SOFTENG306 Project 2 Team 5
Final Report

Marcus Li
Do Hyun Lee
Danil Linkov

1 Introduction

SE Tech is an android application aimed at showcasing computer parts specifically CPUs, GPUs, and motherboards. The application was designed to be minimalistic and easy to use. This is because its main goal is to showcase computer items and therefore the information needs to be easy to find, read and understand so as to be as effective and as useful to the user as possible. This was achieved through multiple iterations of colours, fonts, and UI layouts as well as database schemas aimed at maximizing the efficiency of searching all items in the application.

The application consists of 4 views/activities consisting of a home screen (Main activity), items in category screen (List activity), search results of all items and categories screen (Search activity) and a details page of a selected item (Details activity).

Some extra features implemented in our application are “Sort by” feature in Search activity and List activity. It allows the user to sort the displayed items by price, name and view count in increasing or decreasing order. We have also implemented the ability for users to search for items in list activity the same as they would be able to in search activity. Lastly we included a “Most viewed in category” side scrolling list in Details activity below all the item information. This is to allow the user to have more freedom when browsing for items they are interested in.

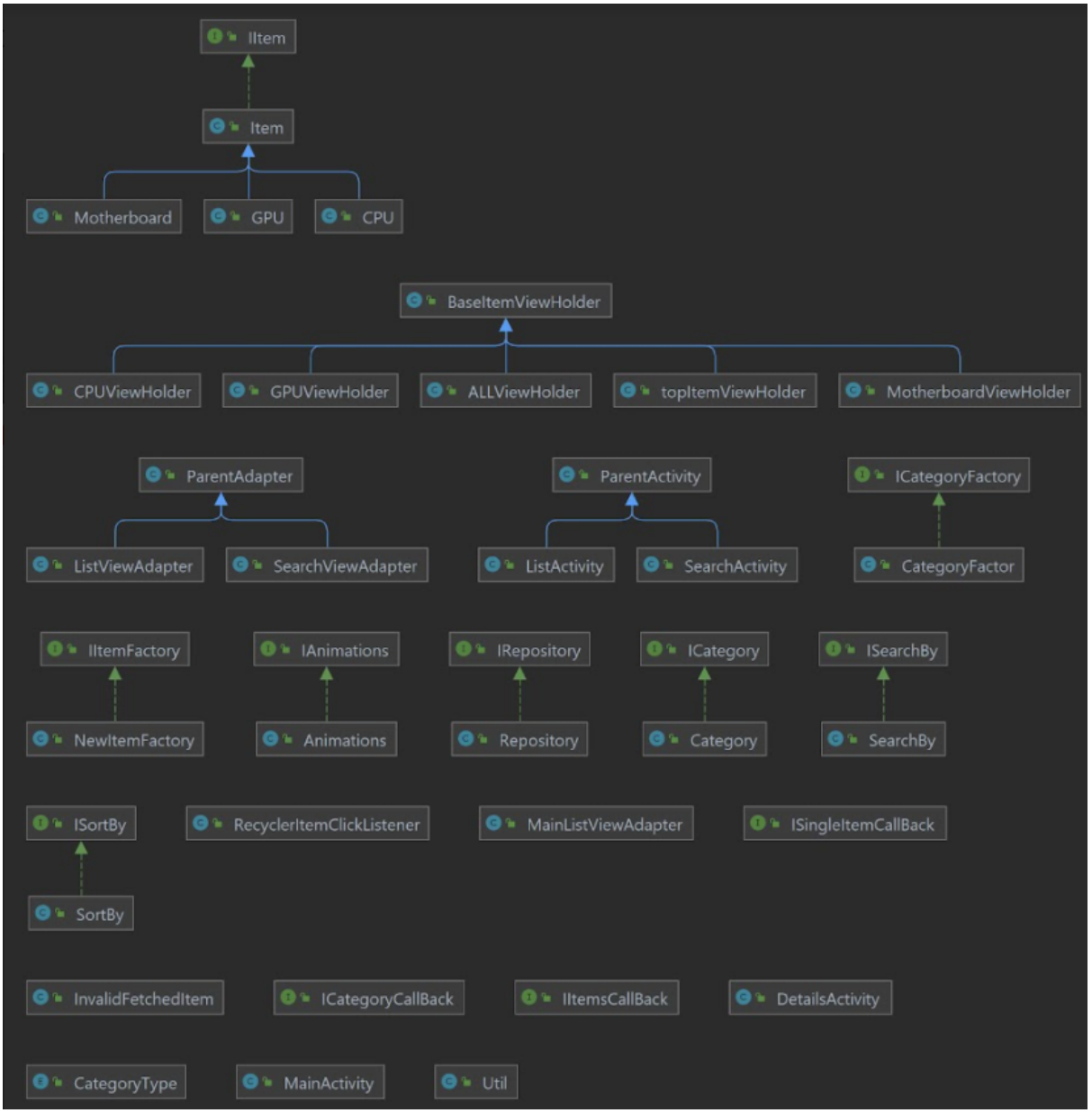
This document describes how the final application follows the SOLID principles, any diversions from the plan, what good coding practices were used, extra features and the final UML diagram of the design.

Github repository to the application:
https://github.com/SoftEng306-2021/project-2-project-2-team-5

2 SOLID Principles

Below we have provided links to jpg images of the diagrams that we will be referencing in our solid principles. We have provided multiple views of the UML diagram as well as the inheritance tree to allow for easier marking with different perspectives. Please use the provided google drive links in order to see a proper view of each diagram.

Inheritance tree partial (no params or methods):

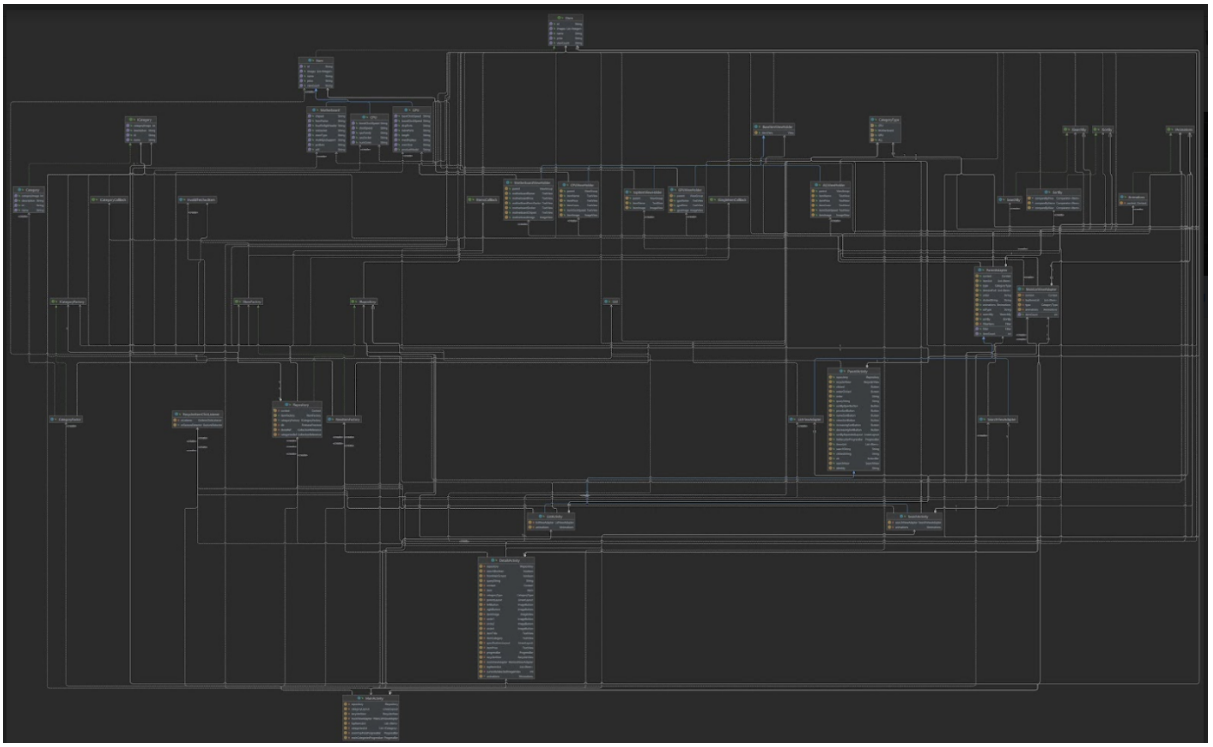


https://drive.google.com/file/d/1mljApG1Q4Wi0ogONx40iE3EBZ5YTIKNx/view?usp=sharing

Inheritance tree:

https://drive.google.com/file/d/1FX0QwJ1dNzKUgg8rQ3Ig0l544tCG3C2p/view?usp=sharing

UML diagram:

