Advanced Mobile Application Development

Recipe App: UI Design

We'll be taking a look at the UI design of our application in today's demo and see if we can make the app look a little nicer. We'll work on the views for the cells in our RecyclerViews, styles and themes, and layouts. The main goal today is to show some different possibilities for customizing the views of your app, but we certainly won't get to anything. Also note that UI design is not necessarily my strong suit so it will still be kinda ugly at the end, but it will at least be a custom type of ugly. Feel free to make it look better as you follow along

New Changes

I'll be working from the 07-ui-START branch in my repo, so be sure to check that out now.

I made some changes to the app since we left off last week:

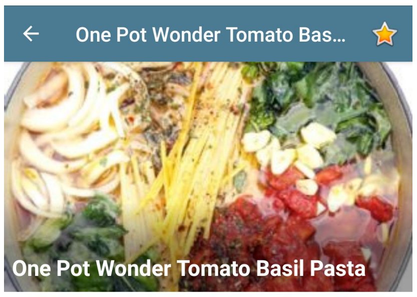
- Colors.xml (updated AppTheme in styles.xml)
- Long press to delete on Favorites list
- Large background image (we'll use this today)
- Sort favorites by date added (FavoriteDAO)
- Encode search term to avoid crash (RecipeRepo)
- Glide placeholder

RecipeDetailsFragment Style

Let's start by making the recipe details fragment look a little nicer. In reality, this screen should probably include more of the details about the recipe, but I'm sure you all could handle adding a couple more TextViews on your own time so we'll skip that for the demo.

Top Image/Title Thing

The first thing we'll do is make the image and title overlap and add an opaque gradient background to make the title pop out regardless of what the image looks like underneath. It will look like this:


One Pot Wonder Tomato Bas...
One Pot Wonder Tomato Basil Pasta

First, we need to change the size of the image we request from the server so that it will work better for our layout.

Go to *RecipeDetailFragment*
Inside the *updateViewWithDetail* Observer, **make the following change to the load method of Glide**:

```
.load("${IMAGE_BASE_URL}/${it.id}-312x231.jpg")
```

Next, we'll create some global styles to set the important attributes for our title text (large, white, bold, etc.). **Open** *style.xml* and **create the following styles**:

```
<style name="TitleText" parent="@style/TextAppearance.AppCompat.Large">
  <item name="android:textColor">@color/primaryTextColor</item>
  <item name="android:textStyle">bold</item>
</style>

<style name="RecipeDetailTitle" parent="@style/TitleText">
  <item name="android:background">@drawable/primary_gradient</item>
  <item name="android:paddingStart">8dp</item>
  <item name="android:paddingEnd">8dp</item>
  <item name="android:paddingBottom">10dp</item>
  <item name="android:paddingTop">50dp</item>
</style>
```

Styles in android should always extend an existing style to make sure that everything remains compatible. The basic process is we adopt a base style that's close to what we want and then we override the properties we'd like to change. This type of styling is very modular and easy to use throughout the entire application after you create them.

You'll see that we're using a drawable resource that doesn't exist yet so let's create it.

Ctrl-click the *res/drawables* folder and **select** *New->Drawable Resource File*
Change "Root element" to *shape* and **name it primary_gradient**
Add the following gradient:

```
<?xml version="1.0" encoding="utf-8"?>
<shape xmlns:android="http://schemas.android.com/apk/res/android">
  <gradient
     android:startColor="#00000000"
     android:centerColor="@color/opaqueDark"
     android:endColor="@color/opaqueDark"
     android:angle="270"
     />
</shape>
```

