Report

Activity-MainActivity:

1. The activity extends the `AppCompatActivity` class, which provides compatibility for older versions of Android.
2. The code defines various UI elements such as `Button`, `EditText`, `TextView`, `RecyclerView`, and `ImageView` by retrieving references to them using their corresponding resource IDs.
3. There are two `RadioGroup` elements (`radio_group_price` and `radio_group_day`) that handle the selection of price ranges and days respectively. They have `OnCheckedChangeListener` implementations to capture the selected options.
4. The `refresh_button` has an `OnClickListener` that performs certain actions based on the `isActive` flag. If `isActive` is `false`, it reads input values, filters the data based on the selected price range and day, and displays the filtered results in a `RecyclerView`. If `isActive` is `true`, it resets the UI to its initial state.
5. The `select_location_button` has an `OnClickListener` that shows a toast message indicating that the user's location has been detected.
6. The `readLocalData` method reads JSON data from a raw resource file (`cleaners.json`) using an `InputStream`, `BufferedReader`, and `StringBuilder`. It then converts the JSON data to a list of `Cleaner` objects using the Gson library.
7. The `filtering` method filters the list of `Cleaner` objects based on the selected price range and day. The filtered results are added to the `mainList` ArrayList.
8. The `setAdapterForCleaners` method sets up the `RecyclerView` to display the filtered results by creating an instance of the `AdapterCleaner` class and setting it as the adapter for the `RecyclerView`.
9. There are getter and setter methods for various variables used in the activity.

Activity-CleanerDetailActivity:

1. The code includes import statements for required classes and libraries.
2. The `CleanerDetailActivity` class extends `AppCompatActivity`, which is the base class for activities in Android.
3. The class includes various member variables that correspond to different views in the layout file (`activity_cleaner_detail.xml`).
4. The `onCreate` method is the entry point of the activity and is called when the activity is being created. It initializes the views, sets event listeners, and updates the total price.
5. The `onButtonClick` method is called when the booking button is clicked. It retrieves the selected options from the checkboxes and stores them in the `selectedOptions` variable.
6. The `showDetail` method reads the cleaner data from a local JSON file and populates the views with the cleaner's information.
7. The `updateTotalPrice` method calculates and updates the total price based on the selected options and other factors.
8. The `readLocalData` method reads the cleaner data from a JSON file located in the `res/raw` directory using a `BufferedReader` and `Gson` library to deserialize the JSON into objects.
9. The `getScore` method determines the appropriate score image based on the cleaner's success score.
10. The `setAdapterForComment` method sets up the `RecyclerView` to display the comments and ratings for the cleaner.
11. The `getReservation` method is called when the user clicks the booking button. It shows a custom dialog for making a reservation.

Adapter-AdapterCleaner:

`MainActivity` is an activity class responsible for managing the main functionality of the app, while `AdapterCleaner` is a RecyclerView adapter class used to display a list of cleaners.

1. The class extends `RecyclerView.Adapter<AdapterCleaner.ViewHolder>` to create a custom adapter for the RecyclerView.
2. The class defines two arrays `cleanerImageId` and `scoreImage` that hold the resource IDs for cleaner images and score images respectively.
3. The class has an ArrayList variable `cleanerList` to store the list of cleaners.
4. The constructor of the adapter takes an ArrayList of cleaners and sets it to the `cleanerList` variable.
5. The `onCreateViewHolder` method inflates the layout for each item in the RecyclerView and returns a ViewHolder object.
6. The `onBindViewHolder` method is called for each item in the RecyclerView to bind the data to the views in the ViewHolder.
7. Inside `onBindViewHolder`, Glide library is used to load cleaner images and score images from the corresponding resource IDs into ImageView elements.
8. The `itemView` (the root view of the item layout) is set with an `OnClickListener` to handle clicks on the cleaner item. In this case, it creates an intent to launch the `CleanerDetailActivity` and passes the cleaner ID as an extra.
9. The `getItemCount` method returns the size of the `cleanerList`.
10. The `ViewHolder` class is a static inner class that holds references to the views in the item layout. These references are assigned in the constructor of the ViewHolder.

