ATLS 4120: Mobile Application Development
Week 9: Android Design

App Design

The fundamentals of app design are the same regardless of platform.

1. Design for the device
   - Mobile, mobile, mobile
   - All apps should be easy to figure out and use
   - Make onboarding quick and easy
     - Download, install, start instantly, no lengthy startup
     - Avoid requiring initial signup/login
     - Show content immediately
   - Avoid unnecessary interruptions
   - Interaction is through taps and gestures
     - The comfortable minimum size of tappable UI elements is 44 x 44 points
   - No "Home" in apps
   - Artwork and image should be useful and draw the user in
     - Adapt art to the screen size, high quality media is expected
   - Handle different device sizes and orientations

2. Content
   - Get users to the content they care about quickly
   - Provide only relevant, appropriate content that's useful to the immediate task
   - If in doubt, leave it out

3. Focus on the User
   - Know thy user - target apps to a specific user level
   - Put the user in control
   - Think through the user flow
   - Get them to the relevant information quickly
   - Provide subtle but clear, immediate feedback
   - Create a compelling user experience
     - User interaction consistency

4. Focus on the task
   - The goal of your app and the problem you're trying to solve

5. Understand platform conventions
   - Android launch screens historically have not been recommended
     - Material Design includes a launch screen pattern
       https://material.io/design/communication/launch-screen.html
     - There is no launch screen ability built into Android Studio
     - Don't use timers, it only delays the user and makes your app feel slow
     - Use a launcher theme or activity if you want a custom launch screen
       https://www.bignerdranch.com/blog/splash-screens-the-right-way/
       https://android.jlelse.eu/the-complete-android-splash-screen-guide-c7db82bce565

https://plus.google.com/+IanLake/posts/SW7F2CJvAmU

- Android devices have a back button, your apps don't need one https://developer.android.com/training/implementing-navigation/temporal

Material Design

Material Design https://material.io/

Google launched Material design in 2014 to provide guidance and advice to developers designing and developing apps. (not just Android)

Material is an adaptable system of guidelines, components, and tools that support the best practices of user interface design. Backed by open-source code, Material streamlines collaboration between designers and developers, and helps teams quickly build beautiful products.

Making Material Design

https://www.youtube.com/watch?v=rrT6v5sOwJg&list=PL8PWUWLnnIXPD3UjX931fFhn3_U5_2uZG

Design

Material System Introduction https://material.io/design/introduction/

- Material Design is a visual language that synthesizes the classic principles of good design with the innovation of technology and science.

Material Foundation Overview https://material.io/design/foundation-overview/

- Layout
  - Similar principles from iOS – predictable, consistent, responsive
  - Pixel density https://material.io/design/layout/pixel-density.html#pixel-density
- Color https://material.io/design/color/the-color-system.html
- Material Design color tool https://material.io/tools/color/#!/?view.left=0&view.right=0&primary.color=673AB7
- 2014 Material Design color palettes
- Typography https://material.io/design/typography/the-type-system.html#
  - Type scale
  - Font size units
- Iconography https://material.io/design/iconography/product-icons.html

Launcher Icons

More details on creting launcher icons.

https://developer.android.com/guide/practices/ui_guidelines/icon_design_adaptive

- Launcher(app) icons represent your app. They appear on the home screen, settings, sharing, and can also be used to represent shortcuts into your app
  - Legacy launcher icons Android 7.1(API 25 and before) 48x48dp
  - Adaptive icons were introduced in Android 8.0(API 26) which display as different shapes as needed. Each OEM provides a mask which is used to render the icons. You can control the look of your adaptive launcher icon by defining 2 layers, consisting of a background and a foreground.
    - Both layers must be sized at 108 x 108 dp.
    - The inner 72 x 72 dp of the icon appears within the masked viewport.

