Рекомендации

Чтобы помочь в дальнейшем развитии в разработке под андроид, мы собрали основные рекомендации, что стоит изучать, какие библиотеки нужно знать и где все это можно найти.

Во-первых, очень важно понимание того, как стоит разрабатывать дизайн вашего приложения в лучших традициях UX. Это касается не столько разработки конкретно дизайна (что в общем-то задача дизайнеров), сколько умения реализовывать приложения так, чтобы выглядели отлично на любом экране и в любой ориентации и были удобны в использовании. Хорошей начальной точкой в изучении этой тематики является курс на Udacity по Material Design. К тому же у гугл есть очень хороший гайд по всем принципам Material Design, который стоит прочитать и применять на практике.

Во-вторых, к сожалению, разработка под андроид с голым сдк без различных библиотек - это крайне печальное и нелегкое занятие. Есть определенный набор стандартных библиотек, которые значительно упрощают создание приложений:

- support-v7, support-design и другие библиотеки поддержки
- OkHttp + Retrofit для работы с сетью
- И различные библиотеки для работы с базами данных: Realm, greenDAO, SQLBright.

Эти библиотеки помогут в реализации дизайна согласно принципам Material Design и в упрощении работы с сетью (очень сильно помогут) - так что стоит их изучить и применять. Конечно, при этом важно знать и то, на чем они основаны, то есть как разрабатывать на голом сдк, тоже надо понимать.

И в-третьих, чтобы успешно развиваться и дальше, вам нужно две вещи:

1. Много практики - нужно придумывать идеи различных приложений и реализовывать их. При этом нужно стараться реализовывать их разными способами, с использованием разных технологий, библиотек и фреймворков. Так вы наберете необходимый опыт, что поможет вам дальше изучать более сложные вещи и быстрее решать возникающие проблемы.
2. Много знаний и постоянное развитие - система Android непрерывно развивается и с каждым днем становится все больших новых технологий и сложностей. Поэтому нужно хорошо знать и изучать основы и при этом всегда следить за новостями из мира Android.

a. Хорошо знайте Java core - коллекции, многопоточность, рефлексия и другие элементы.
b. Изучите основные компоненты Android - Activity, Fragment, Application, Service, BroadcastReceiver.
c. Разберитесь, как устроена система работы, как работает ее виртуальная машина, какова работа с памятью.
d. Изучите различные View и ViewGroup, чтобы уметь лучше строить интерфейс пользователя: от LinearLayout до ConstraintLayout.
e. Разберитесь с проблемой обработки поворота экрана и со способами борьбы с ней.
f. Попробуйте фреймворк RxJava, который приносит новую модель работы с многопоточностью и данным, а новая модель работы - это всегда развитие и новый опыт.
g. Изучайте паттерны проектирования, архитектурные паттерны и код других разработчиков.
h. Следите за новостями из мира Android на самых популярных ресурсах: Android Developers Blog, Medium, AndroidWeekly, AndroidArsenal.

Конечно, не получится описать огромный путь Android-разработчика на пару страниц, но мы надеемся, что эти простые советы помогут вам в дальнейшем изучении. Кроме того, следите за нашими новостями, ведь после курса мы поделимся всеми ресурсами с него, а они будут куда обширнее, чем эти простые рекомендации.

Недочеты

Мы также собрали различные недочеты как в плане дизайна приложения, так и в плане кода, которые встречались в большом количестве присланных тестовых заданий. Большинство из этих недочетов не являются ошибками, но они либо ухудшают UX приложения, либо являются неоптимальными в плане решения с точки зрения кода. Вы могли избежать всех этих недочетов в своем приложении, но с этим списком все равно стоит ознакомиться, так вы будете знать, с чем вы потенциально можете встретиться в будущем.

Улучшения в UI / UX:

1) На экране списка городов обычно отображается лишь малая часть информации о погоде в данном городе (например, только название города

и температура), и очень часто для списка городов использовались карточки (CardView), что в данном случае не оправдано, лучше подойдет обычный список с разделителями. Понимать различия и что когда применять - очень важно, поэтому прочитать гайды по Material Design будет очень полезно https://material.google.com/components/cards.html#cards-usage

