WIX3004
MOBILE APPLICATION DEVELOPMENT
LAYOUT

Created by S. Ong

Semester 1, 2018/2019

simying.ong@um.edu.my

VIEWGROUP
REVISION FROM LAST WEEK…

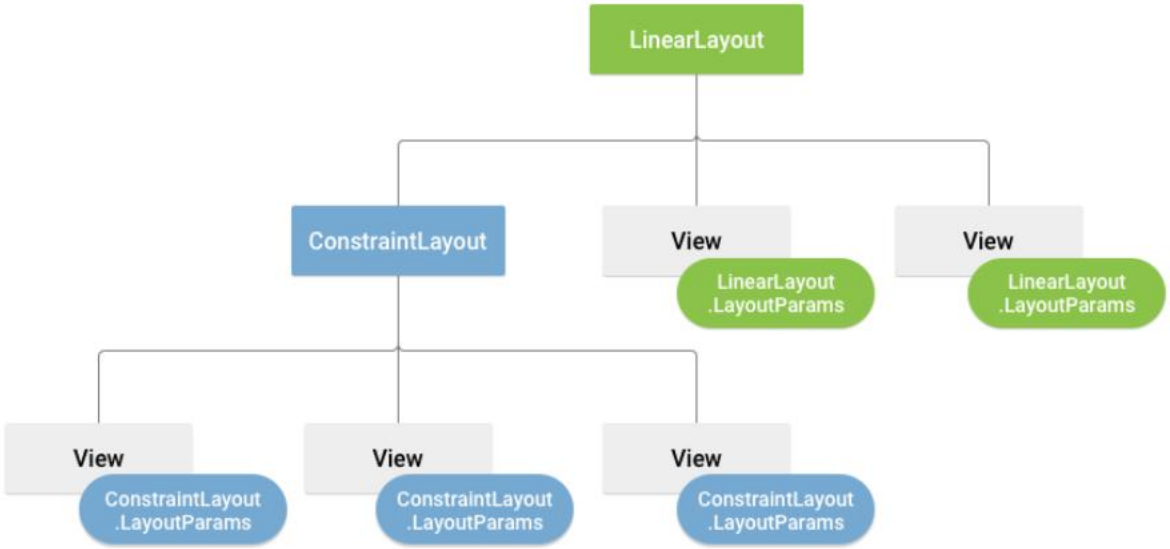
- What is ViewGroup?

LAYOUT PARAMETER

- In XML file, attributes starts with *layout_something* **defines layout parameters** for the View that are **appropriate for the ViewGroup in which it resides**.
- All ViewGroup class implement a nested class that extends *ViewGroup.LayoutParams*.
  - Contains **property types that define size and position of each child view**.

LAYOUT PARAMETER

- Parent ViewGroup defines layout parameter for all child view.



Visualization of a view hierarchy with layout parameters associated with each view.

LAYOUT PARAMETER

- Each *ViewGroup.LayoutParams* has their own syntax to set values.
- But generally, all of them have syntax to set width and height (*layout_height* and *layout_width*):
  - Specifying these settings with **absolute units** such as pixels is **not recommended**.
  - Use **relative measurement is better approach**.
  - *wrap_content*: tells your view to size itself to the dimensions required by the content.
  - *match_parent*: tells your view to become as big as its parent ViewGroup will allow.

LAYOUT POSITION

- A view has its **location**, and can be expressed using its **top and / or left coordinates**.
- Expressed in **pixels**.
- Possible methods to retrieve a View's location:
  - *getLeft*(): returns left or X coordinate of the rectangle representing the view.
  - *getTop*(): returns top or Y coordinate of the rectangle representing the view.
  - Other methods: *getBottom*() and *getRight*()

SIZE, MARGIN AND PADDING

- Size of a view is expressed by **weight and height**.
- Two pairs of weight and height values for each view:
  - **Measured width** (*getMeasuredWidth*()) and **measured height** (*getMeasuredHeight*()): define how big a view wants to be within its parent.
  - **Width** (getWidth()) and **height** (getHeight()) (drawing width and drawing height): define actual size of the view on screen.
  - These two pairs may or may not be different.

