Mobile Application Architecture

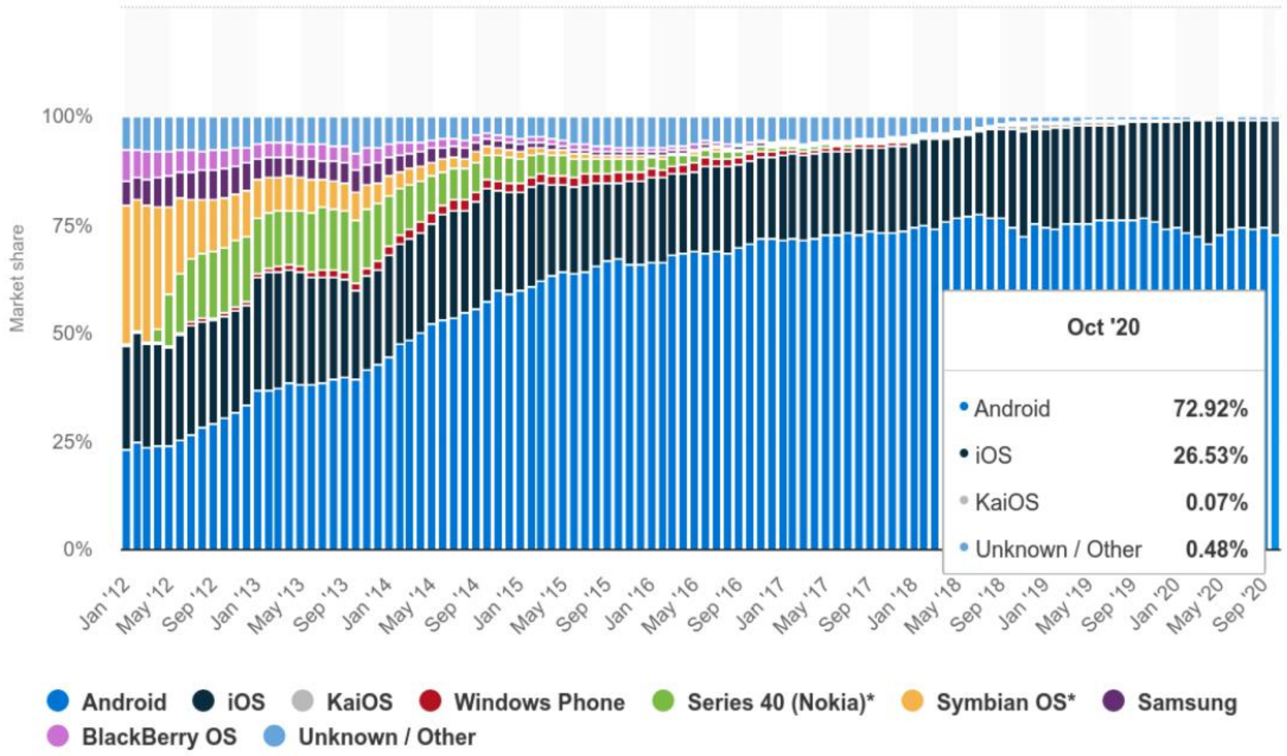
Hua Huang

What is Android?

- Android is world’s leading mobile operating system
    - Open source (https://source.android.com/setup/)

- Google:
    - Owns Android, maintains it, extends it
    - Distributes Android OS, developer tools, free to use
    - Runs Android app market

Mobile OS Market Share



- Android: 72.9%
- IOS: 26.5%

What Android devices have you used?

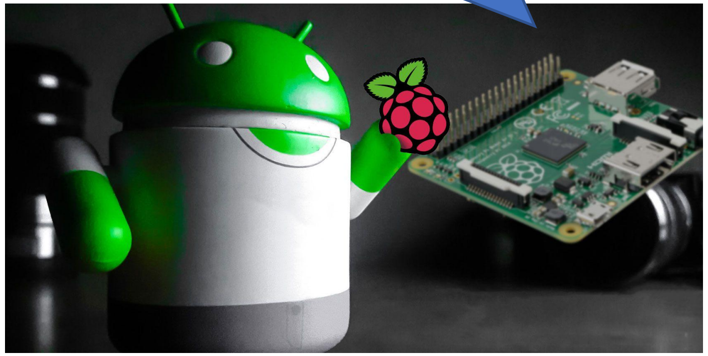
Android Tablet

Android Auto

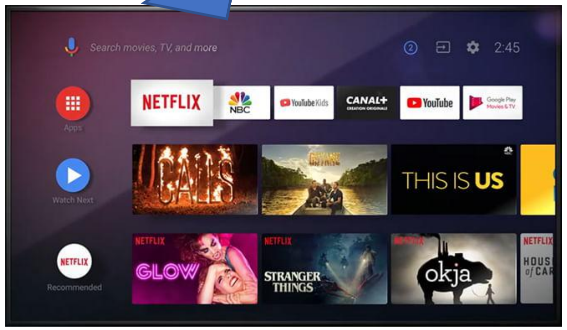
Smartwatch

Google Glass

Smartphone



Embedded devices
(e.g. Raspberry Pi)



Android TV

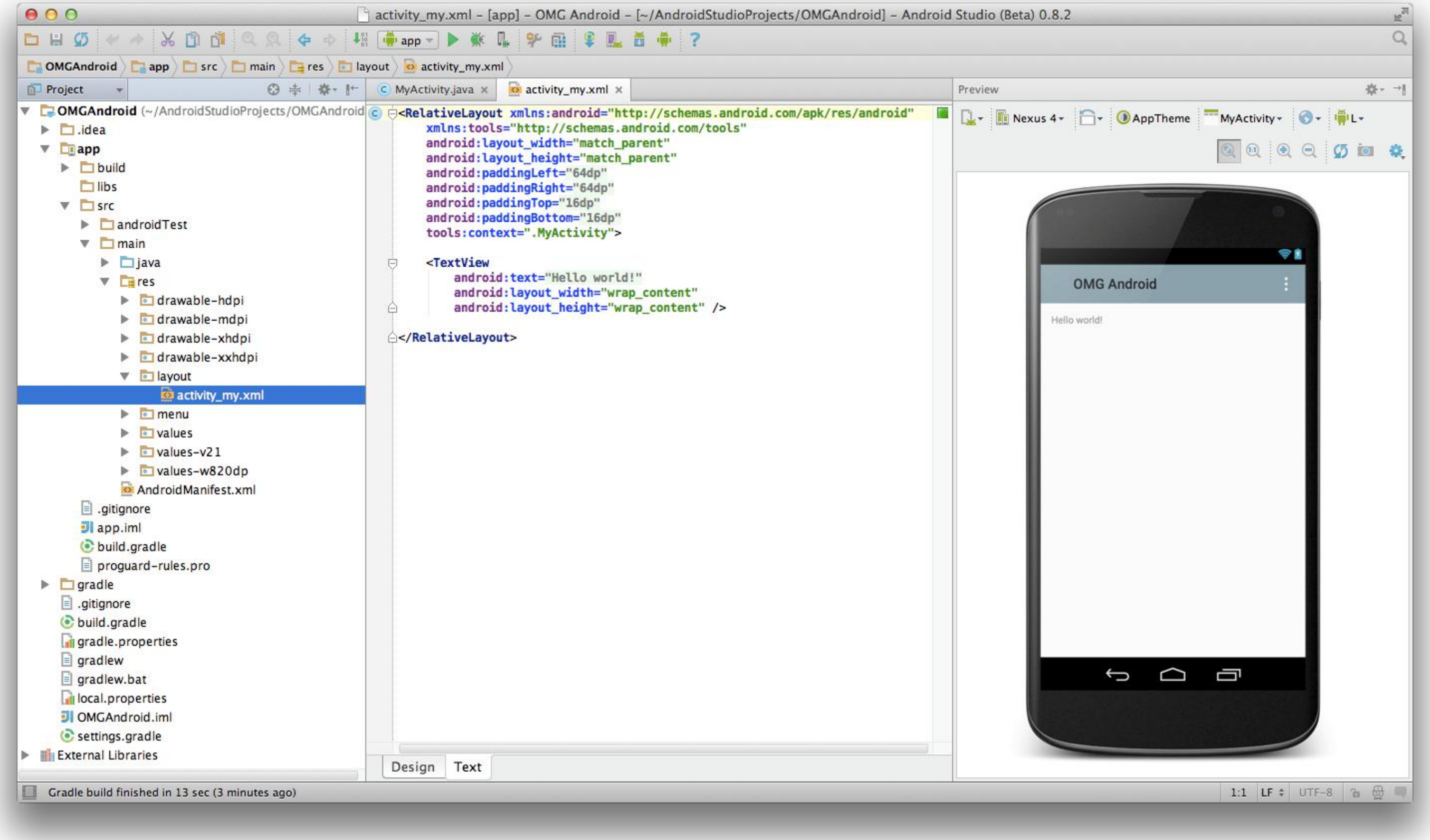
Why Android?

