Федеральное государственное бюджетное образовательное учреждение высшего образования
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
АРХИТЕКТУРНО-СТРОИТЕЛЬНЫЙ УНИВЕРСИТЕТ

Факультет строительный
Кафедра информационных технологий

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА

Разработка приложения на Android с проектированием баз данных в SQLite
(тема ВКР)

Работу выполнил обучающийся Широбокова Полина Андреевна, ПМИ-4
Фамилия Имя Отчество № группы

Направление подготовки 01.03.02 Прикладная математика и информатика
Направленность (профиль) Прикладная математика и информатика

Руководитель:
канд. техн. наук Букунов С.В. _____
ученая степень, ученое звание Фамилия И.О. подпись

Консультант:
_____ _____ _____
ученая степень, ученое звание/ должность, место работы Фамилия И.О. подпись

Нормоконтролер Мовсесова Л.В. _____
Фамилия И.О. подпись

Допустить к защите

Заведующий кафедрой
_____ А.А. Семенов

«18» июня 2019г.

Санкт-Петербург 2019

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

С развитием технологий в XXI веке помимо прогресса появляется множество новых вызовов, требующих решения. Главным образом выделяют экологические катастрофы, развитие новых болезней и демографический кризис. Но образовывается еще одна проблема, которая не кажется устрашающей с первого взгляда, однако уже наносит удары по, например, экономике — это ухудшение ментального здоровья людей на фоне изменения мира коммуникаций, распространения интернета и социальных сетей и других перемен в образе жизни человечества. Имеются в виду заболевания по типу неврозов, депрессии, а также повседневный стресс.

Актуальность темы подтверждают следующие данные:

- из более 450 миллионов человек, страдающих ментальными расстройствами, лишь одна треть получает помощь [1];
- 20 278 человек в России совершили суицид в 2017 году. Большинство — взрослые;
- по прогнозам Всемирной Организации Здравоохранения к 2020 году психические расстройства войдут в первую пятерку болезней человечества;
- на 100 000 человек приходится только один психиатр в более чем половине стран мира;
- снижение производительности труда, связанное с развитием тревожных расстройств, обходится в $1 трлн для глобальной экономики.

Для лечения ментального здоровья можно обратиться к специалистам: психиатрам, психотерапевтам и психологам. Но в России далеко не каждый может позволить себе оплату многократных сеансов, а также сказывается отсутствие глобальной культуры поддержки ментального здоровья. Многие не признают свой недуг за серьезную проблему, не желая выглядеть слабыми и боясь осуждения.

Тем не менее, улучшать свое состояние самостоятельно возможно. Для этого есть медитации, различные физические практики и тренировка осознанности. Однако людям в подавленном состоянии тяжело дается самоконтроль, особенно в век перенасыщения информацией. Необходимы простые и понятные инструменты.

Благодаря широкой распространенности Интернета и мобильных устройств в развивающихся странах (80,9% населения на 2017 год), отслеживание и контроль собственных состояний возможно осуществлять с помощью приложения, установленного на телефоне.

Целью данной выпускной квалификационной работы является создание мобильного приложения, в котором возможно:

- оценивание своего настроения;
- указание последних действий, выполненных человеком;
- добавление более детального словесного описания в заметках;
- возможность указания количественной характеристики, относящейся к каждому действию
- просмотр истории записей.

1 Исследование предметной области и постановка задачи

1.1 Анализ существующих приложений

Поскольку на российском рынке направление отслеживания настроений не развито, исследовались зарубежные приложения. В качестве критерия отбора использовались ключевые слова поиска: «mood», «moodtracking», «diary», «journal», «happiness». Всего было рассмотрено 20 приложений из сервисов AppStore и GooglePlay, а в данной работе приведены три самые востребованные и отличающиеся друг от друга версии.

1.1.1 Приложение Daylio

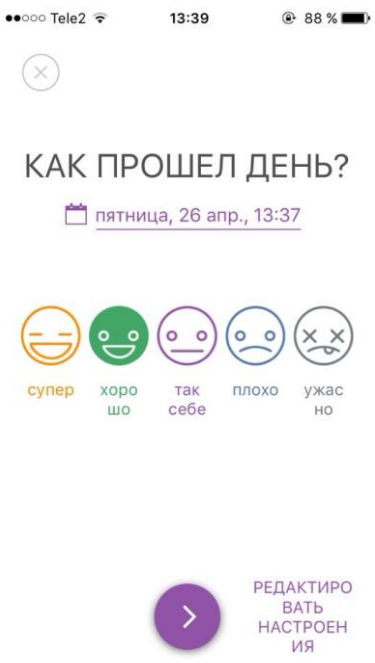

Рис. 1.1. Экран приложения Daylio

Приложение Daylio (рисунок 1.1) является одним из самых популярных на рынке. Оно представлено как на платформе iOS, так и на платформе Android.

В приложении реализованы возможности оценивания настроения по пятибалльной шкале, связывание настроений с действиями, отображение нескольких видов графиков зависимости настроения от выбранного фильтра.

Однако, в приложении недостаточно реализовано добавление заметок: они добавляются лишь в виде коротких примечаний, что значительно ограничивает пользователя в изложении мыслей. Также реализованы не все необходимые фильтры. Например, при просмотре графика зависимости настроений нельзя отсортировать факторы, влияющие на каждое настроение и сделать необходимые выводы. В Daylio отсутствует возможность добавления количественных характеристик.

1.1.2 Приложение eMoods

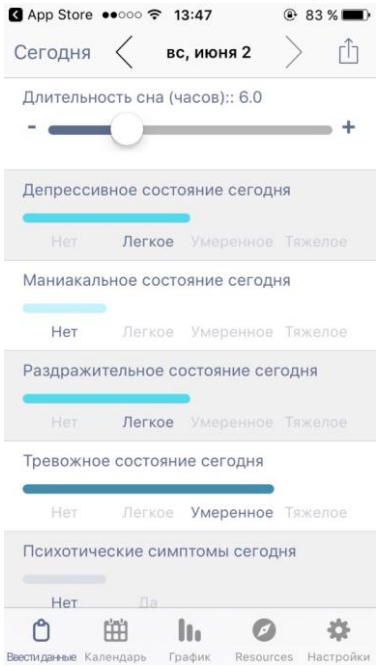

Рис. 1.2. Экран приложения eMoods

Приложение eMoods (рисунок 1.2) предлагает пользователю более обширный набор функций: семь параметров при оценке состояния. Тем не менее, данная программа предназначена для людей, страдающих от

биополярного расстройства. Поэтому, несмотря на более детальное описание настроения, оно подходит лишь ограниченному кругу людей.

1.1.3 Приложение Moodtrack Diary

Рис. 1.3. Экран приложения Moodtrack Diary

Приложение Moodtrack Diary (рисунок 1.3) — своего рода небольшая социальная сеть, где люди в открытом доступе публикуют отчеты о своих настроениях. Пользователь записывает свое состояние, сопровождая его комментарием, и на графике видит все введенные данные. Также пользователю доступно «облако настроений», которые в зависимости от частоты использования имеют больший или меньший размер шрифта. Недостаток этого приложения в открытом доступе пользователей к данным друг друга — закрытый режим доступен только в платной версии. Также недостаточно гибкая фильтрация: нужное слово приходится искать долгой прокруткой, нет поиска. Отсутствует раздел заметок.

1.2 Постановка задачи

Основываясь на целях данной ВКР, а также преимуществах и недостатках существующих решений, были выделены следующие задачи для реализации приложения:

- описание требований к приложению
- выбор необходимых средств и технологий для разработки (операционная система, среда разработки, СУБД и так далее);
- проработка функционала приложения;
- создание ER-модели базы данных;
- написание программного кода;
- оформление приложения

1.3 Требования к приложению

2 Обзор технологий для разработки приложения

2.1 Выбор операционной системы

Наиболее популярными мобильными операционными системами (ОС) на данный момент являются Android (86.2% рынка) и iOS (13.7% рынка). Разработка приложений на Android более доступна, так как является полностью бесплатной и может вестись с любой десктопной операционной системы, в отличие от iOS, где написание приложений требует наличие установленной macOS. Также даже для тестирования собственного приложения на iOS требуется заплатить взнос в $99.

Операционная система Android поддается большему количеству настроек: регулировать можно практически все параметры. В то время как часть кода операционной системы iOS находится в закрытом доступе.

Таким образом, наиболее оптимальной мобильной операционной системой для разработки приложения была выбрана Android.

2.2 Выбор API

В связи с тем, что Android имеет широкое распространение и используется на устройствах разных компаний, существует и немалое количество версий данной операционной системы. Подробная статистика распространения версий Android указана в таблице 2.1. Эта статистика учитывает устройства, в которых хотя бы один раз запускалось приложение Google Play (магазин Android).

Таблица 2.1. Распространение версий Android

Версия	Название	Год	Доля
2.3	*Gingerbread*	2010	0,3 %
4.0	*Ice Cream Sandwich*	2011	0,3 %
4.1	*Jelly Bean*	2012	1,2 %
4.2		2012	1,5 %
4.3		2013	0,5 %
4.4	*KitKat*	2013	6,9 %
5.0	*Lollipop*	2014	3 %
5.1		2015	11,5 %
6.0	*Marshmallow*	2015	16,9 %
7.0	*Nougat*	2016	11,4 %
7.1		2016	7,8 %
8.0	*Oreo*	2017	12,9 %
8.1		2017	15,4 %
9.0	*Pie*	2018	10,4 %
10.0	*Q*	2019	< 0,1 %

При выборе версии важно помнить об обратной совместимости: приложение будет работать одинаково на всех версиях, выпущенных после указанной минимальной. При разработке приложения на версии Android 4.4 (Kit Kat), оно будет доступно на 95.3% всех устройств, поддерживающих данную ОС, поэтому именно эта версия и соответствующий ей API 19 были выбраны для разработки.

