Quality Report

Team 23: Jason Wang, Samuel Chen, Kayla Kautai

Introduction

Our final Android application, **Motukā**, helps users browse information about different vehicles. The app was developed to meet the specifications given. More importantly, the team properly applied and realised SOLID principles during the period of this project. This report will demonstrate how our application satisfies the requirements and how SOLID principles are applied both structurally and within the classes as proposed in the design doc. It will also detail the quality attributes our design achieves, the inconsistencies with the design doc and the coding practices we employed. Lastly, we conclude by reflecting on the lessons we learned during the project and list future work areas.

Application Overview

Our team was asked by *YouSee Soft* to participate in a competition. This involved developing a native android app that can be used to showcase or sell items. Hence, a brief was provided to ensure we develop software that meet the requirements for the desired application. The following will discuss the app specifications and how our application Motukā satisfy them.

Overall User Experience

Firstly, we needed to ensure that the app is responsive. So we implemented each activity to be resizable across different screen size and to correctly adjust to the orientation of the device. We were also instructed to use animations and transitions. Therefore, to improve the UX of our app, we added smooth transitions between pages, progress bars when loading content and animations when displaying messages.

MainActivity

We satisfied the requirements by presenting a "Top Picks" panel (seen in Figure 1) that randomly generates vehicles from the database to update the RecyclerView as the user interacts with the app. As for the categories, we have provided three forms of vehicles: Electric, Hybrid and Petrol (seen in Figure 2). Most importantly, we ensured to follow a proper inheritance structure which we talk in depth about in the SOLID principles section of this report.

Finally, we were asked to include searching functionalities that invoke a search request from MainActivity and shows the results of the search term. Hence, we implemented a search bar, seen in Figure 1, which a user can search for any vehicle by name and leads to the ResultsActivity.

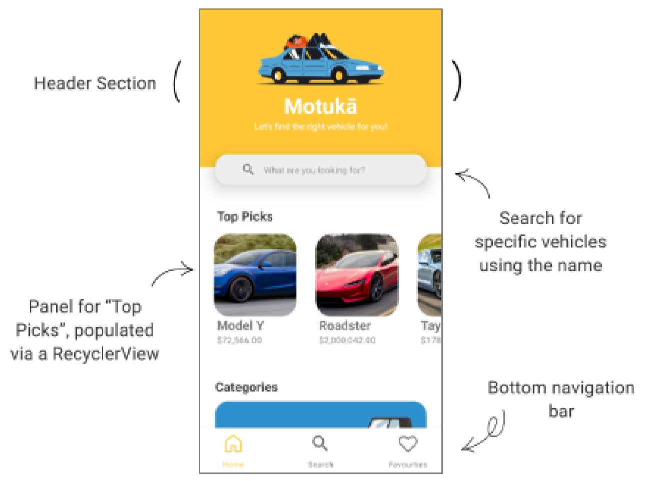

Header Section
Motukā
Let's find the right vehicle for you!
What are you looking for?
Top Picks
Model Y
$72,566.00
Roadster
$2,000,042.00
Categories
Home
Search
Favourites
Search for specific vehicles using the name
Panel for "Top Picks", populated via a RecyclerView
Bottom navigation bar

Fig 1. Screenshot of MainActivity

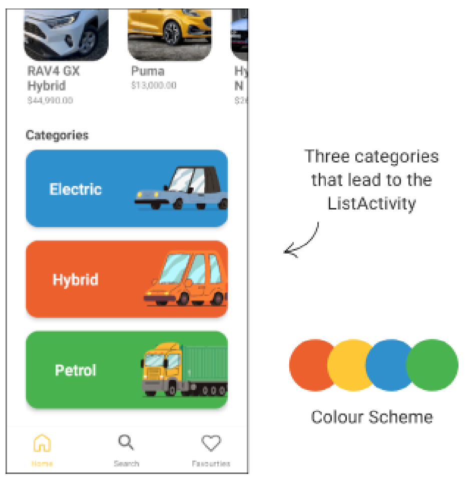


Fig 2. Screenshot of MainActivity (cont.)

ListActivity

As for ListActivity, we chose a RecyclerView to dynamically present the vehicles from the database associated with the selected category. As per the brief, we provided precisely ten vehicles for each category for the user to scroll through. The list presents each vehicle item as an object of their respective categories class that inherits the Vehicle class.

Another requirement listed was that the GUI design differs for each category. In our case, we used a single adapter that alters the header's styling, car illustration and overall category colour. For example, blue portrays the category Electric; thus, the header background and all buttons are blue, as seen in Figures 3 and 4. We further discuss the justification as to why we used one custom adapter in the maintainability section.

