Design Document - Team 11

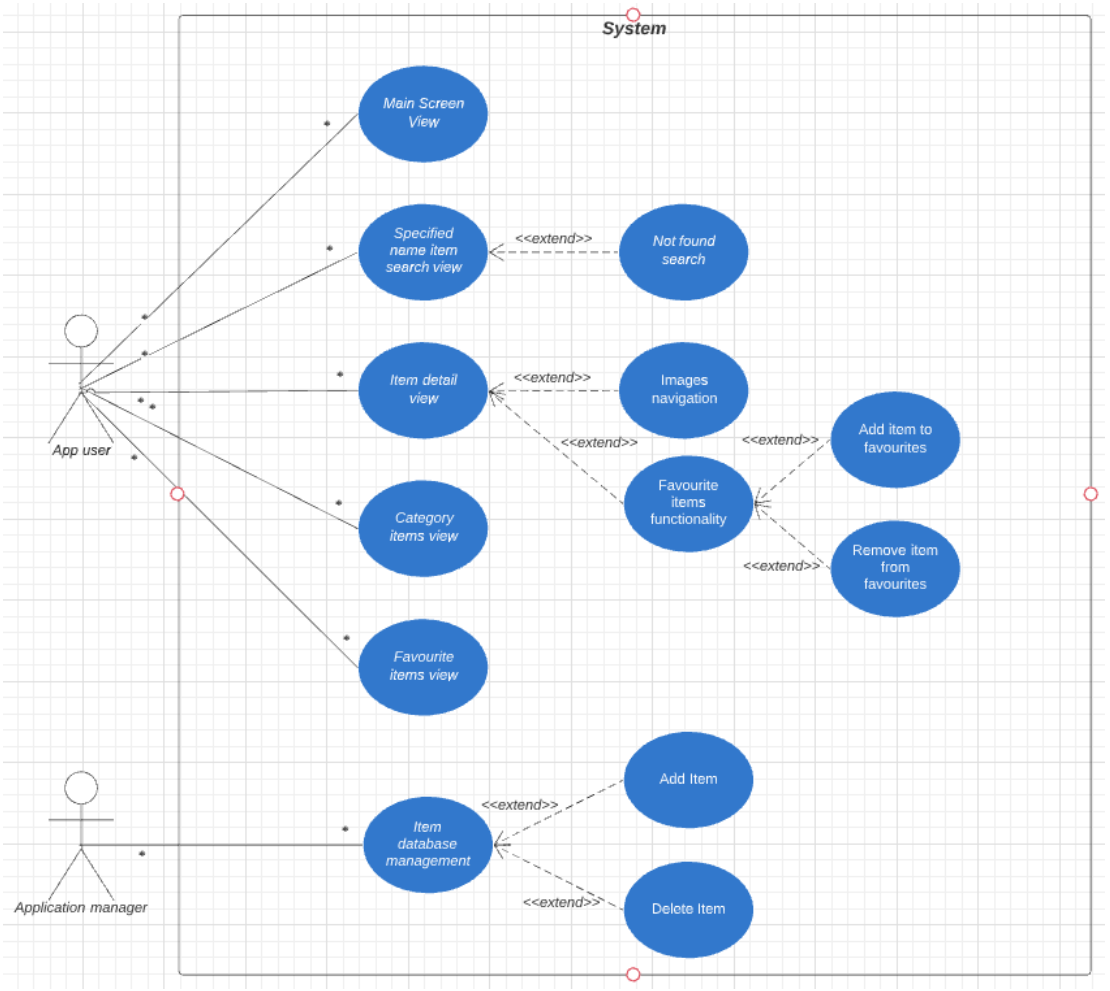
-- Jiayou Yao, Johnny Zheng, Raymond Li

1. Introduction

The aim of this project is to develop a native Android application that can be used to showcase furniture products and items. In which these furniture products can be further divided into a number of categories for instance Tables, Chairs, Bed etc. The user will be provided with a main screen that displays some popular items and a list of categories they may tap on and view products within that category. In addition, the app is also providing users with the searching functionality with input specified key words and related items will be displayed. By entering the item details page, information like details description and several images are provided and there is also a feature to add/remove this product to/from user's personal favourite list so that they can easily find this product next time.

The following design document begins with a use case diagram modeling the entire system, a UML class diagram describing the System design, the GUI mockup design prototypes, a data schema used to store products and items of this application and the team & project structure.