- Contains rich mobile and ubicomp programming modules
  - Sensors: GPS, microphone, camera, IMU, ...
  - Data processing: machine learning
  - Application: activity recognition, bio-sign monitoring, speech recognition, ...

The Android Development Environment

Android Studio

- The official Integrated Development Environment (IDE)



activity_my.xml - [app] - OMG Android - [~/AndroidStudioProjects/OMGAndroid] - Android Studio (Beta) 0.8.2

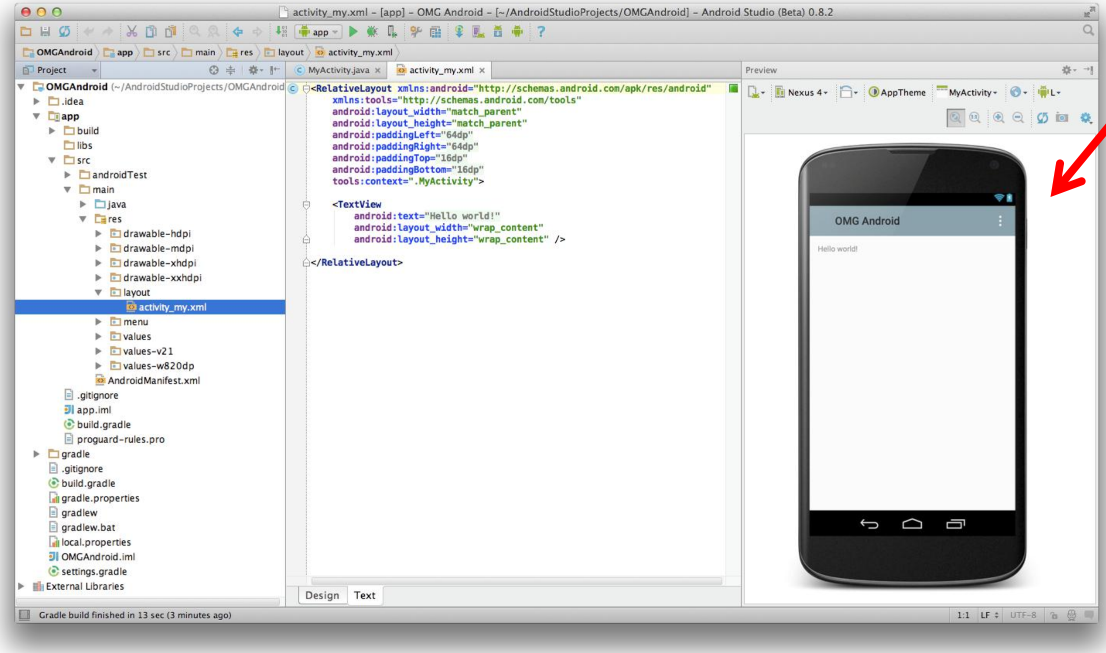
```xml
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:paddingLeft="64dp"
    android:paddingRight="64dp"
    android:paddingTop="16dp"
    android:paddingBottom="16dp"
    tools:context=".MyActivity">

    <TextView
        android:text="Hello world!"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content" />

</RelativeLayout>
```

Gradle build finished in 13 sec (3 minutes ago)

Where to run Android Apps

- Android app can run on:
  - Real phone (or device)
  - Emulator (software version of phone)

Emulated Phone in Android Studio



activity_my.xml – [app] – OMG Android – [~/AndroidStudioProjects/OMGAndroid] – Android Studio (Beta) 0.8.2

```
OMGAndroid (~/AndroidStudioProjects/OMGAndroid)
  .idea
  app
    build
    libs
    src
      androidTest
      main
        java
        res
          drawable-hdpi
          drawable-mdpi
          drawable-xhdpi
          drawable-xxhdpi
          layout
            activity_my.xml
          menu
          values
          values-v21
          values-w820dp
        AndroidManifest.xml
    .gitignore
    app.iml
    build.gradle
    proguard-rules.pro
  gradle
  .gitignore
  build.gradle
  gradle.properties
  gradlew
  gradlew.bat
  local.properties
  OMGAndroid.iml
  settings.gradle
External Libraries
```

```xml
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:paddingLeft="64dp"
    android:paddingRight="64dp"
    android:paddingTop="16dp"
    android:paddingBottom="16dp"
    tools:context=".MyActivity">

    <TextView
        android:text="Hello world!"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content" />

</RelativeLayout>
```

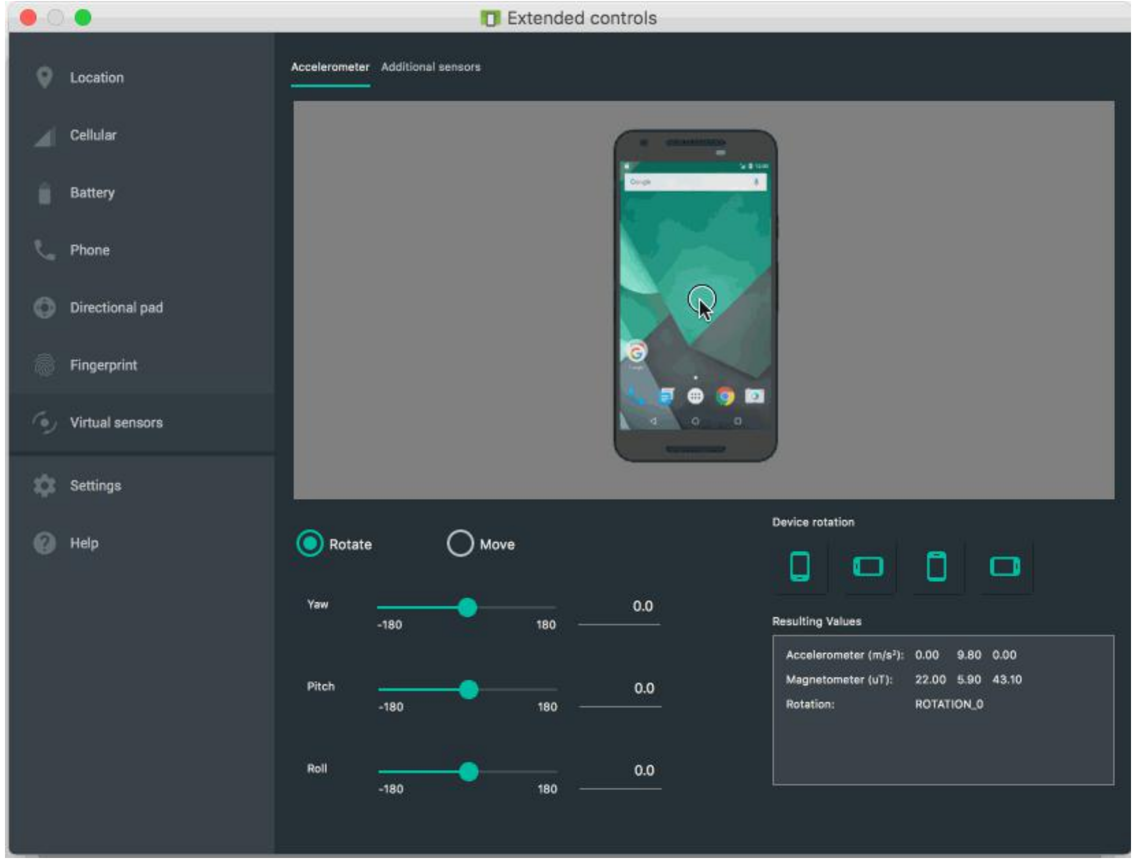
Gradle build finished in 13 sec (3 minutes ago)