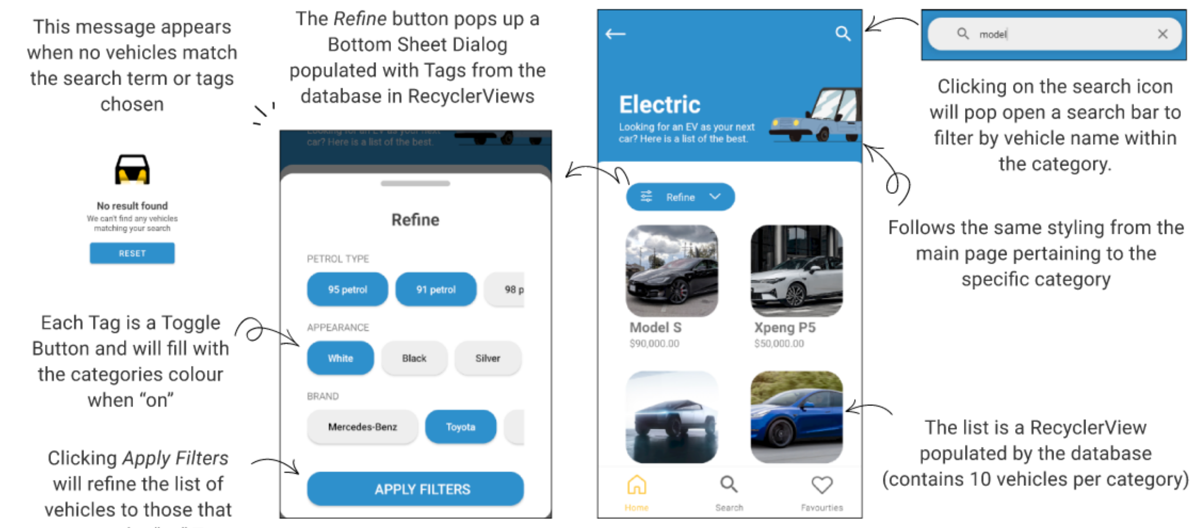

This message appears when no vehicles match the search term or tags chosen
No result found
We can't find any vehicles matching your search
RESET
Each Tag is a Toggle Button and will fill with the categories colour when "on"
Clicking *Apply Filters* will refine the list of vehicles to those that possess the "on" Tags
The *Refine* button pops up a Bottom Sheet Dialog populated with Tags from the database in RecyclerViews
Refine
PETROL TYPE
95 petrol
91 petrol
98 p
APPEARANCE
White
Black
Silver
BRAND
Mercedes-Benz
Toyota
APPLY FILTERS
Electric
Looking for an EV as your next car? Here is a list of the best.
Refine
Model S
$90,000.00
Xpeng P5
$50,000.00
Home
Search
Favourites
model
Clicking on the search icon will pop open a search bar to filter by vehicle name within the category.
Follows the same styling from the main page pertaining to the specific category
The list is a RecyclerView populated by the database (contains 10 vehicles per category)

Fig 3. The 'Refine' dialog on the ListActivity

Fig 4. Screenshot of ListActivity

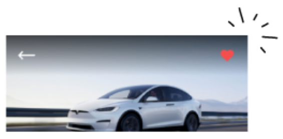
DetailsActivity

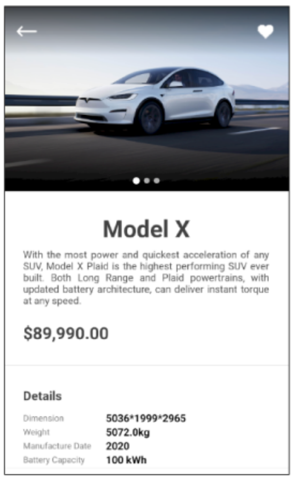
A vehicle item provides access to the DetailsActivity. As specified in the requirements, we provided the user with three vehicle images in an image slider sourced from a GitHub library [1]. Furthermore, this activity displays information stored in the database, such as price and dimensions, which all items share in the Vehicle model. However, vehicles from different categories will also display key properties that are unique to the model type. For example, an electric car displays battery capacity, and a petrol car will show tank capacity, etc.

An additional feature we added was a button to "favourite" a vehicle. The animation of this button was also implemented from a GitHub library [2].

Back icon is a button that finshes the activity and returns to the previous one.

The heart icon can be clicked to “favourite” a vehicle. A user will then be able to access the specific vehicle in the Favourites tab.





Model X

With the most power and quickest acceleration of any SUV, Model X Plaid is the highest performing SUV ever built. Both Long Range and Plaid powertrains, with updated battery architecture, can deliver instant torque at any speed.