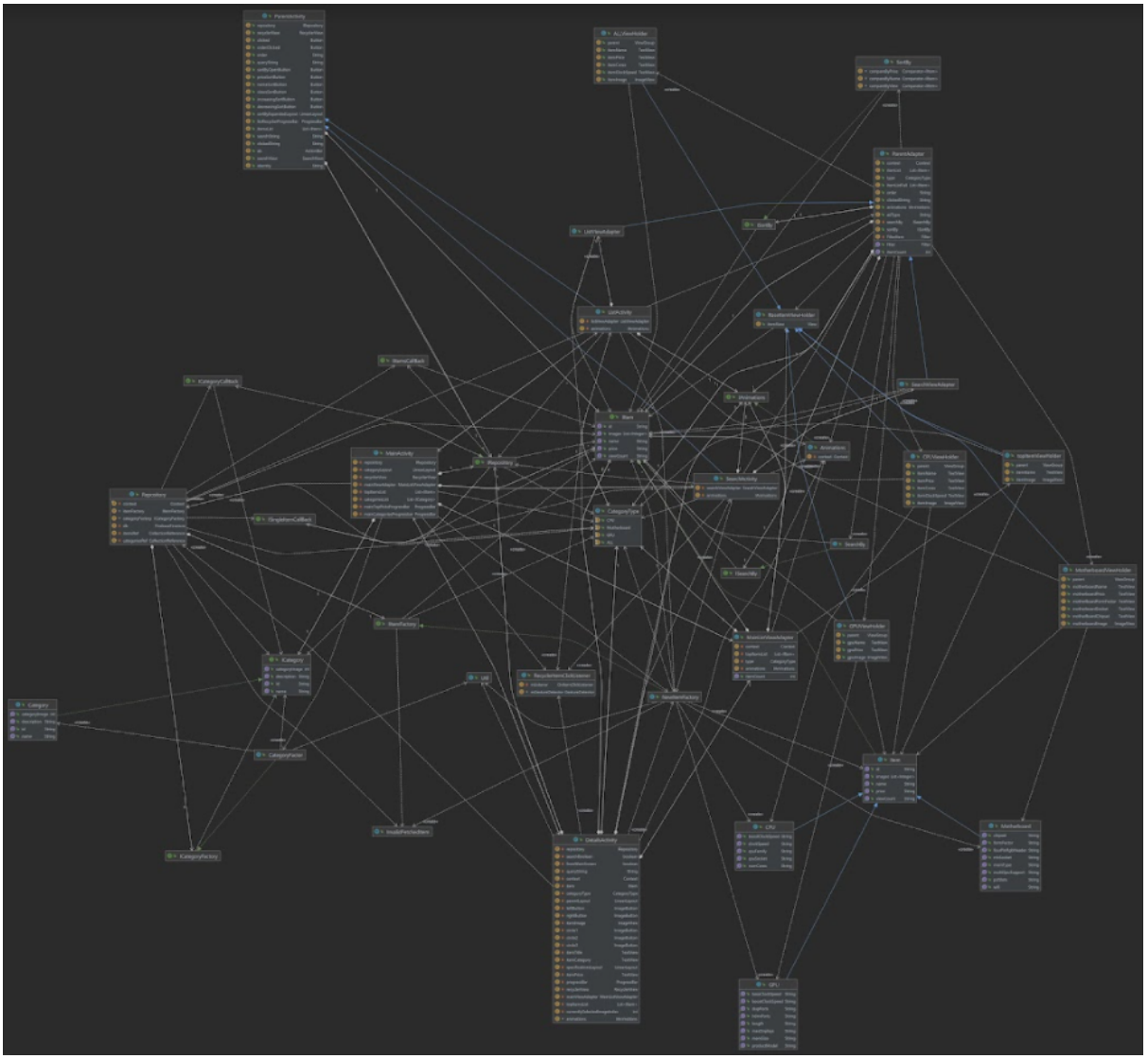


https://drive.google.com/file/d/1ZvjPG-LYcsxGGW64kqnVdDZne2og3eGu/view?usp=sharing

UML diagram compact:

https://drive.google.com/file/d/1xw_RZDLVmTAN-nCANgHn2dD16gZxeAWo/view?usp=sharing

UML diagram edge full:

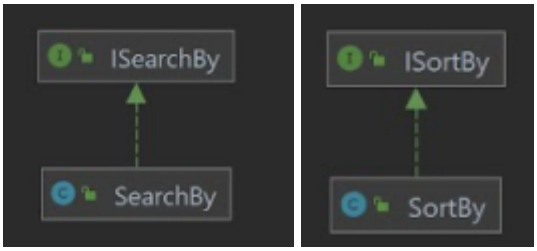


https://drive.google.com/file/d/1tyVPptJlXHwQRg99gENXXfYGXAG6yGIK/view?usp=sharing

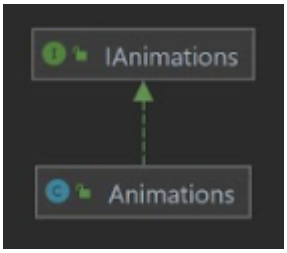
2.1 Single responsibility principle

The single responsibility principle was followed throughout the application through the heavy use of interfaces. This allowed us to split up modules into smaller ones until high cohesion was achieved and each module was responsible for only the required logic.

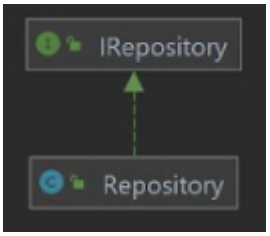
Good examples of where this was applied are:



ISearchBy and ISortBy interface which SearchBy and SortBy classes extend respectively. This means that the logic for search filters and sorting item lists is separate from where it's required and therefore providing better cohesion for classes such as Search and List activity.



IAnimations and the Animation class that implements it. This allows for the animation logic to stay outside of the activities and therefore also improves cohesion.

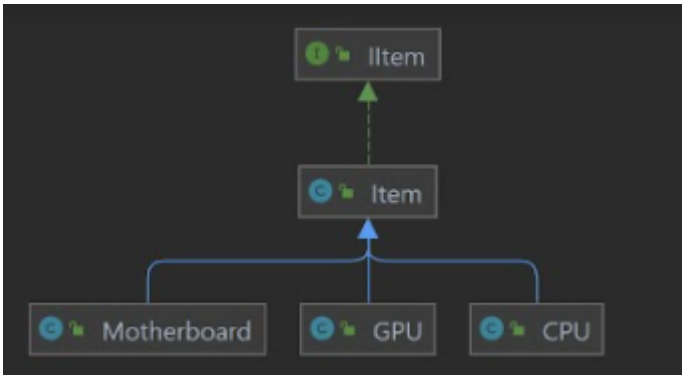


Another example is IRepository and the Repository class that implements it. Initially we had each activity have fetching logic to fetch the required data from the firestore database however this was later refactored into IRepository and Repository. This allows the database interaction logic to stay outside of the activities and therefore again improves cohesion and helps follow the single responsibility principle.

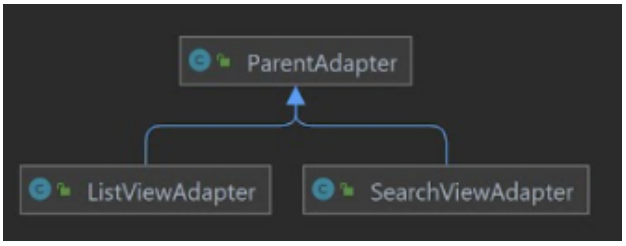
2.2 Open-closed principle

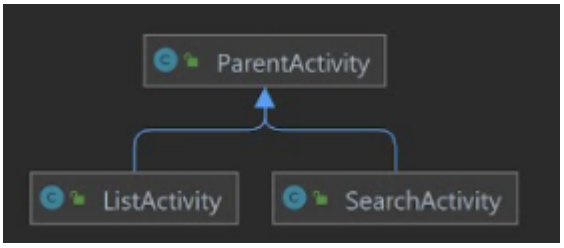
The open-closed principle was followed through the use of inheritance. This allowed us to remove a lot of duplicate code by creating a parent that contains the code that is duplicated and then the required classes inherit from it.

Good examples of where this was applied are:



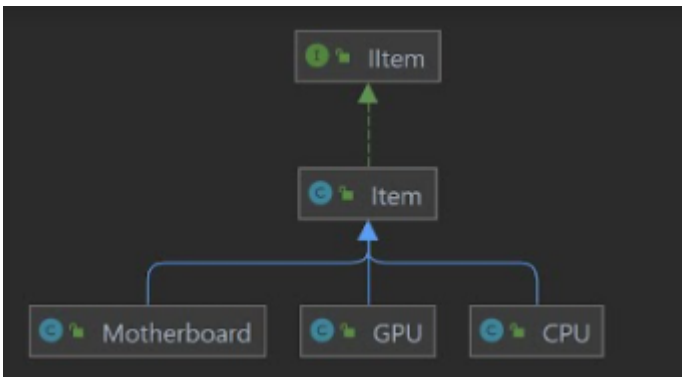
IItem interface which the Item implements and then any specific item such as GPU, CPU and motherboard can inherit from this and extend it by adding any extra parameters or methods that may be required. This means we can easily add any category item by just extending the Item class.





Another good example of the use of the open-closed principle is with the List and Search activities which both have very similar functionality in terms of both allowing to sort and search items. Therefore we have a Parent Adapter and Activity from which we can extend on which will allow us to reuse the searching and sorting functionality of items and therefore allowing us to extend on those features if needed.

2.3 Liskov substitution principle

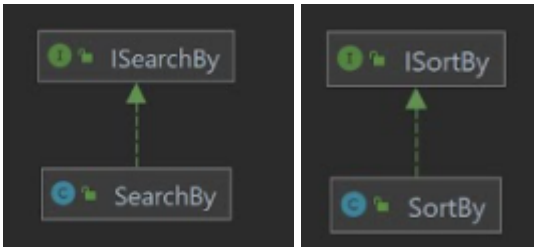


The liskov substitution principle is not specifically used however our application still does follow it as for example any Item children can be used to replace the parents functionality if required.

2.4 Interface segregation principle

The interface segregation principle is not specifically used however if required our application does follow it as we have opted to use a lot of interfaces throughout our code base.

For example:



If required a class could implement both ISearchBy and ISortBy and combine the logic such as sort items differently depending on a specific search query.

2.5 Dependency inversion principle

A lot of the examples already provided also demonstrate good application of the dependency inversion principle and show how our code follows it. The use of this principle has allowed us to easily make modifications to our code as we could easily swap out a different implementation of a given interface and therefore only had to change the areas of the code where this implementation is injected.