SIZE, MARGIN AND PADDING

- Padding is included into a view's dimension.
- Expressed in **pixels** for the **left, top, right, bottom parts of the view**.
- Can also be used as **offset** to the content.
- Define and retrieve padding:
  - *setPadding (int, int, int, int)*
  - *getPaddingLeft(), getPaddingTop(), getPaddingRight(), getPaddingBottom()*.
- View **does not provide support to margin**.
  - But ViewGroup does.
- Difference between margin and padding?

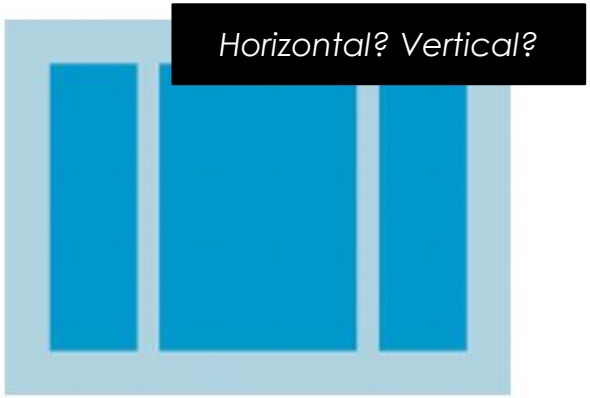
MARGIN VS PADDING

COMMON LAYOUT

- Each class of ViewGroup provides a **unique way to display the view you nest within it**.
- Some common layout:
  - Linear Layout
  - Relative Layout
  - Constraint Layout
  - Etc.

Note: Although you can nest one or more layouts within another layout to achieve your UI design, you should strive to keep your layout hierarchy as shallow as possible. Your layout draws faster if it has fewer nested layouts (a wide view hierarchy is better than a deep view hierarchy).

COMMON LAYOUT
LINEAR LAYOUT



- Organize its children into a **single horizontal or vertical row**.
- Create scrollbar if the length of the window exceeds the length of the screen.

