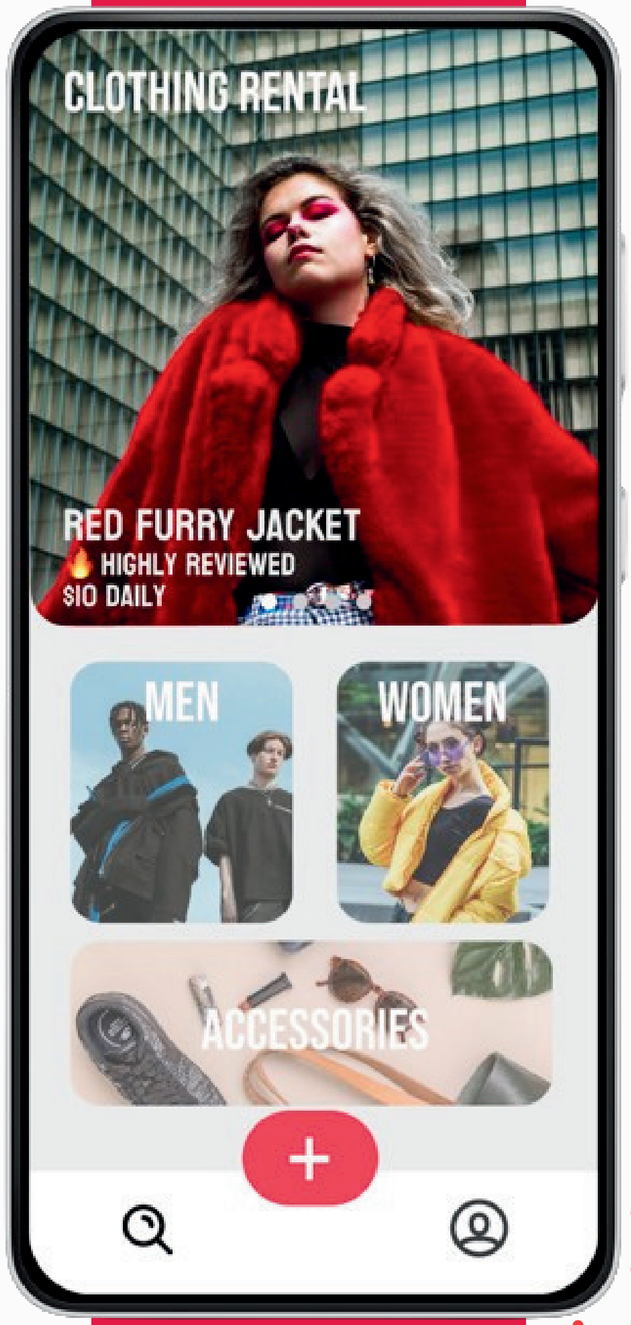
Group 45

FLEXHAVEN

CHECKOFF 2

Mok Shum Jung	1006619
Ong Jung Yi	1006655
Tejaswini	1007014
Ernest Tan	1006883
Luong Viet Hung	1007160
Raymond	1007040



Problem Statement and Solution

Target Audience: **Students**

Problem Statement: Singaporean students face **financial constraint that limits access** to essential items such as iPads and formal clothing with high upfront cost.

Solution: Our rental app FlexHaven **offers cost-effective, affordable rentals,** rewarding responsible use and enabling an income stream through item rentals, enabling students to rent items without financial strain.

Features

User Login and Logout Pages

- Collects user data and stores it in database

HomePage

- A navigation page for users to conveniently browse for trending items

Search Page

- Allows users to search for specific products

Individual Listings Pages

- Provides users with the details of product they are interested in

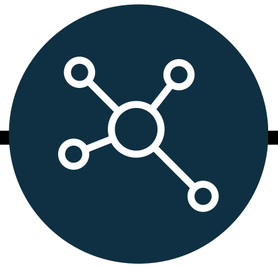
Seller Profile Page

- Allows users to view the seller's profile where they can find the seller's listings and product reviews

Progress

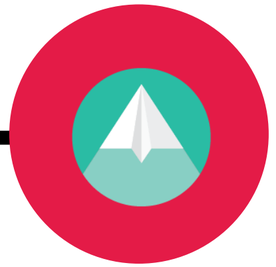
Completed the UI pages on Android Studio

Implemented Firebase to store user data



Completed the connection between the UI and the database using Recycler view adapter

Need to merge the pages together on Android Studio



Use Glide to display images

Choice of Data Structures

Programming Language: Java 

01

Data storage

Firebase

- to store user and item classes

02

Local storage

Array List implemented with priority queue

- To place items so that users with more points will have their items displayed

03

Item Display

RecyclerView

- Dynamically display items

Singleton and Encapsulation

- The CommonData class illustrates the Singleton design pattern, ensuring a single instance throughout the app.