2. System Modelling



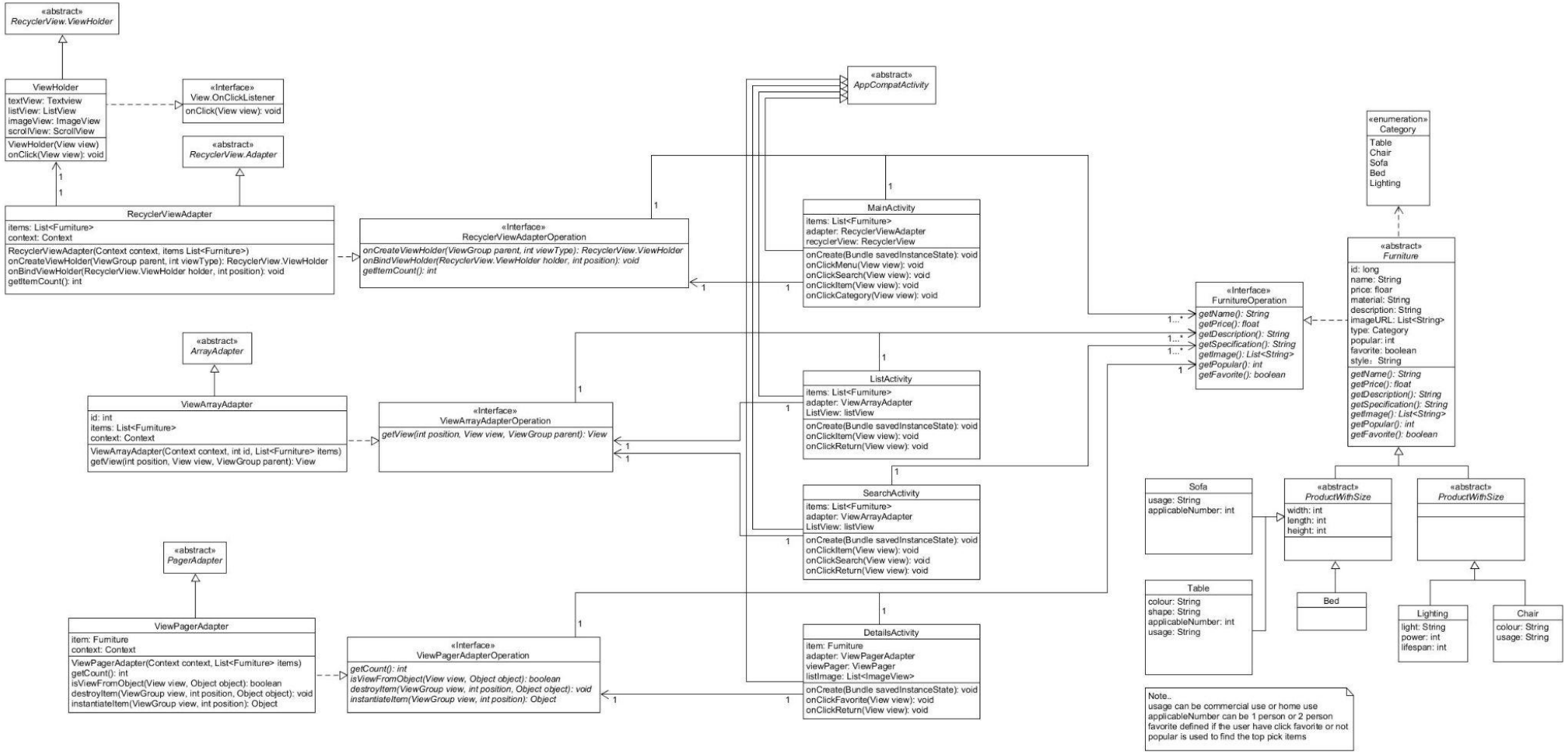
The above use case diagram describes a number of primary functionalities(use cases) identified in our application. There are two actors interacting with the system, which is our Android application. They are identified with roles: “App user” and “Application manager”.

Starting with the role “App user”, it is able to interact with 5 main sections of use cases.

1. The first one “Main Screen View” represents the use case of the user viewing the Main Screen activity page of this application.
2. The “Specified name item search view” identifies the searching functionality for the users that list matching products on the screen and the use case “Not found search” extends this use case. This is because a different response message will be displayed when there is no result for that search and this will be a different branch behaviour. So that it is presented as an individual use case.
3. The “Item detail view” use case represents the use case for viewing item detail pages, it also has two use cases extending this- ”Image navigation” implies the functionality for users to navigate through multiple images of this specific product. “Favourite item functionality” implies functionalities for users to add this item to favourites or remove it from the set.
4. “Category items view” represents the activity for viewing the list of items that all belong to a specific category.
5. “Favourite items view” represents the activity for viewing the list of items that is in the user’s “favourites” list, if no items are currently in “Favourites”, it will just display an empty list.

Another identified role “Application manager” interacts with one use case called “Items database Management” which has two use cases “add item” and “delete item” extends to it. Both implies operations manipulating item information in the database.