        - The system reserves the outer 18 dp on each of the 4 sides to create visual effects, such as parallax or pulsing.
    - We'll use the Image Asset Studio to create all our icons
- Android projects include default icons - ic_launcher.png and ic_launcher_round.png
- Launcher icons go into density specific res/mipmap folders (i.e. res/mipmap-mdpi)
- If you only include the higher resolution version of the icon Android will generate the lower resolutions
- There are also required icons for the action bar, dialog & tab, notifications, and small contextual
- Tablets and other large screen devices request a launcher icon that is one density size larger than the device's actual density, so you should provide your launcher icon at the highest density possible.

Material Guidelines

- Material Theming https://material.io/design/material-theming/overview.html#material-theming
    - Themes are used for create and apply a design across your mock-ups and app
    - Material theme editor available as Sketch plugin
- Implementing your theme

Material Design for Android
More details on creating styles and themes
https://developer.android.com/guide/topics/ui/look-and-feel/
Android styles and themes let you separate the app design from the UI structure and behavior, similar to CSS in web design.
Styles and themes are declared in the style resource file res/values/styles.xml.
A style is a collection of attributes that specify the appearance for a single View.

- You can only apply one style to a View, but some views like TextView let you specifiy additional styles through attributes such as textAppearance on TextViews.
- You can extend an existing style from the framework or support library or your own project using the "parent" attribute.
- They are not inherited by child views

A theme is a type of style that's applied to an entire app, activity, or view hierarchy—not just an individual view. When you apply your style as a theme, every view in the app or activity applies each style attribute that it supports.

To apply a theme use the `android:theme` attribute on either the `<application>` tag or an `<activity>` tag in the `AndroidManifest.xml` file.

Android lets you set styles attributes in different ways:

- directly in a layout
- programmatically
- apply a style to a view
- apply a theme to a layout

You should use themes and styles as much as possible for consistency. If you've specified the same attributes in multiple places, Android's style hierarchy determines which attributes are ultimately applied. The list is ordered from highest precedence to lowest:

1. Applying character- or paragraph-level styling via text spans to TextView-derived classes
2. Applying attributes programmatically
3. Applying individual attributes directly to a View
4. Applying a style to a View
5. Default styling
6. Applying a theme to a collection of Views, an activity, or your entire app
7. Applying certain View-specific styling, such as setting a TextAppearance on a TextView

Components
Details on designing and creating various components across platforms.

Develop
Material Design Develop https://material.io/develop/
Android components, documentation, and theming.

Resources
https://material.io/resources/
Tools

- Color tool https://material.io/resources/color/#!/?view.left=0&view.right=0

Android also provides Quality Guidelines https://developer.android.com/develop/quality-guidelines/

Halloween (theme)
In the Android manifest file you'll see the activity app theme

```
android:theme="@style/AppTheme"
```

If you open styles.xml you'll see where the style name AppTheme is defined.

Let's look at how we can customize the theme and styles of our app.
When you create a project with Android Studio, it applies a theme to your app by default, as defined in your project's `styles.xml` file. This `AppTheme` style extends a theme from the support library and includes overrides for color attributes that are used by key UI elements.

```
<style name="AppTheme"
parent="android:Theme.Material.Light.DarkActionBar"> </style>
```

Let's change the AppTheme style to use a material design theme as the parent (I had to type and use auto complete)

```
parent="android:Theme.Material.Light.DarkActionBar"
```

Run the app to see what it looks like. Try some other themes to see what they look like.

Open colors.xml

```
<color name="colorPrimary">#008577</color>
<color name="colorPrimaryDark">#00574B</color>
<color name="colorAccent">#D81B60</color>
```

You can customize your theme by adding or changing the colors in the colors.xml file.
Add a new color resource. When choosing a color pick the closest material color.

```
<color name="background">#ff9640</color>
```

We can use these by adding them in styles.xml in our AppTheme style.

```
<item name="android:colorPrimary">@color/colorPrimary</item>
<item
name="android:colorPrimaryDark">@color/colorPrimaryDark</item>
<item name="android:colorAccent">@color/colorAccent</item>
<item name="android:background">@color/background</item>
```

We use @color because the style values are references to our color resources.

