Mobile Application Development
Week 9: Android Design

With 2.9 million apps in the Google play store users have many choices when choosing an app.

- 25% of apps aren't used more than once
- 29% of users will switch to another app if an app doesn't meet their needs
    - 70% due to lagging load times
    - 67% too many steps
- 34% aren't opened more than 11 times

To attract and keep users, apps need to be well designed and provide a delightful app experience.

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
    - Apps should have a well-defined target 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
    - Apps should have a well-defined goal
    - What problem is your app solving?

- How is it better or different from other apps?

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
- Android devices have a back button, your apps don't need one

Principles of Mobile App Design: Engage Users and Drive conversations
http://think.storage.googleapis.com/docs/principles-of-mobile-app-design-engage-users-and-drive-conversions.pdf (checklist at the end)

For each section what's a good or bad example of one of the principles in an app you use?

Chapter 1: App Navigation and Exploration
- Show the value of your app upfront
- Organize and label menu categories to be user friendly
  - Be predictable
  - Less is more
- Allow users to go back easily in one step
  - Back button to navigate in reverse-chronological order through the history of screens the user has recently worked with.
  - Up navigation to navigate to the logical parent screen in the app's hierarchy.
- Make it easy to manually change location
- Create frictionless transitions between mobile apps and the mobile web.

Chapter 5: Form Entry
- Build user-friendly forms
  - ensure that form fields are not obstructed from view by interface elements such as the keyboard.
  - automatically advance each field up the screen.
  - include efficiencies like auto-populate, auto-capitalization, and credit card scanning.
  - Use the hint attribute instead of default text in an edittext so the user doesn't need to delete the default text
- Communicate form errors in real time
- Match keyboard with the required text inputs
  - Specify keyboard type with the "inputType" attribute for EditText with values such as "phone" or "textPassword".
  - Can combine values for various behavior
  - android:inputType="textCapSentences|textAutoCorrect"

    - Specify the user action button with the “imeOptions” attribute with an action value such as "actionSend" or "actionSearch".
- Provide helpful information in context in forms

Chapter 6: Usability and Comprehension

- Speak the same language as your users
    - No big words
    - Avoid jargon or unconventional terminology
    - Be descriptive
    - Be succinct
    - Avoid truncation
    - Make text legible
    - Clear communication and functionality should always take precedence over promoting the brand message.
    - Language will depend on your target user
        - Expert vs novice will expect different language
    - Mental model
        - A mental model is a user’s beliefs and understanding about a system based on their previous experiences
        - A users mental model impacts how they use a system and helps them make predictions about it
        - Mismatched mental models are common, especially with designs that try something new
- Provide text labels and visual keys to clarify visual information
    - Labeled icons are more easily understood
    - Categories with visual indicators should include a key
- Be responsive with visual feedback after significant actions
    - Provide clear feedback
    - Communication
        - Toasts are used to provide the user with simple feedback in a small popup https://developer.android.com/guide/topics/ui/notifiers/toasts
            - no action required from the user, not clickable
            - fills a small amount of space
            - automatically disappears
        - SnackBars are very similar to toasts but more customizable. Snackbars animate up from the bottom of the screen and can be swiped away from the user. They are prominent enough that the user can see it, but not so prominent that it prevents the user from working with your app. They automatically disappear like Toasts.
          https://developer.android.com/training/snackbar
            - Snackbars allow you to add an action for a user response
                - A button will be added to the right of the message
            - Snackbars supercede Toasts and are the preferred method for displaying brief, transient messages to the user
        - A dialog is a small window that prompts the user to make a decision or enter additional information.
          https://developer.android.com/guide/topics/ui/dialogs

        - Dialogs take over the screen and requires the user to take an action before they can proceed.
        - They are disruptive as the user must act on them in order to continue.
        - Only use a Dialog when the user's response is critical to your app flow as they are disruptive, otherwise use a SnackBar or Toast.
- Let the user control the level of zoom
- Ask for permissions in context
    - Communicate the value the access will provide
    - Users are more likely to grant permission if asked in the context of the relevant task
    - Users are only prompted once so you want to ask them at the most likely place for them to agree
        - Users can change permissions in settings but many won't
    - Every Android app runs in a limited-access sandbox. If an app needs to use resources or information outside of its own sandbox, the app has to request the appropriate permission.
    - You declare that your app needs a permission by listing the permission in the app manifest using a <uses-permission> element. In Android 6.0 Marshmallow(API 23) the permission system was redesigned so apps have to ask users for each permission needed at runtime. You must call the permission request dialog programmatically.
        - When the user responds to your app's permission request, the system invokes the onRequestPermissionsResult() method, passing it the user response. Your app has to override that method to find out whether the permission was granted and behave accordingly.
        - If an app tries to call some function that requires a permission which user has not yet granted, the function will suddenly throw an exception and the application will crash.
        - Your app must handle if the user denies permission or selects the "Don't ask again" option in the permission request dialog
        - Also, users are able to revoke the granted permission anytime through the device's settings so you always need to check to see if they've granted permission and request again if needed.

Small group discussion (lab 6):
What's an example of an app you've used that either does a good job, or not, of one of these design principles?