- Encapsulation is enforced within CommonData and Item classes, with private fields and public methods controlling the access.

```
private CommonData(){
}
8 usages   shummy
public static synchronized CommonData getInstance(){
    if (instance==null){
        instance = new CommonData();
    }
    return instance;
}
```

“Global point of reference for data”

```
private ArrayList<String> categoriesToSearch;
2 usages
private ArrayList<Item> itemsToDisplay;
```

Robust Data Handling and UI Flexibility

- The Item class represents a robust data model, encapsulating the properties of items.
- ListingAdapter employs the Adapter pattern, adapting Item objects for use in a RecyclerView, facilitating efficient memory usage and smooth scrolling.

```
public Item(String email, String itemName,
    this.userEmail = email;
    this.name = itemName;
    this.category = category;
    this.condition = condition;
    this.price = price;
    this.location = location;
    this.itemDescription = itemDescription;
    this.imageUrl = uriString;
```

“Backbone”

```
public class ListingAdapter extends RecyclerView.Adapter<ListingAdapter.ListingViewHolder> {
    1 usage
    private final Context context;
    2 usages
    private LayoutInflater mInflater;
    6 usages
    private ArrayList<Item> items;
```

“Bridges item with RecyclerView”

Object-Oriented Best Practices and Logic Encapsulation

- The ‘Listings’ class follows OOP best practices with a clear focus on listing-related information and behaviors, demonstrating Single Responsibility Principle.
- Listings responsible for generating listing only. Data setting and layout display by adapter and recycler view respectively.
- It also features encapsulated logic to compute user tiers, showcasing information hiding and modular design.

Inner Classes

```
public ListingAdapter(Context context, ArrayList<Item> items) {
    this.context = context;
    this.mInflater = LayoutInflater.from(context);
    this.items = items;
}
1 usage  shummy
public void updateItems(ArrayList<Item> newItems) {
    items.clear();
    items.addAll(newItems);
    notifyDataSetChanged();
}
shummy
@NonNull
@Override
public ListingViewHolder onCreateViewHolder(@NonNull ViewGroup parent, int viewType) {
    View itemView = mInflater.inflate(R.layout.row_item, parent,  attachToRoot: false);
    return new ListingViewHolder(itemView);
}
```

```
private void computeTier(){
    if (this.userPoints<250){
        this.userTier = "Bronze";
    }
    else if ((this.userPoints>=250)&&(this.userPoints<750)){
        this.userTier = "Silver";
    }
    else if ((this.userPoints>=750)&&(this.userPoints<1500)){
        this.userTier = "Gold";
    }
    else{
        this.userTier = "Platinum";
```

Maxpriority queue

THANK YOU

Timeline

Phase 4: Week 12-14

Finalise App for presentation
Submit project

Phase 3: Week 10-11

Integration between front and back ends
User Testing and Refinement

Phase 2: Week 7-9

Implementation of crucial features (P0, P1)
User Testing

Phase 1: Week 5-6

Finalise details of our project (2D aspects, UML class diagrams)