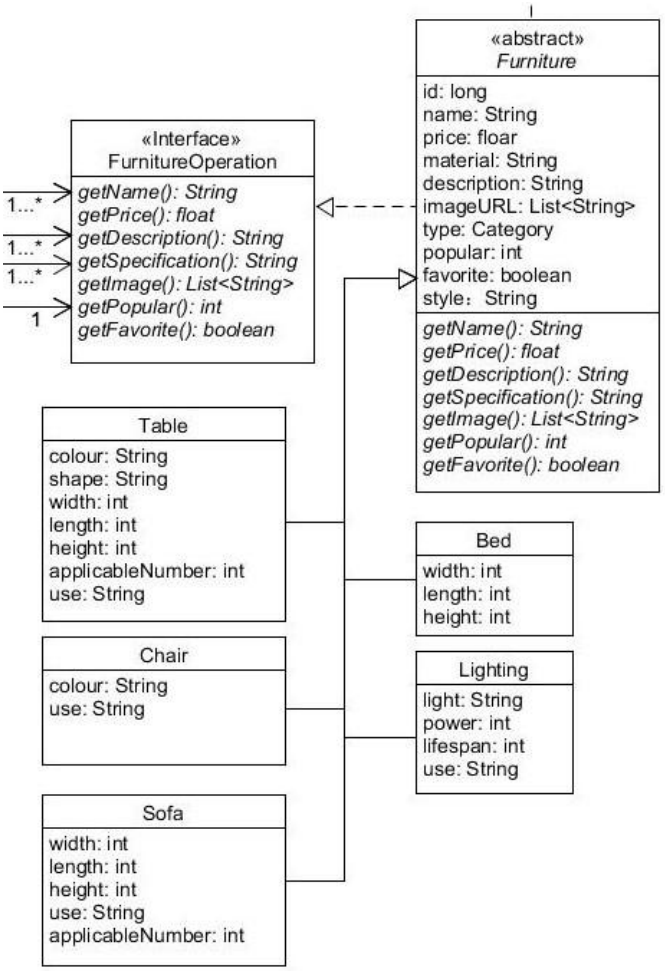
3. System Design: Provide a high-level class diagram



This class diagram is the latest version, which has been optimised after 4.c.

4. Design Analysis:

a. Explain what design smell(s) you identified in your initial attempts of the class diagram.



The above screenshot is our initial attempt of the class diagram, the team were able to identify several design smells.

The first one appeared was the Needless repetition. The Needless Repetition smell happens when the design contains repeating structures that could be unified under a single abstraction. For our initial design of class diagram, some fields in the subclasses of Furniture occur many times, for instance the width, length and height of the product. This type design smell may lead to repeated changes to all these fields if one of them needs to be changed, which is definitely what the team doesn't want. .After this smell is detected and removed, we then further unified those fields into the Furniture abstract parent class to remove this smell.

Based on feedback from other members in the team, the opacity smell is found to be another potential issue. The meaning of some field names is not clear, for instance initially there is a field called “use” in class Sofa. Which identifies the product to be commercial or home-use. However the naming of this field confused other developers in the team. Which indicates this may not be a suitable field name and leads to opacity smell in the class diagram.

b. For each SOLID principle, explain how it is applied in the class diagram. If a principle was not considered, explain why and justify.

The Single Responsibility Principle (SRP) recommends methods to perform only one task and a single class should be responsible to one actor. In the class diagram, this principle is enforced in the majority of the classes. For instance, the model in this application is a furniture product, a furniture product contains attributes such as name, price, image URL related to it. Meanwhile, based on the identified use case diagram, different types of furniture products need to be distinguished and will be put into different collections interacting with users(called categories). Products from different categories may have their own attributes others don't. Based on this condition, we implement the class structure for models to be having an abstract Furniture class and specific category items like table, bed, lighting to be subclasses of it. In this way , the SRP principle is preserved that class is responsible for one actor.

The Open-Closed Principle(OCP) is also well followed by the class diagram structure. From the definition of open for extension and closed for modification. New category models can be easily added into the current hierarchy of Furniture class with no change would be broadcasted to other entities in the application system. The case for activities in our application is also following the same rule that if new activity wants to be added in the further development, it will have no effect on the current existing ones.

The Liskov Substitution Principle(LSP) was considered when designing the hierarchy of models and preserved. Itt is achieved by only having necessary attributes and methods in the super class so that a newly added class will not violate any of them. Which matches the indication by LSP that any object that interacts with the superclass will be able to be instantiated by any of the subclasses without violating any policies and methods.

The interface segregation principle(ISP) is not considered in our class diagram as there aren't cases that need to leverage polymorphism and use of an interface for uses from different classes/users.

The dependency inversion principle(DIP) is preserved that classes in this diagram are associated through interfaces instead of concrete classes, such as adaptor classes and activity class, activity classes and model classes etc. In this way of design structure, when changes are made in those concrete classes, the interface linking various classes are less likely to be needed to change.

c. How the smell(s) identified in section 4.a are removed or improved, or even got worse in the class diagram.

