Android Application Development

Microservices , APIs

Varun Lellapalli

Agenda

1. Installing Android Studio
2. Designing a basic UI involving buttons, combo boxes, radio buttons etc.
3. Adding controls to upload, capture images
4. Basic programming to browse among screens, displaying text, labels etc.
5. Advanced programming - connecting to web service with case study of Disaster Reporting App

Microservices

1. Traditional Application Development

   Programs on Desktop Machine

   One Piece

   New Functionality → Add more code

2. Modular Application Development

   Modules - Independent parts

   Reusable , Fragmented

   When Deployed → One File (Final Executable)

Development 1 :

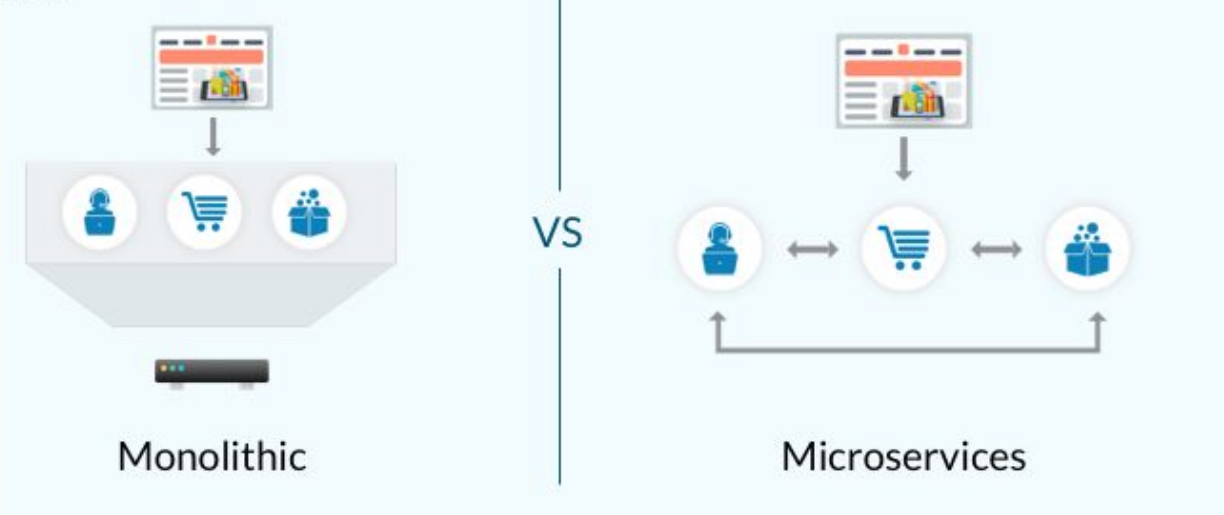
Web Applications

No Apps are deployed on your machine. Deployed on Server and Served using HTML.

→ Automatic Updates

Search Engines , Large E Commerce

As Complexity grows you also have to handle at deployment side (Not only on Development Side)



Monolithic

1. Difficult to deploy

Add a small feature -> Complete test and deploy

2. Scalability

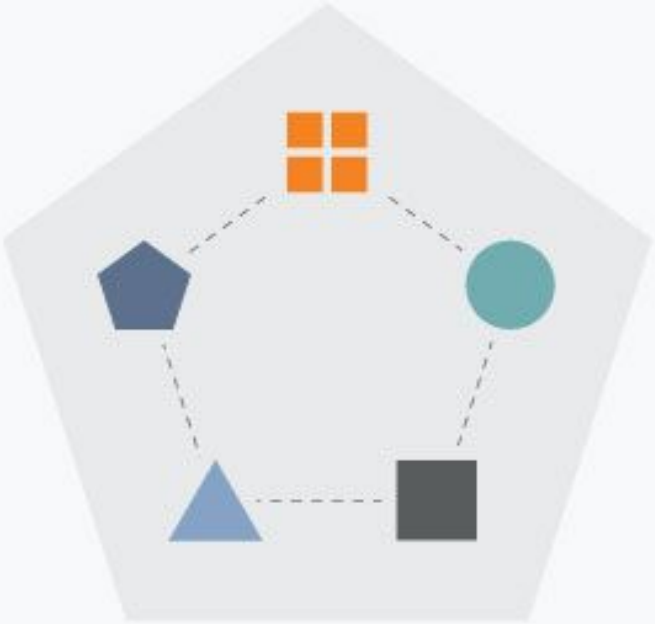
Elastic servers during offer times (complete app scaling)

