Core Task 2 – Spike: Picture This (Core Only)

Goals:

This section is an overview highlighting what the task is aiming to teach or upskill.

This app is based on two activities to interact with the application. On launch, the user will be greeted with four different food and some of their details. When the user clicks on any of the food images, they will be directed to a second activity which shows the full details of each individual food. This app seeks to achieve multi-activity, data sharing using intents, and using resources and images.

The following list outlines the goal broken down into more specific knowledge gaps involved in the goal.

- UI/UX design using other software
- Multi-activity
- Intents and Parcelable protocol
- Bundles
- Resources and images

Github Repo : **https://github.com/Eric-Sow/coretask2-extension-Joel-103366239**

Tools and Resources Used

This section lists related software, tools, libraries, API's, and other resources used for this knowledge gap.

- Android Studio
- Passing image from one activity to another - https://www.tutorialspoint.com/how-to-pass-an-image-from-one-activity-to-another-activity-in-kotlin
- Parcelize function - https://stackoverflow.com/questions/64925126/how-to-use-parcelize-now-that-kotlin-android-extensions-is-being-deprecated
  - https://stackoverflow.com/questions/64981310/plugin-kotlin-parcelize-not-found

Knowledge Gaps and Solutions

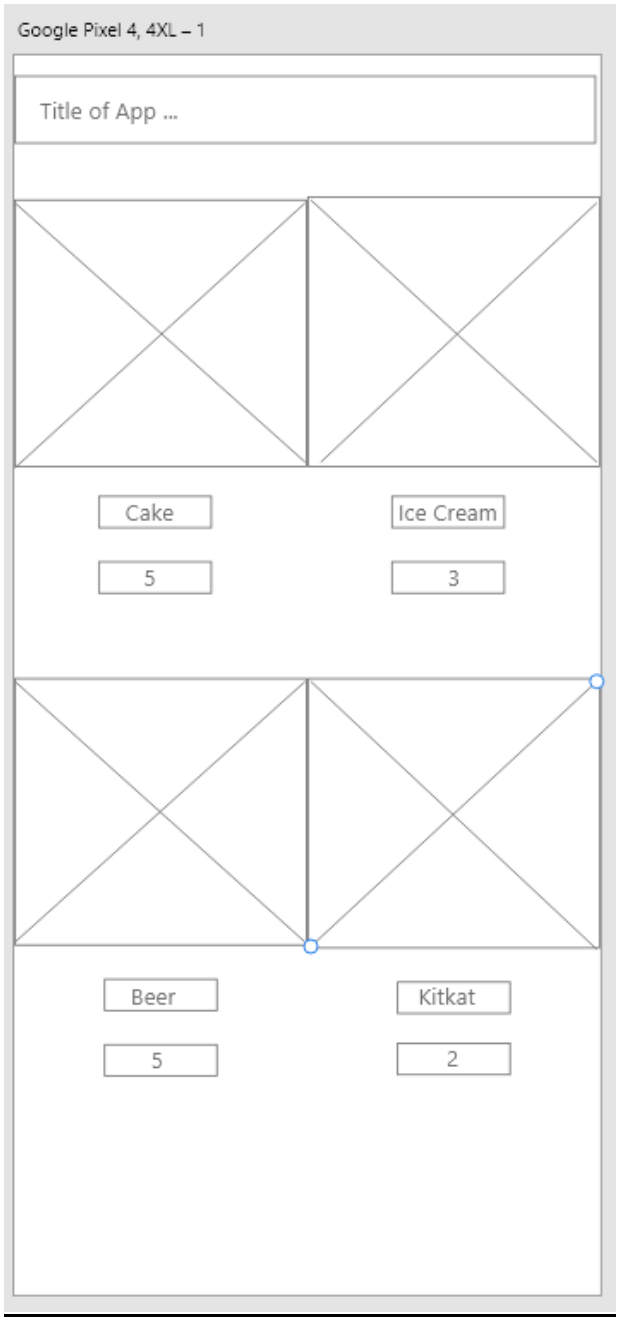
This section presents the listed knowledge gaps and their solutions with supporting images, screenshots and captions where appropriate/required.