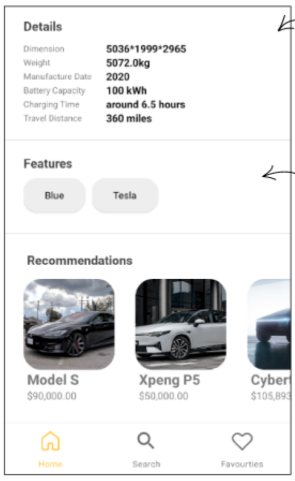
$89,990.00

Details

Dimension	5036*1999*2965
Weight	5072.0kg
Manufacture Date	2020
Battery Capacity	100 kWh

Fig 5. Screenshot of DetailsActivity

An image slider, with an auto scoll features, shows 3 unique images for each vehicle selected.



Details

Dimension	5036*1999*2965
Weight	5072.0kg
Manufacture Date	2020
Battery Capacity	100 kWh
Charging Time	around 6.5 hours
Travel Distance	360 miles

Features

Blue Tesla

Recommendations

Model S $90,000.00

Xpeng P5 $50,000.00

Cyber $105,893

Fig 6. Screenshot of DetailsActivity (cont.)

Vehicle information is displayed in sections such as the vehicle’s description, price, weight etc.

Features are all the tags a vehicle holds in the database and can be filtered by in other activities.

A recommendation section proposes vehicles of the same type as the selected vehicle. This is displayed in a horizontal recycler view.

SearchActivity and ResultsActivity

The SearchActivity was not one of the three compulsory activities, however, we implemented one so that a user can easily access the functionality at their convenience from the bottom navigation bar. Very similar to the search bar on the main activity, a user can search for a vehicle by name but may also combine with various tags a vehicle may possess to refine the search.

A search request leads to the ResultsActivity (Figure 8) which was also an additional, separate activity. We chose to structure the app in this manner to cater to the user experience, and due to the differences in the GUI design we could not reuse the ListActivity.

These activities follow similar approach to other well-known Android apps to provide familiarity to users. It also handles all exceptions and provides a message to the user if no vehicle matched their search. The message displays a fun JSON animation sourced from Lottie Files [3] who supply open source animation files.

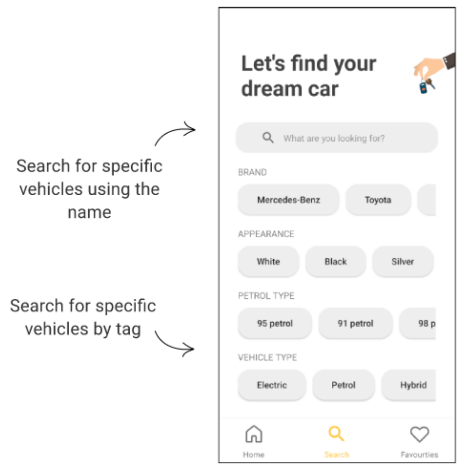


Fig 7. Screenshot of SearchActivity

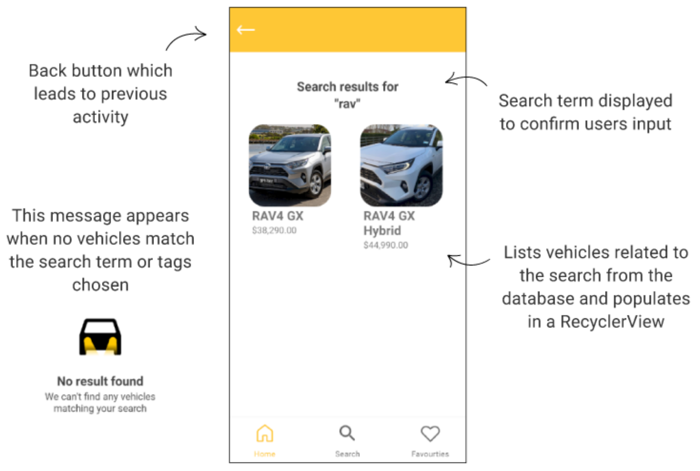

Back button which leads to previous activity
This message appears when no vehicles match the search term or tags chosen
No result found
We can't find any vehicles matching your search
Search results for "rav"
RAV4 GX
$38,290.00
RAV4 GX Hybrid
$44,990.00
Search term displayed to confirm users input
Lists vehicles related to the search from the database and populates in a RecyclerView
Home
Search
Favourites

Fig 8. Screenshot of ResultshActivity

FavouritesActivity

Finally, the FavouritesActivity was added to the app to compliment the "favourite" button on the DetailsActivity. The simple functionality of this page is to list all vehicles the user has added to their favourites list. Moreover, we wanted this activity to be easily located in the bottom navigation bar.