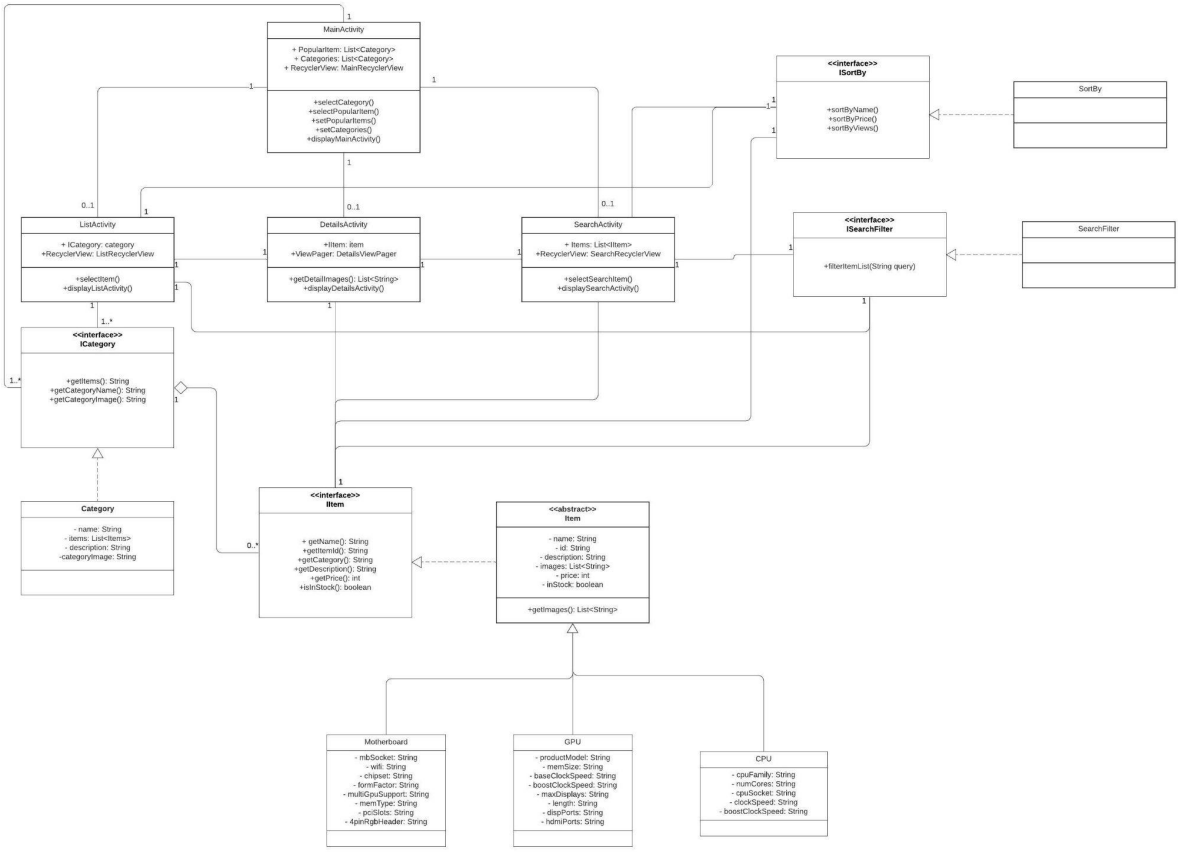
For example:

In the image above multiple interfaces can be seen that were used to allow for dependency injection and for our code to depend on abstractions instead of concrete implementations. Therefore for example we can replace the repository implementation with another implementation and only change the areas of code where it is injected in such as all the activities.

3 Diversions from the Plan

3.1 Consistency with the design doc (UML)

Original UML diagram:
https://drive.google.com/file/d/1znWRA8bRmTd-N5yooA8xUfHR_9xbkAvc/view?usp=sharing



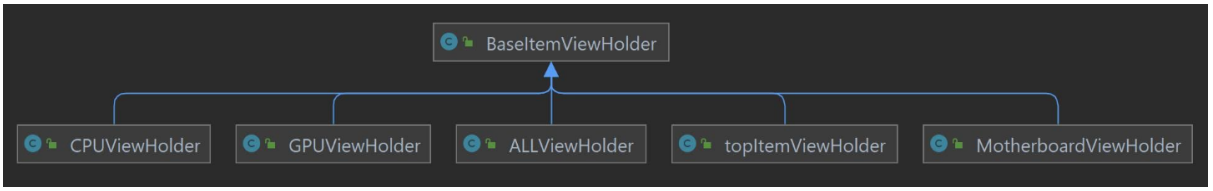
Throughout the code there were many diversions from the UML diagram however the general logic/layout has stayed consistent.

Some specific changes to it were:

Adding a repository interface and class for the activities to use in order to fetch the data. This was done in order to better follow the SOLID principles, specifically the single responsibility one. As it allowed for us to keep the logic dealing with the database interaction separate from all the activities and any other classes that required it.

Our original UML diagram was also missing the adapters and viewholders for the recycler views that were used throughout the application.

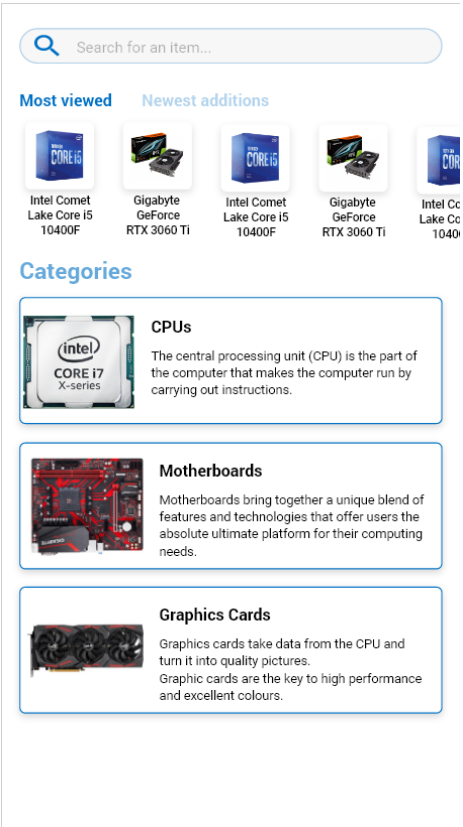
Such as:



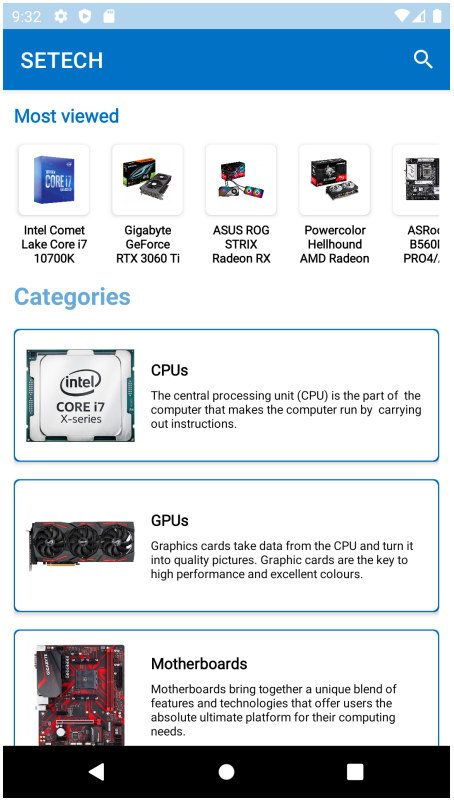
Also a factory pattern was used in order to instantiate items and categories fetched from the database which was not originally planned. The reasons for this change are explained in section 4.1.

3.2 Consistency with the design doc (GUI)

In terms of the graphical user interface, there were 3 diversions between the final application and the design doc.
Firstly, an extra feature of Newest Additions was removed. This feature would display the items that were most recently added into the database. The team decided that this additional feature was not necessary as the database itself was being populated by the team and not by other third parties. However, a field called dateAdded was kept in the Firestore database to allow for the possible implementation of this feature in the future.
Additionally, the search toolbar layout was slightly altered to provide consistency throughout the application. The initial design had a different layout of the search bar, resulting in inconsistent design and possibly causing confusion to the user. Hence, the search toolbar was changed to keep it consistent throughout the application and to follow Neilsen's usability heuristics of consistency and standards. These first two alterations can be seen in the two screenshots below.
Initial GUI:

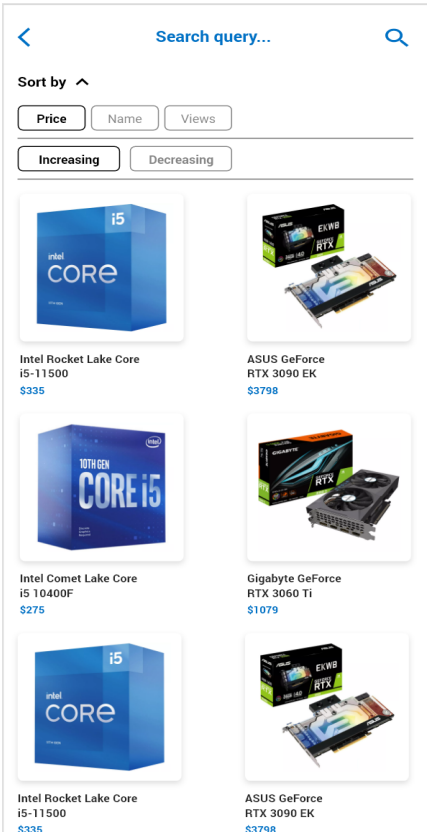


Final GUI:

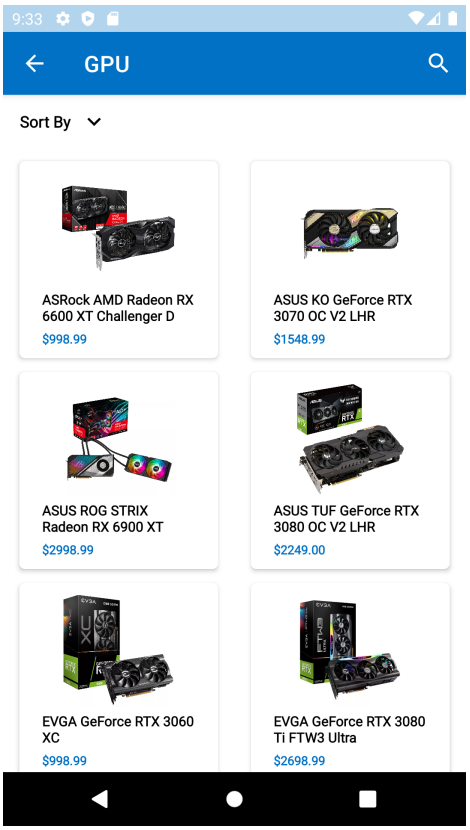


Finally, the card view of the GPU list was changed slightly as there was too much spacing between the item card views. Changing the card view so that there would be less spacing between the items on the screen would reduce the chance of the user misclicking the empty space thus needing to click again. User convenience was an objective of high importance to the team hence the team decided that this change was necessary to improve user experience.