Microservices

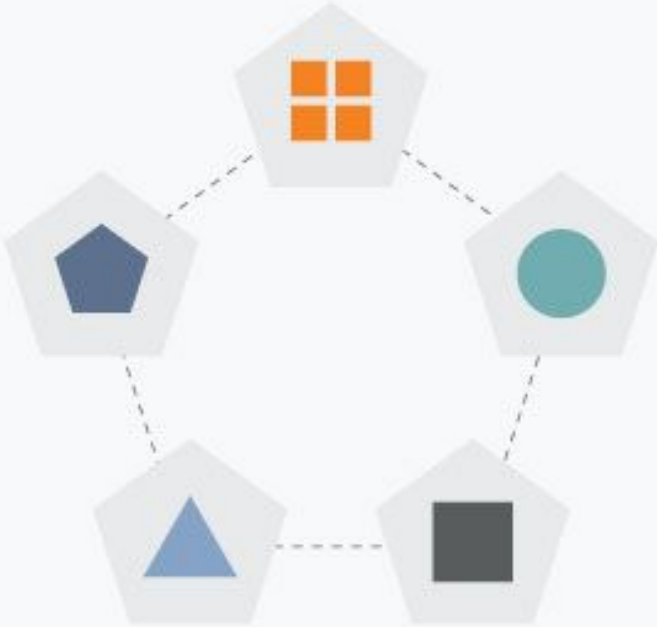
Each service in different server, Scale individual servers separately.

→ Flexibility (Language independent)[Technology , Deployment]

Monolith



Microservices



What is Android Studio

Android Studio is the official Integrated Development Environment (IDE) for Android app development, based on IntelliJ IDEA . On top of IntelliJ's powerful code editor and developer tools, Android Studio offers even more features that enhance your productivity when building Android apps, such as:

- A flexible Gradle-based build system
- A fast and feature-rich emulator
- A unified environment where you can develop for all Android devices
- Apply Changes to push code and resource changes to your running app without restarting your app
- Code templates and GitHub integration to help you build common app features and import sample code
- Extensive testing tools and frameworks
- Lint tools to catch performance, usability, version compatibility, and other problems

BusinessToday.In

NAMING HISTORY OF ANDROID RELEASES SO FAR

Android Release	Dessert Name
Android 1.5	Cupcake
Android 1.6	Donut
Android 2	Eclair
Android 2.2	Froyo
Android 2.3	Gingerbread
Android 3	Honeycomb
Android 4	Ice Cream Sandwich
Android 4.1	Jelly Bean
Android 4.4	KitKat
Android 5	Lollipop
Android 6	Marshmallow
Android 7	Nougat
Android 8	Oreo
Android 9	Pie