Open *recipe_detail_fragment.xml* so that we can apply our new style and fix the constraints and other layout properties to look like the following:

```
<ImageView
  android:id="@+id/recipeImageView"
  android:layout_width="0dp"
  android:layout_height="231dp"
  tools:srcCompat="@tools:sample/backgrounds/scenic"
  android:contentDescription="@string/recipe_image"
  android:scaleType="centerCrop"
  app:layout_constraintEnd_toEndOf="parent"
  app:layout_constraintStart_toStartOf="parent"
  app:layout_constraintTop_toTopOf="parent" />

<TextView
  android:id="@+id/recipeTitleTextView"
  style="@style/RecipeDetailTitle"
  android:layout_width="match_parent"
  android:layout_height="wrap_content"
  android:text="@string/title_text"
  app:layout_constraintBottom_toBottomOf="@+id/recipeImageView"
  app:layout_constraintEnd_toEndOf="parent"
  app:layout_constraintHorizontal_bias="0.0"
  app:layout_constraintStart_toStartOf="parent" />
```

Subheadings

Before we look at the RecyclerViews, **let's add some basic styling to the Ingredients and Instructions subheadings**:

```
<style name="SubtitleText" parent="@style/TextAppearance.AppCompat.Medium">
  <item name="android:textColor">@color/primaryTextColor</item>
</style>

<style name="SubtitleText.color" parent="@style/TextAppearance.AppCompat.Medium">
  <item name="android:textStyle">bold</item>
  <item name="android:textColor">@color/secondaryColor</item>
</style>
```

Be sure to apply the SubtitleText.color style to the headings in fragment_recipe_search

Instructions and Ingredients RecyclerViews

Next, we're going to add some styling to the cells in our RecyclerViews. Unfortunately, this requires a bit of a refactor to our application code due to the way our Adapter is set up, but nothing too crazy.

My design calls for some bold text for the ingredient amount/unit and the instruction number, so I went with a cell design that uses two TextViews side by side and a small underline.

Add some styles for body text:

```
<style name="BodyText" parent="@style/TextAppearance.AppCompat.Body1">
```

```
  <item name="android:textColor">@color/bodyTextColor</item>
</style>

<style name="BodyText.bold" parent="BodyText">
  <item name="android:textStyle">bold</item>
</style>
```

Open *ingredient_list_item.xml* and **make it look like the following**:

```
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout
xmlns:android="http://schemas.android.com/apk/res/android"
  xmlns:app="http://schemas.android.com/apk/res-auto"
  android:layout_width="match_parent"
  android:layout_height="wrap_content"
  android:layout_marginStart="4dp"
  android:layout_marginEnd="4dp"
  android:background="@color/opaqueSecondaryLight">

  <androidx.constraintlayout.widget.ConstraintLayout
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:layout_marginBottom="1dp"
    android:background="@color/appBG"
    app:layout_constraintBottom_toBottomOf="parent"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintHorizontal_bias="0.0"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toTopOf="parent"
    app:layout_constraintVertical_bias="1.0">

    <TextView
      android:id="@+id/detailRecyclerMainText"
      style="@style/BodyText.bold"
      android:layout_width="0dp"
      android:layout_height="wrap_content"
      android:layout_marginStart="4dp"
      android:layout_marginTop="8dp"
      android:text="TextView"
      app:layout_constraintStart_toStartOf="parent"
      app:layout_constraintTop_toTopOf="parent" />

    <TextView
      android:id="@+id/detailRecyclerDetailText"
      style="@style/BodyText"
      android:layout_width="0dp"
      android:layout_height="wrap_content"
      android:layout_marginStart="8dp"
      android:layout_marginEnd="4dp"
      android:layout_marginBottom="8dp"
      android:text="TextView"
```

```
            app:layout_constraintBottom_toBottomOf="parent"
            app:layout_constraintEnd_toEndOf="parent"
            app:layout_constraintHorizontal_bias="0.0"
            app:layout_constraintStart_toEndOf="@+id/detailRecyclerMainText"
            app:layout_constraintTop_toTopOf="@+id/detailRecyclerMainText" />

    </androidx.constraintlayout.widget.ConstraintLayout>

</androidx.constraintlayout.widget.ConstraintLayout>
```

If you want, refactor the name of this layout file to better reflect what it's used for.

In order to use this layout, I decided to add a basic data class that would make it easier to work with ingredients or instructions in the adapter.