2.3 Выбор среды разработки

На данный момент ведущими средствами разработки являются следующие:

- Android Studio. Программное обеспечение, созданное компанией Google на основе Intellij Idea от JetBreains. Возросший рост популярности ОС сподвиг Google создать собственную IDE, несильно отличающуюся от прародителя. Тем не менее, Android Studio содержит нововведения, позволяющие разработчикам гораздо проще и удобнее разрабатывать приложения. Данное ПО поддерживает разработку исключительно для Android. Полностью бесплатна.
- Intellij Idea. Среда разработки, позволяющая работать со многими языками, среди которых самыми популярными являются Java, JavaScript и Python. Существуют платные и бесплатная версии. Чтобы разрабатывать на Android, необходимо дополнительно устанавливать недостающие компоненты, например, Android SDK.
- Eclipse. Среда разработки, ориентированная, в основном, на язык Java. Тем не менее, благодаря подключаемым модулям, в среду возможно интегрировать практически все языки. Разработка на

Android возможна так же с помощью установки дополнительных расширений.

В качестве оптимальной среды разработки была выбрана Android Studio, как разработанная специально для этого компанией Google.

2.4 Выбор языка программирования

- Java – типизированный объектно-ориентированный язык программирования, являющийся официальным для написания приложений на Android. Устроена технология Java следующим образом: написанный программистом код компилируется в понятный компьютеру байт-код, после чего происходит интерпретация полученных инструкций с помощью Java Virtual Machine – вычислительного устройства [2]. Такая методика позволяет делать приложения, написанные на Java кроссплатформенными, то есть доступными на любых устройствах с установленной Java Platform. Плюсом разработки на Java является самое большое на данный момент количество написанной документации по разработке на Android среди всех языков программирования.
- Kotlin – типизированный объектно-ориентированный язык программирования, презентованный два года назад компанией JetBrains. Язык создавался для упрощения и минимизации кода, а также для устранения возникновения некоторых нюансов, например: избежание ошибки использования шулевого указателя NullPointerException [3]; ошибки компиляции из-за забытого знака «;». Kotlin, как и Java, интерпретируется с помощью Java Virtual Machine. Эта совместимость гарантирует стабильную работу приложения, написанного на обоих языках. Таким образом,

обновлять ранее созданное приложение можно без его переписывания, только лишь дополняя. Язык Kotlin также является официальным языком разработки для Android.

- C/C++ — типизированные объектно-ориентированные языки программирования. Несмотря на поддержку этих языков с помощью Android Native Development Kit, разработка под выбранную операционную систему на них непроста и рекомендуется в исключительных случаях. Например, при написании сложных игр, графики или при большом взаимодействии с памятью, сенсорами и другими элементами устройства. Этот подход позволяет получать максимум ресурсов для приложения.

Остальные языки программирования не поддерживаются средой разработки Android Studio, хоть и позволяют проектировать приложения. Несмотря на то, что Kotlin в сравнении с Java выигрывает в лаконичности кода и обходе некоторых ошибок, для данной работы был выбран язык Java, так как по нему все еще можно найти большее количество документации и информации.

2.5 Выбор способа хранения данных

Всего Android предлагает 3 способа хранения данных:

1. Preferences. В результате выполнения программы создается файл с расширением .ini, являющийся файлом настроек. Таким способом возможно хранить пары ключ-значение. Этот способ не подходит для хранения больших данных и не предоставляет гибкого функционала для использования данных.
2. Использование внутреннего/внешнего хранилищ. Android позволяет пользователю сохранять текстовые и двоичные файлы в специальных каталогах. Этот подход аналогичен потоковому подходу (Stream) в

языках C++ и Java. Способ неактуален для данной работы из-за сложного манипулирования данными.

3. СУБД. В Android есть встроенная система управления базами данных SQLite. Она отличается быстрой производительностью, гибкостью и возможностью хранить большие объемы данных. Для удобства обработки команд в Android есть встроенный класс SQLiteOpenHelper, организовывающий взаимодействие приложения с данными, выполнение запросов. Возможность написания полноценных SQL запросов позволяет получать сложные зависимости. Также для облегчения работы с данной СУБД созданы несколько надстроек. Например, ORM (Object-Relational Mapping) – технология, позволяющая с помощью соединения принципов объектно-ориентированного программирования и основ баз данных создавать виртуальную объектную базу данных. Основные ORM для Android: Room, Realm, ORMLite, GreenDAO. Для облачного хранения предлагается использовать Firebase, основанный на SQLite.

В итоге для хранения данных приложения был выбран способ с использованием СУБД, а именно SQLite.

3 Разработка функционала и архитектуры приложения

3.1 Необходимый функционал приложения

Приложение, создаваемое в рамках данной ВКР должно позволять выполнять следующие функции:

- отмечать текущее настроение с выбором из предложенных вариантов;
- отмечать последние выполненные действия из списка существующих;
- указывать количественные характеристики, относящиеся к действию (например, было прочитано n-ое количество страниц);
- добавлять заметки;
- просматривать существующие записи;
- отображать график зависимости настроения от времени;
- отображать график зависимости настроения от времени по выбранному фильтру действия;
- отображать статистику средних настроений по каждому действию;
- отображать статистику количественных характеристик по каждому действию.

3.2 Схема работы приложения

На схеме, представленной на рисунке 3.1, показаны основные окна приложения и их взаимосвязь. Через стрелки демонстрируются возможные переходы пользователя в интерфейсе. Помимо этого, к трем основным окнам: окну выбора настроений, окну с выбором графиков и окну с просмотром

существующих записей можно получить доступ через нижнее навигационное меню приложения.

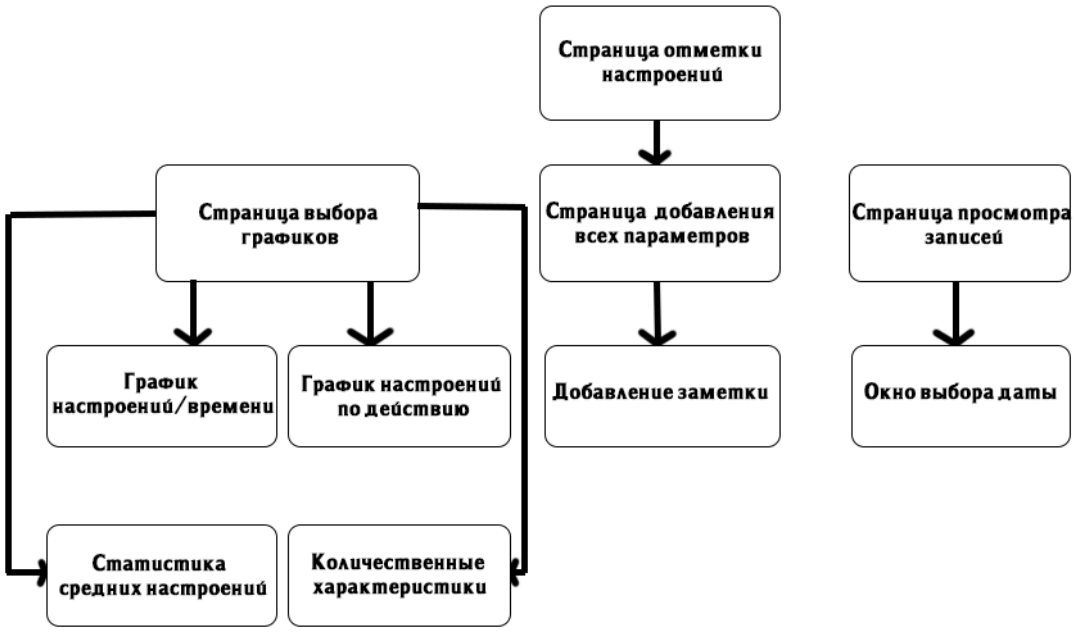

Рис. 3.1. Схема работы приложения

3.3 Описание страниц приложения

В приложении представлены следующие окна:

- Страница для выбора текущего настроения;
- Страница добавления всех необходимых параметров;
- Страница редактирования заметки;
- Страница с представлением выбора графиков и статистики;
- Страница с графиком зависимости настроения от времени;
- Страница с графиком зависимости настроения от времени для конкретного действия;
- Страница с показателями средних настроений для каждого действия;

- Страница с показателями количественных характеристик для каждого действия;
- Страница отображения записей по выбранному дню;
- Страница для выбора даты;
- Страница с информацией по выбранной записи.

На странице выбора настроения, которая является также главной, так как именно она отображается при запуске приложения, пользователь выбирает текущее настроение из пяти представленных. Настроения отображаются в виде иконок, загружаемых в соответствие с информацией в таблице базы данных и подписанных соответствующим им текстом.

