13 лекция

Тема: Преобразуйте приложение Android, написанное на Flutter, в iOS и разместите приложение в Play Market и Apple Store.

План:

1. **Иконка приложения**
2. **Подпись приложения**
3. **Связь хранилища ключей с приложением**
4. **Активация Proguard**

В привычном цикле разработки Flutter приложения мы запускаем `flutter run` или используем опции `Run` или `Debug` в IDE. По умолчанию Flutter создает версию приложения для отладки.

И вот версия для публикации готова к размещению, например в Google Play Store. Перед тем как публиковать приложение необходимо несколько завершающих штрихов:

- Добавить иконку приложения (launchpad icon)
- Подписать приложение (signing)
- Активировать Proguard
- Проверить манифест приложения (AndroidManifest.xml)
- Проверить конфигурацию сборки (build.gradle)
- Создать версию приложения для публикации (--release)
- Опубликовать в Google Play Store
- Еще раз посмотреть Android release FAQ

1. Иконка приложения

Когда новое Flutter приложение создается, туда добавляется иконка по-умолчанию (логотип Flutter). Для настройки иконки можно воспользоваться пакетом flutter_launcher_icons. Альтернативный вариант, сделать всё самостоятельно, выполнив следующие шаги:

- Изучить, или хотя бы просмотреть рекомендации и правила Material Design Product icons.
- В директории `<app dir>/android/app/src/main/res/`, поместить файлы иконки в директории соотвествующие соглашению. Стандартные `mimmap`-директории демонстрируют правильное именование.
- В файле `AndroidManifest.xml` в теге `application` обновить атрибут `android:icon` для соответствия иконкам (из предыдущего шага), (для например - `<application android:icon="@mimap/ic_launcher" ...`
- Для проверки, что иконка заменилась, запустить приложение и проверить "иконку запуска"

2. Подпись приложения

Для публикации приложения в Play Store необходимо подписать приложение цифровой подписью.

Создание хранилища ключей

На Mac/Linux

```
# хранилище можно сохранить в файл ~/key.kjs
# или в ~/.keystore - это имя/расположение по умолчанию
keytool -genkey -v -keystore ~/.keystore \
        -keyalg RSA \
   -keysize 2048 \
   -validity 10000 \
   -alias key
```

сохраняйте файл от публичного доступа и конечно не публикуйте его в репозитории проекта

формат файла начиная с версии JAVA 9 по -умолчанию PKS12. Если вы создавали файл ранее, то keytool предложит конвертировать файл в формат PKS12.

3. Связь хранилища ключей с приложением

Создать файл `<app dir>/android/key.properties`, который содержит ссылку на хранилище

```
storePassword=<пароль>
```

```
keyPassword=<пароль>
keyAlias=key
storeFile=<путь к файлу ключа например /home/user/.keystore>
```

и здесь тоже, хранить файл `key.properties` *в секрете и не публиковать его в репозиторий проекта*

Конфигурация подписи в gradle

в файле `<app dir>/android/app/build.gradle`

1. заменить следующее

```
android {
```

на информацию о файле конфигурации

```
def keystoreProperties = new Properties()
def keystorePropertiesFile = rootProject.file('key.properties')
if (keystorePropertiesFile.exists()) {
        keystoreProperties.load(new FileInputStream(keystorePropertiesFile))
}

android {
```

2. Заменить конфигурацию сборки релиза

```
buildTypes {
        release {
        // TODO: Add your own signing config for the release build.
    // Signing with debug keys for now,
    // so `flutter run --release` works
    signingConfig signingConfig.debug
  }
}
```

на следующую информацию, содержащую параметры подписи

```
signingConfigs {
        release {
        keyAlias keystoreProperties['keyAlias']
    keyPassword keystoreProperties['keyPassword']
    storeFile file(keystoreProperties['storeFile'])
    storePassword keystoreProperties['storePassword']
  }
}
buildTypes {
```

```
        release {
        signingConfig signingConfigs.release
    }
}
```

Теперь сборка релиза будет подписываться автоматически.

4. Активация Proguard

По умолчанию Flutter не проводит обусфикацию и минификацию кода Android. Если есть намерение использовать сторонние библиотеки Java, Kotlin или Android, то имеет смысл снизить размер APK и защитить код от "reverse engineering".

Шаг 1 - конфигурация Proguard

Создать файл `/android/app/proguard-rules.pro` и добавить правила

```
## Flutter wrapper
-keep class io.flutter.app.** { *; }
-keep class io.fluter.plugin.** { *; }
-keep class io.flutter.util.** { *; }
-keep class io.flutter.view.** { *; }
-keep class io.flutter.** { *; }
-keep class io.flutter.plugins.** { *; }
-dontwarn io.flutter.embedding.**
```

Эти правила относятся только ко Flutter, для других библиотек нужны свои собственные правила, ну например для Firebase.

