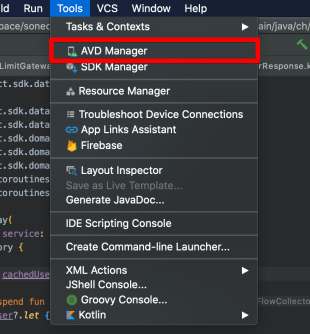
12 Опишите процесс создания и настройки виртуального устройства, укажите основные особенности

## Что такое AVD в Android?

AVD расшифровывается как Android Virtual Device, а именно виртуальное Андроид устройство. На вашем компьютере вы открываете заранее выбранную и сконфигурированную систему. Это удобно даже когда у вас есть реальный телефон. В эмуляторе можно изменить версию ОС, размер экрана и другие параметры. Это позволяет протестировать то, как работает ваше приложение на разных версиях Android.

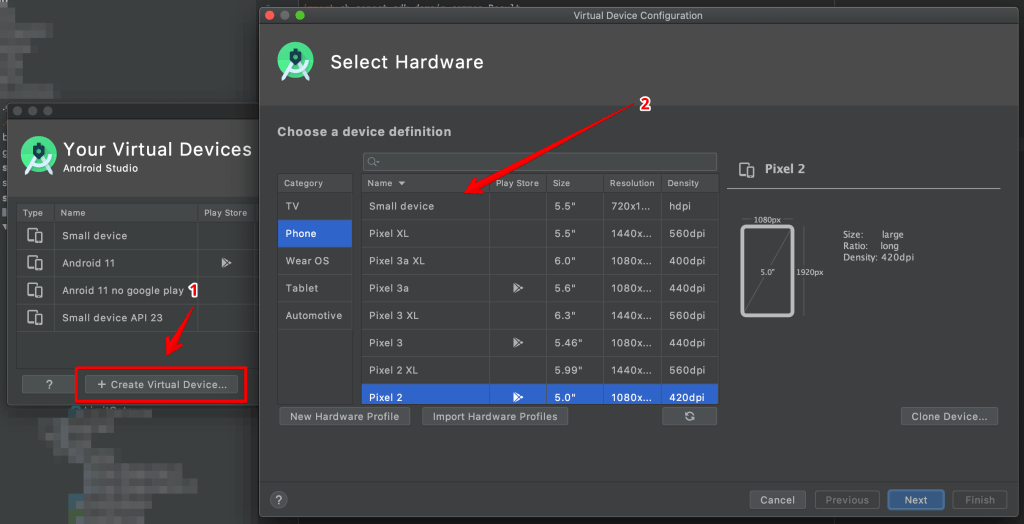
## AVD Manager

В Android Studio всеми виртуальными устройствами управляет AVD manager. Его можно открыть через меню Tools в Android studio.

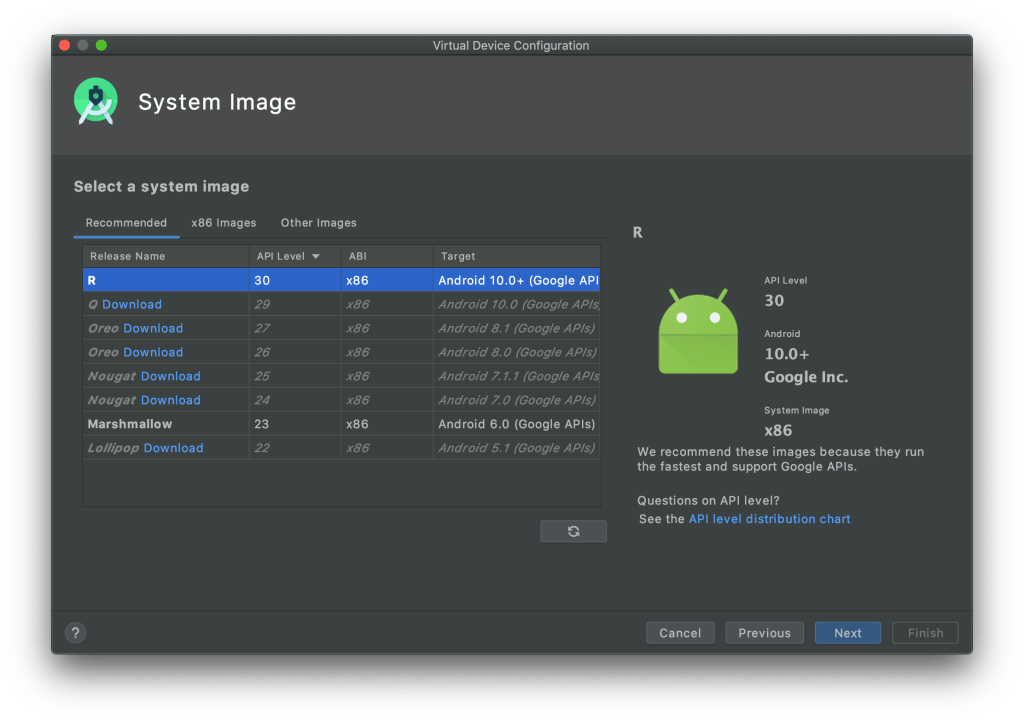
AVD Manager в Android Studio

### Создание нового Андроид эмулятора

В AVD Manager нужно нажать на кнопку создания нового виртуального устройства и выбрать предустановленный шаблон или же создать свой новый. Давайте выберем Pixel XL.