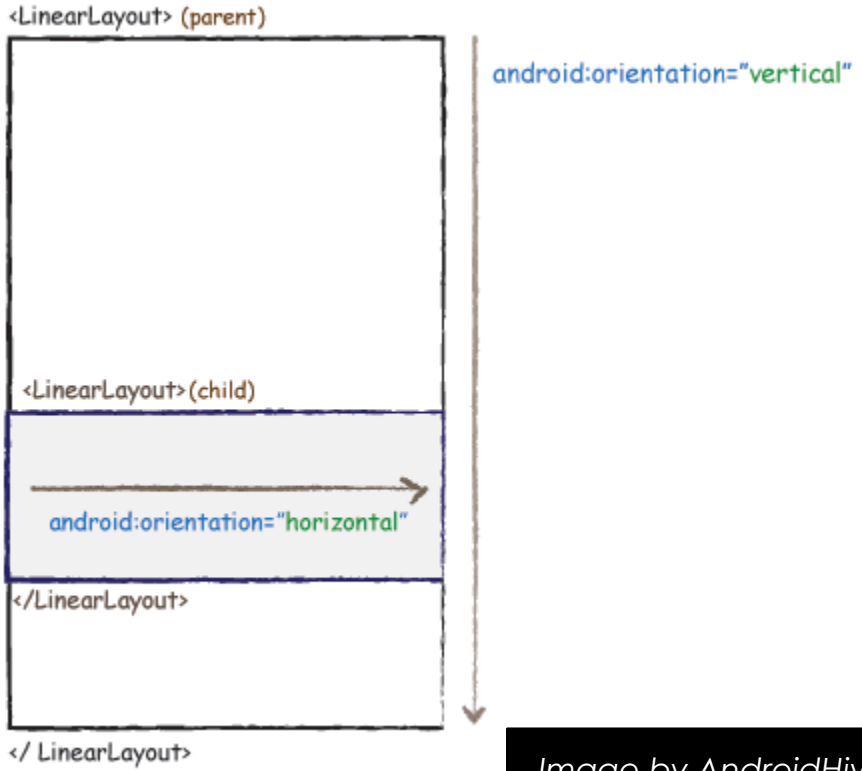


Image by AndroidHive

COMMON LAYOUT
LINEAR LAYOUT

```
<?xml version="1.0" encoding="utf-8"?>
<!-- Parent linear layout with vertical orientation -->
<LinearLayout
  xmlns:android="http://schemas.android.com/apk/res/android"
  android:orientation="vertical"
  android:layout_width="match_parent"
  android:layout_height="match_parent">

  <TextView android:layout_width="fill_parent" android:layout_height="wrap_content"
            android:text="Email:" android:padding="5dip"/>

  <EditText android:layout_width="fill_parent" android:layout_height="wrap_content"
            android:layout_marginBottom="10dip"/>

  <Button android:layout_width="fill_parent" android:layout_height="wrap_content"
            android:text="Login"/>

  <!-- Child linear layout with horizontal orientation -->
  <LinearLayout android:layout_width="fill_parent"
                      android:layout_height="wrap_content"
              android:orientation="horizontal" android:background="#2a2a2a"
              android:layout_marginTop="25dip">

  <TextView android:layout_width="fill_parent" android:layout_height="wrap_content"
         android:text="Home" android:padding="15dip" android:layout_weight="1"
         android:gravity="center"/>

  <TextView android:layout_width="fill_parent" android:layout_height="wrap_content"
         android:text="About" android:padding="15dip" android:layout_weight="1"
         android:gravity="center"/>

  </LinearLayout>

</LinearLayout>
```

What's the output if an activity is using this XML file?

Example by AndroidHive

COMMON LAYOUT
RELATIVE LAYOUT

- Specify the **location of child objects relative to each other**.
  - E.g.,: Child A to the left of Child B
- Or **to the parent**.
  - E.g., aligned to the top of parent.

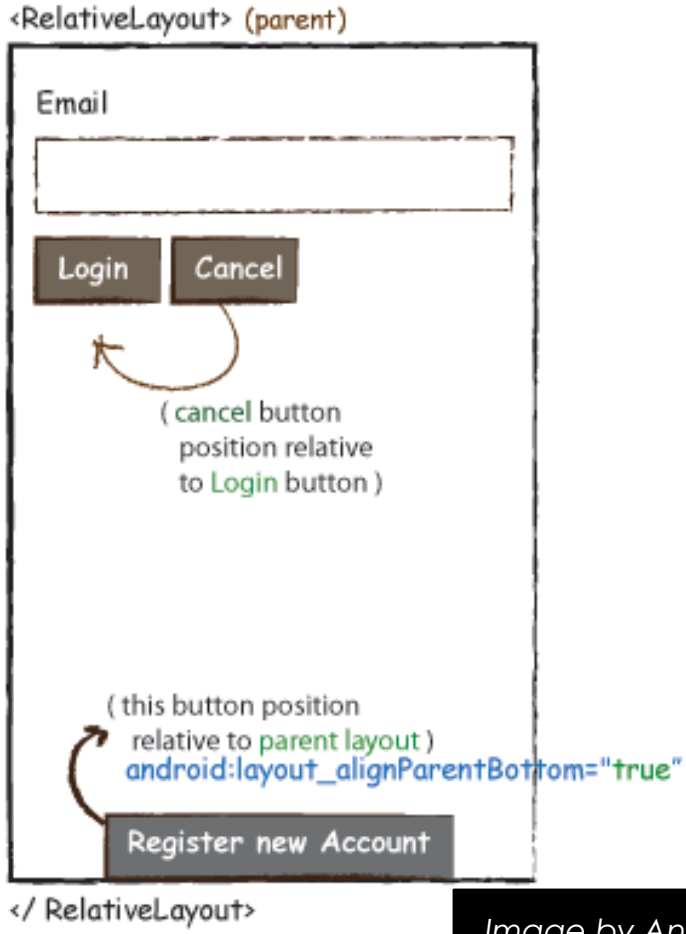


Image by AndroidHive

COMMON LAYOUT
RELATIVE LAYOUT

```
?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
                android:layout_width="fill_parent"
                android:layout_height="wrap_content">

    <TextView android:id="@+id/label" android:layout_width="fill_parent"
              android:layout_height="wrap_content" android:text="Email" />

    <EditText android:id="@+id/inputEmail" android:layout_width="fill_parent"
              android:layout_height="wrap_content" android:layout_below="@id/label" />

    <Button android:id="@+id/btnLogin" android:layout_width="wrap_content"
            android:layout_height="wrap_content" android:layout_below="@id/inputEmail"
            android:layout_alignParentLeft="true" android:layout_marginRight="10px"
            android:text="Login" />

    <Button android:layout_width="wrap_content" android:layout_height="wrap_content"
            android:layout_toRightOf="@id/btnLogin"
            android:layout_alignTop="@id/btnLogin"  android:text="Cancel" />

    <Button android:layout_width="wrap_content" android:layout_height="wrap_content"
            android:layout_alignParentBottom="true" android:text="Register new Account"
            android:layout_centerHorizontal="true"/>
</RelativeLayout>
```