Two additional abstract classes were added, one is **ProductWithSize**, and the other is **ProductWithoutSize**. They will all inherit the original Furniture class. Then all the furniture subclasses that have width, length and height fields can inherit ProductWithSize. If the subclass doesn't have any size data, it can inherit another ProductWithoutSize class. Rather than directly inherit the Furniture class, this modification can provide a better abstract for our project structure and reduce needless repetition. (removed the needless repetition smell)

For another opacity smell, we decided to modify the field name of field use (replaced with a more appropriate usage). Significantly, we add a note to the UML class diagram (just like explaining that

Furniture class is an abstract class) to explain the fields that may be vague. It will be easier for other developers to understand the meaning of the value stored in each object. In addition, a better understanding of project data can effectively improve development efficiency and reduce potential errors.

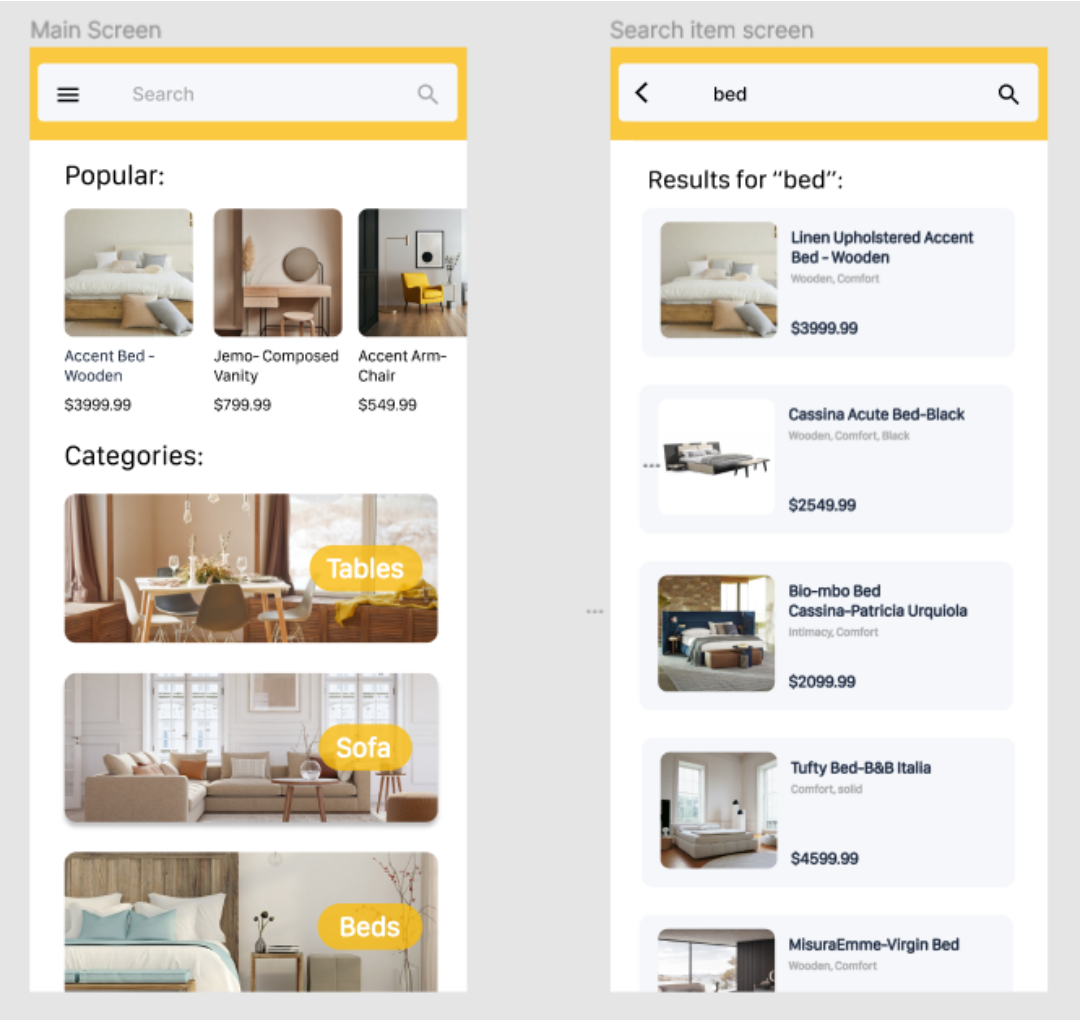
d. Analyse, compare, and explain your design with respect to maintainability, reusability, coupling, and coherency. Your judgment should be objective and based on metrics: justify your judgment by properly measuring software metrics and argue software quality attributes.