Initial GUI:



Final GUI:



3.2 Other notable changes

The viewCount value for all the items inside the database was originally planned to be an Integer however we then thought we might want to display it down the line therefore made it a String.
Also we originally planned to have a dateAdded value for each item however during development we decided to remove the “Newest added” feature therefore this value was no longer required but is still kept in the database in case for any plans to use it again down the line.

4 Good Coding Practices

Good coding practices are important for any software engineering project as it can increase efficiency, minimise complexity of the program and can make it easier to maintain. Below are some examples of good coding practices we used in this project.

4.1 Factory pattern

Factory pattern is introduced in this project instead of direct object construction for Item and Category objects fetched from that database. Factory pattern is chosen to allow the project to follow interface segregation and dependency inversion principle. It allows mock testing of the code and gives more flexibility when changing implementations without

changing the dependent codes. The Factory classes in this project are NewItemFactory class for Item object and CategoryFactory class for Category object.

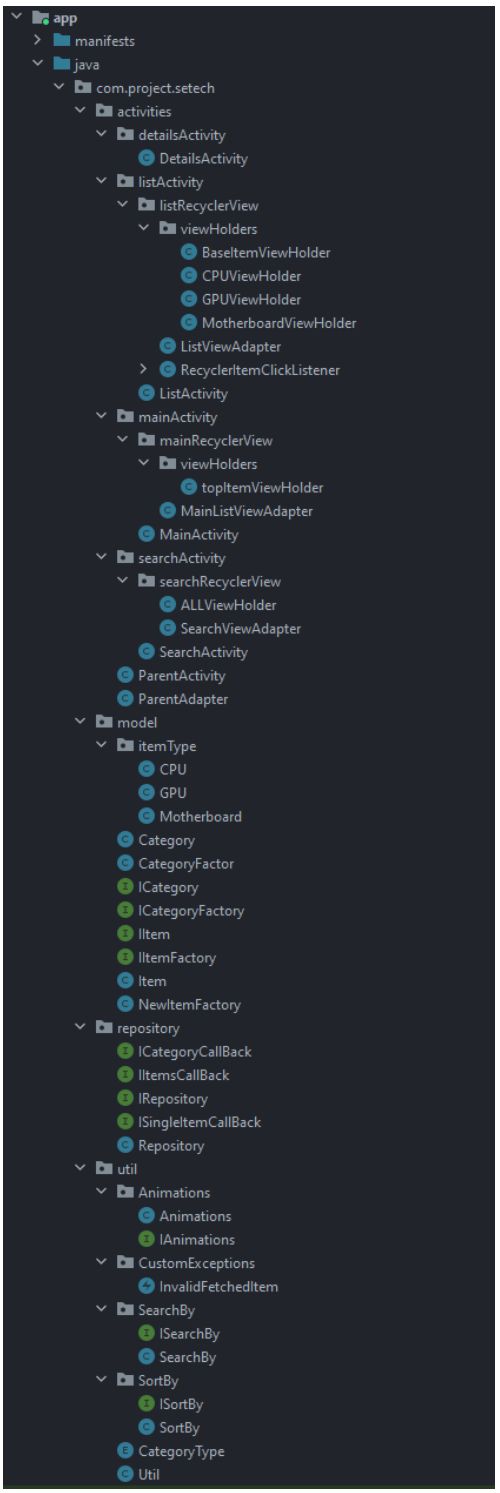
4.2 Inheritance

Inheritance is introduced in this project to allow polymorphism and reduce duplicate codes. An inheritance leads to less development and maintenance costs as well as providing a clear model that can be easily understood. First example of inheritance in our project is the parent class BaseItemViewHolder, it extends RecyclerView.ViewHolder. Children classes like CPUViewHolder, GPUViewHolder, MotherboardViewHolder, ALLViewHolder and topItemViewHolder all extend from BaseItemViewHolder. Second example is the creation of ParentActivity and ParentAdapter parent classes. ParentActivity class is extended by ListActivity and SearchActivity classes to reduce the duplication of methods and code. Code defined in parent class can be reused in child class without rewriting it, new codes can be added on top of that to make that method different from other children. On the other hand, ParentAdapter is extended by ListViewAdapter and SearchViewAdapter with the same reason.

4.3 Interfaces

Interfaces are introduced in this project with a sole purpose of forcing a clean separation between interface and implementation. Interfaces include a set of abstract methods which other classes can implement. Example of uses of interfaces are ICategory, ICategoryFactory, IItem, IItemFactory, ICategoryCallBack, IItemsCallBack, IRepository, ISingleItemCallBack, IAnimations, ISearchBy and ISortBy. All these interfaces are implemented by their corresponding classes which provide a separation of interfaces and implementations.

4.4 Package structure



This is the package structure that we followed during the building process.
The main parent packages are:

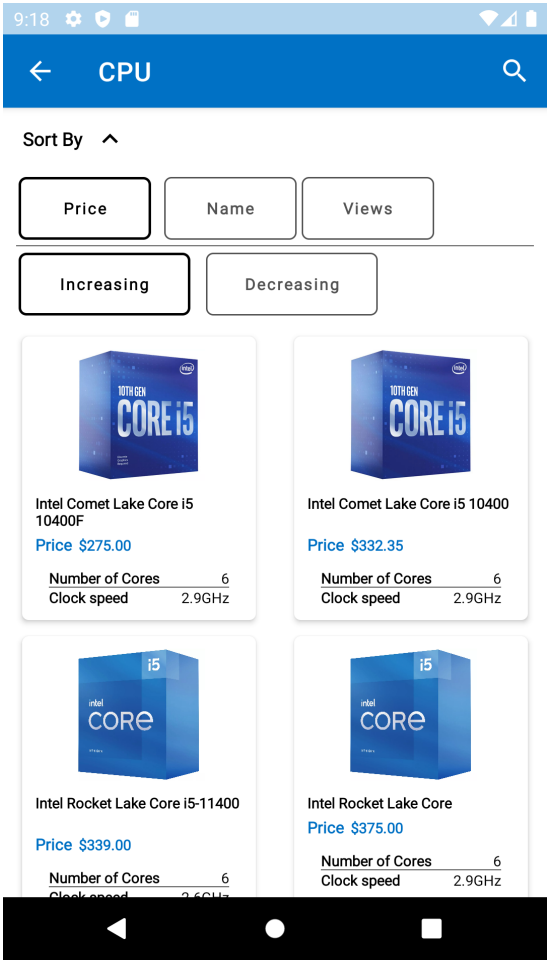
Activities - Contains a package for each activity and any other logic required with them.
Model - Contains the model interfaces and classes as well as any factory pattern logic.
Repository - Contains the code for interacting with the database.
Util - Anything that may be used throughout the application and is fairly general such as Animations and Exceptions.

5 Extra features

SE TECH has multiple extra features that ultimately aim to achieve better user experience and convenience. The four extra features implemented by Team 5 were Sort By, Search in List Activity, Most Viewed in Category, and Animations.

5.1 Sort By

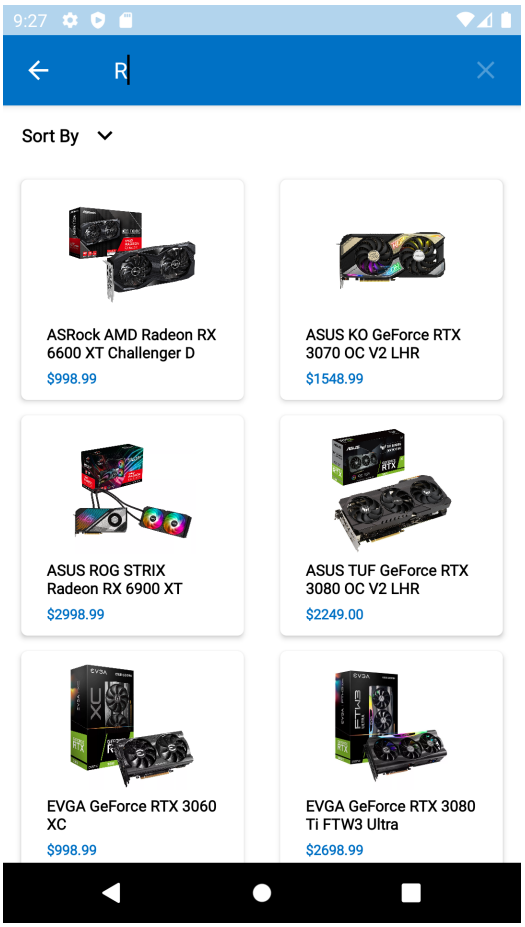
SE TECH provides an extra feature that can sort items by price, alphabetical order, and by the number of views. The order can also be in ascending or descending order. After thorough research into similar mobile applications such as SE TECH, it was decided that the Sort by feature would be an essential functionality for this application.