Создание нового AVD в Android Studio

Следующим пунктом необходимо выбрать образ операционной системы. Это важный шаг, т.к. именно от него будет зависеть как версия Android будет установлена на новом виртуальном устройстве.

Выбор образа для Android AVD

### Управление настройками и запуск эмулятора

Когда все будет готово останется лишь запустить созданный эмулятор Android устройства. Делается это в уже знакомом AVD Manager. Также в нем вы можете найти возможные настройки и пункты управления девайсом.

Настройка Android эмулятора

Вот список возможных опций и что они означают:

| **Название опции** | **Что она означает** |
| --- | --- |
| Duplicate | Дублировать уже созданный эмулятор |
| Wipe Data | Очистить данные устройства. Полезно, если нужен «чистый» девайса с теми же настройками |
| Cold Boot Now | Холодный старт. Бывает, что эмулятор, например, завис. Это способ его перезагрузить |
| Show On Disk | Показать папку, в которой хранятся все файлы, относящиеся к эмулятору |
| View Details | Посмотреть все настройки и конфигурации устройства |
| Delete | Удалить AVD |
| Stop | Остановить уже запущенный эмулятор |

Опции управления эмулятором в AVD Manager

52 Перечислите методы класса BroadcastReceiver, укажите особенности работы с названным классом

Иногда для правильной работы приложению следует знать о текущем заряде батареи или наличии интернета. Если делать проверку самостоятельно на постоянной основе, то на систему идёт большая нагрузка. Но в этом нет нужды, система сама следит за подобными вещами и готова поделиться со всеми нуждающимися своими сообщениями. Ваша задача - реализовать у себя широковещательный приёмник (Broadcast Receiver). Получатель трансляции всегда будет получать уведомления, независимо от состояния вашего приложения - работает ли ваше приложение в фоновом режиме или вообще не работает в данный момент.

Широковещательные сообщения делают приложение более открытым; передавая события, использующие сообщения, вы открываете компоненты своего приложения для сторонних приложений, и сторонние разработчики реагируют на события без необходимости изменять ваше оригинальное приложение. В своём приложении вы можете прослушивать широковещательные сообщения других приложений, заменить или улучшить функциональность собственного (или стороннего) приложения или реагировать на системные изменения и события приложений.

Вы тоже можете организовать отправку широковещательных сообщений, а другие приложения могут обзавестись приёмниками для обработки ваших сообщений.

Приёмник широковещательных сообщений — это компонент для получения внешних событий и реакции на них. Инициализировать передачи могут:

* другие приложения или службы
* сама система
* ваше собственное приложение

Широковещательные сообщения можно разделить на две группы:

* Неявная широковещательная трансляция (implicit broadcast) — сообщения рассылаются всем желающим, а не конкретно вашему приложению. Вам нужно только зарегистрироваться для получения этих сообщений через фильтр **IntentFilter** в манифесте. Система просматривает все объявленные фильтры намерений в вашем манифесте и проверяет, есть ли совпадение. Из-за этого поведения неявные широковещательные сообщения не имеют целевого атрибута
* Явная трансляция (explicit broadcast) предназначена для конкретных приложений. В атрибуте **target** указывают имя пакета приложения или имя класса компонента, по которому можно найти получателя

Класс **BroadcastReceiver** является базовым для класса, в котором должны происходить получение и обработка сообщений, посылаемых клиентским приложением с помощью вызова метода **sendBroadcast()**.

Зарегистрировать экземпляр класса **BroadcastReceiver** можно динамически в коде или статически в манифесте.

Для статической регистрации в файле манифеста в секции **application** следует создать секцию **receiver** и указать класс приёмника. Атрибут **android:exported="true"** сообщает получателю, что он может принимать широковещательные сообщения вне области приложения. Внутри секции указывается фильтр намерений в виде строки, чтобы определить, какие сообщения приёмник должен прослушивать.

<receiver

android:name=".YourReceiver"

android:enabled="true"

android:exported="true" >

<intent-filter>

<action android:name="ru.alexanderklimov.action.CAT" />

</intent-filter>

</receiver>

Если регистрация была сделана через манифест, приложение не обязано работать, чтобы ваш приёмник среагировал на трансляцию намерения. Приложение запустится автоматически, когда подходящие намерение будет транслировано. Система сама сканирует содержимое манифеста всех приложений и делает за нас всю работу. Это хорошее решение, позволяющее экономить ресурсы. Такой подход позволяет создавать приложения, способные реагировать на события даже после завершения или принудительного завершения.

Динамическая регистрация происходит с помощью метода **Context.registerReceiver()**.

this.registerReceiver(mTimeBroadCastReceiver, new IntentFilter(

"android.intent.action.TIME\_TICK"));

Перед этим создаётся класс, расширяющий базовый класс **BroadcastReceiver** и реализуется метод обратного вызова **onReceive()** обработчика событий.

public class TimeBroadcastReceiver extends BroadcastReceiver {

public TimeBroadcastReceiver() {

}

@Override

public void onReceive(Context context, Intent intent) {

...