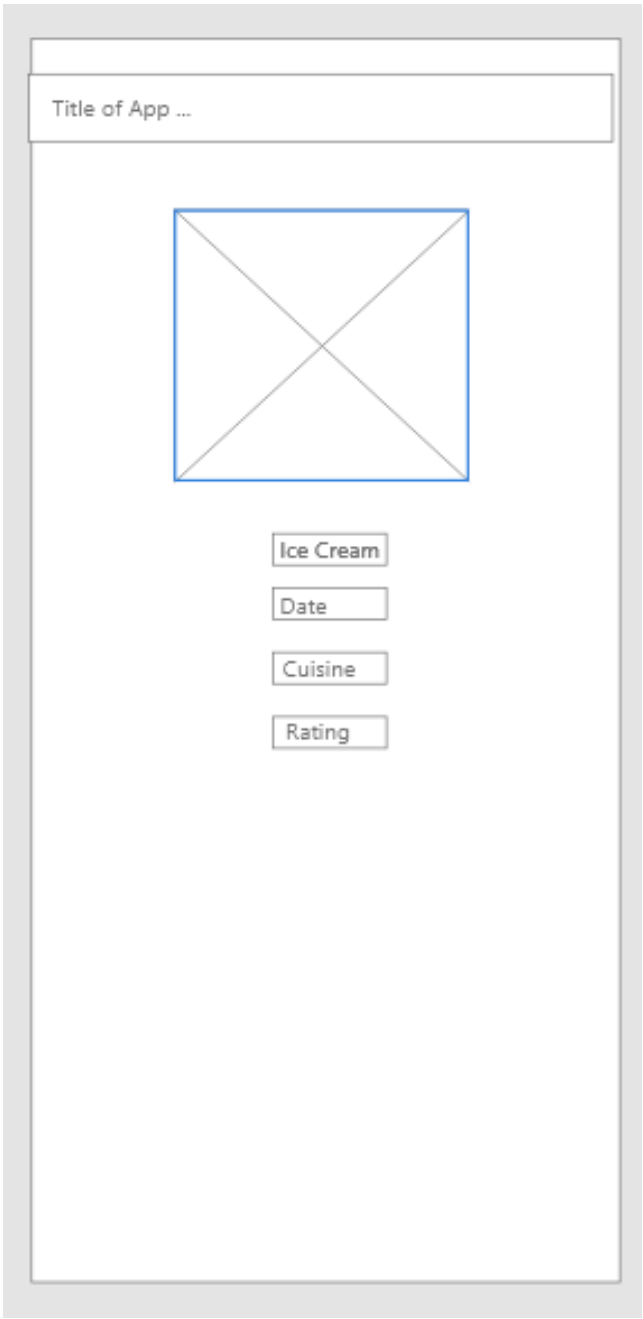
Gap 1: UI/UX Design

As this task seeks to test the ability of using third party design software to make rough sketches of the user interface, a sample of a sketch will be attached below. The sample is drawn using Adobe XD as it is popular and widely used among designers. In other units, this software has been used, therefore, I have some experience with using it. Lastly, Swinburne provides access to this software, so it will be a good choice to use it also.

Previous topics covered UI/UX design roughly as this unit is about mobile app development therefore a rough sketch is used to represent the design.