Run Android App on Real Phone

- Computer side: install Android Debug Bridge (ADB)
- Phone side: enable USB debugging
- Tutorial: https://beebom.com/how-to-install-adb-windows-mac/

Support for sensors in emulators

- Can now emulate some sensors (locations, acceleration)



● Rotate ○ Move

Yaw 0.0 (-180 to 180)

Pitch 0.0 (-180 to 180)

Roll 0.0 (-180 to 180)

```
Accelerometer (m/s²):  0.00   9.80  0.00
Magnetometer (uT):     22.00  5.90  43.10
Rotation:              ROTATION_0
```

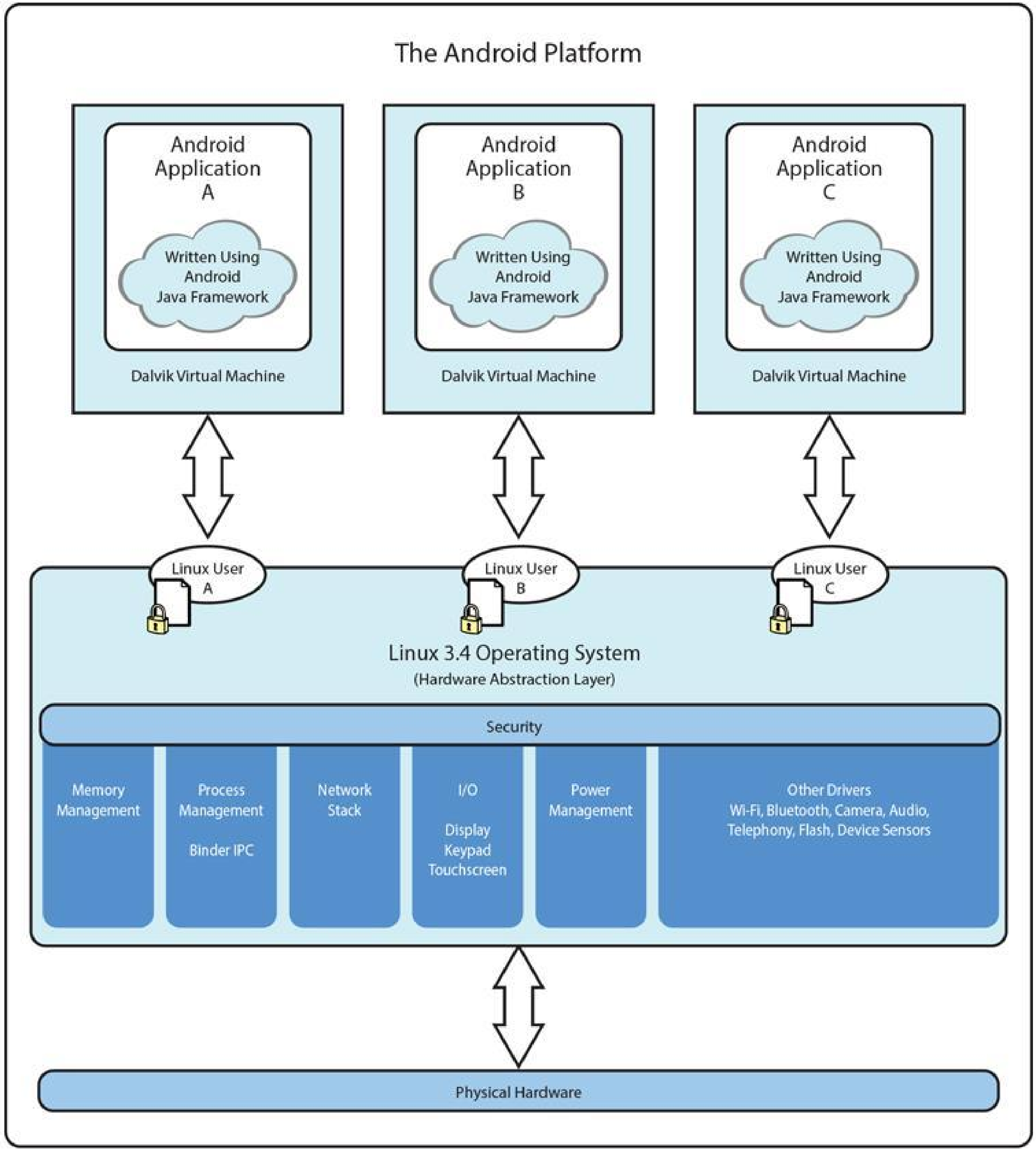
Android Architecture

The Basics

- In general, all apps are written in Java or Kotlin
- A compiled Android app is an .apk
- Android apps must be digitally signed in some way to execute
- This digital signature can be a debug certificate that comes default with any installation

Software Framework

- Each Android app runs in its own security sandbox (VM, minimizes complete system crashes)
- Android OS multi-user Linux system
- Each app is a different user (assigned unique Linux ID)
- Access control: only process with the app's user ID can access its files

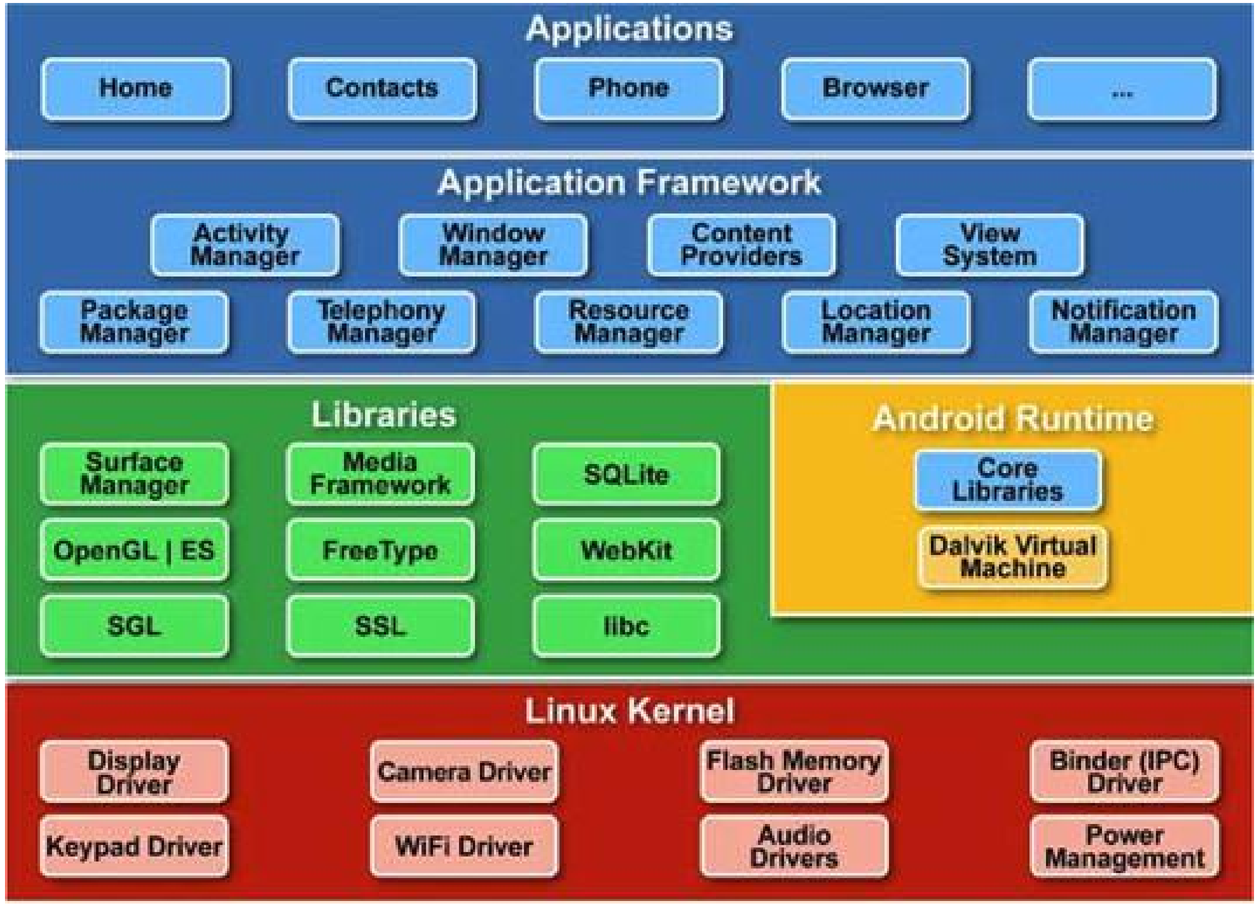


How to share data across apps?