Example by AndroidHive

COMMON LAYOUT
TABLE LAYOUT

- Same as HTML table layout
- **Define rows and columns** in XML file
- Cell must be added to a row in a decreasing column order.

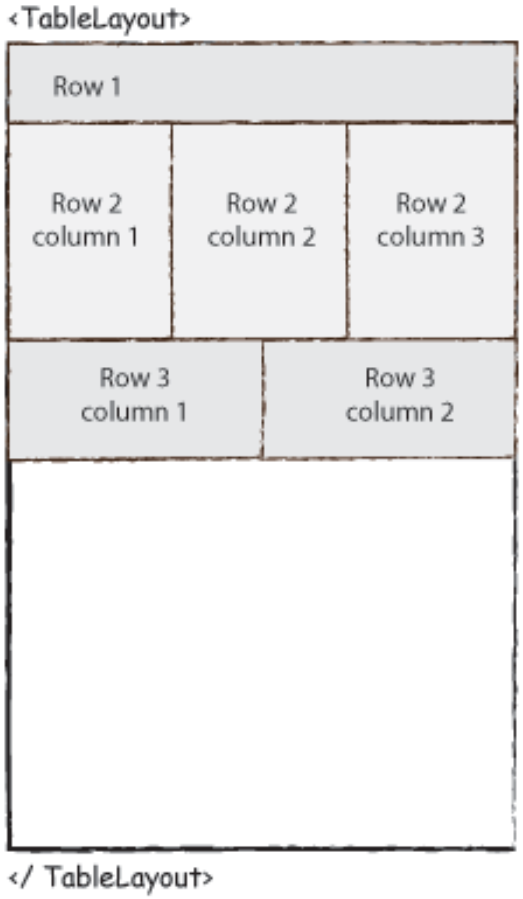


Image by AndroidHive

COMMON LAYOUT
TABLE LAYOUT

```
<TableLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:shrinkColumns="*"  android:stretchColumns="*" android:background="#ffffff">
    <!-- Row 1 with single column -->
    <TableRow
        android:layout_height="wrap_content"
        android:layout_width="fill_parent"
        android:gravity="center_horizontal">
        <TextView
            android:layout_width="match_parent" android:layout_height="wrap_content"
            android:textSize="18dp" android:text="Row 1"  android:layout_span="3"
            android:padding="18dip" android:background="#b0b0b0"
            android:textColor="#000"/>
    </TableRow>

    <!-- Row 2 with 3 columns -->
    <TableRow
        android:id="@+id/tableRow1"
        android:layout_height="wrap_content"
        android:layout_width="match_parent">
        <TextView
            android:id="@+id/TextView04" android:text="Row 2 column 1"
            android:layout_weight="1" android:background="#dcdcdc"
            android:textColor="#000000"
            android:padding="20dip" android:gravity="center"/>
```

Example by AndroidHive

COMMON LAYOUT
TABLE LAYOUT

```
        <TextView
            android:id="@+id/TextView04" android:text="Row 2 column 2"
            android:layout_weight="1" android:background="#d3d3d3"
            android:textColor="#000000"
            android:padding="20dip" android:gravity="center"/>
        <TextView
            android:id="@+id/TextView04" android:text="Row 2 column 3"
            android:layout_weight="1" android:background="#cac9c9"
            android:textColor="#000000"
            android:padding="20dip" android:gravity="center"/>
    </TableRow>

    <!-- Row 3 with 2 columns -->
    <TableRow
        android:layout_height="wrap_content"
        android:layout_width="fill_parent"
        android:gravity="center_horizontal">
        <TextView
            android:id="@+id/TextView04" android:text="Row 3 column 1"
            android:layout_weight="1" android:background="#b0b0b0"
            android:textColor="#000000"
            android:padding="20dip" android:gravity="center"/>

        <TextView
            android:id="@+id/TextView04" android:text="Row 3 column 2"
            android:layout_weight="1" android:background="#a09f9f"
            android:textColor="#000000"
            android:padding="20dip" android:gravity="center"/>
    </TableRow>

</TableLayout>
```