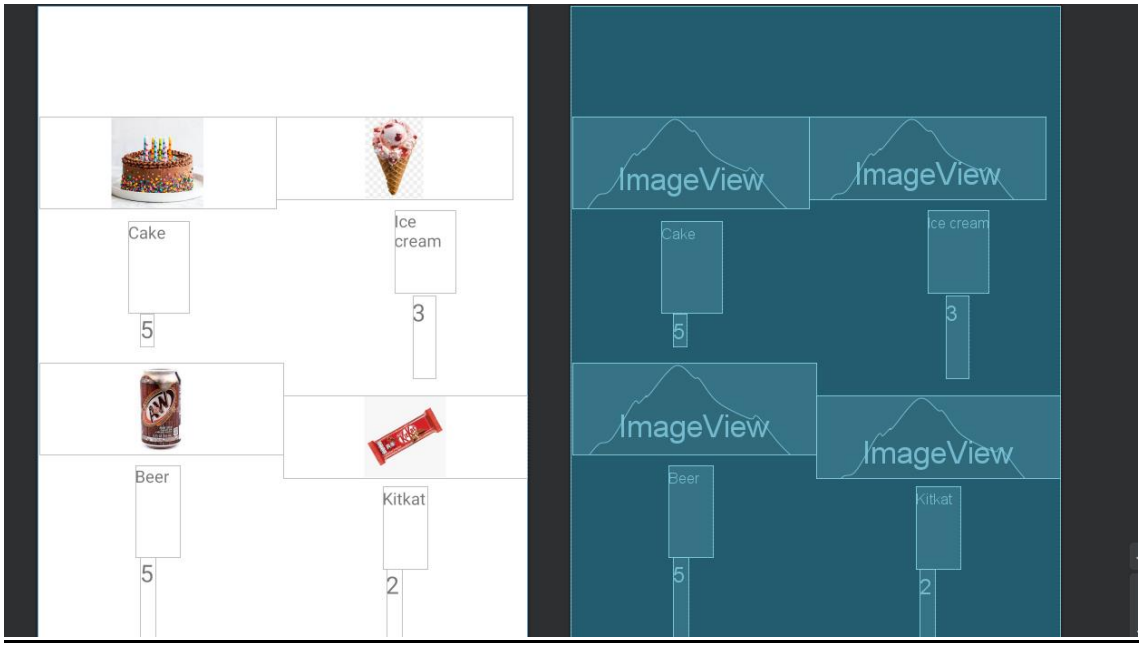
UI Design (Adobe XD) – Wireframe/sketch

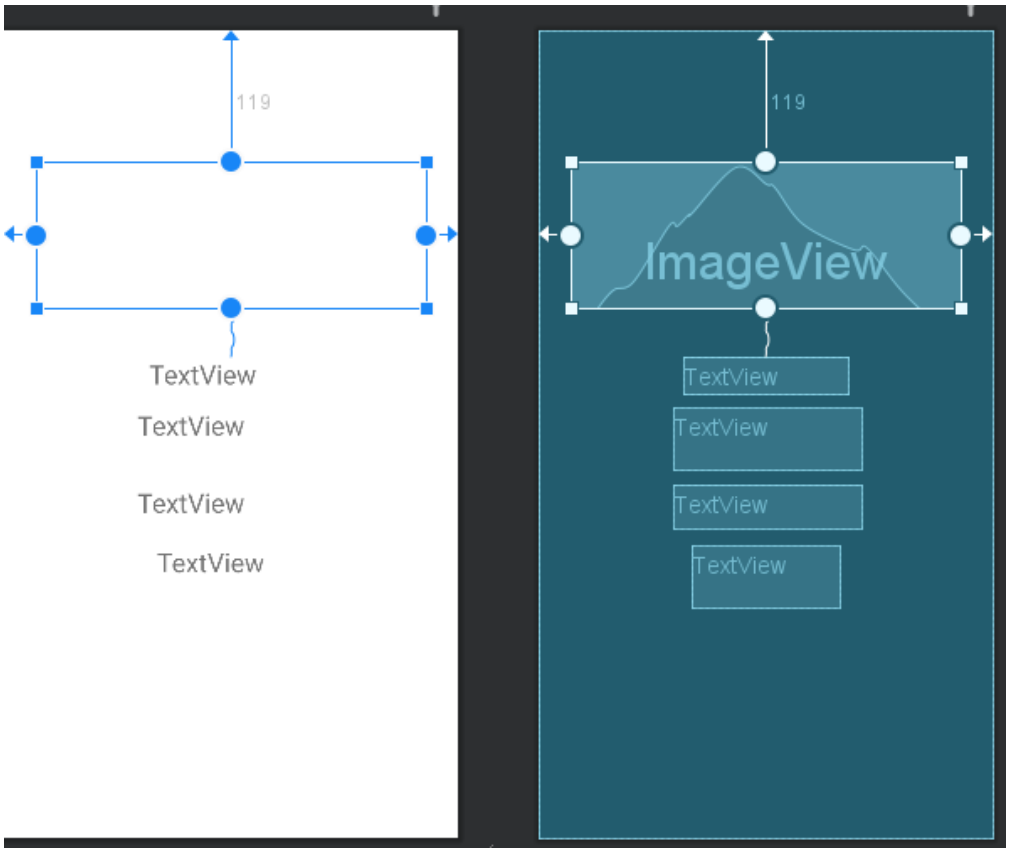




UI Design (Android Studio) – Full implementation

This app uses constraint layout as the main layout for the app.