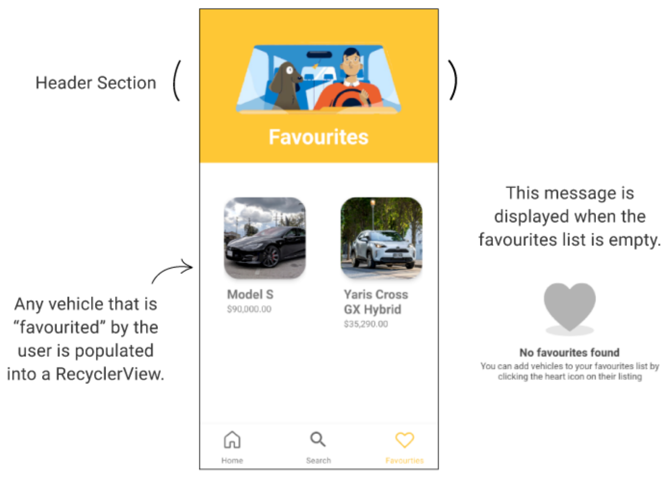

Header Section
Favourites
Model S
$90,000.00
Yaris Cross GX Hybrid
$35,290.00
Home
Search
Favourites
Any vehicle that is "favourited" by the user is populated into a RecyclerView.
This message is displayed when the favourites list is empty.
No favourites found
You can add vehicles to your favourites list by clicking the heart icon on their listing

Fig 9. Screenshot of FavouritesActivity

Database

A Firestore database was implemented for our app to store, organise and retrieve information about each vehicle and tags. It was sorted by five collections. The first three are *electric*, *hybrid* and *petrol* - used to store their associated vehicles properties. As for the *tags* collection, it stores all the tags that can be associated to a vehicle - providing the tag name and their tag type which tags can share. Lastly, the *user* collection stores a favourites array of vehicles pertaining to an specific user.

Solid principles

Single Responsibility Principle

SRP is applied in the development of the project. We used an abstract Vehicle class and split the responsibilities into three child class where each class is only responsible for containing attributes that are only unique to the type of the Vehicle.

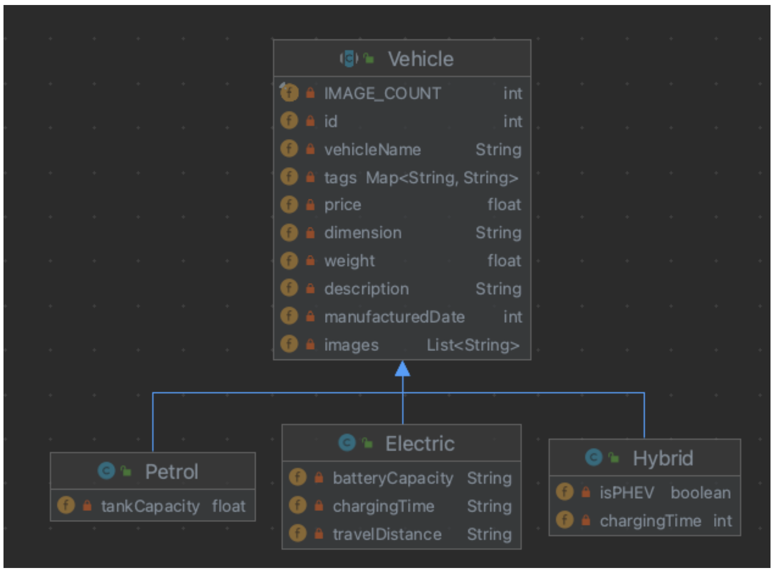


Figure 10. Abstract class Vehicle and its sub classes

The activity classes are implemented in a way that also followed SRP as each Activity class is only responsible for one scene for the GUI and each is independent of each other.

The adapter classes also are implemented in a way that follows SRP as each adapter is only responsible for one particular section of content. We have four adapters implemented, Top adapter is used for displaying the top picks section and Vehicle Adapter is used for displaying vehicle in list activity. We keep them separate from each other even though the logic is similar. The reason is to minimise coupling, for instance if we want to change the implementation and the look for the top pick section but not the list activity, it will be hard to do so if they share the same adapter.

The methods implemented inside each class also followed SRP as each method is responsible a specific part of the view in each activity. For example, in Main activity, the GUI construction is broken into five methods where each method is implemented for a particular part of the GUI like top picks, loading screen, navigation buttons, categories list, search bar. The benefit of following SRP for methods is that it makes the code more readable and easier for future modification without interfering other components within the same class.

Open Closed Principle

Our product applied OCP properly. This can be shown with the use of abstract classes of Vehicle(figure 6) and CoreActivity which open for more extension. For example, if a new category of vehicle comes out that is different from electric, hybrid and petrol which has its own unique attributes, the design allows an extension by creating a child class of Vehicle to contain its unique attributes. Hence, no modification needs to be made to existing classes and also open for extension of vehicle type in the future. Other parts including IVehicleDataAccess and CoreActivity also follow this principle, as they are both interfaces allowing for different implementations while having some basic requirements to implement the interfaces.