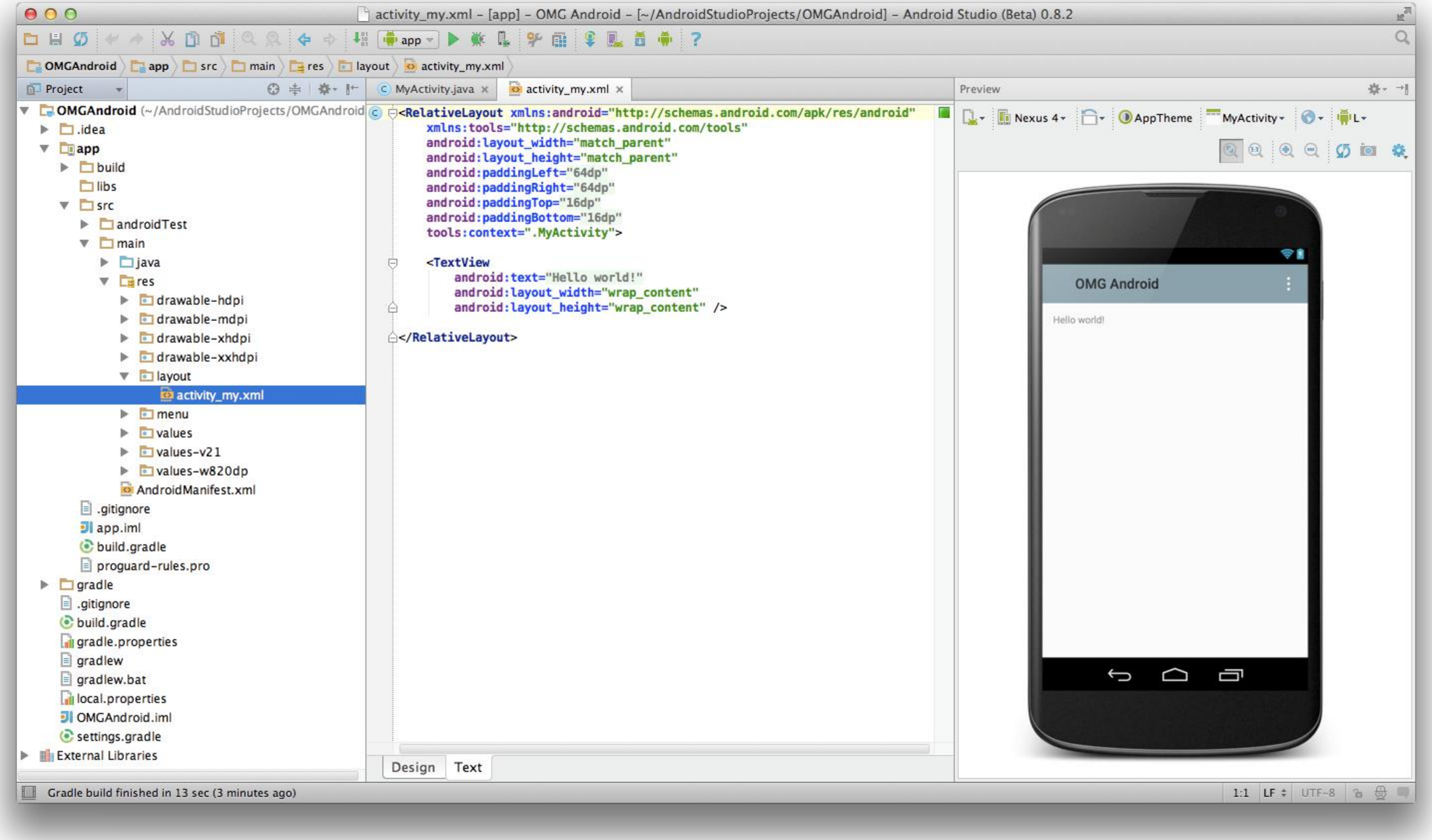
- It is possible for apps to share data with other apps
  - Use intents to send simple data
  - Save to the shared storage
  - Send resource URL and grant temporary access
  - Two apps can have the same Linux user id (and thus share resources) if and only if they are signed by the same digital certificate

The Android Architecture

- **OS**: Linux kernel, drivers
- **Apps**: programmed & UI in Java
- **Libraries**: OpenGL ES (graphics), SQLite (database), etc



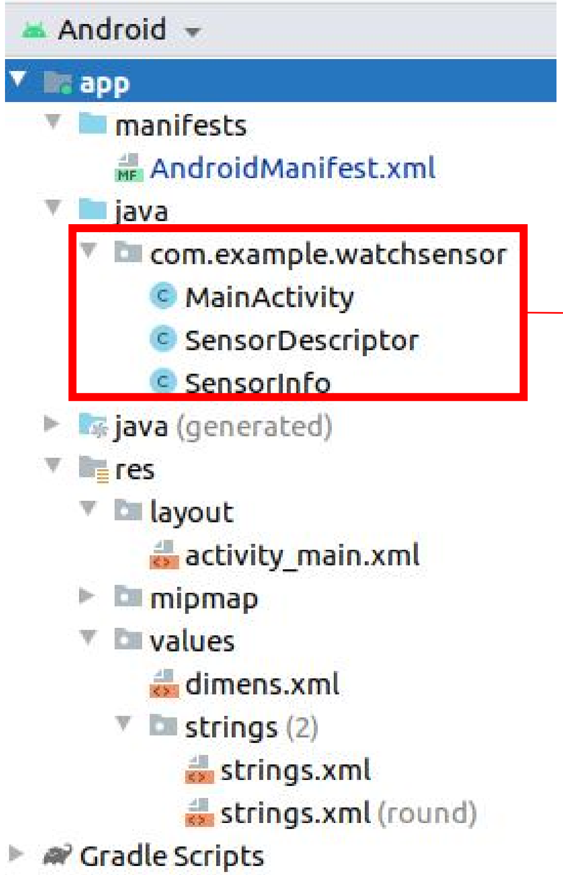
Android Studio Layout



activity_my.xml - [app] - OMG Android - [~/AndroidStudioProjects/OMGAndroid] - Android Studio (Beta) 0.8.2

```xml
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:paddingLeft="64dp"
    android:paddingRight="64dp"
    android:paddingTop="16dp"
    android:paddingBottom="16dp"
    tools:context=".MyActivity">

    <TextView
        android:text="Hello world!"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content" />

</RelativeLayout>
```

Gradle build finished in 13 sec (3 minutes ago)



Java Program file

Key Concepts of Android Apps

- Activities – represent a single screen with a UI
- Services – represents a process running in the background
- Content Provider – a link back to the data
- Broadcast Receiver – listens for system-wide messages to respond to
- Application – a set of Activities that make up a cohesive unit
- Intent – a message to be passed

Activity

- Conceptually, an Activity is a single screen of your application
- In other words, an App really is a collection of related Activities
- Consider each Activity both a screen and a feature
- Apps can activate Activities in other Apps

Service

- A Service is a component that runs in the background to perform long-running operations
- A Service has no UI
- Examples of Services:
  - Playing music in background
  - Gathering GPS data
  - Downloading a data set from the server

Broadcast Receiver

- A Broadcast Receiver responds to system-wide announcements (which are manifested as Intents)
- System status information is delivered this way (i.e. device turned on side, screen off, low battery, phone call incoming, etc.)
- Broadcast Receivers typically don’t have a UI, but could have a status bar icon