Шаг 2 - активация обусфикации и/или минификации

Откроем `/android/app/build.gradle`, найдём определение `buildTypes`. Внутри конфигурации `release`, добавим `minifyEnabled` и `useProguard`, а также укажем файл с конфигурацией Proguard.

```
android {
        ...

        buildTypes {
        release {
        singingConfig singingConfigs.release
```

```
            minifyEnabled true
            useProguard true

            proguardFiles getDefaultProguardFile('proguard-android.txt'), 'proguard-rules.pro'
        }
    }
}
```

конечно минификация и обусфикация увеличать время компиляции файла

Проверим манифест приложения

Проверим манифест приложения, `AndroidManifest.xml`, размещенный в `<app dir>/android/app/src/main` и убедимся, что установленные значения корректны:

- `application`: отредактируйте `android:label` в теге `application` чтобы установить реальное, правильное имя приложения.
- `uses-permissions`: Добавьте разрешение `android.permissions.INTERNET` если приложению требуется доступ в Интернет. Стандартный шаблон приложения не включает этот тег, но позволяет устанавливать соединение в процессе разработки между приложением и инструментами Flutter.

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="org.diera.app.digital_era">

            <uses-permission android:name="android.permission.INTERNET" />
            <uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />

    <application ...>
    </appication>
```

Проверим конфигурацию сборки

Проверить файл конфигурации Gradle, `build.gradle`, расположенный в `<app dir>/android/app` и убедиться, что установленные значения верны:

- `defaultConfig`
- `applicationId` - указан уникальный идентификатор приложения

- `versionCode` и `versionName`: указать внутреннюю версию приложения (число), и строковую версию приложения (для показа пользователям)
- `minSdkVersion` и `targetSdkVersion`: указать минимальный уровнь API, и уровень API для которого было разработано приложение.

`versionCode` - это положительное число, которое используется в качестве внутренней версии приложения. Это число используется только для определения является ли одна версия более новой чем другая. Чем больше число, тем новее версия. Пользователи не видят эти версии. ***Максимально допустимое число - 2100000000.***

`versionName` - строка, используемая в качестве номера версии, показываемой пользователям. Единственная цель versionName - отображение пользователю.
Посмотреть распределение версий можно на [странице статистики версий](). Список [API Levels]()

Сборка релиза

Есть два возможных варианта публикации в Play Store

- App bundle (рекомендуемый)
- APK

Сборка app bundle

Если была произведена настройка подписи, то пакет будет подписан автоматически. Перейти в директорию приложения и выполнить
сейчас (2019) это рекомендованный способ публикации приложений

```
flutter build appbundle --release
```

версии сборки и приложения определяются на основе файла `pubspec.yaml`. `version: 1.0.0+1` - версия приложения это первые три цифры разделенных точками, а версия сборки это число после знака `+`. Это значения можно переопределить при выполнеии команды используя опции `--build-name` (versionName) и `--build-number` (versionCode).

Тестирование app bundle

Есть несколько путей чтобы протестировать app bundle, здесь представлены два

Оффлайн, используя bundle tools

1. Если еще не имеет `bundletools`, то получаем это здесь
2. Генерируем набор APK из пакета приложения
3. Устанавливаем APK на присоединенные устройства

Онлайн, используя Google Play

1. Загрузить пакет в GooglePlay и протестировать. Можно использовать alpha или beta-релизы перед реальной публикацией приложения.
2. Процесс загрузки пакета приложения

Сборка APK

Несмотря на то, что сборка пакета приложения считается приоритетной перед сборкой APK. Могут быть случаи когда пакеты поддерживаются механизмом распространения. Тогда надо будет выпустить APK для каждой цели ABI (Application Binary Interface).

Если цифровая подпись уже настроена в приложении, все APK будут подписаны:

```
flutter build apk --relese --split-per-abi
```

на выходе получим два APK

- `<app dir>/build/app/outputs/apk/release/app-armeabi-v7a-release.apk`
- `<app dir>/build/app/outputs/apk/release/app-arm64-v8a-release.apk`

Если убрать опцию `--split-per-abi`, то получим один "жирный" файл содержащий весь код для всех ABI целей. И пользователю также придется загружать код который не совместим с его платформой.

Для установки APK на устройство, подключим его через USB и выполним команду `flutter install`