5.2 Search in List Activity

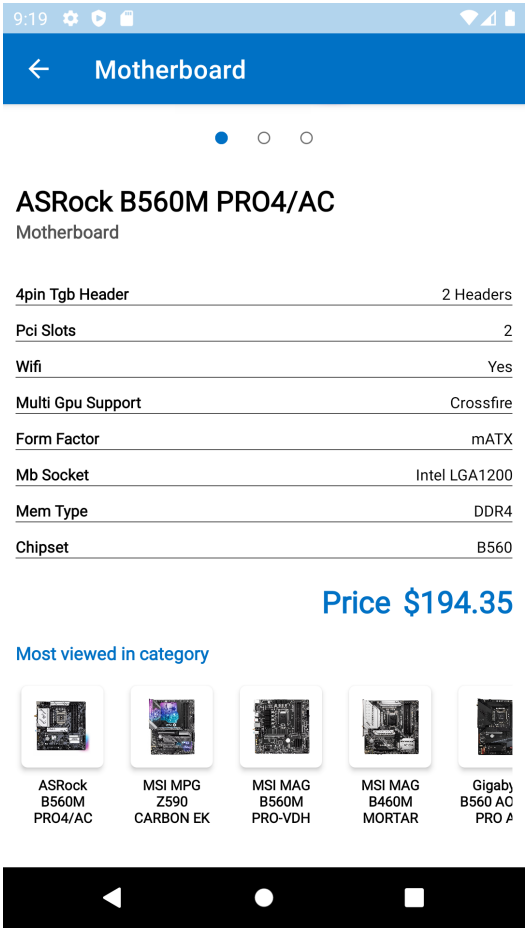
Another extra feature that was added in SE TECH was the search functionality in the List Activity screen. Whilst a search functionality in the MainActivity would be searching for an item within the entire database, the search functionality in List Activity would serve a slightly more specific purpose. This search functionality would only search for items that matched the search query only in the specific category that they were in. This would allow for the user

to be able to search for an item easier if they were already aware of what type of PC part the item was.



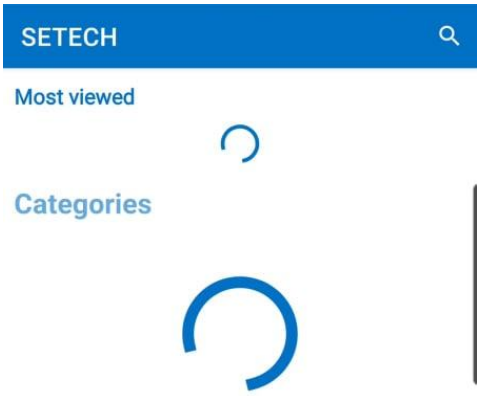
5.3 Most Viewed in Category in Details Activity

The addition of the most viewed items in the Details Activity screen was an extra feature that the team agreed to be quite important. To further achieve the goal of convenience and ease of use, the most viewed items list would display items that were only of the same category, as the user would most likely be searching for different items of the same category. Although this initial feature was not present during the design stage of the project, the team unanimously agreed that such a list displayed on the bottom of the Details Activity screen would be of perfect fit to our ultimate goal of user convenience.



5.4 Animations and Progress bar

Animations were used throughout the entire application to ensure smooth transitions and flow between activity screens. The animations used were fade, slide up, slide down, and fall down and a progress bar animation. The progress bar animation in particular would be used while the data was being fetched from Firebase to let the user know that the application was loading and not frozen. Since all the activity classes used animations, all the animation methods were put in a separate *Animations* class and an interface for the animations class, *IAnimations*, was used for abstraction and loose coupling.



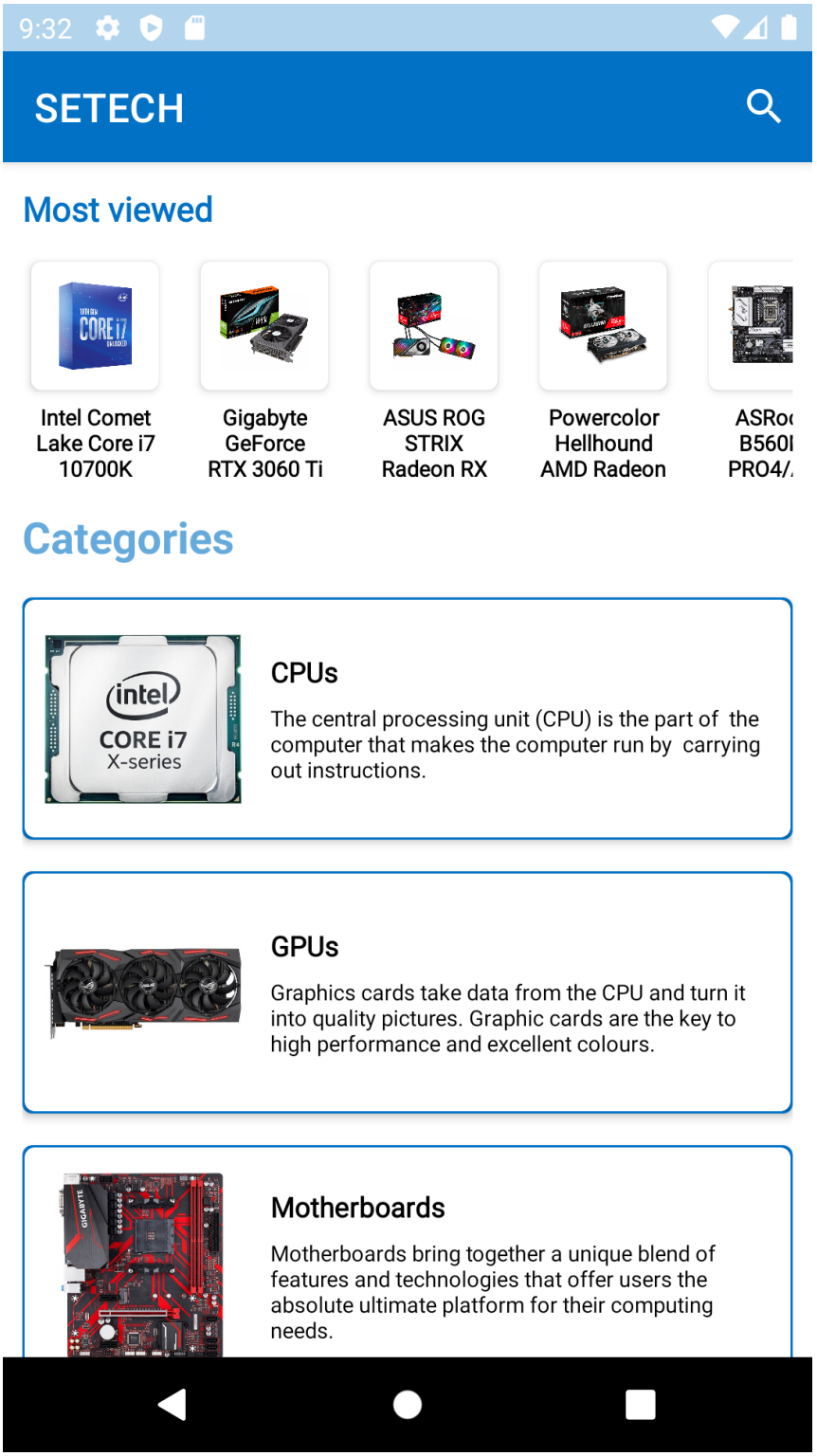
5.5 Application Icon

To make SE TECH more related to the theme as a computer parts showcasing application, we have decided to change the icon of the application from a default icon to a thunder icon representing technology.