Before analyzing those 4 measurements, a few software metrics are first measured with our class diagram.
Starting with, our class diagram has max Depth in Inheritance Tree(DIT) value of 4, in the model furniture hierarchy. The root of this tree is the interface taking the role of association between furniture model and activities and adaptors. And nodes and leaves are furniture classes and divided "categories" classes including tables, beds etc.
The Number of Children(NOC) in model furniture class is currently 5, which means 5 direct subclasses of this Furniture class.
The average Coupling Between Objects(CBO) of this class diagram is calculated to be 2.64. Which is an indication of how much coupling exists. This low CBO value means that it is relatively less likely that changing one class will affect other classes in the diagram.

In the stage of structuring our class diagram, [model classes], [adaptor classes] and [activity classes] are put into different modules and can only associate with classes from other modules through interfaces. This reduces the degree of interdependence between them and contributes to a low coupling. The CBO metric value also reflects on this measurement. Meanwhile the degree of cohesion within each module is maximized such as those subclasses of Furniture class belong to this module closely. This structure also increases the reusability with the NOC metric supporting.
When considering the maintainability of this class structure, the DIT metric and NOC metric both support that the current system allows further development on the Furniture model class easily and maintainable. Adding a new category or changing existing categories will not have effect on the classes at the same hierarchical level.

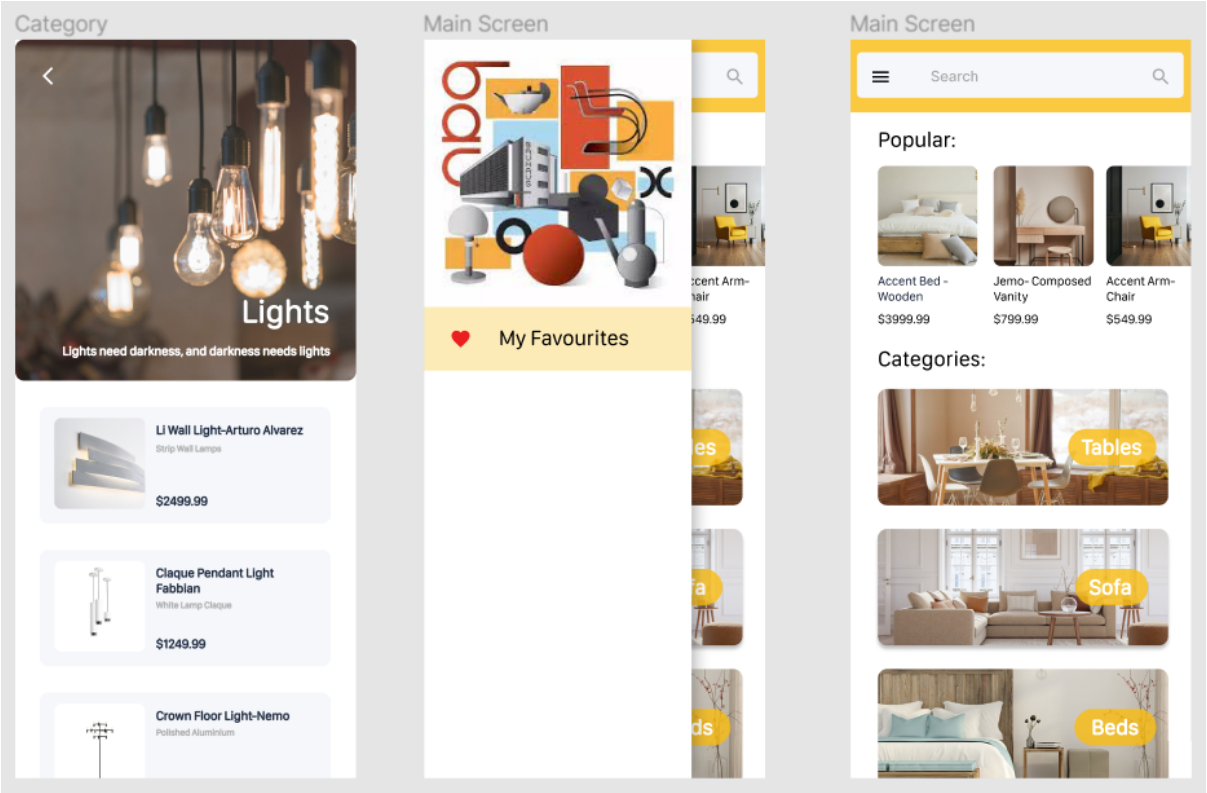
5. GUI Mockups:

The Graphic user interface of this application uses a consistent and thoughtful color theme, using bright yellow as the main element colour. Which intends to provide the users with a warm and comfortable feeling when looking at furniture products.