Liskov Substitution Principle

Our product applied LSP as the subclasses to Vehicle class do not violate any of the parent class policies and methods(shown in figure 6). The abstract class Vehicle contains the common attributes that all three subclasses would need, like weight, price and dimension. Hence, when an object is trying to get the weight, price or dimension from a subclass of Vehicle, the implementation of getting common attributes is defined in the superclass. Hence, it would not violate the superclass's policies and methods. Therefore, LSP is followed.

Interface Segregation Principle

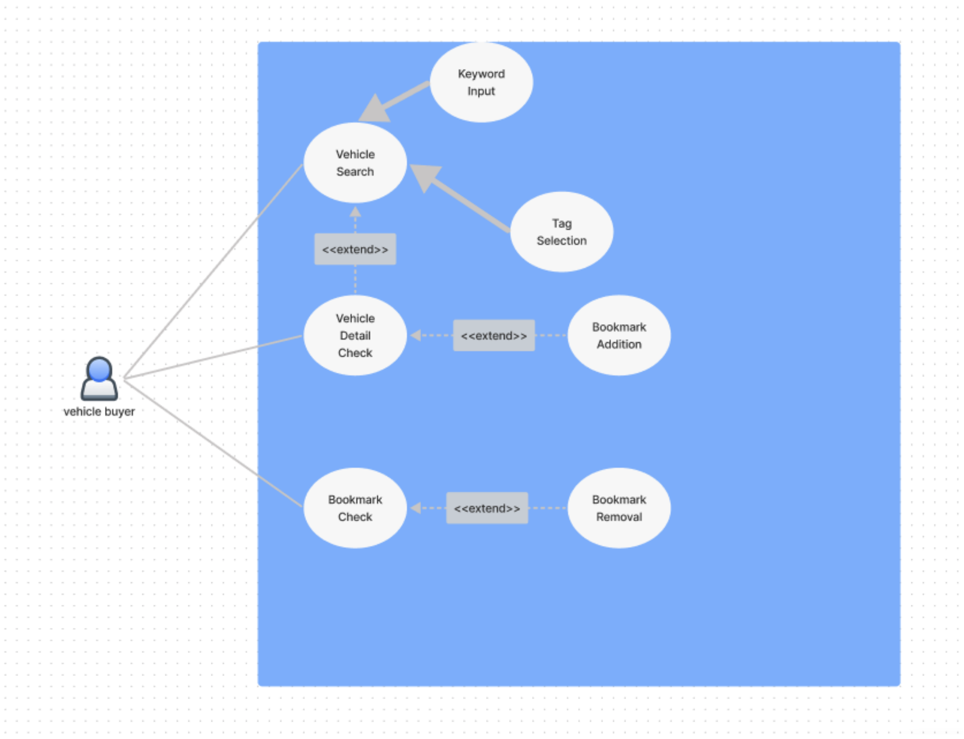


Figure 11. Use Case Diagram

Based on our use case diagram, the application is designed for only one type of actor which is the vehicle buyer. The application will not be use by a different actor. As a result, all the methods in the application are used by the actor. Therefore, structurally, ISP is not considered.

Dependency Inversion Principle

Dependency Inversion Principle if followed in the design of the program through the use of Dependency injection. This can be seen between the relationships between the Interface CoreActivity, IVehicleDataAccess, concrete class VehicleDataAccess and VehicleService, where CoreActivity is dependent on an implementation of IVehicleDataAccess, and its use can be initialized by the provided VehicleDataAccess class or any implementation developed by the client. This implementation creates loose coupling between the classes and allows changes to implementation to the code without affecting the depending classes. (This is evident in our UML diagram in figure 14)

Quality Attributes

Maintainability

In our final product, only one adapter is used for the list activity of different vehicle type. The adapter is designed in a way to accommodate different design in different list activity. As shown below, our list activity has different colour background and different vehicle graphics depends on the type of vehicle list. One advantage of using single adapter for list activity is that it promote maintainability. Using single adapter lowers the amount of coupling between objects which makes future modification to be easier. As a result, it will have a better maintainability.

