SEENG440 Assignment 1 Post Mortem

Sam Annand | sga111 | 48562140

24 / 04 / 20

Application Purpose:

I have created an android application – *CarSpotter* – aimed at users who like cars, and enjoy going to car meets, local racing events, etc. The purpose of the application is to allow users to collect real life cars in the app, building up a collection of the special cars they have seen in real life, so that they can browse through this collection, and remember the moment when they saw an awesome car.

Development Process:

The development process for this applications started with me often asking my Dad, "Have we seen one of those in the flesh, I think I have?", referring to whether or not we had seen a car in real life before. I then came up with the idea for an app that would allow someone to keep track of this, and started with some basic wireframes. I looked to apps such as *Pokémon Go* for inspiration, as I was interested in gamifying my app with achievements and other features.

In terms of management, I created a Trello board to use as a Kanban board, and put all mandatory and grade bearing requirements on the board as backlog items. Progress was slow initially, with most of my app's development being completed over the term 1 university break.

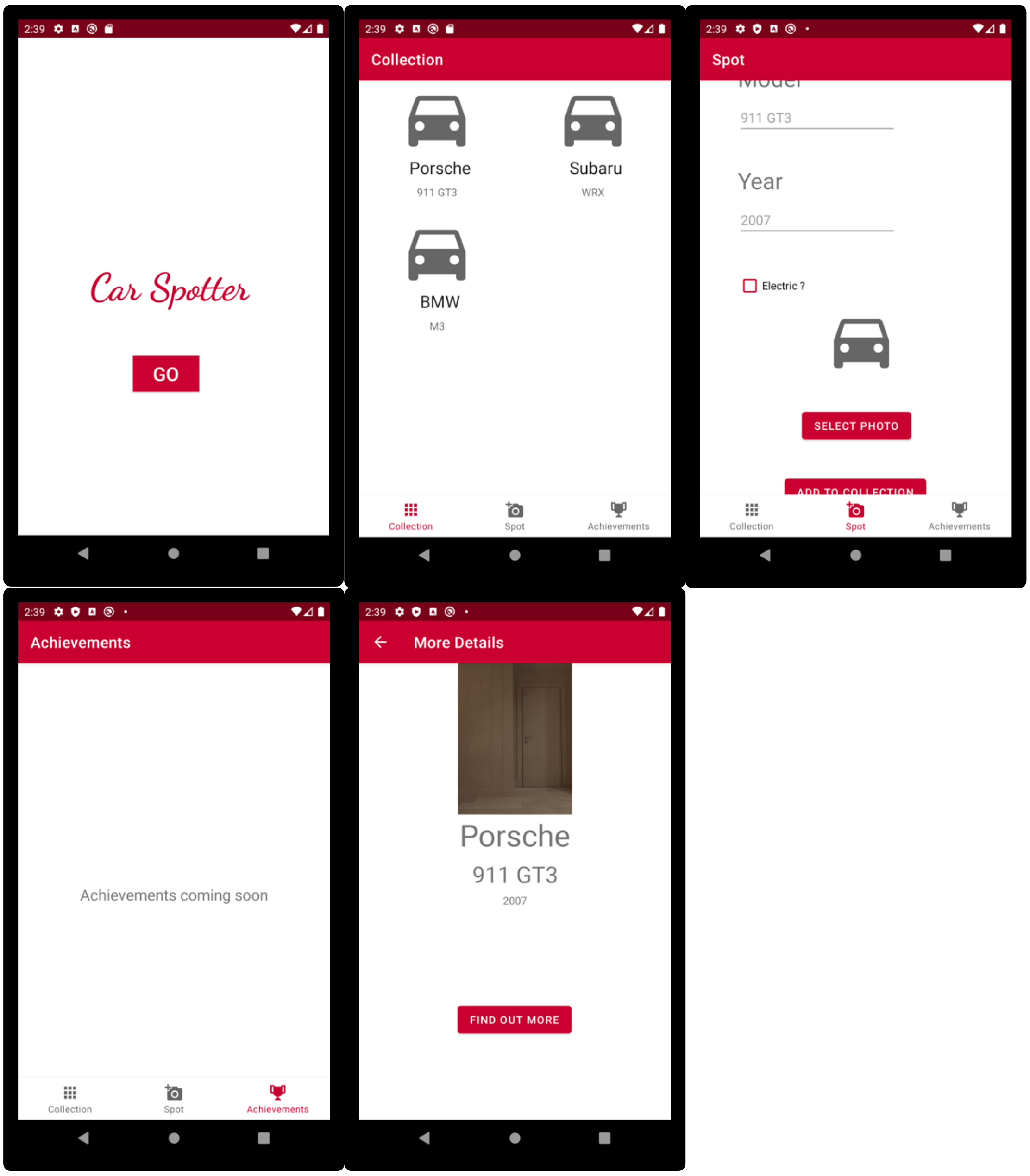
Having completed the app, I am happy with the end result. Unfortunately I was unable to complete the achievements functionality, and was unable to style the app to a level that I would be perfectly happy with. This leaves plenty of room for future development and polishing.