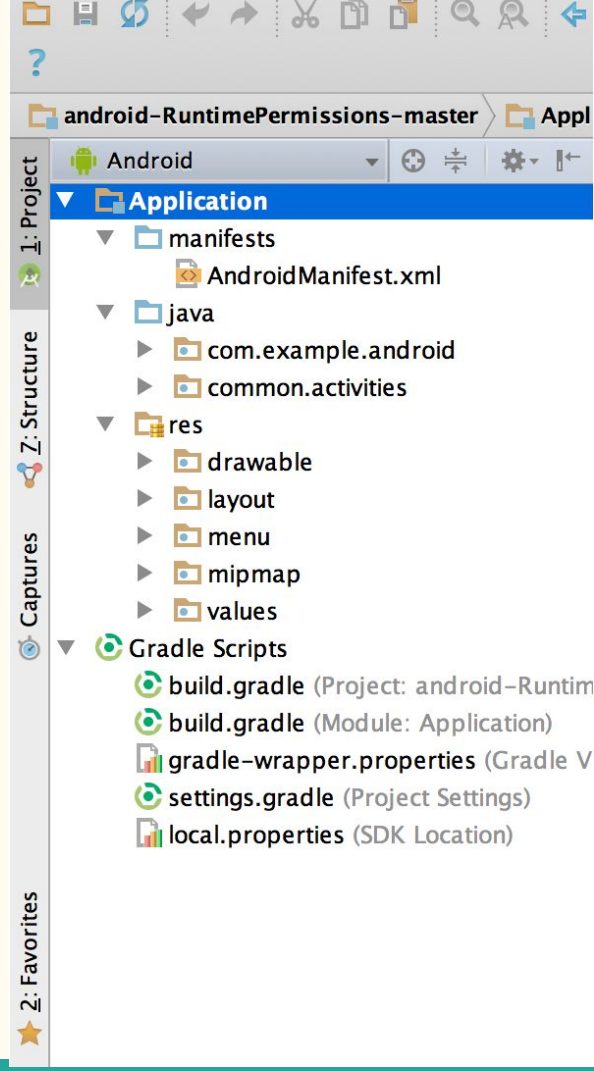
Project structure

By default, Android Studio displays your project files in the Android project view, as shown in figure 1. This view is organized by modules to provide quick access to your project's key source files.

All the build files are visible at the top level under **Gradle Scripts** and each app module contains the following folders:

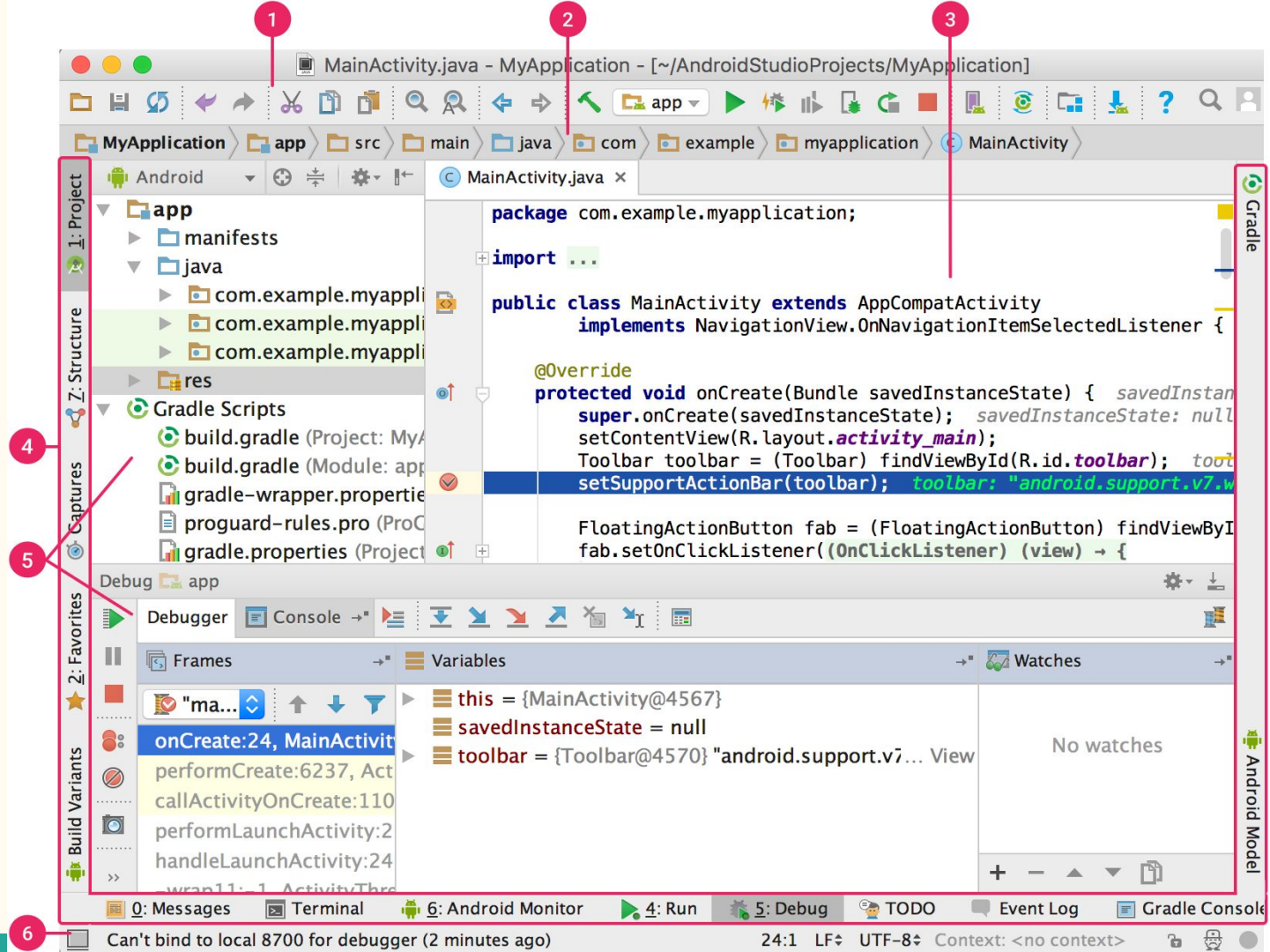
- **manifests**: Contains the `AndroidManifest.xml` file.
- **java**: Contains the Java source code files, including JUnit test code.
- **res**: Contains all non-code resources, such as XML layouts, UI strings, and bitmap images.