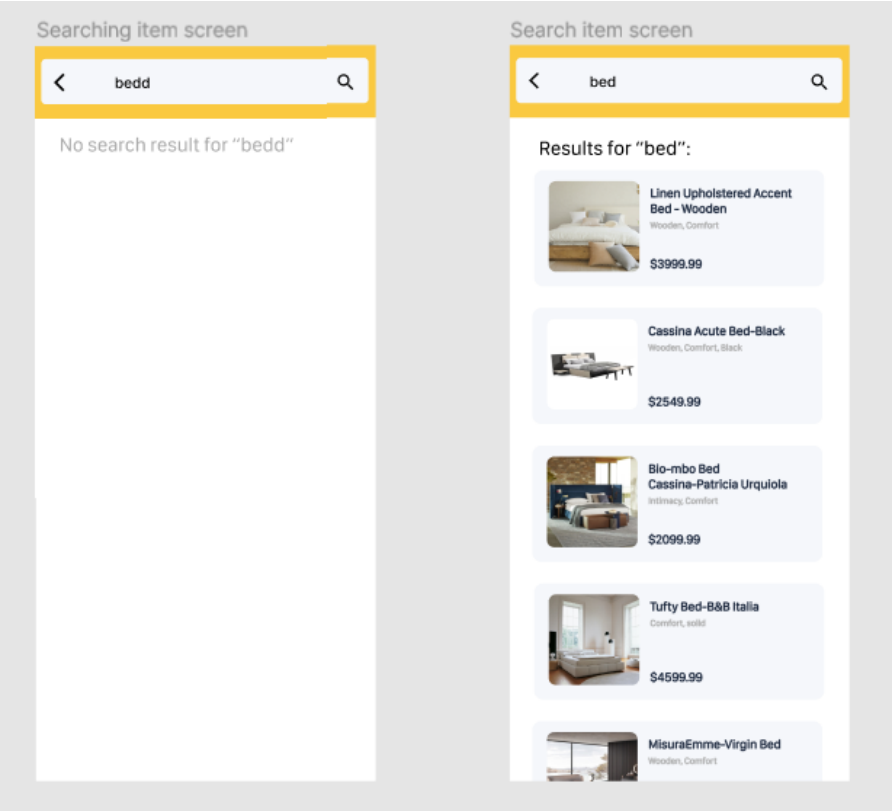
In MainScreen and search item Screen, a top app bar is designed to contain the searching and navigation actions. The top bar is persistent and always appears at the top of the screen and will disappear upon scroll by users. This provides a reliable way to guide our users through this app whether they want to search something or go to the navigation drawer on the side (only from the main screen). This app bar with consistent position and content follows the material design principles and can increase familiarity.

In MainScreen, the card-type containers used for displaying various categories are placed in a way that hierarchy is clearly indicated. This kind of layout allows the user to easily scan for relevant and actionable information. In this case the category card contains an image related to that category and a label with category name. It follows the material design principle of independent, individual and contained. Which means these containers stand alone, are identifiable as a single unit and cannot merge with other containers.