Requirement enumeration:

1. **Compose your app out of at least three screens, where a screen is either an Activity spawned via an explicit Intent, or a Fragment with a full screen layout.**

   CarSpotter is composed of five screens in total:
   - A splash screen as an entry point for the app
   - A Collection screen that display's cars a user has 'spotted'
   - A Spotting screen that allows a user to add a new car to their collection
   - An achievements screen that was unable to be implemented, and has no current functionality
   - A more details screen, that allows a user to view the photo attached to a car, and find out more details about the car in their collection

These screens can all be seen in the screenshots below:

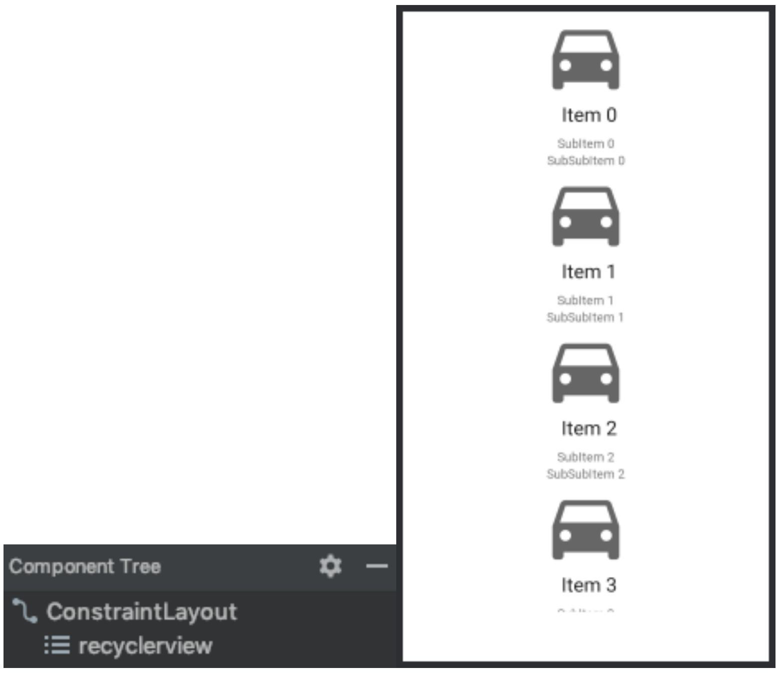


2. **Invoke at least one other app on the system via an implicit Intent.**

   A web browser is invoked when clicking the ‘FIND OUT MORE’ button on the more details view, as seen above, to find out more details about a car using Wikipedia.

3. **Include a list view, preferably using RecyclerView.**

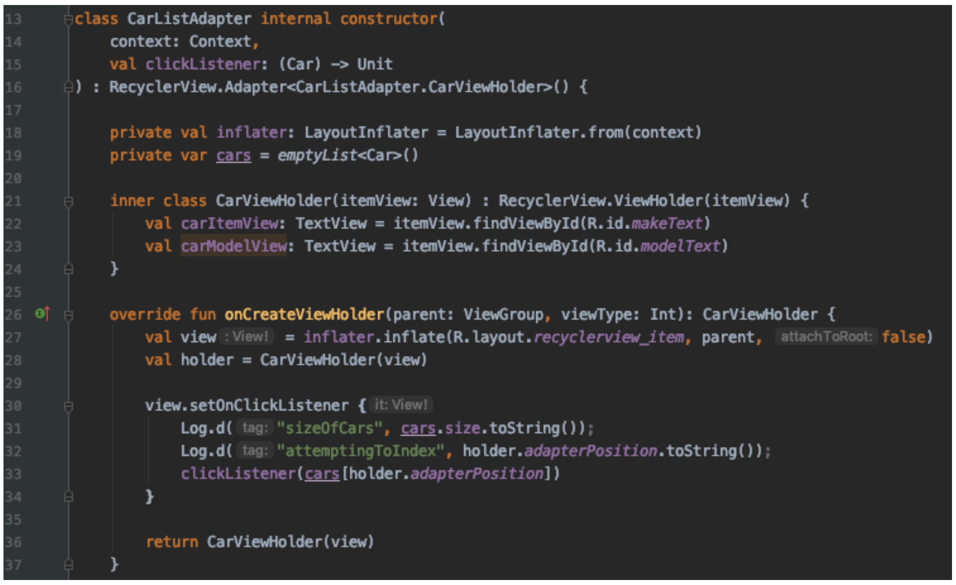
   I used a RecyclerView to display the cars in a user’s collection on the Collection screen.