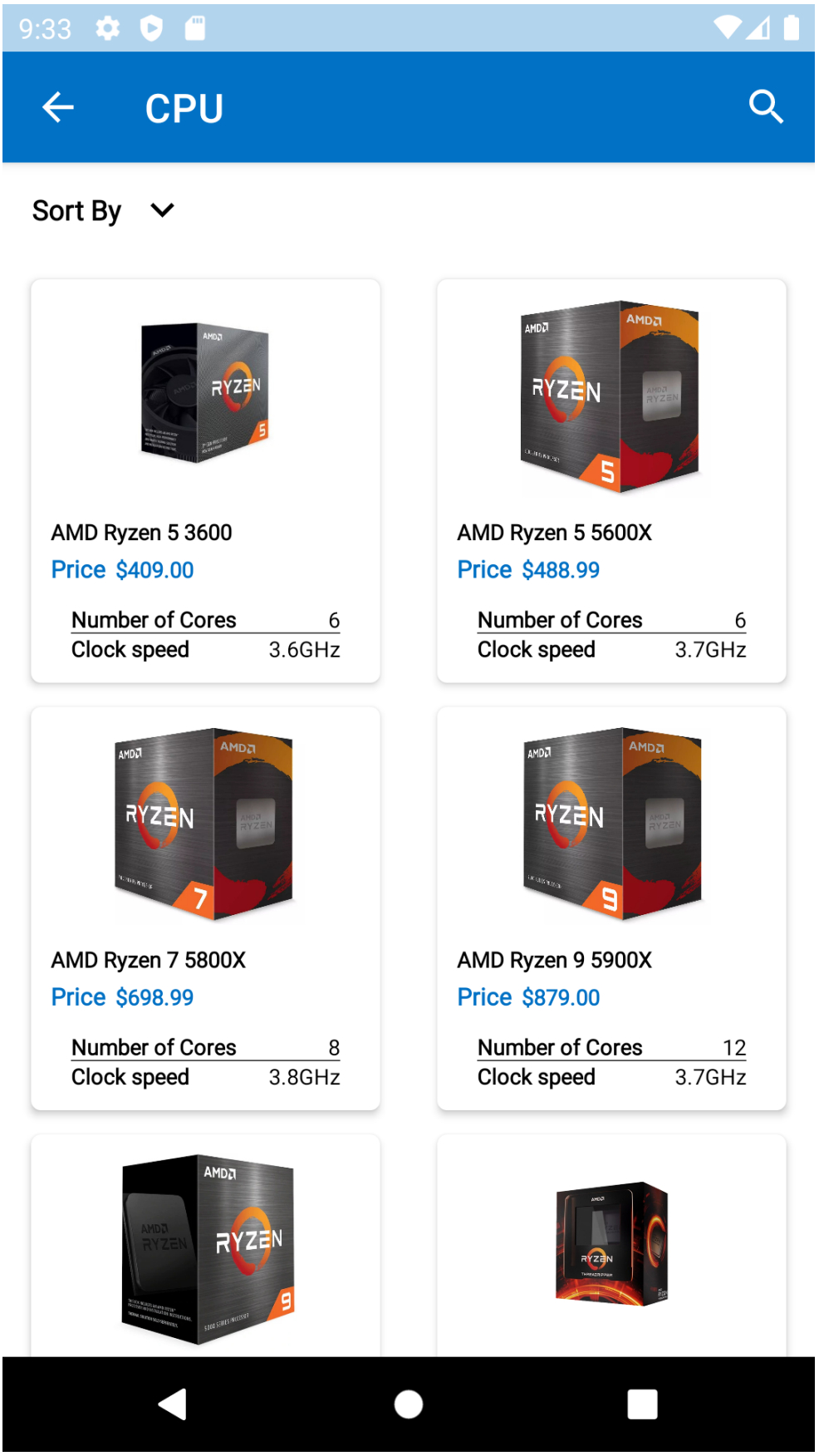
6 GUI designs

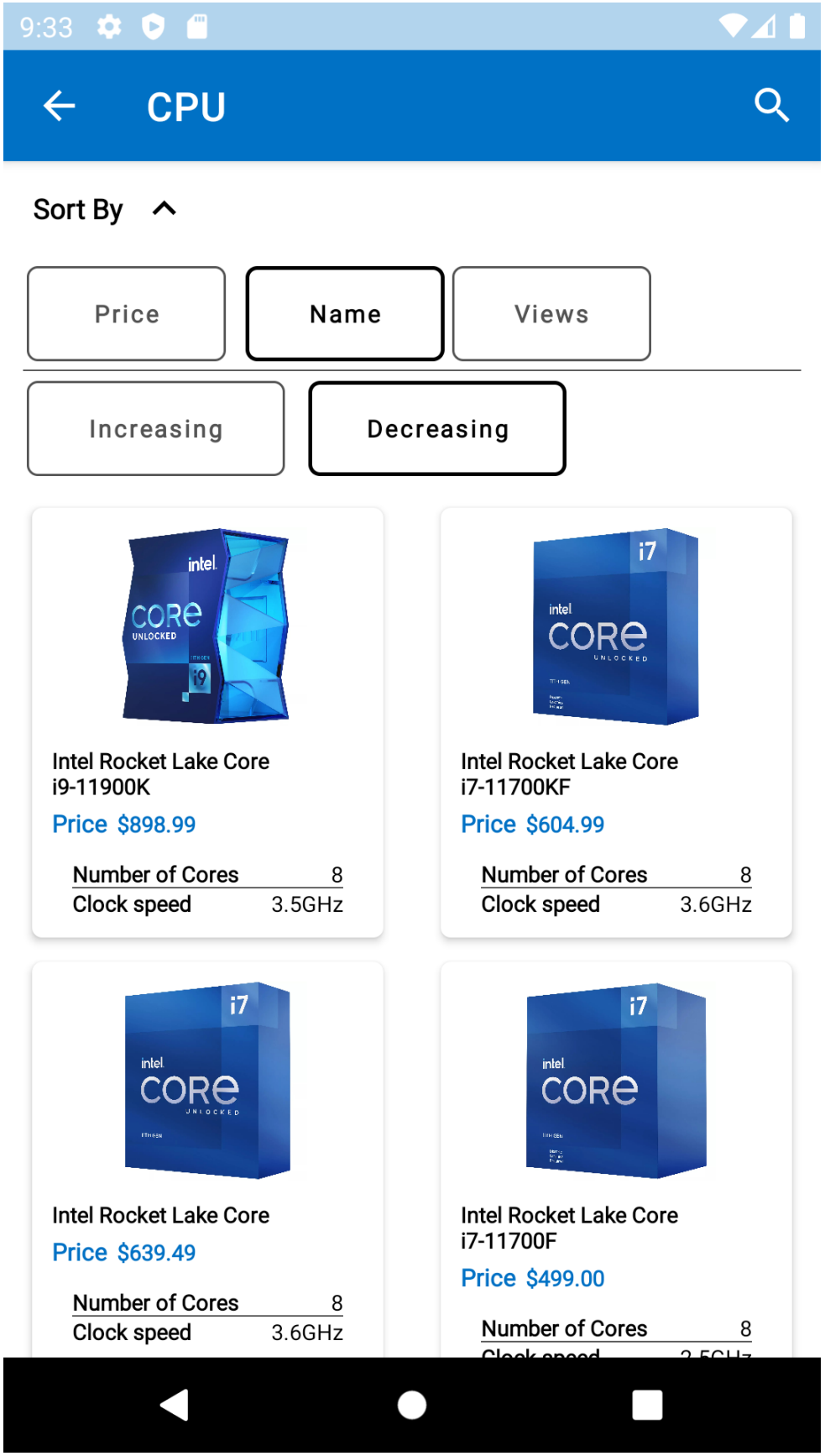
The following GUI screenshots are to provide a visual representation of the application without needing to launch the application itself.

6.1 Main Activity



6.2 List Activity




9:33
CPU
Sort By
Price
Name
Views
Increasing
Decreasing
Intel Rocket Lake Core i9-11900K
Price $898.99
Number of Cores 8
Clock speed 3.5GHz
Intel Rocket Lake Core i7-11700KF
Price $604.99
Number of Cores 8
Clock speed 3.6GHz
Intel Rocket Lake Core
Price $639.49
Number of Cores 8
Clock speed 3.6GHz
Intel Rocket Lake Core i7-11700F
Price $499.00
Number of Cores 8