Intent

- An Intent is a message that requests an action from another component of the system
- This includes the “please start up your App” Intent that the system sends when a user clicks on your App icon

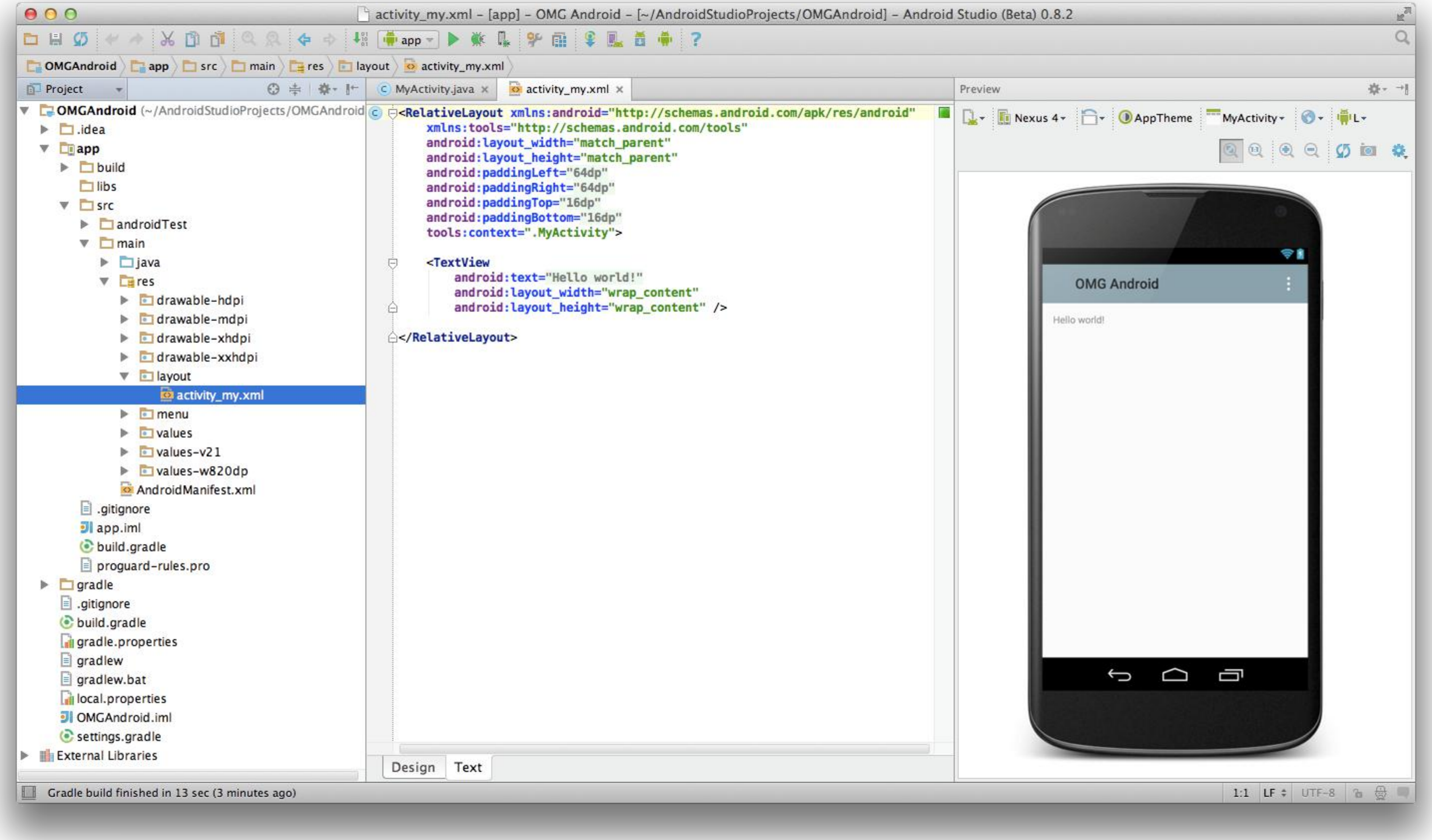
Connected Apps

- Due to the component nature of Apps (made up of Activities, Services, etc.), it is easy to build features of your App using existing system components
- For example, if your App needs to take a picture, you can query the Camera Activity to handle that request and return the resulting image
- This is handled through Intents

Put them all together

- If an App is made up of all these disparate parts, what holds them all together?
- The AndroidManifest.xml file!
  - Sets up all permissions the user has to agree to (i.e. Internet, GPS, contacts, etc.)
  - Declares the API level of the App
  - Requests hardware features needed
  - Needed libraries
  - Which Activities are part of this App

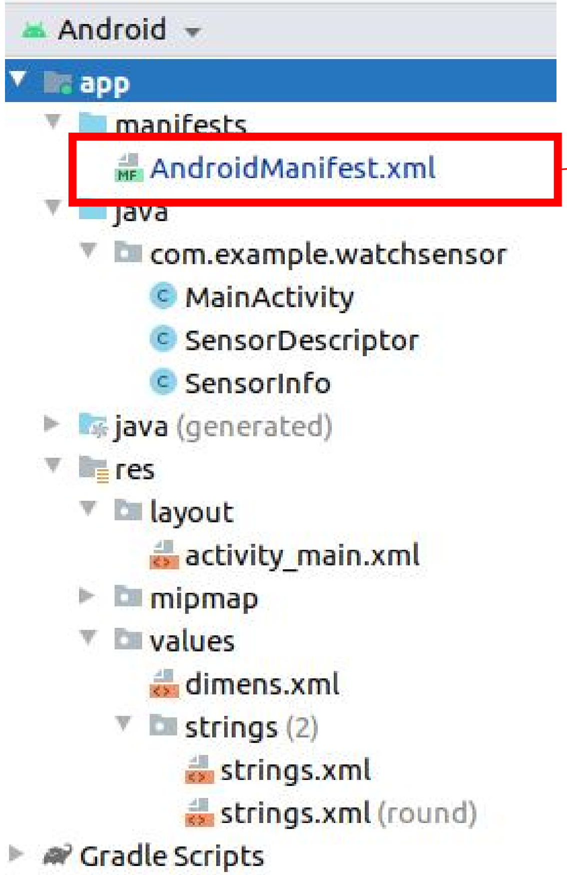
Android Studio Layout



activity_my.xml – [app] – OMG Android – [~/AndroidStudioProjects/OMGAndroid] – Android Studio (Beta) 0.8.2

```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:paddingLeft="64dp"
    android:paddingRight="64dp"
    android:paddingTop="16dp"
    android:paddingBottom="16dp"
    tools:context=".MyActivity">

    <TextView
        android:text="Hello world!"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content" />

</RelativeLayout>
```

Gradle build finished in 13 sec (3 minutes ago)