После выбора настроения пользователь перенаправляется на страницу выбора дополнительных параметров. Выбранное настроение отображается в самой верхней части окна. Из списка действий, загруженных из базы данных, возможно выбрать последние выполненные действия. Длительное нажатие на иконку действия спровоцирует появление счетчика: это необходимо для указания количественных характеристик. При указании количества иконка действия станет черной, а в целом при выборе действия фон иконки окрасится в синий цвет. Ниже в окне есть поле для добавления заметок, нажатие на которое приведет к отображению соответствующей страницы. Кнопка сохранить сохраняет все веденные параметры в базу данных, считывая текущие время и дату.

Страница добавления заметок состоит из текстового поля и кнопки сохранить, нажатие на которую вернет пользователя на предыдущее окно выбора параметров. Введенная текстовая запись отобразится в окне выбора в поле для заметок.

Страница с представлением выбора графиков и статических показателей содержит четыре иконки, нажатие на которые приведет к открытию соответствующих им окон.

Страница с графиком зависимости настроения от времени содержит этот график с возможностью увеличения его для просмотра статистики на более мелком временном промежутке.

Страница с графиком зависимости настроения от времени по действию содержит переключатель со списком всех действий. Ниже расположена диаграмма. При выборе действия диаграмма обновится.

Страница с отображением средних настроений по всем действиям содержит список, в каждом пункте которого содержится иконка, соответствующая действию, название действия и среднее настроение.

Страница с показателями количественных характеристик для каждого действия содержит список с отображением иконок, названий действий и соответствующих им суммам всех количественных характеристик, введенных пользователем.

На странице с отображением записей по дате есть текстовое поле для показа выбранной даты, кнопка, вызывающая окно с календарем, а также список всех записей, показывающий время добавления записи и иконку настроения.

При нажатии на пункт записи пользователь будет перенаправлен на страницу с подробным отображением данных для выбранной записи. В верхней части окна показаны дата и время записи, ниже указанное настроение, далее выбранные действия и, если были выбраны, количественные характеристики. В нижней части окна отображаются заметки.

В окне выбора даты для отображения записей открывается календарь, нажатие на дату в котором закроет текущее окно и переключит внимание пользователя на предыдущее окно.

4 Разработка базы данных

4.1 Построение ER модели

Перед тем, как начать работать с базой данной и в целом приступать к ее созданию, необходимо проработать ER модель базы.

ER модель (entity-relationship model) олицетворяет собой схему «сущность»-«связь» среди объектов базы данных. Создавать ER модели возможно как на бумаге, так и в специальных системах управления базами данных. Текущая ER модель была создана в программе Microsoft Access.

Главным элементами, входящими в реляционную модель БД, являются сущность, связь, атрибуты, первичный ключ.

Сущность – представление таблицы базы данных. Сущность отвечает за объект данных, который будет использоваться в программе.

Атрибуты – свойства, характеризующие объект данных. Имена атрибутов также служат наименованием для колонок в таблицах сущностей.

Первичный ключ – главный атрибут, по которому возможно осуществлять связь таблицы с другими. Чаще всего в качестве первичного ключа выбирают ID таблицы, так как главное требование к ключу – его уникальность, а идентификационный номер чаще всего не повторяется.

Связи – характеризуют отношения между таблицами. Есть три вида связей: один к одному, один ко многим, многие ко многим.

Связь один к одному применяется в ситуациях, когда одной конкретной записи в первой таблице может соответствовать только одна запись из второй таблицы. Чаще всего такой подходи применяется для сокращения информации в базе данных, чтобы избежать дублирования информации.

Связь один ко многим является, наверное, самой распространенной. При такой архитектуре одной записи в первой таблице может соответствовать несколько записей в другой.

Связь многие ко многим используется при наличии в обеих таблицах многократных отсылок к строкам друг друга. Этот подход реализовывается за счет создания вспомогательной таблицы, в которой соединяются первичные ключи строк из двух таблиц. Такая реализация сделана для предотвращения дублирования информации, сохранения уникальности данных в исходных таблицах, и вынесении всех связей в другую сущность.

Для реализации приложения, отслеживающего эмоциональное состояние, была создана база данных следующей структуры, показанной на рисунке 4.1:

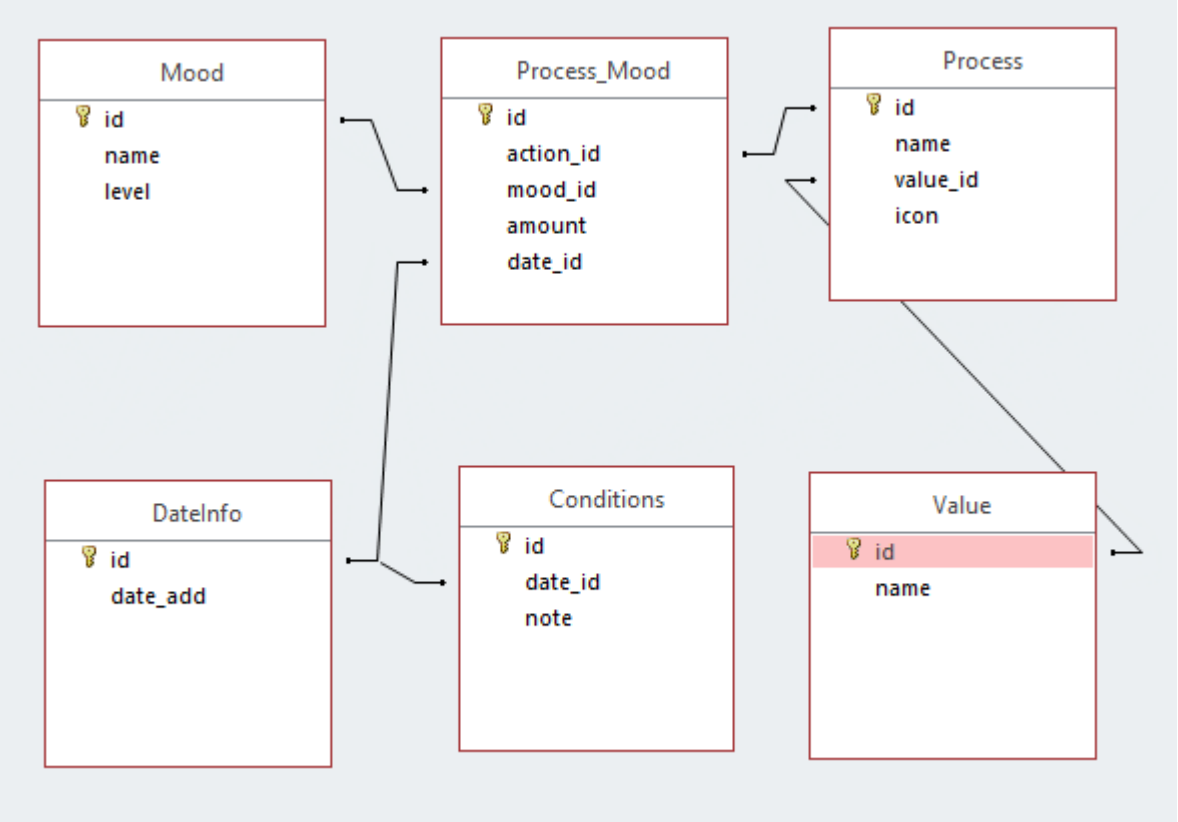

Рис. 4.1. ER модель базы данных

Таблица Mood отвечает за сущность «настроения». Первичным ключом служит поле id, в колонке name указываются названия настроений (ужасное, плохое, среднее, хорошее, отличное), в поле level указывается уровень настроения по шкале от одного до пяти.

Таблица Process представляет сущность «действия». В поле id указывается первичный ключ – идентификационный номер действия, в поле name – название действия (например, спорт, учеба, работа), в поле icon –

название иконки для отображения в программе. Иконки загружаются в проект в папку res/drawable, после чего извлекаются по имени. Поле value_id необходимо для связи с таблицей Value, чтобы при отображении количественных характеристик указывать значение единицы измерения.

Таблица Process_Mood служит как промежуточная таблица между Process и Mood для реализации связи многие ко многим. После добавления новой записи идентификационные номера обеих сущностей добавляются в эту таблицу вместе со значением количественной характеристикой действия, которая указывается в поле amount. Поле date_id необходимо для связи с таблицей DateInfo, которая предоставляет список всех дат, которые использовались при добавлении записей.

Таблица DateInfo имеет поля id – идентификационный номер и поле date_add – саму дату. Вынесение даты в отдельную таблицу создано для более точной работы с датами, ведь при обращении непосредственно к дате на всех этапах запросов пришлось бы постоянно конвертировать строковый формат SQLite, в котором хранится запись в базе данных, в форматы Date и Calendar у Java.

Таблица Conditions содержит поля id – идентификационный номер, поле date_id, связывающее сущность с таблицей DateInfo, и поле Note, которое отвечает за хранение заметок. Вынесение хранения заметок сделано в отдельную таблицу специально для того, чтобы при дублировании записей в таблице Process_Mood не происходило дублирования заметок, ведь иногда они могут содержать большое количество символов и занимать больше памяти. А в случае разграничения получить запись из обеих таблиц можно через двойной запрос по ключам date_id.

Таблица Value хранит в себе единицы измерения, которые присваиваются каждому действию из таблицы Process. Среди единиц измерения есть, например, часы, километры, штуки, страницы.

4.2 Проектирование базы данных в Room

Для предотвращения возможных ошибок и нагромождений в коде вместо исключительного использования стандартных методов в Android для обращения с базой SQLite используется библиотека Room.