9:33

GPU

Sort By

ASRock AMD Radeon RX 6600 XT Challenger D

$998.99

ASUS KO GeForce RTX 3070 OC V2 LHR

$1548.99

ASUS ROG STRIX Radeon RX 6900 XT

$2998.99

ASUS TUF GeForce RTX 3080 OC V2 LHR

$2249.00

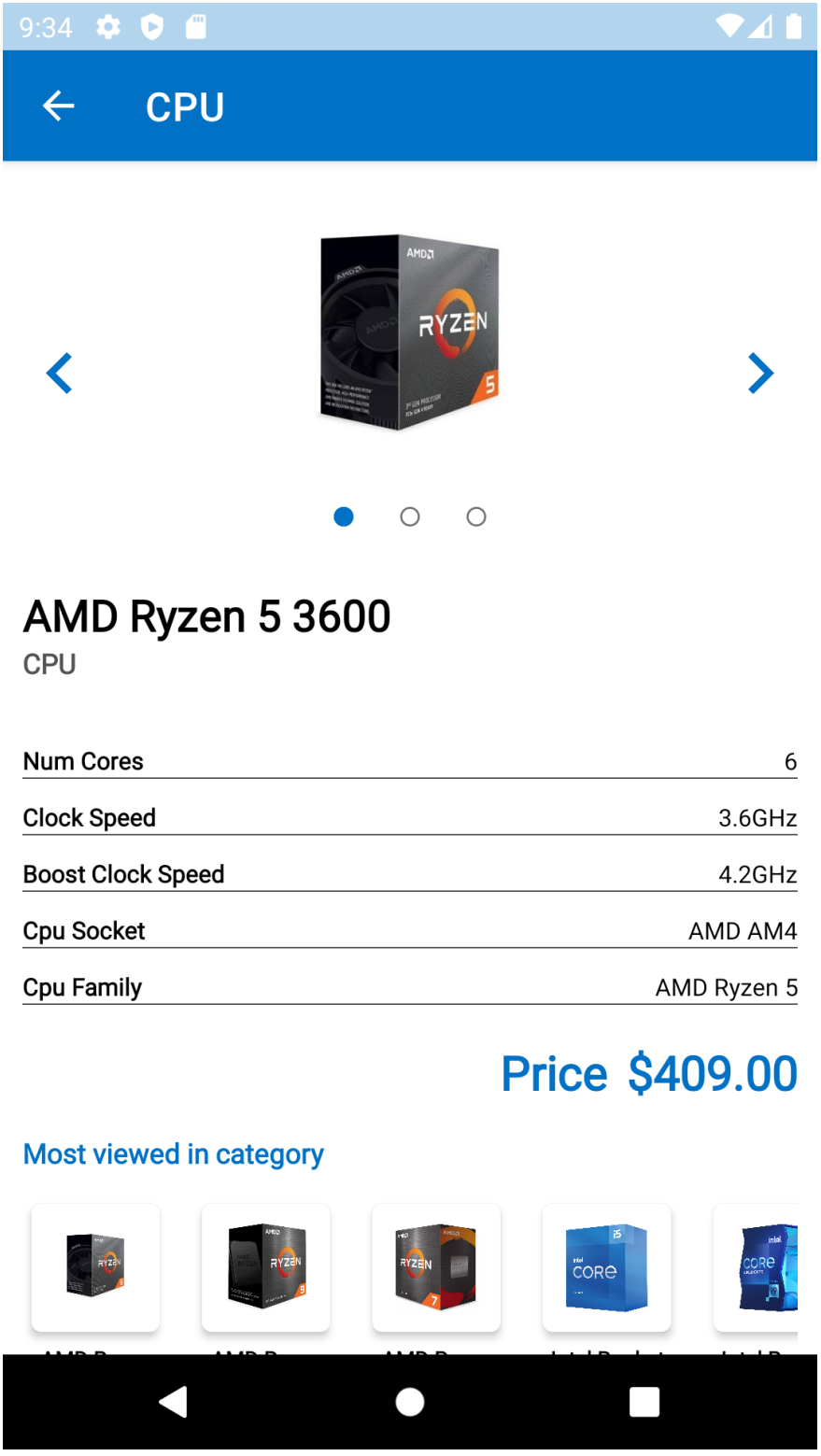
EVGA GeForce RTX 3060 XC

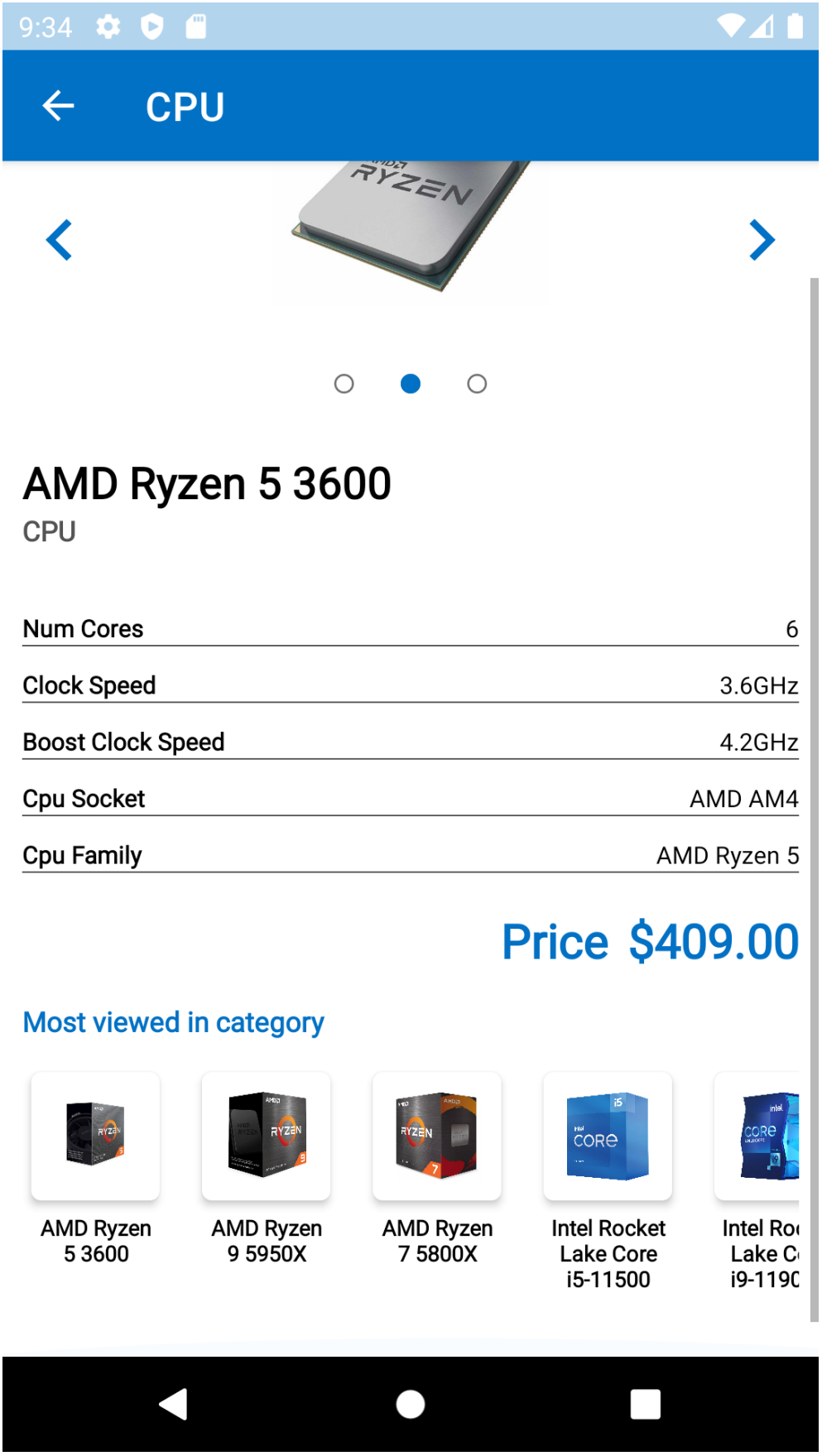
$998.99

EVGA GeForce RTX 3080 Ti FTW3 Ultra

$2698.99

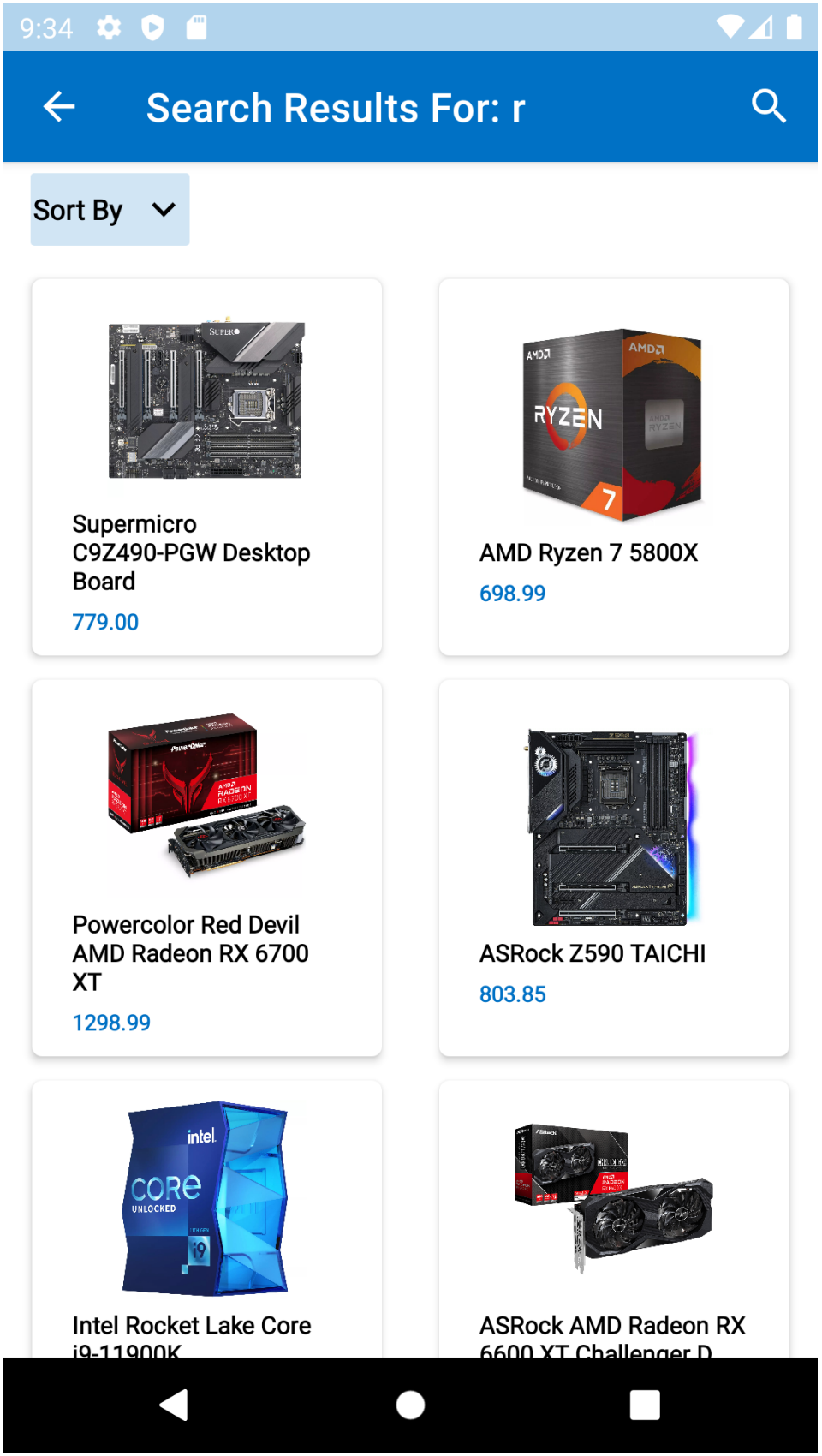
6.3 Details Activity

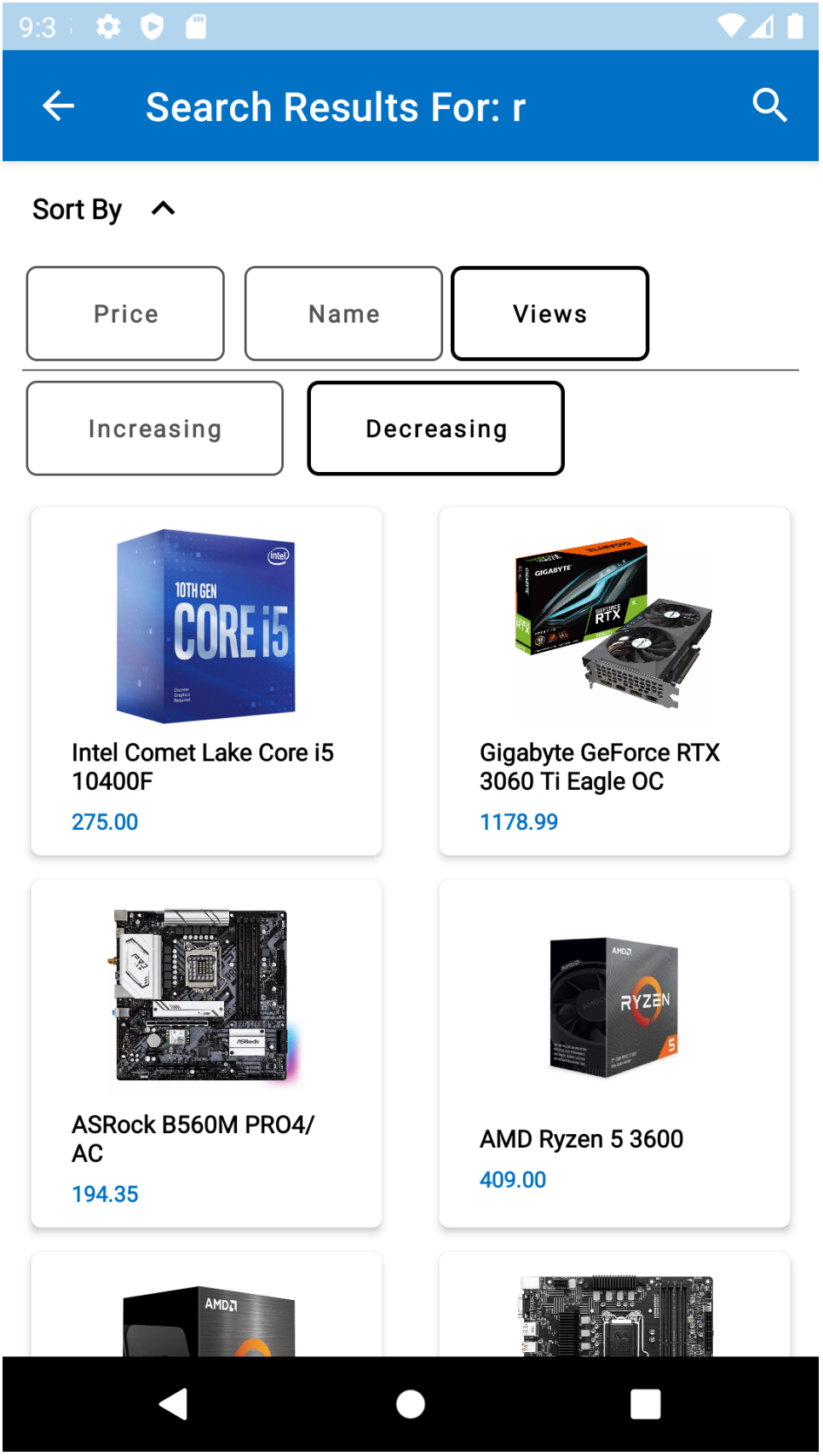



9:34
CPU
AMD Ryzen 5 3600
CPU
Num Cores 6
Clock Speed 3.6GHz
Boost Clock Speed 4.2GHz
Cpu Socket AMD AM4
Cpu Family AMD Ryzen 5
Price $409.00
Most viewed in category
AMD Ryzen 5 3600
AMD Ryzen 9 5950X
AMD Ryzen 7 5800X
Intel Rocket Lake Core i5-11500