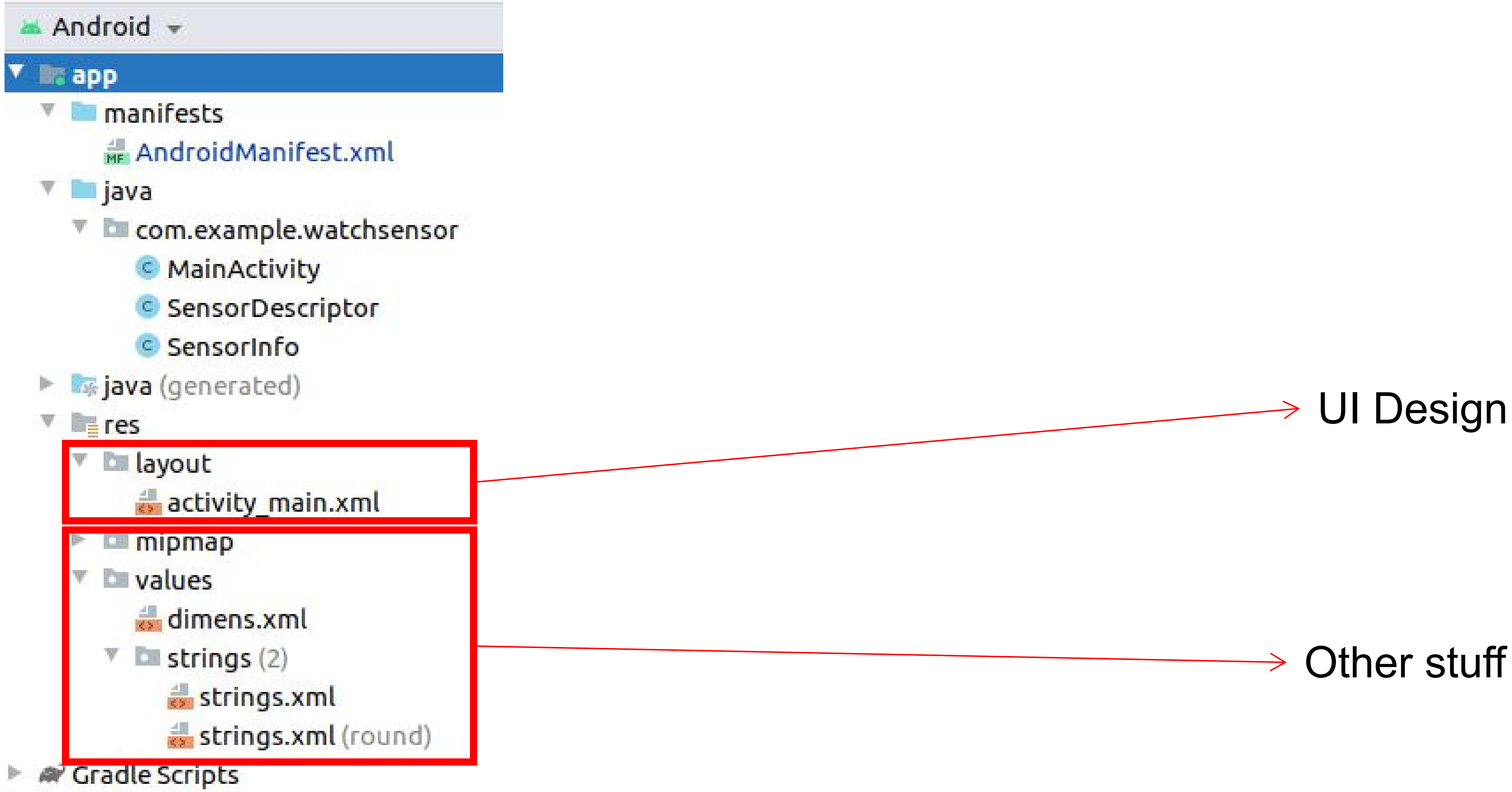


```
Android
▼ app
    ▼ manifests
        AndroidManifest.xml
    ▼ java
        ▼ com.example.watchsensor
            MainActivity
            SensorDescriptor
            SensorInfo
    ▶ java (generated)
    ▼ res
        ▼ layout
            activity_main.xml
        ▶ mipmap
        ▼ values
            dimens.xml
            ▼ strings (2)
                strings.xml
                strings.xml (round)
▶ Gradle Scripts
```

Manifest file

Where’s the UI?

- The User Interface for an Android App is defined in the layout xml files
- Each layout xml file should correspond to an Activity



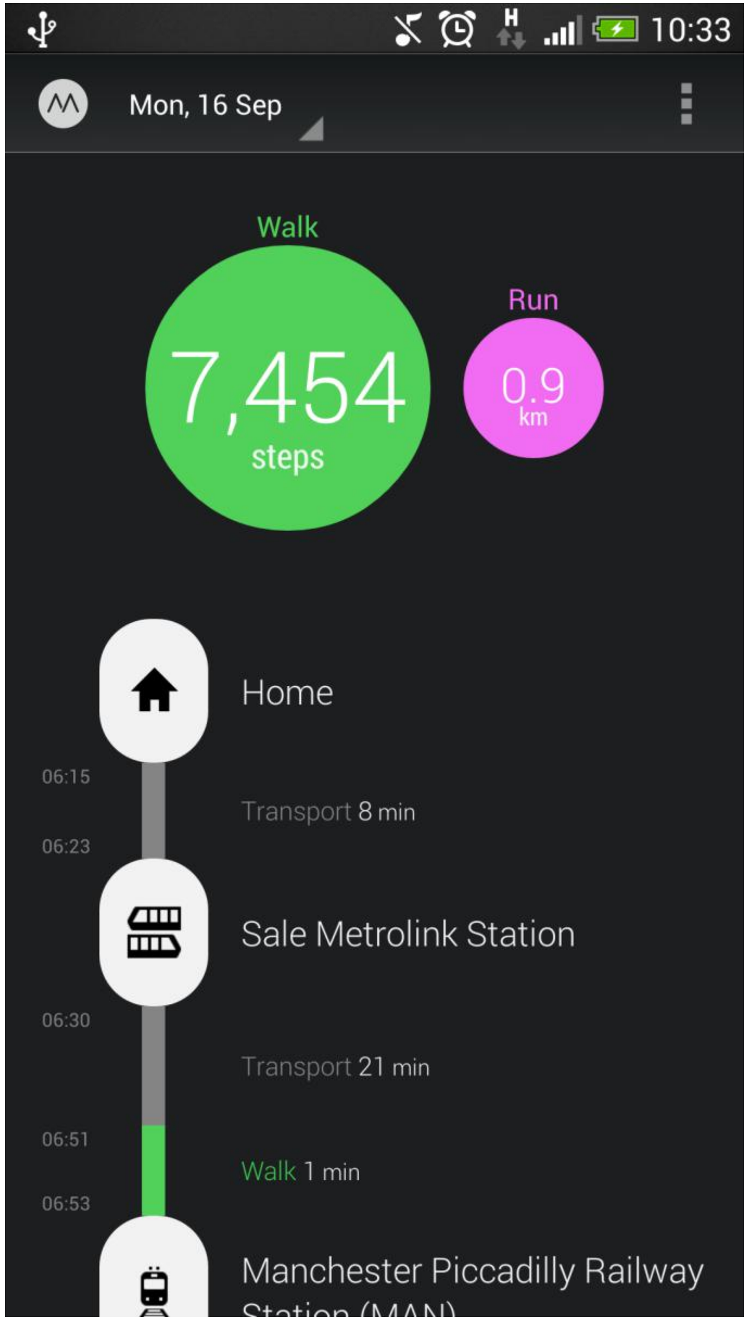
```
Android
▼ app
  ▼ manifests
      AndroidManifest.xml
  ▼ java
    ▼ com.example.watchsensor
        MainActivity
        SensorDescriptor
        SensorInfo
  ▶ java (generated)
  ▼ res
    ▼ layout
        activity_main.xml
    ▶ mipmap
    ▼ values
        dimens.xml
      ▼ strings (2)
          strings.xml
          strings.xml (round)
▶ Gradle Scripts
```

UI Design

Other stuff

UI Design using XML

- UI design code (XML) separate from the program (Java)
- Why? Can modify UI without changing Java program
- Example: Shapes, colors can be changed in XML file without changing Java program
- UI designed using either:
  - Drag-and drop graphical (WYSIWYG) tool or
  - Programming Extensible Markup Language (XML)
- XML: Markup language, both human-readable and machine-readable''



