CSCI3310 Mobile Computing & Application Development

Lab 02 –Master-Detail Navigation

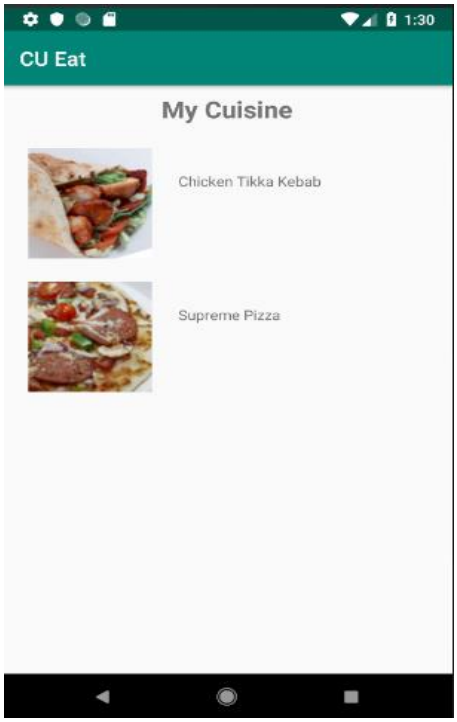
Introduction

Recall the fact that the user interface (UI) that appears on a screen of an Android-powered device consists of a hierarchy of objects called *views*. Every element of the screen is a `View`, which is the base class for classes that provide interactive UI components, such as `Button` elements.

View represents the basic building block for all UI elements. One can turn any `View`, such as an `ImageView`, into a UI element that can be tapped or clicked. In this lab, you learn how to use images as elements that the user can tap or click, and navigate to another screen.

Objectives

1) How to use an image as an interactive element to perform an action.
2) How to set attributes for `ImageView` elements in the layout editor.
3) How to add an `onClick()` method via XML or code to launch another screen.



In this lab, you create and build a new app starting with the Empty Activity template that imitates cuisine details displaying the app. The user can tap an image to perform an action—in this case, display a `Toast` message proceed to the next `Activity`.

1. Add images to the layout

You can make a view clickable, as a button, by adding the `android:onClick` attribute in the XML layout. For example, you can make an image act as a button by adding `android:onClick` to the `ImageView`.

In this task, you create a prototype of an app for displaying cuisines details. After starting a new project based on the Empty Activity template, you modify the "Hello World" `TextView` with appropriate text and add images that the user can tap.

1.1 Start the new project

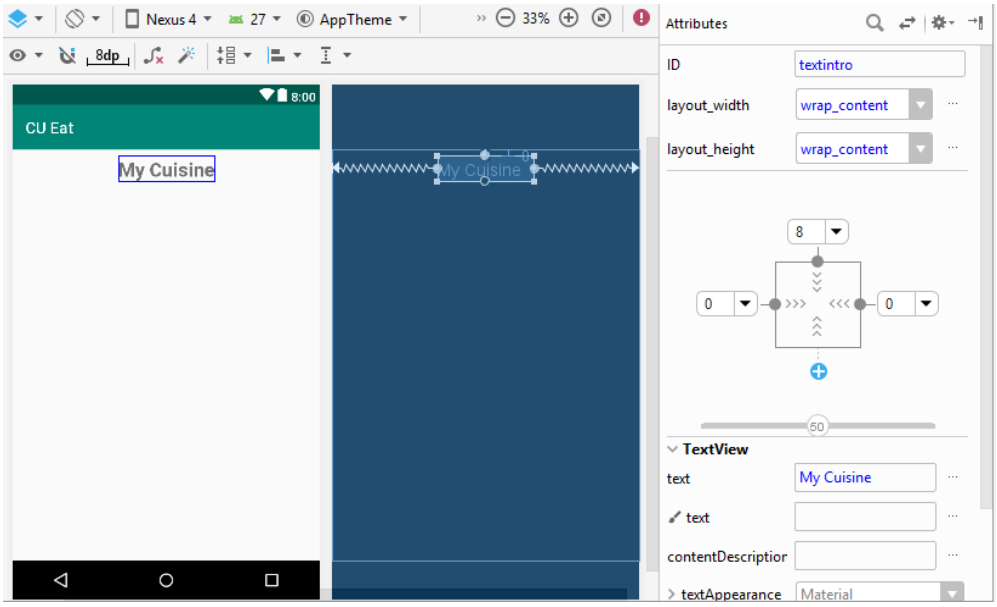
1) Start a new Android Studio project in Java with the app name **CU Eat**.
2) Choose the **Empty Activity** template, and accept the default `Activity` name (`MainActivity`) and click **Finish**.
3) Open **activity_main.xml**, select the "Hello World" `TextView` via the layout editor and open the **Attributes** pane.
4) Give an ID of textintro with extra attributes as follows:

Attribute field	Enter the following:
ID	textintro
text	Change Hello World! to My Cuisine
textStyle	B (bold)
textSize	24sp

5) Delete the constraint that stretches from the bottom of the textintro `TextView` to the bottom of the layout,

so that the `TextView` snaps to the top of the layout;

6) Choose 8 (8dp) for the top margin of the `TextView`.



1.2 Add the images

Two images (`kebab_tn.png`, and `pizza_tn.png`) are provided for this example, which you can download from Blackboard. As an alternative, you can substitute your own images as PNG files, but they are assumed be sized at about 113 x 113 pixels for best results.

1) Copy the image files into your project's drawable folder.

Find the drawable folder in a project by using this path:

```
project_name > app > src > main > res > drawable
```

2) Drag an `ImageView` to the layout, choose the kebab_tn image for it, and constrain it to the top `TextView` and to the left side of the layout with a margin of 24 (24dp) for both constraints.

3) In the **Attributes** pane, enter the following values for the attributes (You can copy/paste the text into the field.):

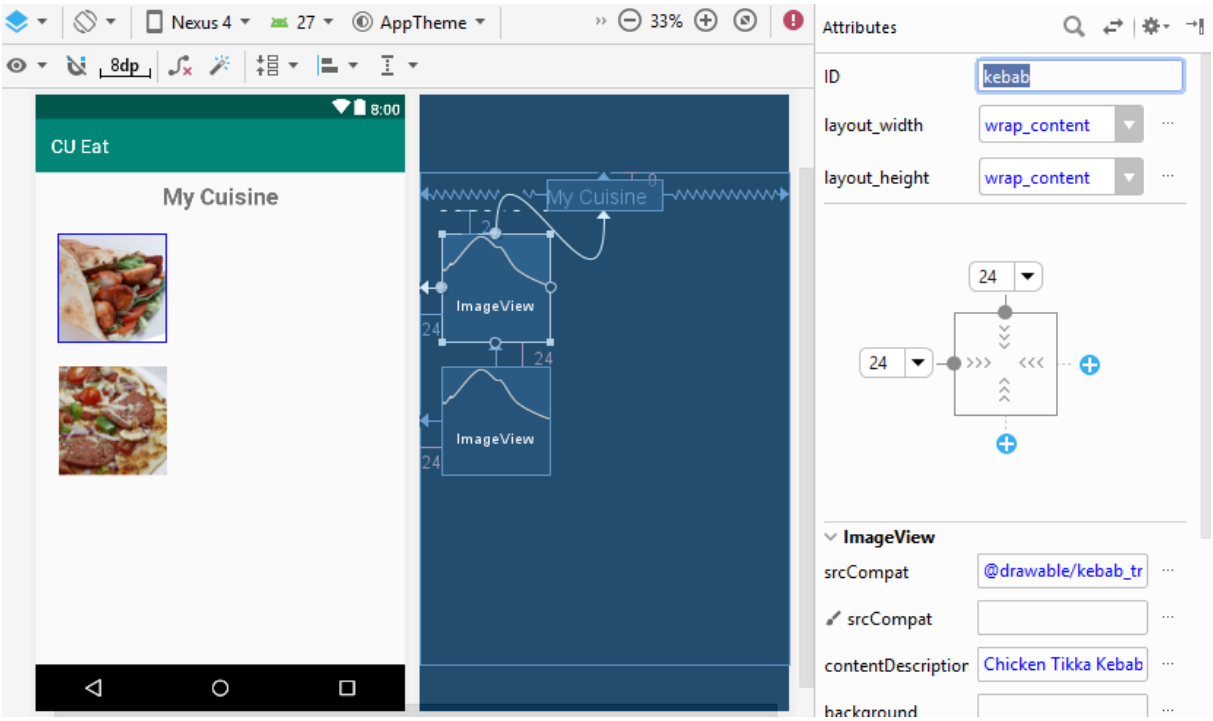
Attribute field	Enter the following:
ID	kebab
contentDescription	Chicken Tikka Kebab