}

}

Метод **onReceive()** будет выполнен при получении широковещательного намерения, если полученное намерение соответствует фильтру. Приложения с зарегистрированными приёмниками широковещательных намерений будут запущены автоматически при получении соответствующего намерения. Метод должен быть завершён в течение пяти секунд, иначе появится диалоговое окно о принудительном закрытии.

Когда широковещательное сообщение прибывает для получателя сообщения, Android вызывает его методом **onReceive()** и передаёт в него объект **Intent**, содержащий сообщение. Приёмник широковещательных сообщений является активным только во время выполнения этого метода. Процесс, который в настоящее время выполняет **BroadcastReceiver**, т. е. выполняющийся в настоящее время код в методе обратного вызова **onReceive()**, как полагает система, является приоритетным процессом и будет сохранён, кроме случаев критического недостатка памяти в системе.

Когда программа возвращается из метода **onReceive()**, приёмник становится неактивным и система полагает, что работа объекта **BroadcastReceiver** закончена. Процесс с активным широковещательным получателем защищён от уничтожения системой. Однако процесс, содержащий неактивные компоненты, может быть уничтожен системой в любое время, когда память, которую он потребляет, будет необходима другим процессам.

Это представляет проблему, когда ответ на широковещательное сообщение занимает длительное время. Если метод **onReceive()** порождает отдельный поток, а затем возвращает управление, то полный процесс, включая и порождённый поток, система Android считает неактивным (если другие компоненты приложения не активны в процессе), и считает этот процесс кандидатом на уничтожение.

В частности, вы не можете отобразить диалог или осуществить связывание со службой внутри экземпляра **BroadcastReceiver**. Для первого случая необходимо вместо этого использовать методы класса **NotificationManager**. Во втором случае можно использовать вызов метода **Context.startService()**, чтобы послать команду для запуска службы.

Решение этой проблемы возможно, если запустить в методе **onReceive()** отдельную службу вместе с **BroadcastReceiver** и позволить службе выполнять задание, чтобы сохранить содержание процесса активным в течение всего времени вашей операции.

Также можно зарегистрировать широковещательный приёмник не через манифест, а программно. Приёмник, зарегистрированный таким способом, будет отвечать на поступающие намерения только в том случае, если компонент приложения, которому он принадлежит, выполняется в этот момент.

Это может быть полезным, когда приёмник используется для обновления элементов пользовательского интерфейса внутри активности, запуска служб или уведомления через **NotificationManager**. В таких случаях вы можете отменять регистрацию широковещательного приёмника, если активность не отображается на экране или находится в неактивном состоянии.

В коде программы можете написать приблизительно такой код (обычно используют метод **onResume()**):

// Создаём и регистрируем широковещательный приёмник

IntentFilter filter = new IntentFilter(NEW\_CAT\_DETECTED);

CatDetectedBroadcastReceiver receiver = new CatDetectedBroadcastReceiver(receiver, filter);

Для отмены регистрации используется метод **unregisterReceiver()** в контексте приложения, передавая ему в качестве параметра экземпляр широковещательного приёмника (обычно в методе **onPause()**):

unregisterReceiver(receiver);

Для объекта **BroadcastReceiver** нет никаких возможностей видеть или фиксировать намерения, используемые в методе **startActivity()**. Аналогично, когда вы передали намерение для запуска активности через объект **BroadcastReceiver**, вы не сможете найти или запустить требуемую активность. Эти две операции семантически полностью различаются: запуск активности через намерение является приоритетной операцией для системы, изменяющей содержимое экрана устройства, с которым в настоящее время взаимодействует пользователь. Передача широковещательных сообщений для системы является фоновой работой, о которой обычно не знает пользователь и которая, соответственно, имеет более низкий приоритет.

## **Приёмники системных событий**

Android использует широковещательные сообщения для системных событий, таких как уровень зарядки батареи, сетевые подключения, входящие звонки, изменения часового пояса, состояние подключения данных, входящие сообщения SMS или обращения по телефону. Вы можете использовать эти сообщения, чтобы добавлять к вашим собственным проектам новые функциональные возможности, основанные на системных событиях.

Следующий список показывает некоторые из встроенных действий, представленных как константы в классе **Intent**, которые используются для того, чтобы проследить изменения состояния устройства:

* **ACTION\_BOOT\_COMPLETED** — передаётся один раз, когда устройство завершило свою загрузку. Требует разрешения **RECEIVE\_BOOT\_COMPLETED**
* **ACTION\_CAMERA\_BUTTON** — передаётся при нажатии пользователем клавиши **Camera**
* **ACTION\_DATE\_CHANGED** и **ACTION\_TIME\_CHANGED** - запускаются при изменении даты или времени на устройстве вручную пользователем
* **ACTION\_SCREEN\_OFF** и **ACTiON\_SCREEN\_ON** — передаются, когда экран выключается или включается
* **ACTION\_TIMEZONE\_CHANGED** — передаётся при изменении текущего часового пояса