The Android project structure on disk differs from this flattened representation. To see the actual file structure of the project, select **Project** from the **Project** dropdown (in figure 1, it's showing as **Android**).



The User Interface

1. The **toolbar** lets you carry out a wide range of actions, including running your app and launching Android tools.
2. The **navigation bar** helps you navigate through your project and open files for editing. It provides a more compact view of the structure visible in the **Project** window.
3. The **editor window** is where you create and modify code. Depending on the current file type, the editor can change. For example, when viewing a layout file, the editor displays the Layout Editor.
4. The **tool window bar** runs around the outside of the IDE window and contains the buttons that allow you to expand or collapse individual tool windows.
5. The **tool windows** give you access to specific tasks like project management, search, version control, and more. You can expand them and collapse them.
6. The **status bar** displays the status of your project and the IDE itself, as well as any warnings or messages.


1
2
3
4
5
6
MainActivity.java - MyApplication - [~/AndroidStudioProjects/MyApplication]
app
MyApplication
app
src
main
java
com
example
myapplication
MainActivity
Android
MainActivity.java
app
manifests
java
com.example.myappli
com.example.myappli
com.example.myappli
res
Gradle Scripts
build.gradle (Project: MyA
build.gradle (Module: app
gradle-wrapper.propertie
proguard-rules.pro (ProC
gradle.properties (Project
package com.example.myapplication;
import ...
public class MainActivity extends AppCompatActivity
implements NavigationView.OnNavigationItemSelectedListener {
@Override
protected void onCreate(Bundle savedInstanceState) {
super.onCreate(savedInstanceState);
setContentView(R.layout.activity_main);
Toolbar toolbar = (Toolbar) findViewById(R.id.toolbar);
setSupportActionBar(toolbar);
FloatingActionButton fab = (FloatingActionButton) findViewByI
fab.setOnClickListener((OnClickListener) (view) → {
Debug
app
Debugger
Console
Frames
Variables
Watches
onCreate:24, MainActivit
performCreate:6237, Act
callActivityOnCreate:110
performLaunchActivity:2
handleLaunchActivity:24
this = {MainActivity@4567}
savedInstanceState = null
toolbar = {Toolbar@4570} "android.support.v7... View
No watches
1: Project
7: Structure
Captures
2: Favorites
Build Variants
Gradle
Android Model
0: Messages
Terminal
6: Android Monitor
4: Run
5: Debug
TODO
Event Log
Gradle Console
Can't bind to local 8700 for debugger (2 minutes ago)
24:1
LF
UTF-8
Context: <no context>