4) Drag a second `ImageView` to the layout, choose the pizza_tn image for it, and constrain it to the bottom of the first `ImageView` and to the left side of the layout with a margin of 24 (24dp) for both constraints.

5) In the **Attributes** pane, enter the following values for the attributes:

Attribute field	Enter the following:
ID	pizza
contentDescription	Supreme Pizza

After all, the layout should look like the one shown below:



6) Click the warning icon ⚠ in the layout editor one by one to fix each hardcoded text warning via extracting the string.

Enter the following names for the string resources:

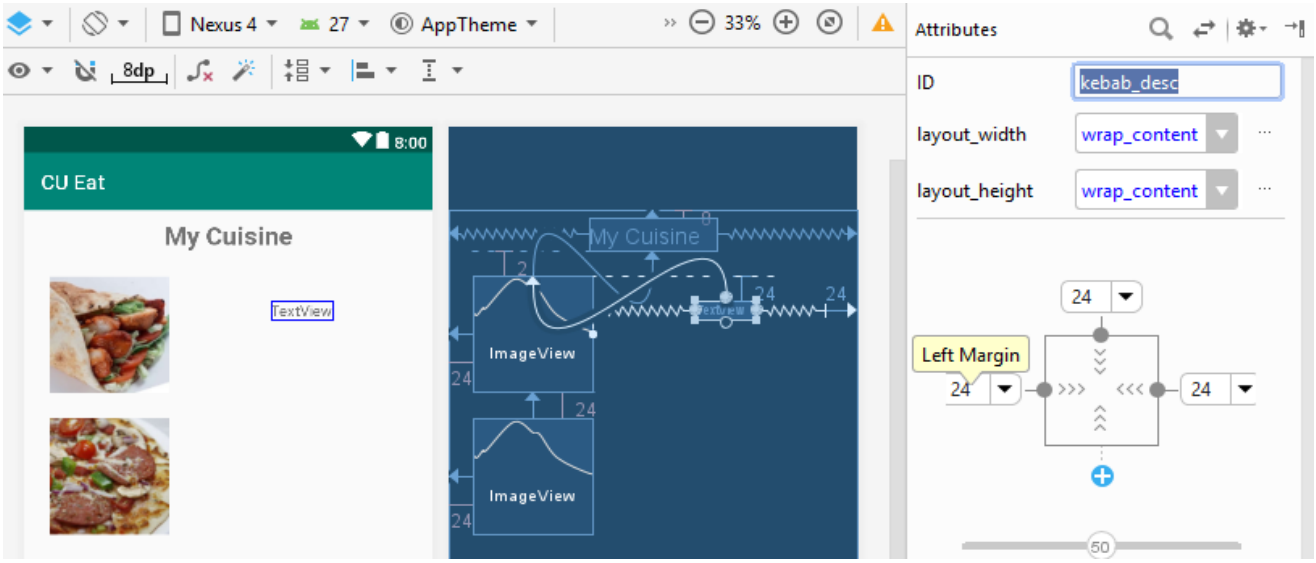
String	Enter the following:
My Cuisine	intro_text
Supreme Pizza	pizza
Chicken Tikka Kebab	kebab

1.3 Add the text descriptions

In this step, you add a text description (`TextView`) for each cuisine. Because you have already extracted string resources for the `contentDescription` fields for the `ImageView` elements, you can use the same string resources for each description `TextView`.