Creating a new theme

You can create as many themes as you want by just adding them to styles.xml
Let's first pick out the colors for our new theme.
You can use the color picker tool to generate a color palette https://material.io/tools/color/ Once you generate a palette you can export for Android and it will create an xml file for you. (On Mac this opens in Xcode as default)
Copy these into colors.xml.

```
<color name="primaryColor">#ff8f00</color>
<color name="primaryLightColor">#ffc046</color>
<color name="primaryDarkColor">#c56000</color>
<color name="secondaryColor">#6d4c41</color>
<color name="secondaryLightColor">#9c786c</color>
<color name="secondaryDarkColor">#40241a</color>
<color name="primaryTextColor">#000000</color>
<color name="secondaryTextColor">#ffffff</color>
```

Then in styles.xml create a new theme.

```
<style name="AutumnTheme"
parent="android:Theme.Material.Light.DarkActionBar" >
    <item name="android:colorPrimary">@color/primaryColor</item>
    <item
name="android:colorPrimaryDark">@color/primaryDarkColor</item>
    <item
name="android:colorAccent">@color/secondaryLightColor</item>
    <item
name="android:textColorPrimary">@color/secondaryDarkColor</item>
    <item
name="android:textColor">@color/secondaryDarkColor</item>
    <item
```

```
name="android:colorBackground">@color/primaryColor</item>
    <item
name="android:colorButtonNormal">@color/primaryDarkColor</item>
</style>
```

To use this theme for your whole app update AndroidManifest.xml
```
android:theme="@style/AutumnTheme"
```

You can also define different styles for different Android versions if you want to use newer features while still being compatible with old versions. All you need is another styles.xml file saved in a values directory that includes the resource version qualifier.
res/values/styles.xml # themes for all versions
res/values-v21/styles.xml # themes for API level 21+ only
The common structure to do this is to define a base theme and then use it as the parent for any version specific themes so you don't need to duplicate any styles.

If you get the error "You need to use a Theme.AppCompat theme (or descendant) with this activity" when using a Material theme it's due to the new AndroidX library and how it supports Material design.
Go into your gradle scripts and open build.grade(Module: app).
Under dependencies add the following to include the new materials library:
```
implementation 'com.google.android.material:material:1.0.0'
```
The click Sync now at the top of the gradle files. This will make the new Material Component themes available.
Now in styles.xml you should be able to use the themes under Theme.MaterialComponents.* as the parent theme and not get an error.
```
parent="Theme.MaterialComponents.Light.DarkActionBar"
```
(more info and all Material Component themes https://material.io/develop/android/docs/getting-started/)

Launcher icons
https://developer.android.com/studio/write/image-asset-studio.html
Add your own launcher icon to your app.
Select the res folder right-click and select New > Image Asset
Or File > New > Image Asset.

Icon type: Launcher Icons (Adaptive and Legacy)
Use Adaptive and Legacy if you're supporting Android 8, otherwise you can just do Legacy.
Name: leave as ic_launcher so you don't need to change it in the Android_manifest.xml file

Foreground Layer:
Layer Name: ic_launcher_foreground
- Select Image to specify the path for an image file. (pumpkin.png)
- Select Clip Art to specify an image from the material design icon set.
- Select Text to specify a text string and select a font.

Scaling – trim and resize as needed.

Background Layer:
Layer Name: ic_launcher_background

- Asset Type, and then specify the asset in the field underneath. You can either select a color or specify an image to use as the background layer.

Scaling – trim and resize as needed.

Next
Res Directory: main
Output Directories: main/res

Finish
Image Asset Studio adds the images to the mipmap folders for the different densities.

You can see the various launcher files created in the mipmap folder.

Now run your app and look at your launcher icon by clicking the right button, or go to the home screen.

You can download the Android Icon Templates Pack
https://developer.android.com/guide/practices/ui_guidelines/icon_design.html
You can also use Android Asset Studio to create all your launcher images.
http://romannurik.github.io/AndroidAssetStudio/index.html