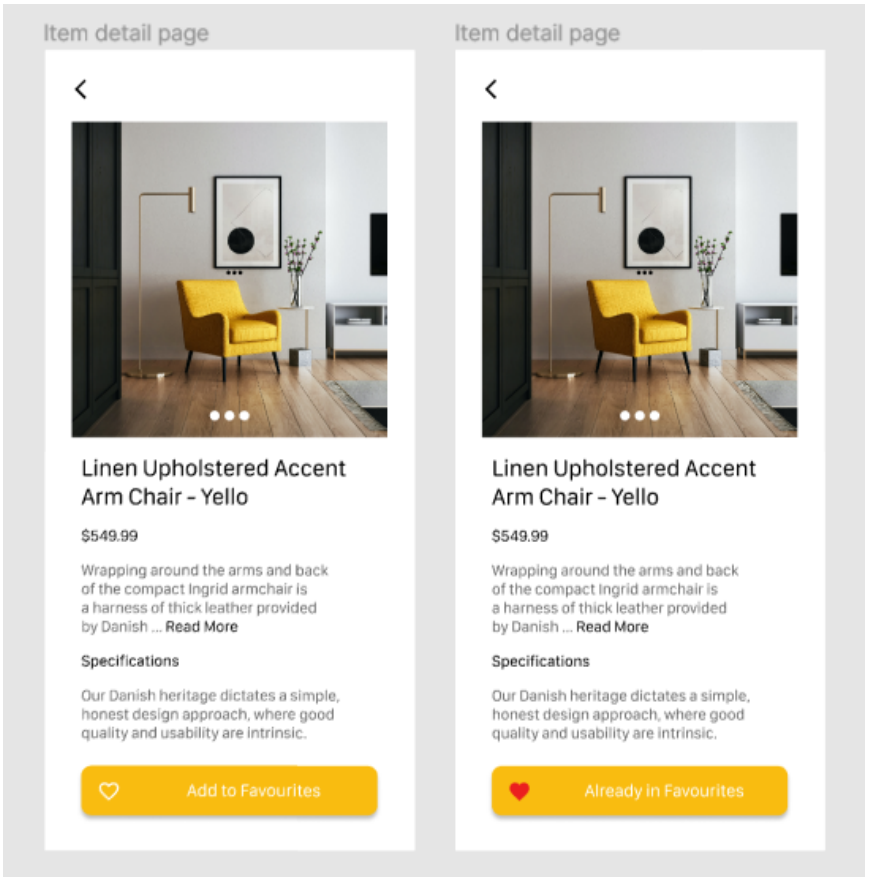


The frame on the left is the interface for items listview when the user taps on the category cards in the main screen and all products belonging to that category will be displayed to them. The title texts and background image on the top of this page can help users recognize the theme of this page easily.

The navigation drawer (the middle one above) accessed from the main screen currently only contains the “my favourite” section, which leads the user to the destination of viewing products that are in their favourite list. Although only one destination is currently being designed, the use of navigation drawer allows better extensibility and maintainability in further development of this application. It also follows the material design principle of identifiable that using list-style content of navigation drawers identify them as navigation to the user.



As identified in the use case diagram, a different message needs to be displayed when there is no result for that search. In our GUI design, a line of text is shown instead of the item lists.



In the item detail page, the application allows users to navigate through different images of that product. A number of white dots is placed at the center bottom of the image frame to indicate users with this functionality. They can interact with the images by either swiping pictures left and right or clicking dots to jump to the wanted one. In the detail description section below, by clicking the “Read more” text label allows the user to unfold the text description of the product if initially there isn’t enough layout for the description to be shown completely.

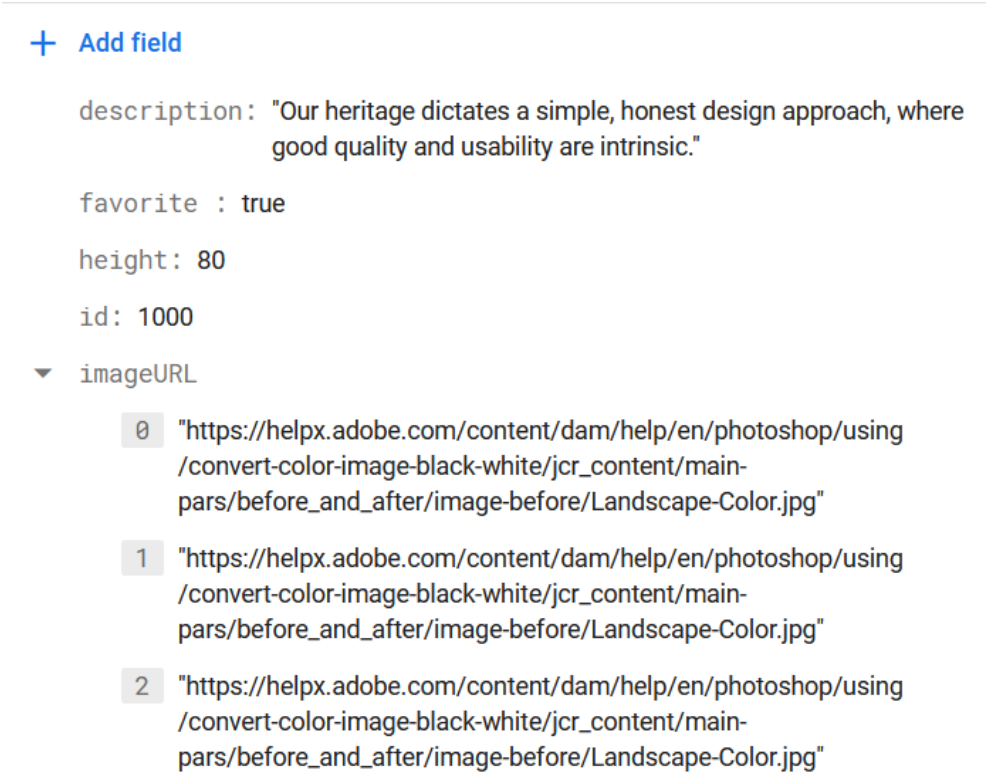
The floating contained button at the bottom of the item detail page allows the user to add this item to favourites or remove it by tapping the button one more time. The reason behind putting this functionality into a separated, emphasized button is to better allow users to notice the “favourites” feature and increase familiarity. The pattern of this fully contained button also follows the material design principles that this button is easy to be noticed/find among other elements, and it clearly indicates that it can trigger an action.