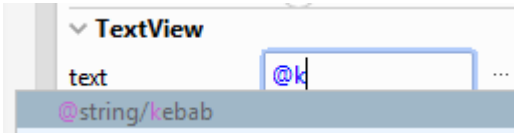
1) Drag a `TextView` element to the layout.
2) Constrain the element's left side to the right side of the kebab `ImageView` and its top to the top of the kebab `ImageView`, both with a margin of `24 (24dp)`.
3) Constrain the element's right side to the right side of the layout, and use the same margin of 24 (24dp).
4) Enter **kebab_desc** for the `ID` field in the Attributes pane.

The new `TextView` should appear next to the kebab image as shown in the figure below.



5) In the **Attributes** pane change the layout_width to **Match Constraints**.

6) In the **Attributes** pane, begin entering the string resource for the `text` field by prefacing it with the **@** symbol: @k.



Click the string resource name (@string/kebab) which appears as a suggestion.

7) Repeat the steps above to add a second `TextView`

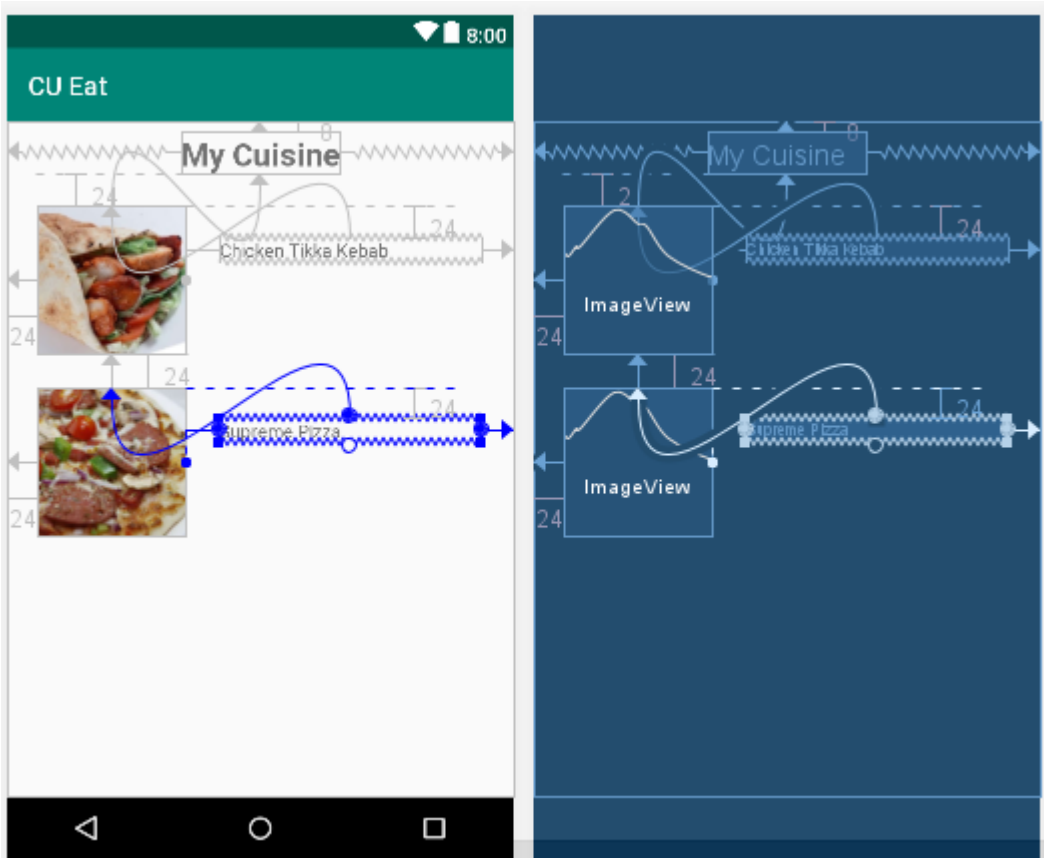
The view should be constrained to the right side and top of the pizza `ImageView`, and its right side to the right side of the layout.