Adapter- AdapterReviewsAndRating:

1. The class extends `RecyclerView.Adapter<AdapterReviewsAndRating.ViewHolder>` to create a custom adapter for the RecyclerView.
2. The class defines two arrays `scoreImage` and `commentImage` that hold the resource IDs for score images and comment images respectively.
3. The class has variables for tracking the number of likes, dislikes, and the like/dislike state.
4. The class also has an ArrayList variable `ratingList` to store the list of ratings.
5. The constructor of the adapter takes an ArrayList of ratings and sets it to the `ratingList` variable.
6. The `onCreateViewHolder` method inflates the layout for each item in the RecyclerView and returns a ViewHolder object.
7. The `onBindViewHolder` method is called for each item in the RecyclerView to bind the data to the views in the ViewHolder.
8. Inside `onBindViewHolder`, Glide library is used to load score images, comment images, and user profile images from the corresponding resource IDs or URLs into ImageView elements.
9. The click listeners for the like and dislike buttons are implemented. When clicked, the like/dislike count is updated, and the button states are changed accordingly.
10. The `itemView` (the root view of the item layout) is set with an `OnClickListener`, but currently, it doesn't have any functionality defined.
11. The `getItemCount` method returns the size of the `ratingList`.
12. The `ViewHolder` class is a static inner class that holds references to the views in the item layout. These references are assigned in the constructor of the ViewHolder.

Entity- Cleaner:

1. The class defines several private member variables that represent different attributes of a cleaner, such as `photo`, `firstName`, `lastName`, `gender`, `age`, `insuranceInfo`, `phoneNumber`, `email`, `availability`, `introduction`, `cleaningMethods`, `rateMultiplier`, `ratings`, `totalRatings`, and `successScore`.
2. Each member variable is annotated with `@SerializedName` and `@Expose`. These annotations are used by the Gson library for serialization and deserialization of JSON data.
3. The class provides getter and setter methods for each member variable to access and modify their values.
4. The `availability` member variable is of type `ArrayList<String>`, which represents the available time slots or schedules of the cleaner.
5. The `ratings` member variable is of type `ArrayList<Rating>`, which represents the ratings and reviews received by the cleaner.
6. The `Cleaner` class provides public methods to get and set the values of these member variables.

Entity- Rating:

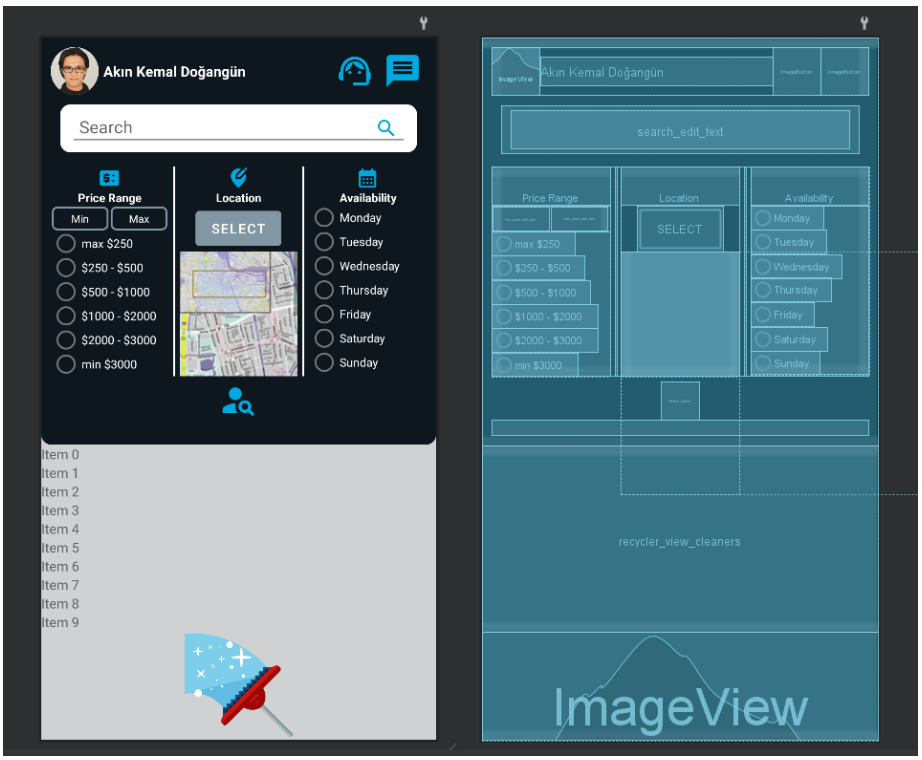
1. The class defines private member variables that represent different attributes of a rating, such as `user`, `rating`, `comment`, `date`, `photos`, `likes`, and `dislikes`.
2. Each member variable is annotated with `@SerializedName` and `@Expose`. These annotations are used by the Gson library for serialization and deserialization of JSON data.
3. The class provides getter and setter methods for each member variable to access and modify their values.
4. The `photos` member variable is of type `ArrayList<String>`, which represents the photos associated with the rating.
5. The `Rating` class provides public methods to get and set the values of these member variables.