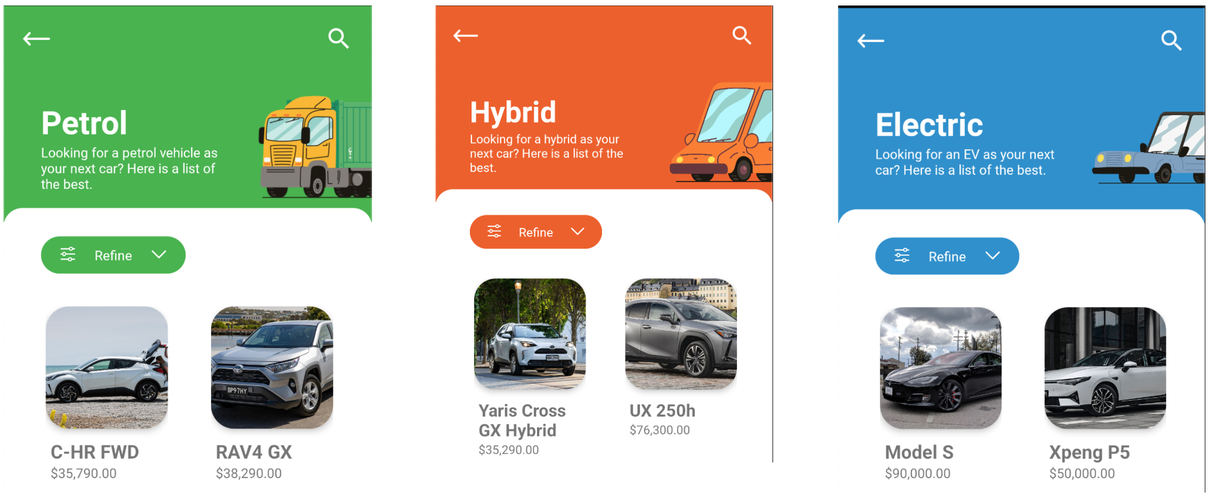

Petrol
Looking for a petrol vehicle as your next car? Here is a list of the best.
Refine
C-HR FWD
$35,790.00
RAV4 GX
$38,290.00
Hybrid
Looking for a hybrid as your next car? Here is a list of the best.
Refine
Yaris Cross GX Hybrid
$35,290.00
UX 250h
$76,300.00
Electric
Looking for an EV as your next car? Here is a list of the best.
Refine
Model S
$90,000.00
Xpeng P5
$50,000.00

Figure 12. Screenshots of each category in the ListActivity

Reusability

Our code promotes reusability mainly by inheritance, and this can be seen in the class diagram structure of the abstract class Vehicle and its children. The three main category used in our app is Petrol, Hybrid and Electric, and since all these are vehicles, they have some common qualities between them, hence, the abstract class Vehicle was developed, which is extended by the three categories based on their unique requirements. This design not only follows the Open Closed Principle mentioned above, it also allows code reuse as much of the functionalities of the three categories is provided by the Vehicle class, and no repeating code is needed for the vehicle categories to function, while maintaining their unique attribute and functionalities.

Another example of reusability of our app is in the design of our GUI. Despite our app having very diverse colour themes and designs in different activities, many styling of the GUI are actually reused. This not only allows the reuse part of the design, but also able to keep the feel of our app consistent throughout different activities.

A *themes* XML resource was used as a collection of attributes that could be applied to any section of the app. This included a type scale that follows a similar range of contrasting styles as the material design system. In our case we including a Heading 1, Heading 2, Heading 3, Subtitle 1, Subtitle 2 and Body to reuse the styling and create consistency within the app. Other styling was also reused such as a RoundedBackground which is referenced to from this resource in several activities and set in various view backgrounds.

Inconsistencies

Roles and responsibilities are slightly different from the design doc. Initially, we plan to have everyone work on both the frontend and backend so everyone was labeled as full-stack developer. But later we realised that having one person working on backend implementation is better because we are able to keep the backend implementation consistent through out the development.

One other aspect of the development also differed from the plan. Initially, we have all members working on the same activity, but then we changed it such that each member worked on single activity individually. We found out that it is hard to allocate the tasks for each member without affecting the tasks done by another member on the same activity class. The better approach we did is one member implements one activity and other members will refine it in the later stage.

