Digital
Turbine Technical Test

1. Create an Android app that consists of two screens. The first screen contains a scrolling list of products, which uses the following URL as the datasource. In this screen, each item should display at least name, thumbnail, rating of the product. When the user clicks on one of the items, it brings the user to the second screen. The second screen should contain all information provided by the aforementioned datasource. Clicking the back button should bring the user back to the first screen.

http://ads.appia.com/getAds?id=236&password=OVUJ1DJN&siteId=10777&deviceId=4230&sessionId=techtestsession&totalCampaignsRequested=10

For every request to the URL above, add a parameter called "lname" and insert your last name as the value.
Document the design (class diagram and sequence diagram)
Bonus: suggest and describe UI responsiveness improvements

See DTClassAndSequence.pdf
As for UI responsiveness improvement, the only issue with this app was loading the thumbnails. One approach might be to preload and cache the thumbnails and access them locally when you populate the adapter or load them from the network the first time time you populate the adapter and cache them as you load them. Then anytime you populate an element you could check to see if the thumbnail was cached and if so grab it locally. I used an image loading library called Glide which handles threading of the network transactions and if I am not mistaken also automatically caches at least some of the images.

2. Explain how you handle Android segmentations in the past (e.g., different OS version, different screen-size, different behaviors on different devices). Please give actual examples.

OS segmentation: In my experience development is usually limited to no more than n-3 or 4 (where n is the most recent android release). If you are supporting pre-marshmallow releases, you have to take care to check the OS version when handling permissions. It's also very important to test on all the OS versions you intend to support. Even with the support libraries you can come across methods or classes that are not supported on some earlier OSs so you have to find a workaround. I've done a number of camera apps and I prefer to use the camera2 api so for those apps I limited the OS to Lollipop or newer.

Different behaviors on different devices usually comes in to play when you are dealing with device hardware – particularly cameras. You can't assume that all cameras support the same features

(such as focus mode) so you have to check to make sure the modes you want to use are present. Vendor specific implementations such as softkeyboard can also cause issues – for instance next or done button implementations or the keyboard layouts in landscape mode. Again, it's important to test on as many different devices as is reasonable. Hard navigation keys (Samsung) and soft navigation keys are another issue. Your app won't know if there are soft navigation keys present or not. One solution is to hide
the soft nav keys (they appear when swiping up) programmatically or you can just assume they are always there and design appropriately. More recently some screens are coming with a longer height (by height I mean height in portrait) dimension. You can deal with those by checking the screen dimensions and providing layouts for the different options, but I prefer to try to do something that is fairly aesthetically pleasing that just ignores the extra length. I did an app for a client that streamed video from an ip camera, but it was important to minimize latency. Using the native mediacodec on phones, I generally got a latency of about 2 s. But on tablets, it was consistently on the order of 10 to 20 s. We had to integrate a third party codec to solve the issue.

Different screen sizes: I start every layout as a constraintlayout. Avoid defining view dimensions in terms of hard coded sizes if possible and never use absolute pixel dimension. If you have to (and you very frequently will) use dp and always sp for text. Text usually brings the most problems – it needs to be large enough to read but small enough to fit. I frequently rely on ellipsis when I am dealing with unknown text lengths. I also rely on weighting linearlayouts. For instance in one activity I did for a client, I needed a map fragment and a recyclerview to share the screen. You can hard code the heights in dp, but you are not guaranteed a good layout. By setting height to zero and using relative weights you get a consistent result across screen sizes. One client I developed for had a customer base that was much if not most tablets. While my layouts worked well across phones, there were issues on the tablets – some of the views were too small and textsize was an issue. Android lets you define layouts for different screen sizes (regular, large, xlarge) so I took advantage of that to resize some views and the text as well. I also defined different layouts for portrait and landscape.

3. Explain 2 instances where you have learned something only through experience (Ex: Solving a production issue) and could not have learned about that problem through theoretical knowledge or reading through technical documentation.

   It's really hard to say that something is not documented somewhere in all the thousands of pages of java/kotlin/android documentation. I can point to the latency issues for tablets I discussed previously. I'd also go a little out of bounds here and talk about build/gradle issues in android studio. One recent example is when using android jetpack, android studio kept trying to import a support library instead of an androidx library which caused a conflict that wouldn't allow android studio to find the AppCompatActivity class. I finally figured out that I needed to remove the support library import – there are lots of issues like that I come across. Another recent example has to do with Kotlin data classes in the demo app. I defined a class:

```
@Parcelize
data class Ad( private var a, private var b, …): Parcelable{
      var count = 0
}
```

The count variable was set in the MainActivity and when I passed the instance in a bundle to the DetailActivity the count was always getting zeroed.
I found I had to add it to the constructor for it to maintain state when it was passed in the bundle:

```
@Parcelize
data class Ad(private val a, private val b, …, var count = 0): Parcelable
```

That may be documented somewhere, but I would have no idea where to find it.

4. Given the choice between writing ideal code and meeting a critical deadline, which would you choose and why?

   I'd choose meeting the critical deadline. You can only meet the deadline once, you can always come back and fix the code later. This is with the caveat that you are not releasing something that is likely to crash or present something that is irritating to the user - in those cases you might want to consider altering the deadline.