Entity- Cleaners:

1. The class defines a private member variable called `cleaners`, which is an `ArrayList` of `Cleaner` objects.
2. The `cleaners` member variable is annotated with `@SerializedName` and `@Expose`. These annotations are used by the Gson library for serialization and deserialization of JSON data.
3. The class provides getter and setter methods for the `cleaners` member variable to access and modify the list of cleaners.
4. The `Cleaners` class acts as a container for a collection of `Cleaner` objects.

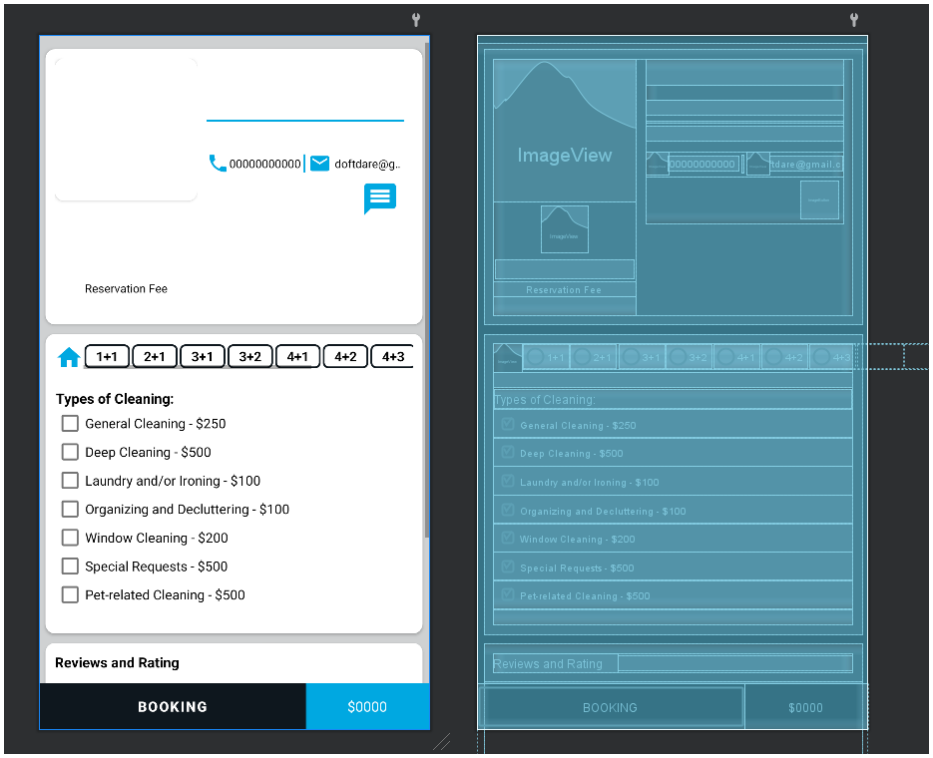
activity_main.xml:

1. The root element is a `LinearLayout` that serves as the container for all the other views in the layout. It has vertical orientation and a background color defined by the `app_color_5` resource.
2. Inside the root `LinearLayout`, there is another `LinearLayout` with an ID of `research_linear_layout`. It represents the header section of the layout and has a vertical orientation. It also has a background defined by the `style_main_layout` drawable.
3. Within the `research_linear_layout`, there is a horizontal `LinearLayout` that contains the user profile information and two `ImageButton` elements.
4. Below the user profile section, there is a `RelativeLayout` with an ID of `research_relativity_layout`. It represents the search bar section and has a background defined by the `style_search_box` drawable. It contains an `EditText` for entering search text.
5. Next, there is a `LinearLayout` with an ID of `filtering_linear_layout`. It represents the filter options section and has a horizontal orientation. It contains three sections: price range filtering, location filtering, and availability filtering.
6. The price range filtering section includes two `EditText` fields for entering minimum and maximum prices, and a `RadioGroup` with multiple `RadioButton` options for predefined price ranges.
7. The location filtering section includes a `Button` for selecting a location and an `ImageView` displaying a large map.
8. The availability filtering section includes a `RadioGroup` with multiple `RadioButton` options for different days of the week.
9. Below the filter options section, there is an `ImageButton` with an ID of `refresh_button`. It serves as a refresh button for updating the search results.
10. Next, there is a `TextView` with an ID of `total_result_text_view`. It displays the total number of search results.
11. Following that, there is a `LinearLayout` with an ID of `result_linear_layout`. It represents the container for the search results and has a vertical orientation.
12. Inside the `result_linear_layout`, there is a `RecyclerView` with an ID of `recycler_view_cleaners`. It is used to display a list of cleaners.
13. Finally, there is an `ImageView` with an ID of `background_image_view` that displays a background image defined by the `background` drawable.



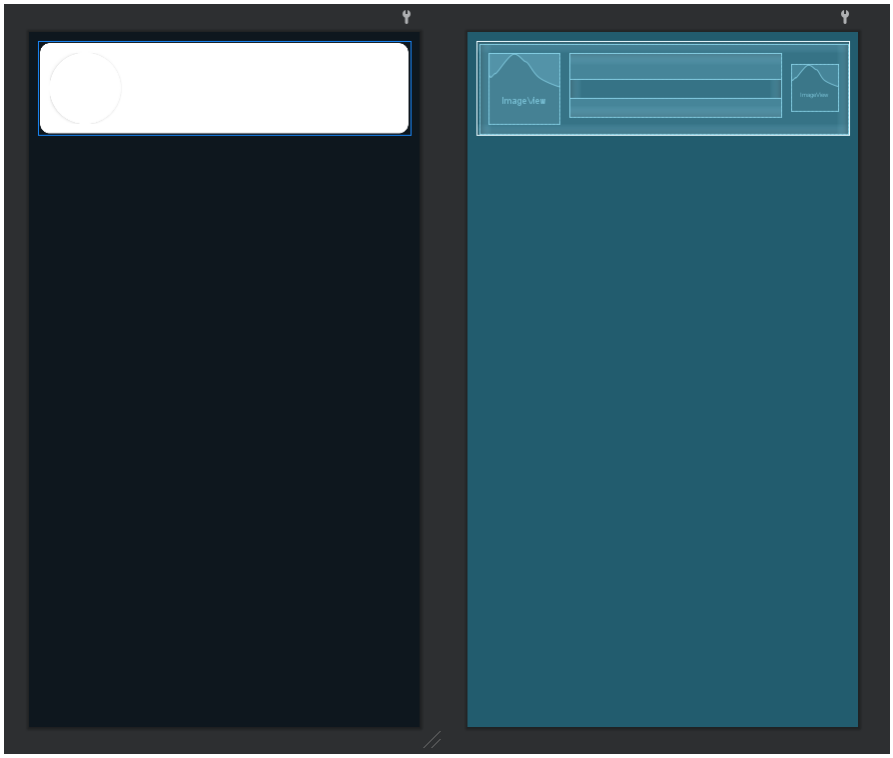
activity_cleaner_detail.xml:

1. It consists of a vertical LinearLayout as the root element, with a background color, and orientation set to vertical.
2. Inside the LinearLayout, there is a ScrollView that occupies the available space with a weight of 1. It contains another LinearLayout as its child, which is used to arrange the content vertically and apply padding.
3. The first CardView contains a horizontal LinearLayout as its child, which further contains two LinearLayouts. The first LinearLayout has a width of 150dp and contains a CardView with an ImageView inside it. It also includes an ImageView, two TextViews, and a TextView for displaying information related to the cleaner's image, score, insurance, and price.
4. The second LinearLayout has a weight of 1 and contains TextViews and ImageViews for displaying the cleaner's name, gender, age, introduction, cleaning methods, and contact information. It also includes an ImageButton for messaging the cleaner.
5. The second CardView contains a LinearLayout with a nested LinearLayout and a HorizontalScrollView. The nested LinearLayout is used to display a series of RadioButtons, which can be horizontally scrolled if they exceed the available width. It also includes a TextView for displaying the house price and a vertical LinearLayout for displaying the types of cleaning.



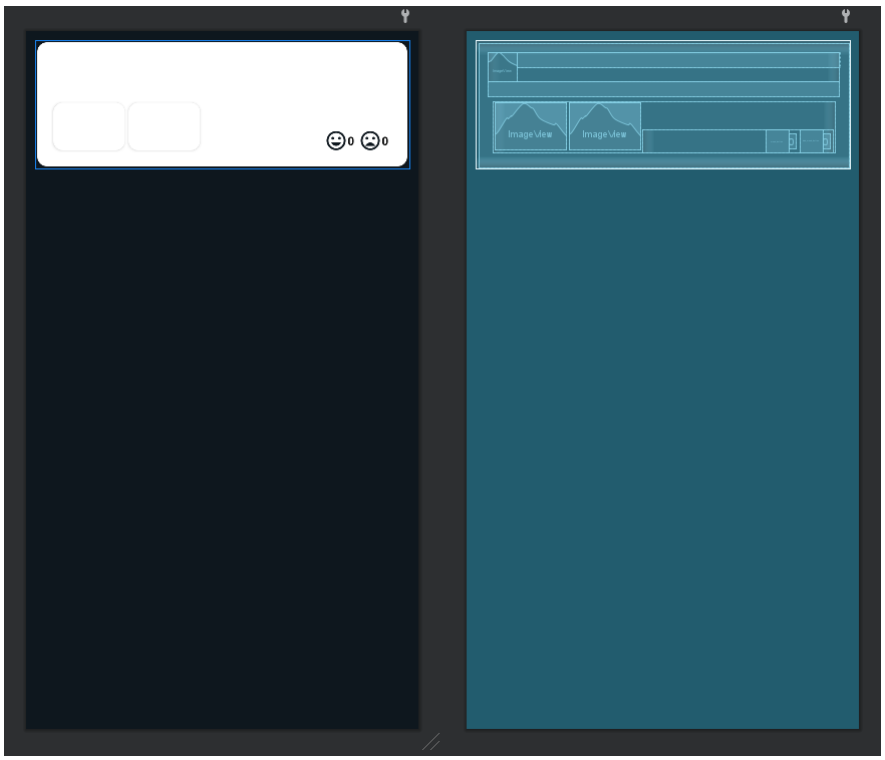
adapter_cleaner.xml:

1. The root element of the layout is a `LinearLayout` with vertical orientation. It serves as the container for the cleaner item.
2. Inside the `LinearLayout`, there is a `CardView` that acts as a container for the cleaner's details. It has rounded corners (`cardCornerRadius`) and a background color (`backgroundTint`).
3. Within the `CardView`, there is another `LinearLayout` with horizontal orientation. It contains the cleaner's profile image, name, gender, age, and price.
4. The cleaner's profile image is displayed using an `ImageView` inside a circular `CardView`. It has a fixed width and height (`layout_width` and `layout_height`).
5. The cleaner's name is displayed using a `TextView` with bold text style (`textStyle`) and a size of 20sp (`textSize`).
6. The cleaner's gender and age are displayed using two `TextViews` within a horizontal `LinearLayout`.
7. The cleaner's price is displayed using a `TextView` with bold text style and a different text color (`textColor`).
8. Lastly, there is an `ImageView` that represents the cleaner's score. It has a fixed width and height.



activity_rewiews_rating_detail.xml:

1. The root element of the layout is a `LinearLayout` with vertical orientation. It serves as the container for the rating item.
2. Inside the `LinearLayout`, there is a `CardView` that acts as a container for the rating details. It has rounded corners (`cardCornerRadius`) and a background color (`backgroundTint`).
3. Within the `CardView`, there is another `LinearLayout` with vertical orientation. It contains the user's name, score image, and date.
4. The user's name is displayed using a `TextView` with bold text style (`textStyle`), a size of 20sp (`textSize`), and a specific text color (`textColor`).
5. The score image is displayed using an `ImageView`. It has a fixed width and height (`layout_width` and `layout_height`), and the content is scaled to fit (`scaleType`).
6. The date is displayed using a `TextView` with a specific text size (`textSize`) and a different alignment (`gravity`) than the previous elements.
7. Below the user's name, there is a `TextView` that displays the comment. It has a normal text style and a specific text color.
8. Below the comment, there is another `LinearLayout` with horizontal orientation. It contains two `CardView` elements, each representing a rating option.
9. Each `CardView` has a fixed width and height (`layout_width` and `layout_height`), rounded corners (`cardCornerRadius`), and a margin (`layout_margin`).
10. Inside each `CardView`, there is an `ImageView` that represents the rating image. It has a match parent width and height (`layout_width` and `layout_height`) and a transparent background (`background`).
11. Below the rating options, there is a nested `LinearLayout` with horizontal orientation. It contains an image button, a text view, another image button, and another text view.
12. The first image button represents a positive mood and has a transparent background (`background`) and a specific image resource (`src`).
13. The first text view displays the number of positive moods and has a specific text color, size, and style.
14. The second image button represents a negative mood and has a transparent background and a specific image resource.
15. The second text view displays the number of negative moods and has a specific text color, size, and style.



dialog.xml:

1. The root element is a `LinearLayout` that serves as the container for other UI elements. It has several attributes such as `xmlns:android`, `xmlns:tools`, `android:layout_width`, `android:layout_height`, `android:orientation`, `android:background`, and `android:padding`.
2. Inside the root `LinearLayout`, there is a `ScrollView` element, which enables scrolling when the content exceeds the screen height.
3. Within the `ScrollView`, there is another `LinearLayout` that serves as the content container. It has attributes like `android:orientation`, `android:layout_width`, and `android:layout_height`.
4. The code includes several `CardView` elements, which are used to display cards with rounded corners. Each `CardView` has attributes like `android:layout_width`, `android:layout_height`, `app:cardCornerRadius`, `android:layout_margin`, and `android:backgroundTint`.
5. Inside the `CardView` elements, there are various UI components such as `HorizontalScrollView`, `RadioGroup`, `RadioButton`, `TimePicker`, `TextView`, `LinearLayout`, `ImageView`, and `Button`. These components have attributes specifying their dimensions, text, appearance, and other properties.