System requirements

Windows

- 64-bit Microsoft® Windows® 8/10
- x86_64 CPU architecture; 2nd generation Intel Core or newer, or AMD CPU with support for a Windows Hypervisor
- 8 GB RAM or more
- 8 GB of available disk space minimum (IDE + Android SDK + Android Emulator)
- 1280 x 800 minimum screen resolution

Mac

- MacOS® 10.14 (Mojave) or higher
- ARM-based chips, or 2nd generation Intel Core or newer with support for Hypervisor.Framework
- 8 GB RAM or more
- 8 GB of available disk space minimum (IDE + Android SDK + Android Emulator)
- 1280 x 800 minimum screen resolution

Linux

- Any 64-bit Linux distribution that supports Gnome, KDE, or Unity DE; GNU C Library (glibc) 2.31 or later.
- x86_64 CPU architecture; 2nd generation Intel Core or newer, or AMD processor with support for AMD Virtualization (AMD-V) and SSSE3
- 8 GB RAM or more
- 8 GB of available disk space minimum (IDE + Android SDK + Android Emulator)
- 1280 x 800 minimum screen resolution

Chrome OS

For information on recommended devices and specifications, as well as Android Emulator support, visit chromeos.dev.

Getting Android Studio

Download Android Studio and SDK tools | Android Developers

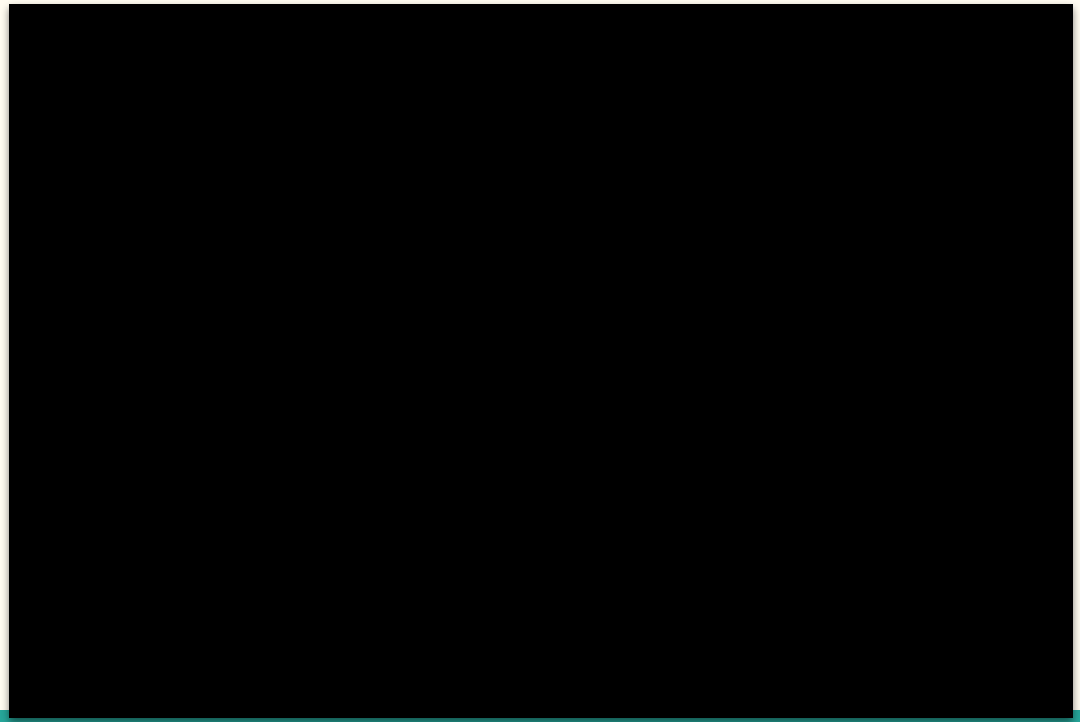
Java SE Development Kit 16 - Downloads (oracle.com)