## **Типы трансляций**

Есть три способа отправки трансляций

* Порядковые сообщения о намерениях (Ordered broadcasts), которые посылаются методом **Context.sendOrderedBroadcast()**. Эти сообщения посылаются только одному получателю за один раз. Поскольку каждое полученное сообщение выполняется по очереди, он может в случае необходимости полностью прервать сообщение, чтобы его не успели передать другим приёмникам. Приёмниками сообщений можно управлять с помощью атрибута **android:priority** фильтра сообщений; приёмники сообщений, имеющие одинаковый приоритет, будут выполнены в произвольном порядке.
* Нормальные сообщения о намерениях (Normal broadcasts) — посылаемые вызовом метода **context.sendBroadcast()** и являющиеся полностью асинхронными. Все широковещательные приёмники получают сообщение и не зависят друг от друга. Это более эффективно, но означает, что получатели не могут использовать результат или прервать сообщение;
* Метод **LocalBroadcastManager.sendBroadcast()** отправляет широковещательные сообщения только получателям, определённым в вашем приложении, и не выходит за рамки вашего приложения

Если важно, чтобы приёмники получали намерения в определённом порядке или могли влиять на транслируемое намерение, можно использовать метод **sendOrderedBroadcast()**:

String requiredPermission = "ru.alexanderklimov.MY\_BROADCAST\_PERMISSION";

sendOrderedBroadcast(intent, requiredPermission);

С помощью этого метода ваше намерение дойдёт до всех зарегистрированных приёмников, обладающих необходимым доступом (если он был указан), в порядке их приоритета. Приоритет задаётся с помощью атрибута **android:priority** внутри узла фильтра намерений в манифесте. Чем больше значение, тем выше приоритет.

Производить упорядоченные трансляции с использованием приоритетов рекомендуется только для тех приёмников, которым необходим конкретный порядок приёма сообщений.

## **Ограничения в Android 8.0 Oreo (API 26)**

Часть неявных (implicit) широковещательных сообщений, задаваемых через манифест, запретили. Теперь нужно регистрировать их программно. Существует список исключений, которые будут работать как прежде без ограничений.

В качестве замены для некоторых случаев подойдёт **JobScheduler**.

Примерами устаревшего способа работы с приёмником является android.net.conn.CONNECTIVITY\_CHANGE, ACTION\_POWER\_CONNECTED и др.

## **Ограничения в Android 9.0 Pie (API 28)**

Стало доступно меньше информации, получаемой при трансляции системы **Wi-Fi** и **Network\_State\_Changed\_Action**.

65 Опишите процесс создания геолокации без использования карт.

Прежде всего, нам нужно API для доступа к информации о геолокации. Мне известно два:

* Google Geolocation API — удобный API, но требуется подключенная платёжная учётная запись для вашего проекта;
* OpenCellID от Unwiredlabs — простой в использовании API, крупнейшая в мире открытая база данных вышек сотовой связи.

Перейдите по ссылке выше, зарегистрируйтесь и получите API-ключ. И почитайте документацию, там всё подробно описано.

### **Как же оно всё работает?**

Ваше приложение должно сделать запрос к API, отправив такие данные:

{

"token": "Your\_API\_Token",

"radio": "gsm",

"mcc": 310,

"mnc": 404,

"cells": [{

"lac": 7033,

"cid": 17811

}],

"address": 1

}

Описание полей для ясности:

* token — ваш API-ключ;
* radio — тип сети, например GSM, LTE и пр.;
* mcc — Mobile Country Code, уникальный идентификатор сотового оператора;
* mnc — Mobile Network Code, код мобильной сети. В комбинации с MCC является уникальным идентификатором сотового оператора;
* lac — Location Area Code;
* cid — Cell ID.

Ответ вы получите в таком формате:

{

"status": "ok",

"balance": 100,

"lat": 39.573726,

"lon": -105.005387,

"accuracy": 730,

"address": "1289-1425 West Mineral Avenue, Littleton, CO 80120, USA"

}

То, что нужно. Приступим к реализации.

### **Реализация**

Запустите Android Studio. Создайте новый проект или клонируйте/импортируйте этот.

#### Создаём модель запроса

Она будет использоваться для запроса к API:

#### Создаём модель ответа

Она будет использоваться при получении ответа от API:

#### Создаём сервис

Он используется для взаимодействия с API:

В приложении мы использовали Retrofit. Полную реализацию здесь я приводить не буду, но исходный код доступен. Мы реализовали ViewModel и Activity, которые будут взаимодействовать с API.

**Важно! Для доступа к сетевой информации на устройстве всё же необходимо запросить соответствующее разрешение:**

<**uses-permission** android:name="android.permission.ACCESS\_FINE\_LOCATION" />

#### Получаем сведения о сети

Мы создали метод getCurrentCellInfo(), который будет извлекать данные:

Для получения сведений о сети нам нужно проверить, является ли тип сети GSM, CDMA или LTE. Для этого создаём метод getCellInfo() для каждого типа сети.

Ниже код для сети типа GSM:

Этот же метод необходимо реализовать для каждого типа сети.