Add the following class to *Recipe.kt*

```
data class DetailRecyclerViewItem(
    val main: String,
    val detail: String
)
```

Now let's make the changes to *DetailsRecyclerAdapter* to utilize our two TextView layout and our new model:

```
class DetailsRecyclerAdapter(val context: Context, private val detailList: List<DetailRecyclerViewItem>) :
RecyclerView.Adapter<DetailsRecyclerAdapter.ViewHolder>() {

    inner class ViewHolder(itemView: View): RecyclerView.ViewHolder(itemView) {
        val mainText: TextView = itemView.findViewById<TextView>(R.id.detailRecyclerMainText)
        val detailText = itemView.findViewById<TextView>(R.id.detailRecyclerDetailText)
    }

    override fun onCreateViewHolder(
        parent: ViewGroup,
        viewType: Int
    ): ViewHolder {
        val inflater = LayoutInflater.from(parent.context)
        val view = inflater.inflate(R.layout.detail_list_item, parent, false)
        return ViewHolder(view)
    }

    override fun getItemCount(): Int = detailList.count()

    override fun onBindViewHolder(holder: ViewHolder, position: Int) {
        val curDetail = detailList[position]

        holder.mainText.text = curDetail.main
        holder.detailText.text = curDetail.detail
    }
}
```

Lastly, we need to make some minor changes to the *updateViewWithDetails* Observer in the *RecipeDetailsFragment* to use the new model class.

Change *ingredientList* to the following:

```
//get ingredient names
val ingredientList = mutableListOf<DetailRecyclerViewItem>()
for(ingredient in it.extendedIngredients) {
  ingredientList.add(DetailRecyclerViewItem("${"%.2f".format(ingredient.amount)} ${ingredient.unit}",
ingredient.name))
}
```

Change *instructionList* to the following:

```
//setup recyclerview for instructions
val instructionList = mutableListOf<DetailRecyclerViewItem>()
for(instruction in it.analyzedInstructions[0].steps) {
  instructionList.add(DetailRecyclerViewItem("${instruction.number}.", instruction.step))
}
```

Now we can **run the app** and see the fruits of our labor when viewing the details for a recipe.

Recipe List Styling

Next, we'll add some styling to our recipe RecyclerView layout which is used in two places: for showing list of favorites, and display list of results. We'll just use the same layout for both of these places since we use the same custom adapter class, but in a real world application these would probably look a little different.

```
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout xmlns:android="http://schemas.android.com/apk/res/android"
  xmlns:app="http://schemas.android.com/apk/res-auto"
  xmlns:tools="http://schemas.android.com/tools"
  android:layout_width="match_parent"
  android:layout_height="wrap_content"
  android:layout_marginTop="8dp"
  android:clickable="true"
  android:elevation="20dp"
  android:focusable="true">
  <ImageView
    android:id="@+id/recipeListImageView"
    android:layout_width="0dp"
    android:layout_height="0dp"
    android:alpha=".9"
    android:scaleType="center"
    android:tint="#99000000"
    app:layout_constraintBottom_toBottomOf="parent"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintStart_toStartOf="parent"
```

```
        app:layout_constraintTop_toTopOf="parent"
        tools:srcCompat="@tools:sample/backgrounds/scenic" />

    <TextView
        android:id="@+id/titleTextView"
        style="@style/TitleText"
        android:layout_width="0dp"
        android:layout_height="wrap_content"
        android:layout_marginStart="8dp"
        android:layout_marginTop="24dp"
        android:layout_marginEnd="16dp"
        android:layout_marginBottom="24dp"
        android:text="TextView"
        android:textAppearance="@style/TextAppearance.AppCompat.Medium"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintEnd_toStartOf="@+id/servingsTextView"
        app:layout_constraintHorizontal_bias="0.0"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent" />

    <TextView
        android:id="@+id/servingsTextView"
        style="@style/SubtitleText"
        android:layout_width="wrap_content"
        android:layout_height="0dp"
        android:layout_marginEnd="8dp"
        android:gravity="center"
        android:text="TextView"
        app:layout_constraintBottom_toBottomOf="@+id/titleTextView"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintTop_toTopOf="@+id/titleTextView" />
</androidx.constraintlayout.widget.ConstraintLayout>
```