1. Install JDK
2. Setting up Environment Variables (Path)
3. Download Android Studio

Installation Process

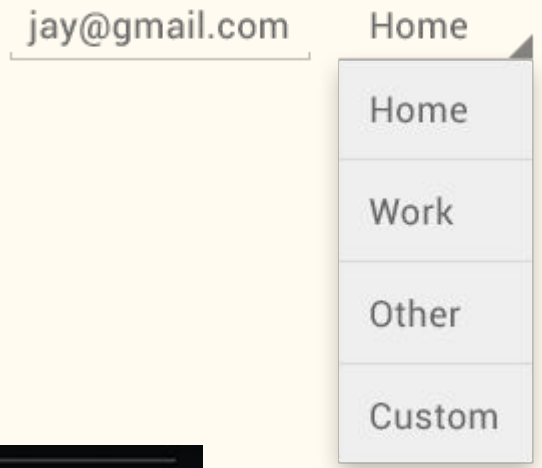
Latest Enhancement :

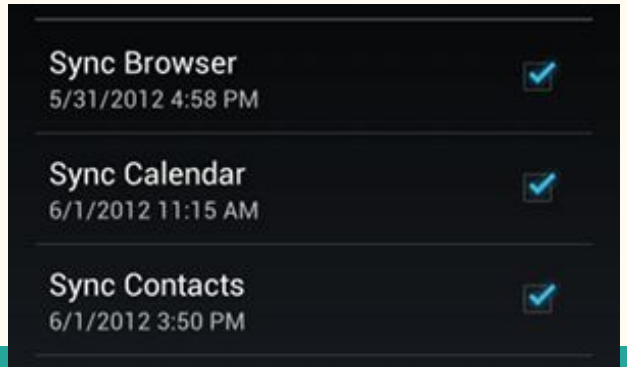
1. Android Studio + Android Virtual Device only
2. Send Usage Statistics to Google
3. Standard/Custom Installation
4. Select UI Theme : Darcula/Light
5. Verify Settings (Standard)
6. Installation Complete - Start a new Android Studio Project

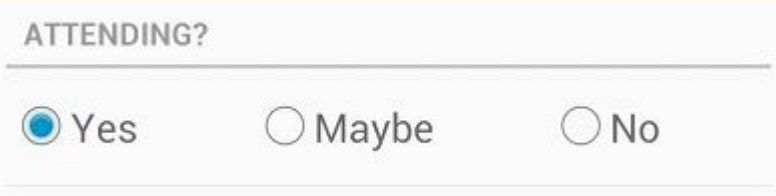


Designing Basic UI

1. Buttons
2. Checkboxes
3. Radio Buttons
4. Spinners







Buttons

```
<Button
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="@string/button_text"
    ... />
<ImageButton
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:src="@drawable/button_icon"
    android:contentDescription="@string/button_icon_desc"
    ... />
<Button
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="@string/button_text"
    android:drawableLeft="@drawable/button_icon"
    ... />
```

Method 1

```xml
<?xml version="1.0" encoding="utf-8"?>
<Button xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/button_send"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="@string/button_send"
    android:onClick="sendMessage" />
```

```java
/** Called when the user touches the button */
public void sendMessage(View view) {
    // Do something in response to button click
}
```