На этом реализация киллер-фичи приложения закончена. Как видите, мы не использовали ни GPS, ни службу определения местоположения. Полученные данные мы отправляем к API, а в ответ сразу получаем информацию о местоположении устройства.

76 Охарактеризуйте способы организации безопасности в мобильных приложениях

При разработке мобильного приложения следует учитывать, что данные, которыми оперирует это приложение, могут представлять определенный интерес для третьих лиц. Степень ценности этих данных варьируется в широких пределах, тем не менее, даже наиболее простая приватная информация, например, пароль входа в приложение, требует проработки ее защиты. Особенно это важно в свете распространения мобильных приложений на все сферы электронных услуг, включая финансовые, банковские операции, хранение и передачу личных данных и так далее. Всем интересующимся — добро пожаловать под кат.

Все нижесказанное — исключительно мой опыт, безусловно, данные могут быть неточными, поэтому буду благодарен за любые поправки и дополнения к статье. Я не нашел исчерпывающих статей в сети на подобную тематику, которые бы собирали всю нужную (хотя бы базовую) информацию в одном месте, поэтому решил обобщить свой опыт в этой области на текущий момент времени.

## **Защита мобильного приложения**

Основные виды атак на мобильное приложение:

* Декомпиляция файла приложения (.ipa-файлы для Apple iOS и .apk-файлы для Google Android) и разбор локально сохраненных данных. Защита этого, наиболее важного в настоящее время, уровня целиком лежит на плечах мобильного разработчика.
* Перехват данных, передаваемых по сети (MITM-атаки). Большинство мобильных приложений являются клиент-серверными, следовательно, постоянно передают и принимают большие объемы информации. И хотя современная мобильная и веб-разработка активно завершают переход на HTTPS-протокол общения, тем не менее, не стоит полагаться на единственный рубеж защиты в виде защищенного канала связи.
* Рутование устройства и атака на приложение и применяемые в нем алгоритмы через внешние отладочные инструменты.

### **Перечень основных уязвимостей приложений**

Рассмотрим уязвимости общего характера, без привязки к конкретной платформе. Здесь и далее используется аббревиатура КВД — критически важные данные пользователей. К КВД относятся любые данные, которые не должны быть доступны третьей стороне, это касается как персональных данных пользователя (дата рождения, адрес проживания, личная переписка), так и его приватных данных (пароли, данные кредитных карт, номера банковских счетов, номера заказов и так далее).

Перечень основных уязвимостей следующий:

* Использование незащищенных локальных хранилищ.

* + **Опасность:** Очень высокая.
  + **Комментарий:** Встречается повсеместно, выражается в хранении КВД в незащищенных или слабо защищенных локальных хранилищах, специфических для конкретной платформы. Вскрытие третьей стороной — элементарное, и, как правило, не требуется наличие специальных навыков у атакующего.
  + **Защита:** Хранить КВД можно только в защищенных хранилищах платформы.
* Хранение КВД в коде.

* + **Опасность:** Высокая.
  + **Комментарий:** Уязвимость касается хранения КВД внутри кода (в статических константных строках, в ресурсах приложения и т.п.). Яркие примеры: хранение соли для пароля (password salt) в константе или макросе, которая применяется по всему коду для шифрования паролей; хранение приватного ключа для асимметричных алгоритмов; хранение паролей и логинов для серверных узлов или баз данных. Легко вскрывается третьей стороной при наличии базовых навыков декомпиляции.
  + **Защита:** Не хранить никакие КВД в коде или ресурсах приложения.
* Применение алгоритмов с хранением приватного ключа.

* + **Опасность:** Высокая.
  + **Комментарий:** Уязвимость актуальна в случае, если приватная информация алгоритма (приватный ключ) вынужденно сохраняется в коде или ресурсах мобильного приложения (чаще всего так и бывает). Легко вскрывается методом декомпиляции.
  + **Защита:** В мобильной разработке желательно применять только современные симметричные алгоритмы с генерируемым случайным одноразовым ключом, обладающие высокой стойкостью с взлому методом грубой силы, либо выводить асимметричный приватный ключ за пределы приложения, либо персонализировать этот ключ (как пример — приватным ключом может выступать пользовательский код входа, сохраненный в зашифрованном виде в защищенном хранилище операционной системы).
* Использование асимметричного алгоритма с приватным ключом, известным серверу.

* + **Опасность:** Зависит от степени защищенности сервера.
  + **Комментарий:** Уязвимость носит двойной характер. Хранение приватного ключа допускает возможность расшифровки пользовательских данных на стороне сервера. Во-первых, это некорректно с точки зрения безопасности (если сервер будет взломан — атакующий также получит доступ к приватным данным пользователей), а во-вторых, это нарушает приватность персональных данных. Пользователь всегда должен быть уверен, что его персональная информация не известна никому, кроме него самого (только если он явно не дал разрешение на ее публикацию). Часто приложения позиционируют себя как защищенные, но на деле таковыми не являются, так как содержат внутри себя средства для расшифровки персональной информации.
  + **Защита:** Без явной необходимости и явного разрешения пользователя (чаще всего через лицензионное соглашение) ни приложение, ни сервер не должны иметь никакой возможности расшифровать приватные данные пользователя. Простейший пример — пароль пользователя должен уходить на сервер уже в виде хеша, и проверяться должен хеш, а не исходный пароль (серверу абсолютно незачем знать пользовательский пароль; если же пользователь его забыл — для такой ситуации существует давно отлаженный механизм восстановления пароля, в том числе с двухфакторной авторизацией клиента для повышенной безопасности процедуры восстановления).
* Использование самописных алгоритмов шифрования и защиты.