Gap 2: Multi-Activity

Activities are an important component of any application. Without activities, the application will not be able to function properly or even be interactive for any user. According to Android Developers, the way activities are built and launched are key parts of android's application model. The android system initiates an instance activity in the main method on launch. It then invokes specific methods that corresponds with other stages of the lifecycle of the activity.

One of the key concepts for activities, in specific, mobile application, is that users do not always begin from the same place in the app. They may begin their interaction from other parts of the application which the activity class is designed to handle. This way, the activity serves as an entry point for the application's interaction with the user.

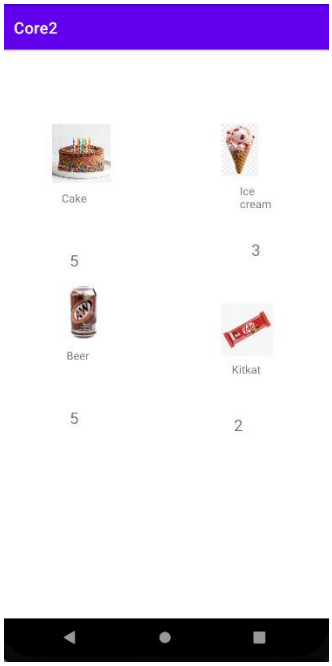
To put the concept of activities into context, this core task uses two different activities when the user interacts with the app.

<u>Activities</u>

- MainActivity – This is the default driver that is invoked when the app is launched, all key functions fall under this activity

- DetailsActivity – This is the second activity to be invoked when the user clicks on the image in the MainActivity. It reveals more details about the food.

MainActivity



DetailsActivity

Gap 3: Intents and Parcelable Protocol

Intents form another crucial component of Android's application model. The main function of intents is to act as a messaging object to request action with or from another application component. Intents have three fundamental uses in the Android system which is stated as below

Start Activity
In starting activities, intents can be used to invoke the StartActivity() function . The intent represents the activity to start and carries crucial data.

Start Service
Intents can be used to start a service, which is a component which runs operations in the background without a user interface, An example use case, is developing a client server interface which binds the service of another component by passing an intent.

Deliver broadcast
A broadcast is a message that can be received from any application. Use cases of broadcast are the Android system delivering broadcast for system events such as booting, charging the device, or playing a notification sound. The broadcast is passed using an intent.

Intent contains two types, **implicit** and **explicit.**

Implicit
Implicit intents do not specify a component, it declares a general action to carry out and allows the component of another application to handle the intent.

Explicit
Explicit intents specify which application will satisfy the intent by furnishing it with the target application's package name or a component class name. Explicit intents are normally used to start a component in the application because the programmer knows the name of the class, activity, or service to start.

In this project, explicit intents are used to supply data from one activity to another. Intents are used to pass the name of the food, date taken, cuisine, and rating from the main activity to the Details Activity when the user clicks on the image.

```
val intent = Intent( packageContext: this, DetailActivity::class.java )
intent.putExtra( name: "resId", R.drawable.beer)
intent.putExtra( name: "name", beer.name)
intent.putExtra( name: "date", beer.date)
intent.putExtra( name: "cuisine", beer.cuisine)
intent.putExtra( name: "rating", beer.rating)
startActivity(intent)
```

It can be observed that when the intent is passed, it is passed to the DetailActivity and additionally supplies the id of the image, name, data, cuisine, and rating of the food together with it. The data gets passes first before the activity launches. This code snippet is from the MainActivity.

```
val beer = FoodData(name = "Beer", date = "2021-10-22",
    cuisine ="Beverage", rating = "5" )
```