Room – это фреймворк, устанавливаемый дополнительно в Android Studio в разделе зависимостей Gradle. Room представляет собой абстрактную «обертку» над стандартами SQLite и обеспечивает более надежный доступ ко всем базам данных при использовании SQLite. Проект создан компанией Google и входит в Android Architecture Components.

Библиотека позволяет создавать кеш данных приложения на устройстве, на котором выполняется приложение в данный момент. Этот кеш служит единственным источником достоверных данных для приложения, помогая пользователям просматривать точную копию ключевой информации в приложении вне зависимости от того, есть ли у пользователей подключение к Интернету.

В основу архитектуры библиотеки Room входят три компонента: Entity, DAO и Database. Однако для большего удобства предлагается использовать также классы ViewModel и Repository [4].

Entity – это аннотированный класс, описывающий таблицу в базе данных. Для того, чтобы класс соответствовал стандартам, необходимо до его объявления указать аннотацию @Entity, затем объявить нужные в таблице поля и обязательно указать первичный ключ с помощью аннотации @PrimaryKey. Для обеспечения доступа базы данных к классу необходимо, чтобы поля были общедоступными и были реализованы методы getters и setters для получения и установки значений.

Построение Entity в проекте на примере таблицы Process_Mood показано на рисунке 4.2:

```java
@Entity
public class Process_Mood implements Serializable {

    @PrimaryKey(autoGenerate = true)
    public long id;
    public long action_id;
    public long mood_id;
    public long amount;
    public long date_id;

    public long getId() { return id; }

    public void setId(long id) { this.id = id; }

    public long getAction_id() { return action_id; }

    public void setAction_id(long action_id) { this.action_id = action_id; }

    public long getMood_id() { return mood_id; }

    public void setMood_id(long mood_id) { this.mood_id = mood_id; }

    public long getAmount() { return amount; }

    public void setAmount(long amount) { this.amount = amount; }

    public long getDate_id() { return date_id; }

    public void setDate_id(long date_id) { this.date_id = date_id; }
}
```

Рис. 4.2. Реализация @Entity таблицы Process_Mood

Здесь реализовано представление 5 полей таблицы: id, action_id, mood_id, amount и date_id. А также описаны методы getters и setters. Параметр autoGenerate включает автоинкрементирование поля id при добавлении данных.

При необходимости особой манипуляции с объектом данных таблицы, добавлении новых полей или методов класса сущности, нужно перед объявлением переменной/метода указать аннотацию @Ingore, и тогда Room не будет конфликтовать с новыми полями. Пример использования этой аннотации показан на рисунке 4.3.

```
@Entity
public class Process implements Serializable {

    @PrimaryKey(autoGenerate = true)
    public long id;
    public String name;
    public int value_id;
    public String icon;
    @Ignore
    public boolean isSelected = false;
```

Рис. 4.3. Часть Entity таблицы Process с @Ingore

Для реализации самого доступа к данным используется интерфейс DAO, который отвечает за манипулирование данных хранимых в таблицах.

В интерфейсе DAO описываются SQL запросы, которые будут вызываться в дальнейшем коде программы. Для вставки и удаления данных в Room реализованы готовые аннотации @Insert и @Delete. Однако для выполнения не совсем стандартных запросов подобного типа такого подхода может не хватить. Для выборки и в целом выполнения любых запросов создана аннотация @Query, после котором записывается нужный SQL запрос. В запрос также можно передавать переменные из функции. Необходимо лишь указать их через двоеточие. Также при описании запроса указывается тип необходимых возвращаемых данных. Поэтому, если ожидается получить единичный объект, то нужно соответствующее указание. И, например, список объектов (List<…>), если объектов будет несколько.

```java
@Dao
public interface Process_MoodDao {
    @Query("SELECT * FROM process_mood")
    List<Process_Mood> getAll();

    @Query("SELECT * FROM process_mood WHERE action_id=:act_id")
    List<Process_Mood> getAllByProId(long act_id);

    @Query("SELECT * FROM process_mood WHERE date_id=:date_id")
    List<Process_Mood> getAllDate(long date_id);

    @Query("SELECT * FROM process_mood GROUP BY date_id")
    List<Process_Mood> getDateMood();

    @Query("SELECT * FROM process_mood")
    Process_Mood[] getAllArray();

    @Insert
    void insert(Process_Mood process_mood);

    @Insert
    void insert(List<Process_Mood> process_moods);
}
```

Рис. 4.4. Реализация DAO таблицы Process_Mood

В примере DAO для таблицы Process_Mood, показанном на рисунке 4.4 описываются основные запросы, используемые при работе с этой сущностью. Как можно заметить, в интерфейсе присутствует разнообразная выборка данных через аннотацию @Query: запросы меняются в зависимости от принимаемых аргументов и типов возвращаемых значений.

Класс Database служит для создания и обновления версий базы данных. Класс объявляется аннотацией @Database. Он должен быть абстрактным и наследовать класс RoomDatabase. В параметрах аннотации указываются все сущности, участвующие в базе данных, а также версия базы данных. Также в этом методе нужно описать абстрактные методы для работы с объектами DAO.

```
@Database(entities = {Mood.class, Process.class, Value.class, Conditions.class, DateInfo.class, Process_Mood.class},
public abstract class MoodDatabase extends RoomDatabase {
    private static String DB_PATH = "/data/data/com.example.android.moodtracker/databases/";
    public abstract MoodDao moodDao();
    public abstract ProcessDao processDao();
    public abstract ConditionsDao conditionsDao();
    public abstract DateInfoDao dateInfoDao();
    public abstract ValueDao valueDao();
    public abstract Process_MoodDao process_moodDao();
    private static volatile MoodDatabase INSTANCE;

    public static MoodDatabase getDatabase(final Context context) {
        if (INSTANCE == null) {
            synchronized (MoodDatabase.class) {
                if (INSTANCE == null) {
                    INSTANCE = Room.databaseBuilder(context.getApplicationContext(),
                            MoodDatabase.class,  name: "moodDb")
                            .build();
                }
            }
        }
        return INSTANCE;
    }
```

Рис. 4.5. Реализация Database для базы данных

На рисунке 4.5 представлена стандартная реализация абстрактного класса Database, в котором единожды создается база данных, и при последующем к ней обращении сущностей будет осуществляться проверка, при которой статическая переменная INSTANCE будет индикатором загруженности или не загруженности базы данных.

Repository – это класс, который абстрагирует доступ ко множественным источникам данных. Repository не является частью библиотеки Android Architecture Components, но служит рекомендуемой и лучшей практикой для правильного разделения кода и усовершенствования архитектуры программы. Класс Repository обрабатывает операции с данными.

Repository управляет потоками запросов и позволяет использовать сразу несколько обращений к базе. В наиболее распространенных примерах Repository реализует логику для принятия решения о том, нужно ли выбирать данные из сети или лучше использовать результаты, кешированные в локальной базе данных.

```
public class Process_MoodRepository {
    private Process_MoodDao mProcess_MoodDao;

    public Process_MoodRepository(Application application) {
        MoodDatabase db = MoodDatabase.getDatabase(application);
        mProcess_MoodDao = db.process_moodDao();

    }

    public List<Process_Mood> getAll() {
        return mProcess_MoodDao.getAll();
    }
    public List<Process_Mood> getAllDate(long date_id) {
        return mProcess_MoodDao.getAllDate(date_id);
    }
    public List<Process_Mood> getAllByProId(long action_id) {
        return mProcess_MoodDao.getAllByProId(action_id);
    }



    public Process_Mood[] getAllArray() { return mProcess_MoodDao.getAllArray(); }

    public void insertList(List<Process_Mood> process_moods) {
        mProcess_MoodDao.insert(process_moods);
    }

    public List<Process_Mood> getDateMood() { return mProcess_MoodDao.getDateMood(); }

}
```

Рис. 4.6. Реализация Repository для таблицы Process_Mood

В примере выше, на рисунке 4.6, описываются методы, реализующие доступ к прямым запросам к базе данных в ранее созданном интерфейсе DAO.

Роль ViewModel заключается в предоставлении данных для пользовательского интерфейса и сохранении изменений конфигурации. ViewModel действует как центр связи между Repository и пользовательским интерфейсом. Также возможно использовать ViewModel для обмена данными между фрагментами. ViewModel является частью библиотеки жизненного цикла.

Отделение данных пользовательского интерфейса приложения от классов позволяет лучше следовать принципу разделенной ответственности: Activity и фрагменты отвечают за отображение данных на экране, а ViewModel позаботится о хранении и обработке всех данных, необходимых для

пользовательского интерфейса. Пример реализации ViewModel показан на рисунке 4.7.

```
public class Process_MoodViewModel extends AndroidViewModel {
    protected Process_MoodRepository mProcess_MoodRepository;
    private List<Process_Mood> mAllProcess_Mood;
    public Process_Mood process_mood;
    public Process_MoodById process_moodById;

    private static final String TAG = "MyAppii";
    private final List<PropertyChangeListener> listeners = new ArrayList<>();

    public Process_MoodViewModel(Application application) {
        super(application);
        mProcess_MoodRepository = new Process_MoodRepository(application);

    }


    public Process_Mood[] getAllArray() {
        Process_Mood[] mAllProcess_Mood = mProcess_MoodRepository.getAllArray();
        return mAllProcess_Mood;
    }


    public void insertList(List<Process_Mood> process_moods, Long date_id, Adding activity) {
        new InsertListAsync(activity, date_id).execute(process_moods);
    }
```

Рис. 4.7. Часть описания класса ViewModel для Process_Mood

В данной работе почти все ViewModel имеют большое описание из-за наличия в них классов AsyncTask, необходимых для создания дополнительных потоков работы приложения, чтобы при работе с данными не блокировался основной поток – поток интерфейса.