8) Enter the various attributes in the **Attributes** pane.

Attribute field	Enter the following:
ID	pizza_desc
Left, right, and top margins	24

layout_width	match_constraint
text	@string/pizza

The layout should now look like the following:



2. Add onClick methods for images

To make a `View` *clickable* so that users can tap (or click) it, add the `android:onClick` attribute in the XML layout and specify the click handler.

For example, you can make an `ImageView` act like a simple `Button` by adding `android:onClick` to the `ImageView`. In this task, you make the images in your layout clickable.

2.1 Create a Toast method

In this task you add each method for the `android:onClick` attribute to call when each image is clicked. In this task, these methods simply display a `Toast` message showing which image was tapped. (In the next task, you modify these methods to launch another `Activity`.)

1) Expand **res > values** in the **Project > Android** pane, and open **strings.xml**. Add the string resources for the strings to be shown in the `Toast` message.

```
<string name="kebab_details_message">You picked a kebab.</string>
<string name="pizza_details_message">You picked a pizza.</string>
```

To use string resources in Java code, you should first add them to the `strings.xml` file.

2) Open `MainActivity`, and add a `displayToast()` method to the end of `MainActivity`

```
public void displayToast(String message) {
  Toast.makeText(getApplicationContext(), message, Toast.LENGTH_SHORT).show();
}
```

Although you could have added this method in any position within `MainActivity`, it is best practice to put your own methods *below* the methods already provided in `MainActivity` by the template.

2.2 Create click handlers

Each clickable image needs a click handler—a method for the `android:onClick` attribute to call. The click handler, if called from the `android:onClick` attribute, must be `public`, return `void`, and define a `View` as its only parameter. Follow these steps to add the click handlers:

1) Add a `showKebabDetails()` method to `MainActivity`, you can reuse the `displayToast()` method.

```
/**
Shows a message that the kebab image was clicked.
*/
public void showKebabDetails (View view) {
    displayToast(getString(R.string.kebab_details_message));
}
```

The first three lines are a comment in the Javadoc format, which makes the code easier to understand and also helps generate documentation for your code. It is a best practice to add such a comment to every new method you create. For more information about how to write comments, see How to Write Doc Comments for the Javadoc Tool.

(Optional) Choose **Code > Reformat Code** to reformat the code you added in `MainActivity` to conform to standards and make it easier to read.

2.3 Add the onClick handlers

In this step, you add `onClick` handlers to each of the `ImageView` elements in the `activity_main.xml` layout or in the code.

1) Open the **activity_main.xml** file, and click the **Text** tab in the layout editor to show the XML code.

2) Add the `android:onClick` attribute to `kebab ImageView`. As you enter it, suggestions appear showing the click handlers. Select the `showKebabDetails` click handler.

The code should now look as follows:

```
<ImageView
  android:layout_width="wrap_content"
  android:layout_height="wrap_content"
  android:padding="10dp"
  android:id="@+id/kebab"
  android:contentDescription="@string/kebab"
```

```
  android:src="@drawable/kebab_tn"
  android:onClick="showKebabDetails"/>
```

The last line (`android:onClick="showKebabDetails"`) assigns the click handler (`showKebabDetails`) to the `ImageView`.

(Optional) Choose **Code > Reformat Code** to reformat the XML code you added in `activity_main.xml` to conform to standards and make it easier to read. Android Studio automatically moves the `android:onClick` attribute up a few lines to combine them with the other attributes that have `android:` as the preface.

3) Follow the same procedure to add the `android:onClick` attribute to the `pizza ImageView` elements.