Example by AndroidHive

COMMON LAYOUT
CONSTRAINT LAYOUT

- Allows you to create **large and complex layout with flat hierarchy** (no nested ViewGroup needed).
- Similar to relative layout: views are all laid out according to relationship between siblings views and parent layout.
  - But more flexible
  - Easy to use with Android Studio Layout Editor
  - Remember how we do it?
- Android 2.3 and higher (Level 9)

COMMON LAYOUT
CONSTRAINT LAYOUT

- Each view needs to define **at least one horizontal and one vertical constraints**.
- Each constraint defines the connection or alignment to other view / parent layout / an invisible guideline.
- **If no constraints defined for the views in the layout, what happen?**

COMMON LAYOUT
CONSTRAINT LAYOUT

- To use constraint layout in project:
  - Ensure you declared maven.google.com in your build.gradle file:

```
repositories {
    google()
}
```

  - Add library as a dependency in the build.gradle file:

```
dependencies {
    implementation 'com.android.support.constraint:constraint-layout:1.1.2'
}
```

Note: Version might be different.

  - In the toolbar, select **Sync project with Gradle Files**.

COMMON LAYOUT
CONSTRAINT LAYOUT

- To use constraint layout in project:
  - Convert **existing layout** to constraint layout:
    - On **Design tab**, in **component tree** window, right click the layout and **Convert Layout to ConstraintLayout**.

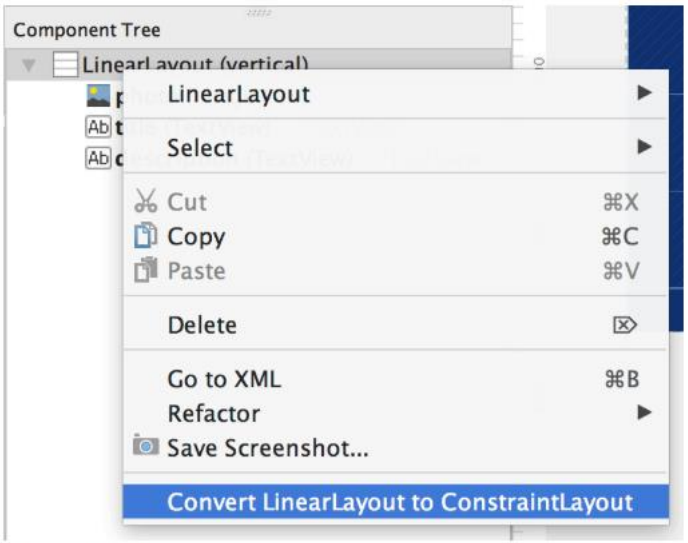


Figure 3. The menu to convert a layout to `ConstraintLayout`

COMMON LAYOUT
CONSTRAINT LAYOUT

- To use constraint layout in project:
  - Create a **new layout**:
    - **Project** window – click the module folder – select **File > New > XML > Layout XML**
    - Enter **name** for the layout file and enter *android.support.constraint.ConstraintLayout* for the **Root Tag**.
    - Click **Finish**.

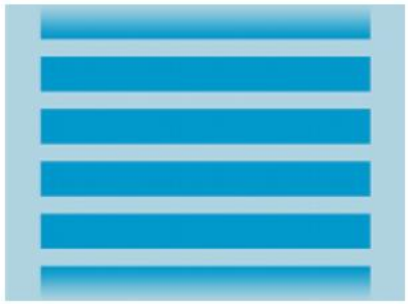
More information: https://developer.android.com/training/constraint-layout/