Then, **go to** *RecipeRecyclerAdapter* and **create a new property** in the *viewHolder* to store the reference to the ImageView

```
val image = itemView.findViewById<ImageView>(R.id.recipeListImageView)
```

Create the loading placeholder as a property of the class:

```
var glideLoading = ResourcesCompat.getDrawable(context.resources,
android.R.drawable.progress_indeterminate_horizontal, null)
```

And use Glide to load the image in *onBindViewHolder()*:

```
//runs on background thread
Glide.with(context)
   .load("$IMAGE_BASE_URL/${curRecipe.id}-312x231.jpg")
   .placeholder(glideLoading)
   .into(holder.image)
```

Run the app and take a look at results and favorites list to see the changes

Search Fragment and Main Activity

I decided to go with a basic full screen image background for the search screen and added some simple styling to the EditText and button.

EditText and Button

Open *styles.xml* and **add the following to style** Buttons and EditTexts:

```
<style name="ButtonStyle" parent="@style/Widget.AppCompat.Button">
  <item name="android:backgroundTint">@color/secondaryColor</item>
  <item name="android:textColor">@color/secondaryTextColor</item>
</style>

<style name="EditTextStyle" parent="@style/Widget.AppCompat.EditText">
  <item name="android:background">@drawable/rounded_corners</item>
  <item name="android:padding">12dp</item>
  <item name="android:elevation">20dp</item>
</style>
```

We want these styles to be applied throughout our application, so **we will add them to** *AppTheme* like so:

```
<item name="buttonStyle">@style/ButtonStyle</item>
<item name="editTextStyle">@style/EditTextStyle</item>
```

We need to create the drawable resource to use as the background
Ctrl-click on the *drawable* folder and **select** New->Drawable Resource File
Set *Root element* to *shape* and **name** it *rounded_corners*
Add the following shape to it:

```
<?xml version="1.0" encoding="utf-8"?>
<shape xmlns:android="http://schemas.android.com/apk/res/android" >
  <solid android:color="@color/opaqueWhite" />
  <stroke
     android:width="1dp"
     android:color="@color/secondaryLightColor" />
  <corners android:radius="5dp" />
</shape>
```

You can check to make sure that new styles applied properly by checking the design view of *fragment_search.xml*

Fullscreen Background Image

I wanted the background image to extend below a slightly opaque bottom nav, so I ended up adding it to the layout of the *activity_main* instead of the *fragment_search*. This will require a couple minor code changes but I think it looks nice.

First, we'll make the following changes to *activity_main.xml*

Add a background color to the top level ConstraintLayout:

```
android:background="@color/appBG"
```

Add an ImageView as the first element inside the ConstraintLayout:

```
<ImageView
  android:id="@+id/mainBgImage"
  android:layout_width="wrap_content"
  android:layout_height="wrap_content"
  android:background="#FFFFFF"
  android:contentDescription="@string/cooking_away"
  android:scaleType="centerCrop"
  app:layout_constraintBottom_toBottomOf="parent"
  app:layout_constraintEnd_toEndOf="parent"
  app:layout_constraintStart_toStartOf="parent"
  app:layout_constraintTop_toTopOf="parent"
  app:srcCompat="@drawable/bg_image" />
```

Make the background color for the BottomNavigationView slightly opaque:

```
android:background="@color/opaqueWhite"
```

Then, **open** *MainActivity.kt* and **make the following changes:**

Add a property to store the ImageView:

```
private lateinit var bgImage: ImageView
```

Instantiate in onCreate():

```
bgImage = findViewById(R.id.mainBgImage)
```

And then **add the following if/else block inside** *navControllerListener* to show/hide the background image depending on what fragment is currently being displayed:

```
if(destination.id == R.id.navigation_search) {
  bgImage.animate().alpha(1f).duration = 200
} else {
  bgImage.animate().alpha(0f).duration = 200
}
```

Run the app and make sure everything works!

In the next demo, we're going to go back to our data and integrate Firebase for cloud data persistence.