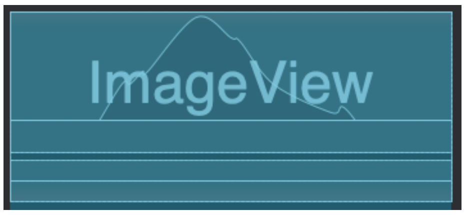
I then set the layout manager to a grid layout manager programmatically.

4. **Compose your list view using a custom adapter whose view creation method uses a custom layout**

The custom adapter (CarListAdapter) and custom layout (recyclerview_item.xml) for the recycler view can be seen in the screenshots below.

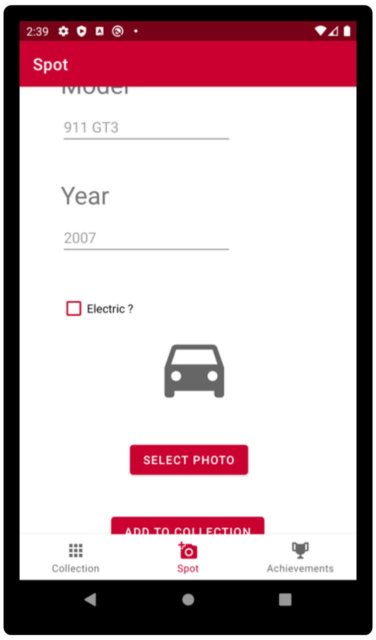


```
class CarListAdapter internal constructor(
    context: Context,
    val clickListener: (Car) -> Unit
) : RecyclerView.Adapter<CarListAdapter.CarViewHolder>() {

    private val inflater: LayoutInflater = LayoutInflater.from(context)
    private var cars = emptyList<Car>()

    inner class CarViewHolder(itemView: View) : RecyclerView.ViewHolder(itemView) {
        val carItemView: TextView = itemView.findViewById(R.id.makeText)
        val carModelView: TextView = itemView.findViewById(R.id.modelText)
    }

    override fun onCreateViewHolder(parent: ViewGroup, viewType: Int): CarViewHolder {
        val view :View! = inflater.inflate(R.layout.recyclerview_item, parent, attachToRoot: false)
        val holder = CarViewHolder(view)

        view.setOnClickListener { it: View!
            Log.d( tag: "sizeOfCars", cars.size.toString());
            Log.d( tag: "attemptingToIndex", holder.adapterPosition.toString());
            clickListener(cars[holder.adapterPosition])
        }

        return CarViewHolder(view)
    }
```



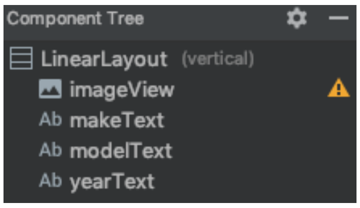
5. **Include at least five different kinds of widgets besides a list view (buttons, textboxes, checkboxes, labels, and so on) in the user interface, and handle their interactions with event listeners.**

   I have used various widgets throughout the application. Below is a screenshot of the Spotting screen, which utilises labels, textboxes, a checkbox, an image view, and two buttons.