5 Реализация приложения

5.1 Выбор макета для приложения

Выбор макета для приложения является важным шагом в разработке, так как при изменении макета меняется и архитектура приложения, что приводит к переписыванию кода и порой даже к изменению логики программы.

Макет определяет визуальную структуру пользовательского интерфейса, например, пользовательского интерфейса операции или виджета приложения. Существует два способа объявить макет:

- Объявление элементов пользовательского интерфейса в XML. В Android имеется удобный справочник XML-элементов для классов View и их подклассов, например, таких, которые используются для виджетов и макетов;
- Создание экземпляров элементов во время выполнения. Приложение может программным образом создавать объекты View и ViewGroup.

В окне создания приложения можно выбрать один из нескольких вариантов. Для данной ВКР был выбран пустой шаблон, что дает больше гибкости. При желании все необходимые элементы можно добавить самостоятельно в программном коде.

Например, самостоятельно была добавлена навигационная нижняя панель – Navigation Bar из аналогичного шаблона. Эта панель позволяет пользователю переключаться между окнами по нажатию на соответствующие им окна. Для этого в код каждого окна, где была необходима навигационная панель, был добавлен следующий код, прописанный на рисунке 5.1:

```
bottomNavigationView.setOnNavigationItemSelectedListener((item) → {
        switch (item.getItemId()){
            case R.id.bot_game:

                break;

            case R.id.bot_love:
                Intent intent1 = new Intent( packageContext: MainActivity.this, AnalysisActivity.class);
                startActivity(intent1);
                break;

            case R.id.bot_rest:
                Intent intent2 = new Intent( packageContext: MainActivity.this, HistoryActivity.class);
                startActivity(intent2);
                break;
        }
        return false;
});
```

Рис. 5.1. Код Navigation Bar в главном окне

В этом коде обрабатываются нажатия на кнопки навигационной панели. Каждой кнопке соответствует соответствующее окно. Оно запускается с помощью объекта класса Intent и статического метода startActivity().

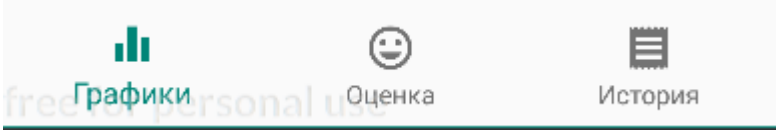

Рис. 5.2. Навигационная панель

На рисунке 5.2 представлен внешний вид навигационной панели. Каждая из трех иконок вызывает соответствующее ей окно.

Представление окон реализуется через специальный компонент – Activity. Activity – отображение операции, предлагаемой пользователю в данный момент работы приложения.

При вызове нового Activity предыдущее Activity ставится на паузу (если разработчик не указал иной вариант в коде). Запущенные Activity хранятся в памяти системы в стеке. С помощью переходов назад (нажатия соответствующей кнопки на телефоне) текущее окно закрывается, удаляется из стека и на экран возвращается предыдущее. Эта последовательность реализует очередность последним вошел – первым вышел (LIFO) [5].

Activity создается в специальном меню Android. Создается класс, расширяющий родительский Activity. Информация о новой операции заносится в файл AndroidManifest.xml. Манифест является одним из основных файлов проекта. Только после прочтения этого файла система начинает выполнять код

приложения. В манифесте также задается имя пакета Java для приложения, которое служит уникальным идентификатором проекта. Все компоненты: службы, операции, приемники и поставщики контента должны быть описаны в Манифесте. В Манифесте также декларируются необходимые разрешения для приложения, если они относятся к защищенным частям API. Помимо этого, в файле Манифеста содержится список подключаемых к проекту библиотек и минимальный уровень API, на котором будет основано приложение.

Activity имеет свой жизненный цикл — начало, когда Android создает экземпляр активности, промежуточное состояние и конец, когда экземпляр уничтожается системой и освобождает ресурсы. Активность может находиться в трех состояниях:

- активная (active или running) — активность находится на переднем плане экрана. Пользователь может взаимодействовать с активным окном;
- приостановленная (paused) — активность потеряла фокус, но все еще видима пользователю. То есть активность находится сверху и частично перекрывает данную активность. Приостановленная активность может быть уничтожена системой в критических ситуациях при нехватке памяти;
- остановленная (stopped) — если данная активность полностью закрыта другой активностью. Она больше не видима пользователю и может быть уничтожена системой, если память необходима для более важного процесса.

Основной метод, участвующий при создании приложения – метод onCreate(). Метод onCreate() вызывается при создании или перезапуска активности. Система может запускать и останавливать текущие окна в зависимости от происходящих событий. Внутри данного метода настраивают статический интерфейс активности. Инициализируют статические данные активности, связывают данные со списками и так далее. Связывают с необходимыми данными и ресурсами. Задают внешний вид

через метод setContentView(). Жизненный цикл Activity показан на рисунке 5.3.

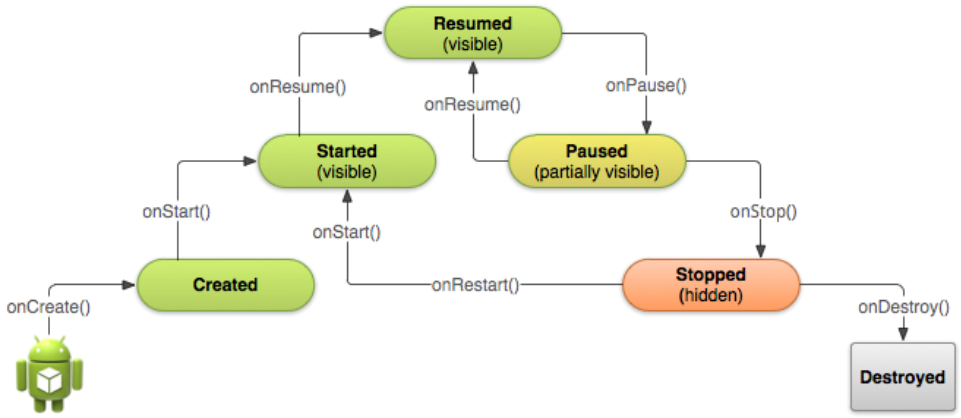

Рис. 5.3 Жизненный цикл Activity

5.2 Пояснения программного кода

5.2.1 Реализация страницы выбора настроений

На экране данной страницы пользователю отображаются пять иконок, соответствующих пяти настроениям на выбор с подписями. Внизу отображается навигационная панель. Внешний вид окна показан на рисунке 5.4.

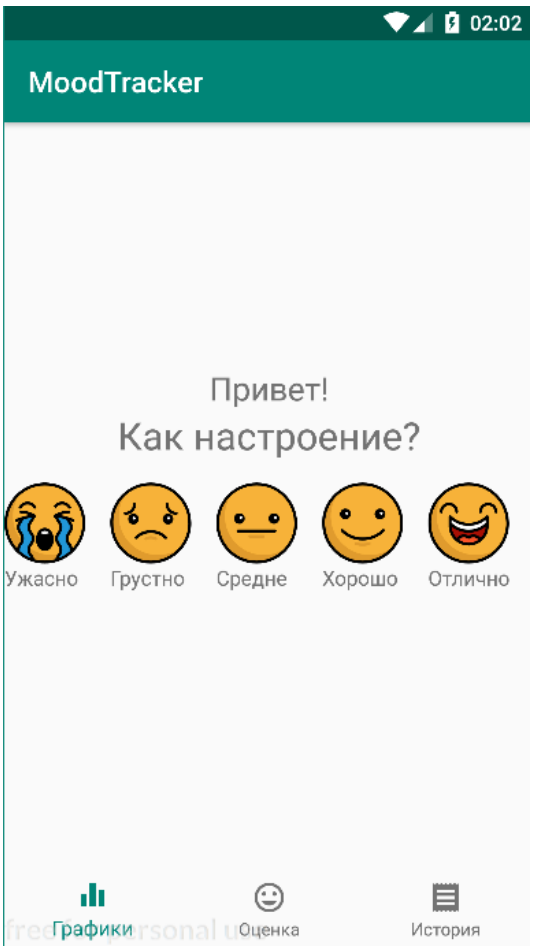

Рис. 5.4. Страница выбора настроений

В методе onCreate() при создании Activity, соответствующей данной странице, вызывается метод класса moodViewModel. Метод служит для получения информации о настроениях из таблицы Mood базы данных.

В архитектуре Android приложений есть важная особенность, которую необходимо учитывать при манипулировании с данными. Ни в коем случае нельзя работать с базой данных из главного потока – потока UI (интерфейса). Это взаимодействие может повлечь за собой длительную задержку из-за большого объема данных, что приведет к зависанию приложения. Если обращаться к базе данных из главного потока, приложение не запустится и выдаст ошибку.

Для обхода этой особенности в языке Java существует многопоточность. Многопоточность создана для того, чтобы перенаправлять более трудоемкие и долгие задачи на отдельные потоки и выполнять несколько задач сразу, асинхронно.