BUILD LAYOUT WITH ADAPTER

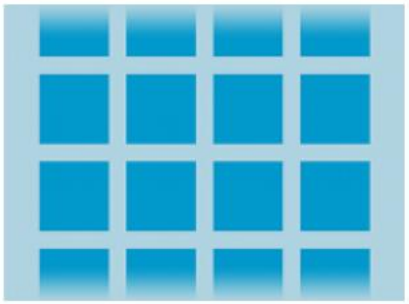
- Content of layout is **dynamic or not pre-determined**
- Use layout of subclasses **AdapterView** to **populate layout with views at runtime**.
- AdapterView class use Adapter to **bind data** to its layout. (Adapter as middleman)
- Common layout backed by Adapter includes:
  - GridView
  - ListView

BUILD LAYOUT WITH ADAPTER

- Common layout backed by Adapter includes:



List View:
displaying single scrolling column list.



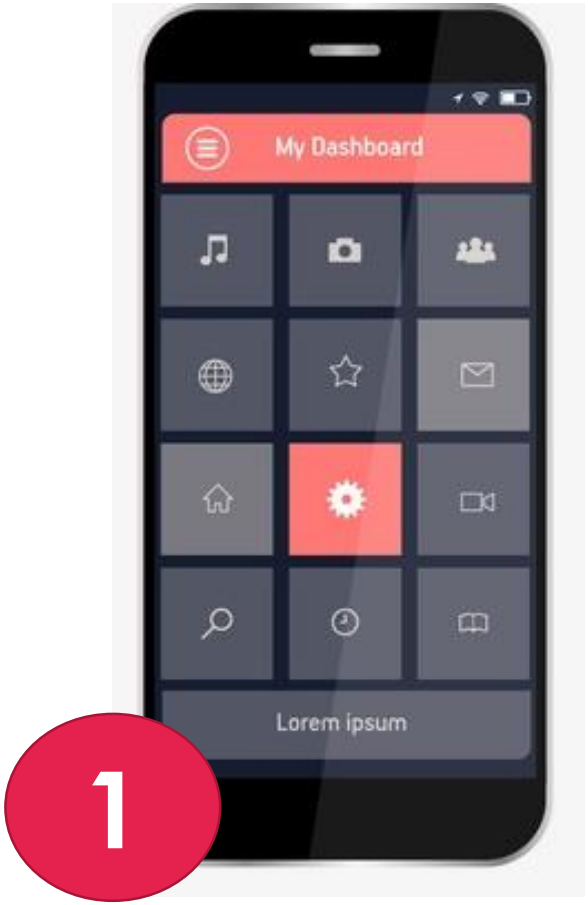
Grid View:
displaying scrolling list of columns and rows.

COMMON LAYOUT
OTHERS

- Absolute layout
- Frame layout
- Grid layout
- Etc.

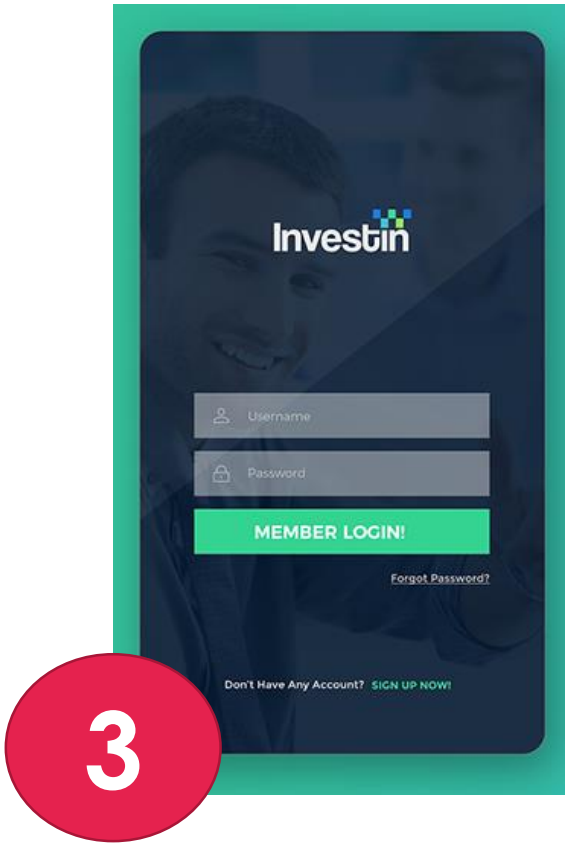
DISCUSSION

WHICH LAYOUT(S) TO USE AND WHY?