* + **Опасность:** Средняя.
  + **Комментарий:** Это прямое нарушение принципа Керкгоффса. Выражается в попытке разработчика изобрести "свой личный, не известный никому, а поэтому супер-защищенный алгоритм шифрования". Любое отклонение от существующих, многократно проверенных и изученных, математически доказанных алгоритмов шифрования в 99% случаев оборачивается быстрым взломом подобной "защиты". Требует наличия средне-высоких навыков у атакующего.
  + **Защита:** Следует подбирать подходящий алгоритм только из отлаженных и актуальных общеизвестных криптографических алгоритмов.
* Передача КВД во внешнюю среду в открытом виде.

* + **Опасность:** Средняя.
  + **Комментарий:** Выражается в передаче КВД без применения шифрования по любому доступному каналу связи с внешней средой, будь то передача данных стороннему приложению или передача в сеть. Может быть вскрыто опосредованно путем вскрытия не приложения, а его хранилища, или целевого приложения. Взлом требователен к наличию навыков у атакующего, при условии, что хранилище является защищенным.
  + **Защита:** Любые КВД перед выходом за пределы приложения должны быть зашифрованы. Локальные хранилища платформы ***не являются*** областью приложения, они тоже должны получать на вход только зашифрованные данные.
* Игнорирование факта наличия рутованных или зараженных устройств.

* + **Опасность:** Средняя.
  + **Комментарий:** Рутованные устройства — это девайсы, где выполнена модификация для получения прав суперпользователя на любые операции, изначально запрещенные производителем операционной системы. Выполняется пользователем на своем устройстве самостоятельно, и не обязательно добровольно (клиент может быть не в курсе, что устройство взломано). Установка приложения на рутованный девайс нивелирует все штатные средства защиты операционной системы.
  + **Защита:** Если это технически возможно для платформы — то желательно запрещать работу приложения, если удалось понять, что запуск производится на рутованном устройстве, или хотя бы предупреждать об этом пользователя (спасибо за дополнение DjPhoeniX).
* Хранение КВД в защищенных хранилищах, но в открытом виде.

* + **Опасность:** Средняя.
  + **Комментарий:** Разработчики зачастую склонны сохранять КВД в защищенные системные хранилища без дополнительной защиты, поскольку системные механизмы хорошо сопротивляются взлому. Однако уровень их стойкости падает до минимума в случае, если устройство рутованное.
  + **Защита:** КВД не должны использоваться в приложении без дополнительного шифрования. Как только надобность в "открытых" КВД отпала — они немедленно должны быть либо зашифрованы, либо уничтожены.
* Перевод части функционала во встроенные веб-движки.

* + **Опасность:** Средняя.
  + **Комментарий:** Чаще всего выглядит как передача КВД во встроенный браузер, где загружается внешняя веб-страница, выполняющая свою часть функционала. Уровень защиты в этом случае резко снижается, особенно для рутованных устройств.
  + **Защита:** Не использовать встроенный браузер и встроенный веб-движок в операциях с КВД. На крайний случай — шифровать КВД перед передачей.
* Реверсивная инженерия алгоритмов, представляющих интеллектуальную ценность.

* + **Опасность:** Низкая, зависит от ценности алгоритма.
  + **Комментарий:** Если при разработке приложения внутри компании используются некие собственные алгоритмы, которые могут представлять высокую ценность для потенциальных конкурентов или взломщиков, то эти алгоритмы должны быть защищены от постороннего доступа.
  + **Защита:** Автоматическая или ручная обфускация кода.

### **Специфика разработки мобильных приложений**

Есть несколько общих для всех мобильных платформ моментов, которые следует соблюдать при разработке.

#### Защита пользовательским кодом

* Если приложение защищено пользовательским паролем (PIN-кодом, сканом отпечатка пальца, графическим паролем и т.д.), то при уходе приложения в фон ("сворачивании") оно должно немедленно отображать окно ввода этого защитного кода, перекрывая собой весь экран приложения. Это исключает возможность для злоумышленника получить приватную информацию в случае кражи устройства, пока приложение все еще запущено и находится в спящем режиме.
* Любой пользовательский код должен иметь ограниченное количество попыток ввода (например, 5 раз), затем, в случае неудачи, приложение должно автоматически разлогиниваться (или и вовсе блокироваться, зависит от конкретного приложения).
* В настоящее время при использовании цифровых кодов строго рекомендуется использовать ограничение на длину кода в минимум 6 цифр (больше можно, меньше — нельзя).