Method 2

```java
Button button = (Button) findViewById(R.id.button_send);
button.setOnClickListener(new View.OnClickListener() {
    public void onClick(View v) {
        // Do something in response to button click
    }
});
```

Checkboxes

```xml
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent">
    <CheckBox android:id="@+id/checkbox_meat"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/meat"
        android:onClick="onCheckboxClicked"/>
    <CheckBox android:id="@+id/checkbox_cheese"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/cheese"
        android:onClick="onCheckboxClicked"/>
</LinearLayout>
```

```java
public void onCheckboxClicked(View view) {
    // Is the view now checked?
    boolean checked = ((CheckBox) view).isChecked();
    // Check which checkbox was clicked
    switch(view.getId()) {
        case R.id.checkbox_meat:
            if (checked)
                // Put some meat on the sandwich
            else
                // Remove the meat
            break;
        case R.id.checkbox_cheese:
            if (checked)
                // Cheese me
            else
                // I'm lactose intolerant
            break;
        // TODO: Veggie sandwich
    }
}
```

Radio Buttons

```xml
<?xml version="1.0" encoding="utf-8"?>
<RadioGroup
xmlns:android="http://schemas.android.com/apk/res/andro
id"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:orientation="vertical">
    <RadioButton android:id="@+id/radio_pirates"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/pirates"
        android:onClick="onRadioButtonClicked"/>
    <RadioButton android:id="@+id/radio_ninjas"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/ninjas"
        android:onClick="onRadioButtonClicked"/>
</RadioGroup>
```

```java
public void onRadioButtonClicked(View view) {
    // Is the button now checked?
    boolean checked = ((RadioButton)
view).isChecked();

    // Check which radio button was clicked
    switch(view.getId()) {
        case R.id.radio_pirates:
            if (checked)
                // Pirates are the best
            break;
        case R.id.radio_ninjas:
            if (checked)
                // Ninjas rule
            break;
    }
}
```

Spinner

```xml
<Spinner
    android:id="@+id/planets_spinner"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
/>
```

```xml
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string-array name="planets_array">
        <item>Mercury</item>
        <item>Venus</item>
        <item>Earth</item>
        <item>Mars</item>
        <item>Jupiter</item>
        <item>Saturn</item>
        <item>Uranus</item>
        <item>Neptune</item>
    </string-array>
</resources>
```

```java
Spinner spinner = (Spinner) findViewById(R.id.spinner);
// Create an ArrayAdapter using the string array and a default spinner layout
ArrayAdapter<CharSequence> adapter = ArrayAdapter.createFromResource(this,
        R.array.planets_array, android.R.layout.simple_spinner_item);
// Specify the layout to use when the list of choices appears
adapter.setDropDownViewResource(android.R.layout.simple_spinner_dropdown_item);
// Apply the adapter to the spinner
spinner.setAdapter(adapter);
```

Intents

It is generally used with startActivity() method to invoke activity, broadcast receivers etc.

The **dictionary meaning** of intent is *intention or purpose*. So, it can be described as the intention to do action.

Implicit Intents

Implicit Intent doesn't specify the component. In such case, intent provides information of available components provided by the system that is to be invoked.

For example, you may write the following code to view the webpage.

```
Intent intent=new Intent(Intent.ACTION_VIEW);
intent.setData(Uri.parse("http://www.pb.com"));
startActivity(intent);
```

```
button.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View view) {
        String url=editText.getText().toString();
        Intent intent=new Intent(Intent.ACTION_VIEW,Uri.parse(url));
        startActivity(intent);
    } });
```

Explicit Intent

Explicit Intent specifies the component. In such case, intent provides the external class to be invoked.