6. **Use at least two different layout groups (e.g. Constraint Layout and LinearLayout) to organise your widgets.**

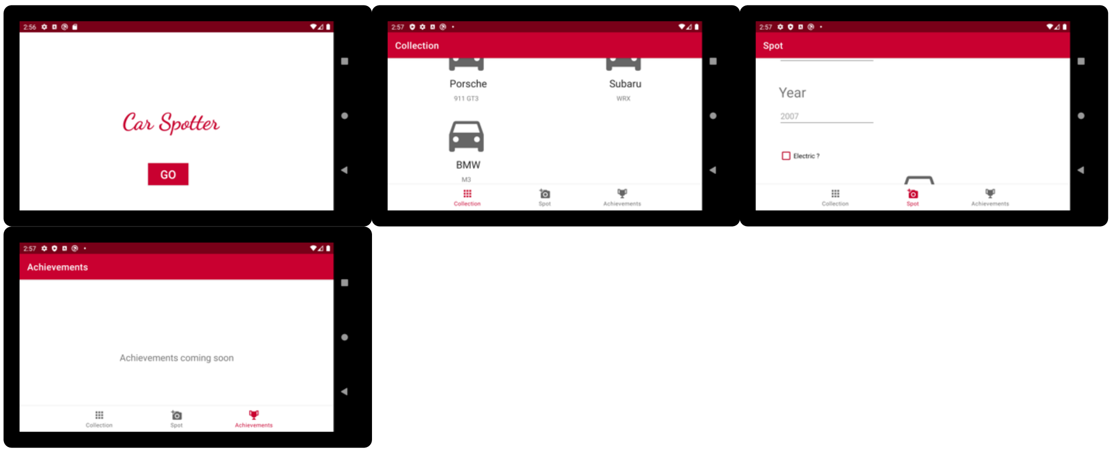
   Most of the layout groups in my application are ConstraintLayout's, however there is also a linear layout used in the recycler view item.



```
Component Tree
LinearLayout (vertical)
  imageView
  Ab makeText
  Ab modelText
  Ab yearText
```

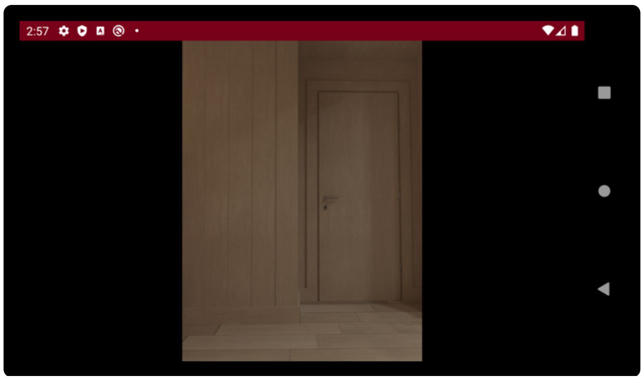
7. **Support both landscape and portrait orientations in all views. In other words, all widgets should be able to be made fully visible in either orientation. This may happen automatically given your layout manager, or you may use a ScrollView, or you may specify two separate layouts.**

   Screenshots of all my main views in landscape mode can be seen below. For the most part this was automatically achieved due to using constraint layouts, however I did have to use a ScrollView for the more details screen. Please note that although things are cut off in the screenshots, they are able to be scrolled into view in all cases.



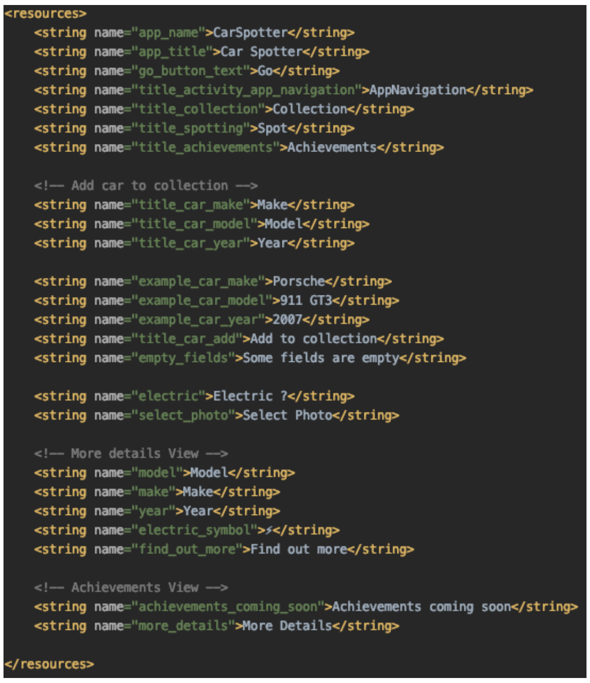
8. **Provide separate landscape and portrait layout resources for at least one of the views.**

   I have created a separate landscape layout for the more details screen, that is a gallery view for just displaying the photo attached to a car.