#### Функционирование клиент-серверного приложения

* Для клиент-серверных приложений очень полезно применять сессионный механизм с ограниченным временем жизни сессии. Это позволит избежать "простаивания" приложения в незащищенном режиме, если пользователь просто забыл закрыть его и оставил устройство в свободном доступе. Следует учитывать, что срок действия сессии и ее идентификатор относятся к КВД, со всеми вытекающими отсюда последствиями. Одним из удачных примеров реализации подобного механизма является получение абсолютного значения времени с сервера после прохождения процедуры авторизации пользователя (дата и время должны показывать, когда именно сессия станет неактивной). Дату и время окончания действия сессии не следует генерировать на устройстве, это снижает безопасность и гибкость приложения.
* Клиент-серверное приложение не должно осуществлять изменение КВД в локальном режиме. Любое действие, требующее изменения КВД, должно проходить синхронизацию с сервером. Исключение из этого правила составляет только пользовательский код входа, задаваемый лично пользователем и сохраненный в защищенном локальном хранилище.

#### Работа с датами

* При оперировании важными для работы приложения датами, вроде времени уничтожения сессии, не следует опираться на относительное время. То есть, передаваемые с сервера данные не должны содержать дату в виде "плюс N секунд/часов/дней от текущего момента". В силу наличия потенциально высоких задержек в передаче данных по сети от мобильного приложения к серверу и обратно, подобный способ синхронизации будет обладать слишком большой погрешностью. Кроме того, атакующий (или просто недобросовестный пользователь) может попросту сменить локальный пояс на устройстве, нарушив таким образом логику работы ограничительных механизмов приложения. Всегда нужно передавать только абсолютное значение времени.
* Абсолютные значения следует передавать с применением универсальных способов обмена подобной информацией, без привязки к часовому поясу конкретного пользовательского устройства. Чаще всего, оптимальным вариантом является поведение приложения, при котором данные отображаются пользователю в его локальном часовом поясе, но их хранение и передача осуществляется в формате, не привязанном к тайм-зоне. Подходящими форматами для дат и времени являются либо универсальный UNIX timestamp, сохраненный в переменной 64-битного целого знакового типа (UNIX timestamp — это количество секунд, прошедшее с 1 января 1970 года), либо, на крайний случай, строка в полном формате ISO-8601 с нулевой тайм-зоной. Предпочтителен именно UNIX timestamp, он позволяет избежать потенциальных ошибок и проблем с конвертацией строк в дату и обратно на разных мобильных платформах.

#### Дополнительные рекомендации

* Приложение не должно отображать приватную пользовательскую информацию большими, яркими, хорошо читаемыми шрифтами, без явной на то необходимости и без отдельного запроса пользователя, чтобы исключить возможность чтения этих данных издали с экрана устройства.
* Не стоит слепо доверять библиотекам с открытым исходным кодом, которые предлагают некую защиту приватным данным пользователей. Исключение составляют библиотеки, проверенные временем и используемые в крупных проектах корпораций (например, встроенное шифрование в открытом движке базы данных Realm). Штатных механизмов защиты операционной системы и общедоступных проверенных криптографических алгоритмов в подавляющем большинстве случаев будет более, чем достаточно.
* Абсолютно недопустимо использовать криптографические библиотеки с закрытым исходным кодом (даже если они платные). В таких решениях вы никак не сможете проверить, насколько эффективна данная библиотека, а также насколько "честная" у нее защита (нет ли там механизма backdoor, или не отсылаются ли "защищенные" данные какой-то третьей стороне).
* В релизных сборках приложений должно быть отключено логгирование данных в системную консоль и незащищенные файлы. Специфические логи для разработчиков могут присутствовать, но желательно в зашифрованном виде, во избежание доступа третьих лиц к закрытой служебной информации, которая может присутствовать в логах.

### **Специфическая информация по платформе iOS**

Рассмотрим доступные для разработчика хранилища данных при разработке под iOS:

* NSUserDefaults.

* + **Штатная защищенность:** Отсутствует.
  + **Комментарий:** Используется только для хранения безопасных для функционирования приложения данных, не связанных с приватной информацией. Чаще всего туда записывают только клиентские настройки интерфейса. Для других данных это хранилище не подходит, так как представляет собой обычный файл и вскрывается за несколько секунд.
* Бинарные файлы (NSKeyedArchiver).

* + **Штатная защищенность:** Средняя, если пользоваться атрибутом NSFileProtectionComplete по ключу NSFileProtectionKey, иначе — отсутствует (спасибо за дополнение agee).
  + **Комментарий:** Сами по себе являются физическими файлами и могут быть элементарно доступны третьей стороне. Уровень защиты зависит исключительно от примененных к данным алгоритмов шифрования. Это не самое удобное место для хранения данных, поэтому без особой необходимости (чаще всего это возможность передавать эти данные, к примеру, по электронной почте) лучше подобный способ не применять. Если файл создан со значением атрибута NSFileProtectionComplete для ключа NSFileProtectionKey, то операционная система удерживает этот файл в зашифрованном состоянии, пока устройство заблокировано или находится в состоянии загрузки, но при несоблюдении этих условий данные будут доступны на чтение и запись, как в обычном файле, так что это лишь временная мера с узкой областью применения.
* Базы данных.