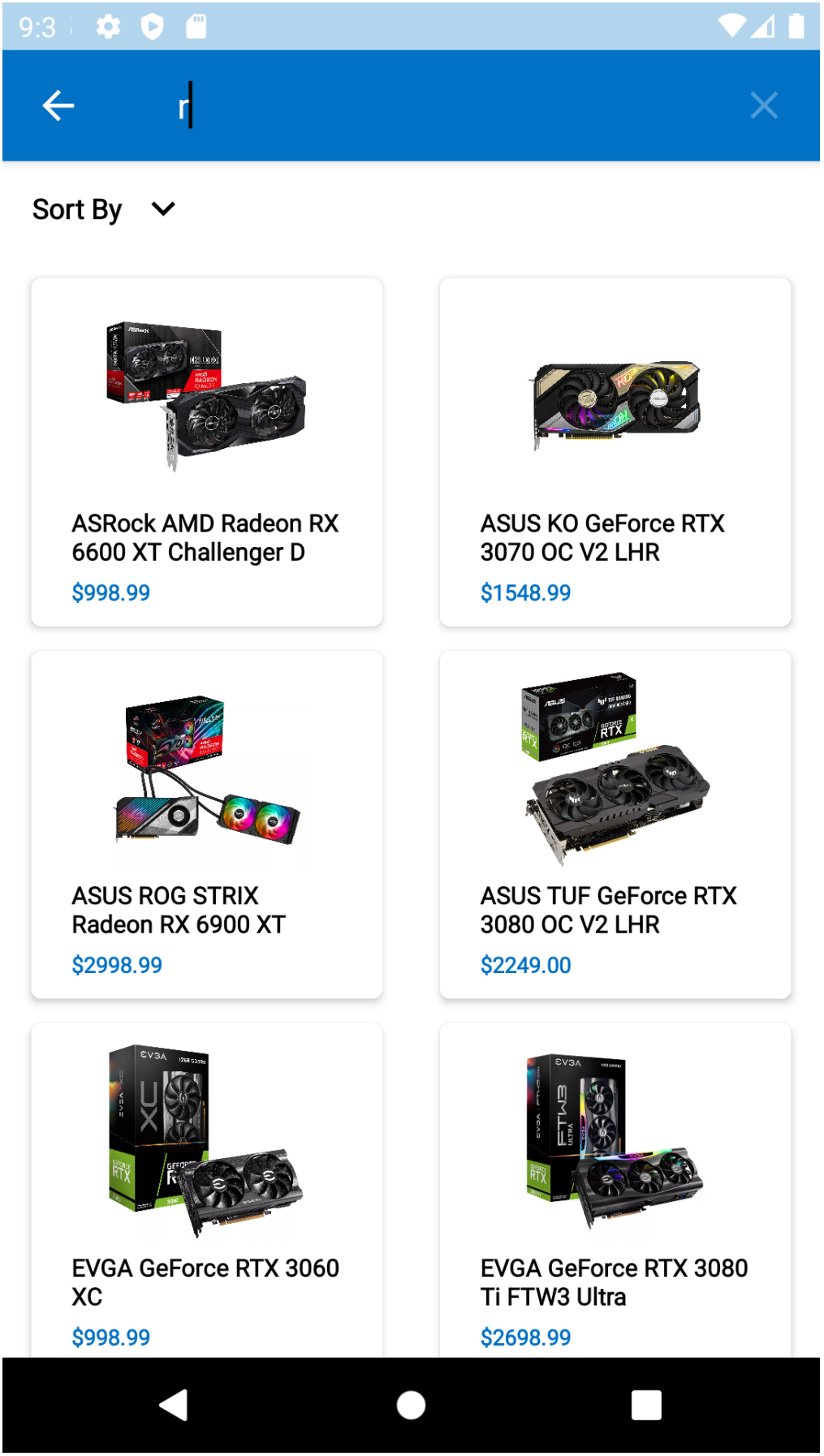
6.4 Search Activity

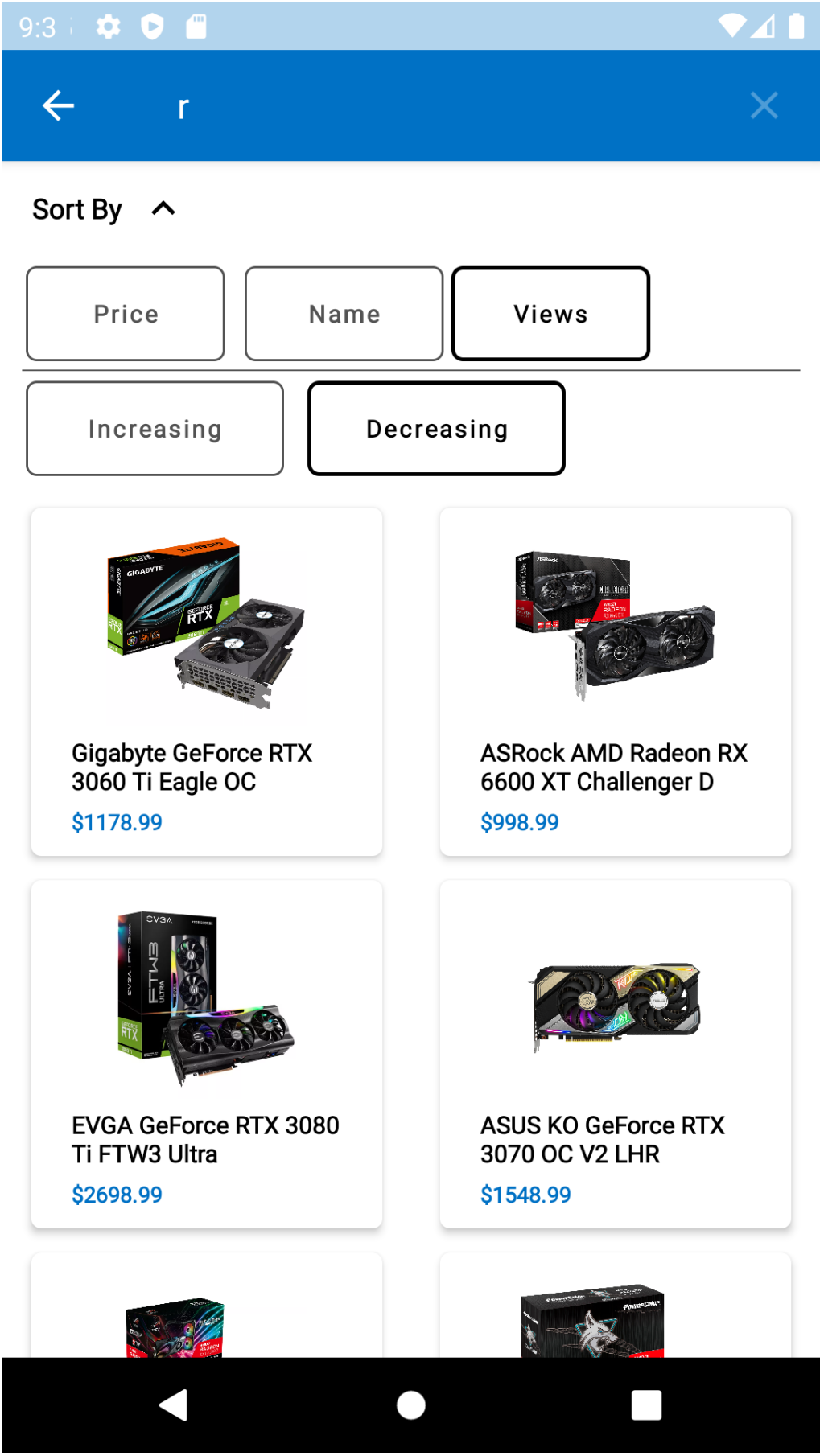
6.4.1 Search from Main Activity




9:3
Search Results For: r
Sort By
Price
Name
Views
Increasing
Decreasing
Intel Comet Lake Core i5 10400F
275.00
Gigabyte GeForce RTX 3060 Ti Eagle OC
1178.99
ASRock B560M PRO4/ AC
194.35
AMD Ryzen 5 3600
409.00

6.4.2 Search from List Activity




9:3
r
Sort By
Price
Name
Views
Increasing
Decreasing
Gigabyte GeForce RTX 3060 Ti Eagle OC
$1178.99
ASRock AMD Radeon RX 6600 XT Challenger D
$998.99
EVGA GeForce RTX 3080 Ti FTW3 Ultra
$2698.99
ASUS KO GeForce RTX 3070 OC V2 LHR
$1548.99