From the code snippet, it can be observed that this is much more succinct way to pass data to the intent rather than creating separate variables for each code which will lead to redundant code and may slow down performance. The FoodData constructor takes the data like beer, beverage, 5, and 2021-10-22 and passes it to the corresponding tags which is then sent to the intent as explained above. The constructor is under the Parcelable protocol which will be explained in later sections.

```
val FoodName = findViewById<TextView>(R.id.Food_name)
val FoodDate = findViewById<TextView>(R.id.Food_date)
val FoodCuisine = findViewById<TextView>(R.id.Food_cuisine)
val FoodRating = findViewById<TextView>(R.id.Food_rating)

val name = intent.getStringExtra( name: "name")
val date = intent.getStringExtra( name: "date")
val cuisine = intent.getStringExtra( name: "cuisine")
val rating = intent.getStringExtra( name: "rating")
FoodName.text = name
FoodDate.text = date
FoodCuisine.text = cuisine
FoodRating.text = rating
```

This code snippet is from the DetailActivity. It can be observed that the name, date, cuisine, rating takes the data from the intent tags. After that the data is assigned to the various textview for the user to see the data. This implementation works for text only, for the image, it requires a slightly different implementation which will be explained in the Bundles section.

Parcelable Protocol

The parcelable protocol acts as the interface which instances can be written to and restored from a Parcel. In addition, the parcel protocol reduces redundant code by allowing the object to act as a container to sent data via the IBinder. IBinder is a lightweight call protocol for performance for in-process and cross process calls.

```
@Parcelize
data class FoodData (val name:String,val date: String,
val cuisine: String, val rating:String):Parcelable
{

}
```

It can be observed that this class is specifically for data to be passed. The Parcelize above enables the class constructor to acts as the container moving data across activities. The Parcelable keyword makes the constructor to be able to receive written data and restore it in another activity via the Parcelize function.

Gap 4: Bundles

Bundles are similar to intents in which they are used to pass data from one activity. However, intents are much more specific and works better with plain text data. To represent an image because it much more complex in nature, bundles are a better option.

```
intent.putExtra( name: "resId", R.drawable.beer)
```

It can be observed that an image has to be drawn to the screen resulting in calling the drawable function. To pass the image, the id of the image is sent to the intent.

```
//Image
val imageView: ImageView = findViewById(R.id.imageView)
val bundle: Bundle? = intent.extras
if(bundle != null)
{
    val resId: Int = bundle.getInt( key: "resId")
    imageView.setImageResource(resId)
}
```

From this code snippet, it then can be observed that bundle receives the image id through the intent. The bundle then checks if it is null, if it is not, it gets the image id and then draws the image to screen.

Gap 5: Resources and images

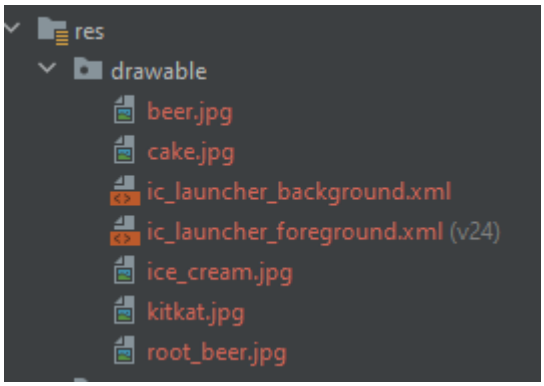
Resources

Resources are additional files and external content that the application uses. It can range from layouts, images, class definitions, user interface strings, and etc. Externalizing application resources help with efficiency and maintenance. It also helps ensure that reusing resources in other applications is made easier.

In this project, string and images resources are used. Attached below are the code snippets demonstrating their usages

```
<resources>
    <string name="app_name">Core2</string>
    <string name="cake">Cake</string>
    <string name="ice_cream">Ice cream</string>
    <string name="beer">Beer</string>
    <string name="kitkat">Kitkat</string>
</resources>
```

It can be observed that string resources used in this project are for the food names. Not hardcoding the strings into their attributes are good Kotlin practices also



Images are stored within the res/drawable as recommended by Android Developers.

Open Issues and Recommendations

Currently, the app is running fine however, in terms, of UI elements, especially after inferring constraint layouts, it downsizes all the elements to smaller sizes. This could be due to the layout type is trying to ensure it fits well on all screen sizes. Other than that, all the other functions of the application requirements are working correctly as expected. I recommend that Android come up with a new layout structure that is similar to constraint layout that allows drag and drop while maintaining sizes of UI elements.