* + **Штатная защищенность:** Зависит от движка и разработчика.
  + **Комментарий:** Современные движки БД поддерживают внутреннее шифрование данных. Разработчику понадобится найти баланс между шифрованием и производительностью, включить шифрование базы для критических данных, а также дополнительно применять шифрование для КВД перед записью в базу.
* Связка ключей (Keychain).

* + **Штатная защищенность:** Максимальная (**не** распространяется на устройства с джейлбрейком).
  + **Комментарий:** Наиболее удачное место для хранения любых КВД. Однако, учитывая, что устройство может быть рутованным, все КВД должны быть зашифрованы дополнительно перед сохранением в Keychain. Кроме того, нужно с осторожностью использовать облачную синхронизацию из-за возможности автоматического сохранения Keychain в облаке. В случае, если приложение использует CloudKit, необходимо внимательно следить за данными, которые не должны синхронизироваться (если таковые имеются), и исключать их из набора копируемых данных.

### **Специфическая информация по платформе Android**

Я слабо разбираюсь в платформе Android, поэтому нижеизложенный список — это краткое тезисное изложение базовых материалов, которые мне удалось найти по этой платформе:

* Наиболее удачным вариантом пользовательского кода является графический код, цифровой (6 цифр или более) — как дополнительный вариант.
* Запрос разрешений (permissions) на определенные виды активности приложения обязателен и должен выполняться явно для пользователя с разъяснением, для чего именно будет использовано разрешение. Запрашивать определенное разрешение нужно только в случае прямой необходимости его использования, также, запрос на разрешение нужно показывать именно в тот момент, когда оно понадобилось, не ранее. Кроме того, если целевая версия Android SDK равна 23 (или новее) — то следует проводить запрос разрешений только через систему разрешений совместимости (Compatibility Permissions System).
* HTTP-клиент должен быть настроен на принудительное использование защищенного канала связи (HTTPS).
* В релизной сборке приложения должны быть отключены все отладочные функции, например, опция debuggable, во избежание возможности подключения к программе внешним отладочным приложением.
* На экранах приложения, где размещается приватная информация пользователя, будет не лишним принудительный запрет на программное создание скриншотов окна приложения, а также отключение показа скриншотов в диспетчере задач.
* Любая приватная информация может дополнительно предваряться запросом личного ключа пользователя, заданного им для входа в приложение (если таковой имеется).
* Внутри кода приложения для резолвинга путей рекомендуется пользоваться getCanonicalPath вместо getAbsolutePath.
* Используемые в приложении открытые компоненты (например, Service или Content Provider) должны быть обязательно закрыты с помощью флага exported = false в манифесте приложения (Android Manifest). Это позволит запретить доступ к этим компонентам из другого приложения.

Кроме того, нужно с осторожностью использовать доступные хранилища информации:

* Shared Preferences.

* + **Штатная защищенность:** Отсутствует.
  + **Комментарий:** Должно использоваться только по прямому назначению, а именно — для хранения общедоступных незащищенных пользовательских настроек. Остальные данные здесь размещаться не должны.
* Базы данных (SQLite).

* + **Штатная защищенность:** Зависит от разработчика.
  + **Комментарий:** Представляет собой обыкновенные файлы, так что все КВД перед записью должны быть соответствующим образом зашифрованы. База допускает возможность автоматического шифрования, его включение будет хорошим усилением защиты данных. В качестве ключа шифрования наиболее целесообразно применять код защиты, задаваемый лично пользователем приложения (код, в свою очередь, должен быть зашифрован и сохранен в Account Manager, см. описание ниже).
* Account Manager.

* + **Штатная защищенность:** Высокая (только до API версии 18).
  + **Комментарий:** Здесь следует размещать все КВД, но предварительно обязательно нужно выполнить дополнительное шифрование данных. После API 18 использование не рекомендуется.
* KeyStore.

* + **Штатная защищенность:** Максимальная (**не** распространяется на рутованные устройства, доступна с API 18).
  + **Комментарий:** Аналогично Account Manager, но настоятельно рекомендуется пользоваться этим хранилищем вместо него, в случае, если KeyStore доступно разработчику (API v.18, Android 4.3 и новее).

## **Заключение**

Также стоит упомянуть, что количество применяемых уровней защиты зависит от конкретного приложения. К примеру, если приложение вообще не является клиент-серверным, не содержит никаких КВД, а также не оперирует ценными внутренними алгоритмами, то вообще нет смысла навешивать на него какую-либо защиту. Если же приложение ориентировано, например, на выполнение банковских операций, или хранение пользовательских паролей, то степень его безопасности должна быть наивысшей. Однако перечисленные ранее общие уязвимости мобильного сектора достаточно легко могут быть исключены из приложения, чаще всего это не вносит особых дополнительных затрат, если применение требуемого уровня защиты было начато на ранних этапах разработки приложения. А вот внедрение защиты пост-фактум в уже работающее приложение вполне может быть сопряжено со значительными затратами сил и времени разработчиков. Поэтому выбор и согласование уровня защищенности, а также перечня КВД в разрабатываемом приложении, должны выполняться на самых ранних этапах проектирования.