DISCUSSION

WHICH LAYOUT(S) TO USE AND WHY?



DISCUSSION

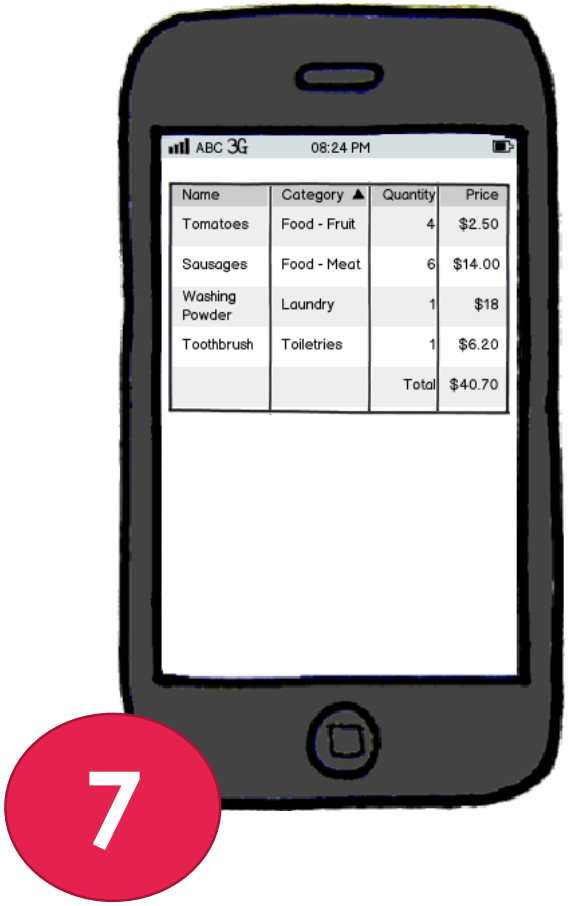
WHICH LAYOUT(S) TO USE AND WHY?

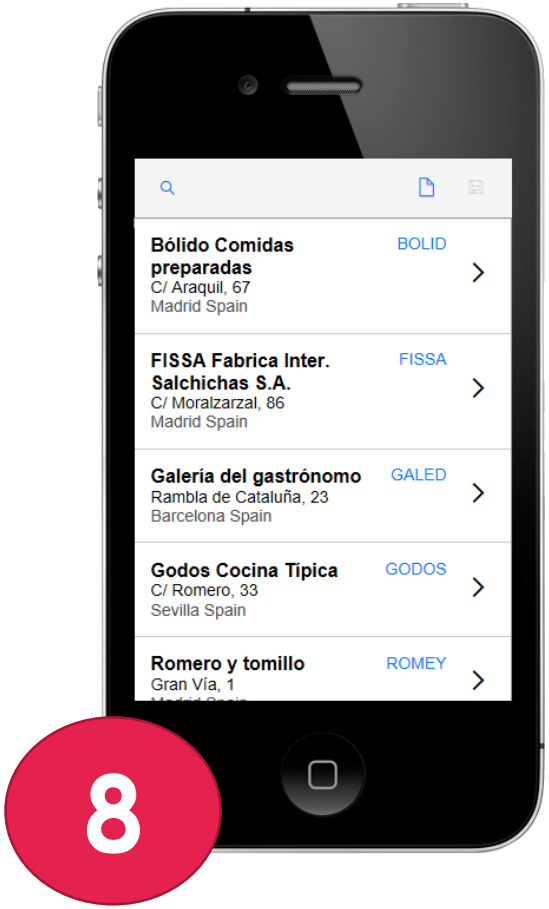
5

6

DISCUSSION

WHICH LAYOUT(S) TO USE AND WHY?





TUTORIAL QUESTIONS

1. In your opinion, which layout is more…
   a) Flexible?
   b) Easy (shorter development time)?
   c) Adaptive / Dynamic?

   Provide ONE(1) justification for each of your answer in (a) to (c) above.
2. Choose any THREE (3) layout discussed in this slide, provide TWO (2) advantages and TWO (2) disadvantages for each of the layout.
3. Debate over the use of nested layout in your mobile app UI.

ASSIGNMENT PROJECT DISCUSSION