В Android есть несколько возможностей реализовать многопоточность и все они хороши для разных целей. Для работы с базой данных вполне достаточно такого класса асинхронной работы, как AsyncTask [6].

В классе moodViewModel был создан класс-наследник AsyncTask. Ему в качестве аргумента была передана текущая активность. Это необходимо для последующего соединения результатов работы метода с главным интерфейсом приложения. В класс-наследник может содержать три параметра: входные данные, данные, отправляющиеся в главный поток во время исполнения метода, данные, получаемые по итогу выполнения работы методов класса.

Обязательным для реализации методом в наследниках AsyncTask является метод doInBackground(). Этот метод выполняется в новом потоке и не имеет доступа к пользовательскому интерфейсу. Он принимает тот набор параметров, который был задан при объявлении его класса.

В методе doInBackground() в классе moodViewModel выполняется запрос к базе данных на получение всех записанных настроений. Это производится через обращение к объекту класса moodRepository, который содержит в себе аналогичный метод, обращаясь к объекту класса moodDao, в котором прописан сам запрос. После полученный список (List<Mood>) передается внутри наследника AsyncTask в следующий метод – метод onPostUpgrade().

Метод onPostUpgrade(), используя переданный Activity, обращается к окну, из которого был вызван и передает в метод addCustomView() полученный список настроений из базы данных.

В методе addCustomView() создается визуальный интерфейс, в котором полученные данные из базы соединяются с соответствующими иконками и отображаются в заданном порядке.

Нажатия на иконки обрабатываются с помощью метода onItemClick(), который вызывает следующую Activity, запоминая сделанный выбор.

5.2.2 Реализация страницы добавления параметров

В странице добавления параметров реализовано принятие данных из предыдущей Activity с помощью метода getExtras() класса Intent. В результате в верхней части окна отображается сделанных ранее выбор настроения в виде соответствующей иконки. Внешний вид окна показан на рисунке 5.5.

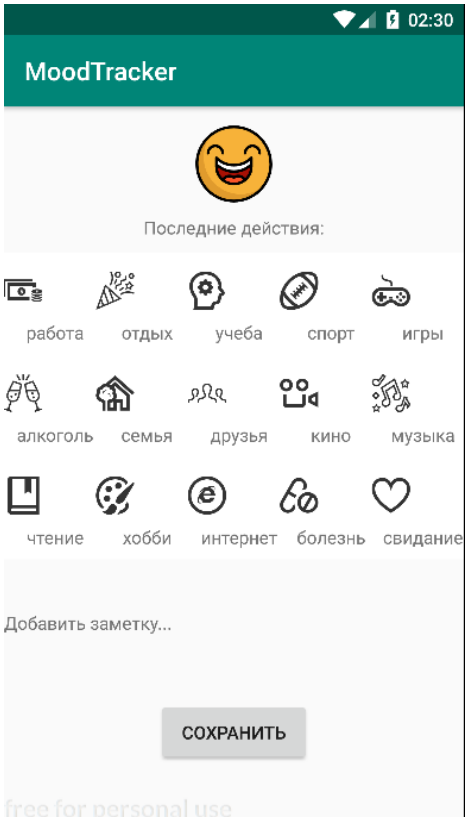

Рис. 5.5 Страница добавления параметров

При инициализации данной Activity в методе onCreate() также происходит заполнение списка доступных действий. Список действий отображается с помощью компонента интерфейса RecyclerView.

RecyclerView – новый компонент интерфейса, пришедший на замену более простым списочным компонентам ListView и GridView и представляет из себя прокручиваемый список.

Одной из проблем предыдущих версий списка была необходимость хранить в памяти сразу все объекты списка. Это могло сильно сказываться на

производительности, ведь в экземплярах списка может хранится немало данных, в том числе и картинки.

Теперь же список хранит только объекты, отображаемые на экране. При прокручивании списка пользователем необходимые данные загружаются из базы и заполняют собой уже созданные ранее экземпляры списка.

Также RecyclerView требует специальной организации классов для более удобной работы с кодом [7].

Например, для создания RecyclerView необходимо добавить этот элемент в layout файл с расширением xml, который хранит в себе информацию об используемых графических компонентах окна.

Далее требуется создать отдельный layout файл для отображения элемента списка. В нем указываются необходимые графические компоненты для элемента. Например, для случая со списком действий файл recycleaction_item.xml содержит в себе ImageView для отображения иконки действия и TextView для подписи.

Помимо этого само заполнение списка происходит в классе адаптера. Адаптер принимает необходимые при заполнении данные (в случае со списком действий – список действий, полученный из базы данных, а также список единиц измерения для подстановки к соответствующим действиям. После реализуется класс ViewHolder.

В объекте класса ViewHolder необходимо реализовать следующие методы:

- getItemCount() – метод, возвращающий общее количество элементов списка. Значения списка передаются с помощью конструктора.
- onCreateViewHolder() создает новый объект ViewHolder всякий раз, когда RecyclerView нуждается в этом. Это тот момент, когда создаётся layout строки списка, передается объекту ViewHolder, и каждый дочерний view-компонент может быть найден и сохранен.

- onBindViewHolder() принимает объект ViewHolder и устанавливает необходимые данные для соответствующей строки во view-компоненте.

В методе onBindViewHolder() программа считывает полученное название действия и название файла иконки и устанавливает их в соответствующие графические компоненты.

Также в методе onBindViewHolder() установлены «слушатели» короткого и долгих нажатий на элемент – setOnClickListener и setOnLongClickListener [8].

Первый слушатель позволяет отметить объект класса Process как выделенный и снять выделение, если объект был нажат повторно.

Второй слушатель открывает диалоговое окно AlertDialog в котором пользователю предлагается установить необходимое количество величин. Единица измерения, соответствующая нажатому элементу загружается из переданного в адаптер списка величин из таблицы Value. При подтверждении выбора иконка действия окрашивается в черный цвет путем установки новой иконки. Если для выбираемого действия нельзя выбрать количественную характеристику, об этом сообщит высплывающее сообщение – Toast.

Ниже списка действий располагается текстовое поле для добавления заметки. При нажатии на него откроется новое Activity. Текст в Activity для редактирования вводится в компоненте EditText. После нажатия кнопки «Сохранить» введенный текст передастся в предыдущее окно через метод onActivityResult() и установится в соответствующий TextView, так как Activity добавления заметки вызывалось с помощью метода startActivityForResult(), что означает инициализацию Activity с ожиданием получения результата.

После нажатия на кнопку «Сохранить» программа устанавливает текущие время и дату с помощью методов getInstance().getTime() класса Calendar. После программа проверяет все объекты класса Process на предмет отмеченности пользователем. Дата добавляется в таблицу DateInfo. Выделенные действия и количественные характеристики добавляются в таблицу Process_Mood вместе с

выбранным id настроения и id даты. А заметки добавляются в таблицу Conditions вместе с id настроения.

5.2.3 Реализация графиков

В приложении представлены два окна, отображающие графики: окно с графиком зависимости настроения от времени и окно с диаграммой зависимости настроения от времени по выбранному фильтру действия. Внешний вид окон показан на рисунке 5.6.

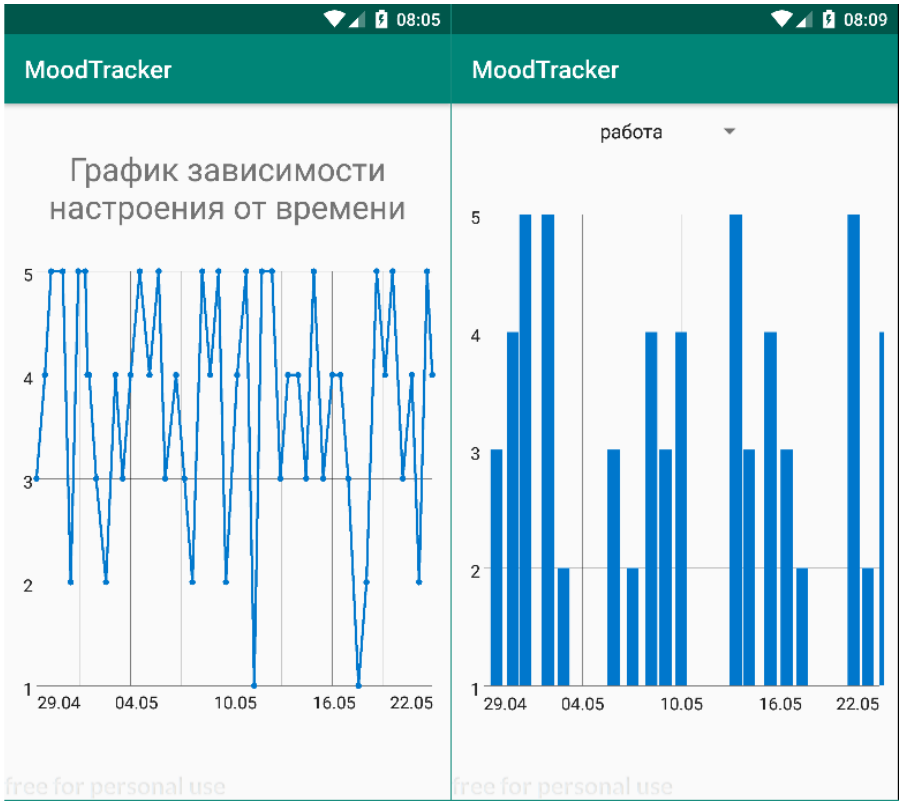

Рис. 5.6. Страницы отображения графиков

Для начала программе необходимо получить нужные данные для отображения. Это делается с помощью описанного выше класса AsyncTask, вызываемого в методе классов ViewModel. Разница между получением данных в первом и втором графиках состоит в том, что в диаграмме с отображением настроений по фильтру действия необходимо получить все записи из таблицы Process_Mood только для конкретного actions_id, соответствующего

выбранному действию. Для этого действия был прописан новый запрос, получающий нужный id и выполняющий по нему выборку в базе данных.