2) Список не во всю ширину экрана, отступ списка от верхнего и нижнего края. Этот недочет также был весьма частым. Суть заключается в том, что есть экран со списком и нужно задать отступы от краев экрана для элементов списка. Это правильное желание, но обычно возникает проблема в реализации в том, что задается отступ (margin) у самого списка. При этом при появлении эффекта при скролле (так называемый Edge Effect), отчетливо видно такой отступ от краев. В случае, когда список логически занимает всю ширину / высоту экрана, это выглядит очень неудачно. Чтобы избавиться от этого, но в то же время оставить отступы, нужно использовать отступы для самих элементов списка (layout_marginLeft и layout_marginRight для боковых отступов). В таком случае RecyclerView будет занимать всю ширину экрана, и Edge Effect будет выглядеть лучше. Аналогично и по высоте, с небольшим отличием: у RecyclerView нужно задать paddingTop и paddingBottom, но чтобы сохранить отступы в нормальном виде, нужно добавить clipToPadding=false.
3) Также распространенная ошибка - это большое количество свободного места, когда либо на экране слишком мало контента, либо он прижат к какому-то краю. Это значит не то, что нужно стремиться забивать экран различным контентом (что нехорошо, так как при большом количестве различных элементов пользователю становится тяжело ориентироваться на экране), а то, что нужно стремиться грамотно и гармонично располагать элементы на экране.
4) Поддерживайте навигацию на тех устройствах, на которых нет аппаратной кнопки back. Стрелка в тулбаре, по нажатию на которую, экран закрывается и пользователь возвращается на предыдущий экран - стандартный и очень удобный паттерн.
5) Уведомляйте пользователя об ошибках. Если по каким-то причинам не удалось получить данные с сервера, скажите об этом пользователю. Это может быть диалог, может быть тоаст, может быть какой-то текст на экране. Но никогда не оставляйте пустой экран, в таком случае пользователь просто не поймет, что он не делает не так. Неплохим вариантом является заглушка с текстом (например, “Не удалось получить данные. Проверьте ваше подключение и повторите попытку) и кнопкой обновить, по нажатию на которую запрос повторяется.

Улучшения по коду:

1) Используйте RecyclerView вместо ListView. Да, тут есть определенные спорные моменты. ListView в определенном смысле проще в использовании различных фич по типу разделителей, листенера на выбор элемента. Но RecyclerView намного мощнее, он предоставляет гораздо более широкие возможности кастомизации, начиная с различных анимаций (которые сделать на ListView было весьма нетривиальной задачей) и заканчивая возможностью произвольного расположения элементов в нем (LayoutManager). К тому же разработчику не нужно вручную реализовывать паттерн ViewHolder. Общие отличия можно почитать в этом ответе. Благодаря всем этим фичам RecyclerView отдается предпочтение сегодня всегда. И есть еще один очень важный момент. ListView - это класс из системы Android. Что означает, что как только вышла новая версия SDK - реализация этого класса фиксирована. Разработчик ничего не сможет сделать с багами, которые могут оказаться в этом системном классе. RecyclerView - это класс из библиотеки поддержки. Если в нем будет найден баг, то его исправят в следующем релизе, и разработчик, просто обновив библиотеку, избавится от этого бага.
2) Не используйте конкатенацию строк из ресурсов, используйте форматирование. Пример. Также по этому поводу есть неплохая статья на хабре.
3) Реализация собственного SQLiteOpenHelper или ContentProvider - это очень полезно для развития, но для упрощения работы стоит использовать современные библиотеки для работы с базами данных, к примеру, Realm, greenDAO, SQLBright.
4) Для загрузки изображений лучше использовать библиотеки, например, Picasso или Glide. Они поддерживают загрузку изображений, трансформации, кэширование и прочие полезные вещи.
5) Выполнять запросы прямо в UI классах (Activity / Fragment) с использованием Thread или AsyncTask - это не лучшее решение. Такой подход чреват большим количеством ошибок, начиная от повторных запросов при поворотах экрана, заканчивая крашами и утечками памяти. Одна из достаточно простых и эффективных альтернатив - это лоадеры.
6) Парсинг JSON с помощью встроенных средств Android (а именно через классы JSONObject и JSONArray) весьма неудобен и не придает красоты и удобства вашему коду. К тому же он не всегда эффективен. Стоит

использовать известные библиотеки для решения этой задачи: Gson, Jackson. Сравнение можно посмотреть в статье.

7) Для тех, кто использовал RxJava:
    a) Для сетевых запросов и работы с базами данных стоит использовать Scheduler io, а не newThread. Про различие Schedulers http://stackoverflow.com/questions/31276164/rxjava-schedulers-use-cases
    b) RxJava предоставляет очень широкие возможности для управления данными и многопоточностью. Достаточно часто в тестовых заданиях встречался код, в котором долгие операции работы с файлами или базами данных осуществлялась при получении в подписчике, то есть в главном потоке. При этом переместить эти операции в фоновый поток очень просто, для этого можно использовать к примеру операцию doOnNext или же любой оператор, например, flatMap:

```
weathersObservable
    .flatMap(forecasts -> {
        //save to database
        return Observable.just(forecasts);
    })
    .subscribeOn(Schedulers.io())
    .observeOn(AndroidSchedulers.mainThread());
```

8) Старайтесь не выполнять достаточно долгие операции, такие работа с файлами или базой данных в главном потоке. Это связано с тем, что в идеале система Android должна перерисовывать экран каждые 16мс, то есть достигать 60fps. Конечно, это не обязательно, есть советы, что незаметными для пользователя останутся задержки вплоть до 64мс, но все равно нужно стремиться не допускать этого и выносить “тяжелую” работу в отдельные потоки. Это особенно критично для динамичного контента (к примеру, анимации).
9) Не игнорируйте предупреждения от Android Studio, обычно это очень правильные и грамотные предупреждения. Если их исправлять, то код станет чище, понятнее и будет содержать меньше потенциальных ошибок.