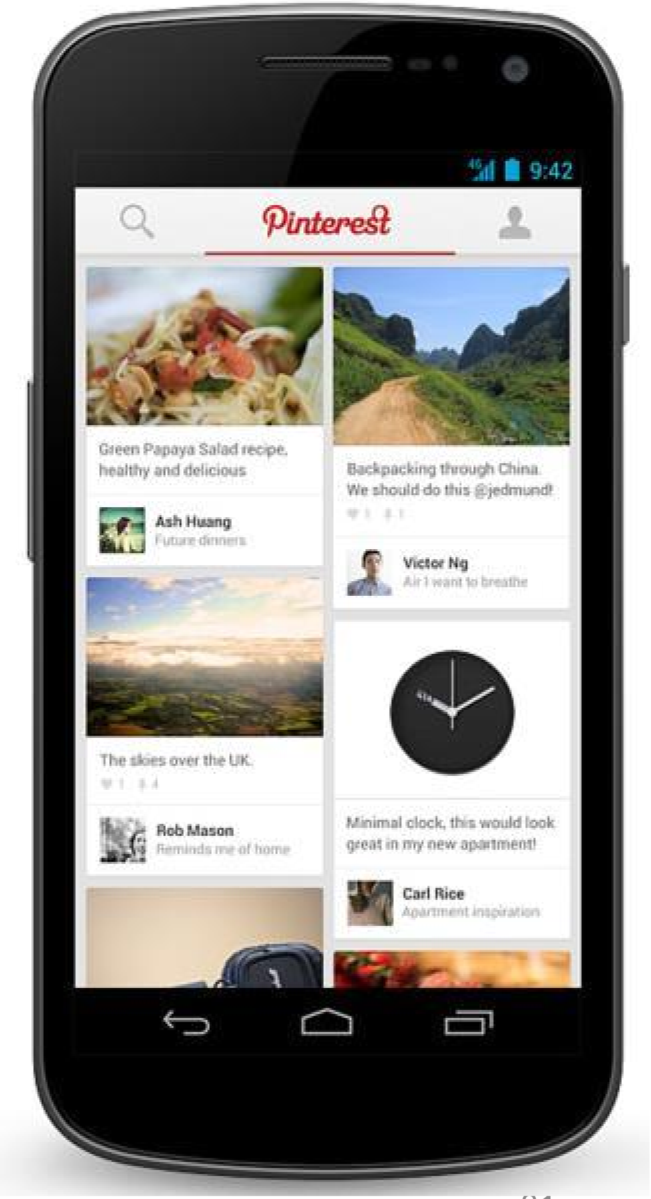
Other stuff

- Typically referred to as “assets,” anything that isn’t code is placed in the res/ folder
- String
- Music
- Images
- Some static data files

- Why?

A Concrete Example

- res/layout: layout, dimensions (width, height) of screen cells are specified in XML file here
- res/drawable-xyz/: The images stored in jpg or other format here
- java/: App’s response when user clicks on a selection is specified in java file here
- AndroidManifext.XML: Contains app name (Pinterest), list of app screens, etc



Green Papaya Salad recipe, healthy and delicious

Ash Huang
Future dinners

The skies over the UK.

Rob Mason
Reminds me of home

Backpacking through China. We should do this @jedmund!

Victor Ng
Air I want to breathe

Minimal clock, this would look great in my new apartment!

Carl Rice
Apartment inspiration

More about Activity Cycles

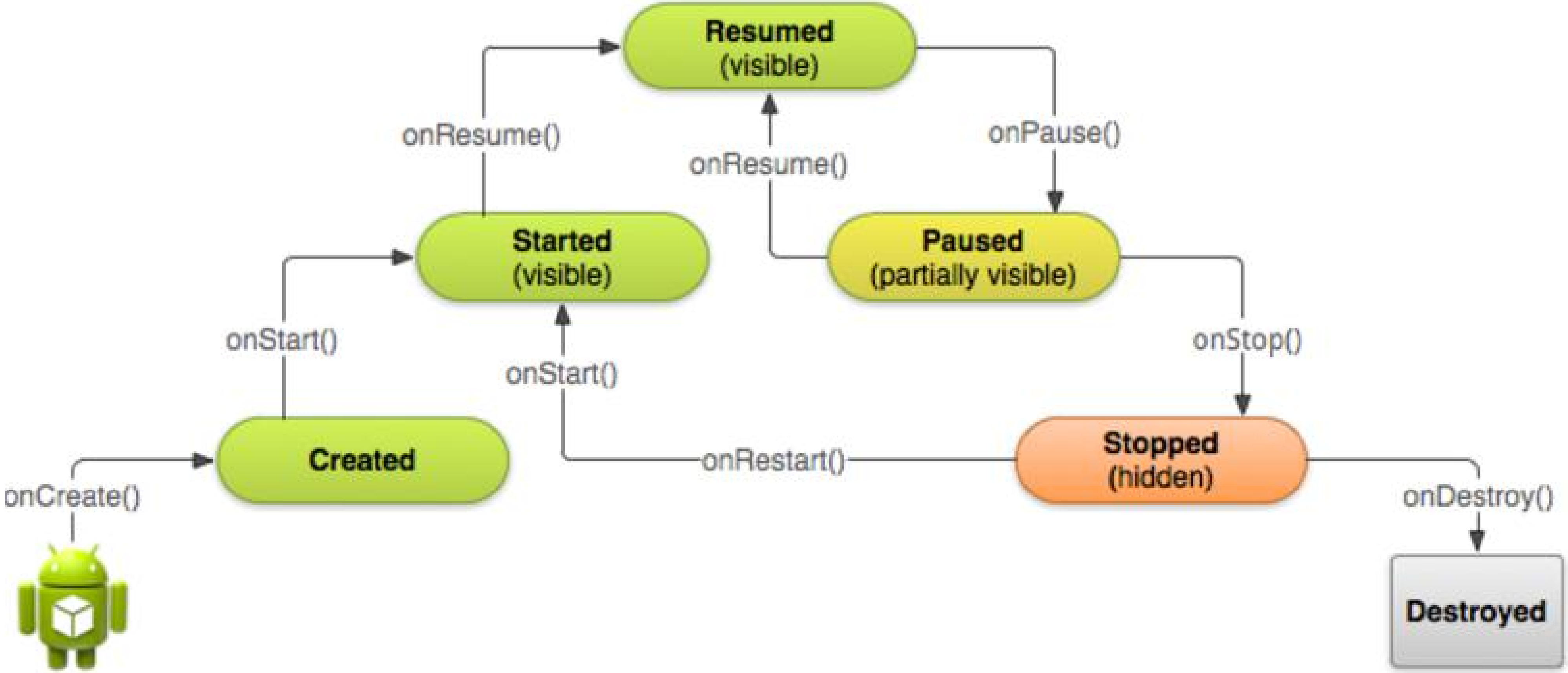
Understand Activities

- Example: login activity and display email activity in an email application
- One Activity defines a single viewable screen
  - the actions, not the layout
- Learn activity life cycle
  - Need to define what to do during the lifecycle

The Android Activity Interface:

```
public class Activity extends ApplicationContext {

    protected void onCreate(Bundle savedInstanceState);

    protected void onStart();

    protected void onRestart();

    protected void onResume();

    protected void onPause();

    protected void onStop();

    protected void onDestroy();
}
```

Activity Lifecycle



- Created: The system calls onCreate when system first creates the activity
    - Perform basic application logics
    - Codes needed once for the entire activity life
- Started: onStarted() makes the activity visible, interactive to the user.
- Resumed: The activity is in the foreground and running
- Paused: Partially visible
- Stopped: invisible, but retains all states and information
- Destroyed: The system completely drops the activity during paused or stopped states

onPause()

- Example: a dialog pops up
- What to do in this function?
  - Save states
  - Release resources

OnStopped()

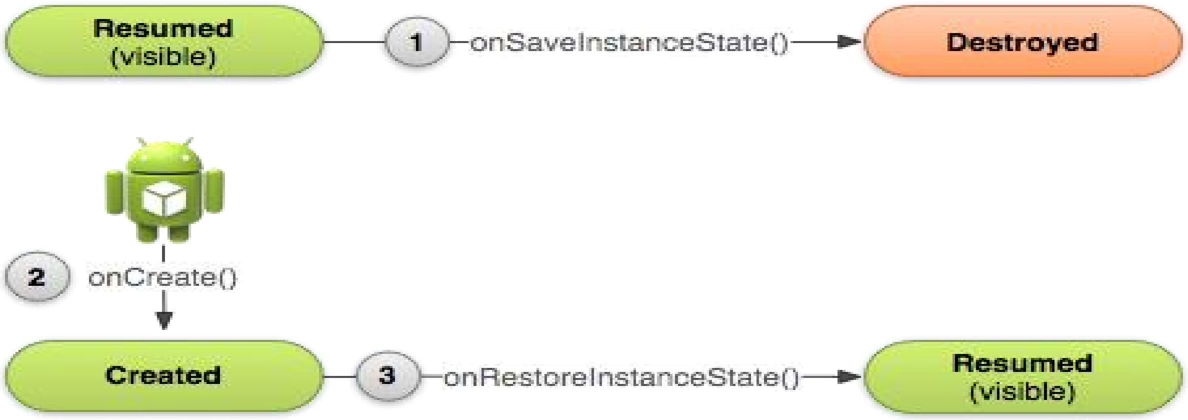
- Example: press home screen, receive a phone call

Discussion

- Discuss with your partner about the following concetps. When do these states happen? What are the differences?:
  - Started vs. Resumed?
  - Stopped vs. Paused

Recreating an Activity

- If the system destroys an activity due to system constraints
  - Saves certain state of your activity that is restored back the next time the activity starts
  - If you want to save your own state before the activity is destroyed:
    - Override the onSaveInstanceState(Bundle)
    - Access your state during onCreate() or onRestoreInstanceState()



Client/Server Architecture

Client/Server Architecture

- Mobile apps locally are great, but so much more can be done when apps are working with cloud services!
- The question is: how much processing / data should be done locally vs. in the cloud?