Priority Levels

P0
Essentials

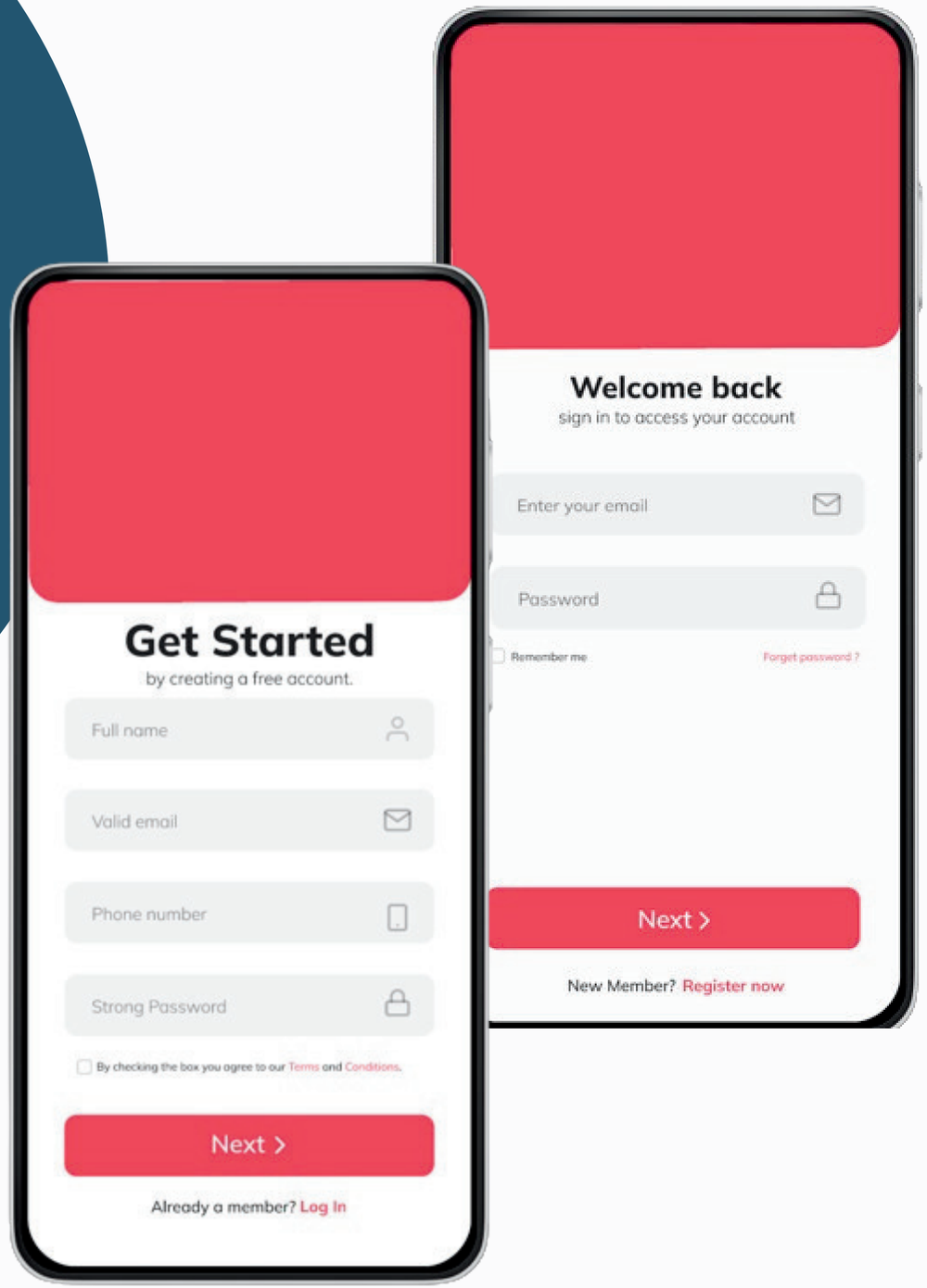
- Statically tie deposit and rental rates to user tier.
- Add and remove products from store page
- Purchase products

P1
Good to have

- Login Page

P2
Extra Features

- Chat feature between renter and rentee
- Search bar
- Dynamically render featured page



User Authentication

Priority : P1

- User-Friendly Form Fields and Placeholders
- Integration with Database to store input fields

Budget

$30	ISTD booth-material for posters, printing,etc
$50	For Computational Structures
$80	Total Budget