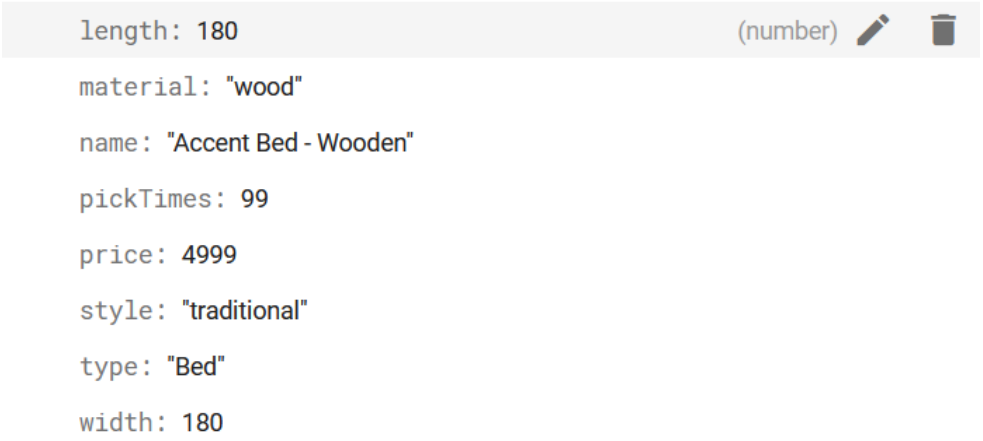
6. Data Schemas

id
name
price
material
description
imageURL
type
style
popular
favorite
colour
shape
width
length
height
applicableNumber

usage
light
power
lifespan

The firestone database of the project contains only one collection, which is utilized to store all the furniture items. In the class diagram, furniture is an abstract class, which is inherited by many different specific categories of furniture, such as table, sofa and chair etc. Because the firestore database in firebase is a NoSQL database, which is a document database, the document can store different <key, value> pairs that provide more flexibility to carry out the development than the traditional SQL database. Therefore, we can use this feature to store all the data and use the type field to identify the category when we get the document from the database. The above table is that all the data fields can occur in the document, but not all of them are included in a document. We can ignore any unnecessary fields and do not add them to the document. The screenshot below shows this feature, which is a Bed object stored in the firestore. We can see that it only store the information from Bed class and its expended class (ProductWithSize and Furniture class)





In addition, the imageUrl is an array of different strings, which will be used in the DetailActivity screen. It can also be seen as a sub-collection in a single document. This feature of NoSQL database is utilized by our team to store the list of image URLs in a single collection. So, there is no need to do any additional interaction with other collections, it can potentially increase the efficiency of the database operation. Furthermore, any change to a class only requires the single collection to be changed.

7. A. Team Roles

Section	Raymond Li	Jiayou Yao	Johnny Zheng
1. Introduction			x
2. System Modelling	x		
3. System Design		x	
4. A. Initial Design Smells		x	
4. B. SOLID principles	x		

4. C. Software Design Smell Analysis			x
4. D. Metrics and quality attributes		x	
5. Mockup GUI			x
6. Data Schemas			x
7.A. Role of each software engineer	x		
7. B. Gantt Chart	x		

Task allocation for the entire following project:

Front end implementation(Main, list and item activity views): Jiayou and Johnny
-MainActivity UI: Johnny
-ItemDetailActivity UI: Johnny
-ListActivity UI: Jiayou
-Responsive UI : Johnny
-Animation: Jiayou

Database implementation: Raymond and Jiayou

Searching logic and activity: Johnny and Raymond

Integration and testing: All team members

After discussion, the team has decided to distribute tasks into several smaller ones including front end(Android activities), searching functionality, database and APIs. Smaller groups of members work together on specific tasks in order to maximize the product quality and meanwhile ensure different sections of the project can be developed simultaneously.

7. B. Gantt Chart:

SoftEng 306 Project 2

Team 11

Project Start: Tue, 9/21/2021

Display Week: 1

TASK	START	END
Planning		
Project Plan	9/21/21	9/21/21
Problem definition	9/21/21	9/21/21
Project search & Brainstorm	9/22/21	9/23/21
Context decision	9/23/21	9/23/21
Use case Diagram	9/24/21	9/26/21
GUI prototype design	9/24/21	9/26/21
Software architecture	9/24/21	9/27/21
Data Structure	9/24/21	9/27/21
Design Document	9/27/21	9/28/21
Implementation		
Implement firestore database	9/29/21	10/3/21
Database integration testing	10/4/21	10/5/21
Implement MainActivity UI	9/29/21	10/1/21
Implement DetailsActivity UI	9/29/21	10/1/21
Implement ListActivity UI	10/2/21	10/4/21
Implement Searching functionality	10/5/21	10/7/21
Responsive UI	10/8/21	10/9/21
Animation	10/10/21	10/11/21
Integration and testing		
Middleware, integration	10/12/21	10/14/21
Integration Testing	10/15/21	10/16/21
Product demo preparation	10/17/21	10/18/21
Project Report	10/19/21	10/22/21