GUI Changes

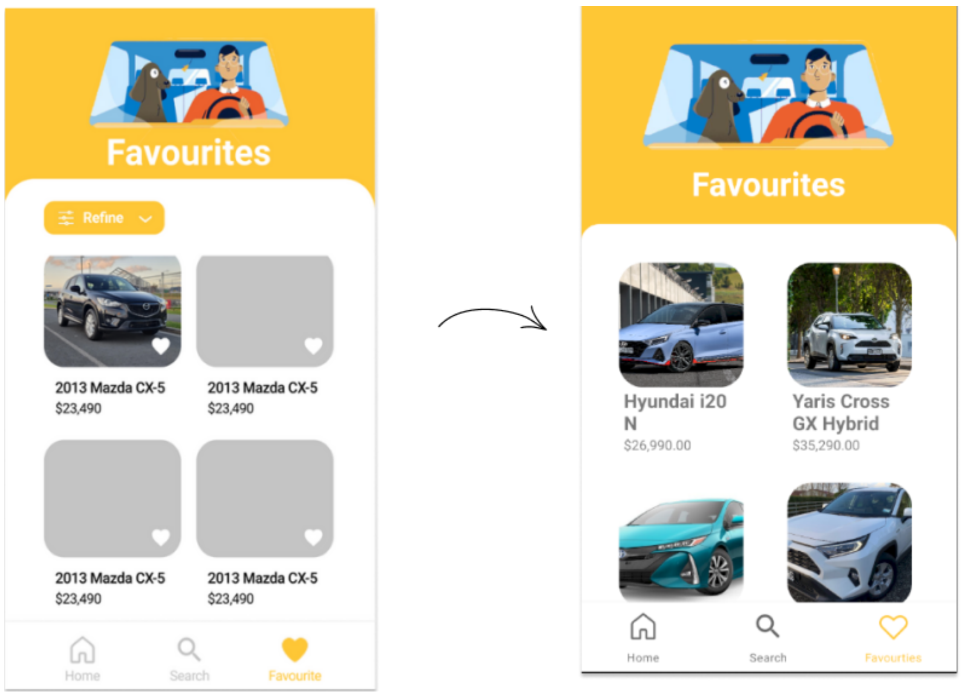


Figure 13. Comparison of the FavouritesActivity GUI mockup (left) vs. final product (right)

Throughout the development of the app, we have made some changes to the GUI of the app for better user experience, most notably:
The removal of the refine search in the Favourites Activity. This change was initially debated amongst members, as this functionality is also provided in many other views including the List Activity and the Search Activity, and was included in the Favourites Activity for the consistency of the GUI. But after a discussion, we realized that it is unlikely the user would want to use the refined search in the Favourites Activity, since if a vehicle has been added to the favourites by the user, the user would either already have some prior knowledge about the vehicle, or just generally interested about the features of the vehicle, and in both cases, the refined search would provide much help for the user to navigate in the Favourites Activity, and with the button removed, the GUI would have a better layout with cleaner design and more space for displaying the vehicle items, hence the refined search in Favourites Activity was removed.

Another change in the GUI of the application is the removal of the Favourites icon in the bottom right corner of vehicle's items. We originally included this feature so that the user could easily view if a vehicle has been favourited before, and tap on the heart icon to add the vehicle to the favourites list without going into the Details

Activity of the vehicle. Although our application does have consistently large icons and buttons for navigation, the heart icon still poses problems thoughtout the development. If the size of the icon is too large, it would obstruct the view for the vehicle in this format, and a miss click could happen if the user tries to enter the details view but accidently taps on to the heart icon, and adds/removes the vehicle from the favourites. In the case where the icon is too small, it would be either too hard to click on to. Overall, this feature brings more possibly for user misclick, and hence was removed for reducing the complexity of the GUI and prevent possible misclicks.

Design Changes

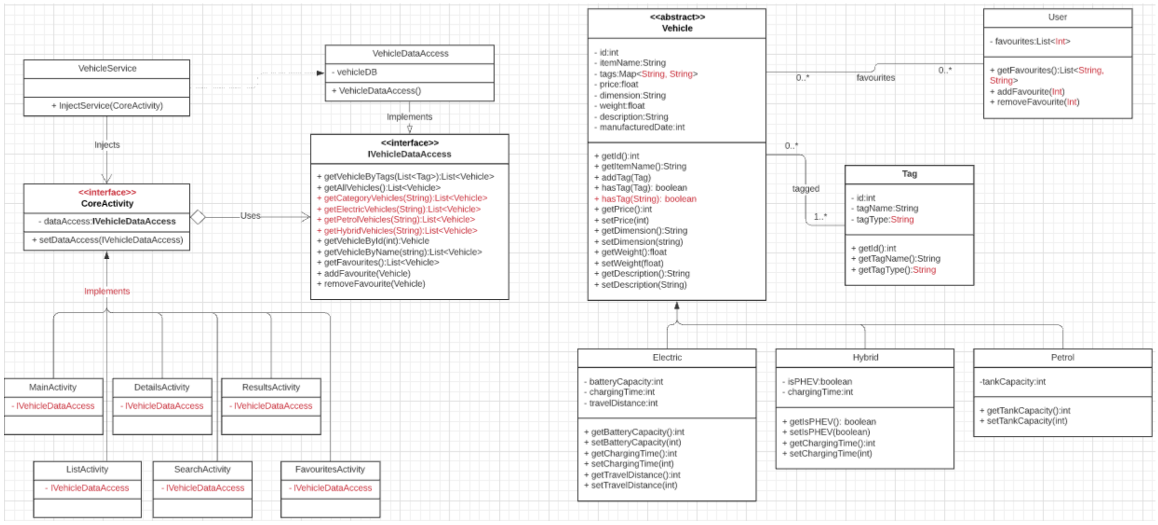


Figure 14. UML Class Diagram

The Design of the class diagram structure stays mostly unchanged from the original design, marked red in the figure 14. The bigger change being that the CoreActivity has been changed to an interface from an abstract class, and the use of Enum is removed from Tag.