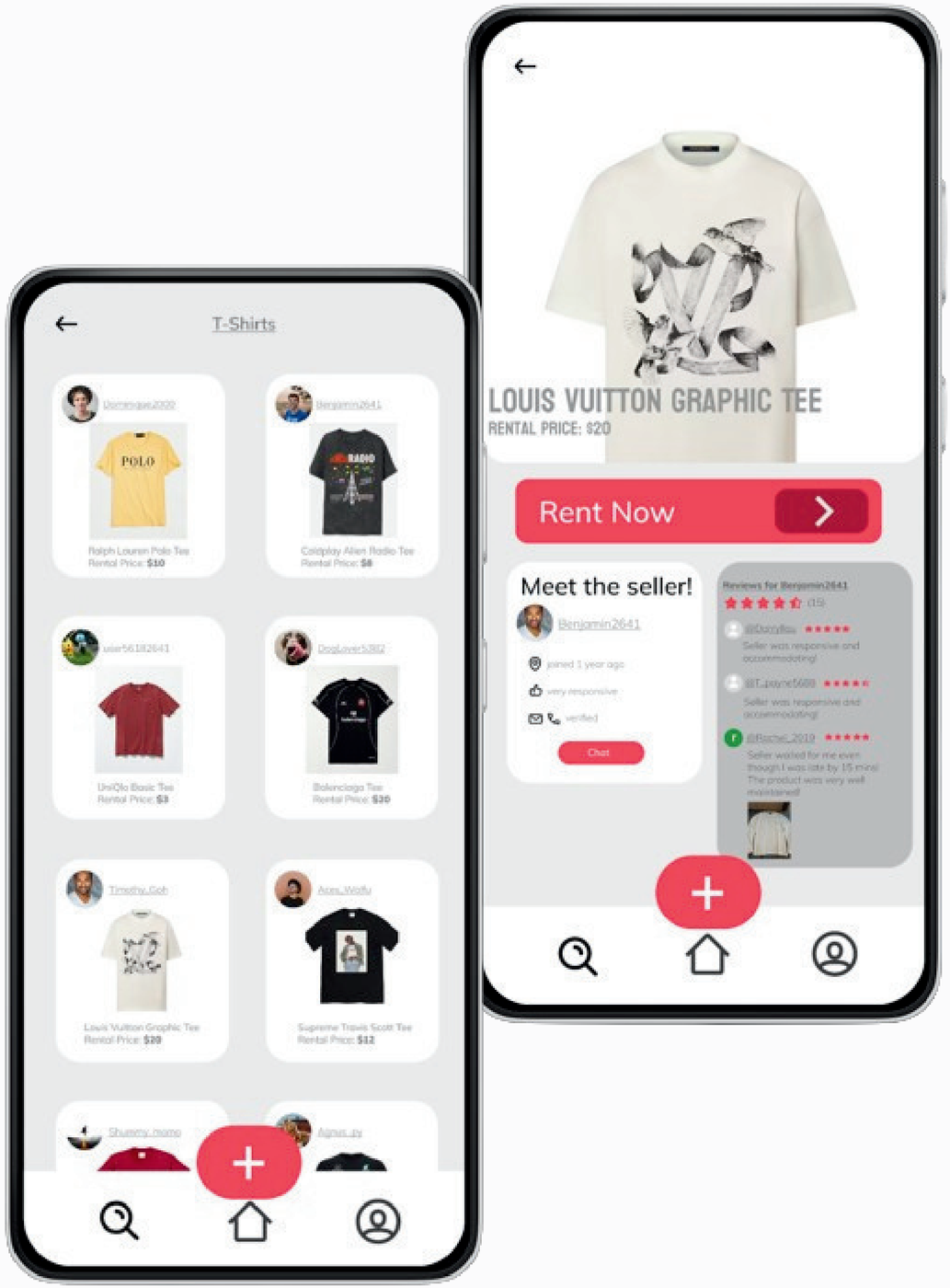
Sustainability, Diversity & Inclusion

Sustainability: UN Goal 12

- Users incentivised to rent items cheaply, instead of purchasing expensive items.
- Unused items reused instead of being inefficiently disposed.

Diversity & Inclusion

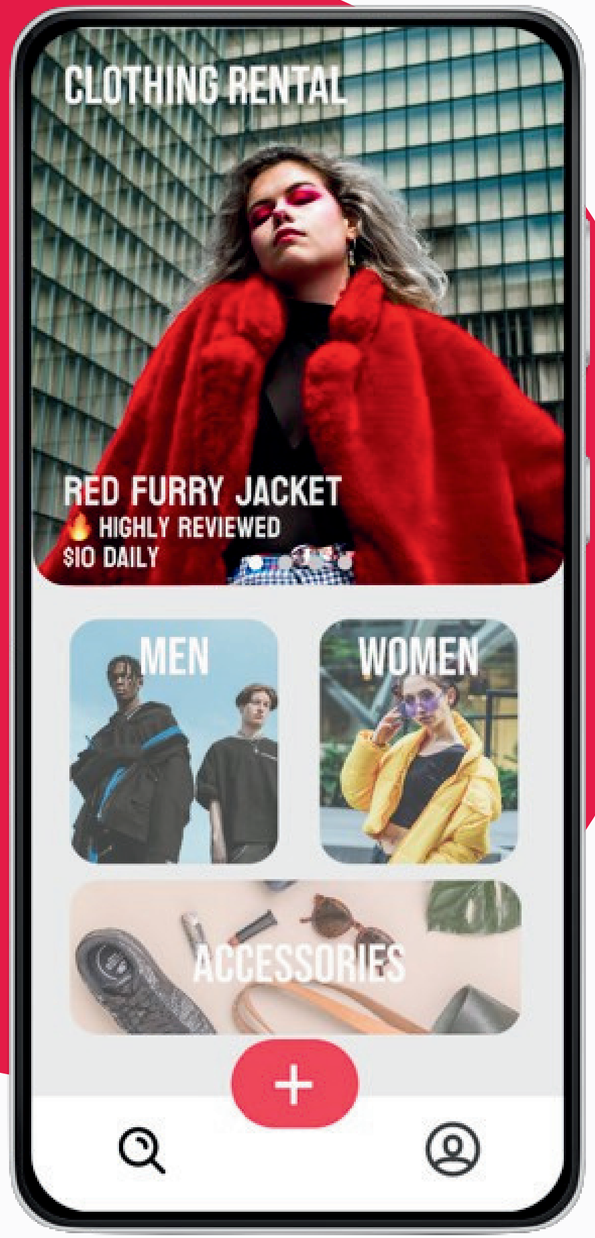
- Straightforward UI, large widgets caters to less technically inclined, older users



Individual Listings

Priority : P0

- Details of product
- Item and Seller Ratings
- Rental and Deposit Rates
- Information stored in Item class



Home Page

Priority : P0

- Clear and intuitive navigation menu
- Displays popular or trending items based on categories (Item class)