Select the `showPizzaDetails` click handlers. You can optionally choose **Code > Reformat Code** to reformat the XML code. The code could now look as follows:

```
<ImageView
    android:id="@+id/pizza"
    android:layout_width="wrap_content "
    android:layout_height="wrap_content"
    android:layout_marginStart="@dimen/margin_wide"
    android:layout_marginTop="@dimen/margin_wide"
    android:contentDescription="@string/pizza"
    android:onClick="showPizzaDetails"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toBottomOf="@+id/kebab"
    app:srcCompat="@drawable/Pizza_tn" />
```

Other than adding the click handler via the `android:onClick` attribute in the XML layout. You can also add click handler in code directly.

4) Add the another `showPizzaDetails()` method similar to that of `showKebabDetails()` method to `MainActivity` **without** adding extra line to the XML.

5) Add the following code snippet within the OnCreate() of `MainActivity`:

```
ImageView imagePizza = (ImageView) findViewById(R.id.pizza);
imagePizza.setOnClickListener(new View.OnClickListener() {
    public void onClick(View v) {
        // Do something in response to image click
        showPizzaDetails(v);
    }
});
```

6) Run the app.

3. Extend the image action

Clicking the kebab, or pizza image displays a `Toast` message about the click. For this task, you change the action for the `ImageView` to launch a new `Activity`.

3.1 Add an Activity

As you learned in various labs, an `Activity` represents a single screen in your app in which your user can perform a single, focused task. You already have one activity, `MainActivity.java`. Now you add another activity called `DetailedViewActivity.java`.

1) Right-click (or Control-click) the **edu.cuhk.csci3310.cueat** folder in the left column and choose **New > Activity > Empty Activity**.

2) Edit the **Activity Name** to be **DetailedViewActivity**, and the **Layout Name** to be **activity_details**. Leave the other options alone, and click **Finish**.

The `DetailedViewActivity` class should now be listed along with `MainActivity` in the **java** folder, and `activity_details.xml` should now be listed in the **layout** folder. The Empty Activity template added these files.

3.2 Change the action

In this step, you change the action on clicking the `ImageView` to launch the new `Activity`.

1) Open MainActivity.

2) Change the `showKebabDetails` and `showPizzaDetails` method to make an explicit intent to start `DetailedViewActivity`.

```
public void showKebabDetails (View view) {
  Intent intent = new Intent(MainActivity.this, DetailedViewActivity.class);
  startActivity(intent);
}
```

3) Run the app. Tap the image.

A blank `Activity` should appear (`DetailedViewActivity`). Tap the Back button to go back to `MainActivity`.

3.3 Sending Message through Intent

The CU Eat app's `MainActivity` launches a second `Activity` called `DetailedViewActivity`. In this task, you shall learn how to send data from an `Activity` to another `Activity`. Change the app to send the Details message for the selected cuisine in `MainActivity` to a new `TextView` at the top of the `DetailedViewActivity` layout with id `Details_textview`.

1) Add a `TextView` at the top of the `DetailedViewActivity` layout with the id `Details_textview`.

2) Create a `private` member `String` variable (`mDetailsMessage`) in `MainActivity` for the Details message that appears in the `Toast`.

```
private String mDetailsMessage;
```

3) Change the `showKebabDetails()` and `showPizzaDetails()` click handlers to assign the message string `mDetailsMessage` before displaying the `Toast`.

For example, the following assigns the `kebab_details_message` string to `mDetailsMessage` and displays the `Toast`:

```
  mDetailsMessage = getString(R.string.kebab_details_message);
  displayToast(mDetailsMessage);
```

4) Add a `public static final String` called `EXTRA_MESSAGE` as a field of `MainActivity` to define the key for an `intent.putExtra`.

```
public static final String EXTRA_MESSAGE =
                     "edu.cuhk.csci3310.cueat.extra.MESSAGE";
```