Android Explicit intent specifies the component to be invoked from activity. In other words, we can call another activity in android by explicit intent.We can also pass the information from one activity to another using explicit intent.

```
Intent i = new Intent(getApplicationContext(), Activity_Two.class);
startActivity(i);
```

```
public void callSecondActivity(View view){
    Intent i = new Intent(getApplicationContext(), SecondActivity.class);
    i.putExtra("Value1", "Android By Javatpoint");
    i.putExtra("Value2", "Simple Tutorial");
    startActivity(i);
}
```

```
Bundle extras = getIntent().getExtras();
String value1 = extras.getString("Value1");
String value2 = extras.getString("Value2");
Toast.makeText(getApplicationContext(),"Values are:\n First value: "+value1+ "\n
Second Value: "+value2, Toast.LENGTH_LONG).show();
```

Camera Controls in Android Studio

```
Intent cameraIntent = new Intent(android.provider.MediaStore.ACTION_IMAGE_CAPTURE);

startActivityForResult(cameraIntent, 101);
```

```
 protected void onActivityResult(int requestCode, int resultCode, Intent data) {

  if (requestCode == CAMERA_REQUEST) {

          Bitmap photo = (Bitmap) data.getExtras().get("data");

               imageView.setImageBitmap(photo);

              }}
```

```
Intent gallery = new Intent(Intent.ACTION_PICK,
MediaStore.Images.Media.INTERNAL_CONTENT_URI);

       startActivityForResult(gallery, 202);


protected void onActivityResult(int requestCode, int resultCode, Intent data){
     super.onActivityResult(requestCode, resultCode, data);
     if (resultCode == RESULT OK && requestCode == PICK_IMAGE){
        imageUri = data.getData();
        imageView.setImageURI(imageUri);
     }

   }
```

App Demonstration & Code walkthrough

https://github.com/Varun-L/Demo_APP

Integrating WebServices in Android

Why Not

Immediacy - Websites Are Instantly Available

Compatibility - Websites are Compatible Across Devices

Upgradability - Websites Can Be Updated Instantly

Time and Cost - Websites are Easier and Less Expensive

Support and Maintenance

Framework	ExpressJS	Symfony2	Ruby on Rails	Flask	ASP.NET
Release Date	2010	2005	2004	2010	2002
Framework	MEAN Stack	Spring	MVC	WSGI	MVC
Language	JavaScript	PHP	Ruby	Python	.NET
Platform	NodeJS	PHP	Ruby	Python	Microsoft's .NET platform
Open Source	Yes	Yes	Yes	Yes	Yes* **platform is not open source*
Best Suited For	Web apps & API	Enterprise level apps	Enterprise level apps	Non - complex apps	Enterprise level apps

Volley

Volley

Volley is a HTTP library developed by Google and was first introduced during Google I/O 2013. This library is used to transmit data over the network. It actually makes networking faster and easier for Apps. It is available through AOSP(Android Open Source Project) repository.

Volley uses cache to improve the App performance by saving memory and bandwidth of remote server.

HTTP Requests :
GET , POST

Add Dependency :

```
dependencies {
    ...
    implementation
'com.android.volley:volley:1.2.0
'
}
```

Add Permission :

```
<manifest ... >
    <uses-permission
android:name="android.permission.INTE
RNET" />
    ...
</manifest>
```

```
final TextView textView = (TextView) findViewById(R.id.text);
// ...
// Instantiate the RequestQueue.
RequestQueue queue = Volley.newRequestQueue(this);
String url ="https://www.google.com";
// Request a string response from the provided URL.
StringRequest stringRequest = new StringRequest(Request.Method.GET, url,
            new Response.Listener<String>() {
    @Override
    public void onResponse(String response) {
        // Display the first 500 characters of the response string.
        textView.setText("Response is: "+ response.substring(0,500));
    }}, new Response.ErrorListener() {
    @Override
    public void onErrorResponse(VolleyError error) {
        textView.setText("That didn't work!");
    }});
// Add the request to the RequestQueue.
queue.add(stringRequest);
```

Flask App

```
from flask import Flask
app = Flask(__name__)
@app.route('/')
def entry_point():
    return 'Hello World!'
if __name__ == '__main__':
    app.run(debug=True)
```

Thank You