The reason for changing CoreActivity is due to that all the activity classes needs to extend from AppCompatActivity and Java does not allow inheritance from more than one class, hence CoreActivity has to be changed to maintain its functionality.

The reason for removing the Enums used in Tag is due to the limited support of Enum types in Firestore Firebase, and hence string is used instead for representing the tag's type in Tag.

There are also several type changes for the fields throughout the development in many classes for more fitted use in the app.

Coding Practices

By following a set of software design practices and principles, it ensured us to produce good code. The first coding practice we adhered to was keeping consistent naming conventions. We agreed to follow the camelCase style and made all names as descriptive as possible. Common conventions were also obeyed, such as naming a method to begin with "get" when it returns an object's property.

We questioned certain features which could be sourced from online to avoid "reinventing the wheel" and account for time constraints. This included features such as the image slider [1] and like button [2] which were retrieved from GitHub libraries. We also avoided hard-coding as much as possible. For instance, all text displayed on the app was stored in a XML resource file called "strings" and referenced to when used.

Not only did we refactor the code to be optimal and efficient but also ensured the code was still readable. Most importantly, we commented on all functions/methods and any code that was particularly complex. Whilst developing the app, we properly utilised packages by bundling similar files together that have the same objective. For example, in Figure 15, the DataModel package holds all of the model classes whereas ViewModel stores all of the UI-related view models. By organising our code in this manner, we were able to give a separate identity to a group of classes. Not to mention, there were over 100 image resources alone, so packages helped us easily stow and locate different types of resources (seen in Figure 16).

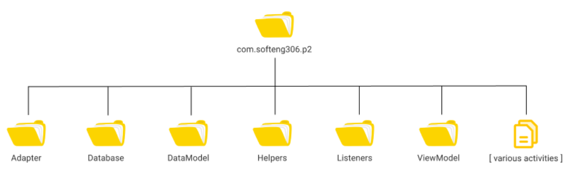


Figure 15. Structure of the packages within the 'com.softeng306.p2' folder

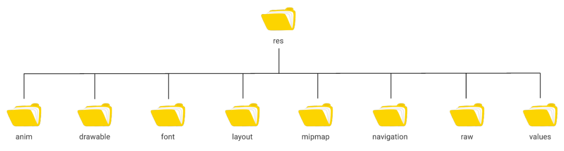


Figure 16. Structure of the packages within the 'res' folder

Lessons learned & Future work

For now the app only supports one user only, but it is possible to extend this function and allow multiple users. This will come with several additional requirements, things to consider including storing user credentials, a reliable authentication system and security for both the data and the app.

A possible improvement that could be done during design time is to design the data structure of the application more suited for Firestore Firebase. Due to our unfamiliarity with firebase, our data structure originaly was not supported by database with types like Enums and attributes that wouldn't be efficient in Firebase, and with how different Firebase is from other databases. This also caused some impact on the performace when retrieving the data from our app. With the experience leant when developing the app, it is possible for us now to come up with a better data structure and design such that it is more efficient and suited to use with Firebase.

In future, more UX testing would be performed with various participants from the target audience. Ultimately, after validating our design choices and conducting market research, we would like to deploy Motukā on Google Play.

Upon reflection of the team's performance, it was demonstrated that we worked effectively together. As we were teammates in the previous project, we were well acquainted with each of our strengths and could allocate tasks correspondingly. If any conflict arose, which was rare, we learned that communication and compromises were vital.

A couple of areas we exceeded in was our time management and organisation. To achieve this, we used the productivity software, Notion - a tool that provides a workspace in which each of our team members can view and edit simultaneously. By setting up this workspace, we were able to plan out a roadmap, record meeting minutes and collaborate on documents. In turn, we were able to achieve our goal of implementing an Android app to the standard we had planned for in the Design Doc. Overall, we are pleased with our final product and how the team performed throughout the project.

Conclusion

In conclusion, SOLID principles are applied throughout our project. The use of SOLID principles significantly helps us in achieving high standard software product as it promotes extensibility, maintainability, modifiability. The only exception is ISP as the product is designed for only one type of actor which is vehicle buyer.

References

[1] D. Coşkun, "GitHub - denzcoskun/ImageSlideshow: Android image slider.", *GitHub*, 2019. [Online]. Available: https://github.com/denzcoskun/ImageSlideshow. [Accessed: 23- Oct- 2021].

[2] J. Dean, "GitHub - jd-alexander/LikeButton: Twitter's heart animation for Android", *GitHub*, 2016. [Online]. Available: https://github.com/jd-alexander/LikeButton. [Accessed: 23- Oct- 2021].

[3] "Free Lottie Animation Files, Tools & Plugins - LottieFiles", *Lottiefiles.com*, 2021. [Online]. Available: https://lottiefiles.com/. [Accessed: 23- Oct- 2021].