) SQLite не является отдельно работающим процессом, с которым взаимодействует программа, а предоставляет [библиотеку](https://ru.wikipedia.org/wiki/%D0%91%D0%B8%D0%B1%D0%BB%D0%B8%D0%BE%D1%82%D0%B5%D0%BA%D0%B0_(%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5)), с которой программа компонуется, и движок становится составной частью программы. Таким образом, в качестве протокола обмена используются вызовы функций библиотеки SQLite. Такой подход уменьшает накладные расходы, время отклика и упрощает программу. SQLite хранит всю базу данных (включая определения, таблицы, индексы и данные) в единственном стандартном файле на том компьютере, на котором исполняется программа. Простота реализации достигается за счёт того, что перед началом исполнения транзакции записи весь файл, хранящий базу данных, блокируется; [ACID](https://ru.wikipedia.org/wiki/ACID)-функции достигаются в том числе за счёт создания файла журнала.

Несколько потоков могут одновременно без каких-либо проблем читать данные из одной базы. Запись в базу можно осуществить только в том случае, если никаких других запросов в данный момент не обслуживается; в противном случае попытка записи оканчивается неудачей, и в программу возвращается код ошибки. Другим вариантом развития событий является автоматическое повторение попыток записи в течение заданного интервала времени.

Также важно отметить об ORM. Cуществует несколько ORM для Андроид разработчиков, но в нашем случае они не нужны. Android SDK предоставляет некоторую упрощенную версию ORM самостоятельно.

В приложении используется очень простая архитектура базы данных. Это одна таблица organizer\_table с двумя полями organizer\_column\_date и organizer\_column\_text. Оба имеют тип TEXT.

**Выводы**

На сегодняшний день реализованы функции просмотра даты в календаре (отображение только месяцев в году) и соответствующих записей, добавление, удаление и обновление данных. Осталось сделать напоминания и группы. К сожалению, я еще не успел реализовать эти функции на должном уровне, поэтому в проекте они не приводятся.

Но уже сейчас приложение выглядит вполне пригодным для использования. Конечно, многие функции еще предстоит добавить, но основные – добавление записей – уже реализованы хорошо. К слову, не раз благодарил себя за то, что сделал работу с базой данных асинхронной. На моем слабом смартфоне, если открыть много приложений, задержка при чтении базы данных становиться весьма заметной.

В процессе написания данной работы, я получил действительно много опыта в Андроид разработке. В частности получил опыт в асинхронном программировании, разработке Material Design интерфейсов (что, к слову, не стал описывать в основной части), работе с базой данных SQLite. Буду дальше развивать данный проект и использовать в будущем многие современные тенденции (чистая архитектура, RxJava, язык Kotlin и другие) по мере их детального изучения.

## Список использованных источников

1. Android Developers [электронный ресурс] – Режим доступа: <https://developer.android.com/index.html?hl=ru> - Дата доступа: 21.04.2017.
2. Android Developers API Guides [электронный ресурс] – Режим доступа: <https://developer.android.com/guide/index.html?hl=ru> – Дата доступа: 21.04.2017.
3. Habrahabr [электронный ресурс] – Режим доступа: <https://habrahabr.ru/post/200612/>. – Дата доступа: 20.04.2017.
4. SQLite [электронный ресурс] – Режим доступа: <https://www.sqlite.org/> – Дата доступа: 07.05.2017.