5) Change the `showKebabDetails()` and `showPizzaDetails()` click handlers to include the `intent.putExtra` statement before launching `DetailedViewActivity`.

```
public void showKebabDetails (View view) {
  Intent intent = new Intent(MainActivity.this, DetailedViewActivity.class);
  intent.putExtra(EXTRA_MESSAGE, mDetailsMessage);
  startActivity(intent);
}
```

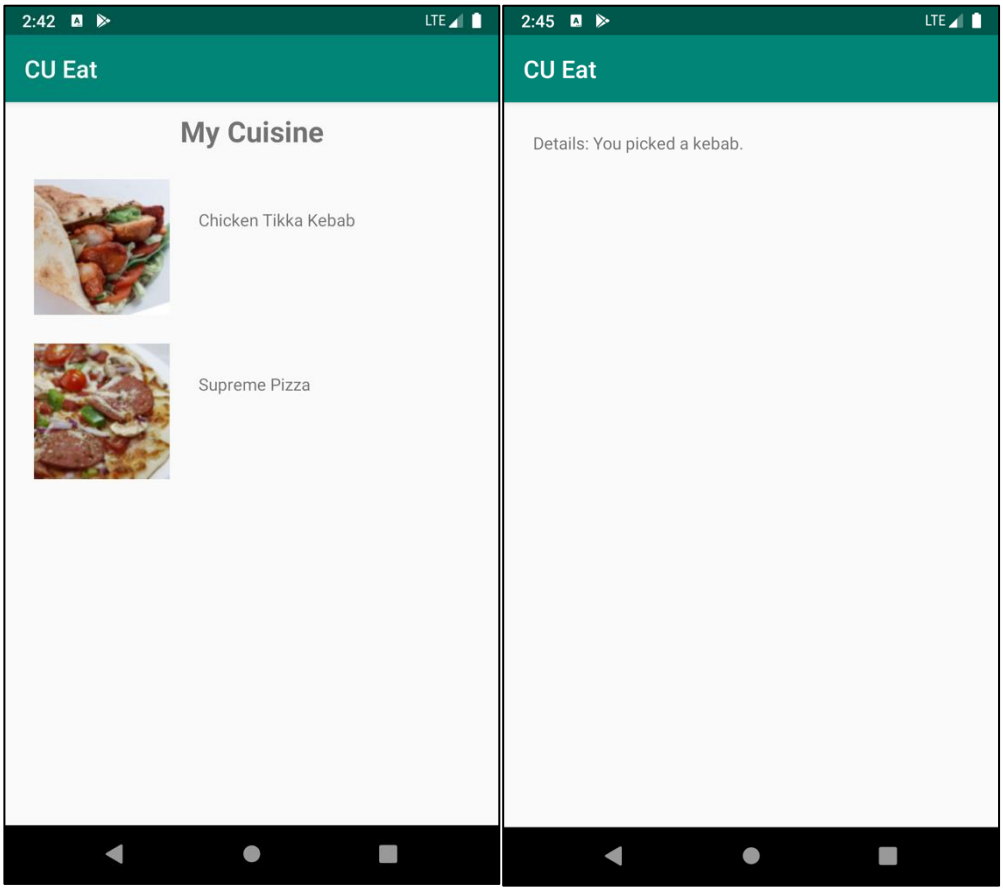
6) In `activity_details.xml`, add one `TextView` with id `Details_textview`.

7) In `DetailedViewActivity`, add code to the `onCreate()` method to get the `Intent` that launched the `Activity`, extract the string message, and replace the text in the `TextView` with the message.

```
  Intent intent = getIntent();
  String message = "Details: " +
        intent.getStringExtra(MainActivity.EXTRA_MESSAGE);
  TextView textView = findViewById(R.id.Details_textview);
  textView.setText(message);
```

8) Run the app.

After choosing a cuisine image, tap the floating action button to launch `DetailedViewActivity`, which should include the Details message as shown in the figure below.



Reference:

Android developer documentation:

1) Intents and Intent Filters
2) Activity Intent

Stack Overflow:

1) How to send data by click back button?