- Thick Client
  - Business and some data services on the phone itself
  - Good for apps that have to run “off the grid”
  - Examples?
- Thin Client
  - Most business and all data services on the server
  - Good for apps that require phone services, but does require Internet connectivity.
  - Examples? amazon
- Which is better? Depends on the app and how it’s used!
  - computation vs communication. energy, time, bandwidth (Youtube)
  - privacy vs accuracy (Nest smart thermostat)

Pros and Cons of using server

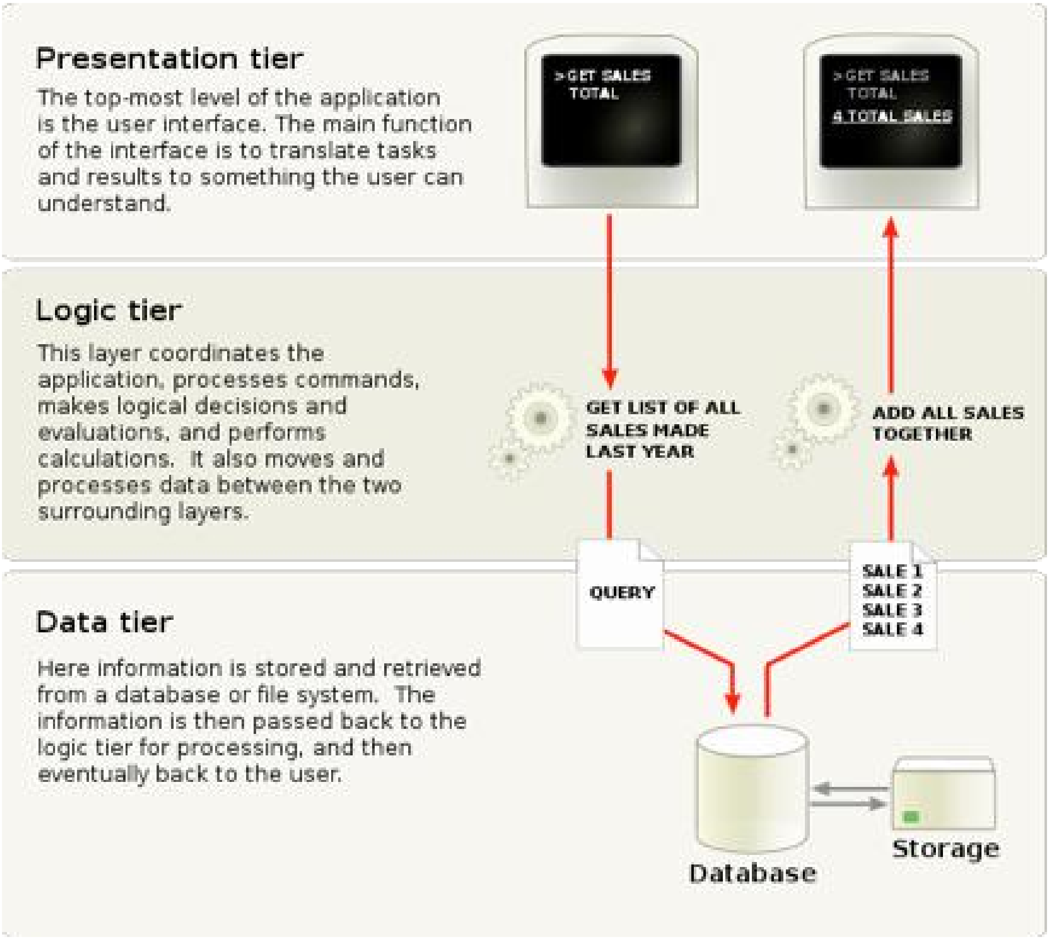
- Pros
  - computation power (e.g. movies)
  - sharing (e.g., facebook, amazon)

- Cons
  - communication cost (e.g. battery, bandwidth)
  - privacy (e.g. waze)

Design Philosophy

Mobile Application Design Philosophy

Layered Design: Model / View / Controller (MVC)



Presentation tier

The top-most level of the application is the user interface. The main function of the interface is to translate tasks and results to something the user can understand.

Logic tier

This layer coordinates the application, processes commands, makes logical decisions and evaluations, and performs calculations. It also moves and processes data between the two surrounding layers.

Data tier

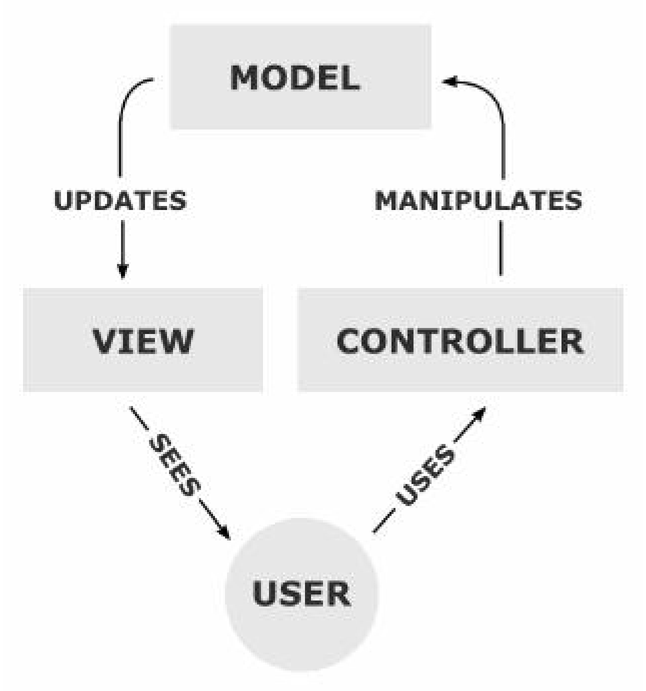
Here information is stored and retrieved from a database or file system. The information is then passed back to the logic tier for processing, and then eventually back to the user.

The Three-Tiered Architecture

- The concepts of the three-tiered architecture apply to many design scenarios
  - Keep the presentation separate so it's lightweight, easier to maintain, and can be tested separately
  - Keep the logic separate so you can change the logic as needed without having to change the presentation too much
  - Keep the data separate because you should NEVER build a system based on the current data values. Why?

Model / View / Controller

- This is the definition of what MVC is
- The MVC pattern maps:
  - Identifies what the user is asking for
  - Loads a particular resource
  - Displays the pertinent info about that resource back to the user
- To Model, Controller, View (in that order)



View

- The closest thing to what you’ve been dealing with so far is the view
- It’s effectively an HTML template that will be populated with the appropriate data from the loaded model
- All UI components go here

Controller

- The role of the controller is basically traffic cop
- It takes the request from the user and (with the assistance of the server and routing rules) turns it into a method call
- It finds the appropriate model to load
- It finds the appropriate view to load
- It returns the whole thing back to the user

Model

- The model is the representation of the data
- This may or may not be directly linked to a database (but often is in larger apps)
- A model is often translated directly into a DB table, with the columns as its attributes
- Think “class definition w/ DB backend”
- Often contains relationship rules (a Student has many Classes, for instance)