Material Design
Android Design https://developer.android.com/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://www.material.io/tools/color/
  - Material Design Palette https://www.materialpalette.com/ (click on two for primary and secondary colors)
- 2014 Material Design color palettes
- Typography https://material.io/design/typography/the-type-system.html#
  - Type scale
  - Font size units
    - always use the "sp" (scale-independent pixel) unit for font sizes as it allows the font size to scale for the user based on their settings
- Iconography https://material.io/design/iconography/product-icons.html

Launcher Icons
More details on creating 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 to create and apply a design across your mock-ups and app
    - Material theme editor available as Sketch plugin
- Implementing your theme

Components

Details on designing and creating various components across platforms.
Android Components https://www.material.io/develop/android/components/dialog/

Develop

Material Design Develop https://www.material.io/develop/
Android Components https://www.material.io/develop/android/components/dialog/
Theming

- Color theming – default colors
- Dark theme
    - The DayNight themes are for the new Dark Mode that was added in Android 10. Before that it was up to each app individually to implement it, or the OEMs that added on dark/night mode support.
- Typography theming – default styles

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/themes.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 specify additional styles through attributes
- You can extend an existing style from the framework 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 at multiple levels (global to local, with local taking precedence):

- applied based on the global attribute (from current theme)
- applied based on the global style of the view (from current theme)
- applied based on the style of the view
- applied based on the style in the XML layout
- applied programmatically

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.

AndroidX Library

Initially when Material Design was released there was no official support on how to implement it. Then Google released the Design support library to help developers implement Material Design. In 2019 Google released AndroidX which is the next evolution of the support library and implements Material Components for Android. With this release the support library won't be maintained in the future so we'll need to use AndroidX.

All new projects are automatically created using AndroidX and starting in AS 4.1 Material Components.

The app Gradle file (Module: app) must have a compileSdkVersion of 28 or later and the following dependency so you can use the Material Components themes (1.1.0 or later).

```
implementation 'com.google.android.material:material:1.1.0'
```

Halloween (theme)

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

```
android:theme="@style/Theme.Halloween"
```

If you open themes.xml you'll see where the style name Theme.Halloween is defined.
This style extends a theme from Material Components.

```
<style name="Theme.Halloween"
parent="Theme.MaterialComponents.DayNight.DarkActionBar">
</style>
```

Customizing themes

These are the Material Component themes that you can use for the parent.

- Theme.MaterialComponents
- Theme.MaterialComponents.NoActionBar
- Theme.MaterialComponents.Light
- Theme.MaterialComponents.Light.NoActionBar
- Theme.MaterialComponents.Light.DarkActionBar
- Theme.MaterialComponents.DayNight

- Theme.MaterialComponents.DayNight.NoActionBar
- Theme.MaterialComponents.DayNight.DarkActionBar

Try some other themes to see what they look like.

Let's look at how we can customize the theme and styles of our app.
In themes.xml our theme has many colors defined. If you look at the values they all start with @color because they are references to color resources.
Open colors.xml to see the actual values for each color name.

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="orange">#ff9640</color>
```

We can use it by adding them in themes.xml in our Theme.Halloween style.

```
<item name="android:background">@color/orange</item>
```

Run the app to see your new background color.

Creating a new theme

You can create as many themes as you want by just adding them to themes.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 so choose and editor instead)
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 themes.xml create a new theme.

```
<style name="Theme.Autumn"
parent="Theme.MaterialComponents.DayNight">
    <item name="colorPrimary">@color/primaryColor</item>
    <item name="colorPrimaryDark">@color/primaryDarkColor</item>
    <item name="colorAccent">@color/primaryLightColor</item>
    <item name="colorOnPrimary">@color/primaryTextColor</item>
    <item name="colorSecondary">@color/secondaryColor</item>
    <item
name="colorSecondaryVariant">@color/secondaryLightColor</item>
    <item
name="colorOnSecondary">@color/secondaryTextColor</item>
```

```
    <item
name="android:colorBackground">@color/secondaryLightColor</item>
</style>
```

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

Creating a new style
This is also a good place to set any styles you want to use throughout your app. So instead of setting the textAppearance for our TextView in the layout file, we can define the style in in themes.xml.
```
<style name="halloweenTextStyle">
     <item
     name="android:textAppearance">@style/TextAppearance.Materia
     lComponents.Headline5</item>
     <item name="android:fontFamily">cursive</item>
</style>
```

And then we can use that style in layout files or in our theme.
```
    <style name="Theme.Autumn"
...
    <item
name="android:textViewStyle">@style/halloweenTextStyle</item>
    </style>
```

AS also supports downloadable Google Fonts.
https://developer.android.com/guide/topics/ui/look-and-feel/downloadable-fonts
In the layout file select the TextView and find the fontFamily attribute. Click on the select list and you'll see the default Android fonts and then More fonts. Click on More fonts and select the downloadable font such as Eater and select Add font to project. This creates a res/font folder with the font file.
In themes.xml change the fontFamily to the one you just added.
```
     <item name="android:fontFamily">@font/eater</item>
```

You can also download other fonts and add them as a resource.

Applying styles
https://developer.android.com/guide/topics/ui/look-and-feel/themes
You can use the different styling methods to take advantage of the precedence order in which they're applied. You should use themes and styles as much as possible for consistency. If you've specified the same attributes in multiple places, the list below 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`

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 themes.xml file saved in a values directory that includes the resource version qualifier.
res/values/themes.xml # themes for all versions
res/values-v21/themes.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.
You can see we already have a res/values-night/themes.xml for themes we want to use in night mode. (Go into Project view to show this).

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)
(Legacy is for pre Android 8)
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