Сами графики были построены с помощью свободно распространяемого виджета GraphView.

Заполнение данных, необходимых для формирования точек графика производилось в цикле перебора всех полученных данных о дате и записях. Переменным с типами BarGraphSeries и LineGraphSeries присваивались данные через метод appendData(). Притом также необходимо было проинициализировать точки *x* и *y*.

Ось *x* отвечает за даты. Поэтому сначала было выполнено форматирование строки, содержащей дату в переменную с типом Date. Это необходимо делать из-за того, что SQLite не позволяет хранить дату в формате Date, и используется тип TEXT (String). Код форматированя показан на рисунке 5.7.

```
for(DateInfo dateInfoItem: dateInfos) {
    try{
    Date dateFormat=new SimpleDateFormat( pattern: "yyyy-MM-dd HH:mm:ss").parse(dateInfoItem.date_add);
        Log.i(TAG,  msg: "все ок");
        DateMood dateMood=new DateMood();
        dateMood.date=dateFormat;
        dateMoods.add(dateMood);}
    catch (ParseException e) {
        Log.i(TAG,  msg: "не ок");
        //Handle exception here, most of the time you will just log it.
        e.printStackTrace();
    }
}
```

Рис. 5.7. Форматирование String в Date

Для уточнения границ оси Y использовались методы setMinY(1), setMaxY(5). Для настройки отображения количества наименований даты в подписях у оси *x* использовался метод setNumHorizontalLabels(5). Для возможности масштабирования и прокрутки графиков были использованы методы setScalable(true) и setScrollable(true).

5.2.4 Реализация списков со статистикой

Для отображения суммы всех количественных характеристик по действиям и средних показателей настроений по действиям были созданы две страницы со списками. Внешний вид окон показан на рисунке 5.8.

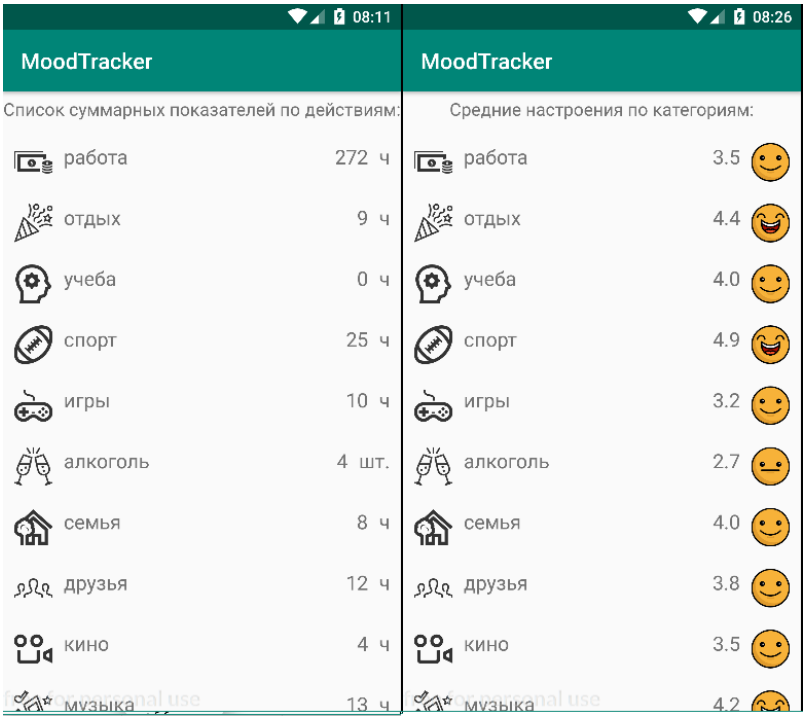

Рис. 5.8. Страницы отображения графиков

Для демонстрации списков был реализован компонент RecyclerView, о котором уже говорилось выше.

Для формирования первого списка было необходимо получить данные сразу из четырех таблиц базы данных. Потребовались данные таблиц Process, Value, Mood, Process_Mood. Поскольку работать с базой данных в Android нужно в отдельном потоке, была реализована последовательная структура выполнения запросов для статистики.

Сначала вызывался метод для получения списка значений из таблицы Value через объект класса ValueViewModel. Затем в методе onPostExecute() полученные действия передавались в метод класса ProcessViewModel для получения списка действий. После отработки этого потока запускался метод

для получения списка записей для каждого действия по отдельности, а после для полной выборки всех записей вместе.

Такая организация потоковой работы с базой позволяет не вмешиваться в работу главного потока с интерфейсом и успешно выполнять необходимые запросы.

В первом списке количественные характеристики получались путем сложения всех количеств по каждому действию в таблице записей Process_Mood.

Во втором списке средние значения настроений были получены как среднее арифметическое всех настроений по каждому действию. Помимо этого всем элементам списка были присвоены иконки настроений, соответствующие полученным средним арифметическим. Если значение получалось в диапазоне от нуля до единицы, присваивалась иконка с «ужасным» настроением и так далее.

5.2.5 Реализация отображения истории

Для просмотра всех записей был добавлен удобный фильтр по дате с возможностью просмотра каждой записи отдельно. Внешний вид окон показан на рисунке 5.9.

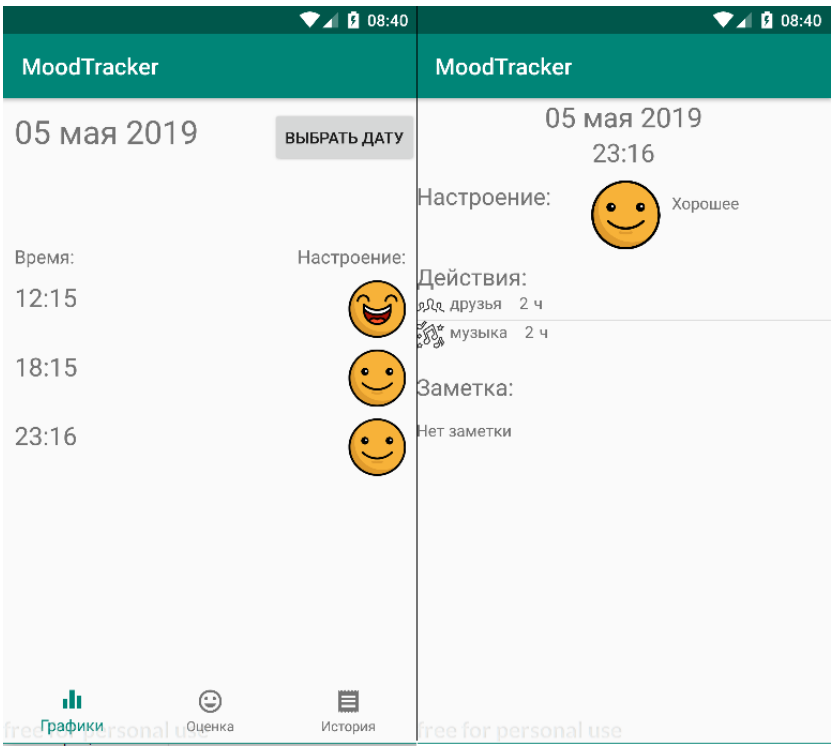

Рис. 5.9. Страницы отображения записей по дате

Для отображения данных при просмотре выбранной записи было задействовано пять таблиц. Манипуляции с ними производились по той же последовательной структуре, как в предыдущем пункте.

Для того, чтобы указать необходимый день, была добавлена кнопка «Выбрать дату», которая вызывает DatePickerDialog.

DatePickerDialog – компонент интерфейса, состоящий из календаря и кнопок отмены и подтверждения выбора. При нажатии кнопки подтверждения было необходимо отобразить выбранную дату в TextView, а также передать в качестве аргумента в объект класса DateInfoViewModel.

Для отображения выбранной даты сначала ее нужно было отформатировать в более понятный и удобный для человеческого восприятия вид. Это было произведено с помощью объектов классов Calendar и DateFormat. Для замены числового значения месяца на его текстовый эквивалент был создан статический метод, который возвращал слова и служил аргументом в методе форматирования SimpleDateFormat() [9].

Так как было нужно получить все записи конкретного дня, то использовалась статичная выбранная дата. Оставалось лишь вывести все записи

по выбранной дате и указать время их создания. Для этого дата форматировалась в формат «год-месяц-день», исключая время. Затем использовалась в качестве аргумента в методе класса DateInfoViewModel, который получал все записи по выбранному дню.

После полученные записи добавляются в список ListView. Нажатие на конкретную позицию в списке провоцирует открытие нового Activity с отображением полной информации по выбранной записи.

В этом окне использовались все таблицы базы данных и доступ к ним осуществлялся последовательно через методы onPostExecute() каждого класса ViewModel.