9. **Use string resources for all static text on the user interface.**

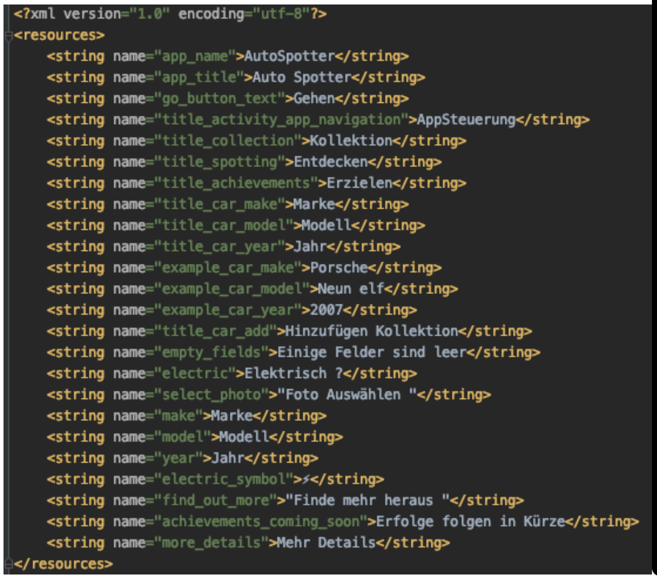
   All static text on the user interface uses string resources specified in the below file.



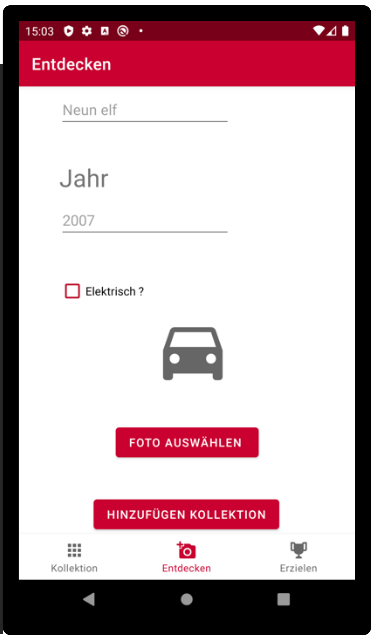
```xml
<resources>
    <string name="app_name">CarSpotter</string>
    <string name="app_title">Car Spotter</string>
    <string name="go_button_text">Go</string>
    <string name="title_activity_app_navigation">AppNavigation</string>
    <string name="title_collection">Collection</string>
    <string name="title_spotting">Spot</string>
    <string name="title_achievements">Achievements</string>

    <!-- Add car to collection -->
    <string name="title_car_make">Make</string>
    <string name="title_car_model">Model</string>
    <string name="title_car_year">Year</string>

    <string name="example_car_make">Porsche</string>
    <string name="example_car_model">911 GT3</string>
    <string name="example_car_year">2007</string>
    <string name="title_car_add">Add to collection</string>
    <string name="empty_fields">Some fields are empty</string>

    <string name="electric">Electric ?</string>
    <string name="select_photo">Select Photo</string>

    <!-- More details View -->
    <string name="model">Model</string>
    <string name="make">Make</string>
    <string name="year">Year</string>
    <string name="electric_symbol">⚡</string>
    <string name="find_out_more">Find out more</string>

    <!-- Achievements View -->
    <string name="achievements_coming_soon">Achievements coming soon</string>
    <string name="more_details">More Details</string>

</resources>
```

10. Provide default definitions for your string resources in English. Provide definitions for one other language. (Use your favourite online translator if necessary.)

All German translations provided.

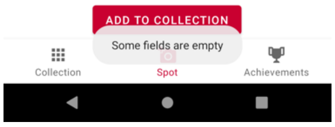


```xml
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string name="app_name">AutoSpotter</string>
    <string name="app_title">Auto Spotter</string>
    <string name="go_button_text">Gehen</string>
    <string name="title_activity_app_navigation">AppSteuerung</string>
    <string name="title_collection">Kollektion</string>
    <string name="title_spotting">Entdecken</string>
    <string name="title_achievements">Erzielen</string>
    <string name="title_car_make">Marke</string>
    <string name="title_car_model">Modell</string>
    <string name="title_car_year">Jahr</string>
    <string name="example_car_make">Porsche</string>
    <string name="example_car_model">Neun elf</string>
    <string name="example_car_year">2007</string>
    <string name="title_car_add">Hinzufügen Kollektion</string>
    <string name="empty_fields">Einige Felder sind leer</string>
    <string name="electric">Elektrisch ?</string>
    <string name="select_photo">"Foto Auswählen "</string>
    <string name="make">Marke</string>
    <string name="model">Modell</string>
    <string name="year">Jahr</string>
    <string name="electric_symbol">⚡</string>
    <string name="find_out_more">"Finde mehr heraus "</string>
    <string name="achievements_coming_soon">Erfolge folgen in Kürze</string>
    <string name="more_details">Mehr Details</string>
</resources>
```