ЗАКЛЮЧЕНИЕ

В настоящее время активному пользователю смартфона и прочих гаджетов бывает довольно сложно сконцентрировать свое внимание на важных задачах. Помимо этого, гаджеты и распространенность Интернета меняют человеческую психику, навязывая ложные ценности и вводя человека в подавленное состояние.

В качестве одного из методов решения вышеперечисленных проблем психологи рекомендуют проводить анализ своих дней, поведения и настроений для рефлексии и повышения осознанности, что влечет улучшение качества жизни.

Разработанное Android приложение позволяет в любой момент времени сделать отметку о своем состоянии с указанием подробных показателей, а впоследствии на графиках и в списках увидеть всю динамику настроений. Поиск в истории записей по дате позволяет увидеть полный отчет о каждом дне и каждой записи.

Разработка приложения на операционной системе Android позволяет распространить приложение для большего количества людей из-за доминирующего положения на рынке данной операционной системы.

Использование библиотеки Room исключило множество возможных ошибок при работе с базой данных в Android. Многопоточный доступ к данным благодаря AsyncTask позволяет приложению работать гибко и не зависеть от ожидания ответа базы данных.

На момент создания приложения его аналогов по некоторым функциям не было найдено.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ И ЛИТЕРАТУРЫ

1. World Health Organization World health report. [Электронный ресурс] URL: https://www.who.int/whr/2001/media_centre/press_release/en/ (дата обращения: 18.01.2019)
2. Шилдт Г. Java 8. Руководство для начинающих. 6-ое изд.: Изд-во Вильямс, 2015 – 712 с.
3. Блох Д. Java. Эффективное программирование.: Изд-во Лори, 2014. – 310 с.
4. Google Android Room with a View – Java [Электронный ресурс] URL: https://codelabs.developers.google.com/codelabs/android-room-with-a-view/index.html (дата обращения: 03.04.2019)
5. Android Development Documentation. [Электронный ресурс] URL: https://developer.android.com/ (дата обращения: 07.05.2019)
6. Александр К. Android: AsyncTask. [Электронный ресурс] URL: http://developer.alexanderklimov.ru/android/theory/asynctask.php (дата обращения: 10.05.2019)
7. Bill Ph. RecyclerView Part 1: Fundamentals For ListView Experts. [Электронный ресурс] URL: https://www.bignerdranch.com/blog/ recyclerview-part-1-fundamentals-for-listview-experts/ (дата обращения: 13.05.2019)
8. Гамма Э., Хелм Р. Приемы объектно-ориентированного проектирования. Паттерны проектирования.: Изд-во Питер, 2013. – 366 с.
9. Java Platform Standart Edition 8 Documentation. [Электронный ресурс] URL: https://docs.oracle.com/javase/8/docs (дата обращения: 16.05.2019)

ПРИЛОЖЕНИЕ А

Код адаптера для RecyclerView

```
public class ProcessRecAdapter extends
RecyclerView.Adapter<ProcessRecAdapter.MyViewHolder> {
    public int counter = 0;
    public List<Process> mModelList;
    public List<Value> valueList;
    public List<ValueProcess> valueProcessList;
    Context mContext;
    public Adding activity;
    public int pose;
    AlertDialog alert;


    public ProcessRecAdapter(Adding activity, Context context, List<Process>
modelList, List<Value> valueList) {
        mModelList = modelList;
        mContext = context;
        this.activity = activity;
        this.valueList=valueList;
        valueProcessList=new ArrayList<>();
        for (Process item:mModelList) {
            ValueProcess valueProcess= new ValueProcess();
            valueProcess.id=item.id;
            valueProcess.icon=item.icon;
            valueProcess.name=item.name;
            valueProcess.value_id=item.value_id;
            valueProcess.isSelected=item.isSelected;
            valueProcessList.add(valueProcess);
        }
    }

    @Override
    public MyViewHolder onCreateViewHolder(ViewGroup parent, int viewType) {
        View view =
LayoutInflater.from(parent.getContext()).inflate(R.layout.recycleaction_item,
parent, false);
        return new MyViewHolder(view);
    }

    @Override
    public void onBindViewHolder(final MyViewHolder holder, int position) {
        final Process process = mModelList.get(position);
        final ValueProcess valueProcess = valueProcessList.get(position);
        pose = position;
        holder.setData(mModelList.get(position));
        holder.view.setBackgroundColor(process.isSelected() ? Color.CYAN :
Color.WHITE);
        holder.view.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View view) {
                Log.i(TAG, "onBind " + process.isSelected + " " + process.name);
                process.setSelected(!process.isSelected());
                valueProcess.setSelected(!valueProcess.isSelected());
                String imgStr = process.icon;
                int id = mContext.getResources().getIdentifier(imgStr,
"drawable", mContext.getPackageName());
                int idWhite = mContext.getResources().getIdentifier(imgStr,
"drawable", mContext.getPackageName());
```

```
                holder.view.setBackgroundColor(process.isSelected() ? Color.CYAN
: Color.WHITE);
                holder.imageView.setImageResource(process.isSelected() ? idWhite
: id);

            }
        });
        holder.view.setOnLongClickListener(new View.OnLongClickListener() {
            @Override
            public boolean onLongClick(View v) {
                String name="";
                for(Value value:valueList){
                    if(value.id==process.value_id){
                        Log.i(TAG, "значение " +value.id);
                        name=value.name;
                    }
                }
                if (name.equals("none")){
                    Toast toast2 = Toast.makeText(mContext,
                            "Для этого действия нельзя выбрать
продолжительность",
                            LENGTH_SHORT);
                    toast2.setGravity(Gravity.TOP, 0, 0);
                    toast2.show();
                    return true;
                } else{
                final AlertDialog.Builder mBuilder = new
AlertDialog.Builder((activity));
                LayoutInflater inflater = (LayoutInflater)
mContext.getSystemService(Context.LAYOUT_INFLATER_SERVICE);
                View mView = inflater.inflate(R.layout.dialog_amount, null);
                TextView valueText = (TextView)
mView.findViewById(R.id.valueText);

                final EditText alertEdit = (EditText)
mView.findViewById(R.id.countText);
                Button alertMinus = (Button)
mView.findViewById(R.id.buttonMinus);
                Button alertPlus = (Button) mView.findViewById(R.id.buttonPlus);
                Button alertOk = (Button) mView.findViewById(R.id.alert_ok);
                Button alertCancel = (Button)
mView.findViewById(R.id.alert_cancel);

                valueText.setText(name);
                counter=0;
                alertEdit.setText(String.valueOf(counter));
                mBuilder.setView(mView);
                alertMinus.setOnClickListener(new View.OnClickListener() {
                    @Override
                    public void onClick(View v) {
                        if (counter == 0) {
                        } else {
                            counter--;
                            alertEdit.setText(String.valueOf(counter));

                        }
                    }
                });
                alertPlus.setOnClickListener(new View.OnClickListener() {
                    @Override
                    public void onClick(View v) {
                        counter++;
                        alertEdit.setText(String.valueOf(counter));
```

```
                    }
                });
                alertCancel.setOnClickListener(new View.OnClickListener() {
                    @Override
                    public void onClick(View v) {
                        alert.cancel();
                    }

                });
                alertOk.setOnClickListener(new View.OnClickListener() {
                    @Override
                    public void onClick(View v) {
                        valueProcess.setSelected(!valueProcess.isSelected());
                        String imgStr = process.icon;
                        process.setSelected(!process.isSelected());
                        valueProcess.amount =
Integer.parseInt(alertEdit.getText().toString());

valueProcess.value_id=Integer.parseInt(String.valueOf(alertEdit.getText()));

                        if (process.value_id > 0) {
                            int id =
mContext.getResources().getIdentifier(imgStr, "drawable",
mContext.getPackageName());
                            int idBlack =
mContext.getResources().getIdentifier(imgStr + "_black", "drawable",
mContext.getPackageName());
                            holder.view.setBackgroundColor(process.isSelected()
? Color.CYAN : Color.WHITE);

holder.imageView.setImageResource(process.isSelected() ? idBlack : id);
                        }
                        alert.dismiss();
                    }
                });
                alert = mBuilder.create();
                alert.show();
                return true;
            }}
        });

    }

    public List<Process> getmModelList1() {
        List<Process> tempList = new ArrayList<Process>();

        for (int i = 0; i < mModelList.size(); i++) {
            if (mModelList.get(i).isSelected)
                tempList.add(mModelList.get(i));
        }
        return tempList;

    }
    public List<ValueProcess> getmModelList() {
        List<ValueProcess> tempList = new ArrayList<ValueProcess>();

        for (int i = 0; i < valueProcessList.size(); i++) {
            if (valueProcessList.get(i).isSelected)
                tempList.add(valueProcessList.get(i));
        }
        return tempList;
    }
    }
```

```
    @Override
    public int getItemCount() {
        return mModelList == null ? 0 : mModelList.size();
    }

    public class MyViewHolder extends RecyclerView.ViewHolder {

        public View view;
        private TextView textView;
        public ImageView imageView;
        public Process item;

        private MyViewHolder(View itemView) {
            super(itemView);
            view = itemView;
            textView = (TextView) itemView.findViewById(R.id.action_text);
            imageView = (ImageView) itemView.findViewById(R.id.action_image);

        }


        public void setData(Process item) {
            this.item = item;
            String imgStr = item.icon;
            int id = mContext.getResources().getIdentifier(imgStr, "drawable",
mContext.getPackageName());
            textView.setText(item.name);
            imageView.setImageResource(id);

        }

    }
}
```