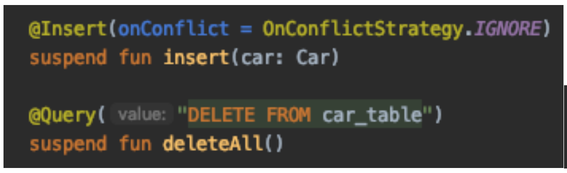
11. Use a Toast message or dialog to alert or interact with the user.

If a user attempts to add a car leaving either the make, model, or year field blank, a Toast message appears.

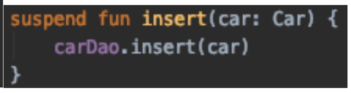


12. Use an AsyncTask to trigger some computation without blocking the user interface.

I instead used the more modern coroutines to trigger Room database transactions without blocking the UI. Some examples can be seen below, the ‘suspend’ keyword is the giveaway.



```
@Insert(onConflict = OnConflictStrategy.IGNORE)
suspend fun insert(car: Car)

@Query( value: "DELETE FROM car_table")
suspend fun deleteAll()
```



```
suspend fun insert(car: Car) {
    carDao.insert(car)
}
```

13. Share a plan for your app before Saturday of week 2 in a post on #project1 in Slack – before you’ve written any code or created any layouts. Include hand-drawn sketches or wireframes.

https://seng440-2020.slack.com/archives/CTZUZ7BU0/p1582866373030500

14. Share an update of your work before Saturday of week 3 in a post on #project1 in Slack. Include screenshots.

https://seng440-2020.slack.com/archives/CTZUZ7BU0/p1583484081114000?thread_ts=1583484081.114000

15. Share an update of your work before Saturday of week 4 in a post on #project1 in Slack. Include screenshots.

https://seng440-2020.slack.com/archives/CTZUZ7BU0/p1584087738048700

16. Share an update of your work before Saturday of week 5 in a post on #project1 in Slack. Include screenshots.

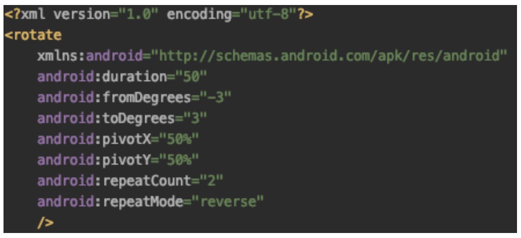
https://seng440-2020.slack.com/archives/CTZUZ7BU0/p1584693844122500?thread_ts=1584693844.122500

17. Share an update of your work before Saturday of week 6 in a post on #project1 in Slack. Include screenshots.

https://seng440-2020.slack.com/archives/CTZUZ7BU0/p1587092267049600

18. **Incorporate an animation into your UI, preferably one specified in XML. We will not discuss these in lecture. You should be able to find out more information on the Android developer website.**

    When the car spotting form is unable to be submitted due to some fields being left blank, the ‘Add to collection’ button shakes according to the following xml animation.

```xml
<?xml version="1.0" encoding="utf-8"?>
<rotate
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:duration="50"
    android:fromDegrees="-3"
    android:toDegrees="3"
    android:pivotX="50%"
    android:pivotY="50%"
    android:repeatCount="2"
    android:repeatMode="reverse"
    />
```

19. **Incorporate some other Android feature not mentioned above into your app.**

    I have implemented a Room database to persist a user’s car collection.

20. **Incorporate some other Android feature not mentioned above into your app.**

    I have implemented photo support into my application. A user is able to attach a photo to a car they are adding using a file picker, and that photo is then displayed when clicking on the more details view.

    Another feature that could be considered an extra feature for requirements 19 or 20 is the bottom navigation menu.